Programación Aplicada y Lab.

PRÁCTICA No. 2

César Mauricio Arellano Velásquez

Raúl González Portillo

Allan Jair Escamilla Hernández

Este documento presenta la resolución de un programa, que a través del uso del algoritmo de Huffman, permite introducir símbolos con su probabilidad de aparición, para poder generar un árbol con el que se pueda obtener códigos de compresión para cada letra.

Tras generar los códigos, se podrá codificar y decodificar cualquier mensaje que este contenido en los símbolos de la lista dinámica.

1. INTRODUCCIÓN

En esta práctica se presenta un programa capaz de generar códigos de Huffman para realizar una comprensión en la codificación de los símbolos que el usuario desee introducir.

Así lograr una compresión en los bits que el programa pueda generar a lo largo de acuerdo al tipo de mensaje.

**Explicación del algoritmo de Huffman:**

El proceso de asignación de códigos se lleva a cabo mediante la generación de un árbol binario, desde las hojas hacia la raíz, de tal modo que los nodos hoja son los símbolos del alfabeto. En la construcción del árbol, los nodos menos probables se unen sucesivamente para formar otro nodo de mayor probabilidad, de forma que cada uno de los enlaces añade un bit al código de los símbolos que estamos juntando. Este proceso termina cuando sólo se dispone de un nodo, de forma que éste representa la raíz del árbol, es decir cuando la suma de probabilidad sea del 100%.

1. ANÁLISIS

Para la resolución del problema, se aplicaron los conocimientos visto en clase como: listas dinámicas, árboles binarios, punteros, asignación dinámica de memoria y manejo de archivos de texto.

*Función Principal*

En la función principal como tal se encuentran las llamadas a las funciones correspondientes (dichas funciones se mencionan a lo largo del documento) y el menú de opciones, el usuario determina qué opción utilizar de las diez disponibles.

Menu:

1.- Introducir Símbolo.

2.- Listar Símbolos.

3.- Borrar Símbolo.

4.- Modificar Símbolo.

5.- Guardar Símbolos/prob en archivo.

6.- Leer Símbolos/prob en archivo.

7- Generar códigos.

8.- Codificar mensaje.

9.- Decodificar mensaje.

10.- Salir.

1. Créditos.

En la función, no se le declaran parámetros de entrada, ni de salida, ya que ésta sólo presenta los créditos del programa, es decir, los nombres de los desarrolladores, y el objetivo general del programa.

1. Introducir\_simbolo.

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Procesos** | **Salida** |
| Símbolo  Probabilidad | En esta función se inserta el símbolo y su probabilidad a la lista dinámica, en esta también se verifica si es que el símbolo existe o no. | Inicio |

Con esta función a partir de los parámetros de entrada se crea espacio en memoria para poder insertarlos en la lista dinámica, sin embargo antes se verifica que este símbolo no haya sido insertado anteriormente, tras haber verificado esto procede a insertar en inicializar banderas.

1. Listar\_simbolos.

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Procesos** | **Salida** |
| Inicio | Desplegar información |  |

Esta función se encarga de imprimir los símbolos, su probabilidad de todos los elementos que hay en la lista.

1. Borrar\_simbolo.

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Procesos** | **Salida** |
| Símbolo | Borra el nodo de la lista que se le indicó. | Inicio |

Para dicha función se utiliza el parámetro símbolo para buscarlo en la lista dinámica, si lo encuentra elimina el nodo y liga el anterior con el siguiente elemento del nodo actual, sino lo encuentra imprime un mensaje diciéndole al usuario que no existe ese símbolo en la lista.

1. Modificar\_simbolo.

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Procesos** | **Salida** |
| Símbolo  Nuevo | Busca en lista símbolo  Modifica su valor de probabilidad. | Inicio |

Está función permite buscar el símbolo en la lista dinámica, y si es que lo encuentra procede a modificar el valor de la probabilidad que este tiene, sino imprime un mensaje de error.

1. Guardar\_simbolos\_Arch.

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Procesos** | **Salida** |
| Inicio | Abre archivo a guardar información.  Imprime en archivo  Cierra archivo. |  |

Esta función le pide al usuario el archivo a guardar la información, abre o crea el archivo en modo escritura y procede a imprimir en él, lo haya en la lista dinámica.

