Pseudocódigo: Códigos de Huffman con Algoritmo Greedy

Algoritmo Greedy

25 de septiembre de 2025

1. Descripción del Problema

El problema de codificación de Huffman consiste en encontrar el código de longitud variable óptimo para un conjunto de símbolos dados sus frecuencias de aparición. El objetivo es minimizar la longitud promedio de codificación, que se calcula como:

$$\text{Costo} = \sum_{i=1}^{n} f_i \cdot l_i$$

donde f_i es la frecuencia del símbolo i y l_i es la longitud del código asignado al símbolo i.

El algoritmo greedy de Huffman resuelve este problema construyendo un árbol binario óptimo mediante la selección repetitiva de los dos nodos con menor frecuencia y combinándolos en un nuevo nodo interno.

2. Pseudocódigo

Algorithm 1 Construcción del Árbol de Huffman

Require: Lista de símbolos s y frecuencias $freq$
Ensure: Lista de códigos Huffman para cada símbolo

$n \leftarrow$ longitud de s
$pq \leftarrow$ cola de prioridad (min-heap) vacía
{Crear nodos hoja para cada símbolo}
for $i = 0$ **to** $n - 1$ **do**
    $tmp \leftarrow Node(freq[i])$
    $heapq.push(pq, tmp)$
end for
{Construir árbol de Huffman}
while $|pq| \geq 2$ **do**
    $l \leftarrow heapq.pop(pq)$ {Nodo izquierdo (menor frecuencia)}
    $r \leftarrow heapq.pop(pq)$ {Nodo derecho (segunda menor frecuencia)}
    $newNode \leftarrow Node(l.data + r.data)$
    $newNode.left \leftarrow l$
    $newNode.right \leftarrow r$
    $heapq.push(pq, newNode)$
end while
$root \leftarrow heapq.pop(pq)$
$ans \leftarrow$ lista vacía
$preOrder(root, ans, "")$
return ans

Algorithm 2 Recorrido Preorden para Generar Códigos

Require: Nodo raíz $root$, lista de respuestas ans, código actual $curr$
Ensure: Lista de códigos actualizada

```
if root = null then
    return
end if
if root.left = null and root.right = null then
    ans.append(curr) {Nodo hoja representa un carácter}
    return
end if
preOrder(root.left, ans, curr +' 0')
preOrder(root.right, ans, curr +' 1')
```

3. Explicación del Algoritmo

3.1. Estrategia Greedy

El algoritmo greedy de Huffman utiliza la siguiente estrategia:

- **Selección greedy:** En cada iteración, selecciona los dos nodos con menor frecuencia
- **Combinación:** Combina estos nodos en un nuevo nodo interno cuya frecuencia es la suma
- **Construcción incremental:** Construye el árbol de abajo hacia arriba
- **Optimalidad:** La elección greedy garantiza la construcción del árbol óptimo

La elección de combinar siempre los dos nodos de menor frecuencia es óptima porque:

- Los símbolos con menor frecuencia deben tener los códigos más largos
- Al combinar los dos de menor frecuencia, se minimiza el costo total

3.2. Propiedad de Elección Greedy

El algoritmo de Huffman satisface la propiedad de elección greedy:

Teorema: Si x y y son los dos caracteres con menor frecuencia, entonces existe un código óptimo donde x y y son hermanos (comparten el mismo padre) y tienen las longitudes de código más largas.

Demostración intuitiva:

- Si x y y no son hermanos en algún código óptimo, se puede intercambiar con sus hermanos actuales
- Esto no aumenta el costo total y mantiene la optimalidad
- Por lo tanto, siempre es seguro combinarlos primero

3.3. Complejidad

- **Tiempo:** $O(n \log n)$
  - $O(n)$ para crear los nodos iniciales
  - $O(n \log n)$ para las operaciones del heap (n-1 extracciones y inserciones)
  - $O(n)$ para el recorrido preorden
- **Espacio:** $O(n)$ para el heap y el árbol resultante

La eficiencia del algoritmo se debe al uso de un min-heap para mantener los nodos ordenados por frecuencia.

4. Resolución Paso a Paso

4.1. Ejemplo: Símbolos = .ªbcdef", Frecuencias = [5, 9, 12, 13, 16, 45]

Paso 1: Identificar parámetros del problema

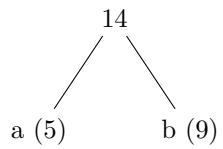
Símbolo	Frecuencia
a	5
b	9
c	12
d	13
e	16
f	45

Paso 2: Inicialización del heap

- Crear nodos hoja para cada símbolo
- Heap inicial: [5(a), 9(b), 12(c), 13(d), 16(e), 45(f)]
- Orden en heap: a(5), b(9), c(12), d(13), e(16), f(45)

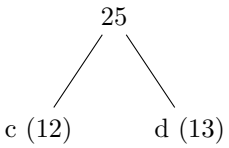
Paso 3: Construcción del árbol - Iteración 1

- Extraer: l = a(5), r = b(9)
- Crear nodo interno: 5 + 9 = 14
- Heap actualizado: [12(c), 13(d), 14(ab), 16(e), 45(f)]



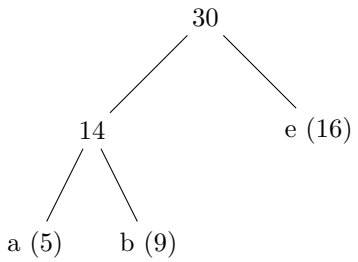
Paso 4: Construcción del árbol - Iteración 2

