INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY
DIVISION DE GRADUADOS E INVESTIGACION
PROGRAMA DE GRADUADOS EN INGENIERIA

PRIMER ACERCAMIENTO PARA SOLUCIONAR EL PROBLEMA DE AGRUPACION DE FONEMAS EN PALABRAS, DEL ESPAÑOL, POR MEDIO DE LA REVISION DE LEXICO PARA UN SISTEMA DE RECONOCIMIENTO DE VOZ.

T E S I S

ELVIA PATRICIA BARRON CANO

MAYO 1991

INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY
DIVISION DE GRADUADOS E INVESTIGACION
PROGRAMA DE GRADUADOS EN INGENIERIA

PRIMER ACERCAMIENTO PARA SOLUCIONAR EL PROBLEMA DE AGRUPACION DE FONEMAS EN PALABRAS, DEL ESPAÑOL, POR MEDIO DE LA REVISION DE LEXICO PARA UN SISTEMA DE RECONOCIMIENTO DE VOZ.

T E S I S

ELVIA PATRICIA BARRON CANO

MAYO 1991

INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS MONTERREY
DIVISION DE GRADUADOS E INVESTIGACION
PROGRAMA DE GRADUADOS EN INGENIERIA

PRIMER ACERCAMIENTO PARA SOLUCIONAR EL PROBLEMA DE AGRUPACION DE FONEMAS EN PALABRAS, DEL ESPAÑOL,POR MEDIO DE LA REVISION DE LÉXICO PARA UN SISTEMA DE RECONOCIMIENTO DE VOZ

TESIS

PRESENTADA COMO REQUISITO PARCIAL PARA
OBTENER EL GRADO ACADEMICO DE

MAESTRO EN INGENIERIA
ESPECIALIDAD EN INGENIERIA DE CONTROL

ELVIA PATRICIA BARRON CANO

MAYO DE 1991

El principio de la
sabiduría es el
temor a Dios

A mis padres:
Laura Cano de Barrón
Enrique Barrón Matínez
y a mis hermanos.

RESUMEN

El objetivo de esta tesis es el contribuir a los estudios realizados en nuestro país sobre el área de reconocimiento de voz, en especial a los basados en la identificación de alófonos o fonemas, y usar algunas de las técnicas de Inteligencia Artificial, como algoritmos genéticos, sistemas de clasificadores, o redes neurales.

Se hace un previo estudio sobre las características del habla continua. Como resultado de éste, se implementa exclusivamente la fase de revisión de léxico basándose en su transcripción fonológica, es decir en fonemas, se creo una base de datos con un vocabulario básico de 22,883 palabras, y se hizo uso de redes neurales. Se usa una función de *hashing* para el direccionamiento, más otros archivos auxiliares. El sistema puede proporcionar como salida todas las posibles oraciones válidas, y para ello hace uso de pilas. Como criterio para realizar una separación previa de las palabras se analizan los grupos consonánticos, además por medio de estos es posible detectar la aparición de grupos inválidos, cuyo origen puede ser una pronunciación deficiente, o algún error en las fases de identificación de sonidos o de segmentación.

Para llevar a cabo la revisión, primeramente se analizan los grupos consonánticos, después se procesa cada grupo de sonidos enmarcados por pausas haciendose una búsqueda de izquierda a derecha para encontrar todas las posibles palabras contenidas; pero validando que pueda ser inicio de palabra. Se almacena sólo la primera acepción de

cada palabra, pudiendose pedir posteriormente las otras palabras homófonas con distinta categoría gramatical. Hechas las consultas, se pueden pedir las oraciones válidas.

La implementación fue completa, pero las transcripciones no fueron totalmente revisadas. El mecanismo de aprendizaje que se usó para entrenar las redes fue *backpropagation*, pero debido a una mala codificación de las entradas no se logró la convergencia de los pesos, aunque para probar el resto del sistema se trabajó con dos redes de pesos calculados. La primera red clasifica los grupos de dos consonantes, y la segunda los de tres y cuatro sonidos. No es indispensable la formación de dos redes sino que se pueden incorporar en una; sin embargo en nuestro caso, como se planteó en un principio el entrenamiento de la más simple y el posible no entrenamiento de la otra, se dejó de esa manera.

AGRADECIMIENTOS

Quisiera agradecer a las muchas personas que me ayudaron de una u otra manera para hacer este trabajo posible.

Agradezco al Dr. Manuel Valenzuela, a la Lic. Mirthala García, al Ing. Jesús Santana, al Ing. Francisco Cantú, a la Lic. Laura Medina, al Ing. Jorge Garza, al Dr. Federico Viramontes, a la Lic. María Guadalupe Torres, a la Lic. Ruth Esther Angel, a la Dra. Irene Gatrz, a la Lic. Cristina González, a la Lic. Celita Alamilla, a la Lic. Dolores Rangel, a la Lic. Marcia Galván, al Ing. José Luis Beltrán, al Ing. Pablo Ramírez, a la Lic. Moraima Cambell, al Ing. Gustavo Treviño, al Ing. Octavio Juárez, al Ing. Jesús Alanis, al Ing. Hugo Terashima, a la Ing. Martha Sordia, a la Ing. Lorena Gómez, a la Ing. Esperanza Garza, a la Sra. Leticia Rodríguez, a la Srita. Ana María Castillo, al Ing. Horacio Martínez, al Ing. Juan Nolazco, al Ing. Enrique Sánchez, al Lic. Alfonso Cota, al Ing. Armando Ruiz, al Ing. Eduardo Olivares, al Ing. Antonio Dávila, al Ing. Hector Méndez, al Ing. Sergio Treviño, al Ing. Ricardo Sahagún, al Ing. Alejandro Madariaga, y al Ing. Alejo Mosso por su muy valiosa ayuda.

Tabla de Contenido

Agradecimientos
Resumen
Lista de Figuras
Lista de Tablas
Introducción....1
Trabajos realizados sobre reconocimiento de voz....7
Capítulo 1
Trabajos realizados sobre reconocimiento de voz....8
1.1 Definición....9
1.2 Componentes....10
1.3 Características....12
1.4 Evaluación....15
1.5 Aplicaciones....15
1.6 Investigación desarrollada en el campus Monterrey..18
1.7 Resumen....22
Fundamentos....24
Capítulo 2
Redes Neurales....25
2.1 Características....25
2.2 Modelos de redes neurales....28
2.3 Modelos de aprendizaje....34
2.4 Resumen....35
Capítulo 3
Características del habla española....38
3.1 El idioma español....39
3.2 Ciencia del habla....39
3.2.1 El habla....43
3.2.2 Cualidades del sonido....44
3.2.3 Intensidad lingüística....47
3.2.4 Estilos de pronunciación en español....49
3.3 Fonética y Fonología (o Fonemática)....50
3.4 Grafías, fonemas y alófonos....52
3.5 Alfabetos....53
3.5.1 Ortográfico....53
3.5.2 Sistemas fonéticos y fonológicos....54
3.6 Neutralización de oposiciones....58

3.7 Cambios en el acento de intensidad y en la entonación 59
3.8 Señales demarcativas 62
3.9 Resumen 65
Capítulo 4
Conceptos relacionados al almacenamiento 70
4.1 Definición 70
4.2 Tipos de organizaciones 70
4.2.1 Archivos con función de dispersión 71
4.2.2 Archivos indizados 73
4.3 Resumen 74
Implementación 78
Capítulo 5
Justificación del alcance de la tesis 79
5.1 Tipo de revisión 79
5.2 Definición de la entrada 84
5.3 Definición de la salida 88
5.4 Uso de las redes neurales 89
5.5 Método de acceso a la Base de Datos 90
5.6 Resumen 90
Capítulo 6
Metodologías 93
6.1 Revisión fonológica 93
6.2 Métodos relacionados a la Base de Datos 94
6.2.1 Acceso 94
6.2.2 Estructura de la información 100
6.3 Métodos sobre la red neural 103
6.4 Resumen 113
Capítulo 7
Sistema de revisión de léxico 115
7.1 Procedimientos e ideas descartadas 115
7.2 Versión final 119
7.2.1 Programa Principal (Revlex) 120
7.2.2 Subrutina rev_ent 123
7.2.3 Subrutina memo 127
7.2.4 Subrutina consulta 127
7.2.5 Subrutina reporta 131
7.2.6 Programas para la creación de la Base de Datos 133
7.2.7 Programas de apoyo para la creación de las redes 136
7.3 Resumen 139

Capítulo 8
Resultados 141
8.1 Redes Neurales 141
8.2 Base de Datos 145
Capítulo 9
Conclusiones 148
Apéndice A 151
A.1 A partir de la escritura 151
A.2 A partir del registro 151
Apéndice B 153
B.1 Declaración de tipos y constantes 153
B.2 Declaración de variables globales 155
B.3 Código del programa Revlex.c 156
B.4 Código del programa Red.c 164
B.5 Código del programa Consulta.c 177
B.6 Código del programa Reporta.c 192
B.7 Código del programa AgregaSon.c 203
B.8 Código del programa CreaBD.c 213
B.9 Código del programa EstEsp.c 224
B.10 Red 1 230
B.11 Red 2 232

Referencias 235

Lista de Figuras

Figura 1.1. Componentes típicos de un sistema de reconocimiento de voz. 10

Figura 1.2. Diseño del reconocedor de voz Vox-Tec..... 21

Figura 2.1. Componentes básicos del modelo PDP 31

Figura 4.1. Ejemplo de archivo con función de dispersión, ilustrando una cubeta que contiene cuatro registros en dos bloques 73

Figura 4.2. Ejemplo de archivo indizado, mostrando sólo los valores claves 74

Figura 6.1. Acceso a la Base de Datos mediante la tabla de cubetas 99

Figura 6.2. Ejemplo de acceso a la Base de Datos mediante el uso de los archivos auxiliares a la cubeta 100

Figura 6.3. Organización de un registro en la Base de Datos... 103

Figura 6.4. Unidades de entrada de la red entrenada 105

Figura 6.5. Unidades de salida de la red entrenada 105

Figura 6.6. Unidades de salida de la red no entrenada 108

Figura 6.7. Algunos ejemplos del diseño de la red no entrenada 110

Figura 6.8. Dos ejemplos del archivo .net 112

Figura 7.1. Obtención de todas las posibles oraciones de la frase /elkampesinosabe/ 117

Figura 7.2. Diagrama de flujo del programa principal 126

Figura 7.3. Diagrama de flujo de rev_ent() 128

Figura 7.4. Diagrama de flujo de consulta() 129

Figura 7.5. Diagrama de flujo de reporta() 134

Figura 7.6. Formato del archivo de salida del programa AgregaSon 135

Figura 7.7. Diagrama de flujo de CreaBD() 138

Lista de Tablas

Tabla 6.1. Palabras definidas en la página 1 de García-Pelayo (1972). 95

Tabla 6.2. Palabras definidas en la pág. 294 de García-Pelayo (1972) 96

Tabla 6.3. Abreviaturas usadas por Juilland et al. (1964) para las categorías gramaticales 103

Tabla 6.4. Sonidos consonánticos que se presenta al inicio de palabra 106

Tabla 6.5. Sonidos consonánticos que se presenta al final de palabra 106

Tabla 6.6. Grupos de dos consonantes que se presenta a mediación de palabra 106

Tabla 6.7. Grupos de dos consonantes que sólo se presenta entre palabras 107

Tabla 6.8. Grupos inválidos de dos consonantes 107

Tabla 6.9. Grupos de tres y cuatro consonantes que se presenta dentro de palabra 108

Tabla 6.10. Grupos de cuatro consonantes que sólo se presentan entre palabra 109

Tabla 6.11. Grupos de tres consonantes que sólo se presentan entre palabra 109

Tabla 7.1. Variables globales 124

Tabla 7.2. Lista de tipos más importantes 125

Tabla 7.3. Información almacenada en arrpalenc después de haber consultado las base de datos para la frase /elkampesinosabe/ 125

Tabla 7.4. Estado de las pilas después de pedir la primera oración para el ejemplo de la Tabla 7.3.....132

Tabla 7.5 Relaciones usadas para la conversión de símbolos ortográficos a fonemas.....137

Tabla 8.1. Patrones mal clasificados para la red de 19 unidades143

Tabla 8.2. Unidades ocultas para la 1ª capa, en la segunda red144

Tabla 8.3. Unidades ocultas para la 2ª capa, en la segunda red144

Tabla 8.4. Unidades ocultas para la 3ª capa, en la segunda red145

Tabla 8.5. Lista del número de palabras con respecto a su longitud.....146

Tabla 8.6. Datos acerca de los arreglos generados de tres dimensiones.....147

Introducción

Lo que me motivó a realizar el presente estudio fue el deseo de contribuir en alguna manera a la implementación del sistema de reconocimiento de voz que se está elaborando en el Centro de Inteligencia Artificial, en el Instituto Tecnológico y de Estudios Superiores de Monterrey, ITESM, campus Monterrey. Lamentablemente, no me fue posible realizar la tesis sobre algún tópico de procesamiento de señales, porque ya se llevaban tres años y medio buscando e implementando los métodos más prometedores, para la obtención de las características de las señales de voz, para síntesis, e incluso para la identificación de los patrones de sonidos en alófonos.

Ante esas alternativas, y no perdiendo el ánimo, creí que quizás la podría realizar acerca de alguna fase posterior a la identificación de sonidos; es decir, ya sea en la obtención de las posibles oraciones o en las diferentes validaciones para encontrar la oración correcta, utilizando alguna técnica conocida en Ingeniería de Control, pero ahora aplicándola en el área de la Lingüística. De hecho, la identificación de los alófonos es realizada mediante una red neural. Una red neural o neuronal, concretamente, es una emulación simplificada del comportamiento de una red real de neuronas. Las redes neuronales han sido aplicadas en el área de Control para resolver problemas de Planeación y Control de Procesos, en Robótica y Análisis de Señales. Sin embargo, existen otros métodos de Inteligencia Artificial como algoritmos genéticos y sistemas de clasificadores, que también se han

usado en control, y que en determinado momento podría optar por utilizar.

Busqué, entonces, artículos relacionados sobre reconocimiento de voz y lingüística. Encontré dos, uno que se refiere a reglas fonológicas (Grasser y Lee, 1990a) y otro a reglas morfonémicas (Grasser y Lee, 1990b); aparte de aplicaciones en la fase de identificación de sonidos (Kohonen, 1989) y (Carrijo y Attikiouzel, 1990), y dos sistemas integrados para reconocimiento de palabras, uno mediante redes (McClelland y Rumelhart, 1986, cap. 15) y otro mediante algoritmos genéticos (Badii, Binstead, Jones, Stonham y Valenzuela, 1989). Además, revisé si había algún trabajo de tesis que hablara al respecto, pero no existe ninguno; y revisé los libros con relación a las materias en la Biblioteca Central y Cervantina del Instituto, en la Biblioteca Alfonsina y en la Facultad de Filosofía y Letras de la Universidad Autónoma de Nuevo León (UANL). Muchos de los libros localizados en Bibiloteca Central, con relación a Lingüística, los encontré en la UANL.

Un poco después de iniciado el período escolar, me decidí a tratar de implementar la primera fase; es decir, resolver el problema de la separación de los grupos de sonidos, que formen palabras, para la obtención de las posibles oraciones; y si era posible dependiendo de la complejidad del problema, realizar la revisión de concordancia.

Aunque la información proporcionada por las reglas fonológicas y morfonémicas no nos proporcionan indicadores directos para establecer los fines de palabras, creo y tengo como hipótesis lo siguiente:

Las palabras y frases o enunciados en español siguen una cierta estructura que facilita a los individuos comprender el mensaje hablado, aun cuando se exprese éste en forma bastante rápida; es decir, de alguna forma podemos identificar palabras aunque no exista una pausa real entre ellas. Además de esto, las personas tienen la facilidad de recordar e identificar las palabras, sean muchas o pocas, del idioma que hablan; y la facultad para hacer generalizaciones en base a algunos ejemplos, manejar excepciones y buscar lo más parecido, adivinar, cuando el mensaje no sea claro.

También se espera; es decir, se tiene como hipótesis, que el usuario exprese sus ideas o lea algo en forma coherente.

Para definir cómo, con qué y hasta dónde implementar, empecé a leer, casi al mismo tiempo, sobre Lingüística y las características de las diferentes técnicas aplicadas en Control. Después de analizar las técnicas, opté por redes neurales, por ser la más sencilla de las tres, aparentemente suficiente para lograr el objetivo de la tesis y con software disponible en ese momento. No fue posible implementar la revisión de concordancia; porque existe un número considerable de factores a considerar que se presentan en el habla continua, además de la dificultad de contar con el apoyo y el tiempo de personas expertas en el área de Lingüística, que revisen y validen cada una de las transcripciones de los sonidos y las reglas implementadas.

En los campus Morelos y Estado de México del ITESM, se están realizando trabajos sobre el área de reconocimiento de voz. En el campus Morelos se implementaron dos sistemas de reconocimiento de voz, uno usando la técnica de cadenas de Markov ocultas y el otro

mediante redes. Las limitantes de los sistemas son la restircción de habla no continua y de vocabulario reducido; es decir, esta orientado a la identificación de palabras. En el Estado de México, se implementó un pequeño sintetizador de voz en hardware y con un software que ocupa tan sólo 3Kbytes, basado en la concatenación de sonidos.

Podría llegar a afirmar que en todo sistema de reconocimiento de voz basado en la identificación de palabras, tan sólo puede manejarse un vocabulario relativamente reducido, por el número de patrones, el tiempo y el espacio que se requiere para la identificación.

Además, se tienen noticias de algunas otras instituciones en el país que trabajan en reconocimiento de voz o en lenguaje natural:

1. Escuela de Ingeniería de la Universidad La Salle, ULSA, en procesamiento de señales, y más especificamente en la segmentación de los sonidos.
2. Instituto de Matemáticas Aplicadas y Sistemas de la Universidad Nacional Autónoma de México, UNAM, el Dr. Héctor Haro en procesamiento de señales, y el Dr. Andrés Buzo con vectores cuantizados y cadenas de Markov ocultas.
3. Centro de Investigación y Estudios Avanzados del Instituto Politécnico de México, la Dra. Ana María Martínez en lenguaje natural, y el Ing. Hugo Sánchez en procesamiento.
4. Universidad de las Américas, Ana Jiménez Melo en lenguaje natural.
5. Instituto Tecnológico Autónomo de México, Osvaldo Cairo y Silvia Batistuti en lenguaje natural.

6. Colegio de México, María Isabel García Hidalgo en lenguaje natural.
7. Facultad de Ingeniería de la UNAM, Dr. Ismael Espinosa Espinosa con neurocomputadoras.
8. Universidad Autónoma Metropolitana-Iztapalapa, el grupo de Inteligencia Artificial, con procesadores paralelos y celulares, y con sistemas conexionistas.

Al parecer, éste es el primer estudio y trabajo realizado, de este tipo, en México, útil en sistemas de reconocimiento de voz basados en la identificación de los sonidos en alófonos o fonemas.

El procedimiento que se siguió en la elaboración de este trabajo fue el siguiente: documentarse en las áreas de Lingüística y en redes neurales, recibir asesoría de lingüístas, en especial de personas relacionadas con fonética, decidir qué tipo de transcripción se iba a manejar, tratar los aspectos particulares que se presentan en voz continua, decidir cómo se iban a resolver dichos problemas, definir hasta dónde se podía abarcar, formular los métodos para realizar lo establecido, implementar y documentar.

La exposición del trabajo es un poco diferente al procedimiento seguido, debido a dos aspectos fundamentales, es la primera tesis dentro del campus Monterrey que trata sobre reconocimiento de voz y es del tipo interdisciplinario. Esto me lleva a exponer lo mínimo indispensable sobre los temas de:

1. Trabajos realizados sobre reconocimiento de voz.
2. Fundamentos.
3. Implementación.

La tesis está estructurada precisamente en tres partes que cubren estos puntos.

La primera parte sólo consta de un capítulo. La segunda consta de tres capítulos; el primero de ellos habla sobre redes neurales; el segundo, y más extenso de los tres, habla sobre las características del habla española, y en el tercero se trata el manejo de bases de datos. La parte de implementación está dividida en cinco capítulos; el primero trata de justificar los alcances de la tesis, mediante un análisis de los fundamentos expuestos en el capítulo tres, en el segundo se definen los métodos a usar y se apoya la exposición en los capítulos tres y cuatro; en el tercero se mencionan brevemente los procedimientos descartados y la versión final del sistema;y en el cuarto y quinto capítulo se realiza un análisis de los resultados y se mencionan algunas conclusiones.

Se incluyen al final dos anexos, uno en donde se explican cuáles son los pasos que hay que seguir para la construcción de la base de datos y otro donde se incluyen los listados de los programas.

Trabajos realizados sobre reconocimiento de voz

Capítulo 1

Trabajos realizados sobre reconocimiento de voz

Las ideas acerca de la utilidad de máquinas parlantes son muy antiguas, de hecho se tienen antecedentes de que las civilizaciones Griega y Romana se interesaron en ello (Flanagan, 1976). Peacocke y Graf (1990) hacen alusión de que fue con el advenimiento de la computadora digital cuando empezó la investigación moderna sobre el reconocimiento de voz, facilitando a los investigadores la tarea de buscar caminos para la extracción de las características de la voz que permitieran hacer distinción entre palabras.

Ellos dicen que esto pasó a finales de 1950, pero, por ejemplo Sánchez (1986) menciona que es en los últimos años de 1940; la verdad es que la primera computadora análoga se reporta en 1927, y en 1937 las computadoras Mark I y ABC (Fuori y Aufiero, 1986); por otro lado, la referencia más antigua que se pudo obtener acerca de una investigación formal, la cual trata sobre la factibilidad de la identificación de la voz humana, fue la de McGehee (1937). Al parecer no es posible dar una fecha exacta, pero lo que sí se puede decir es que cada avance en la tecnología sirve de apoyo al desarrollo de las demás.

Peacocke et al. (1990), también, mencionan que en la década de 1960 se dieron avances en la segmentación automática de voz, en nuevos relacionadores de patrones y en algoritmos de clasificación; y en la década de 1970 hubo en los Estados Unidos un gran apoyo del gobierno para el desarrollo de investigaciones en el área de

reconocimiento de voz, y en nuevas direcciones como el estudio de las restricciones lingüísticas y el análisis automático de sintáxis y semántica (Flanagan, 1976). Rabiner (1976) menciona que la fabricación de microprocesadores y de sistemas especializados ha contribuido a la investigación.

Peacocke et al. (1990) dan en forma muy concreta las dos características que hacen deseables a los sistemas de reconocimento de voz, que son la posibilidad de dar o pedir información mientras se camina o se realiza alguna otra actividad, y la de proveer una medio natural y confortable de comunicación.

En este capítulo se explicarán algunas características de los sistemas de reconocimiento de voz, como ¿qué son?, ¿cuáles son sus componentes?, ¿en qué se diferencian los distintos sistemas?, y se mencionarán algunas aplicaciones.

1.1 Definición

El reconocimento de voz consiste en reconocer la expresión hablada con respecto a un idioma dado. Por lo tanto, un sistema de reconocimiento de voz es una aplicación que tiene la finalidad de identificar qué fue lo que dijo cualquier usuario y ejecutar una acción específica.

1.2 Componentes

La infraestuctura básica para que un sistema realice reconocimiento de voz (Pecocke et al., 1990) se muestra en la Figura 1.2 y consta de:

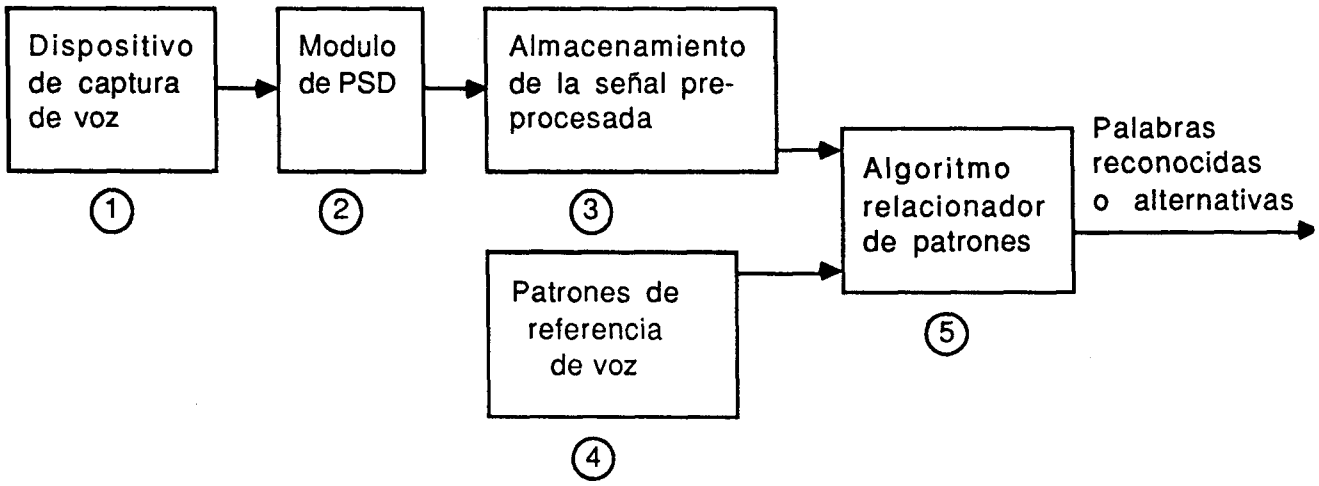


Figura 1.1. Componentes típicos de un sistema de reconocimiento de voz.

1. Dispositivo para captura de voz.
2. Módulo de procesamiento de la señal digital (PSD). Procesa la señal para obtener las características útiles para fines de reconocimento.
3. Almacenar la señal preprocesada.
4. Patrones de voz usados como referencia. Estos pueden representar la información estructurada de las características obtenidas por el PSD o sonido.

5. Algoritmo relacionador de patrones. El algoritmo debe medir la semejanza entre la señal preprocesada, de la voz del usuario, y todos los patrones almacenados. Los dos relacionadores de patrones más usados son el *dynamic time warping* y los modelos basados en cadenas ocultas de Markov, HMM. Pero recientemente se están empleando las redes neuronales (Kohonen, 1989), (Carrijo et al., 1990).

La técnica de *dynamic time warping* consiste en comparar la onda de voz preprocesada directamente con los patrones de referencia, sumando la distancia, es decir la diferencia entre las ondas. Para resolver los problemas de desalineamiento lo que se hace es ensanchar el patrón en algunos lugares y comprimir en otros hasta encontrar la relación óptima. Generalmente se usa programación dinámica para encontrar dicha relación.

Un modelo oculto de Markov es un proceso doblemente estocástico que sirve para producir una secuencia de señales observadas. La secuencia de pasos, estados, que se realizan para obtener la salida es desconocida, pero el fin es escoger los parámetros del modelo tal que se encuentre la relación óptima entre la señal real y la generada por el modelo (Rabiner, Juang, Levinson, y Sondhi, 1985).

Se tiene referencia de dos trabajos realizados donde las fases dos, tres, cuatro y cinco están integradas en un solo módulo; es decir usando una sola técnica. El trabajo realizado por McClelland y Elman (McClelland et al., 1986, cap. 15) mediante redes neurales, y el elaborado por Badii et al. (1989) mediante algoritmos genéticos.

1.3 Características

Los sistemas de reconocimiento de voz se caracterizan (Pecocke et al., 1990) por:

1. El tipo de entrada, pausada o continua.

   El reconocimiento de palabras o frases aisladas es más fácil porque no se presentan los efectos de coarticulación[1]. Una desventaja para los usuarios, es que hablar pausado reduce la velocidad de entrada de la información desde un rango de 150 a 250 palabras por minuto a 20 - 100 palabras por minuto.

2. El número de personas que puede reconocer, dependiente o independiente del usuario, o adaptivo al usuario.

   La voz de una sola persona puede fácilmente ser reconocida de una variedad de individuos porque muchas representaciones paramétricas de voz son sensibles a las características particulares del usuario. Gran cantidad de sistemas son dependientes del locutor, entrenándose el algoritmo para usuarse con cada operador. Una regla empírica usada por muchos investigadores es que, para una misma tarea, un sistema dependiente del usuario puede tener razones de error de 3 ó 5 veces más pequeños que los sistemas independientes del emisor.

   Una forma de hacer que un sistema sea independiente del locutor es mezclar patrones de entrenamiento de una amplia variedad de individuos, y una aproximación más sofisticada es observar las

---

[1]La pronunciación de una palabra cambia dependiendo de su posición relativa con respecto a las otras en el enunciado.

características que son relativamente invariantes entre las personas y usar éstas como referencia. Para encontrar las características que son similares, es necesario hacer un análisis estadístico ya sea sobre la intensidad de las señales, el tono, el timbre, la rapidez del habla, etc.

El término adaptivo al usuario implica que cualquier persona lo puede usar sin muchos inconvenientes, sólo hablándole a la computadora un cierto tiempo para que ella identifique las características particulares del hablante. Un sistema adaptivo tiene integrado un mecanismo automático para hacer esto; en cambio los dependientes del usuario no cuentan con dicho mecanismo.

3. Tamaño de vocabulario y gramática.

   El tamaño del vocabulario está fuertemente relacionado con la exactitud del reconocimiento y el tiempo de búsqueda, ya que si éste es grande es más probable encontrar palabras ambiguas. Las palabras ambiguas son aquéllas cuyos patrones son similares para el algoritmo de clasificación usado por el reconocedor. Los sistemas que contienen muchos patrones requieren generalmente técnicas de optimización, que pueden ser heurísitcas o no, para reducir el número de operaciones.

   La gramática establece la secuencia válida de palabras para el idioma en cuestión. Si se crean aplicaciones para uso específico o se toma como referencia el tipo de aplicación que está ejecutando el usuario, entonces se puede limitar el vocabulario, o

también, el espacio de búsqueda, pudiendo obtenerse un mejor reconocimiento.

4. Tipo de funcionamiento, operación en línea o fuera de línea.

   Si el sistema está diseñado de tal forma que el mecanismo de recepción de la señal de entrada es independiente de la frase de reconocimiento, y cada vez que se recibe nueva información se activa la rutina de reconocimiento, es decir que el sistema está recibiendo información e identificando a la vez, entonces dicho procedimiento es en línea o en tiempo real, y en caso contrario fuera de línea.

5. Adaptabilidad a cambios en el medio ambiente.

   El ruido de fondo, los cambios en las características del micrófono y la acústica pueden afectar dramáticamente la exactitud del reconocimento. Esto se ha tratado de solucionar usando el teléfono o los micrófonos de cabeza, y eliminado las componentes de frecuencias que no están en el rango audible.

El ideal a alcanzar por un sistema de reconocimiento de voz se define como un sistema que reconozca lo que exprese cualquier usuario (independencia de usuario), que opere en forma continua y sin restringir el vocabulario ni la gramática, más que la del idioma en si, que realice el proceso en línea y que se vea poco afectado por los cambios del medio ambiente. Tomando en cuenta el punto de referencia anterior, el sistema más restringido será dependiente del usuario, haciendo pausas al decir cada palabra, de vocabulario y gramática reducida, fuera de línea y muy sensible a cambios en el medio.

1.4 Evaluación

Para cuantificar qué tan compleja es la gramática de un sistema de reconocimiento de voz dado, se usa el término *perplejidad,* que indica el número de palabras que pueden estar asociadas a otra.

Se habla también de razones de error por:

a) *Rechazo o eficiencia.* Es el porcentaje de palabras que siendo válidas fueron rechazadas. Se maneja también el término opuesto que es la *exactitud*, y éste es el porcentaje de palabras que siendo válidas son reconocidas.

b) *Sustitución.* Es el porcentaje de palabras que siendo inválidas, pero muy similares a las conocidas, fueron aceptadas.

c) *Equivocada aceptación.* Es el porcentaje de palabras que siendo inválidas fueron aceptadas.

1.5 Aplicaciones

La tecnología de reconocimiento de voz ha emigrado de minicomputadoras y computadoras del tipo mainframe a estaciones de trabajo y computadoras personales. Muchas de las aplicaciones usan procesadores especiales para el procesamiento de señales, como los INMOS T800 Transputers y los TMS 320C25, con el fin de que el sistema sea operable. Dada la complejidad del proceso de reconocimiento, la tendencia actual es proveer a las estaciones de trabajo con componentes estándares para el procesamiento de señales digitales, como en la VAX station 3100 y la NEXT N1000. También se

han diseñado arquitecturas en paralelo (Lee, Tseng, et al., 1990) y (Alexandres, Morán, Carazo, y Santos, 1990), y neurocomputadoras (Espinosa, 1988), (Figueroa y Vargas, 1988).

Se han realizado innumerables estudios por parte de instituciones educativas, instituciones de investigación, empresas privadas y gubernamentales en algunos países como en Japón, Finlandia, Australia, Francia, España, Inglaterra, Estados Unidos, Canada y aquí en México (aunque apenas florece). Dado que en el mundo hablamos diferentes lenguas, la búsqueda de soluciones óptimas al reconocimiento de voz ha generado diferentes soluciones con diferencias en arquitectura y en lógica.

Existen muchos sistemas de reconocimiento que aceptan palabras en forma pausada, éstos no se mencionarán aquí, sino que se presentan algunas referencias sobre sistemas de reconocimiento continuo:

Sphinx, un reconocedor desarrollado por la Universidad de Carnegie Mellon, para voz continua, independiente del usuario, y con un vocabulario de 1,000 palabras. Usa codificación lineal predictiva (LPC) para la obtención de características, modela mediante cadenas ocultas de Markov cada alófono. Maneja dos bases de datos, una independiente del usuario y la otra dependiente, almacenando en ellas las características derivadas de LPC. Han reportado una máxima exactitud en el reconocimiento de palabras de 96% para una gramática con perplejidad de 20, de 94% con perplejidad de 60, y de 71% con una perplejidad de 997, para un vocabulario de 997 palabras (Lee, Hon, et al., 1990).

En el Instituto de Investigaciones Científicas Nacionales en Telecomunicaciones de la Universidad de Quebec, se ha desarrollado un sistema de reconocimiento de voz adaptivo, de palabras aisladas, con un vocabulario de 86,000 palabras en Inglés y con una exactitud del 93%. El vocabulario está basado en fonemas, por lo cual, no se requiere que cada nuevo usuario cree el vocabulario. La adaptación es realizada de 100 a 200 frases, es decir con unas 1,000 ó 2,000 palabras. Recientemente, se ha enriquecido este algoritmo permitiendo la entrada continua de voz y se cuenta con un vocabulario de 5,000 palabras (Lennig, 1990).

Byblos de BBN y un sistema desarrollado por laboratorios Lincoln tienen una exactitud en palabras de 88.7% y 87.4% respectivamente para una perplejidad de 60. El sistema Byblos requiere cerca de dos minutos de voz para adaptarse a un usuario en particular antes de alcanzar este nivel de eficiencia. Texas Instruments y el Instituto de Investigaciones de Stanford, han reportado sistemas con 44.3% y 40.4% de exactitud con una perplejidad gramatical de 997. El reconocedor Tangora de IBM, es capaz de una exactitud del 97% con 20,000 palabras de vocabulario y NEC con 97.5% para 1,800 palabras (Peacocke et al., 1990).

El reconocimiento de voz es aplicado, frecuentemente, en manufactura para compañías que necesitan la voz como entrada de datos o comandos mientras el operador ejecuta otra acción Existen aplicaciones relacionadas en inspección y control de calidad de productos, manejo de materiales, en la programación del control numérico de máquinas herramientas, como entrada a computadoras;

por ejemplo en Delco Electronics que emplea una IBM PC/AT Cherry Electronics y el sistema INTEL RMX86 de reconocimiento para el control de calidad de las tarjetas; en Southern Pacific Railway los inspectores envían información a la computadora mientras revisan los automóviles, usan el sistema VOTAN y equipo PC, en el control de tráfico aéreo, como entrada de datos barométicos y cartográficos (Martin, 1976).

Peacocke et al. presentan una lista de algunos sistemas comerciales de reconocimiento de voz; entre ellos el más completo es el diseñado por "Speech Systems Inc." llamado "Phonetic Engine"; es independiente del usuario, para reconocimiento continuo de voz, con un vocabulario de 10,000 a 40,000 palabras, su exactitud es alrededor del 95% y su costo oscila entre $10,500 a $47,100 dólares, y está disponible para estaciones de trabajo Sun. Otros sistemas operan en equipo IBM PC, en Machintosh.

1.6 Investigación desarrollada en el campus Monterrey

En agosto de 1986 se formó un pequeño equipo de estudio y de trabajo en el Centro de Investigación en Informática, actualmente ubicado en el Centro de Inteligencia Artificial, y al proyecto se le denominó Vox-Tec. El equipo estuvo inicialmente integrado por el Dr. José Antonio de la O, como coordinador, el Ing. René Rodríguez y el Ing. Horacio Martínez, se trabajó principalmente sobre síntesis de voz y un poco en el área de reconocimiento, para ello se utilizó una IBM PC AT. En 1988, el Dr. de la O y el Ing. Rodíguez dejaron de colaborar e ingresaron el Ing. Jesús Santana, como coordinador, el Ing. Juan

Nolazco, y posteriormente el Ing. José Luis Contreras, a partir de esa fecha el proyecto se abocó a la fase de reconocimiento y se empezó a trabajar en una Macintosh II. El objetivo a lograr era, y es, implementar un sistema de reconocimento de voz que cuente con las siguientes características: independiente del usuario, habla continua, tamaño de vocabulario y gramática no restingida, operación en línea e inmune al ruido; es decir, obtener casi el sistema ideal de reconocimiento de voz. Como submeta se tiene el utilizar un vocabulario restringido y no operar en tiempo real. Por último, en 1990 dió inicio esta tesis.

Para lograr este objetivo, se han buscado, analizado y evaluado las técnicas más prometedoras para la obtención de las características de la señal de voz, en la frecuencia y en el tiempo, y recientemente en la identificación de los sonidos. Sin embargo, un sistema de reconocimiento de voz no abarca sólo eso sino que es necesario implementar de alguna forma los mecanismos para la obtención de posibles oraciones en base al léxico, a partir de los sonidos ya identificados, y la eliminación de posiblidades mediante las revisiones de gramática y semántica.

El propósito u objetivo de esta tesis se fijó en estudiar e implementar hasta donde sea posible los mecanismos restantes para la conclusión del sistema de reconocimiento, haciendo uso de alguna técnica conocida en Ingeniería de Control, pero ahora aplicandola en el área de la Lingüística, como lo son redes neurales, algoritmos genéticos, o sistemas de clasificadores. El alcance de la tesis estaría

fuertemente ligado a la complejidad de los fenomenos que se presentan en el habla continua.

Las redes neurales son una emulación simplificada del comportamiento de una red real de neuronas. Los.algoritmos genéticos, son una técnica de optimización y búsqueda, y los sistemas de clasificadores, son sistemas basados en reglas, altamente paralelos en los que un algoritmo genético genera nuevas reglas y el peso de ellas se ajusta automáticamente.

Se encontrarón dos trabajos con redes neurales aplicados al área de lingüística, los realizados por Grasser y Lee (1990a y 1990b) donde se expone que las redes neurales son capaces de aprender reglas fonológicas y morfonémicas. Pero, mediante las otras técnicas nada, aún sin embargo se contempló la posibilidad de usarlas. Un poco después del inicio de esta tesis se creyó suficiente el uso de las redes neurales.

La base para la solución del problema planteado en esta tesis es la existencia de patrones en nuestra lengua que nos facilitan la identificación de palabras, y el uso de frases bien construidas. Para realizar el reconocimento de voz, los humanos tenemos la facultad de recordar un buen número de palabras, de reglas gramaticales y sintácticas, de poder hacer generalizaciones en la composición de palabras mediante sufijos y prefijos o en las conjugaciones, discernir qué palabras pudieron ser las pronunciadas cuando se tienen patrones confusos. La última hipótesis es importante, aunque parezca no serlo en primera instancia, porque de no ser así, no se podría usar el

análisis de concordancia y sintaxis como criterio para eliminar posibilidades.

Un esquema del sistema Vox-Tec se presenta en la Figura 1.2.

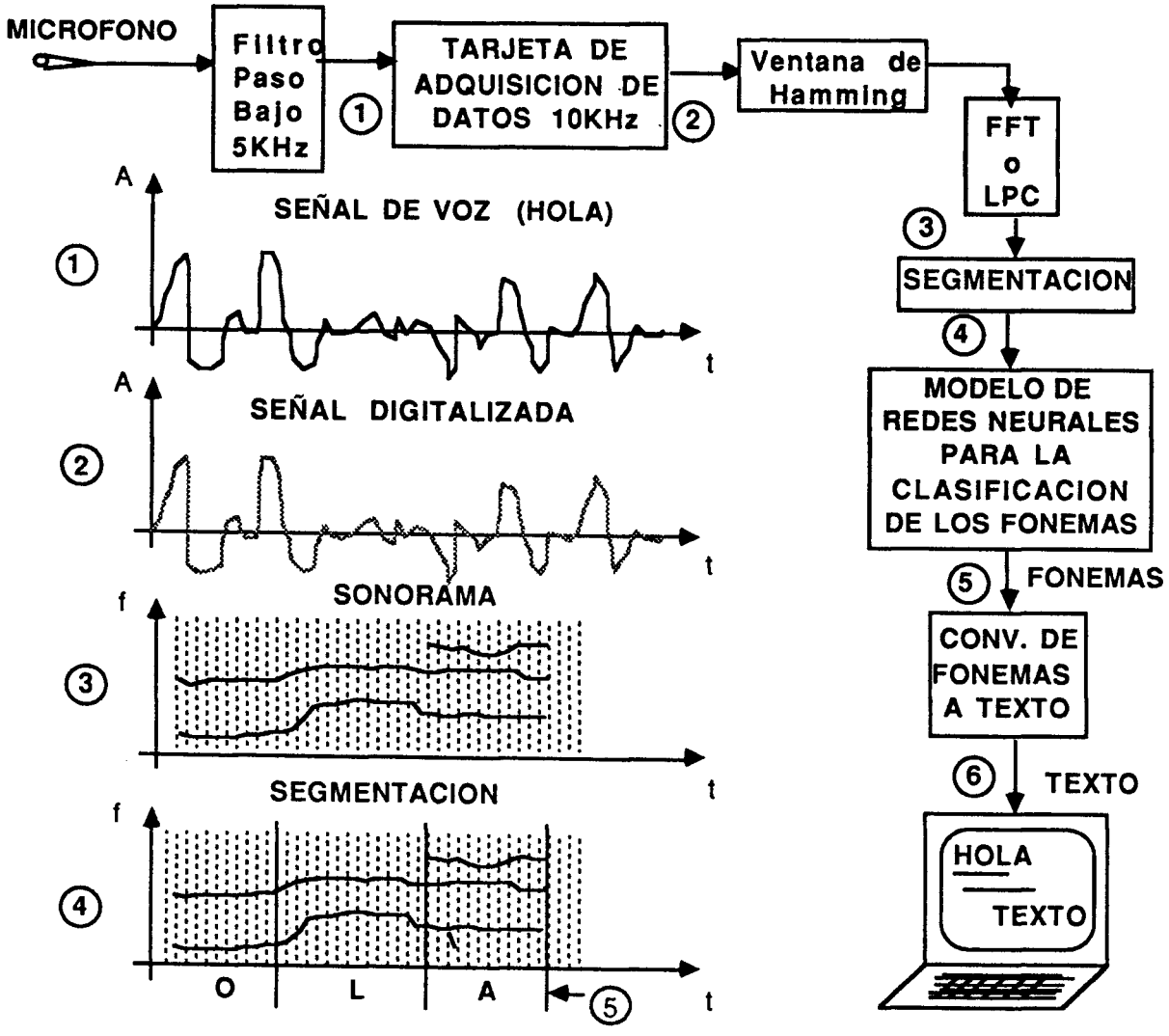


Figura 1.2. Diseño del reconocedor de voz Vox-Tec

1.7 Resumen

Un sistema de reconocimiento de voz es una aplicación que tiene la finalidad de identificar qué fue lo que dijo un usuario y ejecutar una acción específica.

Todo sistema de reconocimiento de voz está constituido por: un dispositivo de captura de voz, un módulo de procesamiento de señal, el almacenamiento de la señal preprocesada, patrones de referencia y un algoritmo relacionador de patrones. Algunas técnicas empleadas para hacer la comparación de patrones son *dynamic time warping*, modelos basados en cadenas de Markov ocultas, redes neurales y algoritmos genéticos.

Los sistemas se pueden distinguir por cinco aspectos:

1. El tipo de entrada, pausada o continua.
2. El número de usuarios, dependiente, independiente o adaptivo.
3. El tamaño del vocabulario y la restricción en la gramática.
4. Tipo de operación, en línea o fuera de línea.
5. La adaptabilidad a los cambios del medio ambiente, ruido de fondo, cambios en las características del micrófono o en la acústica.

Al hacer la evaluación de los sistemas se habla, generalmente, de su perplejidad, de razones de error debido al rechazo de palabras válidas, a la aceptación de palabras inválidas.

La tecnología de reconocimiento de voz se puede encontrar en todo tipo de máquinas, desde mainframes hasta computadoras personales. Se han realizado innumerables estudios en diferentes países. Existen

muchos sistemas de reconocimiento que aceptan entradas pausadas, y en los últimos años han aparecido en el mercado sistemas de reconocimiento continuo, en la Sección 1.5 se mencionan algunas de estas aplicaciones.

Para concluir, se habla sobre el sistema Vox-Tec y se incluye un diagrama del mismo.

Fundamentos

Capítulo 2
Redes Neurales

El objetivo de este capítulo es dar una idea de lo que son las redes neurales artificiales; es decir, qué representan, qué ventajas ofrecen con respecto a la tecnología actual, cuáles son sus características y qué tipo de modelos de aprendizajes se han implementado con ellas. El fin es proporcionar la terminología básica y los conceptos que se utilizarán en capítulos posteriores.

Se consultaron las referencias (Guyton,1975), (Tapia,1987), (Lynn, Meyer, y Hamilton,1967) y (Holland,1986) para cubrir los puntos de las características del cerebro humano y su comparación con los circuitos microelectrónicos. La información proporcionada sobre las ventajas, características y aplicaciones de las redes neurales artificiales se basa en el libro de Rumelhart et al. (1986).

2.1 Características

La unidad funcional principal del sistema nervioso es la neurona, llamada también célula nerviosa. La neurona está formada por un cuerpo celular típico con una o más extensiones filiformes largas denominadas fibras nerviosas. Estas fibras interconectan el cerebro, la médula y los nervios periféricos. Transmiten señales desde las partes periféricas del cuerpo al cerebro para señalar a la psique las sensaciones, incluso intervienen en la transmisión de señales a nivel

subconsciente; por ejemplo, para controlar funciones corporales como presión arterial, temperatura corporal, secreción de jugos digestivos y respiración. También son las neuronas y las correspondientes fibras neurales las que rigen los procesos de pensamiento en el cerebro, y las fibras llevan las señales del cerebro a los músculos para producir movimientos corporales.

Nuestro cuerpo posee un gran número de neuronas; existen 12,000 millones de éstas, y se encuentran localizadas en el cerebro y en la médula espinal. Las células nerviosas que se hallan en la corteza cerebral constituyen el mayor depósito de información, con 9,000 millones de neuronas, usado para el almacenamiento de recuerdos, respuestas motoras, etc.

El sistema nervioso central está constituido por centenares de clases de neuronas. Algunas tienen cuerpo celular muy grande, como de medio milímetro, y otras muy pequeño, de milésimas de milímetro. Hay neuronas que tienen la facultad de transmitir hasta 1000 impulsos por segundo y otras que sólo transmiten de 25 a 50 impulsos por segundo. Así mismo, varía el voltaje umbral de las distintas neuronas; éste se encuentra en el rango de los milivoltios. Las neuronas son elementos altamente interconectados que en ocasiones llegan a tener varios cientos de conexiones a otras neuronas, pero tansmitiendo una señal única.

Los investigadores en el área de ciencias cognositivas han realizado estudios sobre el funcionamiento de nuestro sistema de redes neuronales y han encontrado que cada neurona es una unidad sencilla de funcionamiento, pues no posee funciones psicológicas como

memoria e inteligencia, sino que estas propiedades son el reflejo del comportamiento global de la red; las neuronas son de respuesta lenta y trabajan en paralelo. Las neuronas captan las señales del medio y transmiten la información, su hipótesis, a las otras que actualizan sus estados asignándoles una ponderación, o fuerza, a la información proporcionada. Esta fuerza puede contribuir positiva o negativamente a la activación de esas neuronas; es decir, puede contribuir a su excitación o inhibición. Otra característica es que las conexiones pueden ser bilaterales lo cual implica que cada aspecto puede actuar sobre otros y ser influenciados por estos últimos. Las propiedades esenciales del cerebro están determinadas por la topología y la dinámica de la propagación de impulsos por las neuronas.

