Universidad de San Buenaventura

Facultad ingeniería de sistemas

UNIVERSIDAD DE
SAN BUENAVENTURA
CALI

Taller 5 Corte 3

Análisis de algoritmos

Presenta:

Juan Felipe Hurtado Villani

Samuel Martínez

Codificación Huffman:

Prefacio (se puede omitir):
La codificación de Huffman puede comprimir datos de forma muy eficaz: normalmente puede ahorrar entre un 20% y un 90% de espacio, y la tasa de compresión específica depende de las características de los datos. Consideramos que los datos se comprimen como una secuencia de caracteres. De acuerdo con la frecuencia de cada personaje, el algoritmo codicioso de Huffman construye la representación binaria óptima del personaje.
Suponga que queremos comprimir un archivo de datos de 10 caracteres. La siguiente tabla muestra los caracteres que aparecen en el archivo y su frecuencia. En otras palabras, solo aparecieron 6 caracteres diferentes en el archivo, de los cuales el personaje a apareció 45.000 veces.

	a	b	c	d	e	f
Frecuencia (miles)	45	13	12	16	9	5
Codificación de longitud fija	000	001	010	011	100	101
Codificación de longitud variable	0	101	100	111	1101	1100

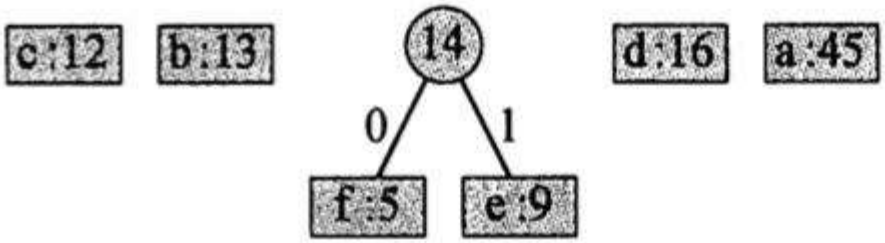
- Un problema de codificación de caracteres. Un archivo de 100.000 caracteres contiene solo 6 caracteres diferentes de la A la F. La frecuencia de aparición se muestra en la tabla anterior. Si especificamos una palabra de código de 3 dígitos para cada carácter, podemos codificar el archivo con una longitud de 300 000 bits. Pero usando la codificación de longitud variable en la tabla anterior, podemos codificar el archivo con solo 224,000 bits. Entonces vemos que esta es la codificación de caracteres óptima para el archivo.

Así que conocemos la codificación de Huffman.
La codificación de Huffman es un tipo de codificación de longitud de palabra variable. Huffman diseñó un algoritmo codicioso para construir el código de prefijo óptimo, que se llama código de Huffman. Árbol de Huffman: El árbol de Huffman también se denomina árbol binario óptimo. Es el árbol binario con la menor longitud de ruta ponderada WPL entre todos los árboles binarios compuestos por n nodos de hojas ponderados.
A continuación, llegamos a comprender el proceso específico del algoritmo de Huffman
Nota: Cada vez que se construye el árbol binario, se ha ordenado en orden creciente de frecuencia de letras.
Paso: cada vez que se construye un árbol binario, los dos árboles de baja frecuencia se fusionan

1. Tomamos prestado el ejemplo anterior para obtener el nodo hoja de letra inicial:

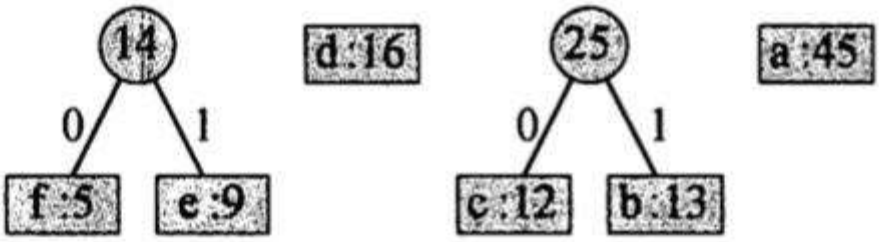
(a)