1. Leer\_simbolos\_Arch.

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Procesos** | **Salida** |
|  | Abre archivo  Lee datos de archivo  Cierra archivo | Inicio |

Para esta función el archivo verifica que el archivo a leer exista, si existe lee los datos y los carga en la lista dinámica, sino le dice al usuario que no existe tal archivo.

1. Valida\_Generacion

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Procesos** | **Salida** |
| Inicio | Recorrer lista de elementos  Sumar porcentajes de elementos | Validacion |

Antes de generar el árbol, es necesario verificar que la suma de los porcentajes de sus elementos sea 100.

Esta función recorrerá la lista a la vez que suma los porcentajes de sus elementos, una vez que llegue al final de la lista, la función revisará que el resultado de la suma sea 100 y regresará la flag de Validación a donde fue llamada.

1. Generar\_Arbol

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Procesos** | **Salida** |
| Inicio | Recorrer lista  Guardar dos valores menores  Cambiar estado de elementos  Crear nuevos nodos  Encadenar nuevos nodos | Raiz |

Esta función es la encargada de generar el árbol binario, y necesita de los elementos en la lista dinámica así como las flags que estas contienen.

La función recorre la lista en busca de dos valores menores que no hayan sido utilizados, una vez que los encuentre cambiará la flag de su estado y los encadenara a un tercer nodo que tendrá como valor la suma de los porcentajes de sus 2 nodos hijos.

Una vez que el nodo en el que estemos ubicados sea igual a 100 la función se detendrá.

1. Codigo

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Procesos** | **Salida** |
| Temp  Raiz | Recorrer árbol  Guardar valores por los que pase | Codigotemp |

Código es una función recursiva la cual recibe el elemento de la lista el cual contiene el valor al que se le quiere calcular el código.

La función recorrerá el árbol hasta dar con el símbolo que hay en Temp, cada que se avance hacia uno u otro lado, se va a guardar en un arreglo el valor correspondiente al paso que se dió, los valores no utilizados serán desechados automáticamente ya que el arreglo también se guardará de manera recursiva

1. Codificar

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Procesos** | **Salida** |
| Inicio,  Archivo 1, Archivo 2 | Abrir archivos  Leer linea  Recorrer lista  Imprimir código en archivo |  |

Esta función recorrerá el archivo de origen hasta terminar, cada línea del archivo se guardará en una cadena temporal la cual va a ser recorrida letra por letra, una vez encontrada la letra a codificar, la función escribirá el código correspondiente en el archivo de destino

1. Decodificar

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Procesos** | **Salida** |
| Raiz,  Archivo1, Archivo2 | Abrir archivos  Llamar a decode  Imprimir letra decodifcada en archivo | decodificado |

Esta función abre el archivo de origen, lee su contenido, lo decodifica con la función decode y guarda el mensaje decodificado en una cadena temporal, abre el archivo de destino y escribe en el nuevo archivo el contenido de la cadena temporal

1. Decode

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Procesos** | **Salida** |
| Raiz,  linea | Recorrer el árbol  Guardar valor decodificado en cadena temporal  Llamar a decode | decodificado |

Decode es la parte recursiva de decodificar, esta recorre el árbol en las direcciones que le indique el archivo de origen, una vez que se encuentre un símbolo en el árbol, se agrega a la cadena temporal con el mensaje decodificado.

1. Reiniciar\_Lista

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Procesos** | **Salida** |
| Inicio | Modifica el valor del estado de los símbolos |  |

Esta función sirve para modificar los valores de estado que tiene cada símbolo a 0.

1. Borrar\_Lista

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Procesos** | **Salida** |
| Inicio | Borra el contenido que hay en la lista. |  |

En esta función permite liberar espacio en memoria que hay en una lista dinámica.

1. Borrar\_Arbol

|  |  |  |
| --- | --- | --- |
| **Entrada** | **Procesos** | **Salida** |
| Raiz | Borra el contenido que hay en la árbol. |  |

Esta función permite borrar el espacio en memoria generado por el árbol.