En el área de Computación se ha buscado aprovechar estos estudios para diseñar una arquitectura similar a la del cerebro humano que esté más orientada a las tareas de procesamiento de información natural. Dado que a pesar de contar con componentes que operan a muy altas velocidades, de tamaño reducido y que realizan funciones complejas, no se ha podido diseñar una máquina que piense o al menos que realice reconocimiento de imágenes o de voz.

Las computadoras actuales usan compuertas lógicas con velocidades de respuesta en el rango de los nanosegundos (10^{-9}), el consumo de potencia es alrededor de los miliwatts, y el requerimiento en las fuentes de alimentación depende del tipo de tecnología utilizada en los integrados, los hay de -24, -5, 3, 5, 12, 24 Volts, etc. La rapidez de los microprocesadores está en el rango de los microsegundos.

Haciendo una comparación entre los circuitos integrados y su contraparte biológica las neuronas, se puede decir que las neuronas son más lentas (aprox. 10^6 veces), consumen menos energía (aprox. 10^6 veces), su densidad de empaquetamiento[1] es mayor (aprox. 10^7 veces), existe un alto grado de paralelismo y redundancia en contraste con los circuitos integrados que trabajan en serie o en paralelo a baja escala.

Para tener una visión un poco más amplia sobre las limitaciones de las computadoras actuales se puede decir que los humanos en comparación con las computadoras son mejores para percibir y manejar objetos en escenas naturales, notar relaciones, clasificar y entender el lenguaje escrito y hablado, hacer planes y llevar a cabo acciones adecuadas en situaciones fuera de contexto. En otras palabras, los humanos son mejores para interpretar y modificar el medio. Inclusive, tienden a realizar las cosas con más exactitud y facilidad por medio del proceso de experiencia, a recobrar información contextual apropiada, asociar ideas, considerar múltiples restricciones, aunque éstas sean ambiguas o imperfectas, y a realizar generalizaciones.

2.2 Modelos de redes neurales

Se han creado modelos en base a los estudios realizados sobre el comportamiento de las neuronas reales, y éstos están siendo aplicados en diferentes áreas del conocimiento como en control adaptivo,

---

[1]Número de unidades por metro cúbico.

reconocimiento de imágenes, procesamiento de voz y en el procesamiento de lenguaje natural. A los modelos de redes neurales también se les llama modelos de procesamiento distribuido paralelo (modelos PDP).

Lo que hace diferente a una neurona de otra, aparte de su principio de funcionamiento, es la fuerza o ponderación de las conexiones y la polarización o energía base con la que trabaja; de esta manera, cuando se desea que un sistema de neuronas aprenda es necesario determinar precisamente eso, la fuerza en sus conexiones y su polarización.

En otros modelos de procesamiento cognositivo el conocimiento es almacenado como una copia estática de un patrón, y la finalidad del aprendizaje es formular reglas explícitas, proposiciones, producciones, etc. En cambio, en los modelos PDP lo que se almacena son las conexiones entre las unidades con las cuales se puede recrear el patrón, y el fin del aprendizaje, como se dijo antes, es la adquisición de la ponderación de los pesos, los cuales permiten a una red de unidades simples actuar como si conocieran o discernieran reglas.

Todo modelo de procesamiento distribuido paralelo está constituido por un grupo de unidades de procesamiento (neuronas), por un patrón de conectividad entre ellas (w_{i1}, w_{i2}, ...), un medio ambiente dentro del cual el sistema opera, un estado de activación ($a_i(t)$), una salida ($o_i(t)=f(a_i)$), una regla de propagación ($net_i=g(w_{ij},o_j)$), una regla de actualización de los estados de activación, llamada regla de activación ($F(a_i,net_i)$) y una regla de aprendizaje. Un ejemplo con tres unidades se encuentra en la Figura 2.1.

El primer paso para especificar un modelo PDP es definir la topología de la red, es decir su arquitectura, y definir qué representarán las unidades. Es posible manejar una relación uno a uno entre concepto y unidad, o de uno a muchos, donde cada unidad asocia una microcaracterística común a muchos patrones.

Existen tres tipos de neuronas de acuerdo a su ubicación en la red, que son: de entrada, de salida y ocultas. A las unidades de entrada llegan las señales externas a la red, éstas pueden venir de sensores o de otro proceso dentro del cual el modelo está embebido. Las unidades de salida envían señales hacia afuera de la red. Las unidades ocultas son aquellas cuyas entradas y salidas son internas a la red.

El estado de la red en el tiempo t es especificado por un vector de N números reales, a(t), que representa el patrón de activación sobre el grupo de unidades de procesamiento. La activación de la unidad u_i en el tiempo t es designada por $a_i(t)$. Los valores de activación pueden ser continuos o discretos, limitados o ilimitados.

La salida de cada unidad es una función de su estado actual de activación ($o_i(t)=f(a_i)$). El grado con que afecta a sus vecinas es determinado por la intensidad de los impulsos enviados, por el peso de la conexión entre ellas y por las características de la regla de propagación, se denota por net_i y es $net_i=g(w_{ij},o_j)$.

Generalmente, el patrón de conectiviadad puede verse como una matriz de pesos, W, en el cual w_{ij} representa la fuerza de la conexión de u_j a u_i; de esta forma la entrada neta a la unidad i puede expresarse como $net_i=Wo(t)$. Los pesos pueden ser positivos (exitatorios) o negativos (inhibitorios).

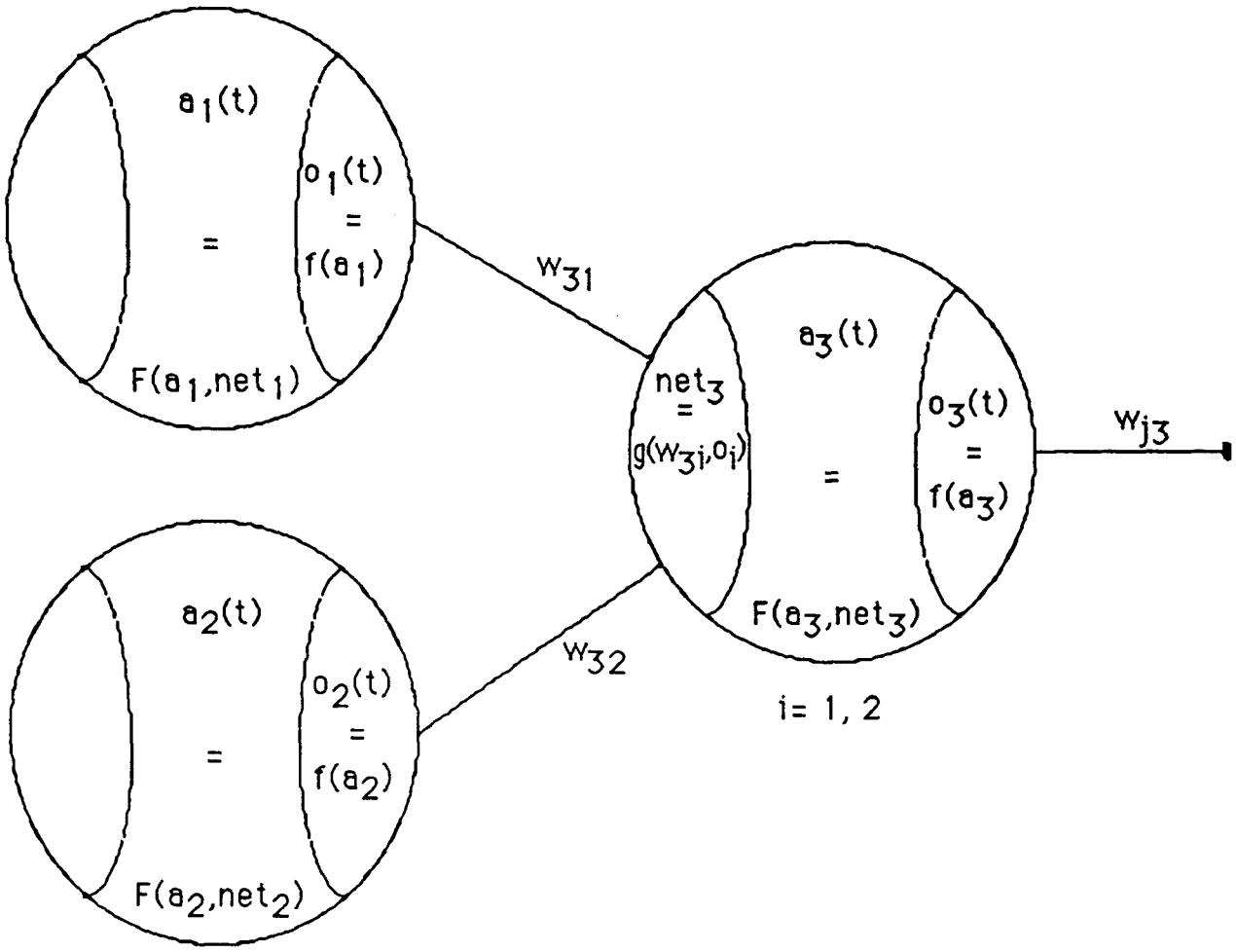


Figura 2.1. Componentes básicos del modelo PDP

En la naturaleza se observan patrones de conectividad aún más complejos, donde la entrada neta no es igual a la suma ponderada de las señales recibidas. En el libro de Rumelhart et al.(1986) se habla muy brevemente sobre esto y menciona dos patrones de conectividad diferentes.

La clase de funciones que normalmente se utiliza, tanto en la regla de activación, como para el cálculo de la salida, es usualmente la indentidad, igualándola al nivel de activación, una función umbral[2], una función sigmoidal, una función estocástica o en general semilineal[3].

Cambiar la estructura del proceso o del conocimiento implica modificar los patrones de interconectividad y esto se puede realizar de tres diferentes maneras:

1. Desarrollando nuevas conexiones.
2. Perdiendo conexiones existentes.
3. Modificando la fuerza de las conexiones que ya existen.

Existen muy pocos trabajos que involucran los primeros dos tipos de aprendizaje; sin embargo, éstos pueden considerarse como casos especiales del tercero, ya que se puede simular la creación o la anulación de nuevas conexiones al asignarles un peso diferente o igual a cero.

La forma general de la expresión para cambiar los pesos es $\Delta w_{ij}=g(a_i(t),t_i(t))*h(o_j(t),w_{ij})$; esta expresión nos dice que el cambio en el peso resulta de la multiplicación de dos funciones y que una de éstas depende del nivel de activación presente en la unidad y de la magnitud de una señal maestra o de referencia, y la otra función depende de las señales recibidas y de su ponderación.

---

[2]Si la entrada es mayor que el valor umbral la salida es diferente de cero.

[3]Función no decreciente y diferenciable.

Una regla de aprendizaje muy conocida es la regla de Hebb, la cual nos indica que el cambio de peso es proporcional al producto de la actividad de la unidad, por la intensidad de la señal recibida, $\Delta w_{ij}=\eta a_i o_j$. A la constante de proporcionalidad se le conoce como razón de aprendizaje. La regla de Hebb se usa en redes de dos capas, una con unidades de entrada y la otra con las unidades de salida. Mediante esta regla se pueden aprender perfectamente patrones ortonormales[4].

Otra regla es la de Widrow-Hoff de mínimos cuadrados o mejor conocida como regla Delta. El cambio de los pesos es proporcional al error, la salida esperada menos su activación actual que multiplicada por la señal de entrada nos da la regla de aprendizaje; es decir $\Delta w_{ij}=\eta(t_i-a_i)o_j$. También se usa en redes de dos capas y requiere que los patrones a aprender sean linealmente independientes[5].

Si las unidades son lineales pero manejan un nivel umbral para la activación, es decir

$$a_i=\begin{cases}1 \text{ si } net_i > \theta;\\ 0 \text{ si } net_i \leq \theta;\end{cases}$$

entonces se puede resolver cualquier tipo de función booleana, sin embargo el arreglo puede requerir unidades ocultas. Este tipo de topología en redes de dos capas no ofrece ninguna ventaja con respecto a las que usan una regla de activación lineal ($a_i=net_i$).

---

[4]Dos vectores son ortogonales cuando el producto cruz de éstos es cero y la magnitud de cada uno de los vectores es uno.

[5]Esto implica que, ninguna de las entradas pueda ser expresada como una combinación lineal de las otras.

Pueden existir redes cuya topología sea de más de dos capas, pero si las unidades siguen un comportamiento lineal ($a_i=net_i$ y $o_i=a_i$), éstas siempre se pueden reducir a dos, una capa de entrada y una capa de salida.

Cuando se manejan unidades semilineales, es decir que $o_i(t)=f(a_i)$ y $a_i=net_i$ donde f es una función diferenciable y no decreciente, se aplica lo que se conoce como regla Delta Generalizada o *backpropagation*. La cual se expresa como $\Delta w_{ij}=\eta\delta_i o_j$, donde: $\delta i=f'(net_i)\sum_k \delta_k w_{ki}$.

Esta implementa un método de descenso por gradiente. La entrada no se restringe.

2.3 Modelos de aprendizaje

Los mecanismos de aprendizaje que han sido implementados mediante el uso de redes neurales artificiales se pueden dividir en cuatro áreas que son:

Asociadores de patrones.

Se tiene como propósito aprender a relacionar un grupo arbitario de patrones de entrada con otro grupo arbitario de patrones de salida.

Auto Asociadores.

El objetivo es aprender a recordar un grupo de patrones y adquirir la habilidad de completar los patrones.

Clasificadores.

Es un caso particular del asociador de patrones donde se espera que la salida sea clasificada en cierto grupo.

Detectores de regularidades.

El sistema debe descubrir las características estadísticas de la población de entrada. Cada patrón de estimulo es presentado con una cierta probabilidad y el sistema desarrolla su propia representación interna, haciéndolo mediante el uso de las características más sobresalientes. Esto, en general, es parecido al clasificador pero no se cuenta con categorías predefinidas.

Las diferentes capacidades de las aplicaciones, dentro de un mismo modelo de aprendizaje, se deben a sus diferecias en la regla de aprendizaje, o en las reglas de propagación, activación o de salida.

2.4 Resumen

Los investigadores del área de Computación están interesados en los descubimientos realizados en las ciencias cognositivas, pues están buscando nuevos caminos de solución para resolver los problemas de reconocimiento de imágenes, de voz, de lenguaje natural, de inteligencia y aprendizaje.

Como fruto de ese interés se han obtenido modelos del comportamiento de las neuronas. Las neuronas han sido el principal foco de atención, debido a que éstas son las unidades fundamentales del sistema nervioso, que transmiten las señales recibidas desde las partes periféricas del cuerpo al cerebro, y del cerebro a los músculos

para producir los movimientos corporales y también rigen los procesos del pensamiento en el cerebro. Sus características son: funcionamiento sencillo, dispuestas en parelo, velocidad de respuesta lenta, altamente interconectadas y manejan niveles de voltajes bajos.

Las redes neurales artificiales son una emulación de las redes de neuronas. Dentro de la red existen tres tipos de unidades que son de entrada, ocultas y de salida. Las neuronas de entrada se caracterizan por recibir las señales externas de la red, las de salida trasmiten la información hacia afuera de la red, y las ocultas son las que reciben y trasmiten información interna a la red. A los modelos de redes neurales también se les conoce como modelos de procesamiento distribuido (modelos PDP).

Todo modelo PDP cuenta con unidades de procesamiento, que pueden ser de entrada, ocultas o de salida, un patrón de conectividad, un medio ambiente dentro del cual el sistema opera, un estado de activación por cada unidad, una salida para cada unidad, una regla que establece cómo se propagarán las señales, otra para la actualización de los estados y por último una regla de aprendizaje.

El fin del aprendizaje en los modelos PDP es la adquisición de la ponderación de los pesos, los cuales permiten a una red de unidades simples actuar como si conocieran o discernieran reglas. Se mencionan en este capítulo las tres reglas de aprendizaje más populares, la regla Hebb, la regla Delta y la regla Delta Generalizada.

Por último, se mencionan los diferentes mecanismos de aprendizaje que han sido implementados mediante redes que son: asociadores de patrones, autoasociadores, clasificadores de patrones y detectores de regularidades.

Capítulo 3

Características del habla española

El objetivo de este capítulo es presentar la información utilizada para la implementación, exponer los aspectos que la fundamentan y temas afines que se estima serán factores importantes para futuros trabajos. Se tratará de exponer lo más representativo de cada tema, enfatizando en lo que afecta directamente al trabajo.

Se inicia nuestro estudio con una muy breve explicación del origen del español, su importancia, cuál es la ciencia que lo estudia y sus diferentes disciplinas.

Posteriormente se orienta la exposición al estudio de los sonidos y las dificultades que se presentan al emitir palabras en forma continua. Para ello, primero se menciona qué es el habla, las unidades de expresión, las cualidades del sonido, qué es la intensidad lingüística, estilos de pronunciación, Fonética y Fonología, la representación del sonido hablado: unidades (grafías, alófonos y fonemas) y sistemas (ortográfico, fonético y fonológico). Por último, se mencionan los efectos de la neutralización, cambios de acento, entonación, y se enumeran todas las características encontradas, que se presentan en nuestro idioma y que pueden indicar si se está dentro de una palabra o al final de una palabra.

3.1 El idioma español

El español es un idioma románico o romance que tiene su origen en el latín vulgar, y éste a su vez se encuentra dentro de la rama italocéltica de la familia lingüística indioeuropea. De acuerdo a la clasificación morfológica nuestro idioma es una lengua flexible, debido a que las palabras constan de una raíz y de morfemas, elementos que se combinan con la raíz para formar las palabras (Mateos, 1980).

Según información proporcionada por David Crystal (1987) de un censo realizado en 1977 por C. F. y F. M. Voegelin en "Classification and Index of the World's Languages", de los idiomas más hablados en el mundo, el español ocupa el tercer lugar, como lengua madre, con aproximadamente 250 millones y el cuarto lugar, como lengua oficial, con aproximadamente 280 millones de hablantes. Con respecto a los idiomas que tienen sus origen en el latín, ocupa el primer lugar y también informa que el español es hablado en España, México, Centroamérica, Sudamérica y en las islas Canarias.

3.2 Ciencia del habla

Para poder conocer las características de un idioma es necesario hacer referencia a la ciencia que estudia el lenguaje, es decir a la Lingüística. La Lingüística es la "ciencia que estudia la estructura de las lenguas, la manera como se hablan, su historia, su origen y los fenómenos humanos que se manifiestan en ellas" (Lara , 1986, p. 307).

Las lenguas pueden ser estudiadas desde dos tipos de enfoques que son, realizando un análisis descriptivo del estado actual de las lenguas, o analizando cómo y porqué se realizan cambios en éstas en el transcurso del tiempo. Así surgen los términos de Lingüística Descriptiva o Sincrónica y la Lingüística Histórica o Diacrónica.

Independientemente del tipo de enfoque, las principales disciplinas de la Lingüística son "la Fonética y la Fonología, que estudian el aspecto acústico de las lenguas; la Gramática, que se ocupa de sus aspectos formales; la Semántica y la Lexicología que investigan el sentido y el empleo de los vocablos" (Reader's Digest, 1986, p 2200). Algunos autores como Southworth y Daswani (1974) mencionan solamente a la Fonología, a la Gramática y a la Semántica; dividiendo a la Fonología en Fonética y Fonemática y no mencionando la Lexicología.

Los autores anteriores recalcan el hecho de la existencia de la Fonología Descriptiva e Histórica, la Gramática Descriptiva e Histórica, y la Semántica Descriptiva e Histórica; aparte hacen mención de la Dialectología Geográfica, la Lingüística Generativa Transformacional, la Sociolingüística, la Lingüística Antropológica y la Etnografía.

La Gramática "se divide en Morfología o estudio de la composición de las palabras en morfemas, y Sintaxis o estudio de sus combinaciones en la oración y en el texto o discurso" (Lara, 1986, p. 251). Además está la Ortografía que regula el modo correcto de escribir.

En el desarrollo de futuros trabajos dentro del área de reconocimiento de voz, la Lingüística Generativa Transformacional debe de jugar un papel muy importante, por lo siguiente:

> En la Lingüística Generativa, la gramática de una lengua es el modelo de la competencia ideal que establece una cierta relación entre el sonido (representación fonética) y el sentido (interpretación semántica). La gramática de un lenguaje L genera un conjunto de parejas (s, I), donde s es la representación fonética de una determinada señal e I la interpretación semántica que atribuyen a esta señal las reglas del lenguaje. (Dubois et al., 1983, p. 319)

Rivero (1977) la define como:

> Una gramática generativa y transformacional es un mecanismo capaz de generar un número infinito de oraciones por medio de un número finito de reglas o transformaciones. Las trasformaciones tienen al mismo tiempo la función de mostrar la relación existente entre diversas oraciones. La descripción lingüística que nos ofrece una gramática de este tipo incluye tres componentes: el componente sintáctico, el semántico y el fonológico. El componente sintáctico debe ser considerado como primordial, ya que genera las estructuras abstractas que constituyen la forma básica de toda oración. El componente semántico y el fonológico es puramente interpretativo y aporta a las estructuras formales, generadas por el componente sintáctico, una interpretación semántica el primero y una representación fonológica el segundo. (p. 19)

Hablando de los lenguajes en general, Southworth et al. (1974) hacen referencia que en todos los lenguajes existen tres sistemas organizacionales interrelacionados que son: el sistema fonológico, el gramático, y el semántico. Lo cual remarca el hecho de que si el

sistema de reconocimiento de voz maneja una gramática generativa y transformacional se cubren los tres aspectos organizacionales, y por lo tanto se obtendrá un sistema completo.

Además, Southworth et al. mencionan que cada lengua posee un grupo de sonidos, un grupo de elementos gramaticales, un grupo de reglas gramaticales, que indican cómo combinar los elementos gramaticales para construir oraciones bien formadas sintácticamente, y un grupo de significados. Para describir estos sistemas, los lingüistas han encontrado útiles las siguientes unidades: características distintivas de los sonidos, fonemas, morfemas y componentes semánticos.

Las reglas fonológicas especifican la presencia de cualquier alófono de cada fonema de acuerdo al ambiente fonológico. Esquemáticamente /x/= [y], [z], ... , donde el sonido real y puede ser asociado al fonema x cuando se presenta bajo tal ambiente, etc. La entrada y la salida de la regla no es precisamente el símbolo escrito que representa al sonido real, ni el fonema en si, sino una matriz de características distintivas, sus rasgos fonológicos, para la entrada y la salida (Southworth et al., 1974). Algunos rasgos fonológicos son vocal/no vocal, consonante/no consonante, denso/difuso, grave/agudo, nasal/oral, continuo/interrupto, sonoro/sordo, anterior/posterior, coronal/no coronal, estridencia/no estridencia, bajo/alto (Roca-Pons, 1975). Obsérvese que éstos son binarios. Hagège (1981) hace una crítica sobre diversos tópicos de gramática generativa. Dentro de una de ellas, habla sobre los diferentes tipos de entrada a las reglas; en un principio fueron alófonos, después morfemas, posteriormente rasgos

binarios y está volviéndose a usar el fonema. Los tratados que se han mencionado como referencia manejan principalmente los rasgos binarios.

Las reglas morfológicas indican "las variaciones de los fonemas dentro de los morfemas, o sea, dentro de las unidades mínimas de significación que constituyen estos últimos. Estas variaciones fonológicas de los morfemas pueden ser debidas a fenómenos fónicos generales en la lengua" (Roca-Pons, 1975, p. 154), o morfológicas. Se han formulado reglas para generar los afijos, subfijos y prefijos de una palabra dada, para las conjugaciones verbales, y algunas aplicaciones a la Lingüística Diacrónica.

Ya se habló un poco de Lingüística y las disciplinas que la forman. En la siguiente sección se explica qué es el habla y cómo se realiza.

3.2.1 El habla

El habla consiste de la emisión de sonidos articulados, formando palabras en cierto idioma, y se realiza mediante el proceso de fonación. De acuerdo a Dubois et al. (1983):

> La *fonación* es la emisión de los sonidos del lenguaje mediante un conjunto de mecanismos fisiológicos y neurofisiológicos, cuyas principales etapas son la producción del soplo a través de un movimiento respiratorio específicamente adaptado al acto verbal, la producción de la voz mediante la vibración de las cuerdas vocales, la modulación de la voz en función de las unidades fónicas[1] que

[1]"El termino *fónico* se aplica a todo aspecto relativo a los sonidos del lenguaje, tenga o no importancia lingüística." (Dubois et al., 1983, p. 285).

> se han de realizar mediante la excitación de los diferentes resonadores. (p. 282)

Las unidades fónicas, o unidades del habla, son dos, "las oraciones, delimitadas entre sí por los tonemas terminales que cada idioma utilice" (Gili, 1978, p. 165) y el grupo fónico. "Entendemos por *grupo fónico* cada uno de los miembros en que se divide la oración, *de pausa a pausa*. Dos cosas caracterizan al grupo fónico: la unidad de significación y la necesidad de inhalar aire suficiente para seguir hablando" (Gaona, 1949, p. 87).

> De estas pausas normales, presentes siempre al principio y final del grupo fónico, se distinguen las situadas en el interior del grupo fónico, llamadas virtuales porque pueden realizarse o no. El segemento o la mínima secuencia de segmentos[2] denotada de significado y susceptible de ser aislado por pausas es la palabra. (Real Academia Española [RAE], 1973, p.10)

La palabra puede, por consiguiente, hallarse entre pausas virtuales, entre una pausa normal y otra virtual, entre una pausa virtual y otra normal, o entre dos pausas normales. "El grupo fónico se halla organizado en sílabas" (p. 12) y cuando se presenta una pausa virtual se realiza en el límite silábico, es decir entre dos sílabas.

3.2.2 Cualidades del sonido

Las cualidades del sonido articulado, como las de todo sonido, son intensidad, tono, timbre y cantidad.

---

[2]Sonido.

Acerca de la intensidad, Navarro (1957) nos dice que la intensidad:

> Es el mayor o menor grado de fuerza espiatoria con que se pronuncia un sonido, la cual, acústicamente, se manifiesta en la mayor o menor amplitud de las vibraciones. ... En la intensidad absoluta influyen distintas circunstancias emocionales y lógicas; la intensidad relativa obedece, por su parte, a razones históricas íntimamente unidas a la estructura de cada idioma. Por razón de su intensidad relativa, los sonidos, sílabas o palabras se denominan fuertes o débiles. (p. 25)

Con respecto a tono, Gili (1978) menciona "en el lenguaje, llamamos *tono* a la altura musical de cada sonido. *Entonación* es la curva melódica que la voz describe al pronunciar las palabras, frases y oraciones" (p.54).

> El tono o altura musical del sonido depende de su frecuencia, es decir, del número de vibraciones por segundo. En la voz humana, la frecuencia vibratoria de las cuerdas vocales está determinada por su longitud y por su tensión. La longitud varía de un individuo a otro. Cuanto más largas sean las cuerdas vocales, más grave es el registro de voz; si son cortas, el registro de voz es agudo. (p. 52)

En cuanto a timbre, Navarro (1957) comenta:

> El movimiento vibratorio generador del sonido es, en general, un fenómeno complejo en que intervienen simultáneamente, de una parte, un movimiento vibratorio principal (al. *Grundton*), y de otra, uno o más movimientos vibratorios secundarios (al. *Obertöne*). En el lenguaje, el tono fundamental de cada sonido es, como queda dicho, el que producen las vibraciones de las cuerdas vocales, y los tonos secundarios resultan de las resonancias que aquél produce en

> la cavidad o cavidades formadas en el canal vocal por la especial disposición de los órganos articuladores. A cada cavidad o resonador, según su forma y volumen, le corresponde una nota de una altura determinada (al.*Eigenton*). ... Los sonidos son por su timbre, así como por su tono, agudos o graves, según la altura de la nota que corresponde a su resonador predominante. (p. 24)

Por último, la cantidad conforme a Navarro es:

> La duración del sonido. Todo sonido, para ser perceptible, requiere un mínimun de duración; los sonidos se acercan a este mínimun o se alejan de él, según la mayor o menor rapidez con que se habla. Cantidad absoluta es la que representa númericamente la duración de un sonido a base de la unidad de tiempo; cantidad relativa es la que expresa esa misma duración en relación con la de los demás sonidos; se habla de cantidad absoluta si se dice por ejemplo, que la vocal acentuada de *señor*, en un caso determinado, ha durado 20 centésimas de segundo; y se habla de cantidad relativa si se dice que esta vocal tiene ordinariamente una duración doble que la *e* precedente. Por razón de su cantidad relativa, los sonidos se llaman largos, breves, semilargos, semibreves, etc. La cantidad absoluta varía en cada caso según el temperamento, la edad, la emoción, la costumbre, etc., de la persona que habla; la cantidad relativa depende de ciertos principios fonéticos de carácter general y de determinadas circunstancias históricas particulares de cada idioma. (p. 24)

El acento es la intensidad de energía (potencia) de la onda sonora, el cual depende en general de las cuatro características: intensidad, tono, timbre y cantidad. Pero en la Lingüística carece de valor práctico el timbre, según menciona Gili (1978). Para dar una idea más clara de las características del acento, nos apoyamos de nuevo en Navarro (1957), el cual dice:

> Existen, no sólo entre idiomas distintos, sino aun dentro del habla común de cada país, sutiles diferencias regionales y locales, cuya causa principal obedece al acento. El oído suele ser particularmente sensible a estas diferencias; pero su determinación en forma clara y concreta es uno de los puntos más dificiles del estudio de la pronunciación. El sonido sobre el cual recaen principalmente la intensidad, la cantidad y el tono, se llama sonido acentuado. En el caso en que estos elementos se den separadamente sobre sonidos diferentes, conviene distinguirlos en particular, llamándoles, según el elemento de que se trate, acento de intensidad, acento de cantidad y acento tónico o de altura. (p. 26)

En el español se pueden apreciar dos tipos de acento que es el acento de intensidad y el acento tónico; y tiene como característica que la acentuación favorece el alargamiento de los sonidos, es decir, al aumento en la cantidad de los sonidos vocálicos.

3.2.3 Intensidad lingüística

Gili (1978, p. 30) la define como "la intensidad que en el habla da resalte relativo a ciertos sonidos, sílabas, palabras o frases", en relación a la intención de las palabras. El realce o disminución puede realizarse por medios expresivos que no sean precisamente la variación de la intensidad física. Gili explica este concepto más ampliamente y menciona algunas otras características, por ejemplo

> Hay, pues, una intensidad intencional, que puede coincidir o no con la intensidad física lograda en cada caso.
>
> La intensidad sufre modificaciones no sólo de una oración gramátical a otra, sino también entre los elementos sintácticos y las palabras que componen la oración. Dentro de

> cada palabra, varía según las sílabas, y aun es distinta para cada sonido, según su naturaleza y posición en el conjunto silábico. Estas diferencias se producen por causas emotivas, lógicas, históricas y rítmicas[3].
>
> ... Hay en cada palabra aislada una sílaba que, por tradición etimológica e histórica, tiene mayor relieve fonético, a la cual se distingue con los nombres de sílaba fuerte, acentuada o tónica, en contraste con las sílabas débiles, inacentuadas o átonas. Las palabras se agrupan en la frase en torno a ciertos acentos relativamente más fuertes, situados en lugares fijos en el verso y variables en la prosa, que vienen a constituir la principal nervatura rítmica de la lengua hablada. Por último, las sílabas débiles, en la palabra y en la frase, ofrecen una alternancia rítmica en sus intensidades relativas. (p. 30)

Es decir, que la intensidad lingüística se refiere al acento, que puede ser debido a un aumento en la intensidad o en el tono, y el autor menciona algunas características de éste.

Las palabras debido a la posición de su acento se clasifican en agudas, graves y esdrújulas, si llevan el acento en la última, penúltima o antepenúltima sílaba. Existen casos de palabras con acento en otras sílabas, pero éstas se dan en palabras compuestas como fuérte-ménte, así-mísmo, déja-se-ló, explíca-me-ló. Además en palabras monosilábicas no existen un contraste real.

Otro aspecto muy importante es la entonación.

> La estructura prosódica de nuestras palabras, proviene directamente, en la mayor parte de los casos, de la acentuación latina. El acento recaía, en latín, sobre la penúltima sílaba de las palabras cuando esta sílaba era

---

[3]Ritmo: "el oído, por lo que al acento se refiere, cree percibir un movimiento alternativo de aumento y disminución, en virtud del cual las sílabas débiles, a partir de la sílaba fuerte de cada grupo, se distinguen entre sí, destacándose u obscureciéndose sucesivamente" (Navarro, 1957, p. 195).

> larga, y sobre la antepenúltima cuando la penúltima era breve ... a través de las más graves transformaciones, la sílaba portadora del acento ha mantenido, generalmente, en español su identidad sustancial con la correspondiente base latina. (Navarro, 1957, p. 182)

Una de sus características es que "la entonación de la palabra aislada sigue al acento de intensidad, de manera que la sílaba acentuada es también la de tono más agudo." (Gili, 1978, p. 56). Un último aspecto por comentar antes de pasar a los estilos de pronunciación, es que: "bajo una misma forma se dan dos o tres palabras distintas, que fonéticamente sólo se diferencian por el lugar que en cada una de ellas corresponde el acento de intensidad: límite, limite, limité; célebre, celebre, celebré; depósito, deposito, depositó; miro, miró; calle, callé; llamo, llamó, etc. " (Navarro, 1957, p. 183).

3.2.4 Estilos de pronunciación en español

Harris (1975) hace mención a cuatro estilos de pronunciación: largo, andante, allegretto y presto. Definiéndolos de la siguiente manera:

> Largo: muy lento, deliberado, preciso en exceso; como, por ejemplo, cuando se intenta comunicarse con un extranjero que apenas sabe la lengua, o cuando se corrige un malentendido debido a una conexión telefónica deficiente.
>
> Andante: moderadamente lento, esmerado pero natural; como cuando se pronuncia una conferencia o se da una clase en una aula grande sin amplificación electrónica.
>
> Allegretto: moderadamente rápido, despreocupado, conversacional. En muchas ocasiones se puede alternar entre Andante y Allegretto a mitad de párrafo o incluso en medio de oración.
>
> Presto: muy rápido, completamente descuidado.

> ... El habla culta de la ciudad de México-goza por lo general de prestigio entre los hispanoamericanos, al mismo tiempo que se le conceptúa como esmerado en extremo e incluso ligeramente afectado en cuanto a pronunciación. (p. 24)

Estos también son conocidos como pronunciación: afectada, neutra, familiar y descuidada. Conforme se va incrementado la velocidad en el habla, los sonidos, y particularmente los vocálicos, pierden perceptibilidad.

3.3 Fonética y Fonología (o Fonemática)[4]

Alarcos (1965) define a la fonética y a la fonología de la siguiente manera:

> La fonética estudia los elementos fónicos en sí, en su realidad de fenómenos físicos y fisiológicos, y se plantea el problema de cómo tal sonido y tal otro son pronunciados, y qué efecto acústico producen, pero olvida por completo la relación que tienen con una significación lingüística; puede definirse como la ciencia del plano material de los sonidos del lenguaje humano. (p.28)
>
> Entenderemos por *fonología* la disciplina lingüística que se ocupa del estudio de la función de los elementos fónicos de las lenguas, es decir, que estudia los sonidos desde el punto de vista de su funcionamiento en el lenguaje y de su utilización para formar signos lingüísticos[5]. (p. 27)
>
> La función de estos elementos fónicos varía según la unidad semántica en que se estudien. Puede escogerse como tal la *palabra* o sus componentes semánticos más pequeños, los *semantemas* y *morfemas* (cuando éstos conservan su significado propio, estén o no agrupados en palabras), o bien la *frase* o agrupación de palabras. Según atendamos a la función que cumplen los elementos fónicos en cada una de las

---

[4]El término Fonología es más común en Europa.

[5]El significante (la expresión) y el significado (el contenido) constituyen el signo lingüístico.

> dos unidades semánticas consideradas, hay que distinguir la *fonología de la palabra* (o de sus componentes, morfemas y semantemas) y la *fonología de la frase*.
>
> ... En la fonología de la palabra, los elementos fónicos deben diferenciar la significación de unas palabras de otras, de unos morfemas de otros, de unos semantemas de otros, y en la fonología de la frase deben distinguir unas frases de otras.
>
> Esta es la *función diferencial o distintiva;* por ejemplo, en las palabras *ala, ara, asa, ama, haga, ata, hacha,* las consonantes l, r, s, m, g, t, ch tienen una función distintiva, ya que gracias a ella la significación de esas palabras permanece distinta; la entonación ascendente o descendente permite distinguir entre las dos frases *¿dónde está?* y donde está, y por ello cumple una función diferencial. Pero junto a la función distintiva, los elementos fónicos cumplen otra función, consiste en separar, dentro de la cadena hablada, unas unidades semánticas de otras, sean palabras o frases. Esta es la *función delimitativa o demarcativa,* que permite aislar entre sí las palabras (o los semantemas y morfemas) o las frases.
>
> ... Mientras los elementos fónicos que desempeñan su función dentro de la unidad semántica simple (palabra, o morfema o semantema) varían extraordinariamente de una lengua a otra, los elementos fónicos con función dentro del cuadro de la frase son más generales y semejantes de una lengua a otra. (p.36)

Para tratar de dar una idea más completa de los fenómenos que estudian la fonética y la fonología incluyo lo siguiente (RAE, 1973)

> Fonética y Fonología estudian los sonidos, pero con fines diferentes. La primera establece el repertorio de sonidos de una lengua, con arreglo a las particularidades y a las más pequeñas diferencias articulatorias perceptibles. ... Trata de establecer una correlación, lo más exacta posible, entre la descripción articulatoria y la realidad idiomática del habla. Distingue, por ejemplo, en español dos clases de b (rombo, robo) por sus particularidades articulatorias, lo mismo que se distinguen en el relieve sonoro del habla comunicando un

> especial carácter idiomático a la pronunciación española. La Fonología organiza los sonidos en sistema, valiéndose de sus caracteres articulatorios y de la distribución de estos sonidos en la cadena sonora del habla. Establece así unidades de sonidos que reciben el nombre de *fonemas*. Los fonemas se caracterizan por su función significante, es decir, por su capacidad para diferenciar significaciones. Así ocurre, por ejemplo, con las unidades r, rr y b en caro, carro y cabo. (p. 15)

3.4 Grafías, fonemas y alófonos

La grafía es un "signo o conjunto de signos con que se representa un sonido o se escribe una palabra" (Lara, 1986, p. 251). Es decir, a los símbolos utilizados en el abecedario y en los otros tipos de alfabetos se les llama grafías; pero en particular a las grafías utilizadas en el alfabeto fonético se les llama alófonos y a las utilizadas en el fonológico se les denomina fonemas. Para distinguir entre grafías, alófonos y fonemas se enmarcan estos dos últimos; entre corchetes, los alófonos, y entre diagonales, los fonemas.

Con respecto a los fonemas, Gaona (1949) nos brinda varias definiciones recopiladas de diferentes autores para dicho concepto, y son:

1. El fonema corresponde al concepto abstracto del sonido como unidad fonética y semántica.
2. El fonema representa el tipo que da unidad a la variedad de los sonidos.
3. El fonema es la unidad mínima de sonido diferenciador.
4. Los fonemas no son sino modelos o tipos ideales de sonidos.

5. Un fonema es cada uno de los tipos de articulación que forman el sistema fonético de una lengua.
6. Cuando dos sonidos pueden alternar en la misma palabra sin alterar su significado, son sonidos que pertenecen al mismo fonema; si alteran el sentido, son fonemas diferentes.

De acuerdo a la sección anterior los fonemas se definen a partir de su caracter contrastivo, diferenciador, pero además tiene una función delimitativa. Aparte se habla de fonología de la palabra y fonología de la frase.

En el libro de la Real Academia (1973) se enfatiza que el grupo de sonidos que son fonéticamente semejantes y que pertenecen a un mismo fonema pueden poseer algún o ningún contorno en común[6], y que cada uno de estos sonidos son, pues, una variante combinatoria, o variante posicional o *alófono* de dicho fonema.

3.5 Alfabetos

3.5.1 Ortográfico

Con respecto al alfabeto ortográfico Bello (1951) menciona que:

> En las palabras, voces, vocablos o dicciones de que se compone la lengua, debemos notar primeramente los sonidos de que constan; los cuales son vocales o consonantes.
>
> Las vocales, pueden por sí solas, pronunciarse fácil y distintamente: a, e, i, o, u.
>
> Las consonantes, al contario, no pueden pronunciarse con facilidad y claridad si no se les junta algún sonido vocal.

---

[6]Se nombra contorno o marco de un sonido, a los sonidos contiguos que se presentan con dicho sonido.

> Tales son las que se escriben con las letras b, c, ch, d, f, g, gu, j, l, ll, m, n, ñ, p, qu, s, t, v, y, z. (p. 321)

Falta agragar la h, k, r, rr, w, x que por algún extraño motivo no aparecen en la enumeración, además de presentar g y gu a la vez. Existen algunas discrepancias entre autores, entre otros, el incluir tal o cual grafía dentro del alfabeto, como en el caso de las dobles (gu, ll, qu, rr) o de las usadas en palabras de origen extranjero (k, w, x)[7]. Gili (1978) nos dice al respecto que

> El alfabeto ortográfico no es más que una representación aproximada de la pronunciación. En algunas lenguas, como el francés y el inglés, la ortrografía representa a menudo etapas históricas de la pronunciación distintas del estado actual de dichos idiomas. Además, la ortografía, en todos los países cultos, se complica más o menos con exigencias etimológicas. El alemán, el italiano y el español practican un sistema ortográfico menos complicado y, en general, más cercano a la pronunciación real. Pero aun así, el alfabeto usual de estas lenguas es insuficiente para representar con exactitud lo que se pronuncia. (p. 74)

3.5.2 Sistemas fonéticos y fonológicos

Gili (1978) comenta:

> La Fonética necesita un medio gráfico para representar sonidos pertenecientes a numerosos idiomas y dialectos de escasa o nula tradición ortográfica, o matices particulares que, aun en lenguas de alta cultura literaria, no tienen signos que los representen. Para subsanar estas dificultades se han ideado numerosos sistemas de trascripción fonética, ya como instrumento para registrar pronunciaciones de una lengua

---

[7]Por ejemplo, la x que generalmente los autores españoles la omiten.

> determinada, ya con la aspiración de ser aplicados a la Fonética universal. Fácilmente se comprende que no puede haber sistema alguno tan general que prevea todos los sonidos posibles. Por esto, ninguno de los intentos realizados hasta ahora ha conseguido satisfacer a todos los lingüístas. (p. 75)

Navarro (1957) piensa:

> El alfabeto fonético tiene por objeto representar lo más exactamente posible, por medio de la escritura, los sonidos del lenguaje. En la escritura fonética, cada sonido debe ir siempre representado por un mismo signo, y cada signo debe siempre representar un mismo sonido, no debiendo emplearse signo alguno sin valor fonético determinado y constante. El lingüista, el filólogo y el fonético necesitan este alfabeto para poder expresar breve y concretamente los sonidos a que en cada caso se refiere; en la enseñanza de las lenguas vivas el alfabeto fonético sirve para facilitar el conocimiento de los sonidos de cada idioma, y para representar prácticamente la pronunciación que a cada palabra corresponde. (p. 31)

Para establecer esos alfabetos fonéticos se ha analizado el mecanismo fisiológico que producen los sonidos lingüísticos. Navarro concluye lo siguiente:

> Para darse cuenta exacta de la naturaleza y estructuras propias de una articulación es, pues, necesario considerar en su conjunto la disposición que afecta cada uno de los órganos del canal vocal en el momento en que dicha articulación se produce, debiendo, ante todo, ser tenidos en cuenta los siguientes elementos: a) *punto de articulación,* fundamento de la división de las articulaciones en labiales, dentales, alveolares, palatales, etc.; b) *modo de articulación,* fundamento de las diferencias entre oclusivas, fricativas, etc.; c) *función de las cuerdas vocales*, base de la diferencia

> entre sonoras y sordas; y d) *función del velo del paladar*, base de la diferencia entre bucales y nasales. (p. 22)

El sistema fonológico, como se dijo en la sección de Fonología y Fonética, organiza los sonidos valiéndose de sus caracteres articulatorios y de la distribución de estos sonidos en la cadena sonora del habla.

El fonema se representa, generalmente, con el símbolo más sencillo de entre sus alófonos; y los otros alófonos, en general, utilizan aparte de ese signo una o dos marcas distintivas que nos indican el cambio de alguna o algunas de sus características.

Existen diferentes notaciones, es decir alfabetos. Al respecto de éstos Heffner (1950) menciona que desde nuestro siglo, 1900, se han propuesto muchos sistemas de trascripción fonética, que aparecen editados a partir de la página 67 del libro de "Fletcher, Harvey. *Speech and Hearing*. New York: Van Nostrand, 1929" (p. 69). También menciona que, la causa básica del conflicto de opiniones es el deseo de crear un alfabeto fonético universal, pero debido a las diferentes pronunciaciones de una misma letra y al problema de conocer todas las diferentes articulaciones de los sonidos en los diferentes idiomas, ha existido algo de controversia. Para finalizar, Heffner declara que el alfabeto fonético más utilizado es el de la "International Phonetic Association (I.P.A)" o también conocida como "L'Association Phonétique Internationale" (p.69) y en las páginas 70 y 71 presenta un cuadro del sistema fonético mediante ejemplos. En el libro de Roca-Pons (1975) en la página 110 viene un cuadro del alfabeto fonético internacional en base a las articulaciones.

Navarro (1957) dice algo similar y menciona los sistemas propuestos por "Böhmer, Ascoli y Gillieron" (p. 31). En los paises de habla hispana se usa mucho el sistema propuesto por la Revista de Filología Española (tomo II, 1915, páginas 374-376) el cual está basado en el alfabeto internacional; y se tiene como referencia que éste se emplea en la mayor parte de las revistas de Lingüística hispana y por los principales tratadistas en España y en América.

Sin embargo, se utilizará el presentado por (RAE, 1973), pues éste ya está adaptado a nuestro idioma. A continuación, se presenta dicho repertorio de fonemas y alófonos.

/i/ alto anterior. Alófonos: [i, i̯, j, î̯].
/e/ medio anterior: [e, e̯].
/a/ bajo: [a, a̯].
/u/ alto posterior redondeado: [u, u̯, w, û̯].
/o/ medio posterior redondeado: [o, o̯]. ...
/p/ bilabial oclusivo sordo. Alófonos: [p].
/b/ bilabial sonoro o ensordecido: [b, b̥, b̯̊].
/t/ dental oclusivo sordo: [t].
/d/ dental sonoro o ensordecido: [d, d̥, d̯̊].
/k/ velar oclusivo sordo: [k].
/g/ velar sonoro o ensordecido: [g, g̊].
/f/ labiodental fricativo sordo: [f].
/θ/ *interdental fricativo: [θ, z̪].
/s/ alveolar fricativo: [s, s̪, z, z̪].
/x/ velar fricativo sordo: [x].
/ĉ/ palatal africado sordo: [ĉ].
/r/ alveolar vibrante simple sonoro: [r, ɹ].
/r̄/ alveolar vibrante múltiple sonoro: [r̄].
/l/ alveolar lateral sonoro: [l, l̪ , l̥, l̯̊].
/l̯/ *palatal lateral sonoro: [l̯].
/m/ bilabial nasal sonoro: [m].
/n/ alveolar nasal sonoro: [n, n̪, n̥, ŋ, m̥, n̯̊].

/ɲ/ palatal nasal sonoro: [ɲ].
/y/ palatal sonoro: [y, ŷ].
/ʍ/ velar redondeado sonoro: [ʍ].

> Un mismo sonido puede asignarse como alófono a varios fonemas, si así resulta más conveniente para la simetría del sistema, a condición de que se determine en cada caso el contorno. El sonido [m] aparece en la lista como alófono de /m/ y de /n/. Bastará agregar al cuadro que [m] es alófono de /n/ cuando precede a /p, b, f, m/ en posición heterosilábica y sólo en este caso. Se considera también a [ɲ] alófono de /n/ cuando precede a /ĉ, ʝ, y/, y a [ʝ] alófono de /l/ cuando precede a /ĉ, y/, también en posición heterosilábica. Transcribiremos, pues, fonológicamente /kánpo/, /ánco/, /kólĉa/, etc., y no /kámpo/, /áɲĉo/, /kóʝĉa/. (p. 34)

Los fonemas señalados con asterisco no se emplean en México, debido a los efectos del seseo, s ≈ θ, y del yeísmo, y ≈ ʝ.

3.6 Neutralización de oposiciones

Alarcos (1965) lo define de la siguiente manera: "en determinadas circunstancias, una o varias de las propiedades pertinentes características de un fonema cesa de ser distintiva, y este fonema deja de diferenciarse del fonema o fonemas de que normalmente se distingue. Sabemos que a este fenómeno se le llama *neutralización*" (p. 180). El archifonema o arquifonema es una "unidad fonológica en la cual se suponen reunidos los rasgos fonológicos comunes a los sonidos que no contrastan en dichas posiciones" (RAE, 1973, p. 35).

