PRÁCTICA 1
ALGORITMIA BÁSICA

MEMORIA

Curso 2018-2019

Rubén Rodríguez Esteban 737215
José María Vallejo Puyal 720044

1. Introducción

El objetivo de esta práctica es diseñar un algoritmo voraz de compresión de ficheros huffman. Dicho algoritmo consiste en generar una serie de códigos binarios para cada uno de los caracteres que contiene el fichero original y sustituirlos por sus códigos binarios correspondientes de compresión. La asignación de códigos se efectúa de modo que los caracteres más repetitivos tengan un código de longitud menor que los que aparecen menos. Estos códigos se generan a partir de un árbol binario en cuyas hojas se encuentran los caracteres con sus frecuencias. De este modo, a partir del código binario se obtiene su carácter correspondiente recorriendo el árbol desde la raíz hasta el nodo hoja donde dicho carácter se encuentra, visitando el hijo izquierdo o derecho del árbol dependiendo de si el último bit procesado del código binario es un 0 o un 1 respectivamente.

2. Diseño

En este apartado se va a explicar cuál ha sido el planteamiento de diseño empleado para abordar la práctica.

2.1 Tipos abstractos de datos empleados

- CarFrec: representa las tuplas de parejas <*caracter, frecuencia*> donde *caracter* hace referencia a los diferentes caracteres que aparecen en el fichero y *frecuencia* al número de veces que aparece el carácter en el fichero.

- Heap: representa un montículo de prioridad de mínimos para ordenar las tuplas <*caracter, frecuencia*> por orden de frecuencia creciente.

- ArbolTrie: representa los nodos que conforman el árbol binario de los códigos Huffman. Para cada nodo se almacena la tupla <caracter, frecuencia>, cuyo valor es nulo si es un nodo no hoja, la suma de las frecuencias de los nodos hijos, y los punteros al hijo izquierdo y al hijo derecho.

- Huffman: representa una colección de métodos que permiten llevar a cabo la creación del árbol de códigos Huffman que permite la compresión y descompresión de ficheros.

- Preliminar: contiene una colección de métodos que permiten la lectura inicial del fichero y contar el número de apariciones de cada carácter, creando así, las diferentes tuplas <caracter, frecuencia>.

2.2 Formato del fichero comprimido

El fichero comprimido almacena, en primera instancia, la codificación del árbol huffman que se ha obtenido a partir del fichero original sin comprimir, con el objetivo de poder reconstruirlo y llevar a cabo la descompresión de forma adecuada.

Para codificar el árbol, ya que todos los caracteres podían formar parte del fichero original, y por lo tanto, no se podía leer hasta un carácter delimitador, se ha optado por codificar el árbol en preorden de tal manera que las 'N's' representan nodos que no son hojas y las 'H's' nodos hojas. Cuando se lee una 'H' (nodo hoja), se lee el siguiente carácter que representa al carácter que contiene ese nodo hoja. Como las frecuencias en este caso no tenían ninguna funcionalidad se ha decidido no incluirlas en la codificación del árbol.

Lo siguiente que se puede encontrar en el fichero comprimido es el número de caracteres que tiene el texto original y por tanto el texto tras ser descomprimido. Se ha decidido guardar este valor ya que en algunos casos se ha de añadir una serie de 0's al final del fichero para completar los bits restantes del último carácter que se va a escribir, conociendo el número total de caracteres que hay que decodificar se pueden ignorar estos 0`s de relleno. Por último, y no menos importante, se encuentra el texto del fichero original codificado según la codificación huffman.

3. Problemas encontrados

Durante la realización de la práctica han surgido problemas de distinta índole que han tenido que ser discutidos y analizados adecuadamente para poder abordarlos de la mejor forma posible. Uno de los problemas más importantes ha sido la forma de codificar el árbol ya que no se podía leer en el fichero comprimido hasta un caracter especifico, ya que este tambien se podia encontrar en el texto original, y por lo tanto, no se generaría bien el árbol y al empezar a decodificar caracteres se decodificarían algunos que forman parte de la codificación del árbol y no del texto original.

Durante las pruebas también han surgido problemas con el formato de codificación de los ficheros ya que algunos de los ficheros tenían el final de fichero de windows y otros el de linux. Al ser una secuencia de caracteres distinta no se decodifican bien algunos textos añadiendo un carácter de más. Por lo que finalmente se ha optado solo por decodificar textos con final de fichero de linux.

4. Compilación y ejecución

Para compilar el material entregado se proporciona un archivo makefile que reúne los ficheros auxiliares necesarios para poder generar el archivo principal. Para la compilación se ha empleado el estándar de c++11 junto con el flag de compilación -03 con el fin de optimizar el rendimiento. Los comandos para poder llevar a cabo la compilación figuran en la cabecera del fichero makefile.

5. Pruebas de corrección

Para comprobar el correcto funcionamiento de la compresión y de la descompresión se ha seguido una metodología ascendente en cuanto a nivel de dificultad, es decir, se ha empezado poniendo a prueba el programa con los ficheros más fáciles y posteriormente con los ficheros más difíciles con el fin de facilitar la corrección y depuración del código.

Inicialmente se ha creado un fichero con unos caracteres y unas frecuencias idénticas al problema de codificación proporcionado en el material didáctico de la asignatura (véase transparencia 63). De esta forma se ha podido comprobar que los procedimientos encargados de la generación del árbol y de obtener los códigos de cada carácter presente en el texto original eran correctos. Una vez comprobado el correcto funcionamiento de los procesos anteriores, se ha procedido con la compresión y la descompresión del fichero en cuestión. Posteriormente se ha verificado que ambos ficheros son idénticos. Para ello, basta con ejecutar en el intérprete de comandos la orden: *diff ficheroAntesDeComprimir ficheroDescomprimido.* El proceso anterior se ha repetido después con los demás ficheros.

6. Comparativa de resultados

A continuación se proporciona una tabla comparativa donde se reflejan los ficheros utilizados para poner a prueba el programa, los tamaños de los ficheros antes y después de efectuar la compresión (en bytes) y el tiempo invertido para llevar a cabo ambos procesos (en milisegundos), y la relación entre el tamaño del fichero original y el comprimido (ratio de compresión).

Nombre del fichero	Tamaño original	Tamaño comprimido	Tiempo de compresión	Tiempo descompresión	Ratio de compresión
frecuencias.txt	102 Bytes	59 Bytes	10 ms	10ms	57,84%
profitChecker.sh	1919 Bytes	1352 Bytes	30 ms	40 ms	70,45%
timeChecker.py	2107 Bytes	1476 Bytes	50 ms	10 ms	70,05%
estaciones.txt	60471 Bytes	40463 Bytes	120 ms	60ms	66,91%
usuarios-16.txt	283906 Bytes	137158 Bytes	420 ms	220 ms	48,31%
usuarios-17.txt	284564 Bytes	137689 Bytes	460 ms	220 ms	48,38%
quijote.txt	2198927 Bytes	1171333 Bytes	3290 ms	1790 ms	53,26%
binario.bin	6445508 Bytes	4912850 Bytes	12750 ms	6460 ms	76,22%
usos-16.csv	61610156 Bytes	29348258 Bytes	87940 ms	47040 ms	47,63%