PRÁCTICA 1 ALGORITMOS VORACES

AUTORES: Manel Jordá Puig NIP: 839304
José Miguel Angós NIP: 852646

En esta práctica reflejaremos los conocimientos teóricos aprendidos sobre los algoritmos voraces. En concreto, nos centraremos en los métodos de compresión que hacen uso de dichos algoritmos, como el de Huffman, que es el que implementaremos en esta tarea.

Se dispone de 3 tareas a realizar:

En la tarea 1, el objetivo es diseñar un algoritmo voraz que pueda tomar un texto como entrada y construir un árbol para luego compactar el archivo. Esto supone tener que desarrollar un método que pueda representar eficientemente el texto de entrada en forma de un árbol para luego realizar su compresión.

En la tarea 2, se debe implementar un programa compactador/descompactador llamado "huf" que utilice el método de compresión de archivos basado en el código de Huffman.

En la tarea 3, se tendrá que realizar distintas pruebas para medir la eficiencia y el coste de los algoritmos.

Este programa se ejecuta de la siguiente forma:

Para compactar un archivo:

> **huf -c <nombre de fichero>**
>
> <nombre de fichero> es el nombre del archivo, ya sea de texto o binario. El programa generará un archivo compactado con extensión .huf.

Para descompactar un archivo:

> **huf -d <nombre de fichero>**
>
> <nombre de fichero> es el nombre de un archivo compactado. El programa generará el archivo original a partir del archivo compactado.

El algoritmo de Huffman es un método de compresión que asigna códigos más cortos a los caracteres más comunes y códigos más largos a los menos comunes, basándose en su frecuencia de aparición.

En cuanto a la codificación, se crea un árbol binario donde los caracteres del archivo se representan como hojas. La frecuencia de cada carácter determina su posición en el árbol.

Los caracteres más frecuentes están más cerca de la raíz, mientras que los menos frecuentes están más alejados. Los códigos se generan siguiendo el camino desde la raíz hasta cada hoja y se asigna '0' al moverse hacia la izquierda y '1' al moverse hacia la derecha.

IMPLEMENTACIÓN DE LA COMPRESIÓN

En cuanto a la parte de compresión, primero utilizamos la función **contar_caracteres**, que abre el archivo de entrada, cuenta la frecuencia de cada caracter y devuelve un unordered_map que mapea cada caracter con su frecuencia de aparición.

Luego, creamos una cola de prioridad de pares (caracter, frecuencia) con la función **crear_cola_prio**. Esta cola se utiliza para construir el árbol de Huffman.

Utilizamos la función **construir_arbol** para construir el árbol de Huffman a partir de la cola de prioridad. Creamos nodos para cada caracter y los insertamos en un arreglo de nodos. Luego, combinamos repetidamente los dos nodos con menor frecuencia para construir el árbol. Una vez construido, asignamos códigos binarios a cada caracter utilizando la función **asignar_codigos**. Recorremos el árbol de Huffman y asignamos códigos binarios a cada caracter en función de su posición en el árbol.

Finalmente, utilizamos la función **escribir_fichero_huffman** para escribir el archivo comprimido. Primero, escribimos el nombre del archivo original y el diccionario de codificación en el archivo comprimido. Luego, recorremos el archivo original y reemplazamos cada caracter por su código binario correspondiente.

IMPLEMENTACIÓN DE LA DECODIFICACIÓN

En la función **descomprimir**, declaramos un unordered_map llamado diccionario_decod, el cual será utilizado para mapear los códigos binarios a sus respectivos caracteres durante la decodificación.

Después, abrimos el archivo comprimido fichero_huf para su lectura. Si el archivo no se puede abrir, se muestra un mensaje de error y se termina el programa.

Si el archivo se puede abrir, obtenemos el nombre del archivo original mediante la función **obtener_nombre_fich_decod**. Luego, llamamos a la función **obtener_diccionario** para extraer el diccionario de codificación del archivo comprimido y almacenarlo en diccionario_decod. Este diccionario será utilizado durante el proceso de decodificación para mapear los códigos binarios en los caracteres originales.

Finalmente, llamamos a la función **escribir_fichero_original** para escribir el archivo descomprimido utilizando el diccionario de decodificación. Esta función lee el archivo comprimido, recupera los códigos binarios y los convierte de vuelta a caracteres utilizando el diccionario.

EXPERIMENTACIÓN

Para probar el correcto funcionamiento se han creado varios ficheros de prueba tanto en formato .txt como en binario.

Para la realización de las pruebas, basta con añadir nuevos ficheros a la carpeta "**archivos_de_prueba**" y volver a lanzar el script "**ejecutar.sh**". Se generarán los .huf y se realizarán las comprobaciones correspondientes con los nuevos ficheros.

Para la visualización de los tiempos de ejecución, basta con ejecutar el programa o script directamente, y se mostrará por la salida estándar el tiempo en milisegundos que ha tardado en realizar la compresión (construir el árbol y realizar el diccionario).