Debido a algunas diferencias de opinión tocante a este punto, en cuanto a transcripciones se refiere, no se incluyó en el alcance de la

tesis y por lo tanto no se profundizó más en el conocimiento de este tema.

La teoría de neutralización resuelve de otra manera el problema de la pertenecia de alófonos que pueden asociarse a varios fonemas. Así, por ejemplo, la presencia de /m/ o /n/ está regulada automáticamente por la naturaleza de la consonante a la que precede. En el archifonema "se suponen reunidos los rasgos fonológicos comunes a los sonidos que no contrastan en dichas posiciones" (p. 35).

3.7 Cambios en el acento de intensidad y en la entonación

> Existe una estrecha relación entre el acento de intensidad y la función sintáctica que la palabra desempeña. Se acentúan normalmente los verbos, adverbios, sustantivos, adjetivos y las formas enfáticas de los pronombres. La inacentuación se manifiesta, por lo contrario, en aquellas palabras que, como los artículos, preposiciones y conjunciones, desempeñan el papel secundario de relacionar entre sí los elementos más importantes de la oración. (Navarro, 1957, p. 186)

"La correspondencia entre la función sintáctica y el acento llega hasta el punto de que hay sustantivos, adjetivos y adverbios que se pronuncian como formas débiles al acomodarse a ser empleados en ciertos casos en un concepto inferior a su propio valor gramatical" (p. 186) y viceversa. Por ejemplo: Espera, pues, y escucha lo que te voy a decir; por eso, amigo mío, te recuerdo.

Un ejemplo también bastante ilustrativo dicho por Navarro es con respecto a la palabra "mejor", que tiene su acento de intensidad en la o; pero al pronunciarla en la frase "Ese es tu mejor testigo", las dos sílabas se dicen casi con la misma intensidad; y si pronunciamos "Tu

testigo es el mejor", se tiende generalmente a debilitar la última sílaba.

Gili (1978) menciona otro aspecto que es la separación entre intensidad y tono. "En la frase, la entonación expresa valores sintácticos y emotivos, y dentro de ellos ofrece modalidades propias de cada lengua o dialecto. La correlación entre intensidad y tono se debilita o anula en la frase, porque la curva melódica tiene una función especialmente oracional, y a esta función se supeditan todos los factores fonéticos" (p. 56) que va regida especialmente por sus relaciones gramaticales, semánticas, movimientos afectivos y usos idiomáticos o dialectales. Mediante el acento tónico se puede saber qué tipo de frases u oraciones existen, por ejemplo del tipo afirmativa, interrogativa, subordinada, enumerativa, complementaria, exclamativa, aseverativa, petioria, con paréntesis, etc. (Navarro, 1957).

Un último fenómeno es el de la reducción de grupos vocálicos, formándose diptongos y triptongos. Al respecto de esto, Navarro (1957) dice:

> Los sonidos comprendidos dentro de un mismo grupo fónico, entre dos pausas sucesivas de la articulación, cualquiera que sea el número de palabras de que conste dicho grupo, aparecen en la pronunciación tan íntima y estrechamente enlazados entre sí como los sonidos que componen una misma palabra. Este enlace de los sonidos, ya sea considerado en la palabra aislada, o ya en el grupo fónico, da lugar en español a importantes modificaciones fonéticas.
>
> ... Cuando dentro de una misma palabra o grupo fónico aparecen juntas dos o más vocales sucesivas, lo primero que importa saber es si estas vocales se han de pronunciar en sílabas distintas, o si todas o algunas de ellas han de

> agruparse en una sola sílaba. Aun en el caso de que cada vocal forme de por sí misma una sola sílaba, el paso de una vocal a otra vocal inmediata se hace siempre sin interrupción de sonoridad. (p. 147)
>
> Nuestra pronunciación tiende, preferentemente, a convertir, siempre que es posible, todo conjunto de vocales en un grupo monosilábico.
>
> ... En general, en el lenguaje rápido, la reducción de los grupos vocálicos a una sola sílaba es más frecuente que en lenguaje lento; si las vocales no son acentuadas, su reducción, en igualdad de circustancias, se produce más fácilmente que si alguna de ellas lleva acento; si son iguales, se contraen asimismo más fácilmente que si son diferentes, y si proceden del enlace de palabras distintas [sinalefa], mejor que si se hallan dentro de una misma palabra [sinéresis]. (p.148)
>
> Los prosodistas se han esforzado inútilmente en reducir a reglas fijas tales vacilaciones; dada la libertad de que la lengua dispone en este punto, lo único posible es tratar de señalar en cada caso la forma que hoy tiene un uso más corriente en la pronunciación correcta. (p.149)
>
> Al juntarse en una misma sílaba acentuada, de intensidad relativamente débil, dos o más vocales diferentes, el acento, cualquiera que sea su posición etimológica, cae sobre la vocal más perceptible[8]. Una vocal de perceptibilidad débil sólo puede predominar en un grupo silábico sobre otra vocal de perceptibilidad mayor, cuando un fuerte acento espiratorio, cayendo especialmente sobre dicha vocal débil, refuerza su sonido y le hace destacarse sobre la vocal que le acompaña. (p.170)

Sólo me gustaría agregar que "dos consonantes iguales, en contacto, se pronuncian como si se tratase de un sola consonante relativamente larga y repartida entre las dos sílabas inmediatas" (Navarro, 1957, p. 175); "la duración de ésta no es, pues, igual precisamente a la suma de dos consonantes simples; pero a falta de otro medio mejor empleamos

---

[8]Es la que se distingue más claramente.

en la escritura fonética una consonante doble para representarla" (p. 176); por ejemplo: innumerable, innato, obvio, subvención.

3.8 Señales demarcativas

En la Sección 3.3 se hizo alusión a la función demarcativa de los fonemas. Las señales demarcativas del español pueden ser clasificadas como positivas o negativas, fonemáticas o afonemáticas, simples o complejas. Son positivas si indican el límite entre palabras, y negativas en el caso contrario. Son fonemáticas cuando los sonidos involucrados tienen una función distintiva y afonemáticas en caso contrario. Son simples cuando involucran un sonido y complejo en caso contrario.

Nuevamente Alarcos (1965) nos habla bastante al respecto y de ello incluyo lo siguiente:

> No existen señales positivas fonéticas simples en el español, ya que ningún fonema aparece exclusivamente en la posición inicial o final de una palabra; pero hay indicadores complejos, formados por la combinación de uno de los fonemas o archifonemas susceptibles de comenzar palabra. (p. 206)

Son estos grupos[9]:
D'p: verdad'palmaria; *D'f:* el abad'falleció; *D't:* observad'todo; *D'θ:* virtud'celosa; *D'd:* la maldad'de Juan; *D'ĉ:* la sobriedad'china; *D's:* gastad'sin tasa; *D'ʃ:* el talud'llega hasta la cabaña; *D'ɾ:* salud'robusta; *N'ŋ:* son'ñoños; *L'y:* el'yeso; *L'l:* papel'liso; *L'ʃ* costal'lleno; *R'ʃ:* amanecer'lluvioso; *R'ɾ* por'reír; *θ'f:* coz'furiosa; *θ'θ:* diez'cirios; *θ'ĉ:* fugaz'chubasco; *θ'y:* veloz'yegua; *θ's:* la luz'solar; *θ'x:* actriz'genial; *θ'ʃ:* la emperatiz'lloró; *θ'ɾ:* voz'ronca; *s's:* los'setos; *s'ʃ:* campos'llanos. [subrayado agragado] (p. 207)

Estos grupos tienden a reducirse al segundo fonema y, por lo tanto, su valor demarcativo es relativo. Recuerde, en nuestro caso θ=s y ʃ=y.

Dado que no se van a manejar los archifonemas sólo daremos *por válidas* las combinaciones ilustradas por los ejemplos.

Alarcos menciona, además, que en posición interior de palabra como demarcadores negativos simples se encuentra la /r/, y los archifonemas B (=p/b) y G (=k/g). La /ĉ/ y la /ŋ/ no aparecen a final de palabra y dentro de palabra sus contornos son vocálicos. La presencia de /s/ entre consonantes indica que es interior de palabra, y los grupos de fonemas /ʃ/+/i/ y /y/+/i/ no se presentan al inicio de palabra.

En otra sección del libro de Alarcos se listan las combinaciones de dos o tres consonantes que se presentan dentro de palabra, haciendo uso de los archifonemas. Los ejemplos citados, son:

"(obvio), apto, obtener, abdicar, abcisa, inepcia, subyugar, abyecto, ábside, cápsula, subconsciente, (subgobernador),

---

[9]D, N, L y R son archifonemas y ellos equivalen a t/d, m/n/ŋ, l/ʃ y r/ɾ.

objeto, submarino, abnegación, subrayar; advertir, fútbol, adyacente, adquirir, (cepadgo), (marzadga), adjetivo, atmósfera, admirar, étnico, atlas, adláteres; acto, amígdala, acción, acceso, sexo, dogma, bracman, signo, técnico; campo, cambio, confuso, canto, conde, onza, concha, cónyuge, cansar, ronco, manga, ángel, inmóvil, perenne, connivencia, enlace, desenlodar, conllevar, honra; golpe, calvo, solfa, salto, saldo, alzar, colcha, pulso, calco, galgo, aljibe, alma, alnado, alnafe, malrotar; arpa, hierba, orfebre, corte, cardo, zurcir, marcha, (interyacente), terso, terco, argamasa, marjal, arma, carne, perla; gazpacho, cabizbajo, azteca, lezda, bizco, juzgar, diezmo, jazmín, bizma, rebuzno, (guzla); aspa, resbalar, esfera, estera, desdén, ascenso, deschanzado, deschuponar, deshielo, desyemar, disyunción, asco, rasgo, desjarretar, mismo, asno, muslo, israelita. (p. 190)

/Npr/: compra. */Nbr/:* sombra. */Nfr/:* infringir. */Ntr/:* contrario. */Ndr/:* alondra. */Nkr/:* incrustar. */Ngr/:* sangre. */Npl/:* templo. */Nbl/:* rambla. */Nfl/:* conflicto. */Nkl/:* ancla. */Ngl/:* inglés.

/Lpr/: salpreso. */Lbr/:* albricias. */Lfr/*: (Alfredo). */Ltr/:* faltriquera, altramuz. */Ldr/*: saldrá. */Lkr/*: alcrebite. */Lpl/*: escalplo.

/Rpr/: intérprete. */Rtr/:* pertrechos. */Rpl/*: perplejo. */spr/*: desprender. */sbr/*: desbrozar. */sfr/:* disfraz. */str/*: astro. */sdr/*: esdrújulo. */skr/*: escrúpulo. */sgr/*: esgrimir. */spl/*: explicar. */sfl/*: trasflor. */skl/*: esclusa. */sgl/*: desglosar.

/θkl/: mezclar.

... Raras veces otra consonante diferente de las citadas aparece en la distensión ante un grupo binario: *actriz,* y las voces sentidas como compuestas: *superfecto, subclase, perclorato,* etc. [subrayado agregado] (p. 194)

En referencia a fonología de la frase se tiene que las propiedades prosódicas, que son el acento y la entonación, tienen valor demarcativo. "El acento, por ser culminativo, señala el centro de intensidad de cada palabra, y, por ser libre, puede indicar el límite de las palabras de manera diversa: la presencia de un acento indicará que la palabra acaba en la sílaba en que recae [aguda]: *canción*, o en la

siguiente [grave]: *canto,* o dos sílabas después [esdrújula]: *cántaro.* " (Alarcos, 1965, p. 208); y haciendo un análisis de la curva melódica de la frase u oración se puede determinar si ésta es del tipo: enunciativa, aseverativa, enumerativa, interrogativa, exclamativa, si es subordinada o complementaria (Navarro, 1957).

Alarcos (1965) menciona también combinaciones de señales demarcativas afonemáticas, pero no se incluyen aquí debido a que se piensa considerar tan sólo fonemas.

Se puede agregar de (RAE, 1973) las siguientes observaciones: (1) ninguna palabra empieza con /r/, (2) al inicio se presentan /bl, br, dr, fl, fr, gl, gr, kl, kr, pl, pr, tl, tr/, (3) en final de palabra aparecen /d, s, n, l, r/ y algunas palabras de diferente procedencia (términos antiguos o de introducción reciente) terminan con /p, b, f, t, k, g, x, m/[10], la terminación en dos consonantes dista mucho de ser una forma canónica y se encuentra /ls, ns, rs, nk, rd, nd, lts, ps, ks, tl, nch/, (4) la simplificación, con la eliminación de la primera consonante, afecta los grupos cn, cz, gn, mn, pn,ps, pt, tm, (5) dos sílabas sucesivas con acento de intensidad en general no pertenecen a la misma palabra.

3.9 Resumen

El español es un idoma romance que tiene su origen en el latín vulgar, y es hablando en España, México, Centroamérica, Sudamérica y en las islas Canarias. Según estadísticas realizadas en 1977, con respecto a los idiomas que tienen sus origen en el latín, ocupa el

---

[10]Fonemas que algunas veces no se realizan.

primer lugar; es la lengua natal de aproximadamente 250 millones de personas y 280 millones lo usan como lengua oficial.

Para poder conocer las características de un idioma es necesario hacer referencia a la ciencia que estudia el lenguaje, es decir a la Lingüística. La Lingüística es la "ciencia que estudia la estructura de las lenguas, la manera como se hablan, su historia, su origen y los fenómenos humanos que se manifiestan en ellas" (Lara , 1986, p. 307). Las lenguas pueden ser analizadas desde dos tipos de enfoques descriptivo, o sincrónico, y histórico, o diacrónico. Algunas de las principales disciplinas de la Lingüística son: Fonética, Fonología, Gramática, Semántica, Lexicología, Dialectología, Lingüística Generativa Transformacional, y Sociolingüística. La gramática se divide en Morfología y Sintáxis.

Dentro del área de reconocimento de voz, la Lingüística Generativa Transformacional juega un papel muy importante dado que establece la relación entre el sonido y su interpretación semántica. Dada la teoría de que en todo lenguaje existen tres sistemas organizacionales interrelacionados que son el sistema fonológico, el gramático y el semántico, entonces una gramática generativa y tranformacional es completa pues contempla estos tres factores.

La Lingüística Generativa y Transformacional utiliza reglas fonológicas y morfológicas como herramientas de apoyo. Las reglas fonológicas sirven como medio para relacionar el sonido realizado, alófono, con su modelo ideal, fonema. Las reglas morfológicas nos indican las posibles variaciones de los fonemas dentro de los morfemas; mediante éstas se han formulado reglas para generar

afijos, conjugaciones y aplicaciones a la Lingüística Histórica. Dentro de la gramática se han utilizado o se utilizan fonemas, morfemas y rasgos binarios. Los rasgos binarios son simplemente características que pueden tener dos valores, existe o no existe, y se utilizan en los tres niveles, fonológico, gramático y sintáctico.

La fonética, que estudia el sonido articulado en sí, y la fonología, que estudia el funcionamiento de los sonidos como lenguaje, tienen sus propios sistemas de escritura. En el sistema fonético se trata de representar el sonido real; es decir, las variantes posicionales de los fonemas que son los alófonos. En el sistema fonológico se usa los fonemas. A partir de nuestro siglo se han propuesto muchos sistemas de trascripción fonética. El que se utilizará es el presentado en (RAE, 1973, p. 34) y reproducido en la Sección 3.5.2. El alfabeto ortográfico es tan sólo una representación aproximada de la pronunciación.

El habla, en general, consiste de la emisión de sonidos articulados formando palabras en cierto idioma, y éste se realiza mediante el proceso de fonación. Las unidades del habla son la oración y el grupo fónico. Por unidades del habla se entiende la cantidad de sonido que se emite entre pausas; es decir, en modo continuo.

Las cualidades del sonido articulado son intensidad, tono, timbre y cantidad. El acento depende de estas cuatro características, pues es la potencia de la señal sonora. En el idioma español, se manejan principalmente dos tipos de acentos, el acento de intensidad y el acento tónico; esto quiere decir que el énfasis dado a un sonido se debe principalmente por el cambio en la intensidad o en el tono. Las

palabras debido a la posición del acento se clasifican en agudas, graves y esdrújulas.

Cuando se dicen las palabras en forma pausada, una por una, el acento de intensidad coincide con el acento de tono; pero cuando se realiza esto en forma continua, el acento de intensidad está en relación directa a la función sintáctica de las palabras, y en el acento tónico influyen todo, relaciones gramaticales, semánticas, movimientos afectivos y usos idiomáticos o dialectales. Mediante el acento tónico se puede saber qué tipo de frases u oraciones existe, por ejemplo del tipo afirmativa, interrogativa, subordinada, enumerativa, complementaria, exclamativa, aseverativa, petioria, con paréntesis, etc.

Existen diferentes estilos de rapidez de pronunciación que son largo, andante, allegretto y presto. Conforme se va incrementado la velocidad en el habla, los sonidos, y particularmente los vocálicos, pierden perceptibilidad.

Otros problemas que se presentan en la identificación de los sonidos son la neutralización de características distintivas entre fonemas, surgiendo el concepto de archifonema, y la reducción de grupos vocálicos, formando diptongos y triptongos.

Por último, las señales demarcativas son clasificadas como positivas o negativas, fonemáticas o afonemáticas, simples o complejas. Son positivas si indican el límite entre palabras, y negativas en caso contrario. Son fonemáticas cuando los sonidos involucrados tienen una función distintiva, y afonemáticas en caso

contrario. Son simples cuando involucran un sonido, y complejas en caso contrario.

En la Sección 3.8 se da toda una lista de ellas.

Capítulo 4
Conceptos relacionados al almacenamiento de información en archivos

Se ha tomado como base la sección 11.3 del libro "Estructuras de datos y algoritmos" de Aho, Hopcroft, y Ullman (1988), para explicar los dos tipos de organización y direccionamiento de archivos, a los cuales se hará alusión en los próximos capítulos, y que son archivos con función de dispersión y archivos indizados.

4.1 Definición

Un archivo está constituido por registros, y cada registro contiene la misma secuencia de campos. Los campos pueden ser de longitud fija, con un número predeterminado de bytes, o de longitud variable. "Los archivos con registros de longitud fija se suelen utilizar en los sistemas de administración de bases de datos para almacenar datos muy estructurados. Los archivos con registros de longitud variable se usan típicamente para almacenar información de textos" (p. 360).

Las operaciones básicas con archivos son consultar (recuperar), modificar, insertar o suprimir información en éstos.

4.2 Tipos de organizaciones

Cuando el archivo está desorganizado, los registros están almacenados en cualquier orden, y se realiza una revisión secuencial

de los registros para llevar a cabo cualquiera de las operaciones. A este tipo de organización se le denomina *simple*.

Para accesar la información en forma más eficiente se usan llaves de acceso a la información, donde una llave es un subconjunto de campos que identifica en forma unívoca cada registro. "Otro elemento necesario para lograr operaciones rápidas con archivos es la capacidad de acceder a bloques en forma directa, en vez de recorrer en secuencia los bloques que contienen el archivo. Muchas de las estructuras de datos utilizadas para operaciones rápidas con archivos usarán apuntadores a los propios bloques, los cuales son direcciones físicas de los bloques" (p. 363).

Aho et al. explican cuatro tipos de organización de archivos: en cubetas, ordenados conforme a un valor clave, usando un archivo de índices o un árbol de búsqueda, o en forma aleatoria, usando varios archivos auxiliares; pero aquí sólo se expondrán los primeros dos.

4.2.1 Archivos con función de dispersión

Los registros del archivo se distribuyen en cubetas, donde una cubeta está constituida por uno o varios bloques de información. Los bloques dentro de la cubeta pueden estar desordenados, y por ende, los bloques se encuentran encadenados formando una lista enlazada. A su vez, un bloque está constituido por uno o varios registros de información, y al final de éste cuenta con un apuntador al siguiente bloque.

Se forma una tabla, llamada tabla de cubetas donde se almacena por cada cubeta un apuntador al primer bloque de ésta. Si enumeramos las

cubetas de 0, 1, ..., B-1, una función de dispersión *h*, también conocida como función de *hashing*, hace corresponder cada valor clave con uno de los enteros 0 a B-1. Si *x* es una clave, *h(x)* es el número de la cubeta que puede contener al registro con clave *x*. Mediante *h(x)* se accesa el contenido de la tabla de cubetas, obteniendo el apuntador al primer bloque de la cubeta. Después se realiza una búsqueda secuencial (lineal), dentro de la cubeta, para localizar al registro de clave *x*, si no se encuentra, se concluye que *x* no es la clave de ningún registro. Durante la búsqueda, cada bloque es cargado en memoria. Si la tabla de cubetas no es muy grande, también, es conveniente tenerla en memoria principal, sobre todo si el número de consultas por unidad de tiempo, que se desean hacer, es elevado. Ver Figura 4.1.

El número promedio de accesos a bloques, para llevar a cabo la búsqueda de *x*, es aproximadamente igual al número promedio de bloques en una cubeta; es decir, *n/bk*, donde *n* es el número de registros total, *b* son los que contiene un bloque, y *k* es el número de cubetas. "Una organización de archivos de acceso con función de dispersión bien diseñada, requiere sólo unos cuantos accesos a bloques para cada operación de archivo. Si la función de dispersión es buena y el número de cubetas es más o menos igual al número de registros en el archivo, dividido entre el número de registros que pueden caber en un bloque, entonces la cubeta promedio consta de un bloque" (p. 364).

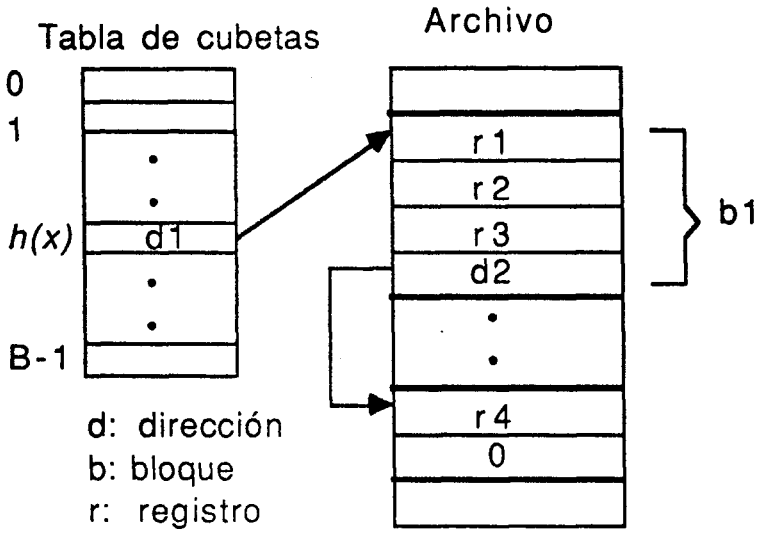


Figura 4.1. Ejemplo de archivo con función de dispersión, ilustrando una cubeta que contiene cuatro registros en dos bloques

4.2.2 Archivos indizados

La idea es tener organizado el archivo en bloques conforme a sus valores claves, generalmente en forma creciente, y contar con un archivo auxiliar, llamado *índice disperso*, donde aparte de almacenar el apuntador al bloque, contenga la clave del primer registro. Entonces, el índice disperso está constiuido por parejas (x,b), donde x es el valor clave del primer regsitro en un bloque y b es el apuntador a éste.

Para localizar el registro con clave x, primeramente se realiza una búsqueda en el achivo de índice, para encontrar la z más grande tal que $z \leq x$, si la encuentra, entonces, cargando el bloque en memoria principal, se realiza una búsqueda sobre el bloque en cuestión. Ver Figura 4.2.

Hay varias estrategias para buscar en el archivo de índices; la más simple es la *búsqueda lineal.* Se lee el archivo índice desde el principio hasta encontrar el par (x,b) o el primer par (y,b), donde $y > x$.

... La búsqueda lineal sólo es posible para archivos de índices pequeños. Un método más rápido es la ***búsqueda binaria.*** Supóngase que el archivo de índices está almacenado en los bloques $b_1, b_2, \ldots, b_n$. Para buscar la clave x, se toma el bloque medio $b_{[n/2]}$ y se compara x con la clave y del primer par de ese bloque. Si $x < y$, se repite la búsqueda en los bloques b1, b2, ..., $b_{[n/2]-1}$. Si $x \geq y$, pero x es menor que la clave del bloque $b_{[n/2]+1}$, (o si n=1 de modo que no hay tal bloque), se utilizará la busqueda lineal para ver si x corresponde al primer componente de un par de índices en el bloque $b_{[n/2]}$. De otra forma, se repite la búsqueda en los bloques $b_{[n/2]+1}, b_{[n/2]+2}, \ldots, b_n$. Con la búsqueda binaria es necesario examinar sólo $[\log_2(n+1)]$ bloques del archivo índice. (p. 365)

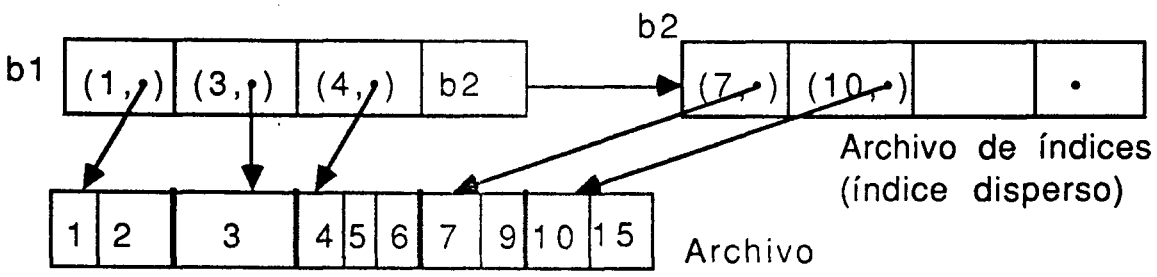


Figura 4.2. Ejemplo de archivo indizado, mostrando sólo los valores claves

4.3 Resumen

Un archivo es una secuencia de registros, donde cada registro consta de la misma secuencia de campos. Los campos pueden ser de longitud fija o variable, y por ende, los registros pueden ser de longitud fija o variable. Los archivos con registros de longitud fija son adecuados

para utilizarse en sistemas administrativos; y los de longitud variable se usan típicamente para almacenar información de textos.

Las operaciones básicas que se pueden realizar con archivos son consultar (recuperar), modificar, insertar o suprimir información en éstos. Cuando los registros están almacenados en cualquier orden, la realización de estas operaciones se efectúa de la manera menos eficiente, pues es necesario hacer una búsqueda secuencial hasta encontrar el campo deseado. Este tipo de organización se denomina organización simple.

Existen organizaciones de archivos en los cuales se puede localizar un registro leyendo sólo una pequeña fracción del archivo completo, el acceso a cada registro es por clave; una clave es un conjunto de campos que identifica en manera unívoca cada registro. Para lograr operaciones rápidas es necesario, además, accesar bloques de información en forma directa. Siguiendo estos lineamientos, existen al menos cuatro tipos de estructuras de datos: organizando la información en cubetas, conforme a su valor clave, en forma aleatoria, y conforme a su valor clave pero usando árboles de búsqueda.

A los archivos que tienen su información organizada en cubetas, se les denomina archivos con función de dispersión, porque dado el valor clave x usan una función, llamada función de dispersión o *hashing* ($h(x)$), que genera el apuntador adecuado para accesar la información contenida en la tabla de cubetas. En dicha tabla, se guardan los apuntadores a cada cubeta. La cubeta a su vez puede estar formada por bloques, y los bloques por registros. Los bloques dentro de la cubeta pueden encontrarse en forma aleatoria, entonces para formar

la cubeta se agrega al final de cada bloque un apuntador al siguiente bloque.

Mediante *h(x)*, se obtiene el apuntador al primer bloque de la cubeta, éste se carga en memoria, se compara secuencialmente las claves de los registros, contenidos en el bloque, con la clave a localizar. Si la búsqueda es infructuosa, se pregunta en los demás bloques que forman parte de la cubeta y si no la encuentra, se concluye que no existe ningún registro con clave *x* dentro de ese archivo. El número promedio de accesos a bloques requerido para una operación es aproximadamente el número promedio de bloques en una cubeta; es decir, *n/bk*, donde *n* es el número total de registros, *b* es la cantidad de registros contenidos por bloque, y *k* es el número de cubetas.

A los archivos que tienen su información organizada de acuerdo a sus valores clave, y que usan tan sólo un índice para posicionarse al inicio del bloque que contenga la clave a localizar, se les denomina archivos indizados. Posteriormente, se puede realizar una búsqueda lineal, es decir secuencial, o binaria. La búsqueda binaria consiste en posicionarse en el bloque intermedio y verificar si la clave del primer registro *y* coincide con el valor de la clave *x* buscada. Si $x < y$ revisa en los bloques anteriores al medio; pero usando la misma lógica, es decir preguntado por la clave contenida en el primer registro del primer bloque intermedio de ese rango. Si $x \geq y$, pero *x* es menor que la primera clave existente en el bloque inmediato superior, se realiza una búsqueda lineal en el bloque intermedio. Si no se cumple la última condición, entonces se revisa en los bloques superiores (pero usando la misma lógica). A el archivo donde se guardan los índices a los

bloques se le llama índice disperso; y contiene pares (x,b), donde x es un valor de una clave y b es la dirección física del bloque en el cual el primer registro tiene la clave con valor x. Con la búsqueda binaria es necesario examinar sólo $[\log_2(n+1)]$ bloques del archivo índice.

Implementación

Capítulo 5
Justificación del alcance de la tesis

Se tratará de explicar en forma detallada los motivos que contribuyeron a establecer el alcance de este trabajo, en cuestión lingüística, al definir el tipo de revisión y el tipo de entrada al sistema, en la aplicación que se le dará a las redes neurales, y el método de acceso a la base de datos. Además se establecerá la clase de información que es proporcionada a la salida de este sistema.

5.1 Tipo de revisión

De acuerdo a lo expuesto en la Sección 3.2 es necesario realizar tres tipos de revisiones, fonológica, gramática y semántica, para poder obtener la forma escrita de la frase hablada. Sin embargo, las revisiones se tienen que hacer precisamente en ese orden, pues no se puede realizar la revisión gramatical si no se cuenta previamente con las palabras o los morfemas, y no tendría caso realizar el análisis semántico de una frase que estuviera mal estructurada.

El punto decisivo que determinó hasta que fase se podría implementar fue el vocabulario, es decir la base de datos, pues no se contaba con nada. Haciendo una búsqueda dentro del ITESM campus Monterrey, se encontró que no existe ningún diccionario o enciclopedia electrónica de palabras españolas, sino tan solo inglesas; pero se

consiguió la lista de palabras que forman el diccionario en español del paquete *Microsoft Word*, para Macintosh.

En un principio, pensé en trabajar sobre esta lista; pero estaba algo inquieta cuestionándome sobre la validez de la base de datos, pues presentaba algunos términos extranjeros; sobre cuál filosofía se iba a manejar, si programar el sistema asumiendo que en la base de datos se almacenará toda la información o no, y si es no, entonces cuáles incluir; acerca de la orientación que se le ha dado a los trabajos desarrollados en los Estados Unidos, que es contar con los morfemas o palabras de más uso en el vocabulario y generar el resto de las palabras; sobre la afirmación mencionada por Allen (1976) donde expresa que no es posible tener una base completa, y en relación a la actual existencia no tan sólo de diccionarios, sino de enciclopedias electrónicas en discos ópticos[1], de palabras inglesas.

Buscando resolver estas dudas, primeramente centré la atención en indagar si en el Instituto se contaba con el Diccionario de la Real Academia, el cual sí se tiene, siendo el más reciente el editado en 1984 que consta de dos volúmenes; el número de palabras que se define no se especifica pero da una idea de la magnitud del problema. Al encontrar el Diccionario de la Real Academia también localizé otros tres libros. En el de Rodríguez (1952) se realizó un recuento de las palabras usadas en el español, buscando en cuarenta fuentes distintas y capturando la información en computadora. El número total

---

[1]También conocidos como compak disc, cuya capacidad de almacenamiento varia desde los 300Mbytes a 100Gbytes de información.

de palabras almacenadas fueron 7,066,637, encontrándose 62,888 palabras distintas y 20,542 unidades léxicas.

Las unidades léxicas que usaron son lexemas, pero cuyas palabras derivadas pertenecen a una sola categoría gramatical. El lexema es la "unidad léxica abstacta que no puede descomponerse en otras menores, aunque sí combinarse con otras para formar compuestos. Se compone de un conjunto de semas, carece de distribución sintáctica y, por tanto, de configuración morfológica concreta, si bien se halla asociada a diversas estructuras fonológicas alternativas; por ej. {FACIL} puede ser el lexema básico a partir del cual puede formarse un nombre, *facilidad*, un verbo *facilitar*, un adjetivo, *fácil*, un adverbio, *fácilmente*, con sus respectivas modificaciones morfemáticas adyacentes" (Cerda, 1986).

En cambio, el estudio realizado por Juilland y Chang-Rodríguez (1964), se aboca a tratar de encontrar el vocabulario básico del español, ellos usan un universo de 500,000 palabras, y realizan diversas estadísticas. El número de unidades léxicas contenidas en vocabulario básico son 5,024; ellos no mencionan el número de palabras que se derivan a partir de las unidades léxicas, pero haciendo un recuento de ellas, el total de palabras suman 22,889.

El tercer libro es el de Lara (1986), el cual es propiamente un diccionario, donde se definen cerca de 7,000 unidades léxicas. Se creen que estas unidades constituyen el vocabulario básico del español usado en México.

Aunque Juilland et al. y Lara hablan sobre un vocabulario básico, los enfoques son diferentes. En el segundo libro se trata de incluir todas

las voces usadas en las ciencias sociales, en las ciencias naturales, matemáticas, etc. que se presentan en los libros de primaria y secundaria, y en el primer libro no.

Siendo la idea de tener un vocabulario básico, se usó la lista de palabras proporcionada por Juilland et al. para facilidad en el reconocimiento del texto. Se piensa que una mejor opción será incluir el vocabulario presentado por Lara; y en un futuro más lejano el diccionario de Lara podría servir como base para agregar los significados a las palabras y así poder realizar un análisis semántico.

Además del trabajo involucrado en la digitalización, y en la depuración del trabajo hecho por la máquina, está también la necesidad de agregar la transcripción ya sea fonética o fonológica de las palabras. Para resolver este último problema se buscó algún tipo de libro o diccionario en el cual viniese en forma enumerada, o más o menos ordenada, la transcripción de cada palabra; pero lo más que se encontró fueron libros que tratan el tema de fonética, donde mencionan reglas y excepciones para transcribir en fonemas la palabra escrita, y cuando se tratan los temas de alófonos, fonemas, neutralización, sinéresis, sinalefa, etc. se dan ejemplos. Existen tres grandes dificultades agregadas al problema de la transcripción, que son la heterogeneidad en el habla de los individuos, la rapidez del habla y las divergencias de opiniones entre fonetistas en cuanto a la transcripción correcta de las palabras en donde ocurren los efectos de neutralización, sinéresis y sinalefa.

Por todos estos problemas, se decidió sólo abarcar la primera revisión; es decir, la revisión fonética o fonológica. Antes de tener la

posiblidad de usar el reconcedor de texto, se había pensado en dejar la base de datos aún más incompleta, con unas 3,000 palabras aproximadamente, sin anexar la categoría gramatical y entonces dedicar tiempo para implementar la revisión de concordancia; es decir, uno de los aspectos que estudia la Morfología. Sin embargo, esta idea fue pronto desechada al darme cuenta primeramente de la necesidad de incluir las categorías gramaticales, lo cual nos regresa a la idea de tener una base de datos completa, y de la complejidad del análisis, si se desea hacer un trabajo completo.

Antes de continuar, el término de concordancia puede definirse como:

> "La armonía o conformidad de accidentes gramaticales que deben guardar entre sí el adjetivo con el sustantivo y el verbo con el sujeto.
> El adjetivo concuerda con el sustantivo en género y número; v. gr: " libro bueno", "casa amplia". El verbo concuerda con el sujeto en número y persona; v. gr.: "yo medito", "nosotros trabajamos". (González, 1975, p. 238)

González menciona tres reglas cuando el verbo se refiere a varios sujetos o el adjetivo a varios sustantivos, y da trece excepciones a estas reglas. Sin embargo, esto es sólo el inicio, pues, el adjetivo puede jugar la función del sustantivo, los artículos se clasifican como adjetivos, los pronombres pueden funcionar como adjetivos, los verboides como sustantivos o adjetivos, los adverbios como verbo o adjetivos y las preposiciones, conjugaciones, interjecciones pueden funcionar como sustantivos, aparte de sus otras funciones (Revilla, 1988).

González nos da la siguiente definición de accidente gramatical: "A las variaciones que en su estructura sufren las palabras se les denomina *accidentes gramaticales.* Siete conocemos en castellano; a saber: género, número, declinación, aumento, disminución, grados de significación del adjetivo y conjugación" (1975, p. 28).

5.2 Definición de la entrada

¿Qué ventajas o desventajas ofrece el solicitar que la entrada a nuestro sistema sean alófonos, fonemas, o fonemas y archifonemas?. ¿Qué otros tipos de características se pueden extraer de la señal de voz, y qué beneficios obtendríamos al utilizar esta información?.

Tomando como referencia al libro de la Real Academia (1973), se cuenta con 52 alófonos, no existen señales afonemáticas positivas o negativas simples, pero hay alrededor de 170 reglas y 24 exepciones, donde se específican los contornos indicandose si se presentan en la misma sílaba, o pertenecen a sílabas contiguas. Alarcos (1965) da un ejemplo de señal positiva: "un indicador afonemático simple, en español, es la variante [ŷ] del fonema /y/, que sólo surge en principio de palabra: *yacer, hierro, yodo, yugo* (y cuando lo hace en interior de palabra, es como inicial de un monema: *ad-yacente, con-yugal, des-yemar*)" (p. 207). Si se considera que en la implementación no se generan las palabras mediante los subfijos, raices y prefijos, entonces este indicador no será útil; pero en caso contario, sí lo será y se tendría que hacer una investigación exhaustiva para encontrar otros indicadores.

En cambio, los fonemas son 23, aunque no hay señales positivas fonemáticas simples en el español existen indicadores positivos fonemáticos complejos, y negativos simples o complejos. Estos indicadores están dados en la Sección 3.8. Si se analizan los negativos complejos en caso de dos consonantes, se puede observar que éstos no contradicen a los mencionados positivos complejos, y quedan muchas otras combinaciones (23^2-115= 414 combinaciones) que tendrían que verificarse si se pueden dar dentro de palabra, entre palabra o no se presentan en español. Entre más consonantes estén agrupadas, menos combinaciones se dan dentro de palabra, las restantes o se presentan entre palabras o no son factibles en nuestra lengua. Pasa algo análogo con las vocales, pero existen los problemas de sinéresis y sinalefa, que conforme a Navarro (1957) no es posible establecer reglas al respecto y, por lo cual, no se podrían enumerar todas las posibles combinaciones válidas.

A mi parecer, el pedir que la fase previa identifique cada uno de los diferentes alófonos, implicaría una mayor esfuerzo en la etapa de procesamiento de señales e identificación; sin embargo, éste argumento podrá ser ratificado o anulado cuando se concluyan dichas fases.

La elección de fonemas en vez de alófonos creo que es conveniente, no tan sólo por lo anterior expuesto, sino porque las 194 reglas y excepciones existentes no nos dicen en forma determinante cuáles combinaciones de sonidos se encuentran dentro de una palabra o se da entre palabras, dado que se maneja el concepto de sílaba. Recordemos que, las reglas para la separación de palabras en sílabas se extiende a

los sonidos en contacto entre dos palabras, y de que una sílaba puede no siempre estar rodeada de pausas normales (pausas detectables).

En cambio, sí estimo conveniente el uso de archifonemas. Lo considero ventajoso en el sentido de no tener que asignar dos o más transcripciones para una palabra. El único inconveniente es que el uso de éstos es algo anticuado según Roca-Pons (1975), el cual menciona: "recordaremos que el concepto de fonema tuvo, al principio, una fuerte base psicológica, de la cual fue prescindiendo después. Igulamente, fue perdiendo importancia el concepto de archifonema" (pag. 344), o no es un concepto generalizado, Roca-Pons aclara que este término nació en la escuela de Praga y es o fue usado en general por los lingüístas europeos, lo cual podría restarle credibilidad a las transcripciones recopiladas y almacenadas en la base de datos. Otro aspecto a favor del uso de los fonemas es que por definición representan a los sonidos de acuerdo a su función en el lenguaje.

En cuanto a la segunda pregunta, ¿qué otros tipos de características se pueden extraer de la señal de voz, y qué beneficios obtendríamos al utilizar esta información?, la respuesta está en las Secciones 3.2.3, 3.3 y 3.7. En la sección 3.2.3, se especifica que son la intensidad y la entonación, las dos características físicas obtenibles de la señal de voz y de interés en el campo de la Lingüística; teniéndose como referencia que en español la cualidad de cantidad está en relación al acento, el acento es la conjunción de las tres características intensidad, tono, y cantidad; pero más extrictamente de cuatro cualidades: intensidad, tono, timbre y cantidad. En esa sección también nos dice que es posible diferenciar por la posición del acento, las

palabras como límite, limite, limité. En la sección 3.3, se establece lo que es la fonología de la frase y su valor distintivo o demarcativo; y por último, en la sección 3.7 se tratan los problemas que se presentan en el habla continua.

Pudiéndose concluir que al permitir que el usuario le dé entonación a las palabras, ocasiona que la función distintiva del acento en cuanto a fonología de palabra, quede anulada, no pudiéndose distinguir, por ejemplo, entre límite, limite, limité sí estas tres palabras se pronunciasen en forma interrogativa o si se usaran en un concepto inferior a su categoría gramatical. Pero la función distintiva y demarcativa de la frase sí es válida; es decir, que haciendo un análisis de la curva melódica se podría distinguir entre los diferentes tipos de frases, que en general se presentan entre pausas normales, y mediante el estudio de la intensidad se podría obtener información sobre la sintaxis de las frases pronunciadas.

En cuanto a si es importante o no incluir la duración de los sonidos, estoy de acuerdo con lo mencionado por Navarro, quién aclara que la cantidad absoluta varía mucho de persona a persona, pero que la cantidad relativa sí podría ser otra herramienta más de análisis.

Para concluir, ¿qué se espera recibir en la entrada?. Habiendo elegido el realizar únicamente la revisión de léxico y el uso de fonemas, entonces a la entrada se esperan fonemas y pausas; pero ¿qué tan continuas serán esas pausas?. Conforme a la Sección 3.2.1 que trata sobre el habla, se conoce que las pausas se presentarán entre grupos fónicos, y que, lo que caracteriza a los grupos fónicos son el significado y la necesidad de inhalar aire. Sin embargo, ¿qué tan

frecuente es esa necesidad de inhalar aire?; al respecto de ello Gili (1978) comenta:

> Las pausas son un elemento expresivo de gran importancia, tanto en el habla común como en la recitación y en la lectura. Hay pausas puramente respiratorias, causadas por la necesidad de reponer el volumen y la presión del aire necesario para la fonación. Pero éstas son poco frecuentes, fuera de los casos de cansancio o de especial agitación del ánimo. Lo normal es que no gastemos entre pausa y pausa todo el aire disponible en los pulmones, y que aprovechemos para inspirar las pausas largas determinadas por el sentido de lo que se dice. Por consiguiente, las interrupciones de la articulación a las que damos el nombre de pausas, obedecen sobre todo a motivos expresivos y se hallan en estrecha relación con las inflexiones melódicas de los grupos fónicos. (p.49)

Es decir, la aparición de pausas es, principalmente, función del significado de la oración.

5.3 Definición de la salida

La idea en un principio fue favorecer algún tipo de palabra, como la palabra más corta o la más larga, generando una sola oración, y sobre esa frase realizar las siguientes revisiones. Posteriormente, al detectar algún error se solicitaría la siguiente oración; pero ahora favoreciendo las palabras con uno o más sonidos, si se favoreció la más corta, o con uno o varios sonidos menos, si se favoreció la más larga. Debido al tipo de organización de la base de datos se vió que resulta más conveniente extraer todas las posibles palabras que se

pudieran identificar en ese grupo de sonidos antes de proporcionar las oraciones válidas, y no consultar y analizar sucesivamente.

El sistema deberá, entonces, ser capaz de brindar todas las oraciones válidas como salida.

5.4 Uso de las redes neurales

Usar la red como un medio para almacenar el vocabulario básico, o clasificar las palabras, o asociar la información, se juzga que no es conveniente, al menos con el tipo de arquitectura de procesador central, dado que se requeriría de una gran cantidad de neuronas; lo cual implicaría que el número de pesos que se tendrían que almacenar sería muy grande y probablemente el tamaño del archivo que almacena los pesos sería mayor que el área ocupada por un archivo con las palabras en si, y el tiempo empleado para obtener la respuesta de la red sería enorme.

Entonces, se plantea usar la red para realizar la validación de los indicadores fonemáticos, mencionados anterirormente en la Sección 5.2. Sólo que existe un problema, es necesario verificar que la red aprenda a hacer la clasificación en forma suficiente, tal que aunque no sea exacta se pueda distinguir las categorías, porque una mala clasificación generaría falsos errores, y eso no es deseable. Por ese mismo inconveniente, es preferible entrenar la red con las combinaciones reales existentes en el vocabulario y no con todas las encontradas en nuestro idioma, pues al ser más general puede ocasionar que no detecte combinaciones que se dan tan solo entre palabras de la base real, o acepte combinaciones que no están

presentes. La existencia, a la entrada, de una combinación de consonantes inexistente en nuestra base de datos, puede ser debida a dos motivos, por error en alguna fase anterior o por la pronunciación de alguna palabra desconocida a la computadora.

5.5 Método de acceso a la Base de Datos

Se optó por implementar esto en forma sencilla, siguiendo más bien una idea intuitiva más que algún método ya establecido. Se creyó conviente dedicar más esfuerzo en el análisis y búsqueda de soluciones al problema del reconocimiento de voz en el campo de la Lingüística, que en buscar el mejor método de acceso a una base de datos de registros de longitud variable, de gran magnitud y con posibilidad de crecer, no porque sea menos importante sino por ser una área mucho más estudiada. De hecho, el método de acceso y la organización en si de la base son aspectos muy importantes, pues éstos influencian directamente a la posibilidad, en un futuro, de hacer reconocimiento en tiempo real, y a la rapidez de respuesta de la implementación actual.

5.6 Resumen

En cualquier desarrollo teórico o experimiental es necesario poder explicar el objetivo del trabajo, las razones de haber usado tal o cual método, y establecer y justificar sus alcances. Este, por ser el primer capítulo de la parte de implementación trata de justificar o explicar los límites de la tesis.

De acuerdo a la teoría de la gramática generativa, para obtener la correcta escritura de la frase hablada, se requiere realizar ordenadamente tres tipos de revisiones, fonética o fonológica, gramática y semántica, de las cuales sólo se realizará la primera por la necesidad de la creación de una base de datos y por la complejidad de la siguiente revisión como para poder implementar las dos al mismo tiempo.

Se optó por realizar una revisión fonológica más que fonética, porque el manejo de alófonos no brinda para el problema de la agrupación de sonidos en palabras una ventaja sustancial en comparación con los fonemas, y sí la complica. Entonces, la información de entrada será la trascripción fonológica de las frases dichas por el usuario.

Por otra parte, parece prometedor el uso de archifonemas, y el estudio del acento no tan sólo con fines lingüísticos, sino como una herramienta de apoyo para realizar el reconocimiento de oraciones.

En relación a la salida, al principio se había pensado en proporcionar una posible realización, y en base a ella, quitar o agregar sonidos, pero al definir el método de acceso a la base de datos se contempló la conveniencia de extraer las palabras válidas, y dar como salida todas las oraciones.

Para la realización de este trabajo, se estableció como uno de los objetivos importantes el empleo de las redes neurales en alguna de las fases del sistema de revisión de léxico, las cuales se emplearán para clasificar los grupos consonánticos; es decir, la validación de los indicadores fonemáticos.

Se prestó poca atención a la búsqueda del mejor método para el acceso a la base de datos, con el propósito de investigar más a fondo los fenómenos que ocurren en el habla continua y las vías para realizar las fases restantes del reconocedor de voz.

Capítulo 6
Metodologías

En el capítulo pasado se expusieron las razones del alcance del sistema a implementar. Ahora, se presenta cómo se piensa llevar a cabo esto, y se hace una discusión previa al respecto antes de optar por algún método, con exepción a lo referente a la base de datos.

6.1 Revisión fonológica

Al definir el uso de fonemas y el manejo de un vocabulario básico, entonces no se pudo dar continuidad, por el momento, a alguno de los trabajos presentados por Grasser y Lee (1990a, 1990b), donde utilizan redes neurales para el aprendizaje de reglas fonológicas y morfológicas; aunque el primer estudio es para un lenguaje artificial, y en el segundo presenta un caso para agregar un sonido como sufijo o prefijo.

