Práctica 1: Algoritmos Voraces

Implementación del algoritmo de Huffman

Myrtille Knockaert, Maryne Comblat - ERASMUS

Febrero 2025

Índice

1. Introduction

Este proyecto tiene como objetivo implementar un programa de compresión y descompresión de archivos utilizando el algoritmo Huffman. Este algoritmo se basa en un enfoque voraz, lo que significa que en cada paso selecciona los dos elementos más ligeros para fusionarlos progresivamente, hasta formar un árbol binario optimizado. Este proceso garantiza una codificación de prefijo mínima, lo cual es esencial para una compresión de archivos eficiente. El objetivo es reducir el tamaño de los archivos garantizando al mismo tiempo su reproducción fiel tras la descompresión.

La implementación se basa en un conjunto de scripts, cada uno con un rol bien definido. El archivo huf.py es el punto de entrada del programa y permite realizar la compresión y descompresión a través de una interfaz de línea de comandos. Se basa en el archivo functions.py, que contiene todas las funciones necesarias para procesar los datos, incluida la construcción del árbol Huffman, la generación de códigos binarios y la codificación de los archivos. Un script Bash, ejecutar.sh, automatiza la ejecución del programa garantizando que Python esté instalado, iniciando la compresión o descompresión y manejando los parámetros de entrada, incluida la capacidad de limitar la longitud de los códigos generados.

2. Tarea 1. Diseño

El algoritmo de Huffman se basa en la construcción de un árbol binario en el que cada carácter del texto original recibe un código único basado en su frecuencia de aparición.

Para construir el árbol:

Se calculan las frecuencias de cada carácter en el texto. Se crean nodos individuales para cada carácter, organizados en una cola de prioridad. Se fusionan los dos nodos con menor frecuencia en un nodo padre, repitiendo el proceso hasta obtener un solo nodo raíz. Se asigna un 0 al camino izquierdo y un 1 al derecho para generar los códigos binarios. Este proceso da como resultado una codificación óptima que minimiza la longitud total del mensaje codificado. En la figura siguiente, se muestra un ejemplo de un árbol de Huffman generado para las lettras A,B,C,D:

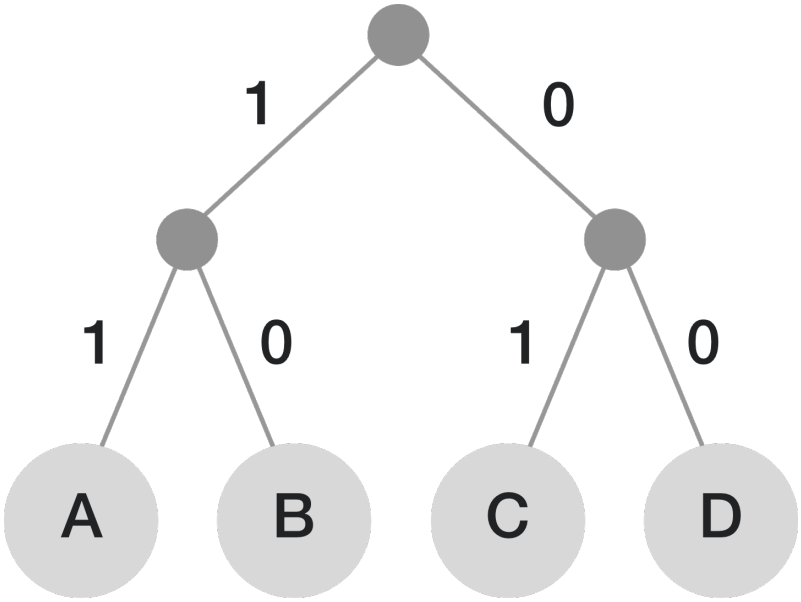

Figura 1: Ejemplo de árbol de Huffman

3. Tarea 2. Implementación

Lo más difícil es encontrar una estructura de árbol y pensar en la implementación del recorrido para la función alphabet.

4. Tarea 3. Mejorando algunos casos

Para mejorar situaciones particulares y perfeccionar la compresión de datos aún más, se ha incorporado la opción de restringir la extensión de los códigos Huffman. Este avance busca asegurar que los códigos generados cumplan ciertas limitaciones de longitud, lo cual resultará beneficioso para aplicaciones específicas que demandan una codificación más uniformemente aplicada.

Para ello, se ha integrado una nueva opción -l max_length -c file en ejecutar.sh. Esta funcionalidad se basa en un archivo Python adicional, mejoras.py, que modifica la generación del árbol Huffman para respetar esta restricción de longitud máxima. Si no se cumplen las condiciones no se realiza la compresión.

5. Tarea 4. Experimentación

Para verificar la precisión del programa y analizar su eficacia, se realizaron varias pruebas en archivos de distintos tamaños y contenidos. Utilizamos seis archivos de prueba que cubrían diferentes escenarios: un texto con frecuencias de caracteres similares, un texto con algunos caracteres muy frecuentes y algunos raros, un texto donde algunos caracteres son muy raros, un texto corto en español que contiene acentos y tildes, y un texto largo.

El experimento consistió en ejecutar compresión y descompresión en estos archivos y luego verificar si el archivo descomprimido era idéntico al original utilizando el comando diff. Los resultados mostraron que la compresión y la descompresión funcionan bien para archivos de texto, con una restitución perfecta del texto original.

6. Conclusion

Este proyecto implementó un programa para comprimir y descomprimir archivos de texto utilizando el algoritmo Huffman, basado en un enfoque codicioso. La experimentación confirmó que el programa funciona bien, reproduciendo fielmente los archivos después de la descompresión, con una compresión más eficiente cuando la distribución de frecuencia es desigual.

En términos de mejoras, se podría optimizar la construcción del árbol de Huffman utilizando una cola de prioridad en lugar de una lista ordenada en cada iteración, lo que reduciría la complejidad de la clasificación. Además, la gestión de caracteres especiales y la integración de un análisis de la tasa de compresión permitirían perfeccionar la eficiencia del programa.

Finalmente, agregar una interfaz simplificada o la integración con otras herramientas podría hacer que este algoritmo sea más accesible y explotable en diversos contextos que requieran la optimización del almacenamiento de datos.