2. Combina f y e:



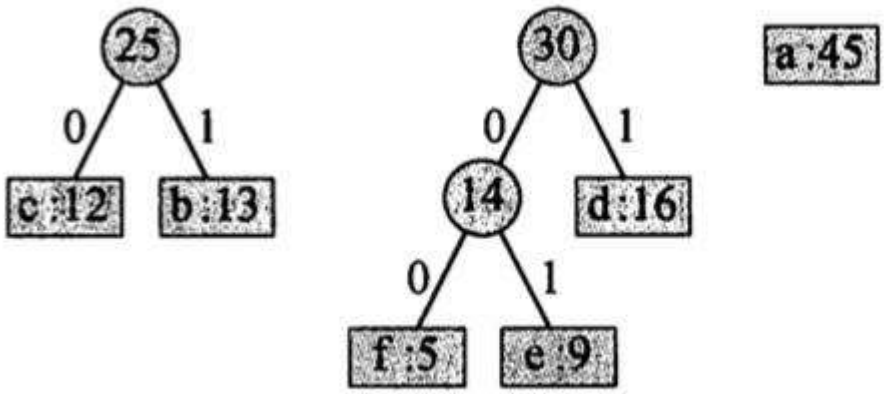
(b)

3. Combina cyb:



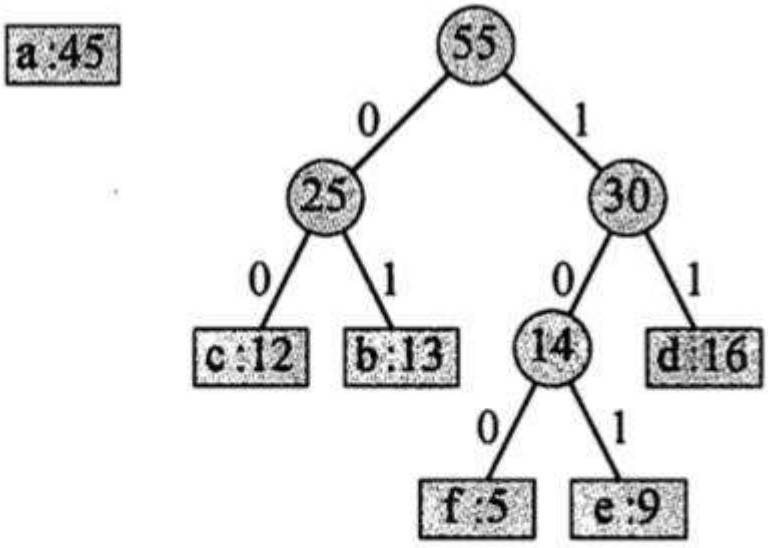
(c)

4. Combina 14 y d:



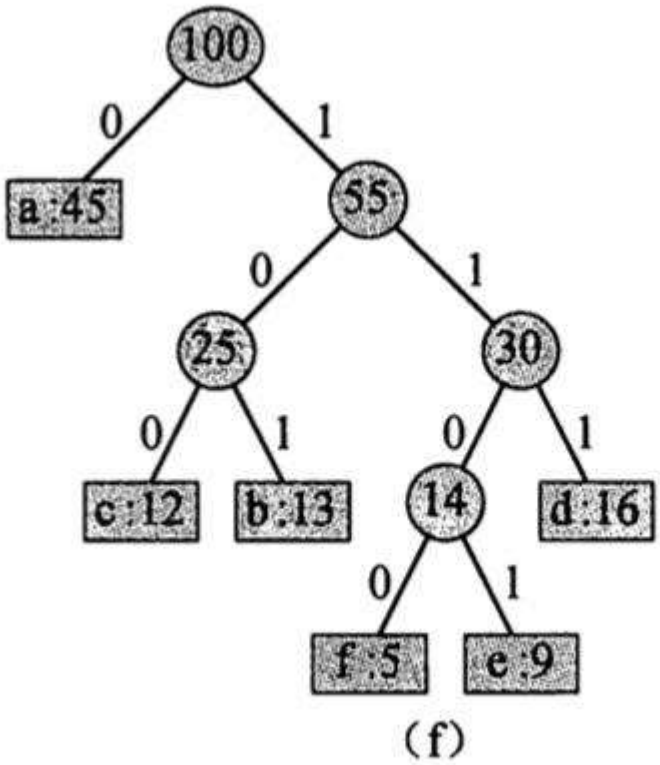
(d)

