Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación
CC3001- Algoritmos y Estructuras de Datos

Tarea N°5
Compresión de Huffman

Alumno: José Pacheco Aguilera
Profesor: Patricio Poblete
Auxiliares: Gabriel Flores
Sven Reissengger
Ayudantes: Gabriel Chandia
Fabian Mosso
Matias Risco
Matias Ramirez
Fecha de entrega: 15 de Agosto de 2018

1. Introducción

En el presente informe se muestra un algoritmo que permite obtener la codificación de Huffman para un archivo de texto cualquiera, es decir, dado un nombre de archivo de texto, el programa lo lee y analiza para lograr obtener otro archivo de texto mostrando una tabla con el carácter el texto original y como queda su nueva codificación. El objetivo de la codificación de Hufmman es comprimir el archivo obteniendo una nueva codificación para cada carácter.

2. Análisis del problema

El problema se trata de comprimir cualquier archivo de texto con la codificación de Huffman que consiste en leer el archivo de texto original, luego se obtiene la frecuencia de cada carácter del archivo y con esto se crea un árbol que contiene dos valores, el valor de carácter en código ASCII y la frecuencia de cada carácter, y luego para ir formando la codificación se escogen los dos arboles con menor frecuencia y luego se fusionan creando un árbol cuyo valor es la suma de la frecuencia de ambos arboles. Luego desde la raíz se va bajando hasta cada carácter,si se avanza hacia la izquierda es un 0 y si se avanza hacia la derecha es un 1, y así se logra obtener la codificación de Huffman.

3. Solución del problema

Para la solución del problema se ejecuto el proceso descrito anteriormente, para lo cual se crearon dos funciones auxiliares,la primera es *generaarbol()* que recibe una lista de arboles cuyos valores son un carácter y su frecuencia,lo primero que hace esta función es tomar el primer y segundo árbol de la lista y ver cual tiene mayor frecuencia, luego se guarda en una variable k la posición del árbol que tiene menor frecuencia y en una variable j la posición del árbol con segunda menor frecuencia, luego se comienza a comparar la frecuencia desde el tercer árbol hasta el ultimo, si la frecuencia del árbol es menor que la frecuencia del árbol en k y del árbol en j, j toma el valor de k y k toma el árbol de la posición donde se encuentra el actual árbol. si la frecuencia del actual árbol es solo menor a la frecuencia del árbol ubicado en j, el valor de j pasar a ser la posición donde se encuentra el árbol actual, si la frecuencia del árbol actual es mayor a las frecuencias de los dos arboles guardados se continua con el siguiente. Una vez que se termina de recorrer la lista, se toman los arboles guardados en k y j y se crea un nuevo árbol con su hijo izquierdo el árbol de menor frecuencia de los dos y su hijo derecho el con mayor frecuencia, y los valores de este nuevo árbol son 0 (ya que el primer valor no sirve de nada) y la suma de las frecuencias de sus hijos, luego se eliminan los arboles de la lista y se inserta el nuevo árbol creado, este proceso se realiza hasta que solo quede una elemento en la lista que va a ser el árbol con los caracteres ordenados según su frecuencia y lo retorna.

La segunda función auxiliar creada es *codigoarbol()* que recibe 4 parámetros, un árbol, un string (*código*), otro string (x) y un HashMap, el objetivo de esta función es escribir la

codificación de cada carácter, lo primero que hace es sumar los strings *código* y *x*, donde *código* es la codificación que lleva hasta el momento y *x* en el valor que se tiene que agregar dependiendo si se avanza hacia el hijo izquierdo o el hijo derecho del árbol.El proceso para obtener el código de cada carácter se realiza con una recursión, en la cual, si el hijo izquierdo y el hijo derecho del árbol que se esta analizando son *null*, se inserta en el HashMap, donde la clave es el carácter en código ASCII y su valor es *código* que representa su codificación. Si no se cumple la condición se vuelve a llamar a la función *codigoarbol* con el hijo izquierdo y el hijo derecho del árbol, el código que se lleva hasta ahora y el valor de *x* dependiendo si es el hijo izquierdo o derecho, si es el derecho 1 y si es el izquierdo 0.

Luego se procedió con el programa, para lo cual se pregunta al usuario el nombre del archivo de texto que se quiere codificar y el nombre del archivo donde se quiere mostrar la codificación, lo siguiente es leer el archivo carácter a carácter guardando su frecuencia en un HashMap, si el carácter no esta en el HashMap se guarda como una llave con clave 1, si el carácter esta en el HashMap la nueva clave es la clave anterior más 1. Una vez creado el HashMap, cada para llave-clave se coloca en el nodo de una árbol y se crea un lista que contiene todos los arboles creados. Esa lista de arboles se entrega a la función *generaarbol()* para que cree el árbol de frecuencias y ese árbol creado se le entrega a la función *codigoarbol* con dos string vacíos y un HashMap igual vacio. Y por ultimo se recorre el primer HasMap creado y se va escribiendo en el archivo de texto el carácter, el carácter en código ASCII, la frecuencia de aparición, el porcentaje de frecuencia de aparición y por ultimo la codificación de Huffman. Además el programa imprime en la pantalla el largo del archivo original y el largo del archivo comprimido para poder compararlos.