1. DISEÑO

**Diagrama IPO (Anexo en Página 13).**

**Principal**

{

Creditos();

Repite

{

Imprimir("Menú Codificación de Símbolos");

Imprimir("1.- Introducir símbolo");

Imprimir("2.- Listar símbolos");

Imprimir("3.- Borrar símbolo");

Imprimir("4.- Modificar símbolo");

Imprimir("5.- Guardar símbolos / prob en archivo");

Imprimir("6.- Leer símbolos / prob en archivo");

Imprimir("7.- Generar códigos");

Imprimir("8.- Codificar mensaje");

Imprimir("9.- Decodificar mensaje");

Imprimir("10.- Salir");

Imprimir("Escoge un opción");

Leer(Opcion);

Selección(Opcion)

{

Caso 1:

Imprimir("Ingresa el símbolo");

Leer(Simbolo);

Imprimir("Ingresa su probabilidad");

Leer(Probabilidad);

Introducir\_Simbolo(Simbolo, Probabilidad | Inicio);

Imprimir("Se ingresó exitosamente el símbolo");

Romper;

Caso 2:

Si(↑Inicio<>NULL)

{

Imprimir("Los símbolos (junto con su probabilidad) en lista son:");

Listar\_Simbolos(↑Inicio | );

}

Si no

{

Imprimir("No hay símbolos para imprimir en la lista");

}

Romper;

Caso 3:

Si(↑Inicio <> NULL)

{

Imprimir("Ingrese el símbolo a eliminar");

Leer(Simbolo);

Borrar\_Simbolo(Simbolo | ↑↑Inicio);

}

Si no

{

Imprimir("No hay símbolos para eliminar en la lista");

}

Romper;

Caso 4:

Si(↑Inicio<>NULL)

{

Imprimir("Ingrese el símbolo a modificar su probabilidad");

Leer(Simbolo);

Imprimir("Ingrese la nueva probabilidad del símbolo");

Leer(Nuevo);

Modificar\_Simbolo(Simbolo, Nuevo | ↑↑Inicio);

}

Si no

{

Imprimir("No hay símbolos para modificar en la lista");

}

Romper;

Caso 5:

Si(↑Inicio<>NULL)

{

Guardar\_Simbolo\_Arch(↑Inicio | );

}

Si no

{

Imprimir("No se puede guardar, ya que no hay elementos en la lista");

}

Romper;

Caso 6:

Leer\_Simbolos\_Arch(↑Inicio | );

Romper;

Caso 7:

if (Inicio <> NULL)

{

Valida\_Generacion (↑Inicio | Validación)

Si (Validacion = 1)

{

Si (↑Raiz <> NULL)

{

Borrar\_Arbol (↑Raiz | );

}

Generar\_Arbol (↑Inicio | ↑Raiz);

Mientras (Temp <> NULL)

{

Codigo (Temp, ↑Raiz | CodigoTemp);

↑Temp.Codigo = CodigoTemp;

Temp = ↑Temp.sig;

}

}

Si no

{

Imprimir ("La suma de los porcentajes de los elementos en la lista actual no es igual a 100");

Imprimir ("Intente modificar, agregar o eliminar elementos en su lista");

}

}

si no

{

Imprimir ("No es posible generar un árbol con una lista vacía");

}

Romper;

Caso 8:

Imprimir ("Ingrese el nombre del archivo de origen");

Leer (NombreArchivo1);

Imprimir ("Ingrese el nombre del archivo de destino");

Leer (NombreArchivo2);

Abrir (Archivo1);

Abrir (Archivo2);

Si ((Archivo1 = NULL OR Archivo2 = NULL))

{

Imprimir ("Hubo un error al abrir los archivos");

}

Si no

{

Codificar (↑Inicio, Archivo1, Archivo2 | );

}

Romper;

Caso 9:

Imprimir ("Ingrese el nombre del archivo de origen");

Leer (NombreArchivo1);

Imprimir ("Ingrese el nombre del archivo de destino");

Leer (NombreArchivo2);

Abrir (Archivo1);

Abrir (Archivo2);

Si (Archivo1 = NULL OR Archivo2 = NULL)

{

Imprimir ("Hubo un error al abrir los archivos");

}

Si no

{

decodificar (Raiz, Archivo1, Archivo2 | decodificado);

}

Romper;

Caso 10:

Salir=1;

Romper;

PorDefecto:

Imprimir ("Ha elegido una opción inválida");

Romper;

}

}

Mientras(Salir=0);

Borrar\_Lista(↑Inicio | );

