UNIVERSIDAD CENTROAMERICANA
JOSÉ SIMEÓN CAÑAS

ANÁLISIS DE ALGORITMOS:

PROYECTO FINAL DE LA MATERIA

DOCENTE DE LA MATERIA:

ENMANUEL AMAYA ARAUJO, MSC

INTEGRANTES DEL GRUPO:

JOSÉ ALEXANDER ROBLES ROMERO (00052121)

MARIO JOSUÉ NIETO FLORES (00087419)

FABIO ALFREDO HERNANDEZ MARAVILLA (00210121)

ALFABETO A TRABAJAR:

ARABE

ANTIGUO CUSCATLAN,

INTRODUCCIÓN

En el contexto constante de la expansión tecnológica de la informática siempre se busca mejorar los métodos de programación, ya sea en términos de efectividad, velocidad, uso de memoria o desarrollando nuevos métodos para abordar las problemáticas emergentes en la vida cotidiana. Para lograr los objetivos y encontrar soluciones se recurre a lo que se conoce como algoritmos, un conjunto de instrucciones definidas y ordenadas.

En este constante desarrollo tecnológico, donde mejorar los métodos de programación es imperativo, uno de los desafíos recurrentes es el almacenamiento eficiente de datos, ya que el almacenamiento que se posee en los dispositivos es limitado, y cada día los archivos se vuelven más densos en cuanto a uso de memoria, consumiendo de manera más rápida e ineficiente nuestra memoria .Ante esta problemática se tomó a bien el desarrollar diferentes soluciones para poder almacenar archivos de manera más efectiva y consumiendo menos recursos. Entre las soluciones destaca el algoritmo de 'Huffman', el cual fue desarrollado por David A. Huffman en 1952 cuando era estudiante en MIT (Massachusetts Institute of Technology) donde estudiaba para obtener su maestría. El algoritmo fue desarrollado como parte de una tarea, donde desarrolló la demostración de la eficacia del algoritmo, dicho algoritmo resultó ser una herramienta muy eficaz para reducir el peso de los archivos y así poder almacenarlos haciendo uso de la mínima cantidad de memoria posible.

Huffman tuvo la idea de combinar árboles binarios para poder formar cadenas de números binarios donde cada una de estas será la representación de un dato contenido en nuestro árbol. El enfoque que Huffman mantuvo fue el asignar una combinación de dígitos más corta a los datos más frecuentes en el archivos y asignar un combinación más largas a los menos frecuentes, haciendo así que los archivos utilicen menos bytes en memoria.

ALGORITMO DE HUFFMAN

El 9 de agosto de 1925, en Ohio, Estados Unidos, vio la luz David A. Huffman, quien más adelante se convertiría en una figura emblemática en el campo de la informática. En 1944, a la temprana edad de 18 años, obtuvo su licenciatura en ingeniería eléctrica de la Universidad Estatal de Ohio. Desempeñó funciones como oficial de mantenimiento de radar en un destructor de la Marina de Estados Unidos,

contribuyendo a la eliminación de minas en las aguas de Japón y China tras la Segunda Guerra Mundial. Más tarde, en 1949, completó su maestría en la Universidad Estatal de Ohio, y en 1953, obtuvo su doctorado en ingeniería eléctrica en el Instituto Tecnológico de Massachusetts (MIT). (David Huffman, n.d.)

Mientras desempeñaba sus estudios en el año 1951, David A. Huffman junto a sus compañeros de clase de la materia "Teoría de la Información" en el MIT se encontraron con la decisión de elegir entre la realización de un proyecto parcial o presentar un examen final. Esta elección fue ofrecida por su profesor, quien era en aquel entonces Robert M. Fano, el cual asignó un trabajo final el cual tenía como objetivo encontrar el código binario más eficiente. Huffman, al hacer pruebas por varias ocasiones al sentirse incapaz de probar que ningún código fuera el más eficiente, estuvo apunto de darse por vencido y optar por la decisión de tomar el examen y así comenzar a estudiar para dicho examen, en ese momento se le vino a la mente la idea de usar un árbol binario ordenado por frecuencia, con esta idea en mente rápidamente demostró que este método era el más eficiente. (Garsia & Wachs, n.d.)