5. Combinar 25 y 30:



(e)

6. Combine 45 y 55, y finalmente obtenga un árbol Huffman completo:



(f)

Entonces, el código de Huffman es:
El código de a es: 0
El código de b es: 101
El código de c es: 100
El código de d es: 111
El código de e es: 1101
El código de f es: 1100

Código para implementar la codificación Huffman

1. Prepara la tabla de probabilidad para cada personaje.

```
def findTheCharFrequency(text):
    result = dict()
    with open(text,'r') as f:
        for line in f.readlines():
            line = line.lower()
            for i in line:
                if i.isalpha():     #Determina si el personaje es una letra inglesa
                    if i in result:
                        result[i] += 1
                    else:
                        result.update({i:1})
    return result
```

2. Crea una clase de nodo

```
class Node(object):
    def __init__(self, name=None, value=None):
        self.name = name
        self.value = value
        self.lchild = None
        self.rchild = None
```

3. Revertir el establecimiento del árbol de Huffman con nodos

```
#Crear árbol huffman
class HuffmanTree(object):
    # Según la idea del árbol de Huffman: basado en el nodo, construya el árbol de Huffman al revés
    def __init__(self, char_Weights):
        self.Leaf = [Node(k,v) for k, v in char_Weights.items()]
        while len(self.Leaf) != 1:
            self.Leaf.sort(key=lambda node:node.value, reverse=True)
            n = Node(value=(self.Leaf[-1].value + self.Leaf[-2].value))
            n.lchild = self.Leaf.pop(-1)
            n.rchild = self.Leaf.pop(-1)
            self.Leaf.append(n)
        self.root = self.Leaf[0]
        self.Buffer = list(range(10))
```

Genera códigos con pensamiento recursivo

```
    def Hu_generate(self, tree, length):
        node = tree
        if (not node):
            return
        elif node.name:
            print(node.name + La codificación de Huffman de 'es:', end='')
            for i in range(length):
                print(self.Buffer[i], end='')
            print('\n')
            return
        self.Buffer[length] = 0
        self.Hu_generate(node.lchild, length + 1)
        self.Buffer[length] = 1
        self.Hu_generate(node.rchild, length + 1)
```

Salida de código Huffman

```
    def get_code(self):
        self.Hu_generate(self.root, 0)
```

Salida

```
if __name__=='__main__':
    text = r'123.txt'
    result = findTheCharFrequency(text)
    print(result)
    tree = HuffmanTree(result)
    tree.get_code()
```

Salida de muestra:

```
La codificación Huffman de e es: 000
 La codificación de Huffman de f es: 001
 La codificación de Huffman de r es: 0100
 La codificación de Huffman de c es: 0101
 El código de s de Huffman es: 0110
 El código de Huffman de i es: 0111
 La codificación de Huffman de m es: 1000
 La codificación de Huffman de t es: 1001
 El código de Huffman de o es: 1010
 La codificación de Huffman de n es: 1011
 La codificación de Huffman de w es: 1100000
 El código Huffman de l es: 1100001
 La codificación Huffman de x es: 1100010
 La codificación de Huffman de b es: 11000110
 La codificación de Huffman de y es: 11000111
 La codificación de Huffman de d es: 11001
 La codificación de Huffman de un es: 1101
 La codificación de Huffman de h es: 1110
 La codificación de Huffman de p es: 111100
 La codificación Huffman de g es: 111101
 La codificación de Huffman de u es: 11111
```

Decodificación Huffman:

Pues resulta que decodificar es mucho más fácil que codificar. Sólo es necesario seguir los siguientes pasos:

1. Cargar el diccionario a la memoria
2. Empezar a leer bits del archivo binario (saltarse el primer 1).
3. Leer bits de uno en uno, y checar si la cadena de bits está en el diccionario
4. Si encontramos la cadena de bits en el diccionario, escribir al archivo de salida
5. Si es el final, terminar. Si no, ir al paso 3.

Como ven, es bastante simple: Leer bits, checar en el diccionario y si es el final, terminar.