}

Introducir\_Simbolo(Simbolo, Probabilidad | ↑Inicio)

{

Verificar=0;

Temp=↑Inicio;

Mientras(Temp<>NULL AND Verificar=0)

{

Si(↑Temp.simb=Simbolo)

{

Verificar=0;

}

Si no

{

Verificar=1;

}

}

Si(Verificar=1)

{

New (Temp2);

↑Temp2.simb=Simbolo;

↑Temp2.prob=Probabilidad;

↑Temp2.codigo = "\0";

↑Temp2.status = 0;

↑Temp2.tipo = 0;

↑Temp2.sig=NULL;

Si(↑Inicio<>NULL)

{

Temp3=↑Inicio;

Mientras(↑Temp3.sig<>NULL)

Temp3=↑Temp3.sig;

Ttemp3.sig=Temp2;

}

Si no

{

↑Inicio=Temp2;

}

}

Si no

{

Imprimir("El símbolo a insertar ya existe en la lista, intente ingresar un símbolo diferente");

}

}

Listar\_Simbolos(↑Inicio | )

{

Temp=Inicio;

mientras(Temp<>NULL)

{

Imprimir("Simbolo:",↑Temp.simb);

Imprimir("Probabilidad:",↑Temp.prob);

Temp=↑Temp.sig;

}

}

Borrar\_Simbolo(Borrador | ↑Inicio)

{

temp= ↑Inicio;

Ant= NULL;

Mientras((temp<>NULL) AND (↑temp.simb<>Borrador))

{

ant=temp;

temp=↑temp.sig;

}

Si(temp <> NULL)

{

Si (Ant <> NULL)

{

↑Ant.sig = ↑Temp.sig;

}

Si no

{

↑Inicio. ↑Temp.sig;

}

free (Temp);

Imprimir("El símbolo se eliminó con éxito");

}

Si no

{

Imprimir("El símbolo no existe");

}

}

Modificar\_Simbolo(Simbolo, Nuevo | ↑Inicio)

{

Verificar=0

temp=↑Inicio;

mientras(temp<>NULL && Verificar=0)

{

Si(↑temp.simb = Simbolo)

{

↑temp.prob=Nuevo;

Verificar=1

}

temp=↑temp.sig;

}

Si (Verificar=1)

{

Imprimir("La probabilidad se modificó al símbolo con éxito");

}

Si no

{

Imprimir("El símbolo no se encontró en la lista, ingrese otro símbolo a modificar.");

}

}

Guardar\_Simbolos\_Arch(↑Inicio | )

{

temp=Inicio;

Imprimir("Ingresa el nombre del archivo a guardar");

Leer(NombreArchivo);

Abrir(Archivo);

mientras(temp<>NULL)

{

ImprimirArchivo(Archivo, ↑temp.simb, ↑temp.prob, ↑temp.codigo);

temp=↑temp.sig;

}

Cerrar(Archivo);

}

Leer\_Simbolos\_Arch ( | ↑Inicio)

{

Imprimir("Ingresa el nombre del archivo a leer");

Leer(NombreArchivo);

Abrir(Archivo);

Si(Archivo=NULL)

{

Imprimir("No existe el archivo,intentelo de nuevo");

}

Si no

{

Mientras ((LeerDesdeArchivo (Archivo, Simbolo | )) = 1)

{

new (Nuevo);

↑nuevo.simb = Simbolo;

LeerDesdeArchivo (Archivo, ↑nuevo.codigo);

LeerDesdeArchivo (Archivo, ↑nuevo.prob);

LeerDesdeArchivo (Archivo, ↑nuevo.status);

LeerDesdeArchivo (Archivo, ↑nuevo.tipo);

↑nuevo.sig = NULL;

Si (↑Inicio != NULL)

{

temp = ↑Inicio;

mientras (↑temp.sig != NULL)

temp = ↑temp.sig;

↑temp.sig = nuevo;

}

Si no

{

↑Inicio = nuevo;

}

}

Cerrar(Archivo);

}

}

Codigo (↑Inicio, ↑Raiz | Codigotemp)

{

Si (↑Raiz <> NULL)

{

Si (↑Raiz.izq <> NULL)

{

Si (↑Raiz.simb <> Inicio.simb)

{

Si (↑Raiz.izq <> NULL)