Este algoritmo consiste en tomar un alfabeto de "n" símbolos, junto con sus frecuencias de aparición asociadas, y produce un código, para ese alfabeto y esas frecuencias. El algoritmo consiste en la creación de un árbol binario que tiene cada uno de los símbolos por hoja, y construido de tal forma que siguiéndolo desde la raíz a cada una de sus hojas se obtiene el código Huffman asociado a él. (Garsia & Wachs, n.d.)

Con el algoritmo se crean nodos, uno por cada uno de los símbolos del alfabeto y su frecuencia de aparición, Se toman los dos nodos de menor frecuencia, y se unen creando un nuevo árbol. La etiqueta de la raíz será la suma de las frecuencias asociadas a los nodos que se unen, y cada uno de estos nodos será un hijo del nuevo nodo. También se etiquetan las dos ramas del nuevo árbol: con un 0 la de la izquierda, y con un 1 la de la derecha. Se repite el paso hasta que se logre formar un árbol donde la raíz será la suma de todas las frecuencias.

El algoritmo tiene la característica de asignar códigos más cortos a símbolos más frecuentes y códigos más largos a símbolos menos frecuentes esto es clave para la eficiencia de la compresión. Al asignar códigos más cortos a los símbolos más comunes, se logra una representación más compacta de los datos, ya que se necesitan menos bits para representar esos símbolos. Por otro lado, los símbolos menos frecuentes, que ocurren con menor frecuencia, pueden tener códigos más largos sin afectar significativamente la eficiencia global del código.

Con este algoritmo Huffman superó a Fano, quien había trabajado con Claude Shannon para desarrollar un código similar. Huffman se dio cuenta que construir el árbol de abajo hacia arriba garantiza la optimización, a diferencia del enfoque de arriba hacia abajo de la codificación de Shannon-xFano (Garsia & Wachs, n.d.).

PASOS PARA EL ALGORITMO

El algoritmo de Huffman es un algoritmo para la construcción de códigos de Huffman, Este algoritmo toma un alfabeto de “n” símbolos, junto con sus frecuencias de aparición asociadas, y produce un código único llamado código de Huffman para ese alfabeto y esas frecuencias. El algoritmo consiste en la creación de un árbol binario que tiene cada uno de los símbolos por hoja, y construido de tal forma que siguiéndolo desde la raíz a cada una de sus hojas se obtiene el código Huffman asociado a él, para los códigos de los símbolos se tiene la característica que los símbolos con mayor frecuencia en el archivo reciben un código más corto y los que tienen mejor frecuencia reciben un código más corto, volviendo así un archivo más liviano. Los pasos que debemos seguir son los siguientes:

Paso 1: Recopilación de estadísticas

- El primer paso en el Algoritmo es recopilar información sobre la frecuencia con la que aparece cada símbolo en los datos sobre los cuales vamos a trabajar.

Paso 2: Creación de nodos hoja

- Cada símbolo se debe representar como un nodo el cual será una hoja dentro de nuestro árbol de Huffman. Cada nodo hoja tiene como mínimo dos propiedades: el símbolo en sí y su frecuencia de aparición.

Paso 3: Creación del árbol de Huffman

- Para crear el árbol, se deben combinar repetidamente los nodos que tienen las frecuencias más bajas para así encontrar los nodos padres. Los nodos padres tendrán una frecuencia que proviene de la suma de las frecuencias de sus hijos. Este proceso se repite hasta que todos los nodos estén combinados en un único nodo raíz.

