Nombre: Juan David Yepez Vélez.

Codigo: 30000089783.

Curso: Análisis de algoritmos.

Actividad: Códigos de Huffman – Algoritmos voráces.

Códigos de Huffman.

Los códigos de Huffman son un método de compresión de datos que permite representar información de manera eficiente, asignando códigos binarios de longitud variable a un conjunto de símbolos. La principal característica de este método es que los símbolos con mayor frecuencia de aparición reciben códigos más cortos, mientras que los menos frecuentes reciben códigos más largos. Esto minimiza el tamaño total del mensaje codificado.

Una propiedad clave de los códigos de Huffman es que son códigos prefijo, lo que significa que ningún código es el prefijo de otro. Esto garantiza que el mensaje pueda ser decodificado de manera unívoca y sin ambigüedades. Su eficiencia y simplicidad lo convierten en un algoritmo ampliamente utilizado en aplicaciones como la compresión de archivos o la transmisión de datos.

Algoritmo.

El algoritmo de Huffman construye un árbol binario óptimo para representar códigos. Comienza inicializando una cola de prioridad Q con los símbolos y sus frecuencias, de forma que los de menor frecuencia estén al frente. Luego, en un bucle que se ejecuta n−1 veces (donde n es el número de símbolos), extrae los dos nodos con las frecuencias más bajas (x y y) de la cola. Estos nodos se combinan en un nuevo nodo z cuyo peso es la suma de las frecuencias de x y y. El nodo z se inserta de nuevo en la cola de prioridad. Este proceso asegura que los nodos con menores frecuencias se encuentran más cerca de la raíz, optimizando la longitud de los códigos binarios generados. Al final, el único nodo restante en la cola es la raíz del árbol de Huffman, que contiene la estructura completa de los códigos prefijo.

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4       allocate a new node z
5       z.left = x = EXTRACT-MIN(Q)
6       z.right = y = EXTRACT-MIN(Q)
7       z.freq = x.freq + y.freq
8       INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```

Código.

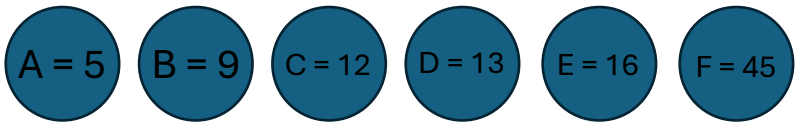
Este código implementa el algoritmo de Huffman para la compresión de datos. Comienza definiendo una clase *Node*, que representa los nodos del árbol de Huffman, donde cada nodo contiene un carácter (*char*), su frecuencia (*freq*), y referencias a sus hijos izquierdo y derecho. La función *huffman_tree* construye el árbol de Huffman tomando un diccionario de frecuencias de caracteres. Utiliza una cola de prioridad implementada con el módulo *heapq*, donde los nodos se ordenan por frecuencia, extrayendo siempre los dos nodos de menor frecuencia y combinándolos en un nuevo nodo cuya frecuencia es la suma de las frecuencias de los nodos combinados. Este proceso se repite hasta que queda un solo nodo, que es la raíz del árbol. Luego, la función *generate_huffman_codes* recorre el árbol de Huffman de manera recursiva, asignando un código binario a cada carácter. Los códigos de Huffman se generan siguiendo el camino desde la raíz hasta cada hoja, donde cada '0' indica un movimiento a la izquierda y cada '1' a la derecha. Finalmente, se imprime el código de Huffman para cada carácter en el diccionario. Este enfoque se utiliza comúnmente para la compresión de datos, ya que asigna códigos **más** cortos a los caracteres más frecuentes, optimizando la representación binaria de los datos.