Creo qué implementar una gramática generativa es el proceso idóneo para resolver el problema de reconocimiento de voz, apoyándose en las palabras de uso más frecuentes, sin buscar que sea totalmente generativo.

Considerando que las reglas fonológicas nos indican a que fonema pertenece cada alófono dependiendo de sus rasgos característicos, se puede preveer que el trabajo realizado no está en desacuerdo con la gramática generativa. Las reglas fonológicas en un futuro podrían

servirnos como una interfase entre alófonos y fonemas, mediante el uso de rasgos característicos; es decir, que en la fase de reconocimiento de sonidos se detecten los rasgos y haciendo uso de reglas fonológicas se obtengan los fonemas, que es la entrada que se espera para realizar la revisión fonológica. Usando reglas morfológicas se podrían generar las terminaciones de las palabras en los casos necesarios, o conjugaciones de los verbos; e implementar las reglas asociadas con la sintaxis y la semántica.

El problema por resolver se ha limitado a crear la base de datos, implementar los métodos y procedimientos para el acceso a ésta, y tener la capacidad de poder generar todas las frases o enunciados posibles a partir de la transcripción fonológica de las frases pronunciadas por el usuario. Donde, en la base de datos se contará con la transcripción, la escritura y la categoría gramatical de las 22,889 palabras proporcionadas por Juillian et al. (1964). Antes de consultarla, se analizaran los grupos consonánticos mediante las redes neurales con el fin de insertar pausas o marcar error.

6.2 Métodos relacionados a la Base de Datos

6.2.1 Acceso

Si observamos cualquier diccionario o enciclopedia, por ejemplo ver Tabla 6.1.

Vemos que hay relativamente pocas palabras que empiecen con una combinación dada de cuatro sonidos, se puede apreciar que hay ocho palabras con *abad* y *abal*, aunque en abal se pueden esperar más al

considerar las formas conjugadas de los verbos. Pero, ahora veamos un ejemplo de palabras que inician con consonante, Tabla 6.2.

A	A, Ab, Abs	Ababa y Ababol
Ababillarse	Abacá	Abacería
Abacero, ra	Abacial	Abaco
Abacorar	Abad	Abada
Abadejo	Abadengo, ga	Abadernar
Abadesa	Abadía	Abadiato
Abajadero	Abajar	Abajeño, ña, Abajero, ra y Abajino, na
Abajo	Abalanzar	Abalaustrado, da
Abaleador, ra	Abalear	Abaleo
Abalizar	Abalorio	Abaluartar
Aballar	Aballestar	Abanar
Abancaino, na	Abancalar	Abancayno, na
Abanderado	Abanderamiento	Abanderar
Abanderizar	Abandonado, da	Abandonamiento
Abandonar		

Tabla 6.1. Palabras definidas en la página 1 de García-Pelayo (1972).

Hay 17 palabras definidas con *decl*, 13 con *deco* y 12 con *decr* ; pero, si se observaran los primeros cinco sonidos en vez de cuatro, se tienen a lo más diez palabras definidas, como en *decla*. La tendencia a necesitar más sonidos, para reducir el espacio de búsqueda en las palabras que inician con consonantes, se debe a la existencia de muchos términos de la forma *cvcc* [1], donde las últimas dos consonantes pueden ser {bl, br, cl, cr, dr, fl, fr, gl, gr, pl, pr, tl, tr} u otras combinaciones. Cuando inician con vocal, estos grupos

[1]La letra *c* representa un sonido consonántico en esa posición y la *v* a uno vocálico.

consonánticos también se presentan, pero se tiene la oportunidad de detectar otra vocal, *vccv*.

Decisión	Decisivo, va	Decisorio, ria
Declamación	Declamador	Declamar
Declamatorio, ria	Declarable	Declaración
Declaradamente	Declarante	Declarar
Declarativo, va o Declaratorio	Declinable	Declinación
Declinante	Declinar	Declinatorio
Declive	Declividad y Declivo	Decocción
Decoloración	Decolorante	Decolorar
Decomisar	Decomiso	Decoración
Decorado	Decorador, ra	Decorar
Decorativo, va	Decoro	Decoroso, sa
Decrecer	Decreciente	Decrecimiento
Decremento	Decrepitación	Decrepitar
Decrépito, ta	Decrepitud	Decrescendo
Decretal	Decretar	Decreto
Decúbito	Decuplar y Decuplicar	Décuplo, pla
Decuria	Decurión	Decurrente
Decurso	Dechado	

Tabla 6.2. Palabras definidas en la pág. 294 de García-Pelayo (1972)

La idea que surgiere esto, es utilizar unos cuantos sonidos para posicionarnos en la base de datos y apartir de allí buscar secuencialmente o en el caso de que sean muchos reposicionarnos de acuerdo al siguiente o siguientes sonidos. Pero, ¿cómo accesar directamente la base de datos por medio de un grupo de sonidos? y ¿hasta qué número de sonidos será posible especificar?.

De acuerdo a lo expuesto en el Capítulo 4, el único método que existe para acceso directo de información, es el que usa la función de dispersión. Una función de dispersión, *h(x)*, que cumple con el requisito de que exista una relación uno a uno entre el dominio y el rango de la función, es una versión modificada de la representación de la trascripción de la palabra en base 23. Donde 23 es el número total de fonemas existentes en nuestro idioma. Por ejemplo:

Si a=0, b=1, ch=2, d=3, e=4, f=5, g=6, i=7, k=8, l=9, m=10, n=11, ñ=12, o= 13, p=14, r=15, rr=16, s=17, t=18, u=19, w=20, x=21, y=22.

/o/ = $(13+1)*23^0-1$

/al/ = $(0+1)*23^1+(9+1)*23^0-1$

/ago/ = $(0+1)*23^2+(6+1)*23^1+(13+1)*23^0-1$

Si consideraramos cinco sonidos, entonces se tendrían 6,436,343(=23^5) posibles combinaciones, pero almacenar una tabla de semejante magnitud no es conveniente. Si fueran cuatro, tendríamos 279,841 combinaciones, éste si es un número razonable. Pero, ¿que les pasará a las palabras que tengan menos de cuatro sonidos?, absolutamente nada, la tabla puede contener sin ningún problema dichas combinaciones y en este caso la tabla de cubetas tendría 292,560 elementos.

Existen dos problemas en su implementación. Por definición la tabla contiene apuntadores a cada una de las cubetas, si los guardaramos usando su representación decimal, los registros de la tabla serían de longitud variable y entonces no se podría hacer un acceso directo a su contenido, a menos que, se cargara previamente en memoria y se leyera apuntador por apuntador. Usando una representación

hexadecimal, el acceso podría ser directo, pero en promedio el tamaño del archivo crecería, al guardar muchos ceros en lugar de un cero por combinación inválida. El número máximo de bytes[2] necesarios para accesar cualquier información en un archivo es ocho, que multiplicados por 292,560 da 2,340,480 bytes; lo cual tampoco es conveniente. Si en vez de guardar la dirección física donde se encuentra ubicada la cubeta, se guarda el número de registro de un archivo auxiliar donde estuvieran almacenadas las direcciones a las cubetas de las combinaciones de sonidos reales existentes, el tamaño de la tabla de cubetas se reduciría bastante. Según un muestreo realizado sobre los dos volúmenes del diccionario de la Real Academia (1984), se encontraron aproximadamente diez mil, lo que implica que con dos dígitos en base 256 son más que suficientes para representar el número de registro en el archivo auxiliar, dado que el número máximo que se puede representar con los dos dígitos es de 65,535. Esta configuración se puede visualizar en la Figura 6.1.

Como la búsqueda para el caso de palabras de cuatro sonidos o más, es en toda la cubeta, cada cubeta estará formada de un sólo bloque. También es conveniente almacenar en el archivo auxiliar el número de bytes que están almacenados en cada cubeta con el fin de cargar la cubeta en memoria, y que la búsqueda de patrones dentro de ésta se realice con mayor rapidez.

---

[2]Un byte es equivalente a ocho bits. Un bit es la representación de un dígito binario.

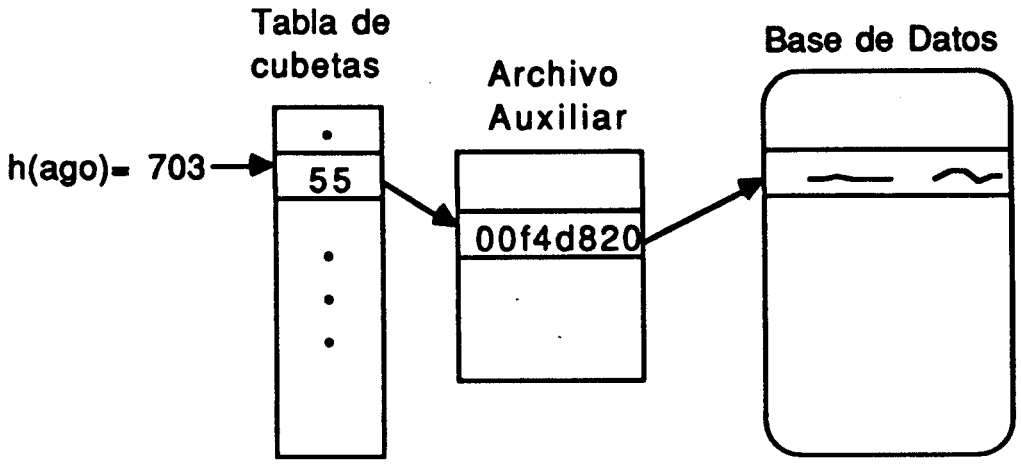


Figura 6.1. Acceso a la Base de Datos mediante la tabla de cubetas

En el caso de que existan muchas palabras, se pensó en usar un vector o una arreglo cuadrado para almacenar los nuevos apuntadores. Como táctica para ahorro de espacio, se usó de nuevo un archivo auxiliar con los apuntadores a las combinaciones reales y almacenándose en el primero el número de registro a accesar. Cada elemento del vector o arreglo ocupa tan sólo un dígito en base 256; es decir, un caracter. Por ejemplo, si se decidiera especificarse dos sonidos más, entonces se tendría almacenada una arreglo de 23 por 23 y que ocuparía un espacio físico de 529 bytes. En la Figura 6.2 se ilustra el acceso a la base para la palabra *estaba*, haciendo uso de los archivos auxiliares a la cubeta *esta*, y que son *esta.arr* y *esta.arrdir*.

El número de accesos máximos de un índice disperso con búsqueda binaria sería igual a 15, si consideramos que cada bloque contiene tan solo un registro y que nuestra base contiene 18,421 palabras diferentes, $\log_2(18{,}421+1)= 14.17$. En nuestro caso, para llegar a la cubeta hay que realizar dos accesos, de ahí tenemos oportunidad de

preguntar a lo más 13 palabras si quisieramos igualar el número accesos de ambos métodos. En el caso de que se haya formado los archivos auxiliares, para accesar las cubetas más llenas, se requiere de cuatro accesos. Después de haber obtenido la dirección definitiva se realiza un búsqueda lineal. Una ventaja que ofrece este método es que la tabla de cubetas y el archivo *.arr* casi no crecen, los archivos auxiliares crecen cuando se habilita una nueva cubeta o algún elemento del arreglo, y la cubeta incrementa su tamaño cada vez que una nueva información es agregada.

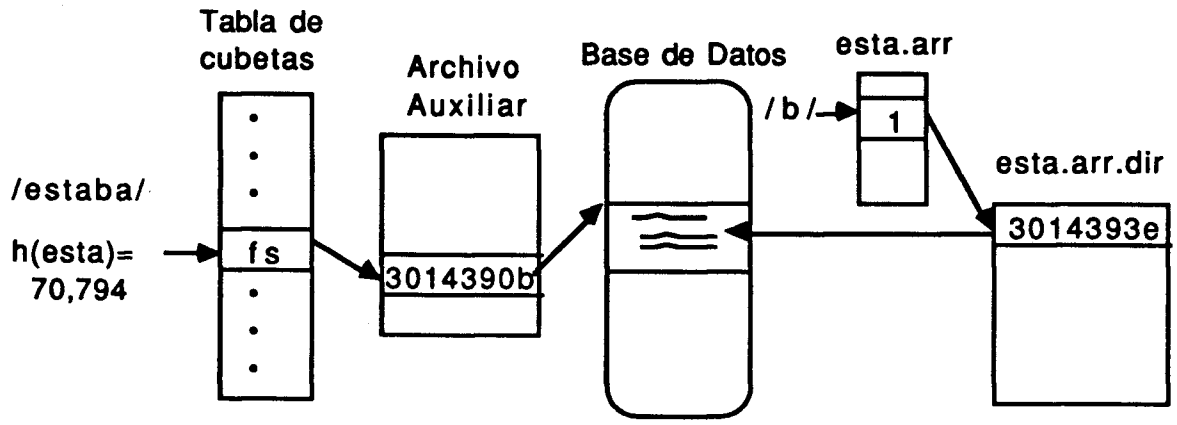


Figura 6.2. Ejemplo de acceso a la Base de Datos mediante el uso de los archivos auxiliares a la cubeta

6.2.2 Estructura de la información

Cada registro de la base está compuesto de la transcripción, la escritura, una ordenada donde se especifíca su clasificación gramatical y la unidad léxica de la cual proviene, y un comentario donde se indica si la transcripción fue revisada ya, o no.

En la transcripición puede estar marcado el acento, las letras usadas para escribir las transcripciones son las mismas que las presentadas en la Subsección 3.5.2, y como se trata de una transcripción fonológica se delimita por diagonales. Si una palabra tiene más de una transcripción es necesario ponerlo como registro aparte, debido a que, como la base de datos está direccionada por los primeros sonidos podrían existir problemas si se permitiera colocar todas sus transcripciones en el mismo registro.

Contemplando la posibilidad del aumento de vocabulario, se pensó que quizás fuera conveniente agrupar las palabras derivadas de un mismo vocablo; por ejemplo,

blanco aj. blanca, blancas, blanco, blancos

como, blanc(a,as,o,os); y de esta manera tener un ahorro significativo de espacio. Además, se consideró también el aumento del número de campos, usando un indicador de principio y continuación de registro, dando margen que los registros ocupen cuantos renglones necesiten. Para hacer esto, se usaron caracteres que normalmente nadien los utilizaría para escribir la información de la base de datos, como los primeros caracteres ASCII.

Dado que la búsqueda sobre la base es lineal y como los registros son de longitud variable, entonces, se agregaron dos campos más que nos indican el número de bytes que hay que avanzar para accesar el siguiente registro y el número de bytes que hay que retroceder para accesar el registro anterior. No queriendo limitar a la base de datos en la longitud de sus registros se utilizan los ocho digitos

hexadecimales, que como se comenta en la subsección anterior con éstos se puede accesar la información de cualquier archivo.

También, para identificar si se crearon los dos archivos auxiliares a la cubeta, es necesario usar otro caracter identificador, y es conveniente almacenar la dimensión del arreglo. Para almacenar éste se usa nuevamente el equivalente en código ASCII.

Juilland et al. (1964) clasifican las palabras de acuerdo a alguna de las siguientes diez categorías gramaticales: adjetivo, adverbio, artículo, conjunción, interjección, nombre, numeral, preposición, pronombre y verbo. Las abreviaturas de éstas se dan en la Tabla 6.3. En las conjugaciones verbales, aparecen los símbolos de I, para especificar el imperativo, de P, para el pretérito, un 3, para indicar tercera persona en los casos donde pueda existir confusión, y un - antes o después del verbo para indicar el uso de los pronombres personales[3], como en *me dijeron*, y *déjaselo*. Estos símbolos se respetan a la hora de formar la base de datos. Por ejemplo, el guión después del verbo servirá en un futuro, para solicitar la generación de sus posibles conjugaciones verbales usando los pronombres personales.

Note que las conjugaciones donde se indica el uso de un pronombre personal antes del verbo constituyen dos palabras, por lo cual éstas son eliminadas al formar la base de datos. Sin embargo, para no perder esta información se agregaron dichas conjugaciones, si no aparecían nombradas sin guión previamente, al fin de cada unidad léxica

---

[3]Por ejemplo: me, te, se, lo, les, nos, os, etc.

omitiendo el guión, y escribiendo un comentario que indica que esa combinación fue agregada.

Adjetivo	aj.	Adverbio	av.
Artículo	a.	Conjunción	c.
Interjección	i.	Nombre	n.
Numeral	n.	Pronombre	pn.
Preposición	p.	Verbo	v.

Tabla 6.3. Abreviaturas usadas por Juilland et al. (1964) para las categorías gramaticales

En resumen, un registro se ve como se muestra en la Figura 6.3.

c1	c2		d1		d2	<-|
/blank(a,as,o,os)/	->	blanc(a,as,o,os)		{aj,blanco}	->	TNR	<-|

donde:

c1= Inicio o continuación de reg., o matriz.
c2= Dimensión de la matriz.
d1= Desplazamiento al registro anterior.
d2= Desplazamiento al próximo registro.
= Un espacio en blanco.
<-| = Caracter de fin de línea.
-> = Tabulador.
TNR = Transcripción no revisada.

Figura 6.3. Organización de un registro en la Base de Datos

6.3 Métodos sobre la red neural

Como se especificó en la Sección 5.4, es necesario que la red aprenda en forma suficiente a clasificar los grupos consonánticos. Con

esta idea se tratará de entrenar una red con todas las combinaciones existentes para el caso de dos consonantes, por ser un número no tan elevado de patrones (324 $=18^2$). Las existentes para tres y cuatro sonidos consonánticos son muchísmas y esto requeriría una enorme cantidad de unidades para entrenarla, su velocidad de respuesta sería lenta, y el tiempo invertido en el entrenamiento muy grande.

Entonces, se plantea la opción de contar con dos redes, una propiamente entrenada con todas las combinaciones de dos consonantes y la otra construida para validar las combinaciones de tres y cuatro sonidos consonánticos. Para ambas redes se usó como entrada neta la suma ponderada de las señales recibidas más un factor de polarización, *bias*, el valor de activación sigue el comporamiento de una función sigmoidal con respecto a la entrada neta, y la función de salida es la identidad; en otras palabras,

$$net_i = \sum_j w_{ij} * o_j + bias_i,$$

$$a_i = \frac{1}{1+e^{-net_i}},$$

$$o_i = a_i.$$

Para el entrenamiento se usó el software proporcionado por McClelland et al. (1988), y la regla de aprendizaje utilizada fue la delta generalizada. Las unidades de entrada se utilizaron en forma binaria, es decir, presentando a su entrada un valor de uno o de cero, y cada una representa los fonemas en cualquiera de las dos posiciones; por lo tanto, se cuenta con 36 unidades de entrada (ver Figura 6.4).

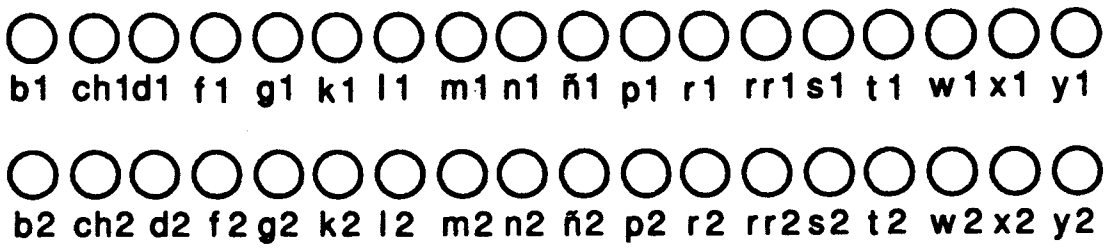


Figura 6.4. Unidades de entrada de la red entrenada

Para esa red, existen tres unidades de salida, una para indicar si esa combinación de dos sonidos se da dentro de palabra, otra cuando se da entre palabras, y otra cuando no se encuentre en el vocabulario; es decir, la red la vamos a usar como un clasificador. Ver Figura 6.5.

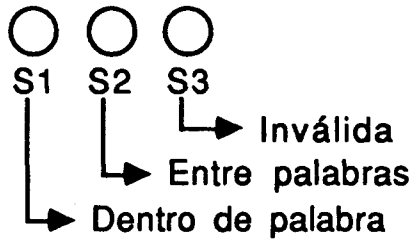


Figura 6.5. Unidades de salida de la red entrenada

Para determinar que combinaciones caen dentro de cierta clasificación, se extrajo de la base de datos los grupos de sonidos consonánticos que se presentan al principio, en mediación y al final de palabra. En las Tablas 6.3, 6.4 y 6.5 se presenta dicha información.

Los grupos que la red clasificará como dentro de palabra, encendiendo la primera unidad de salida, son precisamente los mostrados en la Tabla 6.5. Las combinaciones válidas, de dos

consonantes, que se dan entre palabras, pero que no se encuentren dentro de palabra, se forman al unir una consonante que se presente a final con otra que aparezca al inicio (ver Tabla 6.6). En cuanto a las inválidas, serán todas las que no se encuentren ni dentro ni entre palabras (ver Tabla 6.7).

b	bl	br	ch	d	dr
f	fl	fr	g	gl	gr
k	kl	kr	l	m	n
p	pl	pr	rr	s	t
tr	x	y			

Tabla 6.4. Sonidos consonánticos que se presentan al inicio de palabra

b	bs	d	l	n	nt	r	rd	s	x

Tabla 6.5. Sonidos consonánticos que se presentan al final de palabra

bl	bn	br	bs	bt	bx	db	dk
dm	dr	fl	fr	gl	gm	gn	gr
kd	kl	kn	kr	ks	kt	lb	ld
lf	lg	lk	lm	lp	lrr	ls	lt
mb	mn	mp	nb	nch	nd	nf	ng
nk	nl	nm	nn	nrr	ns	nt	nx
pl	pr	ps	pt	rb	rch	rd	rf
rg	rk	rl	rm	rn	rp	rs	rt
rx	sb	sd	sf	sg	sk	sl	sm
sn	sp	ss	st	tl	tm	tr	ts

Tabla 6.6. Grupos de dos consonantes que se presentan a mediación de palabra

La segunda red requiere 72 (=18*4) unidades de entrada, se utilizará la misma disposición que la otra red. Aparte, requieren dos salidas

más para indicar si la separación de palabra se encuentra entre la primera y la segunda o la segunda y la tercera (Figura 6.6).

bb	bch	bd	bf	bg	bk	bm	bp
brr	by	dch	dd	df	dg	dl	dn
dp	drr	ds	dt	dx	dy	lch	ll
ln	lx	ly	np	ny	rrr	ry	sch
srr	sx	sy	xb	xch	xd	xf	xg
xk	xl	xm	xn	xp	xrr	xs	xt
xx	xy						

Tabla 6.7. Grupos de dos consonantes que sólo se presentan entre palabras

bñ	bw	chb	chch	chd	chf	chg	chk	ckl
chm	chn	chñ	chp	chr	chrr	chs	cht	chw
chx	chy	dñ	dw	fb	fch	fd	ff	fg
fk	fm	fn	fñ	fp	frr	fs	ft	fw
fx	fy	gb	gch	gd	gf	gg	gk	gñ
gp	grr	gs	gt	gw	gx	gy	kb	kch
kf	kg	kk	km	kñ	kp	krr	kw	kx
ky	lñ	lr	lw	mch	md	mf	mg	mk
ml	mm	mñ	mr	mrr	ms	mt	mw	mx
my	nñ	nr	nw	ñb	ñch	ñd	ñf	ñg
ñk	ñl	ñm	ñn	ññ	ñp	ñr	ñrr	ñs
ñt	ñw	ñx	ñy	pb	pch	pd	pf	pg
pk	pm	pn	pñ	pp	prr	pw	px	py
rñ	rr	rw	rrb	rrch	rrd	rrf	rrg	rrk
rrl	rrm	rrn	rrñ	rrp	rrr	rrrr	rrs	rrt
rrw	rrx	rry	sñ	sr	sw	tb	tch	td
tf	tg	tk	tn	tñ	tp	trr	tt	tw
tx	ty	wb	wch	wd	wf	wg	wk	wl
wm	wn	wñ	wp	wr	wrr	ws	wt	ww
wx	wy	xñ	xr	xw	yb	ych	yd	yf
yg	yk	yl	ym	yn	yñ	yp	yr	yrr
ys	yt	yw	yx	yy				

Tabla 6.8. Grupos inválidos de dos consonantes

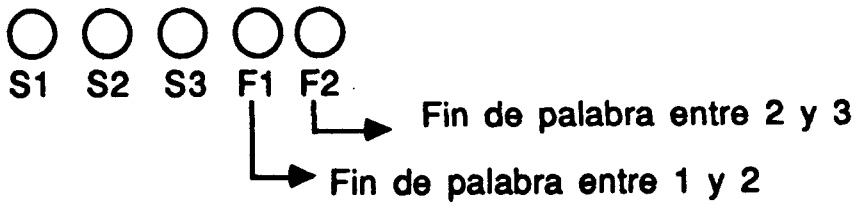


Figura 6.6. Unidades de salida de la red no entrenada

Observe que en el caso de dos consonates, si la red indica separación de palabras, tan sólo hay una opción y, por lo tanto, no se necesita indicar.

Las grupos consonánticos de tres y cuatro sonidos que se dan dentro de palabra se muestran en la Tabla 6.8; y siguiendo un proceso análogo al descrito para el caso de dos consonantes, se presentan en las Tablas 6.9 y 6.10 las combinaciones que se dan exclusivamente entre palabras. Las inválidas no se enumeran pero son todas las restantes. En las Tablas 6.10 y 6.11 se usa el símbolo ' , como demarcador de fin de palabra.

bsk	bst	bstr	ksk	kskl	ksp	kspl	kspr
kss	kst	kstr	ktr	ldr	lkr	ltr	mbl
mbr	mpl	mpr	ndr	nfl	nfr	ngl	ngr
nkl	nkr	nsf	nsk	nsp	nss	nst	nstr
ntr	rfl	rpl	rpr	rsp	rst	sfr	sgr
skl	skr	spl	spr	str			

Tabla 6.9. Grupos de tres y cuatro consonantes que se presenta dentro de palabra

Observando las Tablas 6.9, 6.10 y 6.11, se puede ver que los grupos {bl, br, bs, dr, fl, fr, gl, gr, kl, kr, ks, ns, nt, pl, pr y tr} se usan

frecuentemente; entonces la idea es juntar los sonidos anteriores con el sonido correspondiente y asignarle una unidad oculta cuando habiliten a las mismas unidades de salida. Debido a la validación de las combinaciones de cuatro sonidos existirán tres niveles de capas ocultas. Esto se ilustra en la Figura 6.7.

bs'bl	bs'br	bs'dr	bs'fl	bs'fr	bs'gl	bs'gr	bs'kl
bs'kr	bs'pl	bs'pr	nt'bl	nt'br	nt'dr	nt'fl	nt'fr
nt'gl	nt'gr	nt'kl	nt'kr	nt'pl	nt'pr	nt'tr	

Tabla 6.10. Grupos de cuatro consonantes que sólo se presentan entre palabra

b'bl	b'br	b'dr	b'fl	b'fr	b'gl	b'gr	b'kl
b'kr	b'pl	b'pr	b'tr	bs'b	bs'ch	bs'd	bs'f
bs'g	bs'l	bs'm	bs'n	bs'p	bs'rr	bs's	bs'x
bs'y	d'bl	d'br	d'dr	d'fl	d'fr	d'gl	d'gr
d'kl	d'kr	d'pl	d'pr	d'tr	l'bl	l'br	l'fl
l'fr	l'gl	l'gr	l'kl	l'pl	l'pr	n'bl	n'br
n'pl	n'pr	nt'b	nt'ch	nt'd	nt'f	nt'g	nt'k
nt'l	nt'm	nt'n	nt'p	nt'rr	nt's	nt't	nt'x
nt'y	r'bl	r'br	r'dr	r'fr	r'gl	r'gr	r'kl
r'kr	r'tr	rd'b	rd'ch	rd'd	rd'f	rd'g	rd'k
rd'l	rd'm	rd'n	rd'p	rd'rr	rd's	rd't	rd'x
rd'y	s'bl	s'br	s'dr	s'fl	s'gl	x'bl	x'br
x'dr	x'fl	x'fr	x'gl	x'gr	x'kl	x'kr	x'pl
x'pr	x'tr						

Tabla 6.11. Grupos de tres consonantes que sólo se presentan entre palabra

El nivel de polarización depende de lo que se desee hacer; por ejemplo, en el caso de la unidad que interactúa con las entradas k1, l1, m1, n1 y s1, se busca que ésta se habilite cuando aparezca alguna

entrada y que la salida sea casi cero o casi uno, su bias correspondiente puede ser de menos cinco y los pesos , de cualquiera, igual a diez, dado que con una entrada neta de cinco el nivel de activación es casi uno.

$$a = \frac{1}{1+e^{-(10-5)}} = 0.99$$

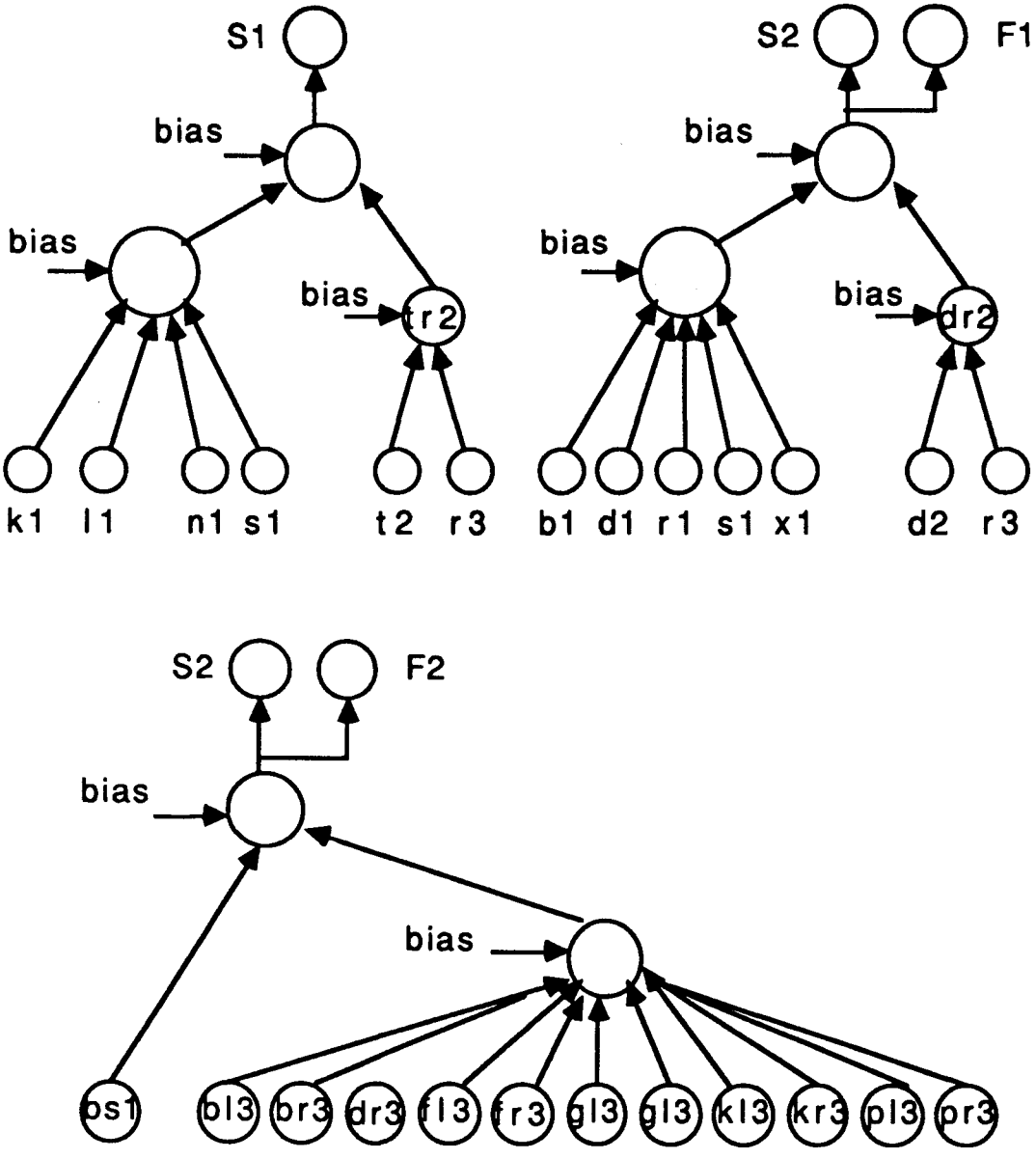


Figura 6.7. Algunos ejemplos del diseño de la red no entrenada

La salida de esta unidad se conecta a otra, que recibe también a t2 y r3. El bias de esa unidad debe ser igual a -25 y los pesos de diez, para permitir que sólo se habilite cuando las tres señales estén presentes, porque

$$a = \frac{1}{1+e^{-(10+10+10-25)}} = 0.99$$

La información de las dos redes debe quedar en el formato especificado por McClelland et al. (1988, pág. 264) para el archivo *.net*. Se agregó la opción de poder usar directamente números en la sección de "network" colocando un *&* en vez de un %, en lugar de tener que manejar un archivo aparte con los pesos de las conexiones.

Conforme a McClelland et al., en el archivo pueden existir cuatro secciones que son *definitions:*, *constraints:*, *network:* y *biases:*. Cada una de ellas debe ser terminada con *end*. En la sección de *definitions:*, sólo se reconocerán los estatutos para definir la red; es decir, *nunits*, *ninputs* y *noutputs*. La sección de *constraints:* es utilizada para definir valores constantes sean enteros o reales, usando etiquetas de un solo caracter; estas etiquetas pueden ser usados en cualquiera de las dos próximas secciones, ya sea para especificar pesos o polarizaciones. La sección de *network:* sirve para definir los pesos de las conexiones entre las unidades; se pueden usar los nombres de las constantes y deben ir escritas sin espacio, o pueden definirse por bloques usando

1. El símbolo de %.
2. El nombre de alguna constante (opcional).
3. Un espacio en blanco.

4. El índice de la primera unidad receptora, espacio, el número de unidades receptoras, espacio, el índice a la primera unidad transmisora, espacio, y el número de unidades transmisoras.
5. Si no se uso la constante, entonces en las siguientes líneas se espera leer cada uno de los caracteres. El caracter punto está definido como cero.

Es posible definir varios bloques. Cuando se quiera especificar los valores, sin la necesidad de usar las constantes, en vez de escribir % usar &. La sección de *biases:* es idéntica a la de pesos con la excepción de que sólo es necesario especificar el índice a la primera unidad y el número de unidades. El índice mínimo para accesar las unidades no es uno, sino cero. Dos ejemplos de esto se muestran en la Figura 6.8.

```
definitions:  definitions:
nunits 10     nunits 10
ninputs 5     ninputs 5
noutputs 2    noutputs 2
end           end
constraints:  constraints:
p 5           p 5
n -5          n -5
network:      network:
.....         % 5 3 0 5
ppppp         .nn.p
nnnnn         pp.pn
npnpn         ppppn
ppnnn         %n 8 2 5 3
end           end
biases:       biases:
%n 5 7        ppn..pn
end           end
```

Figura 6.8. Dos ejemplos del archivo *.net*

6.4 Resumen

De manera introductoria al Capítulo 7 que habla sobre la implementación en sí, se trata de explicar en éste la forma o metodología en que se planea hacer el análisis fonológico, el acceso y la estructura de la base de datos, y el uso que se destinará a las redes neurales.

La revisión de léxico, por el momento, no será del tipo generativo, sino que se hará mediante el apoyo de una base de datos. La base de datos será organizada en cubetas, y accesada mediante una función de dispersión, que es una versión modificada de la representación de la trascripción de la palabra en base 23. Además de la tabla de cubetas, existe un archivo auxiliar para la tabla y más archivos auxiliares cuando el número de registros en la cubeta es relativamente grande. Llegando a la dirección definitiva la búsqueda es lineal.

Cada registro cuenta, por el momento, con cinco campos, aparte de cuatro extras al inicio de éste, uno para la transcripción, otro para la escritura, el que le sigue para la categoría, el penúltimo para indicar de que palabra se deriva, y el último es un comentario, donde se especifica si la transcripción ya fue revisada. Las palabras que tienen la misma categoría y que sólo se diferencian en los últimos sonidos son agrupados en el mismo registro. De los campos agregados al principio del registro, uno se usa para indicar si es inicio de registro, o continuación de él, o la existencia de dos archivos extras para lograr un mejor direccionamiento; el siguiente vale diferente de nulo si existen esos dos archivos e indica la dimensión de uno de ellos, los

otros dos contienen los despalzamientos relativos al registro anterior y al posterior.

Se presentan los grupos consonánticos que las redes deben de clasificar, y la topología de las redes. Se optó por entrenar, primeramente, la red que clasificará los grupos de dos consonantes. Para ambas redes la entrada neta, será la suma ponderada de las entradas más una nivel de polarización, su función de activación es de forma sigmoidal y no depende del estado anterior, la salida es igual al valor de activación, y la regla de aprendizaje que se usará para entrenar es la regla delta generalizada. También se planteó la estructura de la red no entrenada.

Capítulo 7
Sistema de revisión de léxico

En la fase de planeación se idearon algunos procedimientos los cuales al momento de quererlos implementar se vio que eran inconvenientes; pero, en vez de excluirlos totalmente de la tesis se presentan en la primera sección, antes de ultimar detalles sobre la versión final de las rutinas.

7.1 Procedimientos e ideas descartadas

Antes de obtener el apoyo del departamento de humanidades, también se consideraba revisar los grupos vocálicos presentes al principio, en medio y final de palabra, además de los consonánticos, pero dado a la existencia de los problemas de sinalefa y sinéresis se descartó esa idea. Después, estuve analizando si los efectos producidos por la tendencia a la reducción de vocales eran regulares o no; en un principio creí que lo eran, pues se mencionaban reglas como las presentadas por Martínez (1975), y las llegué a tomar en cuenta para la planeación del sistema.

Cuando fue posible conseguir la asesoría de la Lic. María Guadalupe Torres me comentó el hecho de su desacuerdo en querer formular reglas para representar los fenómenos de sinéresis y sinalefa; y leyendo más al respecto, se decidió agregar las diferentes transcripciones en la base de datos. Dado que ésta es accesada por los primeros cuatro sonidos, si las transcripciones alternativas de la

palabra presentaban variaciones en los primeros cuatro sonidos, entonces se planeo ponerlo como un registro aparte y en caso contario ponerlos en el mismo. Al final se optó mejor por colocarlas en un registro aparte, con el propósito de simplificar el proceso de modificación de la base.

En ese tiempo, se pensaba usar el acento como demarcador del inicio de la próxima palabra, y a causa de ello se cuestionaba sobre la identidad del acento, y la facilidad de extraerlo de la señal real. Después de un largo análisis se concluyó que el acento, en conversación continua, no nos sirve como un demarcador de la separación entre palabras sino más bien de las frases u oraciones, y como éstas generalmente van emarcadas entre pausas reales, entonces no tendría sentido.

Antes de que se tuviera la oportunidad de usar el reconocedor de texto, se había pensado en implementar la rutina que accesara los modelos de las conjugaciones verbales[1], y se habían almacenado varios modelos, pero después se le dio más importancia a otras fases del sistema, y posteriormente se invertió más tiempo en depurar la base de datos, y la idea quedó relegada.

Después de establecer el alcance del algoritmo de revisión de léxico, se manejó sólo la opción de generar todas las posibilidades e imprimirlas en un archivo, y se planeaba que las fases posteriores iban a trabajar sobre éste; pero, esta idea va en contra de una implementación en tiempo real, porque de hacerlo así demoraría la

---

[1]La información de los modelos se extrajo de García-Pelayo y Durand (1982).

respuesta del sistema. Antes de eliminar esta idea, y de usarla sólo como una demostración de la potencialidad del mismo, se creyó que un formato adecuado de la información de salida era el manjar llaves y comas para identificar cuando se tenían varios posibles caminos por analizar. Como en el caso de la Figura 7.1, la frase /elkampesinosabe/ quedaría almacenada como: {e} el {kampesino {o} {os sabe, sabe}, kampesinos sabe} {e}. Las letras enmarcadas entre llaves, por ejemplo la {a}, indican que son opcionales; pero se deben de tomar muy en cuenta debido a la existencia de la reducción de sonidos vocálicos repetidos.

El campesino sabe -> /elkampesinosabe/

$$\{e\}\ el \left\{ \begin{array}{l} kampesino\ \{o\} \left\{ \begin{array}{l} os\ sabe \\ sabe \end{array} \right\} \\ kampesinos\ \{sabe\} \end{array} \right\} \{e\}$$

Figura 7.1. Obtención de todas las posibles oraciones de la frase /elkampesinosabe/

Para oraciones más complicadas, la rutina que tratara de obtener la información de este archivo, sería muy compleja, pues, tendría que llevar el control de las llaves que se abren y se cierran, y de la localización de las comas.

Después, se propuso una forma de optimizar el número de consultas, pero se vio que en los casos críticos lo único que haría éste es dificultar las cosas. El principio consistía en comparar sólo las rutas

más largas de cada camino, e ir consultando la base de datos para llevar a cabo esto, y detener el proceso cuando llegaran al mismo punto; sólo que esto falla por la existencia de rutas falsas, a partir de la cual la frase no puede continuar. Las rutas falsas se presentan por dos motivos, por una mala identificación de los sonidos o por la inexistencia de dicho vocablo en la base. La idea de optimizar surgió por la existencia de palabras relativamente largas y si éstas son el único camino existente para continuar la frase, entonces no es necesario buscar las palabras contenidas dentro de éstas, como en la palabra *campesino* donde se puede reconocer las palabras a, pes, si, sin, sino, i, y no.

La estructura y las rutinas de la base de datos, también, fueron replanteadas varias veces al tratar de discernir que información era la necesaria y suficiente, dónde y cómo codificarla.

En relación a la red, también, siguió un proceso evolutivo. En el principio, cuando todavía no me familiarizaba con los efectos que se producen en el habla continua, creí que hacer una agrupación previa de los sonidos en sílabas sería útil como apoyo a la información proporcionada por el acento y así poder determinar algunos fines de palabras.

De hecho, se entrenaron varias redes con diferentes configuraciones, y se obtuvo al final una buena respuesta. Para ello sólo se analizaban grupos de sonidos de a lo más seis, debido a que era el patrón más largo de consonantes enmarcadas entre vocales, vccccv, existente en el vocabulario. La red que tuvo un mejor desempeño tenía 180 unidades, de las cuales 132 eran de entrada, 5 de salida, 28 en la

primera capa oculta y 15 en la segunda. Primero se pensó en codificar la entrada, pero no se logró una buena respuesta de la red dado que no podía hacer una correcta generalización; y se decidió por no codificarla.

7.2 Versión final

Todos los programas fueron escritos en C, y desarrollados dentro del ambiente MPW, en una Macintosh II, con excepción del programa para generar los patrones maestros de la red a entrenar, que se desarrollo en la VAX porque el paquete de entrenamiento de la red neuronal se ejecutó en dicha máquina. Además de las rutinas creadas para la revisión de léxico, también se programaron varias para la creación de la base de datos, y para la obtención de los grupos consonánticos, presentados en la Sección 6.3. Debido a la magnitud de algunos procedimientos se prefiró separar estos en programas individuales, tal como la rutina de la red que está en *red.c*, la rutina de consulta que está en *consulta.c*, y la rutina de reporta está en *reporta.c*. La rutina del programa principal está en *Revlex.c*.

A continuación, se explicarán los procedimientos más importantes, auxiliándonos con diagramas de flujo, figuras y tablas. En los diagramas se siguen las siguientes convenciones: los procedimientos están enmarcados por un rectángulo de borde grueso, en caso contario se usa borde sencillo. Si el procedimiento fue implementado como una subrutina, entonces aparecerá sobre o dentro de éste otro rectángulo más pequeño con el nombre de la subrutina, y si requiere de parámetros se indica encerrándolos entre paréntesis.

7.2.1 Programa Principal (Revlex)

Después de la fase de inicialización de variables, se lee el archivo llamado *Revlex.ent* y se hace un cambio de nomenclatura, para facilidad de las demás rutinas, reemplazando /ch/ por /c/ y de /rr/ por /-/, luego se revisa los sonidos para detectar las combinaciones inválidas de consonates, o las que se dan entre palabras, dentro de cada frase enmarcada por pausas reales. Esto se realiza mediante las dos redes neuronales; si la red detecta fin de palabra dentro de alguna frase, entonces, insertará pausa; pero, si detecta error se insertará un caracter especial al inicio de la frase, y se escribirá en *Revlex.err* la posición donde inicia dicha frase. La frase resultante es almacenada en *Revlex.rev*.

Posteriormente, se procesa cada frase almacenándola en memoria y se realiza una búsqueda de izquierda a derecha, hasta localizar el primer posible inicio de palabra, dentro de los sonidos que constituyen la frase, para encontrar las palabras existentes a partir de cada sonido, y las palabras encontradas se almacenan también en memoria. En general, no son todas las posibles, sino que se registra la primera acepción, pudiendo haber otras con la misma transcripción pero con diferente escritura o categoría. Cuando se hace el almacenamiento en memoria, se pide además memoria para el resto de las variables que así lo requieran.

La idea de verificar por un posible inicio de palabra servirá para descartar más opciones y agilizar la búsqueda de izquierda a derecha. Para implementar esta validación se carga, tan solo una vez, todos los

grupos consonánticos que se presentan al inicio de palabra, reportados en la Tabla 6.4, y almacenados en *cons.ini*. Si a la hora de hacer la búsqueda se encuentra una vocal entonces se procede a la fase de consulta. Pero, si se presenta un grupo consonántico entonces se pregunta si este se presenta a principio de palabra, si es afirmativo continua, y en caso contrario se avanza un sonido y se repite el ciclo hasta encontrar un posible inicio de palabra.

Después de analizada la frase por el sistema, el usuario tiene la libertad de pedirle la información hallada, y para ello existen cinco opciones aparte de las opciones de continuar con la siguiente frase o de salir. Las frases tentativas generadas a partir de la frase real, pero ahora ya con las respectivas pausas entre palabras, se pueden consultar una por una, o pedir un reporte completo de estas, donde no se anota la categoría correspondiente de las palabras, o pedir cambio de categoría o la categoría anterior, o la elección de otra frase. El proceso termina cuando se hayan análizado todas las frases dichas por el usuario.

Las opciones con las que se cuenta son: siguiente oración, cambio de frase, traer la siguiente o la anterior categoría, obtener todas, continuar con la siguiente frase ha analizar y salirse del programa. Mediante la opción de cambio de frase se acelera un poco el proceso de búsqueda.

Antes de revisar la entrada para detectar combinaciones inválidas o fines de palabras, y hacer el cambio de notación, no se abre la base de datos, ni los archivos auxiliares a ésta, para que no existan problemas de memoria, y después de realizado esto, es necesario abrir los

archivos. La base de datos se llama precisamente así *VocBas*, el archivo que contiene la tabla de cubetas se llama *Tab.cub* y su archivo auxiliar *Aux.tab*.

Al elegir la opción de salir, se cierran la base de datos, la tabla de cubetas, y su archivo auxiliar; además, se libera la memoria y se retorna el número de frases con error.

El diagrama de bloques del procedimiento se encuentra la Figura 7.2. La lista de variables globales y declaración de tipos se presenta en la Tabla 7.1, y 7.2 respectivamente. Se definieron como variables globales, las variables que son usadas en mínimo dos rutinas escritas en diferentes documentos; en caso contario se definieron como locales al documento o a la rutina en sí.

El número de caminos que puede tener a lo más cualquier palabra son dos, el primero si se repite su último sonido, y el segundo si continua con el siguiente sonido. Por ejemplo, de la Figura 7.1, *el* tiene un camino, pues no exite ninguna palabra que a partir de allí inicie con *l* y que sea válida, pero *kampesino* tiene dos caminos, porque le pueden seguir *osa* y *sabe*.

Los campos de *trans*, *escri*, *porig* son definidos como arreglos de 30 caracteres, lo cual está adecuado considerando que la palabra de longitud máxima en la base de datos es de 20. El campo de *cat* es definido con un pequeño arreglo de tres caracteres.

La constante *nomaxpal* tiene el valor de diez, se cree que este valor es adecuado, pero en realidad no se ha verificado exhaustivamente. Por ejemplo, si tuvieramos /estasionamiento/, se pueden encontrar seis

palabras a partir de la primera *e*, que son: /e/, /es/, /esta/, /estas/, /estasion/ y /estasionamiento/.

