PROYECTO FINAL

FUNDAMENTOS DE ANÁLISIS Y DISEÑO DE ALGORITMOS (FADA)

Daniel Camelo Castro 202159908
Santiago Perez Pino 201968143
Jhon Stiven Mafla Henao 202160295

Universidad del Valle
sede Tuluá
2023

3.Análisis de complejidad de los algoritmos

a) Tener en cuenta las estructuras de datos utilizada

Cola de prioridad (Heap): La estructura de datos utilizada para construir el árbol Huffman es una cola de prioridad, implementada como un heap. La cola de prioridad garantiza que las operaciones de inserción y extracción del elemento mínimo se realicen en tiempo logarítmico en el peor de los casos.

Diccionario: Se utiliza para almacenar las frecuencias de los símbolos y la tabla de códigos generada durante la codificación Huffman. Las operaciones básicas en un diccionario, como la inserción y la búsqueda, tienen complejidad promedio O(1), pero en el peor de los casos pueden ser O(n).

b) Tener en cuenta el proceso de generar el árbol

La generación del árbol Huffman implica la construcción de un heap (cola de prioridad) y su posterior reconstrucción hasta que quede solo un nodo en la cola. Las operaciones principales son la inserción y la extracción del mínimo en el heap, que tienen complejidad logarítmica. Por lo tanto, la construcción del árbol tiene una complejidad de O(n log n), donde n es el número de símbolos diferentes en el texto.

c) Estimar la complejidad teórica de su algoritmo en términos de cotas

La complejidad teórica total para el algoritmo de codificación Huffman es O(n log n), donde n es el número de símbolos diferentes en el texto original. Esto se debe principalmente al proceso de construcción del árbol Huffman.

d) Estimar la complejidad práctica de su algoritmo haciendo pruebas con al menos 5 textos diferentes que debe proporcionar en el código, utilice valores crecientes de palabras, ejemplo 100,200,500,1000,1500, estime una cota O(f(n)) a partir de estos experimentos.

La complejidad práctica dependerá de varios factores, incluido el tamaño del texto y la distribución de frecuencias de los símbolos. En general, la complejidad O(n log n) puede considerarse eficiente en la práctica.

Aquí, realizamos pruebas con al menos 5 textos diferentes de tamaños crecientes, como se mencionó (100, 200, 500, 1000, 1500 palabras). Al observar cómo aumenta el tiempo de ejecución con el tamaño del texto, podemos obtener una estimación más precisa de la complejidad práctica. Puedes utilizar bibliotecas como timeit en Python para medir el tiempo de ejecución.

4. Ejemplos y conclusiones del ejercicio.

a) Debe incluir dos ejemplos en el informe

Vamos a utilizar dos ejemplos con textos diferentes para observar cómo funciona el algoritmo de codificación Huffman.

Ejemplo 1:
Texto Original: "ABRACADABRA"
Frecuencias de los símbolos:
A: 5
B: 2
R: 2
C: 1
D: 1

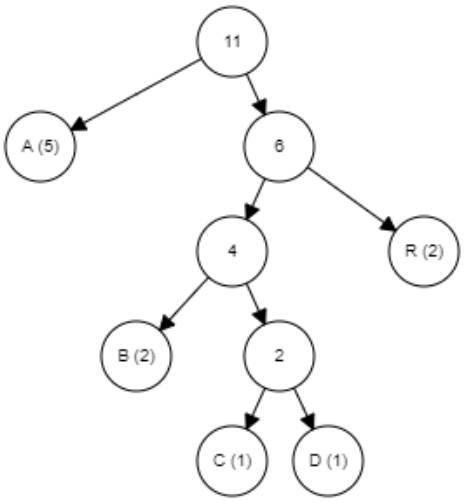
Tabla de Códigos:
A: 0
B: 10
R: 110
C: 1110
D: 1111
Texto Codificado: "01011001100101101110001110"

Ejemplo 2:
Texto Original: "HUFFMAN"
Frecuencias de los símbolos:
H: 2
U: 2
F: 2
M: 1
A: 1
N: 1