Un ejemplo

Antes de presentar código, vamos a hacer un ejemplo rápido. Imaginemos que tenemos un archivo binario que contiene "10000101010111111110", y un diccionario que dice {'a':0, 'b':10, 'c':111, Fin:110}. Empezamos:

1. Cargamos el diccionario: dic = {'a':0, 'b':10, 'c':111, Fin:110}
2. Nos saltamos el primer bit. Entrada: "00001010101111111110"
3. Leemos un bit, y checamos en el diccionario. Bits leídos: "0".
   Entrada: "0001010101111111110"
4. ¿Está ‘bits leídos’ disponible en el diccionario? Sí: 'a':0. Salida: a.
5. Los siguientes tres bits son "000", o tres a. Salida: aaaa.
   Entrada: "101010111111110"
6. Leemos un bit. Bits leídos: "1". Entrada: "01010111111110". No está en el diccionario.
7. Leemos otro bit. Bits leídos "10". Entrada: "1010111111110". Diccionario: 'b':10. Salida: aaaab.
8. Los siguientes cuatro bits son "1010", o dos b. Salida: aaaabbb.
   Entrada: "111111110"

Y de continuar así, es posible ver que los bits que quedan en la entrada "111111110" corresponden a cc Fin.

```python
from node import node
"""
CREAR UN ARCHIVO Y ENVIAR UNA PALABRA EN EL
"""




# función de utilidad para imprimir huffman
# códigos para todos los símbolos en el nuevo
# árbol de Huffman creado


def printNodes(node, val=''):
# código huffman para el nodo actual
    newVal = val + str(node.huff)

    # si el nodo no es un nodo de borde
# luego atraviesa su interior
    if(node.left):
        printNodes(node.left, newVal)
    if(node.right):
        printNodes(node.right, newVal)
# si el nodo es un nodo de borde, entonces
# mostrar su código huffman
    if(not node.left and not node.right):
        file.write(f"{node.symbol} -------- {newVal}\n")


# personajes para huffman tree
chars = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '.', ' ', ',']
freq = []

filename=str(input("Ingrese el nombre del archivo (nombre.extension): "))
f=open(filename,"r")
frase=f.read().rstrip()
f.close
for x in range(len(chars)):
    counter=0
    for i in range(len(frase)):
        if frase[i]==chars[x]:
            counter+=1
    freq.append(counter)
```

```
45
46  nodes = []
47
48  # conversión de caracteres y frecuencias
49  # en los nodos del árbol de huffman
50  for x in range(len(chars)):
51      if freq[x]!=0:
52          nodes.append(node(freq[x], chars[x]))
53  numNodos=len(nodes)
54  while len(nodes) > 1:
55  # ordena todos los nodos en orden ascendente
56  # según su frecuencia
57      nodes = sorted(nodes, key=lambda x: x.freq)
58
59      # elige 2 nodos más pequeños
60      left = nodes[0]
61      right = nodes[1]
62
63      # asignar valor direccional a estos nodos
64      left.huff = 0
65      right.huff = 1
66
67  # combina los 2 nodos más pequeños para crear
68  # nuevo nodo como padre
69      newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)
70
71      # elimine los 2 nodos y agregue sus
72      #padre como nuevo nodo entre otros
73      nodes.remove(left)
74      nodes.remove(right)
75      nodes.append(newNode)
76
77  # ¡Huffman Tree está listo!
78  file = open("decodificacion.txt","w")
79  file.write("Numero de nodos: "+str(numNodos)+"\n")
80  file.write("Profundidad maxima del arbol generado: "+str(numNodos)+"\n")
81  file.write("Simbolo - Codigo\n")
82  printNodes(nodes[0])
83  file.close()
84
85
86
87
```