Paso 4: Asignación de códigos binarios

- Una vez que se ha creado el árbol de Huffman, se asignan códigos binarios a cada símbolo en función de su ubicación en el árbol. Al seguir el camino desde la raíz del árbol hasta un nodo hoja, el código binario se construye tomando "0" cuando se va a la izquierda y "1" cuando se va a la derecha. Cada nodo en el camino desde la raíz hasta un nodo hoja representa un bit en el código binario para ese símbolo. Por lo tanto, al llegar a un nodo hoja,

se ha generado un código binario único para ese símbolo basado en el camino seguido en el árbol.(GeeksforGeeks, 2023)

EJEMPLO:

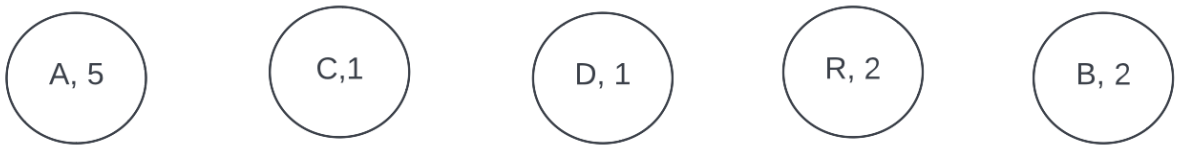
Suponiendo que se nos da una frase y se nos pide encontrar el código de huffman asociado a dicha frase, siendo la frase “ABRACADABRA”.

Para dar solución a lo solicitado vamos a seguir estos pasos:

1. Cómo primer paso vamos a calcular la frecuencia que tiene cada una de las letras en nuestra frase.

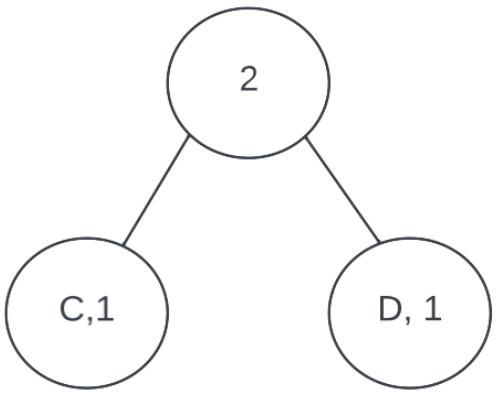
A	B	C	D	R
5	2	1	1	2

2. Ahora que conocemos la frecuencia con la que aparecen las letras vamos a crear un nodo para cada una donde también se almacenará su respectiva frecuencia.

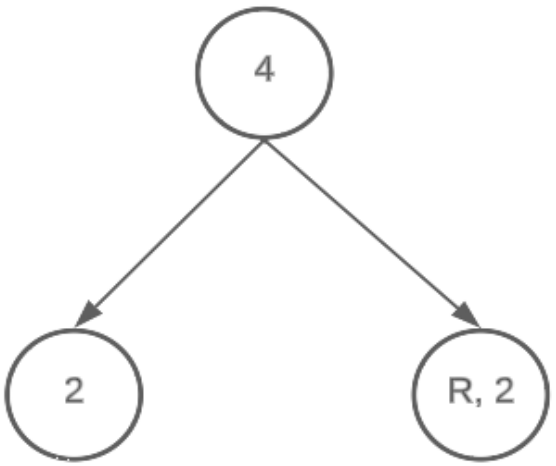


3. Teniendo nuestros nodos podemos pasar a la creación de nuestro árbol, para ello vamos a tomar los nodos con menos frecuencia, sumaremos su frecuencia y el resultado será el valor almacenado en el nodo padre, este paso se repetirá hasta formar el árbol completo.