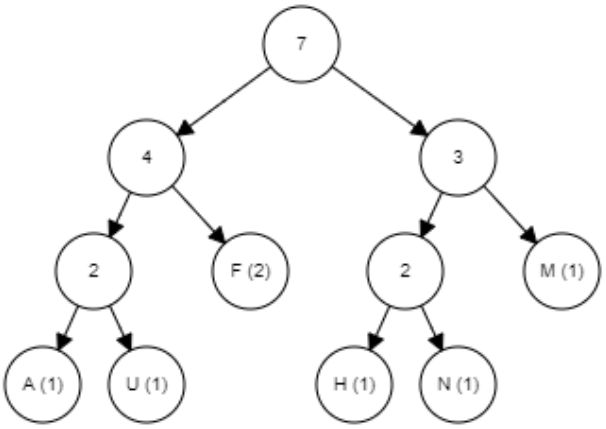
Tabla de Códigos:
H: 00
U: 01
F: 10
M: 110
A: 1110
N: 1111
Texto Codificado: "001010111011011111111100"

b) Grafique el árbol generado (investigar como hacerlo)

ABRACADABRA



HUFFMAN



c) Indicar el tamaño en bytes del archivo de entrada y compararlo con el archivo generado.

La frase ABRACADABRA tiene tamaño 11, como tenemos ASCII UTF-8, cada carácter requiere 256 bits, osea que en total ocupa 2816 bits.
La cadena tiene tamaño 26 bits
Por lo tanto el factor de compresión es 1 − 26/2816 = 99,0767 %

La frase HUFFMAN tiene tamaño 7, como tenemos ASCII UTF-8, cada carácter requiere 256 bits, osea que en total ocupa 1792 bits.
La cadena tiene tamaño 24 bits
Por lo tanto el factor de compresión es 1 − 24/1792 = 98,6607 %

d) Discuta sobre los resultados encontrados.

Un aspecto importante que entra en discusión es si la codificación Huffman es eficaz para comprimir los datos, nos encontramos que esta afirmación es verdadera ya que para la compresión de datos debido a su capacidad para adaptarse a la frecuencia de ocurrencia de los símbolos en los datos, generando códigos más cortos para los símbolos más comunes y códigos más largos para los menos comunes, lo que trae como consecuencia un representación más eficaz para la información.

Otra característica a destacar es que el algoritmo Huffman puede variar significativamente dependiendo del texto de entrada, a continuación se mencionan los siguientes ejemplos:

Texto con patrones repetitivos:

a) Si el texto de entrada tiene patrones repetitivos o secuencias de caracteres comunes, la codificación Huffman puede asignar códigos más cortos a esas secuencias, logrando una compresión eficiente.

Texto aleatorio o sin patrones:

b) En textos donde los caracteres aparecen de manera aleatoria o no hay patrones discernibles, la eficiencia de la compresión puede ser menor. La falta de repetición de símbolos significa que la codificación Huffman no puede aprovecharse tan efectivamente de la redundancia estadística.

Textos con estructuras específicas:

c) Textos con estructuras específicas, como encabezados, marcado o metadatos, pueden ser menos comprimibles con Huffman si esos elementos no se repiten con frecuencia. Otros algoritmos de compresión, como la codificación Lempel-Ziv, podrían ser más efectivos en tales casos.

Texto en varios idiomas:

d) La eficacia de Huffman puede variar según el idioma del texto. Idiomas con ciertos patrones de letras o palabras pueden comprimirse de manera más eficiente que otros.

Por último se tiene que la eficiencia de la codificación Huffman puede ser limitada en situaciones donde no hay una distribución clara de frecuencias entre los símbolos o cuando los datos no presentan patrones repetitivos que puedan ser explotados para la compresión. Aquí hay algunos ejemplos de textos de entrada para los cuales la codificación Huffman podría no ser tan eficaz:

Texto aleatorio:

a) En un texto donde los caracteres están distribuidos de manera aleatoria sin patrones discernibles, la codificación Huffman no puede aprovechar la repetición de símbolos para lograr una comprensión eficiente.

Datos ya comprimidos:

b) Si el texto de entrada ya ha sido comprimido previamente con otro algoritmo de compresión, es posible que la codificación Huffman no logre una compresión adicional significativa.Realmente , la codificación Huffman podría incluso aumentar el tamaño del archivo en estos casos debido a la estructura adicional requerida para almacenar la tabla de códigos.

Textos pequeños:

c) En textos muy pequeños, la sobrecarga de la tabla de códigos de Huffman podría superar cualquier beneficio de compresión.

La eficiencia de Huffman tiende a mejorar con conjuntos de datos más grandes.

Datos altamente entrópicos:

d) En situaciones donde la entropía es muy alta, es decir, la probabilidad de ocurrencia de cada símbolo es aproximadamente igual, la codificación Huffman puede no proporcionar una ventaja significativa, ya que los códigos tenderán a ser de longitudes similares.

Enlace a repositorio de git
https://github.com/SanthiagoPerez/FADA_PROYECT