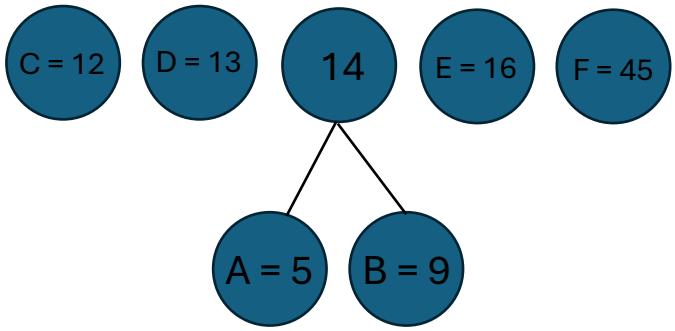
```python
import heapq

# Definición de la clase Nodo
class Node:
    def __init__(self, char, freq):
        self.char = char  # Carácter
        self.freq = freq  # Frecuencia
        self.left = None  # Nodo izquierdo
        self.right = None  # Nodo derecho

    def __lt__(self, other):
        return self.freq < other.freq  # Para que heapq ordene los nodos por frecuencia

# Función para construir el árbol de Huffman
def huffman_tree(frequencies):
    # Crear una cola de prioridad (heap) con nodos de Huffman
    priority_queue = []

    # Crear un nodo para cada carácter y agregarlo a la cola
    for char, freq in frequencies.items():
        heapq.heappush(priority_queue, Node(char, freq))

    # Construcción del árbol de Huffman
    while len(priority_queue) > 1:
        # Extraer los dos nodos con menor frecuencia
        left = heapq.heappop(priority_queue)
        right = heapq.heappop(priority_queue)

        # Crear un nodo padre cuya frecuencia es la suma de las frecuencias de los dos nodos extraídos
        merged_node = Node(None, left.freq + right.freq)
        merged_node.left = left
        merged_node.right = right

        # Insertar el nodo fusionado de nuevo en la cola
        heapq.heappush(priority_queue, merged_node)

    # El único nodo restante es la raíz del árbol de Huffman
    return priority_queue[0]
```

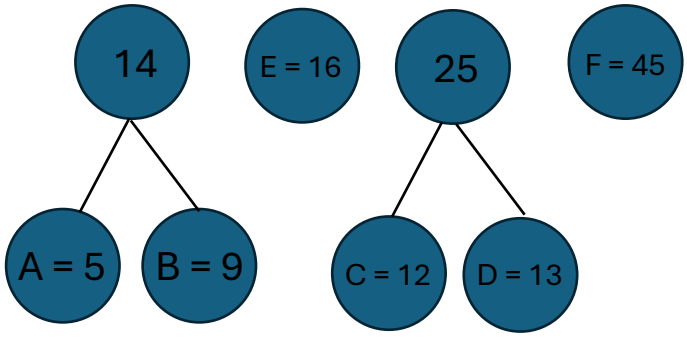
```python
# Función para generar el código de Huffman
def generate_huffman_codes(node, prefix="", codebook={}):
    if node is not None:
        # Si el nodo es una hoja (un carácter), asignar el código
        if node.char is not None:
            codebook[node.char] = prefix
        # Recursión sobre los hijos izquierdo y derecho
        generate_huffman_codes(node.left, prefix + "0", codebook)
        generate_huffman_codes(node.right, prefix + "1", codebook)
    return codebook

# Ejemplo de uso
if __name__ == "__main__":
    # Frecuencias de los caracteres
    frequencies = {
        'a': 5,
        'b': 9,
        'c': 12,
        'd': 13,
        'e': 16,
        'f': 45
    }

    # Construir el árbol de Huffman
    root = huffman_tree(frequencies)

    # Generar el código de Huffman
    huffman_codes = generate_huffman_codes(root)

    # Mostrar los códigos de Huffman
    print("Códigos de Huffman:")
    for char, code in huffman_codes.items():
        print(f"{char}: {code}")
```