En la Tabla 7.3 se despliega la información que contendría el arreglo de palabras encontradas para el ejemplo ilustrado en la Figura 7.1. La información correspondiendiente a cada uno de los campos de *infopal* ha sido separada por una coma. En *repson* puede contener un cero si la palabra a repetir es de un sonido, o un número que se calcula de la siguiente manera: el número de registro + la longitud de la transcripción. En *sinrep* puede contener un número que es igual a *repson* + 1, o tener *fin* si al hacer el cálculo anterior se pasa de *limarrent*; *fin* está definida como una constante cuyo valor es 255. El valor de *fin*, está en relación al número de sonidos que se pueden emitir sin hacer pausa, como se cree que nunca se llegará al valor de 255, entonces se uso dicho número. El campo de dirección no es ejemplificado.

7.2.2 Subrutina rev_ent

Como se explicó en la Sección 6.3, existen dos redes, una que hace la revisión sobre los patrones vccv, y la otra que lo hace para los patrones vcccv y vccccv. De acuerdo al patrón detectado es la red que se usa. Ver diagrama de flujo en la Figura 7.3.

a_r, i_r	Apuntador e índice a Revlex.rev
a_tc, i_tc	Apuntador e índice a Tab.cub
a_at, i_at	Apuntador e índice a Aux.tab
a_vb, i_vb	Apuntador e índice a VocBas
a_s, i_s	Apuntador e índice a Revlex.sal
a_err, i_err	Apuntador e índice a Revlex.err
arrent	Arreglo donde se encuentra la frase a revisar, Fx
tamarrent	Tamaño real del arreglo de entrada
limarrent	Límite del arreglo; es decir, el espacio actual utilizado
iarr	Indice del arreglo; es decir, el sonido a partir del cual se revisará, Fx(i)
nmspc	Número máximo de sonidos por comparar
arrpalenc	Arreglo de las palabras encontradas, definido como tipo *palenc*
bufRep	Buffer de reporte, definido como tipo inforep
banfin	Bandera que reporta el fin de frases a procesar; es decir, el fin de Revlex.rev
nollamaRepxf	Número de llamadas a la rutina de reportes
preg	Pila de registros. Pila donde se almacena el índice, i, de las frases por reportar
ppal	Pila de palabras. Pila donde se almacena el número de palabras encontradas a partir de Fx(i)
pcam	Pila de caminos. Pila donde se almacena el número de caminos con que cuenta cada palabra
pimpp	Pila de indicadores para la impresión de palabras
prepi	Pila de indicadores para la impresión del 1^{er} sonido
prepf	Pila de indicadores para la impresión del último sonido.

Tabla 7.1. Variables globales

palenc Estructura con dos campos:
- no Número de palabras encontradas a partir de Fx(i)
- info Campo de información, arreglo del tipo *infopal* con *nomaxpal* elementos

infopal Estructura con seis campos:
- trans Contiene la transcripción
- escri Contiene la escritura
- cat Contiene la categoría
- porig Contiene la palabra de la cual se deriva
- dir Dirección donde se encuentra esta información
- repson Indice a la siguiente Fx(i) pero repitiendo sonido
- sinrep Indice a la siguiente Fx(i) pero sin repetir el sonido

inforep Estructura de tres campos:
- x Contiene un no. de renglón de *arrpalenc*
- y Contiene un no. de columna de *arrpalenc*
- opci Indicador para definir si es opcional o no.

Tabla 7.2. Lista de tipos más importantes

reg.	# p.	información	
0	2	e, e, c., y, #, 0, 2	el, el, a., el, #, 2,3
1	0		
2	2	kampesino, campesino, aj., campesino, #, 11, 12	
		kampesinos, campesinos, aj., campesino, #, 12, 13	
3	1	a, a, p., a, #, 0, 5	
4	0		
5	1	pes, pez, n., pez, #, 8, 9	
6	2	e, e, c., y, #, 0, 8	es, es, v., ser, #, 8, 9
7	3	si, si, c., si, #, 9, 10	sin, sin, p., sin, #, 10, 11
		sino, sino, c., sino, #, 11, 12	
8	1	i, y, c., y, #, 0, 10	
9	2	no, no, av., no, #, 11, 12	nos, nos, pn., yo, #, 12, 13
10	1	o, o, c., o, #, 0, 12	os, os, pn., tu, #, 12, 13
11	1	sabe, sabe, v., saber, #, 15, 16	
12	2	a, a, p., a, #, 0, 14	abe, ave, n., ave, #, 15, fin
13	1	be, ve, v., ir, #, 15, fin	
14	1	e, e, c., y, #, 0, fin	

Tabla 7.3. Información almacenada en *arrpalenc* después de haber consultado las base de datos para la frase /elkampesinosabe/

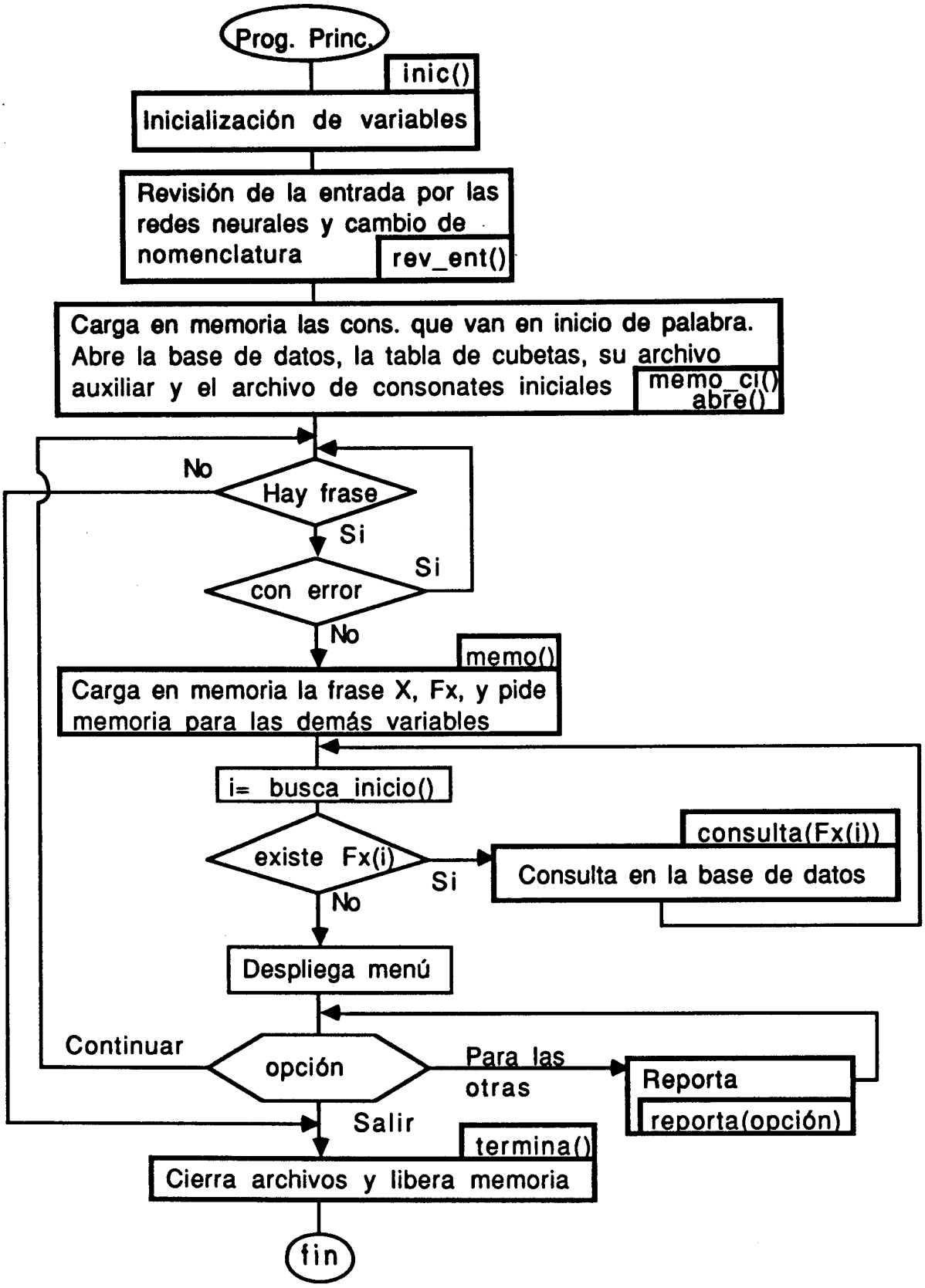


Figura 7.2. Diagarama de flujo del programa principal

7.2.3 Subrutina memo

Primeramente se cuenta el número de sonidos existentes en la frase. La variable limarrent toma ese valor. Después, si es la primera vez que se ejecuta, cuando todos los apuntadores valen cero, se define tamarrent igual a limarrent, se pide memoria para arrent, preg, ppal, pcam, prepi, prepf, pimpp y arrpalenc, de acuerdo a tamarrent. Si ya se corrió al menos una vez, entonces se pregunta si la dimensión que ya se tenía es suficiente para almacenar la nueva frase, si es así sólo se limpia los arreglos, y tamarrent será mayor que limarrent; en caso contario, se selecciona tamarrent igual al limarrent, se relocaliza los arreglos solicitando más área, y se limpian. Finalmente, se mueve la frase al arreglo de entrada, arrent.

7.2.4 Subrutina consulta

Como la tabla de cubetas tiene la información correspondiente a uno, dos, tres y cuatro sonidos, entonces la rutina tiene un ciclo principal de uno a cuatro. Con el sonido, o sonidos en cuestión se determina *h(x)*, se accesa la información de la tabla de cubetas, se lee la del archivo auxiliar, se almacena en memoria la cubeta, se verifica si hay información extra para el acceso, si lo hay se realiza el proceso para la obtención de la dirección definitiva a partir de la cual se buscará. Obteniendo la dirección, se buscan todas las palabras que puedan estar contenidas en *arrent*. Ver Figura 7.4.

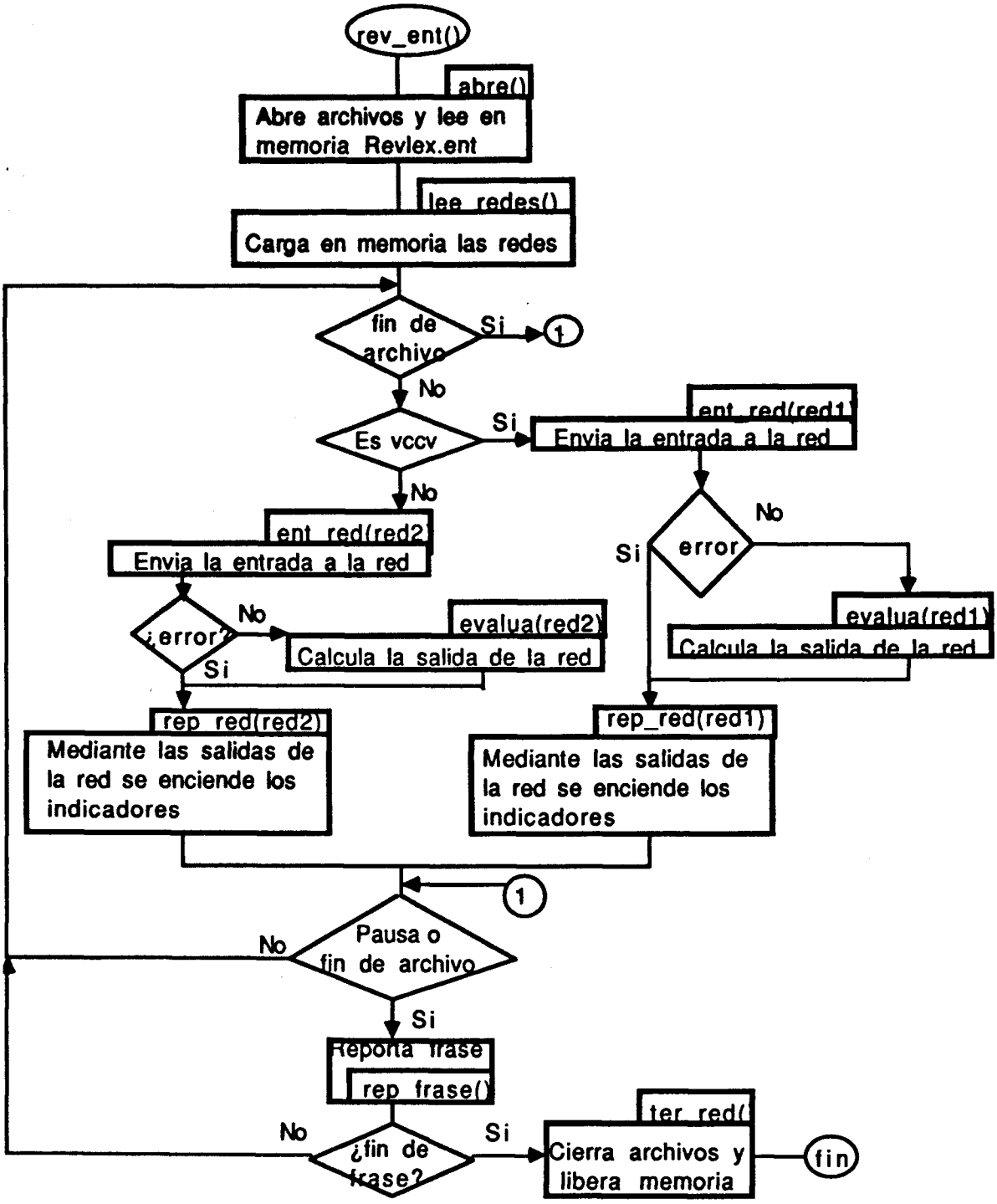


Figura 7.3. Diagrama de flujo de rev_ent()

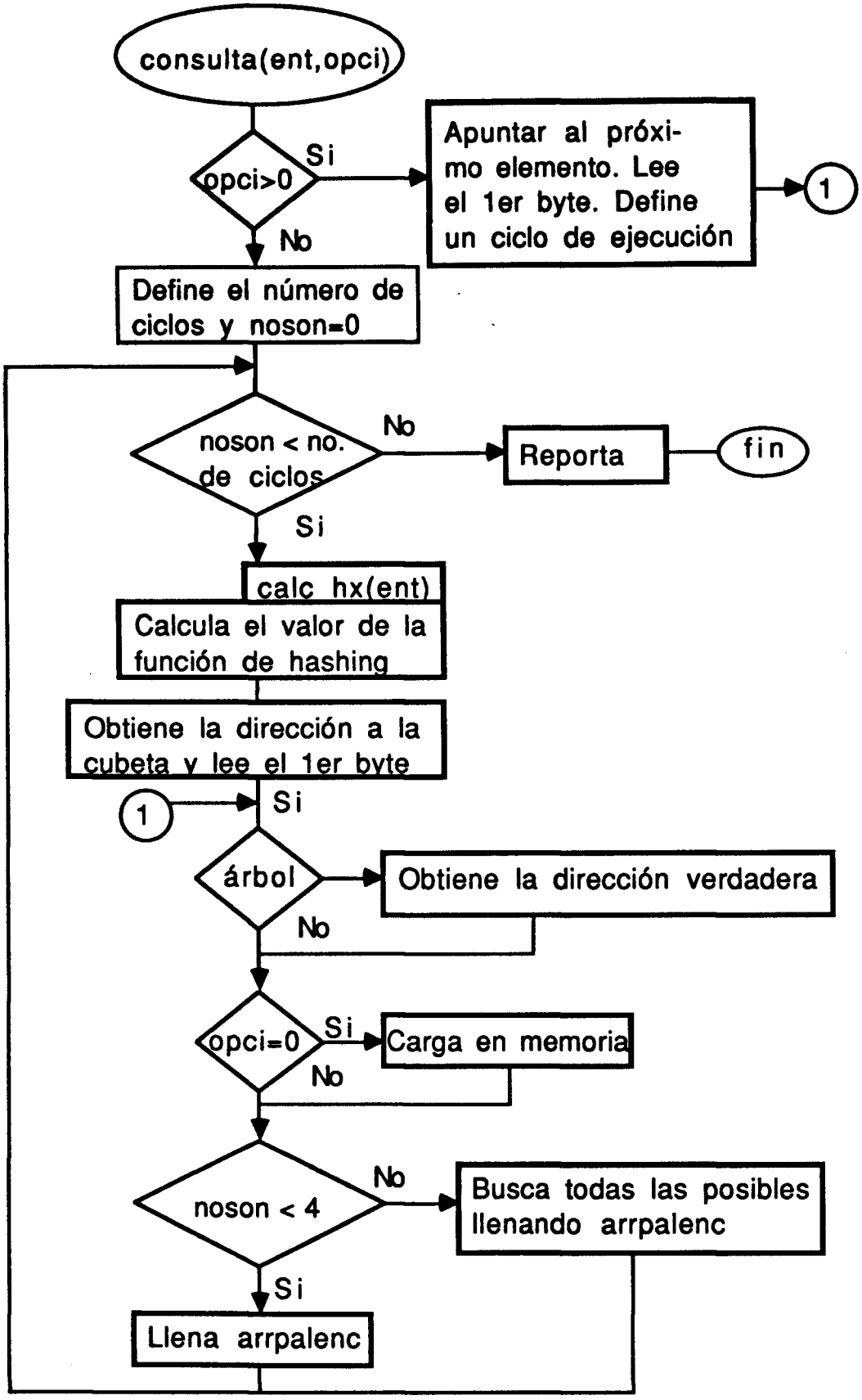


Figura 7.4. Diagrama de flujo de consulta()

En arrpalenc sólo se guardan la primera acepción de cada palabra que se encuentre contenida en la frase por analizar, de todas las que existan con la misma transcripción. Ya se mostró en la Tabla 7.3 un ejemplo de la información que se almacena en el arreglo.

Cuando se está en el cuarto ciclo, el número de sonidos que se compara depende del patrón encontrado en la cubeta y del número máximo de sonidos por comparar, *nmspc*. Recuérdese que cada vez que se hace el rastreo de izquierda a derecha, el índice de *arrent* disminuye y, por lo tanto, *nmspc* también se decrementa, y si la palabra encontrada en la cubeta es mayor que *nmspc* entonces se descarta. La búsqueda se realiza en toda la cubeta. Antes de guardalas se investiga si ya fué dada de alta alguna otra con la misma transcripción.

¿Por qué no se almacenan de una vez todas las acepciones de todas las palabras?. En cuanto a rapidez es mejor, pero es algo difícil determinar la dimensión adecuada de *infopal* si se piensa en la posibilidad de incrementar la base de datos. Pero, cuando se posea una base completa y se disponga de la suficiente memoria, entonces será definitivamente más conveniente. Para realizar el cálculo de infopal bastará asumir que de las diez palabras, *nomaxpal*, con diferentes transcripciones encontraremos *nomaxace* con igual transcripción. Donde *nomaxace* se define como el número máximo de acepciones encontradas en la base de datos para una palabra.

7.2.5 Subrutina reporta

La primera vez que se ejecuta esta rutina, se carga en las pilas la información concerniente al primer registro del *arrpalenc*. En *preg* se guarda el número de ese registro, en *ppal* se almacena el número de palabras asociadas a tal registro, y en *pcam* se introduce un dos, que es el número máximo de caminos que puede contener cada palabra. Las pilas prepi, prepf y pimpp son indicadores para señalar si se imprimirá el primer sonido de la palabra, la palabra en sí, o el último sonido. Después de ello se realiza un procedimiento algo diferente para cada una de las opciones comentadas en el programa principal, traer la siguiente oración, traer la misma palabra pero con la siguiente o anterior categoría, pedir cambio de frase, pedir todas las combinaciones, continuar procesando o salir. En la Figura 7.5 se muestra el diagrama de flujo de esta subrutina.

Para obtener la siguiente oración, se instalan nuevos elementos en las pilas, pero disminuyendo el número de caminos o de palabras de la entrada anterior, según sea lo indicado. Cuando se recorre toda una ruta hasta encontrar un fin se reporta el resultado. El mecanismo para la obtención de todas las combinaciones se basa en el de la oración. La opción de cambio de frase consiste en eliminar el elemento actual de la pila y continuar con la siguiente palabra del elemento anterior. La idea de implementar esta opción es la de dar oportunidad de analizar otra frase diferente a la ya previamente analizada, cuando se determine que la estructura a partir de esa palabra está equivocada; ahora bien, si la estructura es adecuada lo correcto es pedir la

siguiente oración. Aunque antes de pasar a la opción de cambio de frase es conveniente buscar si existen otras palabras con la misma transcripción pero con diferente categoría. Para encontrar las otras acepciones se buscará a partir de la ya obtenida, pudiendose hacer hacia adelante o hacia atras.

En la Tabla 7.4 se muestra como quedarían las pilas al solicitar la primera oración de la frase */elkampesinosabe/* de acuerdo a la información proporcionada en la tabla anterior.

p\r	0	1	2	3	4
preg	0	2	10	11	14
ppal	1	2	2	1	1
pcam	1	1	0	1	0
prepi	1	0	0	0	0
prepf	0	1	0	1	0
pimpp	1	1	0	1	0

Tabla 7.4. Estado de las pilas después de pedir la primera oración para el ejemplo de la Tabla 7.3

La oración resultante es: la segunda palabra del registro cero y repitiendo el primer sonido, que es *e el*, la primera palabra del registro dos y repitiendo el sonido final, que es *kampesino {o}*, no se imprime el registro diez, la primera palabra del registro once y repitiendo el sonido final, que es *sabe {e}*, y la primera palabra del registro catorce no se imprime. Reafirmando, en *preg* se almacenan los números de registros asociados a *arrpalenc*, en *ppal* el número de palabras en el registro en cuestión y en *pcam* el número de caminos; donde *pcam* es decrementado en uno cuando se revisa el siguente camino, y *ppal* se disminuye en uno cuando *pcam* es igual a cero. Las

pilas prepi, prepf, pimpp como se nota nos inidcan si se reporta la vocal inicial, final o la palabra en si.

Cuando se solicita la siguiente oración la información obtenida además de ser desplegada en pantalla es guardada en memoria, indicandose cuando es opcional o no.

7.2.6 Programas para la creación de la Base de Datos

Existen dos programas que dan soporte a la formación de la base de datos. No es posible crearla en forma automática por la existencia de exepciones a las reglas generales de transcripción, además de la necesidad de ordenar la base para formar las cubetas. El primer programa se llama *AgregaSon.c* y el segundo *CreaBD.c.*

AgregaSon sirve para agregar la transcripción a las palabras, además genera el archivo base sobre el cual se realizaran las estadísticas necesarias para la implementación de las redes, y se pueden procesar varios documentos en una sóla corrida. Las reglas que se usaron para transcribir se presentan en la Tabla 7.5.

El formato del archivo de entrada es el usado en Juilland et al. (1964) con las siguientes exepciones:

1. A cada unidad léxica le debe anteceder un símbolo de #.
2. Debe existir un espacio entre el lexema y la categoría. En caso de necesitar hacer algún comentario es posible agregarselo inmediatamente después de la palabra primitiva teniendo el cuidado de no dejar espacios en blanco.
3. Las palabras derivadas que inicien con guión, "-", no son tomadas en cuenta.

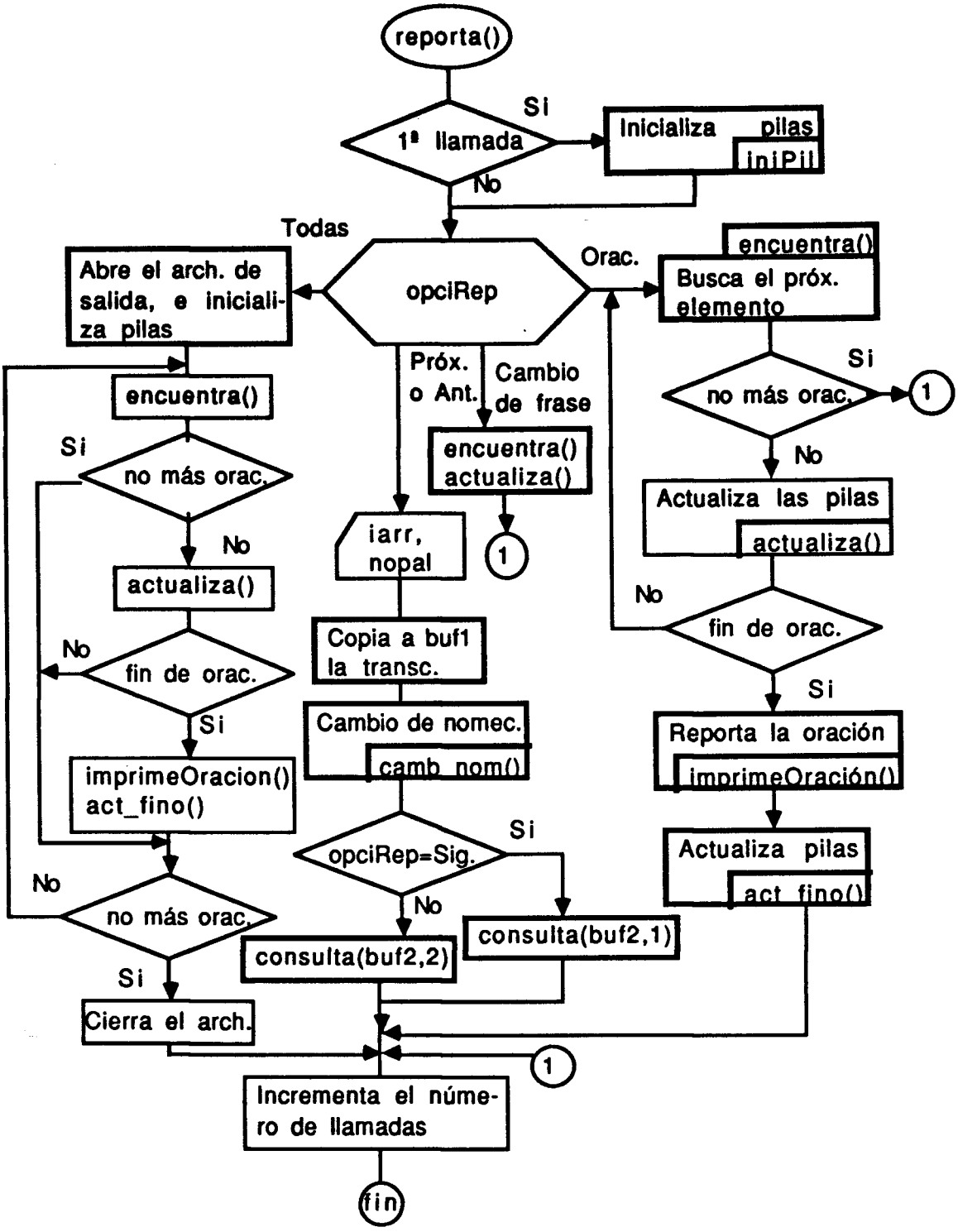


Figura 7.5. Diagrama de flujo de reporta()

4. La lista de palabras debe de finalizar con el símbolo de #.

La salida es escrita en *VocBas.txt*. El archivo de reportes *VocBas.rep* contendrá exclusivamente la transcripción de cada una de las palabras almacenadas en el de salida. El archivo de salida contiene además de las transcripciones, la escritura de la palabra, su categoría y la palabra de la cual se origina; agrupandose lo más posible las palabras derivadas que no sean verbos, ni pronombres, pero que tengan la misma categoría. Dado que no se creó un mecanismo para determinar la posición del acento de las palabras que no lo llevan escrito, entonces la presencia del acento en muchos casos hace que la agrupación termine, o si no es así es debido a la presencia de un sonido diferente.

El formato del archivo de salida se muestra en la Figura 7.6.

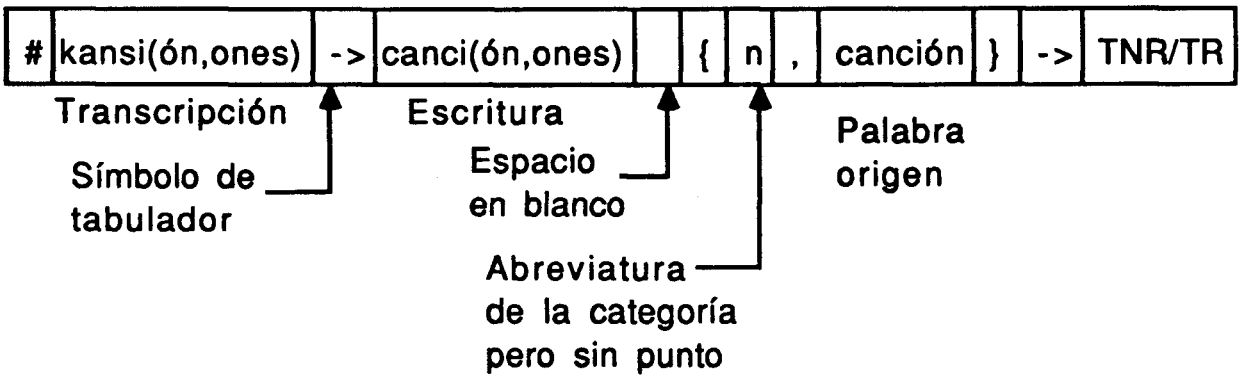


Figura 7.6. Formato del archivo de salida del programa *AgregaSon*

Después de procesar los archivos, es necesario verificar las transcripciones, o agregar más registros cuando alguna palabra tiene más de una transcripción válida, o reagrupar los registros colocando el acento donde es adecuado, y verificandose que se tengan, para el

caso de palabras de más de cuatro sonidos, al menos cuatro de ellos sin agrupar para que el sorteo se realize adecuadamente y no se tenga la necesidad de reacomodar varios registros. La ordenación debe de ser de acuerdo a su transcripción fonológica. Se utilizó *Cuarta Dimensión* para realizar el sorteo, dejando la información en *VocBas.st*.

Existen cuatro estructuras ya creadas en *Cuarta Dimensión*, uno para leer *VocBas.txt*, las otras para leer *const.ini*, *cons.int*, *cons.fin*, archivos formados por *EstEsp*, y realizar la ordenación.

Mediante *CreaBD.c* se forma la base de datos, todos los archivos auxiliares para el direccionamiento, y un archivo de reporte. En la Figura 7.7 se presenta su diagrama de flujo.

7.2.7 Programas de apoyo para la creación de las redes

Se creó un programa llamado *EstEsp* c mediante el cual se cuenta el número de palabras en la base de datos, se lleva un lista del número de palabras por sus diferentes longitudes, se almacena en *cons.ini* *cons.int* y *cons.fin* los limites consonánticos de cada palabra, los de inicio y fin, y los grupos consonánticos que se presente en el interior de palabra, pero que tenga más de un sonido.

Para detectar las consonantes iniciales, se busca de izquiera a derecha hasta encontrar la primera vocal; para detectar las consonates finales, se busca de derecha a izquierda hasta detectar la primera vocal; y para encontar los grupos intermedios, se buscan de izquierda a derecha las dos primeras vocales, si existe más de un sonido intermedio entonces se escribe en *cons.int*, y se repite la operación hasta que no detecte una segunda vocal.

a, á, b, d, e, é, f, i , í, k, n, ñ, o, ó, s, t, u, ú, w	Pasan idénticos.
c	Si le sigue e, é, i, í se transcribe como "s". Si le sigue h se transcribe como "ch". En caso contario como "k".
g	Al inicio de palabra en la comb. gn, se elimina la g. Si le sigue u se transcribe como "g". Si tenemos gue, gué, gui, guí la u se omite, y si no se transcribe. Si después de la g va e, é, i, í la g se transcribe como "x". En caso contrario se transcribe como "g".
h	No se transcribe.
j	Se transcribe como "x".
l	Si le sigue l se transcribe como "y". En caso contrario como "l".
m	Si inicia con mn, se quita la m. En caso contario se transcribe.
p	Si inicia con ps o pt se elmina la p. En otro caso se transcribe.
q	Se transcribe como "k" y se omite la u.
r	Al inicio de palabra, o en lr, ns, y sr se transcribe como "rr". En caso contrario se transcribe como "r".
ü	Se transcribe como "u".
v	Se transcribe como "b".
x	Al inicio de palabra se transcribe como "s". En caso contrario como "ks".
y	Si va a final de palabra se transcribe como "i". En caso contrario como "y".
z	Se transcribe como "s".

Tabla 7.5 Relaciones usadas para la conversión de símbolos ortográficos a fonemas

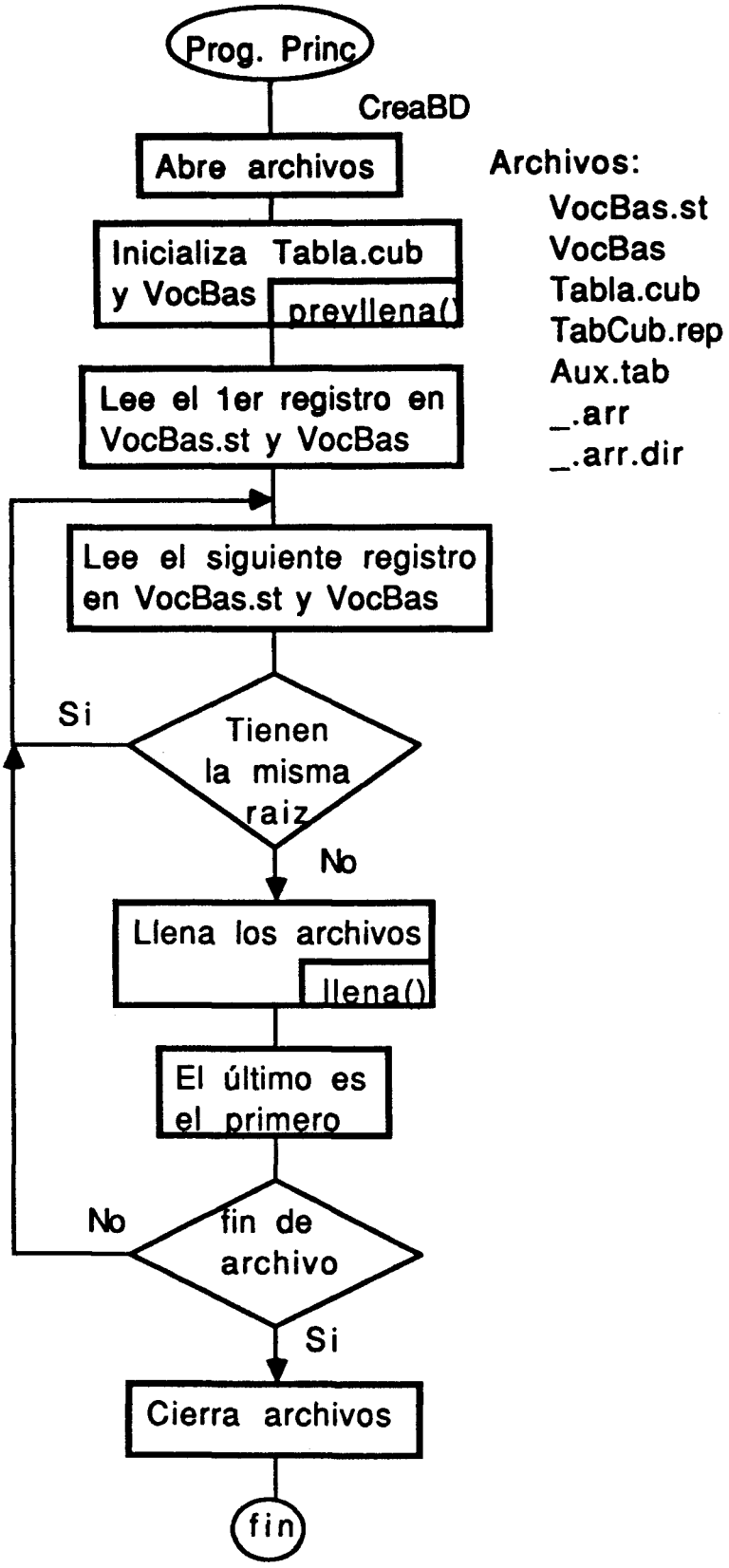


Figura 7.7. Diagrama de flujo de CreaBD.c

Ordenando los archivos y eliminando los patrones repetidos se puede obtener la información reportada en las Tablas 6.4, 6.5, 6.6 y 6.9.

Para generar los patrones maestros de la red se creó un programa llamado *creapat*. El cual lee del *doscons.pat* los grupos de dos consonantes y la salida esperada, y crea *red.pat* donde aparte de esos dos campos agrega las señales de entradas para las 36 unidades de entrada.

7.3 Resumen

Conforme al avance en el conocimiento del problema los mecanismos para solucionarlo fueron cambiando. Muchas ideas fueron olvidadas al comprobar su ineficacia o la imposibilidad de implementarlas. De ellas se pueden mencionar: el recuento de los grupos vocálicos, las reglas para la reducción de vocales, el manejo de las diferentes transcripciones, el empleo del acento como demarcador del inicio de la próxima palabra, el uso de modelos verbales para obtener las conjugaciones verbales, la salida del sistema y el formato de la salida, la optimización en el número de consultas, diferentes estructuras y manejo de la base de datos, el uso de una red neuronal para la división de sílabas.

Se programó en C, y casi todos, excepto uno implementado en VAX, fueron desarrollados dentro del ambiente de trabajo MPW en una Macintosh II. La versión final del sistema se encuentra integrada en el programa *Revlex*, revisión de léxico. Este se forma al compilar y ligar cuatro archivos *Revlex.c*, *red.c*, *consulta.c* y *reporta.c*.

La entrada del sistema, que es la transcripción fonológica de las frases dichas por el usuario, debe ser proporcionada en el archivo *Revlex.ent*. La salida del sistema puede ser mediante el archivo, *Revlex.sal*, si se solicitan todas las frases posibles, o guardada en memoria. Si se detecta alguna combinación de consonantes inválida para el actual vocabulario, la frase que la contega no se analiza y se reporta en *Revlex.err*.

Para llevar a cabo la revisión de léxico, primeramente se pasa la entrada por dos redes neurales, que nos proporcionan información de la presencia de grupos consonánticos que se dan entre palabras o no se encuentran en el vocabulario. Se revisa cada frase de izquierda a derecha encontrando todas las palabras contenidas en esta, haciendo consultas a la base de datos. Después de esto, el usuario puede solicitar la información, pidiendo oraciones, o todas las combinaciones. Es factible solicitar que encuentre las otras acepciones de una determinada palabra o regrese a alguna en particular, y además hacer cambio de frase para acelerar el proceso de búsqueda.

Se explican, dentro de la Sección 7.2, algunas características de los procedimientos más importantes del sistema, y para concluir se mencionan los programas de apoyo que se realizaron para facilitar la construcción de la base de datos y la implementación de las redes neuráles.

Capítulo 8
Resultados

La implementación del sistema es satisfactoria, se realiza la revisión de léxico, y éste puede proporcionar todas las oraciones factibles.

8.1 Redes Neurales

El entrenamiento de la red fue infructuosa. Primero, según lo acordado en la Sección 6.3, se utilizaron todos las combinaciones de dos consonantes para entrenarla. Se hicieron pruebas con diferentes configuraciones, 50, 52, 55, 60 y 80 unidades ocultas dispuestas en una capa, utilizando diferentes valores para el momento y la razón de aprendizaje, con diferentes semillas, presentando los patrones en forma secuencial o aleatoria, pero el mejor valor obtenido para la suma de los errores al cuadrado, *tss*, fue de ocho. Después, se probó con dos capas, usando 30 y 30 ó 40 y 40 unidades ocultas; pero los resultados fueron comparables.

Dado que el seleccionar un número arbitrario de unidades ocultas no dio resultado, entonces revise las Tablas 6.6, 6.7 y 6.8 para tratar de encontrar el número aproximado de unidades que se necesitaban. Haciendo grupos en base a la primera consonante de la Tabla 6.6 se obtienen doce, de la 6.7 son siete, y de la 6.8 hay dieciocho, esto nos da una suma de 37 agrupaciones; es decir, de 37 unidades ocultas,

haciendo pruebas la menor tss obtenida fue de trece. Dado la dificultad aparente me cercioré de que fuera obtenible la salida deseada, calculando los pesos, con -15 como bias en las unidades ocultas, -5 en la de salida, y pesos de diez, la respuesta es correcta. Al hacer esto último, me di cuenta que en realidad no se requiere tener todas las 37 unidades, sino tan sólo 19 (=12+7) unidades, porque los patrones restantes pueden inhibir la salida tres.

Continué con el entrenamiento, pero ahora con 19 unidades, y con la mitad de patrones, eliminado un patrón en forma alternante. El valor obtenido para tss después de 5,550 iteraciones fue de 3.0012, pero la disminución realizada en las últimas 1,200 interacciones fue de .0008, probando todos los patrones se encontró que faltaban de mapear dos patrones, rr con .9 0 0 en vez de 0 0 1 y xr con 0 0 0 en vez de 0 0 1. Continue hasta 3,400 ciclos más y tss obtuvo el valor de 3.0003, sin embargo su decaimiento en las últimas 2,000 fue de 0.0002, viendo cada uno de los patrones maestros, las combinaciones rr y xr seguian igual, y probando el resto de éstos once fueron incorrectos, considerando como uno arriba de 0.5 y cero abajo de 0.5. En la Tabla 8.1 se da la relación de los patrones mal clasificados y su valor esperado.

Como se observa en la Tabla 8.1, no parece haber un patrón regular en las respuestas, aunque es lógico pensar que esto se debe a la mayor presencia de sonidos vocálicos en otros grupos. Si la respuesta, aunque siendo errónea, habilitara en algo a dos o tres de las salidas, dicha red se podría usar para el sistema; pero la habilitación de una sola y en el lugar indebido hace que esta red no sea adecuada.

fs	.87	0	.42		0	0	1
gs	.97	0	.43		0	0	1
km	.99	0	0		0	0	1
kp	.16	0	.29		0	0	1
ml	.43	0	0		0	0	1
sr	.99	0	0		0	0	1
td	.96	0	0		0	0	1
tn	.99	0	0		0	0	1
tp	0	0	.2		0	0	1
tt	.99	0	0		0	0	1
xñ	0	.87	.99		0	0	1
xw	0	.99	.99		0	0	1

Tabla 8.1. Patrones mal clasificados para la red de 19 unidades

Posteriormente, creí que restringir el valor de la salida tres podría ayudar a la red ha aprender más rápidamente y mejor, para ello le restringí el peso de sus conexiones a 10 y su bias a -5, y se entrenó con los grupos consonánticos que se presentan dentro y entre palabras. Su velocidad de aprendizaje fue más rápida, a las 1,675 interacciones tss fue de 2.0019; pero la respuesta fue muy mala nunca se encendió S3, y en ocasiones se habilitaba S1 o S2, o ambas. Pienso que esto se debe a la baja probabilidad que el sistema aprenda a mapear de la misma forma en como yo pensé. Enntonces, no restringí la salida.

Con la idea de que la red habilitara varias salidas cuando la señal era incorrecta, realice dos pruebas más, una usando todos los patrones y otra usando el subgrupo de patrones más los que no se pudieron aprender correctamente; pero ahora usando dos salidas e indicando que en caso de error la salida sea cero. Pero, la respuesta no fue adecuada para ninguno de los dos casos. Por lo tanto, opté por calcular los pesos de las redes.

En relación a la segunda red, los grupos que se formaron, para cada unidad oculta, son los expuestos en las Tablas 8.2, 8.3 y 8.4. Pero, podría verse como una sola capa con señales que interactuan con otras unidades en el mismo nivel. De hecho, no se usan unidades, con función de memoria, para almacenar los valores consonánticos que se suman a los grupos de combinaciones, Tabla 8.4, sino que se toman directamente de la entrada. Conceptualmente esto no es debido, pero con el propósito de ahorro del número de operaciones se diseñó la red de esa forma. Además, para una implementación real es preferible no reducir el número total de unidades, sino el número de capas empleadas que será el número de pasos ejecutados.

bs1, ks1, ns1, nt1, rd1, bl2, br2, dr2, fl2, fr2, gl2, gr2, kl2, kr2, pl2, pr2, sp2, st2, tr2, bl3, br3, dr3, fl3, fr3, gl3, gr3, kl3, kr3, pl3, pr3, tr3, {k, t}3, {k, p, s, t}3, {f, k, p, s, t}3, {b, ch, d, f, g, l, m, n, p, rr, s, x, y}3, {b, ch, d, f, g, k, l, m, n, p, rr, s, t, x, y}3

Tabla 8.2. Unidades ocultas para la 1ª capa, en la segunda red

{dr, kr, tr}2, {bl, br, pl, pr}2, {dr, fl, fr, gl, kl, kr, tr}2, {fl, pl, pr, sp, st}2, {fr, gr, kl, kr, pl, pr, tr}2, {bl, br, dr, fl, fr, gl, gr, kl, kr, pl, pr, tr}2, {bl, br, fl, fr, gl, gr, kl, pl, pr}2, {bl, br, dr, fr, gl, gr, kl, kr, tr}2, {bl, br, dr, fl, gl}2, {bs, ks, ns}1, {kl, pl, pr}3, {bl, br, dr, fl, fr, gl, gr, kl, kr, pl, pr}3,{bl, br, dr, fl, fr, gl, gr, kl, kr, pl, pr, tr}3

Tabla 8.3. Unidades ocultas para la 2ª capa, en la segunda red

bs1+{k,t}3, ks1+{k, p, s, t}3, k1+tr2, l1+{dr, kr, tr}2, m1+{bl, br, pl, pr}2, n1+{dr, fl, fr, gl, gr, kl, kr, tr}2, ns1+{f, k, p, s, t}3, r1+{fl, pl, pr, sp, st}2, s1+{fr, gr, kl, kr, pl, pr, tr}2, b1+{bl, br, dr, fl, fr, gl, gr, kl, kr, pl, pr, tr}2, bs1+{b, ch, d, f, g, l, m, n, p, rr, s, x, y}3, d1+{bl, br, dr, fl, fr, gl, gr, kl, kr, pl, pr, tl, tr}2, l1+{bl, br, fl, fr, gl, gr, kl, pl, pr}2, n1+{bl, br, pl, pr}2, nt1+{b, ch, d, f, g, k, l, m, n,p, rr, s, t, x, y}3, r1+{bl, br, dr, fr, gl, gr, kl, kr, tr}2, rd1+{b, ch, d, f, g, k, l, m, n, p, rr, s, t, x, y}3, s1+{bl, br, dr, fl, gl}2, x1+{bl, br, dr, fl, fr, gl, gr, kl, kr, pl, pr, tr}2, {bs, ks, ns}1+tr3, ks1+{kl, pl, pr}3, nt1+{bl, br, dr, fl, fr, gl, gr, kl, kr, pl, pr, tr}3, bs1+{bl, br, dr, fl, fr, gl, gr, kl, kr, pl, pr}3

Tabla 8.4. Unidades ocultas para la 3ª capa, en la segunda red

8.2 Base de Datos

La base de datos, *VocBas.st*, resultante tiene un tamaño de 500K, si no se realizara la agrupación de palabras, *VocBas.sr*, tendría un tamaño de 764K, lo cual implica que se obtuvo un 35% de reducción.

De la información proporcionada por Juillian et al. sólo se descartaron seis palabras, porque indica la frecuencia de esas palabras en secuencia de otra, por ejemplo: la (que), y que (el). Trabajando con 22,883 palabras, en la Tabla 8.5 se da la relación del número de palabras existentes con respecto a su longitud, es decir, a su número de sonidos. La palabra de mayor longitud es de 20 sonidos, la mayor frecuencia de palabras se registró a los siete sonidos, la longitud promedio de la palabra es de ocho.

El número de cubetas creadas fue de 2,741 (27%), contra las 10,000 existentes en el diccionario de la Real Academia (1984). La condición usada para crear los arreglos es que si existen más de diez palabras y si el tamaño del arreglo no pasa de cuatro, entonces se generan. De las

2,741 cubetas 562 cumplieron la condición propuesta, y de ellas 493 (88%) arreglos fueron generados de una dimensión, 56 (10%) de dos dimensiones, y 13 (2%) de tres dimensiones. En la Tabla 8.6 se presentan algunos datos proporcionados para las trece arreglos de tres dimensiones.

No de sonidos	No. de palabras
1	13
2	81
3	263
4	1114
5	2348
6	3355
7	4129
8	4107
9	3052
10	2153
11	1254
12	555
13	254
14	124
15	64
16	15
17	1
20	1

Tabla 8.5. Lista del número de palabras con respecto a su longitud

Sin embargo la cubeta más llena, con 197 registros, tiene un arreglo de dos dimensiones. El número de sonidos máximo que se puede especificar para accesar algunos registros, es decir el valor limite, es de doce, pero, estas cubetas tienen un solo registro, es decir que las palabras que son de longitud mayor de trece poseen valores limites más pequeños. El espacio ocupado en disco por todos estos arreglos y archivos auxiliares es de 5,220Kbytes.