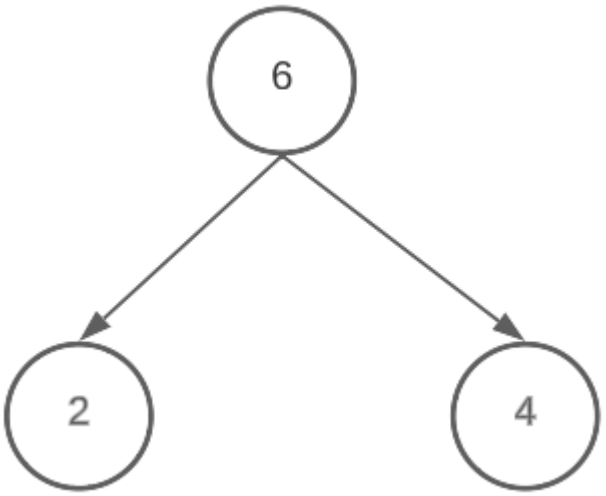
   Primera suma



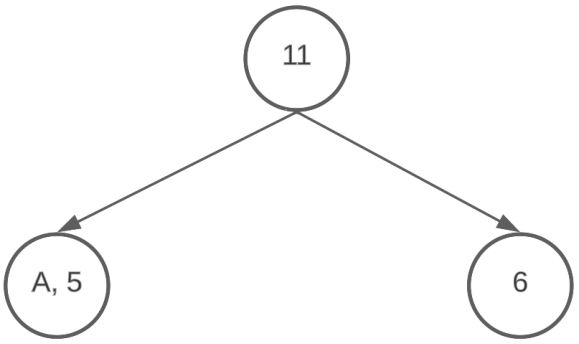
Segunda suma



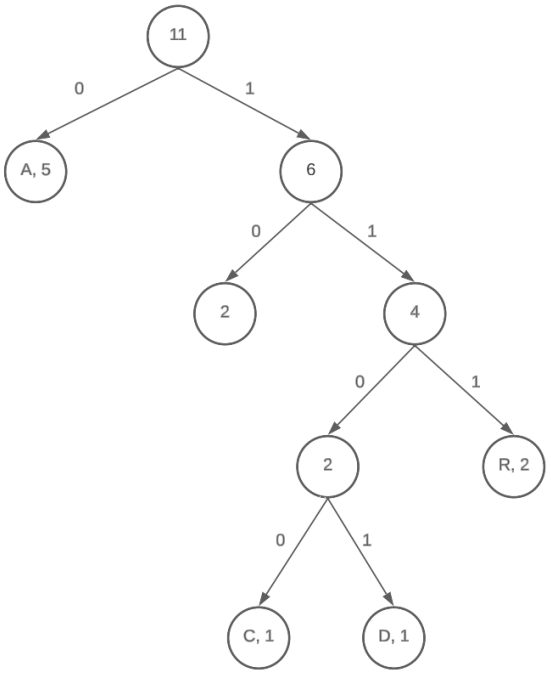
Tercera suma



Última suma



Resultado del árbol



“Otra manera de ver la creación del árbol”

Sería creando tablas de sumas, donde en el primer nivel se colocan los valores de las frecuencias y luego se comienzan a sumar las menores y así sucesivamente hasta que solamente nos quede un valor, si nos damos cuenta nuestro árbol queda construido solo que al revés es decir las hojas arriba.