- Extraer: l = c(12), r = d(13)
- Crear nodo interno: 12 + 13 = 25
- Heap actualizado: [14(ab), 16(e), 25(cd), 45(f)]



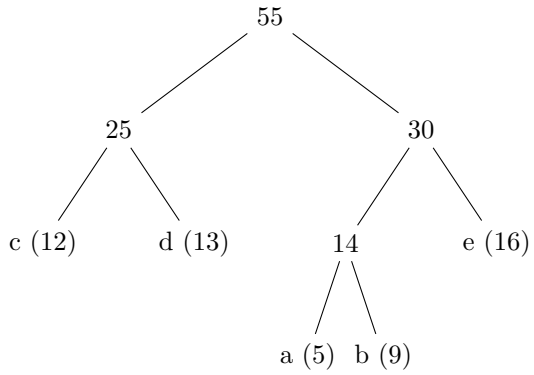
Paso 5: Construcción del árbol - Iteración 3

- Extraer: l = ab(14), r = e(16)
- Crear nodo interno: 14 + 16 = 30
- Heap actualizado: [25(cd), 30(abe), 45(f)]



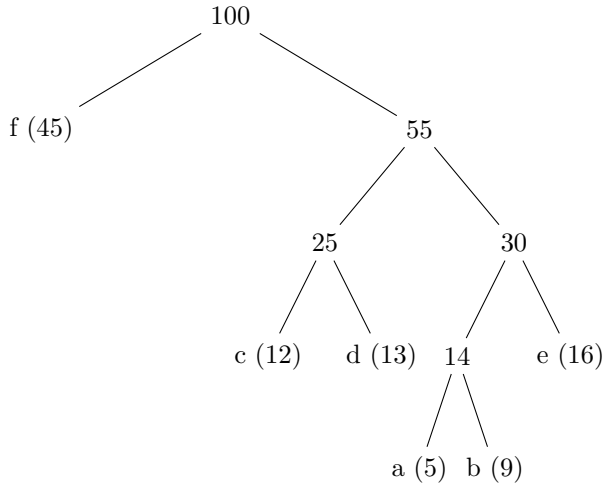
Paso 6: Construcción del árbol - Iteración 4

- Extraer: l = cd(25), r = abe(30)
- Crear nodo interno: 25 + 30 = 55
- Heap actualizado: [45(f), 55(abcde)]



Paso 7: Construcción del árbol - Iteración Final

- Extraer: l = f(45), r = abcde(55)
- Crear nodo raíz: 45 + 55 = 100
- Heap: [] (vacío)



Paso 8: Generación de códigos mediante recorrido preorden

Símbolo	Camino	Código
f	Raíz → Izquierda	0
c	Raíz → Derecha → Izquierda → Izquierda	100
d	Raíz → Derecha → Izquierda → Derecha	101
a	Raíz → Derecha → Derecha → Izquierda → Izquierda	1100
b	Raíz → Derecha → Derecha → Izquierda → Derecha	1101
e	Raíz → Derecha → Derecha → Derecha	111

Resultado Final:

- f: 0
- c: 100
- d: 101
- a: 1100
- b: 1101
- e: 111

Paso 9: Cálculo del costo total

$$\begin{aligned} \text{Costo} &= 45 \cdot 1 + 12 \cdot 3 + 13 \cdot 3 + 5 \cdot 4 + 9 \cdot 4 + 16 \cdot 3 && (1) \\ &= 45 + 36 + 39 + 20 + 36 + 48 && (2) \\ &= 224 && (3) \end{aligned}$$

5. Análisis del Algoritmo

5.1. Optimalidad del Algoritmo de Huffman

El algoritmo de Huffman garantiza encontrar la codificación óptima porque:

- **Subestructura óptima:** El problema se puede dividir en subproblemas óptimos
- **Propiedad de elección greedy:** La elección local óptima lleva a una solución global óptima
- **Demostración:** Se puede demostrar que cualquier código óptimo puede transformarse en el código de Huffman sin aumentar el costo

Comparación con fuerza bruta:

- Huffman: $O(n \log n)$ - Solución óptima en tiempo polinomial
- Fuerza bruta: $O(C_{n-1} \cdot n! \cdot n)$ - Solución óptima en tiempo exponencial

5.2. Ventajas del Algoritmo

- **Eficiencia:** Complejidad $O(n \log n)$
- **Optimalidad:** Garantiza la solución óptima
- **Simplicidad:** Fácil de implementar
- **Aplicabilidad:** Ampliamente usado en compresión de datos

5.3. Análisis del Ejemplo

Para el ejemplo dado, el algoritmo produce:

- **Longitudes de código:** [4, 4, 3, 3, 3, 1]
- **Costo total:** 224 bits
- **Eficiencia:** El símbolo más frecuente (f) tiene el código más corto (1 bit)
- **Balance:** Los símbolos menos frecuentes tienen códigos más largos

6. Conclusiones

El algoritmo greedy de Huffman es una solución elegante y eficiente al problema de codificación óptima. Su fortaleza radica en:

- La demostración matemática de su optimalidad
- Su complejidad polinomial $O(n \log n)$
- Su amplia aplicación en sistemas de compresión de datos
- La simplicidad de su implementación

Este algoritmo demuestra el poder de las estrategias greedy cuando se aplican correctamente a problemas que exhiben subestructura óptima y la propiedad de elección greedy. Es considerado uno de los algoritmos más importantes en el campo de la compresión de datos.