Raíz	No. reg.	Valor limite
adel	11	8
akon	21	8
akre	12	8
asse	11	8
desk	82	8
enko	33	8
konm	11	8
nese	25	8
pref	16	8
preo	15	8
pret	20	8
rrebo	16	8
sorp	12	8

Tabla 8.6. Datos acerca de los arreglos generados de tres dimensiones

En el archivo cons.ini se registraron 12,977 campos; es decir, que de las 22,883 palabras, 12,977 (56%) inician con consonante. Haciendo el ordenamiento y eliminando las repetidas se obtuvieron sólo 27, además, se encontró que 10,983 (48%) palabras terminan en consonante, pero sólo hay 10 sonidos diferentes. Con respecto a los grupos consonánticos que se presentan a mediación de palabra se registró una frecuencia total de 16,403, y sólo 125 grupos diferentes.

Capítulo 9
Conclusiones

Al parecer la dificultad del entrenamiento de la red se debe a la presencia de patrones similares que tienen que ser clasificados de distinta manera, será necesario agregar más entradas y buscar alguna codificación tal que para cada caso los patrones de entrada sean suficientemente diferentes. Si el entrenamiento de la red hubiera sido satisfactorio, usando un subconjunto de las combinaciones, entonces, se hubiera procurado el entrenamiento de la otra red, la que contiene los grupos de tres y cuatro sonidos, e inclusive se podrían unir ambas redes en una sola. Para la implementación actual en el equipo Macintosh no es indispensable haber realizado este tipo de validaciones mediante las redes, sino que manejar las combinaciones como una pequeña base de datos hubiera sido suficiente. Cuando se llege el momento de hacer una implementación en linea pienso que si sería conveniente considerarlas por su operación en paralelo.

Aunque el método de direccionamiento es deficiente, permitió conocer algunas características de nuestro idioma, y pienso que la información proporcionada más la no presentada, debido a su enrome cantidad, podría servir para calcular los índices de un archivo de índices o de un árbol propiamente.

Creo que falta mucho por hacer para terminar de implementar el sistema de reconocimiento de voz, pues quedan pendiente el tratamiento de las palabras no existentes en la base de datos,

optimizar el manejador de la base de datos o la estrucura de la base de datos en si, completar la revisión sobre las trascripciones, aumentar posiblemente la base de datos, e implementar las revisiones de gramática y semántica, y la evaluación del uso del acento de intensidad y tónico como otras fuentes de información. Además de concluir, acoplar y depurar las fases anteriores a la revisión de léxico e implementar formalmente el sistema de síntesis.

Sin embargo, podríamos obtener una ayuda muy valiosa de la Universidad de las Américas, del Colegio de México, de la Facultad de Ingeniería de la Universidad Nacional Autónoma de México o del grupo de Inteligencia Artificial de la Universidad Autónoma Metropolitana - Iztapalapa.

En la Universidad de las Américas (Jiménez, 1988) se reporta la implementación de un analizador determínístico para el Español, mediante el uso de los conceptos de gramática trasformacional, donde a cualquier tipo de oración escrita se le puede realizar un análisis gramatical y semántico.

Otro apoyo podría ser el sistema computacional del Diccionario del Español de México realizado por el Colegio de México (García, 1988), cuyo objetivo original era hacer estadísticas sobre las palabras más frecuentes encontradas en una muestra de dos millones de palabras ya incorporadas en la computadora, pero debido a la falta de un diccionario electrónico en nuestra lengua, se implementaron reglas de reconocimiento de la morfología de vocablos, reglas de precedencia y concordancia para designar la categoría gramatical a las palabras del corpus, y así se formó un diccionario básico, se realizaron las

estadísticas y se implemento también la estructura de información para la recuperación de concordancias.

En la Facultad de Ingeniería de la Universidad Nacional Autónoma de México (Espinosa, 1988), cuentan con la experiencia del manejo de Neurocomputadoras; y en el grupo de Inteligencia Artificial de la Universidad Autónoma Metropolitana - Iztapalapa (Figueroa y Vargas, 1988), han trabajado con procesadores paralelos, procesadores celulares y sistemas conexionistas, tanto en software como en hardware.

Con su ayuda se podría llegar a implementar, en forma más rápida, 1) la Gramática Generativa Transformacional y 2) un sistema de reconocimiento de voz que opere en tiempo real.

Apéndice A

En este apéndice se explican cuales son los pasos que hay que seguir para la construcción de la base de datos a partir de la escritura de las palabras o incorporando registros completos.

A.1 A partir de la escritura

Crear un archivo con el nombre que se guste, siguiendo las convenciones explicadas en la Sección 7.2.6. Ejecutar el comando *AgregaSon,* que fue diseñado como una herramienta de *MPW*, la salida se dará en *VocBas.txt*, el cual es abierto sin destruir su información original, verificar que las palabras que posean más de cuatro sonidos tengan al menos cuatro sonidos no agrupados, y corregir su transcripción en caso de falla. Pasar a la siguiente sección.

A.2 A partir del registro

Si no fue ejecutado *AgregaSon*, entonces se puede añadir registros libremente pero siguiendo el formato especificado en la Sección 6.2.2, pero sin incluir los campos del primer renglón, usados para el acceso a la información.

Habiendo realizado las modificaciones en *VocBas.txt*, abra el folder *Diccf*, y después el archivo *Dicc*, seleccione en el menú de *Use* el comando de *User.* Debe aparecer un letrero que le avise que no hay nada en la base de datos. Seleccione del menú *File* el comando de *Importar* y

abra el archivo de *VocBas.txt*, le preguntará un nombre para la base, podría dejar el default, que es *Dicc.data*. Luego, del menú *Select* escoja la opción de *Sort Select*. Aparecerá un diálogo, donde usted tiene que seleccionar el campo de la transcripción y presionar el botón de que realize el *sort*. Después de haber realizado el ordenamiento, realizé un *Export*, del menú *File*, y almacene los datos en *VocBas.st*, y debe salirse de *Cuarta Dimensión* con la opción de *Quit* del menú File.

Elimine el archivo *.data* creado por *Cuarta Dimensión*, porque al momento de hacer otro *Import* agrega al archivo *.data* los nuevos registros leidos, entonces si no se destruyen a la hora de realizar otro sorteo se tendrá información duplicada. Hecho esto, abra *CreaBD* y ejecute este comando.

Tanto en *AgregaSon* como en *CreaBD* se reporta en pantalla el proceso de los registros, si en determinado momento existe algún problema de ejecución, los programas se paran por sí mismos, o el usuario los puede detener al presionar las teclas de *command* y *punto* a la vez.

Para realizar el ordenamiento de *cons.ini*, *cons.int*, *cons.fin* se tienen tres estructuras guardadas en *Cons_inif*, *Cons_intf*, y *Cons_finf*. El procedimiento seguido es idéntico al descrito.

Apéndice B

B.1 Declaración de tipos y constantes

```
/***** TiposyCtes.h *****/
/*************************************************
* Este archivo contiene constantes, tipos y
* archivos incluidos que son necesarios agregar
* en todos los modulos
*************************************************/
#include <types.h>
#include <stdio.h>
#include <sane.h>
#include <FCntl.h>
#include <ctype.h>

#define tamrai   4
#define base     23
#define nocons   18
#define nocartc  2
#define nocarat  18
#define nocaraa  9
#define nocomb   292560  /*292560=23^4+23^3+23^2+23*/
#define ini_rai  1
#define conti    2
#define arreg    3
#define crit_arr 10
#define nomaxpal 10 / *  número máximo de palabras encontradas al
                         realizar una consulta*/
#define maxpal   30 /*long. máxima de una palabra*/
#define fin      255     /*clave para arrpalenc*/
#define finreg   '\n'
#define tambuf   300
#define bien          0
#define no_mas_palab -1
#define no_mas_orac  -2
#define no_otra_cat  -3
#define tamcat   3   /* el último siempre debe tener 0 */
```

```
#define  defi "definitions:"
#define  nuni "nunits"
#define  ninp "ninputs"
#define  nout "noutputs"
#define  net  "network:"
#define  bias "biases:"
#define  constr   "constraints:"
#define  end  "end"

typedef  enum {a,b,c,d,e,f,g,i,k,l,m,n,nn,o,p,r,rr,s,t,u,w,x,y}  fonemas;
/* nn = ñ, rr = - */
typedef unsigned char      raiz[tamrai+1];
typedef unsigned char      palabra[maxpal];
typedef unsigned char      ds[2];
typedef struct infopal {   /*información sobre una palabra*/
     palabra           trans,escri;
     char              cat[tamcat];
     palabra           porig;
     long              dir;
     unsigned char     repson,sinrep;
} infopal;
typedef struct palenc {/*palabras encontradas a partir de un sonido*/
     unsigned char     no;
     infopal           info[maxpal];
} palenc;
typedef struct inforep {   /*información  reportada*/
     unsigned char     x;   /* # de renglón en arrpalenc */
     unsigned char     y;   /* # de columna en arrpalenc */
     unsigned char     opci;/* bandera que indica si es opcional o no */
} inforep;

typedef struct restric {
     char      nom;
     float     val;
}  restric;

typedef  struct    red {
     unsigned int nunid,nuent,nusal;
     float        *entext,*entnet,*polar,*activ,*pesos;
} red;
```

B.2 Declaración de variables globales

```
/*****    VGlob.h    *****/
/*************************************************
* Este archivo contiene las variables globales
* de REVLEX. Debe ser sólo incluido en el
* modulo que contenga al programa principal
*************************************************/

FILE        *a_r,*a_vb,*a_tc,*a_at,*a_s,*a_err;
      /*    apuntadores a: Revlex.rev, VocBas, Tabla.cub,
            Aux.tab, Revlex.sal y Revlex.err */
long        i_r,i_vb,i_tc,i_at,i_s,i_err;/* indices a los archivos */
unsigned char
      *arrent,tamarrent,limarrent,iarr,nmspc,banfin,noferr;
unsigned char    nollamaRepxf,*preg,*ppal,*pcam,*pimpp,
                 *prepi,*prepf,opciRep;
palenc      *arrpalenc;
inforep     bufRep[tambuf];

/*****    VGlobExt.h *****/
/*************************************************
* Este archivo contiene las variables globales
* externas de REVLEX. Debe ser sólo incluido en
* todos los modulos.
*************************************************/

extern      FILE *a_r,*a_vb,*a_tc,*a_at,*a_s,*a_err;
extern      long i_r,i_vb,i_tc,i_at,i_s,i_err;
extern      unsigned char
            *arrent,tamarrent,limarrent,iarr,nmspc,banfin,noferr;
extern      unsigned char
      nollamaRepxf,*preg,*ppal,*pcam,*pimpp,*prepi,*prepf,opciRep;
extern      palenc     *arrpalenc;
extern      inforep    bufRep[tambuf];
```

B.3 Código del programa Revlex.c

```
/ *  Aqui está el procedimiento principal de Revlex,
     junto con las rutinas para abrir, cargar en
     memoria, buscar inicios de palabras, y terminar */


#include <TiposyCtes.h>
#include <VGlob.h>

/* Declaración de procedimientos presentes en el archivo */

int inic(),memo_ci(),abre(),memo();
int busca_inicio(),termina(),lib_ci(),cierra();

FILE *a_ci; /*apuntador a los archivos con consonantes iniciales */
unsigned char     noci1s,noci2s;
/ *  noci1s: # de cons. iniciales con 1 sonido
     noci2s: # de cons. iniciales con 2 sonidos */

struct ci {   /* Estructura para almacenar las cons. iniciales, */
     unsigned char     *uc;     /* cuando es de un sonido */
     ds                *dc;     /* cuando es de dos sonidos */
} ci;

main() /** Programa Principal **/
{
     unsigned char ii;

  inic();              /*  Inicializa variables */
  rev_ent();           /* Revisa la entrada */
  memo_ci();           / *  Carga en memoria los g. cons. que
                            se dan al inicio de palabra */
  abre();              /* Abre los archivos */
 do {                  /* Ciclo hasta encontrar el fin de archivo */
     memo();          /* Carga en memoria una frase */
     if (limarrent) {   /* Si se encontro alguna frase => continua */
          printf("\n...INFORMACION obtendida de la BASE de DATOS\n");
          /* Mientras no se encuentre el fin de la frase, continua */
          while (iarr < limarrent) {
```

```
            busca_inicio();         /* Busca un inicio de palabra */
                /* Se calcula el no. de sonidos por comparar */
            nmspc=limarrent-iarr;
            consulta(&arrent[iarr],0);/* Consulta la base de datos */
            ++iarr;/* Incrementa el indice del arreglo de entrada */
        }
                /*  Después de haber realizado todas las consultas,
                    se puede solicitar la información obtenida */
        do {    /* Ciclo hasta detectar la opción "c" o "s" */
            printf(" OPCIONES\n\n");
            printf("1) Siguiente oración");
            printf(" 2) Cambio de frase\n");
            printf("3) Traer la sig. categoría");
            printf(" 4) Traer la categoría anterior\n");
            printf("5) Todas las oraciones\n");
            printf("\nc) continuar");
            printf(" s) salir\n");
            printf(" =>\n");
            scanf("%c",&opciRep); /* Lee la opción */
            fflush(stdin);
                /* Si no es ni "c" o "s", se reporta los resultados */
            if ((opciRep!='c')&&(opciRep!='s')) reporta();
        } while((opciRep!='c')&&(opciRep!='s'));
                /* Si se pide continuar, inicializa */
        if (opciRep=='c') {iarr=nollamaRepxf=0;}
        else goto e1;/* en caso contario, termina */
    }
  }while(!banfin);
e1:
  termina();
}

inic()              /* Rutina que inicializa algunas variables */
{
   i_r=i_vb=i_tc=i_at=i_s=i_err=0;
  arrent=0;
  nmspc=banfin=noferr=nollamaRepxf=0;
}

memo_ci()       / *  Rutina que carga en memoria los grupos
                     consonánticos que se presentan al inicio
                     de palabra    * /
```

```
{
    unsigned char     ii;
    int               aux;

 a_ci=fopen("inic.1c","r");/* Abre el que cont. las de 1 sonido */
 if (!a_ci) {                  /* Si hay problema, aborta */
    printf("ERROR, problemas al abrir\n");
    exit(-1);
 }
 fscanf(a_ci,"%d",&aux);   /* Lee el número de grupos */
 noci1s=aux;
                           /* Pide memoria e inicializa a ceros */
 ci.uc= (unsigned char *)calloc((unsigned int)noci1s, (unsigned int)
                           sizeof(char));
                           /* Lee la información */
 for (ii=0;ii < noci1s;++ii) fscanf(a_ci,"%*c%c",ci.uc+ii);
 fclose(a_ci);             /* Cierra el archivo */
 a_ci=fopen("inic.2c","r"); /* Abre el que cont. las de 2 sonidos */
 if (!a_ci) {              /* Si hay problema, aborta */
    printf("ERROR, problemas al abrir\n");
    exit(-1);
 }
 fscanf(a_ci,"%d",&aux);        /* Lee el número de grupos */
 noci2s=aux;
                           /* Pide memoria e inicializa a ceros */
 ci.dc= (ds *)calloc((unsigned int)noci2s, (unsigned int)sizeof(ds));
                           /* Lee la información */
 for (ii=0;ii < noci2s;++ii) fscanf(a_ci,"%*c%s",ci.dc+ii);
 fclose(a_ci);                  /* Cierra el archivo */
}

abre()   / *  Rutina que abre el archivo de entrada, la base de datos
              y su archivos auxiliares. Además, se limpia la salida */
{
 a_r=fopen("Revlex.rev","r");/* Abre archivos */
 a_vb=fopen("VocBas","r");
 a_tc=fopen("Tabla.cub","r");
 a_at=fopen("Aux.tab","r");
 a_s=fopen("Revlex.sal","w");/* Revlex.sal es limpiado */
 fclose(a_s);
 if ( (!a_r)||(!a_vb)||(!a_tc)||(!a_at)) {/* Si hubo problema al abrir, */
```

```
    printf("ERROR, problemas al abrir\n");  /* aborta la execución */
    exit(-1);
 }
}

memo()      /* Rutina que pide memoria, y carga la frase en ésta */
{
    unsigned char    ii,car;

 fseek(a_r,i_r,0);          /* Reposiciona el apuntador a la entrada */
 ii=limarrent=iarr=0;       /*  Inicializa  */
do {                        /* Ciclo hasta detectar pausa o EOF, */
    car=getc(a_r);          /* y para contar el número de caracteres */
    if (car=='£') {/* Si se detecta una frase con error, continua */
                            /* Lee hasta llegar a pausa o EOF */
        do car=getc(a_r);while((car!=' ')&&(car!=EOF));
        car=getc(a_r);   /* Lee el sig. caracter */
        ii=0;    /* e inicializa a cero el contador de caracteres */
    }
    ++ii;                   /* Incrementa el contador */
 }while((car!=' ')&&(car!=EOF));
    /* Valida que la long. de la frase sea a lo más de long. = fin */
 if (ii >= fin) {           /* Si es mayor, para la execución */
    printf("Frase demasiado larga\n");
    exit(-1);
 }
 if (car==EOF) {   /* Si se detectó EOF, enciende bandera de fin, y */
    banfin=1;
    if (ii==1) {            /* si no hay ningún caracter pendiente, */
        lib_ci();     /* libera la memoria ocupada por los g. cons., */
        cierra();     /* cierra los archivos, y para la execución */
        exit(0);
    }
 }
  if (arrent!=0)    {    /* Si ya fueron creados una vez, continua */
    if (ii > tamarrent) /* Si la frase a cargar es más larga que las
                           anteriores, */
        tamarrent=ii;/*  solicita más meomoria e inicializa en cero,
                         y actualiza las variables tamarrent y
                         limarrent    * /
        limarrent=ii-1;
        arrent=(char *)realloc(arrent,(unsigned int)tamarrent);
```

```
        memset(arrent,0,tamarrent);
        arrpalenc=(palenc *)realloc(arrpalenc,(unsigned int)
                    (tamarrent*sizeof(palenc)) );
        memset(arrpalenc,0,(int) tamarrent*sizeof(palenc));
        preg= (unsigned char *)realloc(preg,(unsigned int)tamarrent);
        memset(preg,0,tamarrent);
        ppal= (unsigned char *)realloc(ppal,(unsigned int)tamarrent);
        memset(ppal,0,tamarrent);
        pcam= (unsigned char *)realloc(pcam,(unsigned int)
                tamarrent);
        memset(pcam,0,tamarrent);
        pimpp= (unsigned char *)realloc(pimpp,(unsigned int)
                tamarrent);
        memset(pimpp,0,tamarrent);
        prepi= (unsigned char *)realloc(prepi,(unsigned int)
                tamarrent);
        memset(prepi,0,tamarrent);
        prepf= (unsigned char *)realloc(prepf,(unsigned int)
                tamarrent);
        memset(prepf,0,tamarrent);
    }
    else {    /* Cuando la frase es más corta que las anteriores */
              /* actualiza limarrent, e inicializa el espacio */
        limarrent=ii-1;/* requerido en cero */
        memset(arrent,0,ii);
        memset(preg,0,ii);
        memset(ppal,0,ii);
        memset(pcam,0,ii);
        memset(pimpp,0,ii);
        memset(prepi,0,ii);
        memset(prepf,0,ii);
        memset(arrpalenc,0,ii*sizeof(palenc));
    }
}
else {                  /* Cuando es la primera vez, se */
    tamarrent=ii;       /* define el tamaño del arreglo de entrada, */

    limarrent=ii-1;  /* su long. actual, y se pide memoria */
    arrent= (char *)calloc((unsigned int)tamarrent, (unsigned int)
            sizeof(char));
    arrpalenc= (palenc *)calloc((unsigned int)tamarrent,(unsigned int)
            sizeof(palenc));
```

```
    preg= (unsigned char *)calloc((unsigned int)tamarrent, (unsigned
            int)sizeof(char));
    ppal= (unsigned char *)calloc((unsigned int)tamarrent, (unsigned
            int)sizeof(char));
    pcam= (unsigned char *)calloc((unsigned int)tamarrent, (unsigned
            int)sizeof(char));
    pimpp= (unsigned char *)calloc((unsigned int)tamarrent, (unsigned
            int)sizeof(char));
    prepi= (unsigned char *)calloc((unsigned int)tamarrent, (unsigned
            int)sizeof(char));
    prepf= (unsigned char *)calloc((unsigned int)tamarrent, (unsigned
            int)sizeof(char));
}
        /* Si hay algún error, para la execución */
 if
((!arrent)||(!arrpalenc)||(!preg)||(!ppal)||(!pcam)||(!pimpp)||(!prepi)||(!prepf))
{
    printf("ERROR, falta memoria\n");
    exit(-1);
}
 fseek(a_r,i_r,0);      /* Reposiciona el apuntador a la entrada */
 i_r+=ii;               /* Actualiza el indice a Revlex.rev */
 for(ii=0;ii<limarrent;++ii)  arrent[ii]=getc(a_r); /* Lee a memoria */
 printf("\ni_r= %d, arrent= %s, limarrent= %d\n",i_r,arrent,limarrent);
}

busca_inicio()/* Rutina que localiza un posible inicio de palabra */
{

    unsigned char ii,banini,banvoc,nsc,ban;

do {                        /* Ciclo hasta encontar un inicio */
    if (iarr!=limarrent-1) {    /* Si no se ha llegado al limite , */
        switch (arrent[iarr]) {/* pregunta por arrent[iarr] */
            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u': banini=1;break;/* Si es vocal => es inicio */
            default: banini=0;
        }
    }
```

```
    else banini=1;
    if (!banini) {      /* Si es consonante, continua */
                        /* Calcula el no. de sonidos consonánticos */
        ii=1;banvoc=0;  /* Inicializa */
        do {            /* Ciclo hasta encontrar una vocal */
            switch (arrent[iarr+ii]) {
                case 'a':
                case 'e':
                case 'i':
                case 'o':
                case 'u': banvoc=1;
            }
            ++ii;
        }while((!banvoc)&&(ii<limarrent-iarr));
        nsc=ii-1;
        if (nsc==1) { /* Si es sólo de un sonido, continua */
                      /* Compara contra todas las válidas */
            ii=ban=0;
            do { /* Ciclo hasta preg. por todas o encontrar alguna */
                if (arrent[iarr]==ci.uc[ii]) ban=1;
                ++ii;
            }while((!ban)&&(ii<noci1s));
                /* Si se encontró => enciende la bandera de inicio */
            if (ban) banini=1;
        }
        else if (nsc==2) {      /* Si son dos sonidos, continua */
            ii=0;
            do {/* Ciclo hasta preg. por todas o encontrar alguna */
                ban=memcmp(&arrent[iarr],ci.dc[ii],2);
                ++ii;
            }while((ban)&&(ii<noci2s));
                /* Si se encontró => enciende la bandera de inicio */
            if (!ban) banini=1;
        }
                /* Si no se encontró => preg. por la siguiente */
        if (!banini) ++iarr;
    }
 }while (!banini);
}

termina()       /* Rutina que libera memoria y cierrra archivos */
{
```

```
  lib_ci();              /* Libera la memoria ocupada por los g. cons. */
  free(arrent);          /* LIbera la memoria ocupada por los dos  */
  free(arrpalenc);       /* arreglos, y las pilas */
  free(preg);
  free(ppal);
  free(pcam);
  free(pimpp);
  free(prepi);
  free(prepf);
  cierra();              /* Cierra los archivos */
}

lib_ci()               /* Rutina que la memoria ocupada por los g. cons. */
{
  free(ci.uc);
  free(ci.dc);
}

cierra()               /* Rutina que cierrra los archivos */
{
  fclose(a_r);
  fclose(a_vb);
  fclose(a_tc);
  fclose(a_at);
}
```

B.4 Código del programa Red.c

```
/ *  Mediante este programa se hace el cambio de nomenclatura,
     y se realiza una revisión sobre las consonantes para
     detectar fines de palabra y comb. de cons. inválidas  * /

#include <TiposyCtes.h>
#include <VGlobExt.h>

unsigned char     *aem,indic[fin];
/ *  aem= apuntador al archivo de entrada cargado en memoria
     indic= arreglo de indicadores de pausa */
long          i_e,inicfrase,termifrase,pricons,ultcons,tam_a_e;
FILE          *a_red1,*a_red2;
red           red1,red2;
int           fildes,error;

/* Declaración de procedimientos presentes en el archivo */

int  lee_red(),ent_red(),evalua();
int  rep_red(),rep_frase(),ter_red();

rev_ent()/* Rutina principal que llama a los demás procedimientos */
{
     FILE     *a_e;
     char     c1,c2,cimp,ban;

 a_e=fopen("Revlex.ent","r");/* Abre el archivo de entrada */
 if (!a_e) {                    /* Si hay problemas, termina ejecución */
     printf("ERROR, problemas al abrir\n");
     exit(-1);
 }
 fseek(a_e,0,2);              /* Se obtiene el tamaño del archivo */
  tam_a_e=ftell(a_e);
 rewind(a_e);
              /* Se pide area para cargar el archivo en memoria */
 aem=(char *)calloc((unsigned int)tam_a_e,(unsigned int)
sizeof(char));
 if (!aem) {                    /* Si hay problemas, termina ejecución */
```

```
    printf("ERROR, problemas al tratar de cargar Revlex.ent
    en memoria\n");
    exit(-1);
}
                        /* Abre los archivos de salida */
fildes=open("Revlex.rev",O_WRONLY|O_CREAT|O_TRUNC);
 a_err=fopen("Revlex.err","w");
    /* Se realiza el cambio de nomenclatura  y se almacena en
memoria */
 c1=getc(a_e);              /* Toma los 1os dos sonidos */
 c2=getc(a_e);
 i_e=0;
 if ((c1!=EOF)&&(c2!=EOF)) {/* Si no detecta fin de archivo, continua */
    do {                        /* Ciclo hasta EOF */
                /* Cambio de nomenclatura /ch/->/c/ y /rr/->/-/ */
        if (c1=='c') {          /* Si es "c" imprime sólo "c" */
            cimp=c1;            /* y lee otros dos */
            c1=getc(a_e);
            c2=getc(a_e);
        }
        else if ((c1=='r')&&(c2=='r')) {
            cimp='-';           /* Si es "rr" imprime "-" */
            c1=getc(a_e);       /* y lee otros dos */
            c2=getc(a_e);
        }
        else {                  /* Si no imprime "r" */
            cimp=c1;            /* y lee uno */
            c1=c2;
            c2=getc(a_e);
        }
                /* El caracter adecuado es escrito en memoria */
        sprintf(&aem[i_e],"%c",cimp);
        ++i_e;                  /* Actualiza el indice */
    }while((c1!=EOF)&&(c2!=EOF));
}
if (c1!=EOF) {                      /* Si c2 fue EOF, */
    sprintf(&aem[i_e],"%c",c1);     /* escribe c1 en memoria */
}
    /* Se obtiene el valor real del espacio que ocupa en memoria */
 tam_a_e=i_e+1;
 fclose(a_e);                       /* Cierra entrada */
                        /* Lee la red 1, cargandola en memoria */
```

```
lee_red("red1.net",a_red1,&red1);
                        /* Lee la red 2, cargandola en memoria */
lee_red("red2.net",a_red2,&red2);
i_e=0;
do {    /* Ciclo hasta que exceda el tamaño, es decir hasta EOF */
    if (aem[i_e]==' ') ++i_e;    /* Si es pausa incrementar el indice */
    inicfrase=i_e;    /* Actualiza el apuntador al inicio de frase */
    do { /* Ciclo hasta que encuentre pausa o se exceda del tamaño */
        switch (aem[i_e]) {          /* Pregunta por aem[i_e] */
            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u':ban=1;    /* Si es vocal enciende la bandera */
                     break;
            default: ban=0;
        }
        if (!ban) pricons=i_e-inicfrase;
            /* Si es cons. ya la encontramos */
            /* pricons está dada con respecto al inicio de la frase */
        else {              /* si no hay que buscarla */
                            /* Mientras sea vocal, incrementa indice */
            while ((aem[i_e]=='a')||(aem[i_e]=='e')||(aem[i_e]=='i')
                ||(aem[i_e]=='o')||(aem[i_e]=='u'))
                ++i_e;
/* Si no es pausa => encontramos la 1ª cons., y si no inicializamos */
            if ((aem[i_e]!=' ')&&(i_e<tam_a_e)) pricons=i_einicfrase;
            else pricons=ultcons=0;
        }
            /* Si no llegamos a pausa, buscamos la última cons. */
        if ((aem[i_e]!=' ')&&(i_e<tam_a_e)) {
            ++i_e;
                /* Mientras sea cons., incrementa indice */
            while( (aem[i_e]!='a')&&(aem[i_e]!='e')&&(aem[i_e]!='i')
                &&(aem[i_e]!='o')&&(aem[i_e]!='u')&&(aem[i_e]!=' ')
                &&(i_e <tam_a_e))
                ++i_e;
                /* Si no nos hemos pasado del límite, continua */
            if (i_e < tam_a_e) {
                /* calcula la posición de la ultima cons. */
                ultcons=i_e-inicfrase-1;
                error=0; /* Inicializa */
```

```
                    /* Si fueron dos cons. */
                    if (ultcons-pricons+1 == 2) {
                    /* Se da la entrada a la red 1 */
                        error=ent_red(&red1);
                    /* Si no hay alguna inconsistencia, cont. */
                        if (error!=-1) {
                    /* Obtiene la salida de la red 1 */
                            evalua(&red1);
                    /* Reporta el resultado de la red 1 */
                            rep_red(1,&red1);
                        }
                    /* Si hubo error reporta */
                        else rep_red(1,&red1);
                    }
                    /* Si fueron más de dos cons. */
                    else if (ultcons-pricons+1 > 2) {
                    /* Se da la entrada a la red 2 */
                        error=ent_red(&red2);
                    /* Si no hay alguna inconsistencia, cont. */
                        if (error!=-1) {
                    /* Obtiene la salida de la red 2 */
                            evalua(&red2);
                    /* Reporta el resultado de la red 2 */
                            rep_red(2,&red2);
                    /* Si hubo error reporta */
                        } else rep_red(2,&red2);
                    }
                }
            }
        } while ((aem[i_e]!=' ')&&(i_e < tam_a_e));
                    /* Guarda el índice de la terminación de la frase */
        termifrase=i_e;
        rep_frase(); /* Reporta la frase total */
    } while(i_e < tam_a_e);
    ter_red();          /* Termina */
}

lee_red(nom,a_red,redx)          / * Rutina que lee la red y la carga
                                    en memoria */
char    *nom;
FILE        *a_red;
red     *redx;
```

```
{
unsigned int nres,unirec,nurec,unides,nudes,ini1,lim1,ini2,lim2;
unsigned char    ii,jj,kk,ban;
int          err,cmp1,cmp2;
char         buff[15],car;
float        valor;
restric      *res;
long         dir;

 a_red=fopen(nom,"r");/* Abre el archivo */
        /* Mientras no se encuentre el fin de archivo, continua */
 while( (err=fscanf(a_red,"%s",buff))!=EOF ) {
        /* Pregunta si es la sección de definición */
    cmp1=strcmp(buff,defi);
    if (!cmp1) {            /* Si es, continua */
        do {                /* Ciclo hasta fin de sección */
            fscanf(a_red,"%s",buff);   /* Lee el 1er string */
            cmp1=strcmp(buff,end);/* Preg. si se detectó el fin */
            if (cmp1) {        /* Si no hemos llegado, continua */
                /* Preg. si viene el no. de unidades */
                cmp2=strcmp(buff,nuni);
                /* Si => lee */
                if (!cmp2) {
                    fscanf(a_red,"%u",&(redx->nunid));
                    goto e2;
                }
                /* Preg. si viene el no. de unidades de entrada */
                cmp2=strcmp(buff,ninp);
                /* Si => lee */
                if (!cmp2) {
                    fscanf(a_red,"%u",&(redx->nuent));
                    goto e2;
                }
                /* Preg. si viene el no. de unidades de salida */
                cmp2=strcmp(buff,nout);
                /* Si => lee */
                if (!cmp2) fscanf(a_red,"%u",&(redx->nusal));
e2:         }
        }while (cmp1);
                /* Se pide memoria */
        redx->entext= (float*) calloc(redx->nuent,(unsigned int)
                        sizeof(float) );
```

```
    redx->entnet= (float*) calloc(redx->nunid,(unsigned int)
                    sizeof(float) );
    redx->polar= (float*) calloc(redx->nunid,(unsigned int)
                  sizeof(float) );
    redx->activ= (float*) calloc(redx->nunid,(unsigned int)
                  sizeof(float) );
    redx->pesos= (float*) calloc((unsigned int) (redx->
            nunid*redx->nunid),(unsigned int) sizeof(float) );
    if ((!redx->polar)||(!redx->entext)||(!redx->entnet)||(!redx->
        activ)||(!redx->pesos)) {
        printf("ERROR, problemas al cargar las redes\n");
        exit (-1);     /* Si hay problema, aborta */
    }
    goto e1;
}
            /* Pregunta si es la sección de restricciones */
cmp1=strcmp(buff,constr);
if (!cmp1) {
    dir=ftell(a_red);/* Se almacena la posición actual */
    ii=0;          /* Cuenta el no. de restricciones */
    do {
        fscanf(a_red,"%s",buff);
        cmp2=strcmp(buff,end);
        if (cmp2) ++ii;
    }while(cmp2);
    nres=ii/2;
                          /* Se pide memoria */
    res=(restric *)calloc( (unsigned int)nres,(unsigned int)
            sizeof(restric) );
    fseek(a_red,dir,0);          /* Se reposiciona */
    for (ii=0;ii < nres;++ii) {
        do car=getc(a_red);while(isspace(car));
        res[ii].nom=car;  /* Lee el nombre */
        fscanf(a_red,"%f",&res[ii].val); /* Lee el valor */
    }
    fscanf(a_red,"%s",buff);   /* Elimina el fin */
    goto e1;
}
cmp1=strcmp(buff,net);    /* Pregunta si es la sección de red */
if (!cmp1) {
    dir=ftell(a_red);        /* Se almacena la posición actual */
    do { /* Ciclo hasta terminar de leer todo lo de la seccion */
```

```
fscanf(a_red,"%s",buff);   /* Lee el 1er string */
cmp1=strcmp(buff,end);   /* Preg. por fin */
if (cmp1) {   /* Si no ha llegado, continua */
        /* Si vienen los pesos como reales, los lee */
    if (buff[0]=='&') {
        for (ii=0;ii < redx->nunid;++ii)
            for (jj=0;jj < redx->nunid;++jj) {
                fscanf(a_red,"%f",&redx->pesos
                    [ii*redx->nunid+jj]);
            }
    }
    else {        /* Si no, continua */
        ban=valor=0;
        if (buff[0]=='%') {/* Si viene un bloque, sig. */
                /* ¿Se indica alguna restricción? */
            if (strlen(buff) >1) {
                /* Busca el valor de la restricción */
                car=buff[1];
                ii=0;/* Si no la encuentra, valor = 0 */
                do {
                    if (car==res[ii].nom) {
                        ban=1;
                        valor=res[ii].val;
                    }
                    ++ii;
                }while((!ban)&&(ii < nres));
            }
                /* Lee reg., # reg., col. y # de col. */
            fscanf(a_red,"%u%u%u%u",&unirec,&nurec,
            &unides,&nudes);
            /* Guarda la posición actual */
            dir=ftell(a_red);
            ini1=unirec;
            lim1=unirec+nurec;
            ini2=unides;
            lim2=unides+nudes;
        }
        else {   /* Si es que no, espera todos */
            ini1=ini2=0;
            lim1=lim2=redx->nunid;
        }
        /* Reposiciona el apuntador */
```

```
                    fseek(a_red,dir+1,0);
                    /* Lee los valores */
                    for (ii=ini1;ii < lim1;++ii) {
                        for (jj=ini2;jj < lim2;++jj) {
                            if (ban) /* Si se indicó restricción */
                                redx->pesos[ii*redx->nunid+jj]
                                =valor;
                            else {   /* Si no se indicó */
                                car=getc(a_red);
                                    /* Si encuentra "." asgina 0 */
                                if (car=='.')
                                    redx->pesos[ii*redx->
                                    nunid+jj]=0;
                                    /* En caso contario */
                                else {
                            /* busca el carac. en las restricciones */
                                    valor=kk=0;
                                    do {
                                        if (car==res[kk].nom) {
                                            valor=res[kk].val;
                                            break;
                                        }
                                        ++kk;
                                    }while(kk < nres);
                                    redx->pesos[ii*redx->
                                    nunid+jj]=
                                    valor;
                                }
                            }
                        }
                        /* Lee el fin de linea */
                        if (!ban) getc(a_red);
                    }
                }
            }
        }while(cmp1);
        goto e1;
    }
    cmp1=strcmp(buff,bias);/* Pregunta si es la sección de bias */
    if (!cmp1) {        /* Si es, se sig. un proc. similar al de red */
        dir=ftell(a_red);
        do {
```

```
        fscanf(a_red,"%s",buff);
        cmp1=strcmp(buff,end);
        if (cmp1) {
            ban=valor=0;
            if (buff[0]=='%') {
                if (strlen(buff) >1) {
                    car=buff[1];
                    ii=0;
                    do {
                        if (car==res[ii].nom) {
                            ban=1;
                            valor=res[ii].val;
                        }
                        ++ii;
                    }while((!ban)&&(ii < nres));
                }
                fscanf(a_red,"%u%u",&unirec,&nurec);
                dir=ftell(a_red);
                ini1=unirec;lim1=unirec+nurec;
            } else {ini1=0;lim1=redx->nunid;}
            fseek(a_red,dir+1,0);
            for (ii=ini1;ii < lim1;++ii) {
                if (ban) redx->polar[ii]=valor;
                else {
                    car=getc(a_red);
                    if (car=='.') redx->polar[ii]=0;
                    /* si no existe entonces pone cero */
                    else {
                        valor=jj=0;
                        do {
                            if (car==res[jj].nom) {
                                valor=res[jj].val;
                                break;
                            }
                            ++jj;
                        }while(jj < nres);
                        redx->polar[ii]=valor;
                    }
                }
            }
        }
    }while(cmp1);
```

```
            goto e1;
        }
        cmp1=strcmp(buff,end);
        if (!cmp1) goto e1;
e1:
    }
    free(res);
    fclose(a_red);
}

ent_red(redx)     /* Rutina que habilita las unidades de entrada */
red *redx;              /*  de la red a su valor indicado */
{
        unsigned char     nopos,inic,ii;
        long              jj;

        /* Calcula el # de posiciones */
    nopos= (unsigned char) (redx->nuent/nocons);
        /* Si el # de cons. es > que éste regresa error */
    if (ultcons-pricons+1 > nopos)   return(-1);
    inic=0;
        /* Ciclo desde la 1ª a la última cons. */
    for (ii=0,jj=pricons+inicfrase;ii < nopos;++ii,++jj) {
          switch(aem[jj]) {      /* Preg. por aem[jj] */
              case 'b':redx->entext[inic+0]=1;break;
              case 'c':redx->entext[inic+1]=1;break;
              case 'd':redx->entext[inic+2]=1;break;
              case 'f':redx->entext[inic+3]=1;break;
              case 'g':redx->entext[inic+4]=1;break;
              case 'k':redx->entext[inic+5]=1;break;
              case 'l':redx->entext[inic+6]=1;break;
              case 'm':redx->entext[inic+7]=1;break;
              case 'n':redx->entext[inic+8]=1;break;
              case 'ñ':redx->entext[inic+9]=1;break;
              case 'p':redx->entext[inic+10]=1;break;
              case 'r':redx->entext[inic+11]=1;break;
              case '-':redx->entext[inic+12]=1;break;
              case 's':redx->entext[inic+13]=1;break;
              case 't':redx->entext[inic+14]=1;break;
              case 'w':redx->entext[inic+15]=1;break;
              case 'x':redx->entext[inic+16]=1;break;
              case 'y':redx->entext[inic+17]=1;break;
```

```
    }
    inic+=nocons;          /* incrementa la referencia */
 }
 return(0);              /* Regresa sin error */
}

evalua(redx)
    /* Rutina que obtiene los niveles de activación de la red */
red *redx;
{
    unsigned int ii,jj;

    /* Calcula la actividad de las unidades de entrada */
  for (ii=0;ii<redx->nuent;++ii) redx->activ[ii]=redx->entext[ii];
    / * Ciclo apartir de la 1ª unidad oculta hasta la
         última unidad de salida */
  for (ii=redx->nuent;ii<redx->nunid;++ii) {
    /* Entrada neta inicializada con el valor de polarización */
    redx->entnet[ii]=redx->polar[ii];
    for (jj=0;jj<redx->nunid;++jj)
    /* Cálculo de la entrada neta para la unidad ii */
        if ((redx->activ[jj])&&(redx->pesos[ii * redx->nunid+jj])) {
            redx->entnet[ii]=redx->entnet[ii]+(float)(redx->activ[jj]
                *redx->pesos[ii * redx->nunid+jj]);
        }
    /* Cálculo del nivel de activación para la unidad ii */
    redx->activ[ii]=1/(1+ (float)exp((extended) -redx->entnet[ii]));
 }
}

rep_red(opci,redx)
    /* Rutina que enciende los indicadores para la frase */
unsigned char     opci;/* de acuerdo a los valores de
                          las salidas de la red */
red               *redx;
{

 if (redx->activ[redx->nunid - redx->nusal+2]>0.9)
    error=-1;/*Si S3 está encendida enciende la bandera de error */
 if (error==-1)
    indic[fin-1]= -1;/* Si hay error enciende el último indicador */
 else if (redx->activ[redx->nunid - redx->nusal+1]>0.9)
```

```
{   /* Si no y S2 está encendida */
    if (opci==1)            /* y si es la red de 2 posiciones */
        indic[pricons]=1; /* => hay división entre las cons. */
    else {                  /* Si no, continua */
                            /* Si F1 está encendida */
        if (redx->activ[redx->nunid - redx->nusal+3]>0.9)
            indic[pricons]=1;/* hay división entre la 1ª y la 2ª */
                            /* Si F2 está encendida */
        else if (redx->activ[redx->nunid - redx->nusal+4]>0.9)
            indic[pricons+1]=1;/* hay división entre la 2ª y la 3ª */
    }
}
    /* Inicializa en cero las entradas externas y las activaciones */
  memset(redx->entext,0,(int)(redx->nuent*sizeof(float)));
  memset(redx->activ,0,(int)(redx->nunid*sizeof(float)));
}

rep_frase()       /* Rutina que imprime la nueva frase en la salida */
{
    long     ii,jj;

    printf("Frase  encontrada\n");
    if  (indic[fin-1]==-1)  { /* Si hay error */
    /* Escribe el indicador de error y la frase tal cual */
    write(fildes,"£",1);
    write(fildes,&aem[inicfrase],(unsigned)  (termifrase-inicfrase
            +1));
    printf("£");
    for  (ii=inicfrase;ii  <  termifrase;++ii)  printf("%c",aem[ii]);
        / *  Reporta en el archivo de errores la posición
            donde inicia la frase con error */
    fprintf(a_err,"%ld\n",inicfrase);
}
 else {   /* Si no hay error, continua */
    /* Ciclo para imprimir la frase */
    for  (ii=inicfrase,jj=0;ii  <  termifrase;++ii,++jj)  {
        write(fildes,&aem[ii],1);/* Imprime  un  caracter  */
        printf("%c",aem[ii]);
        if  (indic[jj]==1)  {      /* Preg. si le sigue pausa */
            write(fildes,"  ",1);    /* Si es sí, => escribe pausa */
            printf("  ");
        }
```

```
    }
    if (termifrase!=tam_a_e) {  /* y termina con pausa la frase, al */
        write(fildes,"  ",1);    /* menos de que haya llegado al fin */
        printf("  ");
    }
 }
  memset(indic,0,fin);       /* Inicializa con cero los inidcadores */
  printf("\n");
}

ter_red()          /* Rutina que libera memoria y cierra archivos */
{
  free(red1.entext);
  free(red1.entnet);
  free(red1.polar);
  free(red1.activ);
  free(red1.pesos);
  free(red2.entext);
  free(red2.entnet);
  free(red2.polar);
  free(red2.activ);
  free(red2.pesos);
  close(fildes);
  fclose(a_err);
}
```