5	2	1	1	2

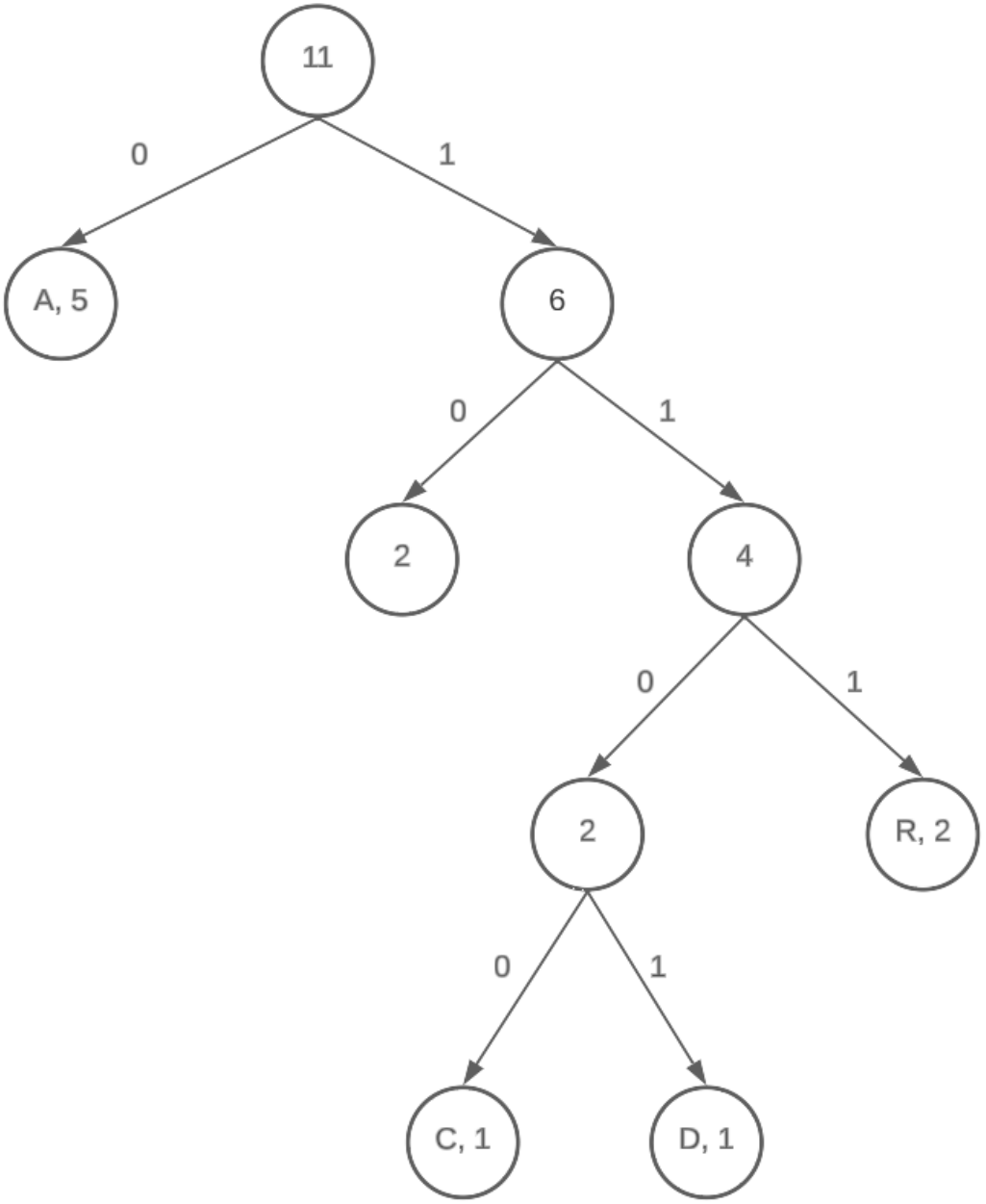
5	2	2	2

5	2	4

5	6

11

4. Una vez que hemos construido el árbol, procederemos a asignar números a los diferentes caminos para identificar cada dato. En este proceso, los caminos hacia la izquierda se etiquetaron como 0, mientras que los caminos hacia la derecha se designarán como 1.



Cuando tenemos nuestro árbol completo y los caminos enumerados comenzamos con la parte final del algoritmo que sería asignarle a cada letra de nuestra frase su respectivo código de Huffman, para ello debemos recorrer el árbol desde la raíz hasta el nodo de interés. Por cada avance que

tenemos en el árbol almacenamos el número asignado al camino y al llegar al nodo tendremos almacenado el código asignado a dicha letra.

Nuestra asignación quedaría de la siguiente manera:

A = 0

C = 1100

D = 1101

R = 10

B = 111

Donde nuestra frase sería: 0 111 10 0 1100 0 1101 0 111 10 0.