4. Modo de uso

Para lograr emplear el programa, lo primero que se debe hacer es compilar la clase *Comprimir*, luego se escribe el nombre del archivo de texto que se quiere comprimir con el .txt y por ultimo se escribe el nombre del archivo de texto donde se quiere escribir la tabla

5. Resultados

A continuación se muestran partes de los resultados para los tres textos de pruebas.

Caracter	Codigo ASCII	Frecuencia de Aparicion	Porcentaje de Frecuencia	Codificacion
\n	10	1483	2.2131355	111110
\r	13	1483	2.2131355	111101
espacio	32	10612	15.8366785	110
¡	161	322	0.48053247	10110100
!	33	34	0.050739456	10101010010
"	34	188	0.28055933	00011010
'	39	2	0.0029846737	101010100111001
(	40	4	0.0059693474	10001100001001
)	41	4	0.0059693474	10001100001000
©	169	188	0.28055933	00011001
,	44	840	1.2535629	000111
-	173	487	0.7267681	1000101
-	45	5	0.0074616843	10101010011111
.	46	559	0.8342163	1010100
‰	8240	6	0.008954021	0001101101100
0	48	6	0.008954021	10110101011111
±	177	143	0.21340416	101010101
1	49	18	0.026862064	101010110001
2	50	17	0.025369728	101010100110
³	179	214	0.31936008	01101000
3	51	17	0.025369728	100011001101
4	52	16	0.02387739	100011001100
5	53	16	0.02387739	100011000011
6	54	10	0.014923369	1011010101110
7	55	6	0.008954021	10110101011110
8	56	6	0.008954021	10110101110101
9	57	6	0.008954021	10110101110100
:	58	87	0.12983331	1011010110

Figura 1: El Cid

Caracter	Codigo ASCII	Frecuencia de Aparicion	Porcentaje de Frecuencia	Codificacion
\n	10	7043	3.7189581	11001
\r	13	7043	3.7189581	11000
espacio	32	28821	15.218528	101
!	33	340	0.17953226	011011011
"	34	32	0.016897155	1101110101011
#	35	1	5.280361E-4	01010100110010110
$	36	1	5.280361E-4	010101001100101111
%	37	3	0.0015841082	1101110101010000
&	38	6	0.0031682164	011011010011101
'	39	1078	0.56922287	0011010
(	40	17	0.008976613	0101010011000
)	41	17	0.008976613	0011001010101
*	42	64	0.03379431	00110010100
+	43	1	5.280361E-4	010101001100101110
,	44	3364	1.7763134	100111
-	45	1048	0.5533818	0011000
.	46	2045	1.0798337	1101111
/	47	12	0.006336433	01101101001111
0	48	34	0.017953226	001100101011
1	49	58	0.030626092	110111010100
2	50	37	0.019537335	010101001101
3	51	10	0.0052803606	01101101001100
4	52	3	0.0015841082	0110110100110111
5	53	3	0.0015841082	0110110100110110
6	54	7	0.0036962526	110111010101001
7	55	3	0.0015841082	0110110100111001
8	56	3	0.0015841082	0110110100111000
9	57	15	0.007920541	11011101010101

Figura 2: Hamlet

Caracter	Codigo ASCII	Frecuencia de Aparicion	Porcentaje de Frecuencia	Codificacion
\n	10	9991	3.0746553	10010
\r	13	9991	3.0746553	10001
„	8222	20	0.0061548497	10011010110010
espacio	32	54696	16.832283	111
!	33	1039	0.31974447	01010001
"	34	20	0.0061548497	10011010110001
¤	164	1318	0.40560463	10110101
'	39	1100	0.33851674	01010111
(	40	6	0.0018464549	100110101000100
)	41	6	0.0018464549	1001101011001111
*	42	1	3.077425E-4	10011010100010110
+	43	88	0.027081339	100110101110
,	44	5240	1.6125706	101100
-	45	637	0.19603197	100110100
.	46	2220	0.6831883	0111000
/	47	10	0.0030774248	100110101000111
1	49	5	0.0015387124	1001101011001100
2	50	6	0.0018464549	1001101011001110
3	51	3	9.2322746E-4	1001101010001010
4	52	5	0.0015387124	1001101010001101
5	53	3	9.2322746E-4	10011010110011011
¶	182	841	0.25881144	110011110
6	54	2	6.15485E-4	10011010100011001
:	58	1289	0.3966801	10110100
»	187	1	3.077425E-4	100110101000101111
;	59	760	0.23388429	101111011
¼	188	1684	0.51823837	11001101
=	61	1	3.077425E-4	100110101000101110

Figura 3: Urfaust

Tabla 1

Nombre del archivo	Largo del archivo original	Largo del archivo comprimido
Hamlet.txt	1515048	900389
ElCid.txt	536072	309052
Urfaust.txt	2599576	1573158

La tabla anterior muestra una comparación entre el largo del archivo de texto original y el largo del archivo comprimido.