B.5 Código del programa Consulta.c

```
/ *  Mediante este programa se obtienen todas
     las palabras contenidas a partir de la
     posición indicada en la frase, reportandose
     por medio del arrpalenc    * /


#include <TiposyCtes.h>
#include <VGlobExt.h>

     unsigned char     ii,jj,kk,noson,nopal,*a_cub;
     palabra           palaux,transcripcion;
     raiz              conv,rai;
     long              tamcub;

long calc_hx();   /* Declara el procedimiento presente */

consulta(ent,opci)
     /* Rutina principal que llama a los demás procedimientos */
unsigned char     *ent,opci;
{
     unsigned char     c1,c2,limite,min,clave,tam;
     unsigned char     nocar,nocarr,cons,ban;
     unsigned char     bfinreg,todas,dimarr;
     int               fildes,compara;
     long              i_vb_r,prox,i_arr,i_cub,i_actual;
     FILE              *arr,*arrdir;
     char              nom[20];

 if (opci) {         /* Si fue llamado en la fase de reporte, continua */
     fseek(a_vb,i_vb,0);    /* Se posiciona al inicio de la cubeta */
     if (opci==1) {         /* Cuando se pide la sig. categoría */
                            /* Próximo es realmente el próx. */
          fscanf(a_vb,"%*c%*c%*x%*c%x",&prox);
          i_vb+=prox;  /* y actualiza el indice dentro de la cubeta */
     }
     else {                 /* Si no, => se busca la cat. anterior */
          clave=getc(a_vb);      /* ¿Estamos dentro de la cubeta? */
          if (clave!=conti) /* Si es que no => Reporta que no hay más */
```

```
            return(no_otra_cat);
        else {              /* Si es que sí, continua */
                            /* Próximo es realmente el anterior */
            fscanf(a_vb,"%*c%x%*c%*x",&prox);
                    /* y actualiza el indice dentro de la cubeta */
            i_vb-=prox;
        }
    }
    fseek(a_vb,i_vb,0);         /* Apunta al nuevo registro */
    clave=getc(a_vb);           /* ¿Estamos dentro de la cubeta? */
                            /* Si es que no y se pidió la próxima => */
    if ((opci==1)&&(clave!=conti))
        return(no_otra_cat); /* reporta que no hay más */
}
                /* El limite a lo más toma el valor de tamrai */
limite= (nmspc<tamrai) ? nmspc:tamrai;
 if (opci) {        /* Si fue llamado en la fase de reporte, continua */
    noson= limite;    /* noson=limite => sólo entra al while 1 vez */
    goto e1;          /* Salta a la etiqueta uno */
}
 else noson=0;                  /* Si no, inicializa con 0 */
 nopal=0;                       /* El contador de palabras se inicializa
con 0 */
 while (noson < limite)         /* Ciclo hasta llegar al límite */
 {
    ++noson;                    /* Incrementa */
            /* Cuando es un sonido no es necesario ir a calc_hx() */
    if (noson==1) {
        compara=1;
        /* Las vocales son las únicas que pueden estar aisladas */
        switch (ent[0]) {
            case 'a':conv[0]=a;break;
            case 'e':conv[0]=e;break;
            case 'i':conv[0]=i;break;
            case 'o':conv[0]=o;break;
            case 'u':conv[0]=u;break;
            default: c1=0;c2=0;compara=0;
        }
        if (compara) {/* Si fue vocal lee de la Tabla de cubetas */
            i_tc=conv[0];
            fseek(a_tc,(long)  (i_tc*nocartc),0);
            fscanf(a_tc,"%c%c",&c1,&c2);
```

```
        }
    }
    else {              /* Cuando es más de un sonido */
        i_tc=calc_hx(ent);    /* Calcula  h(x) */
        fseek(a_tc,(long)  (i_tc*nocartc),0);
        fscanf(a_tc,"%c%c",&c1,&c2);/* Lee de la Tabla de cubetas */
    }
    if (c1||c2) {/* Si la dirección obt. de la Tabla es dif. de cero */
        i_at=256*c1+c2-1;
        fseek(a_at,(long)  (i_at*nocarat),0);
                / *  Lee del archivo auxiliar a la Tabla, el apuntador
                     a la cubeta y el tamaño de ésta */
        fscanf(a_at,"%x%*c%x",&i_vb,&tamcub);
        if (i_vb != EOF) {/* Si se obtiene una dir. válida, continua */
            fseek(a_vb,i_vb,0);
            clave=getc(a_vb);
e1:         if (clave==arreg) {     /* Procedimiento si hay arreglo */
                /* Lee la dim. del arreglo */
                fscanf(a_vb,"%c",&dimarr);
                memset(nom,0,20);
                strcpy(nom,rai);
                strcat(nom,".arr");
                arr=fopen(nom,"r");             /* Abre _.arr */
                strcat(nom,".dir");
                arrdir=fopen(nom,"r");          /* Abre _.arr.dir */
                i_arr=0;          /* Calcula el indice al arreglo */
                for(ii=0,jj=dimarr-1;ii  <  dimarr;++ii,--jj)  {
                    i_arr+=  (conv[tamrai+ii])*ipower((extended)
                        base,jj);
                }
                fseek(arr,i_arr,0);
                fscanf(arr,"%c",&c1);       /* Lee del .arr */
                if (c1) {               /* Si la dir. es dif. de cero */
                    --c1;
                    fseek(arrdir,(long)(c1*nocaraa),0);
                    /* Lee del .arr.dir */
                    fscanf(arrdir,"%x",&i_vb_r);
                    / *  Si se busca la próx. o la anterior
                         dec. el tamaño de la cubeta */
                    if (!opci)
                        tamcub-=i_vb_r-i_vb;
                    /* Actualiza el indice en la cubeta */
```

```
                i_vb=i_vb_r;
                fseek(a_vb,i_vb+2,0); /*  Reposiciona  */
            }
            else  i_vb_r=i_vb;
        }
        else {              /* Si no hay arreglo */
            getc(a_vb);   /* Lee el 2º carac. escondido */
            i_vb_r=i_vb; /*  Actualiza  */
        }
            /* Cuando se analizan con 1,2 y 3 sonidos*/
        if (noson < tamrai) {
            /* Se llena el arreglo de palabras encontradas */
            arrpalenc[iarr].info[nopal].dir=i_vb;
                /* Actualiza prox adecuadamente */
            if (opci==1) fscanf(a_vb,"%x%*x",&prox);
            else  fscanf(a_vb,"%*x%x",&prox);
                /* Lee la transcripción */
    fscanf(a_vb,"%*c%*c%s/",arrpalenc[iarr].info[nopal].trans);
            compara=strlen(arrpalenc[iarr].info[nopal].trans);
                /* Elimina el "/" */
            arrpalenc[iarr].info[nopal].trans[compara-1]=0;
                /* Lee su escritura */
            fscanf(a_vb,"%*c"); /* se esquiva el tabulador */
            ii=0;
            do {
                arrpalenc[iarr].info[nopal].escri[ii]=getc(a_vb);
                ++ii;
            }while(arrpalenc[iarr].info[nopal].escri[ii-1]!='{');
                /* Elimina la "{" */
            arrpalenc[iarr].info[nopal].escri[ii-1]=0;
                /* Elimina la pausa */
            arrpalenc[iarr].info[nopal].escri[ii-2]=0;
                /* Lee la categoría */
            ii=0;
            do {
                arrpalenc[iarr].info[nopal].cat[ii]=getc(a_vb);
                ++ii;
            }while(arrpalenc[iarr].info[nopal].cat[ii-1]!=',');
                /* Elimina la "," */
            arrpalenc[iarr].info[nopal].cat[ii-1]=0;
                /* Lee la palabra origen */
            ii=0;
```

```
            do {
                arrpalenc[iarr].info[nopal].porig[ii]=getc(a_vb);
                ++ii;
            }while(arrpalenc[iarr].info[nopal].porig[ii-1]!='}');
                /* Elimina la "}" */
            arrpalenc[iarr].info[nopal].porig[ii-1]=0;
                /* Llena los campos de repson y sinrep */
            if (noson==1) arrpalenc[iarr].info[nopal].repson=0;
            else arrpalenc[iarr].info[nopal].repson=iarr+noson;
            if (iarr+noson+1>limarrent)
                    arrpalenc[iarr].info[nopal].sinrep=fin;
            else arrpalenc[iarr].info[nopal].sinrep=iarr+noson+1;
    /* Si fue llamada en la fase de consulta, inc. el # de pal. */
            if (!opci) ++nopal;
            else compara=0; /* Inidca que se encontró */
        }

                / *  Cuando se realizan raices de 4 sonidos,
                     y llama en la fase de consulta, */
                     se carga en memoria la cubeta */
        else if (!opci) {
                /* Se cierra la base, y se abre en otro modo */
            fclose(a_vb);
            fildes=open("VocBas",O_RDONLY);
                /* Pide memoria */
            a_cub= (unsigned char *)calloc((unsigned int)
                    tamcub,(unsigned int)sizeof(char));
                /* Si hay algún error, detiene la ejecución */
            if (!a_cub) {
                printf("ERROR. No hay espacio para cargar en
                    memoria\n");
                exit(-1);
            }
            lseek(fildes,i_vb_r,0);
                /* Se carga la cubeta */
            read(fildes,a_cub,(unsigned)tamcub);
            close(fildes);      /* Se cierra la base */
            prox=todas=i_vb=0;
            do { /* Ciclo hasta encontrar todas */
                i_cub=i_vb; /* Inicializa */
                sscanf(&a_cub[i_cub+11],"%x",&prox);
                bfinreg=0;
```

```
            memset(palaux,0,sizeof(palabra));
            memset(transcripcion,0,sizeof(palabra));
            ii=kk=0;i_cub+=21;
/* Lee hasta encontrar un "(" o un "/" (1ª seección) */
            do {
                c1=a_cub[i_cub++];
                if ((c1!='(')&&(c1!='/')) {
                    switch(c1) {
                        case 'á': c1='a';break;
                        case 'é': c1='e';break;
                        case 'í': c1='i';break;
                        case 'ó': c1='o';break;
                        case 'ú': c1='u';break;
                    }
                    /* En transc. se almacena tal cual */
                    transcripcion[kk]=c1;
                    if ((c1=='c')||(c1=='r')) {
                        c2=a_cub[i_cub++];
                        /* /ch/ -> /ch/ */
                        if (c2=='h') transcripcion[++kk]=c2;
                        else if (c2=='r') {
                            c1='-';
                            /* /rr/ -> /rr/ */
                            transcripcion[++kk]=c2;
                        }
                        else --i_cub;
                    }
            / *  En palaxu se almacena elim. acentos y
                cambiando nomenclatura */
                    palaux[ii]=c1;
                    ++ii;++kk;
                }
            }while((c1!='(')&&(c1!='/'));
            nocar=ii;jj=nocar;nocarr=kk;   /* Actualiza */
/* Mientras no haya localizado el fin de reg., conti. */
            while (bfinreg==0) {
                if (c1!='/') {  /* Si es dif. de "/" */
            /* Ciclo hasta detectar ".",",",")" (2ª sección) */
                    do {
                        c1=a_cub[i_cub++];
                        if ((c1!='.')&&(c1!=',')&&(c1!=')')) {
                            switch(c1) {
```

```
                        case 'á': c1='a';break;
                        case 'é': c1='e';break;
                        case 'í': c1='i';break;
                        case 'ó': c1='o';break;
                        case 'ú': c1='u';break;
                    }
                    transcripcion[kk++]=c1;
                    if ((c1=='c')||(c1=='r')) {
                        c2=a_cub[i_cub++];
                        if (c2=='h')
                        /* /ch/ -> /ch/ */
                        transcripcion[kk++]=c2;
                        else if (c2=='r') {
                            c1='-';
                            /* /rr/ -> /rr/ */
                        transcripcion[kk++]=c2;
                        }
                        else --i_cub;
                    }
                }
                /* Actualiza */
                /* se lee la "," (2 símbolos) */
                if (c1=='.') c1=a_cub[i_cub++];
                else if ((c1!=',')&&(c1!=')')) {
                    palaux[jj]=c1;
                    ++jj;
                }
            }while((c1!='.')&&(c1!=',')&&(c1!=')'));
            /* Si acabó con ")" indica fin de reg. */
            if (c1==')') bfinreg=1;
        }
        /* Si es "/" indica fin de reg. */
        else bfinreg=1;
    /*  Si la pal. en la cubeta es de long. menor que
        la frase a comparar => conti. */
        if (jj <= (limarrent-iarr+1)) {
            tam=strlen(palaux);
            c1=palaux[tam-1];
            c2=ent[jj-1];
            if ( ( ((c1=='r')&&(c2=='-')) || ((c1=='-')
                &&(c2=='r')) ) ) palaux[tam-1]=c2;
```

```
/ *  Compara la entrada con la palab.
     leida de la Base de Datos */
compara=memcmp((char *)ent,(char *)
          palaux,(int)jj);
/* Si es preg. si ya fue  registrado */
if (!compara) {
     ban=0;
     ii=0;
     / *  Ciclo hasta encontrar la
          1ª igual o recorrer todas */
     do {
compara=memcmp((char *)
transcripcion,(char  *)arrpalenc[iarr].
info[ii].trans,sizeof(palabra));
if (compara) ban=1;
else ban=0;
++ii;
     }while((ban)&&(ii<nopal));
          /* Si no encontró */
     if (ban) {
          /* Llena el arrpalenc */
          transcripcion[kk++]=0;
memcpy(arrpalenc[iarr].info[nopal].trans,
     transcripcion,kk);
          i_actual=i_cub;
          i_cub=i_vb+20;
          /* busca el tabulador */
          do {
               c1=a_cub[i_cub++];
          }while(c1!=' ');
          ii=0;
          do {
     arrpalenc[iarr].info[nopal].escri[ii]=
          a_cub[i_cub++];
          ++ii;
     }while(arrpalenc[iarr].info[nopal].
          escri[ii-1]!='{');
          arrpalenc[iarr].info[nopal].
          escri[ii-1]=0;
          arrpalenc[iarr].info[nopal].
          escri[ii-2]=0;
          ii=0;
```

```
                                    do {
                                    arrpalenc[iarr].info[nopal].
                                        cat[ii]=a_cub[i_cub++];
                                        ++ii;
                                }while(arrpalenc[iarr].info[nopal].
                                        cat[ii-1]!=',');
                                    arrpalenc[iarr].info[nopal].
                                    cat[ii-1]=0;
                                    ii=0;
                                    do {
                                    arrpalenc[iarr].info[nopal].
                                        porig[ii]=a_cub[i_cub++];
                                        ++ii;
                                }while(arrpalenc[iarr].info[nopal]
                                    .porig[ii-1]!='}');
                                    arrpalenc[iarr].info[nopal].
                                    porig[ii-1]=0;
                                    arrpalenc[iarr].info[nopal].dir
                                    =i_vb_r+i_vb;
                                    arrpalenc[iarr].info[nopal].
                                    repson=iarr+jj;
                                    if  (iarr+jj+1>limarrent)
                                    arrpalenc[iarr].info[nopal].
                                        sinrep=fin;
                                    else  arrpalenc[iarr].
                                    info[nopal].sinrep=iarr+jj+1;
                                    ++nopal;
                                    i_cub=i_actual;
                                }
                            }
                        }
/* Si es más grande jj => no hace nada y pasa a la sig.*/
                        /* Limpia la seg. sección */
                    memset(&palaux[nocar],0,sizeof(palabra)nocar);
                        memset(&transcripcion[nocarr],0,
                        sizeof(palabra)-nocarr);
                        jj=nocar;kk=nocarr;
                    }
                    if (!todas) {       /* Si no son todas */
                        /* posicionarse al inicio de un reg. */
                        i_vb+=prox;
                        c1=a_cub[i_vb];  /* ¿Hay más reg? */
```

```
                memset(palaux,0,sizeof(palabra));
                memset(transcripcion,0,sizeof(palabra));
                if (c1 != conti) todas=1;/* Si no acaba */
            }
        }while (!todas);
        /* Abre de nuevo la base en su modalidad anterior */
        a_vb=fopen("VocBas","r");
    }
    else {
        /*  Cuando se busca la próx. o anterior categoría se
            sig. un procedimiento similar pero accesando la
            información del archivo */
        prox=todas=0;
        do {
            fseek(a_vb,i_vb,0);
            if (opci==1)
                fscanf(a_vb,"%*c%*c%x%*x%*c%*c",&prox);
            else
                fscanf(a_vb,"%*c%*c%*x%x%*c%*c",&prox);
            bfinreg=0;
            memset(palaux,0,sizeof(palabra));
            memset(transcripcion,0,sizeof(palabra));
            ii=kk=0;
            do {
                c1=getc(a_vb);
                if ((c1!='(')&&(c1!='/')) {
                    switch(c1) {
                        case 'á': c1='a';break;
                        case 'é': c1='e';break;
                        case 'í': c1='i';break;
                        case 'ó': c1='o';break;
                        case 'ú': c1='u';break;
                    }
                    transcripcion[kk]=c1;
                    if ((c1=='c')||(c1=='r')) {
                        c2=getc(a_vb);
                        /* /ch/ -> /ch/ */
                        if (c2=='h') transcripcion[++kk]=c2;
                        else if (c2=='r') {
                            c1='-';
                            /* /rr/ -> /rr/ */
                            transcripcion[++kk]=c2;
```

```
                    }
                    else ungetc(c2,a_vb);
                }
                palaux[ii]=c1;
                ++ii;++kk;
            }
        }while((c1!='(')&&(c1!='/'));
        nocar=ii;jj=nocar;nocarr=kk;
        while (bfinreg==0) {
            if (c1!='/') {
                do {
                    c1=getc(a_vb);
                    if ((c1!='.')&&(c1!=',')&&(c1!=')')) {
                        switch(c1) {
                            case 'á': c1='a';break;
                            case 'é': c1='e';break;
                            case 'í': c1='i';break;
                            case 'ó': c1='o';break;
                            case 'ú': c1='u';break;
                        }
                        transcripcion[kk++]=c1;
                        if ((c1=='c')||(c1=='r')) {
                            c2=getc(a_vb);
                            if (c2=='h')
                            /* /ch/ -> /ch/ */
                            transcripcion[kk++]=c2;
                            else if (c2=='r') {
                                c1='-';
                                /* /rr/ -> /rr/ */
                            transcripcion[kk++]=c2;
                            }
                            else ungetc(c2,a_vb);
                        }
                    }
                    if (c1=='.') c1=getc(a_vb);
                    else if ((c1!=',')&&(c1!=')')) {
                        palaux[jj]=c1;
                        ++jj;
                    }
                }while((c1!='.')&&(c1!=',')&&(c1!=')'));
                if (c1==')') bfinreg=1;
            }
```

```
else bfinreg=1;
if (jj <= (limarrent-iarr+1)) {
     compara=memcmp((char *)ent,(char *)
               palaux,(int)jj);
     if (compara==0) {
          transcripcion[kk++]=0;
          memcpy(arrpalenc[iarr].info[nopal].
                    trans,transcripcion,kk);
          i_actual=ftell(a_vb);
          fseek(a_vb,(long) (i_vb+20),0);
          do {c1=getc(a_vb);}while(c1!=' ');
          ii=0;
          do {
               arrpalenc[iarr].info[nopal].
               escri[ii]=getc(a_vb);
               ++ii;
          }while(arrpalenc[iarr].info[nopal].
               escri[ii-1]!='{');
     arrpalenc[iarr].info[nopal].escri[ii1]=0;
     arrpalenc[iarr].info[nopal].escri[ii2]=0;
          ii=0;
          do {
          arrpalenc[iarr].info[nopal].cat[ii]
               =getc(a_vb);
               ++ii;
          }while(arrpalenc[iarr].info[nopal].
                    cat[ii-1]!=',');
     arrpalenc[iarr].info[nopal].cat[ii-1]=0;
          ii=0;
          do {
               arrpalenc[iarr].info[nopal].
               porig[ii]=getc(a_vb);
               ++ii;
          }while(arrpalenc[iarr].info[nopal].
                    porig[ii-1]!='}');
     arrpalenc[iarr].info[nopal].porig[ii1]=0;
     arrpalenc[iarr].info[nopal].dir=i_vb;
          fseek(a_vb,i_actual,0);
          bfinreg=1;
          todas=1;
     }
}
```

```
                    memset(&palaux[nocar],0,sizeof(palabra)nocar);
                    memset(&transcripcion[nocarr],0,
                    sizeof(palabra)-nocarr);
                        jj=nocar;kk=nocarr;
                    }
                    if (!todas) {
                        i_vb+=prox;
                        fseek(a_vb,i_vb,0);
                        c1=getc(a_vb);
                        memset(palaux,0,sizeof(palabra));
                        memset(transcripcion,0,sizeof(palabra));
                        if (c1 == conti) todas=0;
                        else todas=1;
                    }
                }while (!todas);
            }
            if (clave==arreg) {
                fclose(arr);
                fclose(arrdir);
            }
        }
    }
}
 /* Reporta la información a pantalla */
if (!opci) {
    arrpalenc[iarr].no=nopal;
    printf("%d) no= %d,",iarr+1,arrpalenc[iarr].no);
    for (ii=0;ii < arrpalenc[iarr].no;++ii) {
        printf("%d: %s",ii+1,arrpalenc[iarr].info[ii].trans);
        printf("(%d,",arrpalenc[iarr].info[ii].repson);
        printf("%d) ",arrpalenc[iarr].info[ii].sinrep);
    }
     printf("\n");
    }
else {
    if (!compara) {
        printf("\n...INFORMACION obtendida de la BASE de DATOS\n");
        printf("(%d,%d): cat=%s,",iarr,nopal,
            arrpalenc[iarr].info[nopal].cat);
        printf(" porig=%s,",arrpalenc[iarr].info[nopal].porig);
        printf(" escri= %s\n",arrpalenc[iarr].info[nopal].escri);
        return(bien);
```

```
    }
    else  return(no_otra_cat);
 }
}

long     calc_hx(ent)      / *  Rutina que calcula el valor de */
unsigned char    *ent;          la función de dispersión */
{
    unsigned char    limite;
    long             result;

  /* Se determina el limite del ciclo (a lo más puede
     haber un arreglo con 4 dim.) */
  limite= (noson < tamrai) ? noson:tamrai*2;
   memset(rai,0,sizeof(raiz));          /*  Inicializa  */
   memset(conv,0,sizeof(raiz));
  ii=0;
  while (ii < limite) {
     if (ii < tamrai) rai[ii]=ent[ii];  /* En rai se almacena la raiz */
     switch (ent[ii]) {                 /* y en conv el valor numérico */
          case 'a':
          case 'á':conv[ii]=a;break;
          case 'b':conv[ii]=b;break;
          case 'c':conv[ii]=c;break;
          case 'd':conv[ii]=d;break;
          case 'e':
          case 'é':conv[ii]=e;break;
          case 'f':conv[ii]=f;break;
          case 'g':conv[ii]=g;break;
          case 'i':
          case 'í':conv[ii]=i;break;
          case 'k':conv[ii]=k;break;
          case 'l':conv[ii]=l;break;
          case 'm':conv[ii]=m;break;
          case 'n':conv[ii]=n;break;
          case 'ñ':conv[ii]=nn;break;
          case 'o':
          case 'ó':conv[ii]=o;break;
          case 'p':conv[ii]=p;break;
          case 'r':conv[ii]=r;break;
          case '-':conv[ii]=rr;break;
          case 's':conv[ii]=s;break;
```

```
            case  't':conv[ii]=t;break;
            case 'u':
            case  'ú':conv[ii]=u;break;
            case  'w':conv[ii]=w;break;
            case  'x':conv[ii]=x;break;
            case  'y':conv[ii]=y;break;
        }
        ++ii;
    }
    /* Calcula el valor de la función de dispersión */
    result=0;
    for(ii=0,jj=noson-1;ii  <  noson;++ii,--jj)
        result+=  (conv[ii]+1)*ipower((extended)  base,jj);
    --result;
    return(result);
}
```

B.6 Código del programa Reporta.c

```
/ *  Mediante este programa se generan las oraciones o
     se buscan otras categorías gramaticales de palabras * /

#include <TiposyCtes.h>
#include <VGlobExt.h>

static unsigned char  proxcam;/* proxcam= próx. camino, nopal= */
extern unsigned char  nopal;   /* no. de pal., banfino= bandera de */
unsigned char    car,ip,banfino;/* fin de oración, ip= indice de pilas */
unsigned char    pals,tam,indice;
palabra           buf1,buf2;
char              banerr;

/* Declaración de procedimientos presentes en el archivo */

int  iniPil(),encuentra(),actualiza();
int  act_fino(),imprimeOracion(),camb_nom();

reporta()/* Rutina principal que llama a los demás procedimientos */
{
     unsigned int ii;

 if (!nollamaRepxf) {/* Si es la primera vez que se llama, continua */
     iniPil();       /* Se inicializan las cinco pilas */
 }
 ip=proxcam;       /* Actualiza el indice de las pilas */
 memset(bufRep,0,tambuf);
 switch (opciRep) {    /* Preg. por cual opción se solicitó */
     case '1':       /* Si se pidió traer la sig. oración => continua */
          do {       /* Ciclo hasta que encuentre el fin de oración */
               /* Encuentra el próximo elemento a cargar en la pila */
               banerr=encuentra();
                    /* Si no hay más oraciones, termina */
               if (banerr==no_mas_orac) goto e1;
               actualiza(); /* Actualiza las pilas */
          } while (!banfino);
          imprimeOracion();    /* Reporta la oración */
          act_fino();          /* Remueve los elementos innecesarios */
```

```
        ip=proxcam;       /* Inicializa */
        banfino=0;
        break;
    case '2':           /* Si se pidió cambio de frase => continua */
                    /* Busca el próximo elemento a cargar en la pila */
        banerr=encuentra();
        actualiza(); /* Actualiza las pilas */
        break;
    case '3':
    case '4': /* Si se pidió la próx. o la ant. categoría => continua */
        printf("Dé  iarr=\n");        /* Preg. el índice en el arreglo */
        scanf("%u",&ii);
        fflush(stdin);
        iarr=ii;
            /* Preg. el # de palabra */
        printf("Dé el número de palabra=\n");
        scanf("%u",&ii);
        nopal=ii;
        fflush(stdin);
        tam=strlen(arrpalenc[iarr].info[nopal].trans);
            /* buf1 guarda la trans. de la pal.*/
        memcpy(buf1,arrpalenc[iarr].info[nopal].trans,tam);
            /* Obtiene su dir. en la base */
        i_vb=arrpalenc[iarr].info[nopal].dir;
        nmspc=camb_nom();  /* Hace le cambio de nomenclatura */
            / *  De acuerdo a si se busca la prox. o */
                la anterior se llama a consulta */
        if (opciRep=='3') banerr=consulta(buf2,1);
        else banerr=consulta(buf2,2);
        break;
    case '5': /* Si se pidió obtener todas => continua */
        /* Abre el archivo de salida, sin destruir su contenido */
        a_s=fopen("Revlex.sal","a");
        iniPil();
        do { /* Ciclo hasta no encontrar más oraciones */
            /* Busca el próximo elemento a cargar en la pila */
            banerr=encuentra();
            /* Si hay más palabras, continua */
            if (banerr!=no_mas_orac) {
                actualiza(); /* Actualiza las pilas */
                /* Si ya se encontró el fin de oración, continua */
                if (banfino) {
```

```
                    imprimeOracion();    /* Imprime */
                    act_fino();
                    ip=proxcam;      /* Inicializa */
                    banfino=0;
                }
            }
        } while (banerr!=no_mas_orac);
        fclose(a_s);                    /* Cierra el archivo */
        printf("...Información impresa en Revlex.sal\n");
        break;
}
e1: /* Avisos para los diferentes errores */
    if (banerr==no_mas_palab) {
        printf("No hay más oraciones que se deriven\n");
        printf("a partir de este punto\n");
    }
    if (banerr==no_mas_orac) {
        if (opciRep!='5') printf("No hay más oraciones\n");
        iniPil();               /* Inicializa pilas */
    }
    if (banerr==no_otra_cat)
        printf("No hay más palabras con esa misma cat.\n");
    ++nollamaRepxf;             /* Actualiza el contador */
}

iniPil()                /* Rutina que inicializa las pilas */
{
    ip=0;               /* Indice de pilas en 0 */
    preg[ip]=0;         /* Guarda el no. de registro */
    ppal[ip]=arrpalenc[preg[ip]].no; /* Guarda el no. de palabras */
    pcam[ip]=2;                      /* Guarda el no. de caminos */
    pimpp[ip]=1;                     /* Indica que se imprima */
    prepf[ip]=0;        /* Indica que no se reportará el sonido final */
    pals=arrpalenc[0].no;            /* Obtiene el no. de palabras */
                        /* Obtienen el tamaño de la 1ª palabra */
    tam=strlen(arrpalenc[0].info[0].trans);
    /* Si hay más de una palabra o el tamaño de la 1ª >1, conti. */
    if ( (pals>1)||(tam>1) ) {
        car=arrpalenc[0].info[0].trans[0];
        /* Investiga si inicia con vocal */
        switch (car) {
            case 'a':
```

```
                case 'á':
                case 'e':
                case 'é':
                case 'i':
                case 'í':
                case 'o':
                case 'ó':
                case 'u':
                case 'ú':prepi[ip]=1;
                         break;/* Si es asi => enciende el indicador */
                default: prepi[ip]=0;
            }
        }
        else prepi[ip]=0; /* En caso contario, se apaga */
        proxcam=ip;       /* Actualiza */
        banfino=0;
}

encuentra()       / * Rutina que localiza el próximo elemento
                      que se puede agregar a la pila */
{
        unsigned char bcam,bandec,aux,indice1,indice2;
        int           coderr;

if (opciRep=='2') /* Si se pidió cambio de frase, continua */
        if (ip > 0) {  /* Si no es el 1er elemnto en la pila, continua */
            preg[ip]=ppal[ip]=pcam[ip]=0;/* Quita el último elemento */
            pimpp[ip]=prepi[ip]=prepf[ip]=0;
            --ip;     /* Decrementa el indice de las pilas */
            --ppal[ip];/* Decrementa el # de palabras */
            pcam[ip]=2;/* Fija el # de caminos en dos */
            pimpp[ip]=1;
            prepf[ip]=0;/* y apaga el indicador de repetir el final */
        }
        else return(no_mas_orac);/* De llegar al 1º => termina */
 coderr=bien;
 do {     / * Ciclo hasta encontrar una variable o agotar las
                posibilidades */
        if (ip) prepf[ip-1]=0;
        pals= arrpalenc[ preg[ip] ].no;  /* Obtiene el # de palabras */
        if ( ppal[ip] > 0) {             /* Si hay algo, continua */
            /* Si hay 1 o 2 caminos, continua */
```

```
if ((pcam[ip]==2)||(pcam[ip]==1)) {
        /*Obt. los indices a las sig. palabras */
        indice1=arrpalenc[preg[ip]].info[pals-ppal[ip]].repson;
        indice2=arrpalenc[preg[ip]].info[pals-ppal[ip]].sinrep;
    if (pcam[ip]==2) {      /* Si pcam=2, continua */
        if (indice1) {/* Si el indice es dif. de 0, continua */
            /* Obt. el # de palabras, a partir del indice */
            aux=arrpalenc[indice1-1].no;
            if (aux) {      /* Si es dif. de cero, continua */
                    /* Toma el 1er camino */
                indice=indice1;
                bcam=1;
                bandec=0;
            }
            else bandec=1; /* Si es = 0, repetir el ciclo */
        }
        else bandec=1; /* Si es = a cero, repetir el ciclo */
    }
    else {    /* Cuando pcam=1 */
        if (indice2==fin) {/* Si se detectó fin de archivo */
            indice=indice2;/* Toma el 2º camino */
            / *  Se rev. si termina en vocal, si es así no se
                imprime la última vocal y se enciende
            la de repetir sonido de la palabra anterior */
            if (ip) {
                pals=arrpalenc[preg[ip-1]].no;
                aux=arrpalenc[preg[ip-1]].
                    info[pals-ppal[ip1]].sinrep;
                if (indice2==aux) {
                    pimpp[ip]=0;
                    prepf[ip-1]=1;
                }
            }
            bcam=1;
            bandec=0;
        }
        / *  Si el indice es dif. de cero y el no. de palabras a
            partir del indice también lo es, continua */
        else if (arrpalenc[indice2-1].no) {
            indice=indice2;  /* Toma el 2º camino */
            bcam=1;
            bandec=0;
```

```
if (!indice1) {     /* Si viene por 0, conti. */
    if (ip) {
        pals=arrpalenc[preg[ip-1]].no;
        aux=arrpalenc[preg[ip-1]].
            info[palsppal[ip-1]].repson;
        / *  Preg. por el indice2 de la palab.
            ant. guardad en la pila */
        if (aux) {
    / *  Si viene de sinrep. no debe de imprimir
        la pal. que se guardará en la pila */
            if (pcam[ip-1]==1) pimpp[ip]=0;
            else pimpp[ip]=1;
            tam=strlen(arrpalenc[preg[ip1]].
                info[pals-ppal[ip-1]].trans);
            car=arrpalenc[preg[ip1]].
            info[pals-ppal[ip-1]].trans[tam-1];
    /* ¿La pal. anterior termina en vocal? */
            switch (car) {
                case 'a':
                case 'e':
                case 'i':
                case 'o':
                case 'u':prepf[ip-1]=1;
        /* y si es vocal, indica que la repita */
                break;
            }
        }
    }
}
else { /* Si indice1 es dif. de cero */
    tam=strlen(arrpalenc[preg[ip]].
        info[palsppal[ip]].trans);
    car=arrpalenc[preg[ip]].info[palsppal[ip]].
        trans[tam-1];
    /* Se inves. si termina en vocal */
    switch (car) {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':prepf[ip]=1;
        /* y si es vocal, indica que la repita */
```

```
                                break;
                            }
                        }
                    }
                    /* En caso contrario, repite el ciclo */
                    else bandec=1;
                }
                /* Si está habilitada la bandera de decrementar */
                if (bandec) {
                    --pcam[ip];  /* Decrementa pcam */
                    bcam=0;      /* y repite ciclo */
                }
            }
            else {              /* Cuando pcam=0 */
                --ppal[ip];     /* Dec. el # de palabras */
                pcam[ip]=2;     /* Actualiza */
                pimpp[ip]=1;
                prepf[ip]=bcam=0;     /* y repite ciclo */
            }
        }
        else if (ip > 0) { /* Si no hay + palabras, continua */
            if (opciRep==2) /* Si se pidió cambiar de frase, */
                return(no_mas_palab);     /* terminar */
                /* Limpia las pilas en esa posición */
            preg[ip]=pcam[ip]=pimpp[ip]=prepi[ip]=prepf[ip]=0;
            bcam=0; /* Enciende la bandera que indica repetir ciclo */
            --ip;     /* Decrementa el indice a las pilas */
            if (pcam[ip]>0) --pcam[ip];/* Dec. el # de caminos */
        }
        /* Si ya se llegó al 1er elemento, termina */
        else coderr=no_mas_orac;
    } while ((!bcam)&&(coderr!=no_mas_orac));
    return(coderr);  /* Termina notificando */
}

actualiza()             /* Rutina que instala elementos en las pilas */
{

    --pcam[ip];         /* Dec. pcam */
    if (indice==fin)    /* Si ya se llegó al fin, continua */
        banfino=1;      /* Indica que se encontró fin de oración */
    else {              /* SI es dif. de fin */
```

```
    --indice;
    pals=arrpalenc[indice].no;/* Obt. el # de pal. */
                        /* Obt. el tamaño de la 1ª palabra */
    tam=strlen(arrpalenc[indice].info[0].trans);
    ++ip;           /* Instala el nuevo elemento */
    preg[ip]=indice;
    ppal[ip]=arrpalenc[preg[ip]].no;
    pcam[ip]=2;
    pimpp[ip]=1;
    prepf[ip]=0;
    / *  Si hay + de 1 palabra o el tam. de la 1ª es >1,
         preg. por el 1er caracter */
    if ( (pals>1)||(tam>1) ) {
         car=arrpalenc[indice].info[0].trans[0];
         switch (car) {
              case 'a':
              case 'á':
              case 'e':
              case 'é':
              case 'i':
              case 'í':
              case 'o':
              case 'ó':
              case 'u':
              case 'ú':prepi[ip]=1;
              /* Si es vocal, indica que la repita */
                        break;
              /* en caso contrario, no rep. el sonido inicial */
              default:  prepi[ip]=0;
         }
    }
              /* en caso contrario, no rep. el sonido inicial */
    else prepi[ip]=0;
    proxcam=ip; /* Indicar cual será el próximo camino */
    banfino=0;   /* Apagar la bandera */
 }
}

act_fino()    / *  Rutina que sirve para actualizar las pilas después
                   de haber encontrado el fin de oración e impreso */
{
    unsigned char banre;  /* banre= bandera de registro */
```

```
banre=0;
do {/* Ciclo hasta encontrar un reg. con palabras no analizadas */
                    /* Si pcam > 0 => continuar con el mismo indice */
    if (pcam[ip] > 0) {/* Si pcam es ≠ 0 => ya se encontró un reg. */
        proxcam=ip;
        banre=1;
    }
    else {    /* En caso contrario, continua */
        if (ppal[ip] > 0) {  /* si hay más palabras por analizar => */
            proxcam=ip; /*continuar con el mismo indice, */
            --ppal[ip];   /* dec. el no. de palabras, */
            pcam[ip]=2;  /* no. de caminos igual a dos, */
            pimpp[ip]=1; /* habilita la impresión de la palabra, */
            prepf[ip]=0;  /* se indica que no rep. la última vocal */
            banre=1;
        }
        else if (ip > 0) {  /* En caso de no haber palabras, */
                            /* limpia las pilas en esa posición */
        preg[ip]=pcam[ip]=pimpp[ip]=prepi[ip]=prepf[ip]=0;
                            /* continua con el elemento anterior */
            --ip;
                    /* Si # de caminos es ≠ 0, actualiza su valor */
            if (pcam[ip]>0) --pcam[ip];
        }
        else {banre=1;proxcam=ip;}
    }
 } while(!banre);
}

imprimeOracion()      / *  Rutina que se encarga de reportar los
                           resultados tanto en memoria, en pantalla
                           y en archivo */
{
    unsigned char     ii;  /* Indice de pilas */
    int               jj;  /* Indice para bufRep */

  ii=jj=0;
 do {     /* Ciclo hasta no encontrar más elementos en las pilas */
        /* Imprime una vocal opcional al principio */
    pals=arrpalenc[preg[ii]].no;
    tam=strlen(arrpalenc[preg[ii]].info[pals-ppal[ii]].trans);
```

```
    / *  Si está encendida la bandera y la pal.
         a reportar tiene más de un caracter */
if  ((tam>1)&&(prepi[ii])) {
    bufRep[jj].opci=1;/* => Reporta */
    bufRep[jj].x=ii;   /* Llenando en memoria */
    bufRep[jj].y=0;
    ++jj;
    /* Escribiendo la información */
    if (opciRep=='5') {/* Si se pidó todas, reporta al archivo */
        fprintf(a_s,"{%c}  ",arrpalenc[preg[ii]].info[0].trans[0]);
        printf("{%c}  ",arrpalenc[preg[ii]].info[0].trans[0]);
    }
    else  printf("{%c}  ",arrpalenc[preg[ii]].info[0].trans[0]);
}
if (pimpp[ii]) {/* Si está habilitado el indicador de impresión */
    /* => Imprime la palabra */
    bufRep[jj].opci=0;
    bufRep[jj].x=ii;
    bufRep[jj].y=pals-ppal[ii];
    ++jj;
    if (opciRep=='5') {/* Si se pidó todas, reporta al archivo */
        fprintf(a_s,"%s  ",arrpalenc[preg[ii]].info[palsppal[ii]].
                trans);
        printf("%s  ",arrpalenc[preg[ii]].info[pals-ppal[ii]].trans);
    }
    else {
        printf("%s  ",arrpalenc[preg[ii]].info[pals-ppal[ii]].trans);
    }
}
/* Imprime una vocal opcional al final */
    / *  Si está encendida la bandera y la pal.
         a reportar tiene más de un caracter */
if  ((tam>1)&&(prepf[ii])) {
    bufRep[jj].opci=1;     /* entonces, reporta */
    bufRep[jj].x=arrpalenc[preg[ii]].info[pals-ppal[ii]].repson-1;
    bufRep[jj].y=0;
    ++jj;
    if (opciRep=='5') {/* Si se pidó todas, reporta al archivo */
        fprintf(a_s,"{%c}  ",arrpalenc[preg[ii]].info[palsppal[ii]].
                trans[tam-1]);
        printf("{%c}  ",arrpalenc[preg[ii]].info[palsppal[ii]].
                trans[tam-1]);
```

```
            }
            else printf("{%c} ",arrpalenc[preg[ii]].info[palsppal[ii]].
                 trans[tam-1]);
        }
        ++ii;
    } while((preg[ii]!=0)&&(ii < limarrent));
    if (opciRep=='5') {fprintf(a_s,"\n");printf("\n");}
    else printf("\n");
}

camb_nom()      /* Rutina que realiza el cambio de nomenclatura */
{
    unsigned char ii,jj,aux;

    ii=jj=0;
    memset(buf2,0,sizeof(palabra));
   do {
        if (buf1[ii]=='c') {aux=buf1[ii];++ii;} /* ch -> c */
        else if ((buf1[ii]=='r')&&(buf1[ii+1]=='r')) {aux='-';++ii;} /* rr ->  */
        else aux=buf1[ii];
        buf2[jj]=aux;
        ++ii,++jj;
    }while(ii < tam);
    return(jj);
}
```

B.7 Código del programa AgregaSon.c

```
/ *  Mediante este programa se tratá de formar los registos
     de la base de datos, VocBas. Su entrada es la escritura
     de las palabras, y agrupadas junto a su unidad léxica,
     donde también se proporciona la categoría. Como salida
     se generan tres archivos: el que sirve para generar la
     base, el se ocupa para encontrar los grupos cons., y
     otro donde no se agrupan términos. * /

#include <stdio.h>
#include <Types.h>
#include <CType.h>
#include <FCntl.h>
#define tamrai 4
#define tampal 30
#define tamcat 3 /* el último siempre debe tener 0 */
#define maxp_r 5 /* # máximo de palabras en un registro */

FILE            *ent,*sal,*rep,*sr;
char            archent[30];
char            opci,carac0,carac1,aux;
unsigned char   palab[tampal],sonpal[tampal],noson,nocs,nolet;

/ *  palab= almacena la escritura de la palabra,
            más las posteriores especificaciones
     sonpal= almacena la transcipción de la palabra
     noson= # de sonidos
     nocs = # de carcteres empleados en la trascripción
     nolet= # de letras (carcteres) * /

struct  especif {
  unsigned char      pal[tampal]; /* Se almacena la unidad léxica */
  unsigned char      cat[tamcat]; /* Se almacena la catrgoría
gramatical  */
} especif;

struct  registro {
     unsigned char        ns;nl;    /* # de sonidos, # de palabras */
     unsigned char        p[tampal];/* Se guarda la escritura */
```

```
    unsigned char        sp[tampal];/* Se guarda la transcripción */
} reg[maxp_r];

int lee();    /* Declaración de procedimientos presentes en el
archivo */
int tradson();

main() /** Programa Principal **/
{
 unsigned char i,j,ncd,nm,ban;
  /* ncd= # de comparaciones diferentes
     nm= # mínimo de sonidos o letras (en palab de long. >=4) */

     /* Abre los archivos de salida, respetando lo que tenían antes */
sal=fopen("VocBas.txt","a");
rep=fopen("VocBas.rep","a");
sr=fopen("VocBas.sr","a");
     /* Ciclo hasta que no se desee procesar ningún otro archivo */
do {
     /* Pide el nombre del archivo de entrada */
  printf("Dame el arch. de ent.\n");
  scanf("%s",archent);
  fflush(stdin);
  ent=fopen(archent,"r");
  carac0=getc(ent);     /* Lee el indicador de inicio de registro */
  aux=getc(ent);        /* Lee, almacena, y devuelve el sig. caracter */
  ungetc(aux,ent);
 while (aux!=EOF) {     /* Ciclo hasta detectar el fin de archivo */
     memset(&especif,0,tampal+tamcat);    /* Inicialización */
     ban=i=0;
     do { /* Almacena la unidad léxica, hasta encontrar una pausa */
          carac1=getc(ent);
          if ((carac1!=' ')&&(i<tampal)) {especif.pal[i]=carac1;++i;}
     }while(carac1!=' ');
     printf("pal=%-s\n",&especif.pal[0]);
     i=0;
     do {      /* Almacena la categoría, hasta encontrar un punto */
          carac1=getc(ent);
          if ((carac1!='.')&&(i<tamcat)) {especif.cat[i]=carac1;++i;}
     }while(carac1!='.');
     printf("cat=%s\n",especif.cat);
          /* Lee los carac. adiconales hasta encontrar el fin de linea */
```

```
do  carac1=getc(ent);while(carac1!='\n');
    /* Si es pronombre o verbo ban=1 */
if  ((especif.cat[0]=='p')&&(especif.cat[1]=='n'))  ban=1;
if  ((especif.cat[0]=='v'))  ban=1;
if (ban) {     /* Si ban está habilitada, continua */
    do { /* Ciclo hasta encontrar la próxima unidad léxica */
        /* Lee información, y si pudo leer => continua */
        if (lee()) {
            tradson();    /* Pide la transcripción */
                /* Reporta */
            fprintf(sal,"%s    %s {%s,%-s}  TNR\n",
            sonpal,palab,especif.cat,especif.pal);
            fprintf(sr,"%s     %s {%s,%-s}  TNR\n",
            sonpal,palab,especif.cat,especif.pal);
            fprintf(rep,"%s\n",sonpal);
        }
        carac0=getc(ent);
        if  (carac0!='#')  ungetc(carac0,ent);
    }while  (carac0!='#');
}
else {    /* Cuando ban está deshabilitada */
    memset(reg,0,(int)  maxp_r*(2+2*tampal));  /*  Inicializa  */
    i=0;
    do { /* Ciclo hasta encontrar la próxima unidad léxica */
        /* Lee información, y si pudo leer => continua */
        if (lee()) {
            tradson();    /* Pide la transcripción */
                /* Reporta en rep y sr */
            fprintf(rep,"%s\n",sonpal);
            fprintf(sr,"%s     %s {%s,%-s}  TNR\n",
            sonpal,palab,especif.cat,especif.pal);
            /* Si tiene menos de 4 sonidos => reporta en sal */
            if (noson < tamrai)
                fprintf(sal,"%s    %s {%s,%-s}  TNR\n",
                sonpal,palab,especif.cat,especif.pal);
            else {/* en caso contrario, almacena en memoria */
                reg[i].ns=nocs;reg[i].nl=nolet;
                memcpy(reg[i].p,palab,(int)  nolet);
                memcpy(reg[i].sp,sonpal,(int)  nocs);
                ++i;
            }
        }
```

```
    carac0=getc(ent);
    if (carac0!='#') ungetc(carac0,ent);
}while ((carac0!='#')&&(i<maxp_r));
/ *  Acaba, si se requiere más memoria
    de la indicada para almacenar */
if (carac0!='#') {
    printf("Se requiere incrementar maxp_r, i= %d\n",i);
    return(0);
}
 /* Si algo se almaceno => reg[0] debe contener algo */
if (reg[0].ns) {
    if (reg[1].ns) {    /* Si hay + de uno */
        i=0;nm=tampal;
        do {      /* Busca el # mínimo de sonidos */
            if (nm > reg[i].ns) nm=reg[i].ns;
            ++i;
        }while (reg[i].ns);
        i=1;
        do {/* Ciclo para calcular el # de sonidos iguales */
            j=1;ncd=0;
            /* preguntando en todos los elmentos de reg */
            do {
                ncd= (memcmp(reg[0].sp,reg[j].sp,(int) i))?
                    ++ncd:ncd;
                ++j;
            }while (reg[j].ns);
            ++i;
        }while ((!ncd)&&(i <= nm));
        nm=i-2;
            /* Imprime la trans. de la 1ª palabra */
        for (j=0;j < nm;++j) fprintf(sal,"%c",reg[0].sp[j]);
        fprintf(sal,"(");  /* Abre paréntesis ( */
        i=0;
    /* Ciclo mientras se encuentren elementos en reg. */
        do {
            /* Si la pal. tiene nm sonidos => imprime . */
            if (reg[i].ns == nm) fprintf(sal,".");
            else
            / *  Si no, imprime los caracteres restantes */
                for (j=nm;j < reg[i].ns;++j)
                    fprintf(sal,"%c",reg[i].sp[j]);
            ++i;
```

```
                /* Si continua otra palabra => imprime , */
                if (reg[i].ns) fprintf(sal,",");
                /* de lo contario, cierra paréntesis, ) */
                else   fprintf(sal,")");
            } while(reg[i].ns);
    /* Se sigue un proceso similar para escribir la palabra */
            i=0;nm=tampal;
            do {      /* Busca el # mínimo de letras */
                if (nm > reg[i].nl) nm=reg[i].nl;
                ++i;
            }while (reg[i].nl);
            i=1;
            do { /* Ciclo para calcular el # de letras iguales */
                j=1;ncd=0;
                do {
                    ncd= (memcmp(reg[0].p,reg[j].p,(int) i))?
                    ++ncd:ncd;
                    ++j;
                }while (reg[j].nl);
                ++i;
            }while ((!ncd)&&(i <= nm));
            nm=i-2;
/* Separa la trans. de la escritura por medio de un tabulador */
            fprintf(sal," ");
                    /* Imprime la trans. de la 1ª palabra */
            for (j=0;j < nm;++j) fprintf(sal,"%c",reg[0].p[j]);
            fprintf(sal,"(");  /* Abre paréntesis */
            i=0;
            do {
                    /* Si la pal. tiene nm letras => imprime . */
                if (reg[i].nl == nm) fprintf(sal,".");
                / *  Si no, imprime los caracteres restantes */
                else
                    for (j=nm;j < reg[i].nl;++j)
                        fprintf(sal,"%c",reg[i].p[j]);
                ++i;
                    /* Si continua otra palabra => imprime , */
                if (reg[i].nl) fprintf(sal,",");
                    /* de lo contario, cierra paréntesis, ) */
                else   fprintf(sal,")");
            } while(reg[i].nl);
                /* Reporta en sal */
```

```
                fprintf(sal," {%s,%-s} TNR\n",
                    especif.cat,especif.pal);
            }
                /* Si sólo hay una palabra, reporta en sal */
            else  fprintf(sal,"%s   %s {%s,%-s} TNR\n",
                    reg[0].sp,reg[0].p,especif.cat,especif.pal);
        }
    }
    /* Lee y devuelve el sig. caracter */
    aux=getc(ent);
    ungetc(aux,ent);
}
    /* Cierra el archivo de entrada */
 fclose(ent);
    /* Pregunta si se desea continuar */
 printf("Desea correrlo de nuevo (s/n)\n");
 opci=getchar();
 fflush(stdin);
 } while(opci=='s');
    /* En caso contario, cerrar los archivos de salida */
fclose(sal);
fclose(rep);
fclose(sr);
}

lee()                       / *  Rutina que lee una palabra del archivo
                                 de entrada y la almacena en palab. */
{
    unsigned char     i,ban;

 carac1=getc(ent);     /* Lee el 1er caracter */
 if (carac1=='-') {    /* Si empieza con guión, continua, sino a
memset */
                       /* entonces lee toda la palabra */
      do carac1=getc(ent); while (carac1!='\n');
       return(false);  /* e indica que no se almacenó
                          en palab y retona. */
}
 memset(palab,0,tampal); /*  Inicialización  */
 i=ban=nolet=0;
 while (carac1!='\n')       /* Mientras no se detecte el fin
                               de renglón, continua */
```

```
{
    palab[i++]=carac1;     /* Almacena el caracter en palab[i]
                                  e incrementa i */
        /* Si detecta un # o un blanco para de contar, */
    if (isdigit(carac1)||(carac1==' ')) ban=1;
    else if (!ban) ++nolet; /* sino cuenta el # de letras. */
    carac1=getc(ent);          /* Lee el próximo caracter, */
    if ((ban)&&(carac1=='[')) /* Si hay un comentario, ej. [agregado] */
        /* lee hasta fin de renglón. */
        do carac1=getc(ent); while (carac1!='\n');
}
 return(true);          /* Se indica que almacenó algo y retorna. */
}

tradson()     / *  Rutina que obtiene la transcripción de la palabra,
                   guardandola en sonpal, y calcula ncs y noson * /
{
    unsigned char i,j,k,aux;

 memset(sonpal,0,tampal);      /*  Inicialización  */
 i=j=k=0;                      /* i: indica el # de letra, j: el ncs, k: el
noson  */
 do {
    aux=0;
    switch (palab[i]) {        /* Pregunta por palab[i] */
        case 'a':
        case 'á':
        case 'b':
        case 'd':
        case 'e':
        case 'é':
        case 'f':
        case 'i':
        case 'í':
        case 'k':
        case 'n':
        case 'ñ':
        case 'o':
        case 'ó':
        case 's':
        case 't':
        case 'u':
```

```
case 'ú':
case 'w':
     / *  Si son cualquiera de estas, se usa */
     / *  el mismo caracter en la transcipción */
           sprintf(&sonpal[j],"%c",palab[i]);
          break;
case 'c':
     /* Si inicia con cn => c -> /s/ */
           if ((!i)&&(palab[i+1]=='n')) aux='s';
           else if
          ((palab[i+1]=='e')||(palab[i+1]=='i')||(palab[i+1]=='é')
           ||(palab[i+1]=='í')) aux='s';
     /* Si le sig. una e o i => c -> /s/ */
           else if (palab[i+1]=='h') {/* Si es ch=> ch -> /ch/ */
               sprintf(&sonpal[j],"c");
               ++i;++j; /* 1 son. escrito mediante 2 carac. */
               aux=palab[i];
           }
           else aux='k';      /* En caso contrario c -> /k/ */
            sprintf(&sonpal[j],"%c",aux);
           break;
case 'g':
     /* Si inicia con gn => elimina g */
           if ((!i)&&(palab[i+1]=='n')) aux=palab[++i];
     /* Si le sigue u => g -> /g/ */
           else if (palab[i+1]=='u') {
               aux='g';
     /* Si gue,gui -> elimina u, /ge,gi/ */
               if ((palab[i+2]=='e')||(palab[i+2]=='i')
               ||(palab[i+2]=='é')||(palab[i+2]=='í')) ++i;
           }
            else if ((palab[i+1]=='e')||(palab[i+1]=='i')
               ||(palab[i+1]=='é')||(palab[i+1]=='í')) aux='x';
     /* Si no le sigue u => g -> /x/ */
           else aux='g'; /* En caso contrario asigna g */
            sprintf(&sonpal[j],"%c",aux);
           break;
case 'h':
/* No hacer nada, => dec. porque adelante se incrementa */
            --j;--k;break;
case 'j':
            sprintf(&sonpal[j],"x");break;  /* j -> /x/ */
```

```
case 'l':
        /* Si le sigue l => ll -> /y/ */
         if (palab[i+1]=='l') {aux='y';++i;}
         else aux=palab[i];/* en caso contario, l  -> /l/ */
          sprintf(&sonpal[j],"%c",aux);
         break;
case 'm':
        /* En inicio de palabra, mn -> /n/ */
         if ((!i)&&(palab[i+1]=='n')) aux=palab[++i];
         else aux=palab[i];/* en caso contario, m -> /m/ */
          sprintf(&sonpal[j],"%c",aux);
         break;
case 'p':
         if ((!i)&&( (palab[i+1]=='s')||(palab[i+1]=='t') ))
             aux=palab[++i];
         /* En inicio de palabra, si ps o pt => elimina la p */
         else aux=palab[i];/* en caso contario, p -> /p/ */
          sprintf(&sonpal[j],"%c",aux);
         break;
case 'q':
          sprintf(&sonpal[j],"k");++i;break; /* qu -> /k/ */
case 'r':
        /* en inicio de palabra r -> /rr/ */
          if (!i) {sprintf(&sonpal[j],"rr");++j;}
          else if ((palab[i-1]=='l')||(palab[i-1]=='n')||(palab[i-1]
             =='s'))
        {
             /* Si lr, nr, o sr -> /lrr, nrr, srr/ */
             sprintf(&sonpal[j],"rr");
             ++j;
         }
          else if (palab[i+1]=='r') {  /* Si rr -> /rr/ */
             sprintf(&sonpal[j],"rr");
             ++i;
             ++j;
         }
        /* en caso contario, r -> /r/ */
          else  sprintf(&sonpal[j],"r");
         break;
case 'ü':
          sprintf(&sonpal[j],"u");break;  /* ü -> /u/ */
case 'v':
```

```
                    sprintf(&sonpal[j],"b");break;  /* v -> /b/ */
            case 'x':
                    /* Si no es inicio de palabra => */
                     if (i) {sprintf(&sonpal[j],"k");++k;++j;}
                     sprintf(&sonpal[j],"s");    /* x -> /ks/ ; si no /s/ */
                    break;
            case 'y':
                    /* Si termina en y => y -> /i/ */
                     if (i== nolet-1) aux='i';
                     else aux=palab[i];/* en caso contario, y -> /y/ */
                     sprintf(&sonpal[j],"%c",aux);
                    break;
            case 'z':
                     sprintf(&sonpal[j],"s");          /* z -> /s/ */
        }
        ++i;++j;++k;
    }while (i < nolet);
    nocs=j;
   noson=k;
}
```