Pseudocódigo del algoritmo de Huffman

```
    HUFFMAN(n, f)
1   for i := 1 to n do
2       H[i] := (i, f(i))
3       create a leaf node labeled i (both chilren are NIL)
4   BUILDHEAP(H)
5   for i := n + 1 to 2n − 1 do
6       x := EXTRACTMIN(H); y := EXTRACTMIN(H)
7       create a node labeled i with children the nodes labeled x.label and y.label
8       INSERT(H, (i, x.freq + y.freq))
```

1. Se crea un nodo para cada carácter que se ingresan y se almacenan en una cola de prioridad (Montículo de mínimos), cada nodo que se creará contiene el carácter y su frecuencia dentro del texto.
2. Mientras que haya más de un nodo dentro de la cola de prioridad se:

    - Los dos nodos que tienen una frecuencia menor se retiran de la cola de prioridad.
    - Se crea un nodo interno donde la frecuencia del nuevo nodo será la suma de las frecuencias de sus nodos hijos, siendo estos nodos hijos los nodos que se retiraron anteriormente en la cola de prioridad.
    - Se inserta el nodo creado en la cola de prioridad.

3. La raíz del árbol de Huffman será el último nodo que queda en la cola de prioridad.
4. Se generan los códigos de Huffman para cada carácter recorriendo el árbol de Huffman desde la raíz hasta cada hoja. El código de Huffman de un carácter es la secuencia de direcciones agregando un 0 si se dirige a la izquierda y agregando un 1 si toma un camino a la derecha recorriendo el árbol desde la raíz hasta llegar al nodo que contiene el carácter. (University of Toronto, 2023)

Análisis de complejidad del algoritmo:

1. Crear el nodo para cada carácter y almacenarlos en una cola de prioridad toma un tiempo de $O(n\,log(n))$, donde n es el número de caracteres diferentes debido a que cada inserción en una cola de prioridad toma un tiempo logarítmico y esta contiene n cantidad de inserciones.

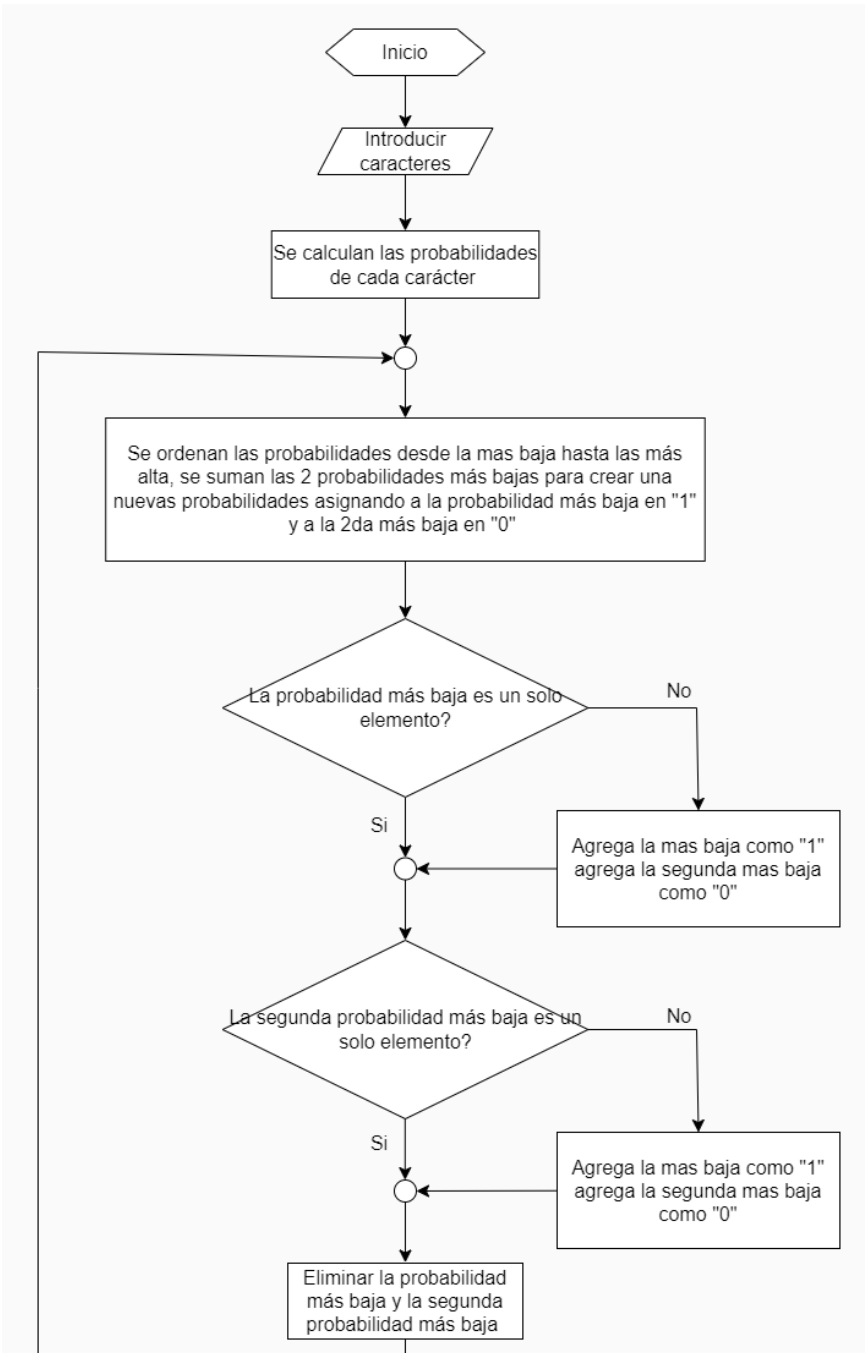
2. Para retirar el nodo menor y el segundo nodo con menor frecuencia de la cola de prioridad y la creación de un nuevo nodo que tenga como hijos a los nodos retirados se requiere un tiempo de $O(n\,log(n))$, ya que cada operación de extracción de un nodo de una cola de prioridad toma un tiempo logarítmico y hay $2\,n\,-\,1$ operaciones de extracción.