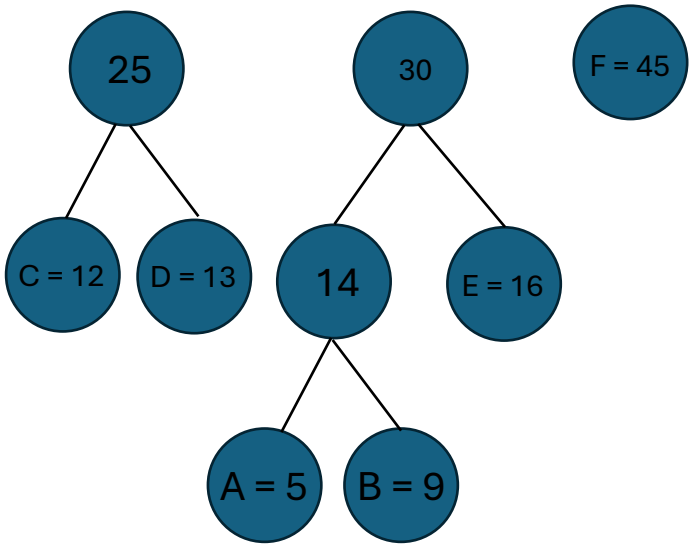
Árbol de Huffman.

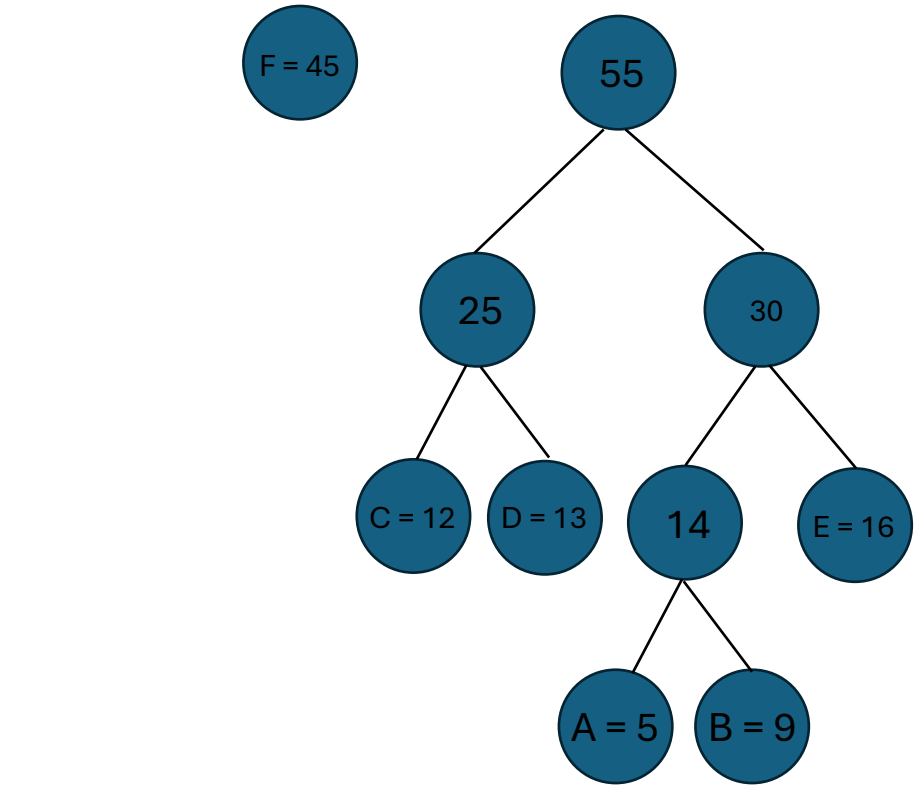
A continuación, se presenta el árbol de Huffman para el ejemplo realizado en el código anterior. Esto se hace para permitir una **compresión eficiente de datos** al asignar códigos binarios de longitud variable a los símbolos, lo que resulta en un uso más eficiente del espacio.