B.8 Código del programa CreaBD.c

```
/ *  Mediante este programa se crea la base de datos,
     VocBas, y todos  sus archivos auxiliares para
     poder recuperar la información de un registro.
     En Tabcub.rep especifica para cada cubeta creada,
     el no. de registros por cubeta y el no. máximo
     de sonidos que se pueden especificar antes del
     paréntesis.  * /


#include <TiposyCtes.h>

FILE            *a_e,*a_r,*a_vb,*a_tc,*a_at,*arr,*arrdir;
palabra         tp1,tp2,conv1,conv2,conv3,conv4;
raiz            rai1,rai2;
unsigned char       ii,jj,car,nc1,nc2,noson1,noson2;
unsigned char       noreg_c,valim,dimarr,*a_cub;
long                a_ini_cub,a_fin_cub,i_tc,i_reg,f_reg,ant,prox,i_cub;
int             err,nocubs,dif;

a_: apuntador a un archivo o a cubeta;
i_: indice en un archivo o en la cubeta;
arrdir: apuntador al archivo "_.arr.dir";
dimarr: La dimensión del arreglo;
arr: apuntador al archivo "_.arr";
noreg_c: no. de reg. en la cubeta;
noreg_c: no de reg. en la cubeta;
tp: transcripción de la palabra;
nocubs: no. de cubetas;
noson: no. de sonidos;
nc: no. de caracteres;
valim:  valor  limite;

/* Declaración de procedimientos presentes en el archivo */

int  prevllena(),convierte(),func_disp();
int  llena(),lee_cub(),llenarr();

main()  /** Programa  Principal **/
```

```
{
        /* Abre archivos */
 a_e=fopen(":BaseDatosf:VocBas.st","r");
 if (!a_e) {
  printf("ERROR, no existe VocBas.st\n");
   return(-1);
 }
 a_r=fopen("TabCub.rep","w");
 a_at=fopen("Aux.tab","w");
    /* Llena previamente la Tabla de cubetas y la base de datos */
 prevllena();
        /* Abre archivos */
 a_vb=fopen("VocBas","r+");rewind(a_vb);
 a_tc=fopen("Tabla.cub","r+");rewind(a_tc);
nocubs=0;   /* Inicializa */
 memset(tp1,0,sizeof(palabra));
 ii=0;    /* Lee la transcripción del 1er reg.en VocBas.st */
do {     /* hasta encontrar un tabulador o un paréntesis */
    tp1[ii]=getc(a_e);
    ++ii;
 }while((tp1[ii-1]!='    ')&&(tp1[ii-1]!='(')&&(tp1[ii-1]!=EOF));
 nc1=ii-1;
 err=tp1[ii-1];
 if ((err==EOF)&&(ii==1)) goto cierra; /* Si encuentra EOF acaba */
        /* Termina de leer el registro */
 do {car=getc(a_e);}while((car!=finreg)&&(car!= EOF));
 err=getc(a_e);   /* se lee el sig. caracter */
 if (err!=EOF) ungetc(err,a_e);  /* Si no es EOF => lo regresa */
 ant=0;
        /* Pregunta en que posición estamos dentro de VocBas */
        /* y se lo asigna al apuntador al inicio de cubeta y al
           inicio de registro */
 a_ini_cub=i_reg=ftell(a_vb);
 fscanf(a_vb,"%*c%*c");          /* Lee el equiv. en VocBas */
 do {car=getc(a_vb);}while((car!=ini_rai)&&(car!= EOF));
 f_reg=ftell(a_vb)-1;       /* Guarda donde termina el registro */
 prox=f_reg-i_reg;     /* Se calcula a cuantos bytes está el próx. */
 fseek(a_vb,i_reg+2+8+1,0);    /* y se registra en VocBas */
 fprintf(a_vb,"%.8x",prox);
 i_reg=f_reg;ant=prox;     /* se actualiza i_reg=f_reg y ant=prox */
        /* Se obtiene su rep. numérica y la raiz */
```

```
convierte(nc1,tp1,conv1,rai1,&noson1);
valim=noson1;
do {    /* Ciclo que termina hasta EOF */
    noreg_c=1;++nocubs;  /* Actualiza */
    if (err==EOF) {          /* Si se detec. EOF, reporta y termina */
        llena();
        goto cierra;
    }
    do {                /* Ciclo hasta que encuentre la prox. raiz */
        memset(tp2,0,sizeof(palabra));     /* Inicializa */
        ii=0;    /* Lee la transcripción del sig. reg.en VocBas.st */
        do { /* hasta encontrar un tabulador o un paréntesis */
            tp2[ii]=getc(a_e);
            ++ii;
        }while((tp2[ii-1]!='    ')&&(tp2[ii-1]!='(')&&(tp2[ii-1]!=EOF));
        nc2=ii-1;
                /* Lee hasta fin de reg.o EOF */
        do {car=getc(a_e);}while((car!=finreg)&&(car!= EOF));
        err=getc(a_e);    /* Lee el próx. caracter */
        if (err!=EOF) ungetc(err,a_e);/* Si no es EOF => lo regresa */
        /* El fin de cubeta se inicializa con el valor de fin de reg. */
        a_fin_cub=f_reg;
        fseek(a_vb,i_reg+2,0);     /* Lee el equiv. en VocBas */
        do {car=getc(a_vb);}while((car!=ini_rai)&&(car!= EOF));
        f_reg=ftell(a_vb)-1;/* Guarda donde termina el registro */
        /* Se calcula a cuantos bytes está el próx. */
        prox=f_reg-i_reg;
        fseek(a_vb,(long)(i_reg+2),0); /* y se registra en VocBas */
        fprintf(a_vb,"%.8x  %.8x",ant,prox);
                /* Se obtiene su rep. numérica y la raiz */
        convierte(nc2,tp2,conv2,rai2,&noson2);
        dif=memcmp(rai1,rai2,tamrai);     /* Compara las raices */
        if (!dif) {          /* Si son iguales, continua */
                    /* Incrementa el contador de reg. por cubeta */
            ++noreg_c;
            /* Registra en VocBas que es continuación de cubeta */
            fseek(a_vb,i_reg,0);
            fprintf(a_vb,"%c",conti);
                /*   Actualiza   valim */
            valim= (valim > noson2)? noson2:valim;
            if (err==EOF) {    /* Si ya detectó EOF */
                llena();       /* reporta y termina */
```

```
                goto cierra;
            }
        }
        fseek(a_vb,f_reg,0);  /* Reposiciona al final del reg. */
                              /* se actualiza i_reg=f_reg y ant=prox */
        i_reg=f_reg;ant=prox;
    } while(!dif);
    llena();        /* Cuando detectá la prox. raiz, reporta, */
                    /* y actualiza las variables relacionadas */
    memcpy(rai1,rai2,sizeof(raiz));
    memcpy(conv1,conv2,sizeof(palabra));
    nc1=valim=nc2;
    noson1=noson2;
    a_ini_cub=a_fin_cub;
 }while(err!=EOF);
cierra:
 fclose(a_e);       /* Cuando se detecta EOF se cierran */
 fclose(a_r);       /* todos los archivos */
 fclose(a_vb);
 fclose(a_tc);
 fclose(a_at);
}

prevllena()         / *  Rutina que llena con los valores iniciales a
                         la Tabla de cubetas y a la base de datos */
{
    char    *buff;
    int     filedes;

    /* Pide memoria para buff e inicializa con cero */
 buff= (char *)calloc((unsigned int)(nocomb*2),(unsigned
int)sizeof(char));
 if (!buff) {   /* Si no hay memoria => termina la execución */
    printf("Problema al inicializar Tabla.cub\n");
    exit(-1);
 }
    /* Abre la Tabla de cubetas, destruyendo lo que tenía antes */
 filedes=open("Tabla.cub",O_WRONLY|O_CREAT|O_TRUNC);
    /* Inicializa toda la Tabla de cubetas con ceros */
  write(filedes,buff,(unsigned)(nocomb*2));
    /* Libera el espacio ocupado por buff */
  free(buff);
```

```
    /* Cierra la Tabla */
 close(filedes);
    /* Abre la base de datos, VocBas, destruyendo lo que tenía antes
* /
 a_vb=fopen("VocBas","w");
    /* Ciclo hasta encontrar el fin de archivo */
 do {
    err=fscanf(a_e,"%s",tp1); /* Lee la transcripción */
    if (err!=EOF) {     /* Si no detecta el fin de archivo, continua */
            /* Imprime la información para direccionamiento */
        fprintf(a_vb,"%c%c00000000  00000000\n",ini_rai,ini_rai);
            /* Imprime la transcripción entre diagonales */
        fprintf(a_vb,"/%s/",tp1);
/* Imprime lo demás hasta encontrar fin de registro o de archivo */
        do {
            car=getc(a_e);
            if (car!=EOF) fprintf(a_vb,"%c",car);
        }while((car!=finreg)&&(car!=EOF));
        err=car;
    }
 }while(err!=EOF);
    /* Cierra VocBas */
 fclose(a_vb);
    / *  Reposiciona el apuntador, al archivo de entrada,
         en el inicio de este */
 rewind(a_e);
}

convierte(nc,tp,conv,rai,noson)     / *  Rutina que relaciona cada
sonido con                              un valor numérico,*/
unsigned char    nc,*noson;         / *  y en rai guarda la raiz. Recibe
el nc= #                                de caracteres */
palabra      tp,conv;               / *  y la transcripción. En noson
regresa el                              # de son        */
raiz             rai;
{

 ii=jj=0;               /*  Inicializa */
                        /* ii: indica el # de caracter, jj: el noson */
 memset(conv,0,sizeof(palabra));
 memset(rai,0,sizeof(raiz));
do {                    /* Ciclo hasta revisar nc caracteres */
```

```
switch (tp[ii]) {  /* Pregunta por tp[ii] */
/* en rai sólo se almacenan como máximo tamrai caracteres */
     case 'a':
     case  'á':conv[jj]=a;if  (jj<tamrai)  rai[jj]='a';break;
     case  'b':conv[jj]=b;if  (jj<tamrai)  rai[jj]='b';break;
     /* /ch/ -> c */
     case  'c':conv[jj]=c;if  (jj<tamrai)  rai[jj]='c';++ii;break;
     case  'd':conv[jj]=d;if  (jj<tamrai)  rai[jj]='d';break;
     case 'e':
     case  'é':conv[jj]=e;if  (jj<tamrai)  rai[jj]='e';break;
     case  'f':conv[jj]=f;if  (jj<tamrai)  rai[jj]='f';break;
     case  'g':conv[jj]=g;if  (jj<tamrai)  rai[jj]='g';break;
     case 'i':
     case  'í':conv[jj]=i;if  (jj<tamrai)  rai[jj]='i';break;
     case  'k':conv[jj]=k;if  (jj<tamrai)  rai[jj]='k';break;
     case  'l':conv[jj]=l;if  (jj<tamrai)  rai[jj]='l';break;
     case  'm':conv[jj]=m;if  (jj<tamrai)  rai[jj]='m';break;
     case  'n':conv[jj]=n;if  (jj<tamrai)  rai[jj]='n';break;
     case  'ñ':conv[jj]=nn;if  (jj<tamrai)  rai[jj]='ñ';break;
     case 'o':
     case  'ó':conv[jj]=o;if  (jj<tamrai)  rai[jj]='o';break;
     case  'p':conv[jj]=p;if  (jj<tamrai)  rai[jj]='p';break;
     case  'r':if  (tp[ii+1]=='r')  {  /* Si es /rr/ -> - */
                    conv[jj]=rr;
                    if  (jj<tamrai)  rai[jj]='-';
                    ++ii;
                }
                else {
                    conv[jj]=r;
                    if  (jj<tamrai)  rai[jj]='r';
                }
                break;
     case  's':conv[jj]=s;if  (jj<tamrai)  rai[jj]='s';break;
     case  't':conv[jj]=t;if  (jj<tamrai)  rai[jj]='t';break;
     case 'u':
     case  'ú':conv[jj]=u;if  (jj<tamrai)  rai[jj]='u';break;
     case  'w':conv[jj]=w;if  (jj<tamrai)  rai[jj]='w';break;
     case  'x':conv[jj]=x;if  (jj<tamrai)  rai[jj]='x';break;
     case  'y':conv[jj]=y;if  (jj<tamrai)  rai[jj]='y';break;
     /* Si no fue ninguna de estos valores => no incrementar jj */
     default:--jj;
             break;
```

```
    }
    ++ii,++jj;
 } while(ii < nc);
  *noson=jj;
}

func_disp()       /* Rutina que calcula la función  de dispersión */
{
    unsigned char    limite;

    /* Valida que noson no sea mayor que tamrai */
  limite= (noson1 > tamrai)? tamrai:noson1;
  i_tc=0; /* Se obtiene el indice adecuado para la tabla de cubetas */
   for(ii=0,jj=limite-1;ii  <  limite;++ii,--jj)
     i_tc+=  (conv1[ii]+1)*ipower((extended)  base,jj);
   --i_tc;
}

llena()  /* Esta rutina escribe todos los resultados */
{
    unsigned char    c1,c2,noelem_r,noreg_elem_arr;
    palabra      pal1,pal2;
    char         nom[50],*buff;
    long             tamcub,i_r1,i_r2,prox,err_dir;
    int          noelem,filedes1,filedes2,dif;

/ *  noelem: no total de elementos
    noelem_r: # de elem. reales
    noreg_elem_arr: # de reg. en un elemto del arreglo
    i_r1,i_r2: indice al registro uno y al dos */

    /*  Reporta  */
 fprintf(a_r,"%d) %s, noreg= %d, valim=
%d\n",nocubs,rai1,noreg_c,valim);
 printf("%d) %s, noreg= %d, valim= %d ",nocubs,rai1,noreg_c,valim);
    /* Escribe en el archivo auxiliar a la Tabla de cubetas */
  tamcub= a_fin_cub-a_ini_cub;/* Calcula el tamaño de la cubeta */
   fprintf(a_at,"%.8x  %.8x\n",a_ini_cub,tamcub);
   printf("a_ini=%ld,  tamcub=  %ld\n",a_ini_cub,tamcub);
    /* Se calcula el valor de la función de dispersión */
  func_disp();
    /* Se convierte nocubs  a dos caracteres */
```

```
 if (nocubs<=255) {c1=0;c2=nocubs;}
 else {c1=nocubs/256;c2=nocubs%256;}
    /* Se posiciona el apuntador de la tabla de cub. */
err_dir=fseek(a_tc,(long)(i_tc*nocartc),0);
    /* Si accesa una localidad inexistente, se para la ejecución */
if (err_dir) {
    printf("Accesando una localidad indefinida. Error= %d\n",err_dir);
    exit(-1);
 }
    /* Escribe en la Tabla de cubetas */
 fprintf(a_tc,"%c%c",c1,c2);
 printf("i_tc= %ld, c1= %d, c2= %d\n",i_tc,c1,c2);
 dimarr=valim-tamrai;
    /* Se calcula la dimensión del posible arreglo */
    /* Si el no. de reg. en la cubeta revasa a crit_arr y
       la dimarr es > 0 y a lo más tamrai => continua */
 if ((noreg_c>crit_arr)&&(dimarr>0)&&(dimarr<=tamrai)) {
    /* Se crea el arreglo y su archivo auxiliar */
    /* Indica en la base que se generó un arreglo y
       que es de tal dimensión */
    fseek(a_vb,a_ini_cub,0);
    fprintf(a_vb,"%c%c",arreg,dimarr);
    /* Calcula el no. de elementos que tendrá el arreglo */
    noelem= (int) ipower((extended)base,(int)dimarr);
    printf("Se genera arreglo para %s, de %d dimensión y %d
          elementos. Clave= %d\n",rai1,dimarr,noelem,arreg);
    /* Inicializa un buffer con ceros */
    buff= (char *)calloc((unsigned int) noelem,(unsigned int)
        sizeof(char));
    /* Abre el archivo _.arr, _ depende de la raiz en cuestión */
    strcpy(nom,rai1);
    strcat(nom,".arr");
    /* Destruyendo su contenido anterior */
    filedes1=open(nom,O_WRONLY|O_CREAT|O_TRUNC);
    /* Inicializa en ceros el archivo */
    write(filedes1,buff,(unsigned) noelem);
    free(buff);        /* Libera buff */
    close(filedes1); /* Cierra .arr */
    /* Se carga la cubeta en memoria, inicializandose
       con ceros primeramente */
    a_cub= (unsigned char *)calloc((unsigned int) tamcub,
            (unsigned int) sizeof(char));
```

```
    fclose(a_vb);
    filedes2=open("VocBas",O_RDONLY);
    lseek(filedes2,a_ini_cub,0);
    read(filedes2,a_cub,(unsigned) tamcub);
    close(filedes2);
/* Abre de nuevo el .arr, pero con la opción de modificar campos */
    arr=fopen(nom,"r+");
    strcat(nom,".dir");
    /* Abre el .arr.dir, destruyendo la inf. que tenía antes */
    arrdir=fopen(nom,"w");
    i_r1=i_r2=0;noelem_r=0;          /* Inicializa */
    /* Se obtiene la dirección al sig. registro */
    sscanf(&a_cub[i_r1+11],"%x",&prox);
    i_cub=i_r1+21;
    lee_cub(pal1,conv3);          /* Lee el 1er registro en pal1 */
    do {                    /* Ciclo hasta revisar toda la cubeta */
        ++noelem_r;noreg_elem_arr=1;
        /* Ciclo hasta encontrar un nuevo elemento en el arreglo */
        do {
            i_r2+=prox;          /* Posiciona en el sig. registro */
            if (i_r2 >= tamcub) {  /* Si ya llegó al fin => reportar */
                llenarr(noelem_r,i_r1,noreg_elem_arr);
                goto e1;
            }
            /* Lee la dir. al próximo reg */
            sscanf(&a_cub[i_r2+11],"%x",&prox);
            i_cub=i_r2+21;
            lee_cub(pal2,conv4); /* Lee el sig. reg en pal2 */
            /* Compara pal1 con pal2 */
            dif=memcmp(pal1,pal2,tamrai+dimarr);
            if(!dif) ++noreg_elem_arr;
        }while(!dif);
            /* Cuando encuentra uno dif. => reporta */
        llenarr(noelem_r,i_r1,noreg_elem_arr);
        i_r1=i_r2;  /* Intercambio de registros */
        memcpy(pal1,pal2,tamrai+dimarr);
        memcpy(conv3,conv4,tamrai+dimarr);
    }while(1);
e1:
    free(a_cub);      /* Libera el espacio ocupado por la cubeta */
    fclose(arr);      /* Cierra los archivos .arr, y .arr.dir */
    fclose(arrdir);
```

```
                        /* Abre la base en la modalidad que tenía */
    a_vb=fopen("VocBas","r+");
 }
}

lee_cub(pal,conv)       / *  Rutina que lee de la cubeta y relaciona
                             cada sonido */
palabra pal,conv;  /*con un valor numérico, y en pal su transcripción*/
{
  jj=0;
  memset(pal,0,sizeof(palabra));
  memset(conv,0,sizeof(palabra));
 do {          /* Ciclo hasta revisar tamrai+dimarr sonidos */
     switch (a_cub[i_cub]) {
          case 'a':
          case  'á':conv[jj]=a;pal[jj]='a';break;
          case  'b':conv[jj]=b;pal[jj]='b';break;
          case  'c':conv[jj]=c;pal[jj]='c';++i_cub;break;
          case  'd':conv[jj]=d;pal[jj]='d';break;
          case 'e':
          case  'é':conv[jj]=e;pal[jj]='e';break;
          case  'f':conv[jj]=f;pal[jj]='f';break;
          case  'g':conv[jj]=g;pal[jj]='g';break;
          case 'i':
          case  'í':conv[jj]=i;pal[jj]='i';break;
          case  'k':conv[jj]=k;pal[jj]='k';break;
          case  'l':conv[jj]=l;pal[jj]='l';break;
          case  'm':conv[jj]=m;pal[jj]='m';break;
          case  'n':conv[jj]=n;pal[jj]='n';break;
          case  'ñ':conv[jj]=nn;pal[jj]='ñ';break;
          case 'o':
          case  'ó':conv[jj]=o;pal[jj]='o';break;
          case  'p':conv[jj]=p;pal[jj]='p';break;
          case 'r':if (a_cub[i_cub+1]=='r') {
                        conv[jj]=rr;
                        pal[jj]='-';
                        ++i_cub;
                   }
                   else {
                        conv[jj]=r;
                        pal[jj]='r';
                   }
```

```
                    break;
            case  's':conv[jj]=s;pal[jj]='s';break;
            case  't':conv[jj]=t;pal[jj]='t';break;
            case 'u':
            case  'ú':conv[jj]=u;pal[jj]='u';break;
            case  'w':conv[jj]=w;pal[jj]='w';break;
            case  'x':conv[jj]=x;pal[jj]='x';break;
            case  'y':conv[jj]=y;pal[jj]='y';break;
            default:--jj; /* Se quita el incremento */
                    break;
        }
        ++i_cub,++jj;
    } while(jj < tamrai+dimarr);
}

llenarr(noelem_r,i_r1,noreg_elem_arr)
/ *  Rutina que escribe en el .arr y arr.dir */
unsigned char     noelem_r,noreg_elem_arr;
long              i_r1;
{
    unsigned char     ii,jj;
    long              i_ar;

    i_ar=0;                 /* Calcula el índice en el arreglo */
    for(ii=0,jj=dimarr-1;ii  <  dimarr;++ii,--jj)
        i_ar+= (long) (conv3[tamrai+ii]*ipower((extended)base,jj));
    fseek(arr,i_ar,0);
    fprintf(arr,"%c",noelem_r);     /* Escribe en el .arr */
    fprintf(arrdir,"%.8x\n",a_ini_cub+i_r1);/* Escribe en el .arr.dir */
    printf("i_ar= %ld, no_elem= %d, dir= %ld, noreg_elem= %d\n",
        i_ar,noelem_r,a_ini_cub+i_r1,noreg_elem_arr);
}
```

B.9 Código del programa EstEsp.c

```
/ *  Mediante este programa se analizan las transcripciones de
     las palabras existentes en VocBas,y se obtiene tres listas:
     grupos cons. que se presentan al inicio, a mediación, y
     final de palabra. Además se reporta el número de palabras
     analizado, la longitud de palabra máxima, y se lleva un
     conteo sobre el número de palabras de cada long. diferente.    * /

#include <stdio.h>
#define max 25

int             longpal,lpr,nopal[max],longmax,nototpal,err;
FILE            *rep,*ini,*inte,*fin;
char            pv,sv,i,banf;
unsigned char   buff[max];

    /* Declaración de procedimientos presentes en el archivo */
int  cuentason(),replim();
int  buscapat(),reppatint();

main() /** Programa Principal **/
{
    /* Abre archivos */
    /* Contiene las transcripciones de cada palabra */
 rep=fopen("VocBas.rep","r");
    /* Contendrá los grupos de cons. que inician palabra */
 ini=fopen("cons.ini","w");
    /* Contendrá los g. cons. que se presentan dentro de palabra */
 inte=fopen("cons.int","w");
    /* Contendrá los g. cons. que terminan palabra */
 fin=fopen("cons.fin","w");
    /* Inicializa variables */
 longmax=nototpal=0;
 memset(nopal,0,max);

 /* Ciclo para analizar cada pal. hasta detectar el fin de archivo */
 do {
    memset(buff,0,max); /* Se inicializa el buffer */
    /* Se lee la sig. palabra y se almacena en buffer */
```

```
    err=fscanf(rep,"%s",buff);
    if (err!=EOF) {     /* Si no encontró el fin de archivo, continúa */
        ++nototpal;  /* Incrementa el contador total de palabras */
                     /* Se obtiene el número de grafías de la pal. */
        longpal=strlen(buff);
        lpr=cuentason(); /* Se calcula el no. de sonidos */
        ++nopal[lpr];       /* Se lleva la cuenta del no. de pal.
                               que contengan la misma long. */
                     /* En longmax se guarda la longitud máxima */
        longmax= (longmax < lpr)? lpr:longmax;
/* Se reportan los límites, es decir las cons. iniciales y finales */
        replim();
            /* Se reportan las cons. intermedias */
        reppatint();
        printf("%d\n",nototpal);
    }
 }while(err!=EOF);
 /* Al encontrar el fin de archivo se reporta el no. de pal. existentes
    de acuerdo a su long., el no. total de palabras, y la long. máxima de
    palabra */
 printf("\nRESULTADOS\n");
 printf("No. Total de palabras con difrentes long.\n");
 printf("long.      # pal.\n");
 for (i=0;i< max;++i) {
    printf("%d    %d\n",i,nopal[i]);
 }
 printf("nototpal= %d, longmax= %d\n",nototpal,longmax);
    /* Cierra los archivos */
 fclose(rep);
 fclose(ini);
 fclose(inte);
 fclose(fin);
}

cuentason()       /* Rutina que cuenta el número de sonidos */
{
    char     i,j;

 i=j=0;
 do {
    switch (buff[i]) {
        /* /ch/ usa dos grafías pero representa 1 sonido */
```

```
            case 'c':if (buff[i+1]=='h') ++i;break;
            /* /rr/ usa dos grafías pero representa 1 sonido */
            case 'r':if (buff[i+1]=='r') ++i;break;
            / *  el resto de los fonemas utilizan 1 grafía para
                 representar el sonido */
            default:break;
        }
        ++i;++j;
    }while(i<longpal);
    return((int)j);
}

replim()      /* Rutina que reporta las consonantes límites */
{
    char      i,j,ban;

 i=ban=0;
 do {         /* Se busca la primera vocal */
    switch  (buff[i])  {
            case 'a':
            case 'á':
            case 'e':
            case 'é':
            case 'i':
            case 'í':
            case 'o':
            case 'ó':
            case 'u':
            case 'ú':ban=1;break;
            default:break;
    }
    ++i;
 }while((!ban)&&(i<longpal));

 /* Se reporta los sonidos encontrados hasta antes de la 1ª vocal */
 /* Si no se detectan consonantes no se imprime nada */

 ban=0;
  for  (j=0;j<i-1;++j)  {fprintf(ini,"%c",buff[j]);ban=1;}
     /* Si reportó algo, separa la combinación con retorno */
  if  (ban)  fprintf(ini,"\n");
  ban=0;i=longpal-1;
```

```
do {     /* Se busca la última vocal */
    switch (buff[i]) {
         case 'a':
         case 'á':
         case 'e':
         case 'é':
         case 'i':
         case 'í':
         case 'o':
         case 'ó':
         case 'u':
         case 'ú':ban=1;break;
         default:break;
    }
    --i;
 }while((!ban)&&(i>=0));

 /* Se reporta los sonidos encontrados después de la última vocal */
 /* Si no se detectan consonantes no se imprime nada */

 ban=0;
  for  (j=i+2;j<longpal;++j)  {fprintf(fin,"%c",buff[j]);ban=1;}
     /* Si reportó algo, separa la combinación con retorno */
  if  (ban)  fprintf(fin,"\n");
}

buscapat(pv,sv)   / *  Rutina que busca algún patrón de consonantes
                       (enmarcado entre vocales) */
char     pv,*sv;       /* Recibe la posición de la 1ª vocal,
                          y regresa la posición de la sig. vocal */
{
     char     i,ban;

                   /* Se analiza a partir del sig. sonido a la 1ª vocal */
 i=pv+1;ban=banf=0;
 do {                  /* Se busca la segunda vocal, */
     switch  (buff[i])  { /* si se detecta 1º el fin de palabra, se */
          case 'a':       /* habilita la bandera de fin (banfin)     */
          case 'á':
          case 'e':
          case 'é':
          case 'i':
```

```
            case 'í':
            case 'o':
            case 'ó':
            case 'u':
            case 'ú':ban=1;*sv=i;break;
            case  0 :ban=1;*sv=i-1;banf=1;break;
            default:break;
        }
        ++i;
    }while((!ban)&&(i<=longpal));
}

reppatint()                 / *  Reporta los patrones de cons.
                                 que se dan dentro de palabra */
{
    char     i,j,ban;

 i=ban=0;
 do {                           /* Se busca la 1ª vocal */
    switch  (buff[i])  {
            case 'a':
            case 'á':
            case 'e':
            case 'é':
            case 'i':
            case 'í':
            case 'o':
            case 'ó':
            case 'u':
            case 'ú':ban=1;pv=i;break;
            default:break;
    }
    ++i;
  }while((!ban)&&(i<longpal));

  /* Se obtienen todos los grupos consonánticos */

  do {     /* Ciclo que se termina cuando detecta el fin de palabra */
     buscapat(pv,&sv);      / *  Llama a buscapat para
                                 obtener la sig. vocal */
              /* Si hay más de una consonante (vcv, etc.), continua */
     if ((sv-pv) > 2) {
```

```
            j=pv+1;
            do {
        /* Si son exclusivamente /ch/ o /rr/, entonces no reportar */
                if ( ((sv-pv)==3) && (( (buff[j]=='c')&&(buff[j+1]=='h') )
                    ||( (buff[j]=='r')&&(buff[j+1]=='r') )) ){
                    ++j;
                    ban=0;
                }
                else {   /* En caso contrario, imprimir */
                    fprintf(inte,"%c",buff[j]);
                    ban=1;
                }
                ++j;
            }while (j < sv);
        /* Si reportó algo, entonces separa la comb. con un retorno */
            if (ban) {
                fprintf(inte,"\n");
            }
        }
        pv=sv;  /* Ahora, la 1ª vocal es la siguiente vocal */
    }while(!banf);
}
```

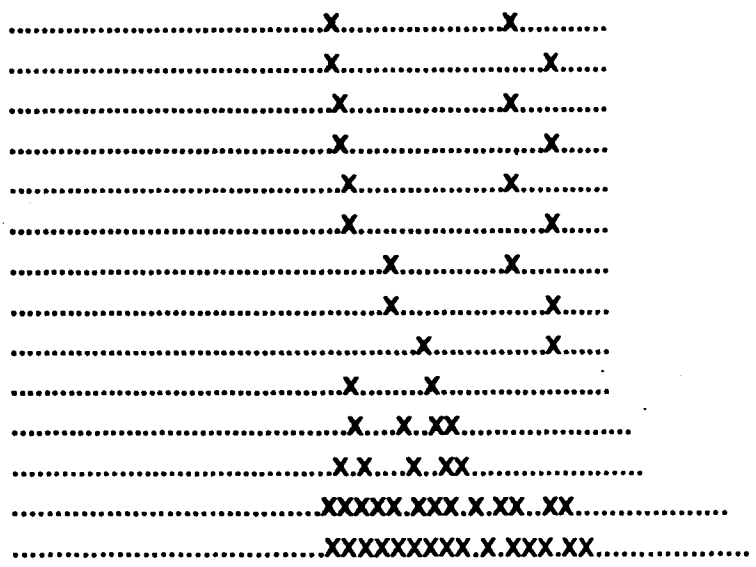
B.10 Red 1

```
/ *    red1.net: Continene las combinaciones
                 de dos consonantes */
definitions:
nunits 58
ninputs 36
noutputs 3
nepochs 25
ecrit 0.04
end
constraints:
u 5.0
v -5.0
x 10.0
y -10.0
z -15.0
end
network:
% 36 19 0 36
x.....x.x..x.xx.x.
..x.....x....x.x...x.....
...x.....x....x.....
....x.....xxx..x.....
.....x.....x...x.x..x.xx...
......x.....x.xxxx.x..x.xxx...
.......x.....x.....x.x.....
........x.....xxxxxxxxx...xxx.x.
.........x.....x....x.xx...
..........x.....xxxxxxxxx.x..xx.x.
...........x....x.xxxxxxx.x..xx...
............x.....xx...x.x....
x.............xxxxxx.x..x.x....x
..x.............xxxx.x.x.x.xxx.xx
......x.....x....x.x.....xx
........x.....x.....x
..........x.....x....x
............x.....x.....x...xx
..............x.xxxxxxxxx.x.xxx.xx
% 55 3 36 19
```

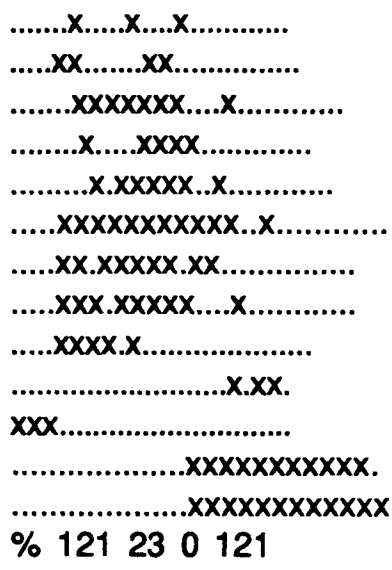
xxxxxxxxxxxx.....
.....xxxxxxx
yyyyyyyyyyyyyyyyyyy
end
biases:
% 36 22
zzzzzzzzzzzzzzzzzzzvvu
end

B.11 Red 2

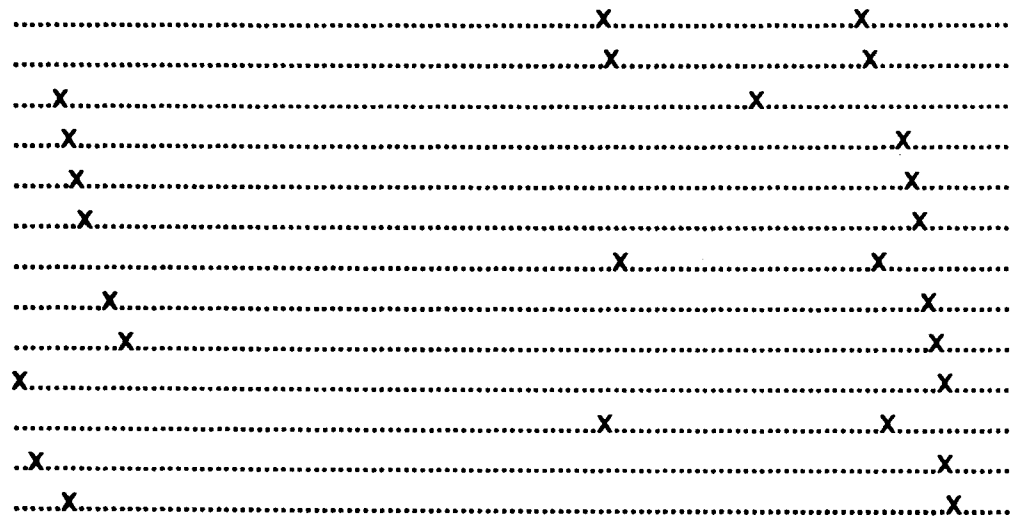
```
/ *  red2.net: Contiene las combinaciones
                de tres y cuatro consonantes*/
definitions:
nunits 149
ninputs 72
noutputs 5
end
constraints:
u 5
v -5
x 10
y -10
z -15
end
network:
% 72 36 0 72
.x.....x.....
....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
```



% 108 13 72 31



% 121 23 0 121



.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x.....
.....x.....x...
.....x.....x..
.....x.....x.
.....x.....x
% 144 5 121 23
xxxxxxxxx.....xx..
.....xxxxxxxxxx..xx
yyyyyyyyyyyyyyyyyyyyyy
.....x.xxx.x.xx....
.....x...x.x....xx
end
biases:
% 72 77
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzvvvvvvvvvvvvvvvvvvvvzzzzzzzzzzzzzzz
zzzzzzzzzvvuvv
end

Referencias

Aho, A. V., Hopcroft, J. E., y Ullman, J. D. (1988). *Estructuras de datos y algoritmos* (A. Vargas V., y J. Lozano M., Tr.). México, DF: Sistemas Técnicos de Edición.

Alarcos Ll., E. (1965). *Fonología española* (4ª ed. rev.). Madrid: Gredos.

Alexandres, S., Morán, J., Carazo, J., y Santos, A. (1990). Parallel Architecture for Real-Time Speech Recognition in Spanish. *Proceedings, ICASSP 90, 1990 International Conference on Acoustics, Speech, and Signal Processing, 2*, pp. 977-980.

Allen, J. (1976). Synthesis of Speech form Unrestricted Text. *Proceedings of the IEEE, 64*, 433-443.

Badii, A., Binstead, M. J., Jones, A. J., Stonham, T. J., y Valenzuela, Ch. L. (1989). Speech recognition based on topology preserving neural maps. En I. Aleksander (Ed.), *Neural Computing Architectures: The design of brainlike machines* [Arquitecturas de Computadoras Neurales: El diseño de máqinas similares al cerebro] (pp. 173-216). Cambridge, MA: Massachusetts Institute of Technology.

Bello, A. (1951). *Estudios Gramaticales* En R. Caldera (Ed.). Caracas: Ministerio de Educación.

Carrijo G., y Attikiouzel Y. (1990). Application of Artificial Networks for vowel, isolated digit and word recognition in Portuguese. En F. J. Cantú y H. Terashima (Eds.). *Proceedings of the Third International Symposium on Artificial Intelligence Proceedings* [La aplicación de redes artificiales para el reconocimiento de voz de vocales, digitos aislados y palabras en portugués] (pp. 188-193). México: Limusa.

Cerdà M., R., Muñoz O., M. del C., López de A., J. L., y Lloret C., J. (1986). En E. Fontanillo M. *Diccionario de Lingüística*. Madrid: E. G. Anaya.

Crystal, D. (1987). *The Cambridge Encyclopedia of Lenguage* [La enciclopedia Cambridge de lenguaje]. New York: Cambridge University.

Dubois, J., Giacomo, M., Guespin, L., Marcellesi, Ch., Marcellesi, J., y Mével, J. (1983). *Diccionario de Lingüística* (I. Ortega y A. Domínguez, Tr.). Madrid: Alianza Editorial.

Espinosa E., I. (1988). Neurocomputadoras y Neurofisiología: Presectivas. *1988, 30 Aniversario de la computación en México 2. Congreso Nacional 3. Pasado, presente y futuro de la computación, 2*, 1479-1498.

Figueroa N., J. , y Vargas Medina, E. (1988). Neurocomputadoras y Neurofisiología: Presectivas. *1988, 30 Aniversario de la computación en México 2. Congreso Nacional 3. Pasado, presente y futuro de la computación, 2*, 1479-1498.

Flanagan, J. L. (1976). Computers that Talk and Listen: Man-Machine Communication by Voice. *Proceedings of the IEEE, 64*, 405-415.

Fuori, W. M., y Aufiero, L. J. (1986). *Computers and information processing.* Englewood Cliffs, NJ: Prentice-Hall.

Gaona, F. L. (1949). *Algunas consideraciones sobre la enseñanza de los sonidos de la lengua española.* Tesis de maestría no publicada, Universidad Nacional Autónoma de México, México, D. F.

García H., M. I. (1988). De la teoría a la aplicación de estructuras arborescentes. *1988, 30 Aniversario de la computación en México 2. Congreso Nacional 3. Pasado, presente y futuro de la computación, 2*, 9571016.

García-Pelayo y G. , R. (1972). *Pequeño LAROUSSE en color.* México, DF: Larousse.

García-Pelayo y G., R., García-Pelayo y G., F., y Durand, M. (1983). *Larousse de la conjugación.* México, DF: Patria.

Gasser, M., y Lee, Ch. (1990a). *Networks that Learn Phonology.* Manuscrito no publicado. Indiana University, Bloomington, IN.

Gasser, M., y Lee, Ch. (1990b). *Networks and Morphophonemic Rules Revisited.* Manuscrito no publicado. Indiana University, Bloomington, IN.

Gili G., S. (1978). *Elementos de Fonética General* (5ª ed. rev.). Madrid: Gredos.

González P., C. (1975). *Manual de gramática castellana* (42ª ed. rev.). México, DF: Patria.

Guyton, A. C. (1975). *Fisiología Humana* (4ª ed., A. Folch y Pi, Tr.). México, DF: Interamericana. (Trabajo original publicado en 1974)

Jiménez M., A. (1988). Analizador determístico para el español. *1988, 30 Aniversario de la computación en México 2. Congreso Nacional 3. Pasado, presente y futuro de la computación, 2*, 943-956.

Juilland, A., y Chang-Rodríguez, E. (1964). *Frequency Dictionary of Spanish Words* [Diccionario de las frecuencias de las palabras en Español] . The Hague, The Netherlands: Mounton.

Hagège, C. (1981). *La gramática generativa. 2. Reflexiones críticas* (G. Ter-Sakarian, Tr.). Madrid: Gredos. (Trabajo original publicado en 1976)

Harris, J. W. (1975). *Fonología Generativa del Español* (A. Verde, Tr.). Barcelona: Planeta.

Heffner, R.-M. S. (1950). *General Phonetics* [Fonética General]. Madison: University of Wisconsin.

Holland, R. C. (1986). *Integrated Circuits and Microprocessors* [Circuitos integrados y microprocesadores]. Gran Bretaña: Pergamon Press.

Kohonen T. (1989). Speech recognition based on topology preserving neural maps. En I. Aleksander (Ed.), *Neural Computing Architectures: The design of brain-like machines* [Arquitecturas de :El diseño de máqinas similares al cerebro] (pp. 26-40). Cambridge, MA: Massachusetts Institute of Technology.

Lara, L. F. (1986). *Diccionario básico del Español en México*. México, DF: El Colegio de México.

Lee, K.-F., Hon, H.-W., y Reddy, R. (1990). An Overview of the SPHINX Speech Recognition System. *IEEE Transactions on Acoustics, Speech, and Signal Processing, 38*, 35-45.

Lee, L.-S., Tseng, Ch.-Y., Gu, H.-Y., Liu, F. H., Chang, C. H., Hsieh, S. H., y Chen, C. H. (1990). A real-time Mandarin dictation machine for chinese language with unlimited texts a very large vocabulary. *Proceedings, ICASSP 90, 1990 International Conference on Acoustics, Speech, and Signal Processing, 1*, pp. 65-68.

Lenning, M. (1990, Agosto). Putting Speech Recognition to Work in the Telephone Network. *IEEE Computer*, pp. 35-41.

Lynn, D. K., Meyer, Ch. S., y Hamilton, D. J. (Eds.). (1967). *Analysis and Design of Integrated Circuits* [Análisis y diseño de circuitos integrados] (The Engineering Staff, Motorola Inc. Semiconductor Products Division). New York: McGraw-Hill.

Martin, T. B. (1976). Practical Applications of Voice Input to Machines. *Proceedings of the IEEE, 64*, 487-501.

Martínez y M., A. (1975). *Curso de pronunciación del español para alumnos extranjeros*. Monterrey: Instituto Tecnológico y de Estudios Superiores de Monterrey.

Mateos M., A. (1980). *Compendio de las etimologías grecolatinas del español* (17ª ed.). México, D. F.: Esfinge.

Navarro, T. (1957). *Manual de pronunciación española* (5ª ed. rev.). New York: Hafner.

McClelland, J. L., y Rumelhart, D. E. (1988). *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises* [Exploraciones en procesamiento distribuido paralelo: Un libro de trabajo de modelos, programas y ejercicios]. Cambridge, MA: Massachusetts Institute of Technology.

McClelland, J. L., Rumelhart, D. E., y PDP Reserarch Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (Vol. 2: Psychological and Biological Models) [Procesamiento distribuido paralelo: Exploraciones en la microestructura del conocimiento]. Cambridge, MA: Massachusetts Institute of Technology.

McGehee, F. (1937). The reliability of the identification of the human voice. *Journal of General Psychology, 17*, 249-271.

Peacocke, R. D., y Graf, D. H. (1990, Agosto). An Introduction to Speech and Speaker Recognition. *IEEE Computer*, pp. 26-33.

Rabiner, L. R. (Ed.). (1976). Special issue on man-machine comunication by voice [Scanning the Issue]. *Proceedings of the IEEE, 64*, 403-404.

Rabiner, L. R., Juang, B.-H., Levinson, S. E., y Sondhi, M. M. (1985). Recognition of Isolated Digits Using Hidden Markov Models With Continuous Mixture Densities. *AT&T Technical Journal*, 64, 1211-1233.

Reader's Digest. (1979). *Gran Diccionario Enciclopédico Ilustrado* (27ª ed., Vol. 7). México, DF: Autor.

Real Academia Española. (1973). *Esbozo de una nueva gramática de la lengua española*. Madrid: Espasa-Calpe.

Real Academia Española. (1984). *Diccionario de la Lengua Española* (20ª ed.). Madrid: Autor.

Revilla de C., S. (1990). *Gramática Española Moderna: Un nuevo enfoque*. (2ª ed.). México, DF: McGraw-Hill.

Rivero, M. L. (1977). *Estudios de Gramática Generativa del Español*. Madrid: Cátedra.

Roca-Pons, J. (1975). *El lenguaje* (2ª ed.). Barcelona: Teide.

Rodríguez B., I. (1952). *Recuento de Vocabulario Español*. EUA: OEA y UNESCO.

Rumelhart, D. E., McClelland, J. L., y PDP Reserarch Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (Vol. 1: Foundations) [Procesamiento distribuido paralelo: Exploraciones en la microestructura del conocimiento]. Cambridge, MA: Massachusetts Institute of Technology.

Sánchez S., L. (1986, Septiembre-Octubre). Modelos auditivos por computadora. *Ciencia y Desarrollo*, pp. 92-99.

Southworth, F. C., y Daswani, Ch. J. (1974). *Foundations of Linguistics* [Fundamentos de Lingüística]. New York: The Free Press.

Tapia, R. (1987). *Las células de la mente*. México, DF: Fondo de Cultura Económica.