3. El trasplante del último nodo dentro de la cola de prioridad tomará un tiempo constante de $O(1)$ y este nodo será la raíz del árbol de Huffman.

4. El recorrido del árbol de Huffman para realizar la codificación de Huffman desde la raíz hasta llegar al caracter tomando en cuenta que se necesita recorrer cada nodo del árbol una vez toma un tiempo $O(n)$.

Concluyendo que la complejidad en la ejecución del algoritmo de Huffman es $O(n\,log(n))$.

Flujograma del algoritmo de Huffman



Análisis del algoritmo de Huffman

Caso base.

Este caso base o caso trivial se da cuando solo se trabaja con un solo dato, teniendo esto no es necesario la construcción del árbol de Huffman debido a que solo se tiene un dato, por esto todo se trabaja de una manera constante, por lo tanto, la complejidad del algoritmo es de orden $O(1)$ en este caso.

Mejor caso.

El mejor caso dentro del algoritmo de Huffman sucede cuando en todos los caracteres de los datos de entrada se tiene la misma frecuencia o probabilidad, cuando esto ocurre dentro del algoritmo el árbol de Huffman se equilibra de la manera más óptima posible, por lo tanto, la longitud de la codificación de Huffman es uniforme y mínima. Como resultado logramos que la complejidad del algoritmo sea de orden $O(n\,log(n))$.

La manera para determinar que se tiene esta complejidad dentro del algoritmo se debe construir un montículo de mínimos con los símbolos iniciales, teniendo en cuenta que tenemos el mejor caso, el montículo de mínimos se mantendrá equilibrado en la insercion y extraccion del mínimo del montículo tendrá una complejidad de $O(log(n))$.

Cuando se ejecuta la construcción del árbol de Huffman, se combinan repetidamente los dos nodos con las frecuencias más bajas y se vuelven a insertar dentro del árbol. Cada vez que se realiza una iteración, se identifican y extraen los dos nodos más bajos, se suman y se crea un nodo con el resultado de la suma.