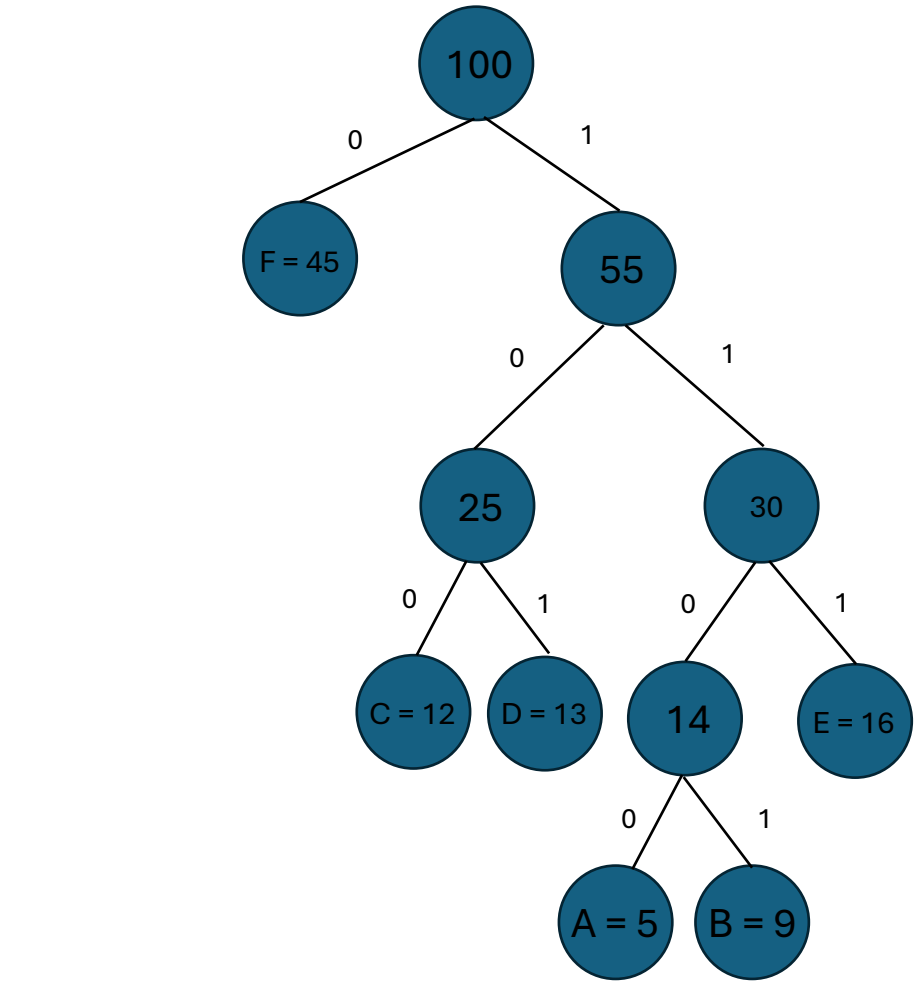

A = 5
B = 9
C = 12
D = 13
E = 16
F = 45


C = 12
D = 13
14
E = 16
F = 45
A = 5
B = 9


14
E = 16
25
F = 45
A = 5
B = 9
C = 12
D = 13


25
30
F = 45
C = 12
D = 13
14
E = 16
A = 5
B = 9


F = 45
55
25
30
C = 12
D = 13
14
E = 16
A = 5
B = 9


100
0
1
F = 45
55
0
1
25
30
0
1
0
1
C = 12
D = 13
14
E = 16
0
1
A = 5
B = 9

Codificación.

Luego de haber armado el árbol de Huffman de acuerdo a las frecuencias de cada letra, se puede armar la codificación de cada una de estas para comprimir la información. Finalmente obtenemos lo siguiente:

Letra	Frecuencia	Código
A	5	1100
B	9	1101
C	12	100
D	13	101
E	16	111
F	45	0

Al asignar códigos más cortos a los símbolos que aparecen con mayor frecuencia y códigos más largos a los menos frecuentes, optimiza el uso del espacio de almacenamiento o de transmisión de datos. Esto es especialmente útil en áreas como la compresión de texto, imágenes y audio, permitiendo ahorrar recursos como ancho de banda y espacio en disco, lo que mejora la eficiencia de los sistemas que gestionan grandes volúmenes de datos.