{

↑Codigotemp = 0;

Codigo (Inicio, ↑Raiz.izq, ↑Codigotemp);

}

Si (↑Raiz.der <> NULL)

{

↑Codigotemp = 1;

Codigo (Inicio, ↑Raiz.der, ↑Codigotemp);

}

}

}

}

}

Valida\_Generacion(↑Inicio | validación) {

sum = 0;

validacion = 1;

temp = ↑Inicio;

mientras(temp <> NULL) {

sum = sum + ↑temp.prob;

temp = ↑temp.siguiente;

}

si(sum <> 100) {

validacion = 0;

}

}

Generar\_Arbol(↑Inicio | ↑Raiz){

sum = 0, temp = ↑Inicio;

men = ↑temp;

men2 = ↑temp;

mientras(sum <> 100){

mientras(↑temp <> NULL){

si(↑temp.prob < ↑men.prob AND men.status <> 1){

men = temp;

men.status = 1;

}

temp = temp.sig;

}

temp = ↑Inicio;

mientras(↑temp <> NULL){

Si(↑temp.prob < ↑men2.prob AND men2.status <> 1){

men2 = temp;

men2.status = 1;

}

temp = temp.sig;

}

new(Nodo);

new(Nodo2);

new(Nodo3);

↑Nodo.simb = ↑men.simb;

↑Nodo.prob = ↑men.prob;

↑Nodo2.simb = ↑men2.simb;

↑Nodo2.prob = ↑men2.prob;

Si(↑Nodo.tipo = 0 and ↑Nodo.2.tipo = 0){

↑Nodo.izq = NULL;

↑Nodo.der = NULL;

↑Nodo2.izq = NULL

↑Nodo2.der = NULL;

}si no{

↑Nodo3.izq = Nodo;

↑Nodo3.der = Nodo2;

}

↑Nodo3.prob = ↑Nodo.prob+↑Nodo2.prob;

↑Raiz = ↑Nodo3;

sum = ↑Raiz.prob;

}

}

Codificar(↑Inicio, Archivo, Archivo2 | ) {

abrir(archivo);

abrir(archivo2);

bandera = 0;

mientras(leerlLinea(linea)){

para i = 0 hasta i = longitud(linea){

temp = ↑Inicio;

bandera = 0;

mientras(temp <> NULL and bandera = 0){

si(temp.simb = linea[i]){

imprimirenArchivo(temp.codigo, Archivo2);

bandera = 1;

}

si(bandera = 0){

imprime(“No se ha encontrado el caracter”);

cerrar(Archivo2);

eliminar(Archivo2);

}

temp = temp.sig;

}

}

}

si(bandera = 1)

cerrar(Archivo2);

cerrar(Archivo);

}

Decodificar(↑Raiz, Archivo, Archivo2 | decodificado){

abrir(Archivo);

leerdeArchivo(Archivo, linea);

cerrar(Archivo);

decode(↑Raiz, linea | decodificado);

abrir(Archivo2);

imprimirenArchivo(Archivo, decodificado);

cerrar(Archivo);

}

Decode(↑Raiz, linea | decodificado){

si(linea <> '\0'){

si(linea = 0)

decode(↑Raiz.izq, linea+1, decodificado);

si(linea = 1)

decode(↑Raiz.der, linea+1, decodificado);

si(↑Raiz.tipo = 0 and tipo = 0){

decodificado = ↑Raiz.simb;

decodificado = decodificado+1;

}

}

Reiniciar\_Lista (↑Inicio | )

{

Temp = Inicio;

Mientras (Temp <> NULL)

{

↑Temp.status = 0;

Temp = ↑Temp.sig;

}

}

Borrar\_Lista(↑Inicio | )

{

Temp = Inicio;

Mientras(Temp <> NULL)

{

Inicio = ↑Inicio.sig;

free(Temp);

Temp = Inicio;

}

}

Borrar\_Arbol(↑Raiz | )

{

Si(Raiz<>NULL)

{

Borrar\_Arbol(↑Raiz.izq);

Borrar\_Arbol(↑Raiz.der);

free(Raiz);

}

}

IV REFERENCIAS

[1] P. sznajdleder Algoritmos a fondo con implementaciones en C y Java, 5, Argentina. Alfaomega, 2017, 513-520

**Diagrama IPO**