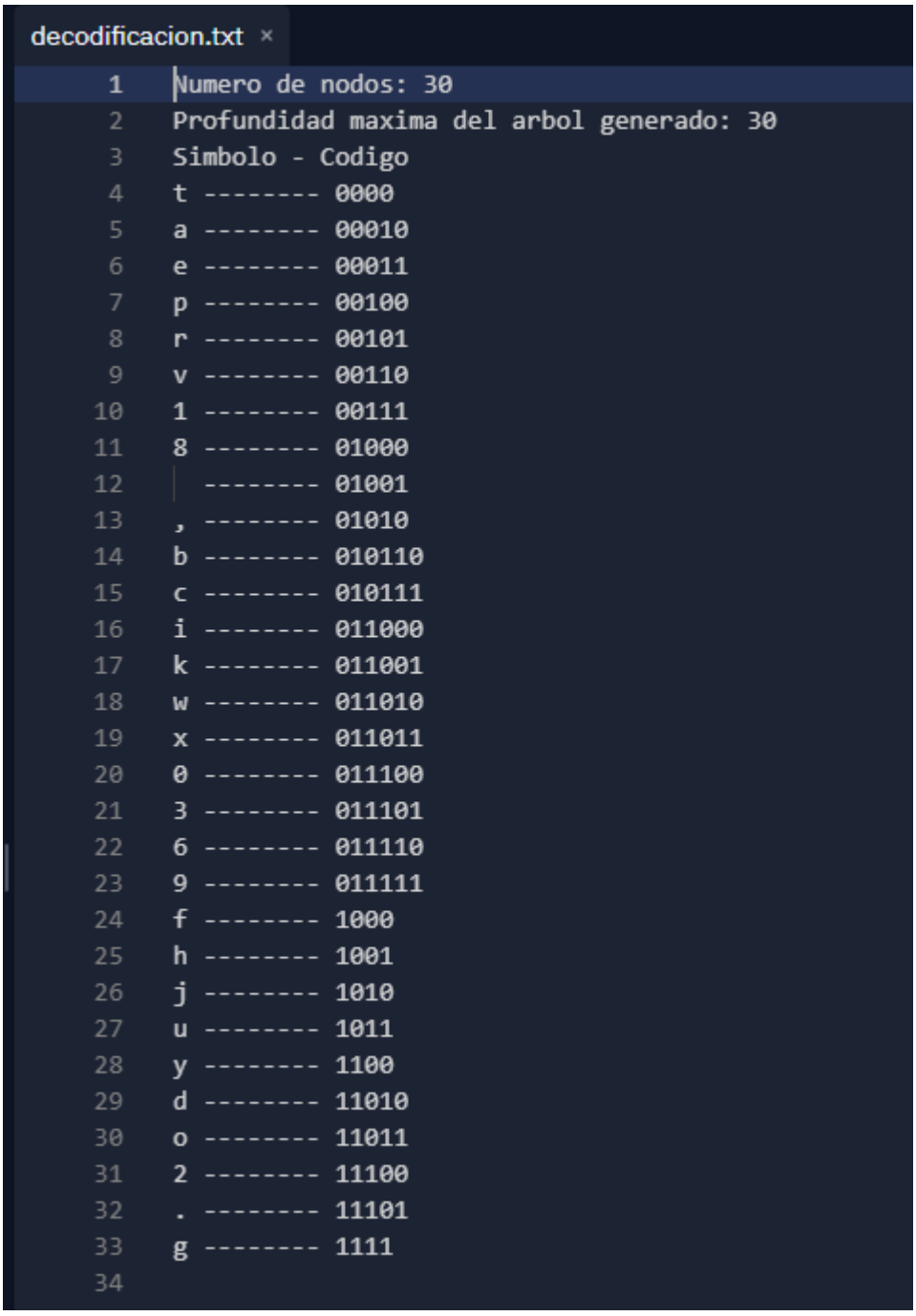
Este es el archivo que usaremos para decodificar un archivo con extensión “.txt”, el cual leerá y calculará el numero de nodos, profundidad del árbol y cada letra/símbolo con su respectivo código.

node.py

```python
class node:
    def __init__(self, freq, symbol, left=None, right=None):
        # frequency of symbol
        self.freq = freq

        # symbol name (character)
        self.symbol = symbol

        # node left of current node
        self.left = left

        # node right of current node
        self.right = right

        # tree direction (0/1)
        self.huff = ''
```

Este archivo es el que usaremos para mover dentro del archivo los nodos del árbol.

Este es el archivo que usamos de ejemplo que será el que se abrirá, y usaremos para la decodificación Huffman.



Este es el archivo file.txt decodificado con Huffman.