Como se tienen n datos, el número total de operaciones de combinación de los nodos más bajos es de $n - 1$, siendo n el número de datos ingresados. Como

resultado tenemos que la complejidad de generación de códigos de Huffman para n símbolos en el mejor caso es de $O(n\ log(n))$.

Peor caso.

El peor caso de la codificación de Huffman ocurre cuando un carácter tiene una frecuencia muy alta y distanciada de las frecuencias de los demás caracteres, esto sucede cuando el árbol de Huffman se ordena de forma desbalanceada haciendo que la codificación de los caracteres con una frecuencia menor tengan una longitud más larga, siendo mucho menos eficiente en la ejecución.

Para determinar el orden de magnitud se analiza la construcción del montículo de mínimos ya que necesita ser reordenado de manera frecuente, gracias al desbalance en la inserción de los nodos que tienen frecuencias distintas y dando como resultado que la complejidad en el peor caso sea de $O(n\ log(n))$ cuando n son el número de caracteres insertados pero la longitud de los códigos de cada carácter será mayor que en el mejor caso.

En el peor de los casos cuando se realiza la construcción del árbol de Huffman es poco eficiente a causa del distanciamiento de las frecuencias entre los caracteres, al momento de extraer el nodo más bajo y el segundo más bajo para realizar la suma, entonces, el montículo de mínimos necesita balancearse de nuevo, teniendo así, un alto número de ejecuciones en la inserción y extracción de nodos.

Conclusiones del resultado de los análisis

La codificación de Huffman es un algoritmo que se encarga de la compresión de datos y esta se utiliza para minimizar la longitud total de las palabras del mensaje. Su rendimiento y eficiencia depende a grandes rasgos de las probabilidades de los caracteres en la entrada.

Comprendiendo que cuando solo se encuentra un carácter dentro del algoritmo siendo este el caso base se comporta de forma muy eficiente, generando un código de longitud 1 para ese carácter. Al igual que las probabilidades son iguales entendemos este comportamiento como el mejor caso del algoritmo, se genera un árbol binario perfectamente equilibrado, lo que genera en la codificación de Huffman una longitud igual a los códigos de todos los caracteres.

También el peor caso se presenta cuando las probabilidades están en una distribución de tipo exponencial, el cual genera un árbol binario desbalanceado, resultando en códigos con una longitud que varía para cada carácter, siendo esto menos eficiente debido a que la longitud de los códigos aumenta.

Concluyendo en que el algoritmo de Huffman es generalmente eficiente en términos de tiempo pero la longitud total de los códigos dependiendo de la distribución del árbol en relación a las probabilidades de los caracteres hace variar la eficiencia en la codificación.

Bibliografía

References

Algoritmo de Huffman. (n.d.). Los diccionarios y las enciclopedias sobre el Académico. Retrieved November 14, 2023, from https://es-academic.com/dic.nsf/eswiki/65575

Comprimiendo datos - el algoritmo de Huffman en Python. (2016, March 28). Bit y Byte. Retrieved November 13, 2023, from https://bitybyte.github.io/Huffman-coding/

David Huffman. (n.d.). EcuRed. Retrieved November 14, 2023, from https://www.ecured.cu/David_Huffman

Garsia, A., & Wachs, M. L. (n.d.). *Codificación Huffman*. AcademiaLab. Retrieved November 13, 2023, from https://academia-lab.com/enciclopedia/codificacion-huffman/

Maluenda, R. (n.d.). *Qué es un algoritmo informático: características, tipos y ejemplos*. Profile. Retrieved November 20, 2023, from https://profile.es/blog/que-es-un-algoritmo-informatico/#%C2%BFQue_es_un_algoritmo_informaticohttps://profile.es/blog/que-es-un-algoritmo-informatico/#%C2%BFQue_es_un_algoritmo_informatico