**Universidad de San Buenaventura**

**Facultad ingeniería de sistemas**



**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 |

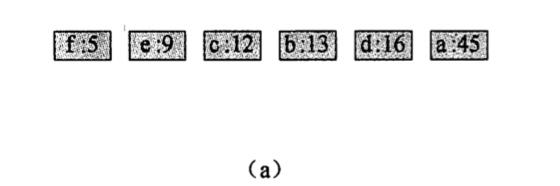
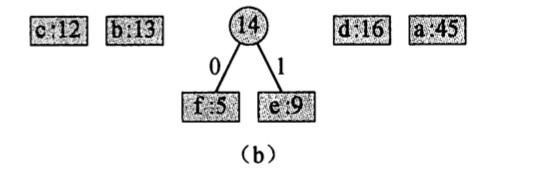
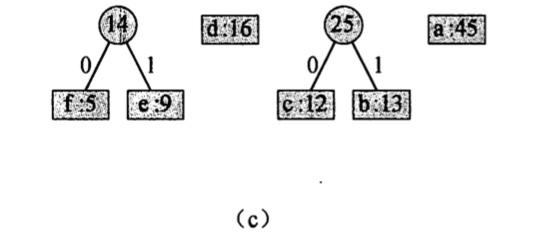
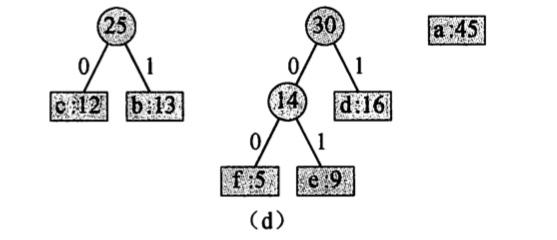
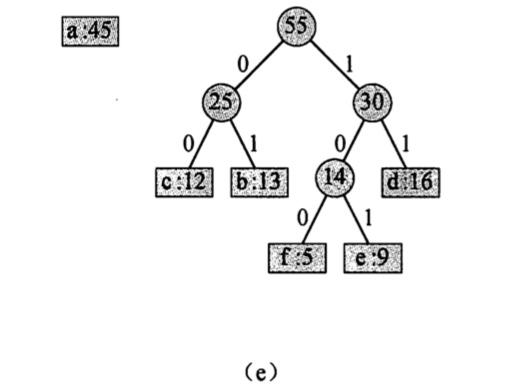
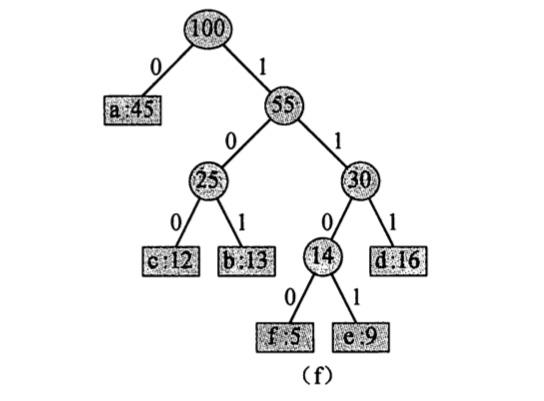
* Un problema de codificación de caracteres. Un archivo de 100.000 caracteres contiene solo 6 caracteres diferentes de la A la F. La frecuencia de aparición se muestra en la tabla anterior. Si especificamos una palabra de código de 3 dígitos para cada carácter, podemos codificar el archivo con una longitud 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

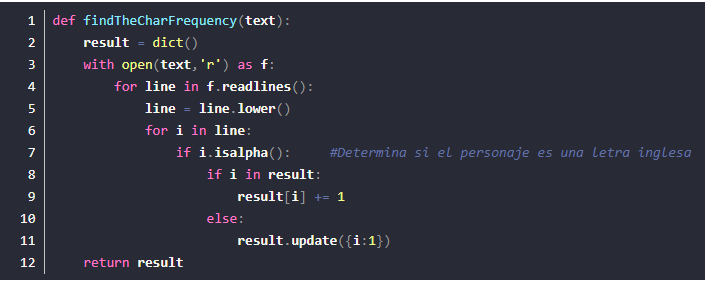
**Nota: Cada vez que se construye el árbol binario, se ha ordenado en orden creciente de frecuencia de letras.  
Paso: cada vez que se construye un árbol binario, los dos árboles de baja frecuencia se fusionan**

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

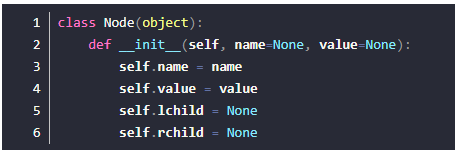
  
2. Combina f y e:  
  
3. Combina cyb:  
  
4. Combina 14 y d:  
  
5. Combinar 25 y 30:  
  
6. Combine 45 y 55, y finalmente obtenga un árbol Huffman completo:  
  
Entonces, el código de Huffman es:  
El código de a es: 0  
El código de b es: 101  
El código de c es: 100  
El código de d es: 111  
El código de e es: 1101  
El código de f es: 1100

### Código para implementar la codificación Huffman

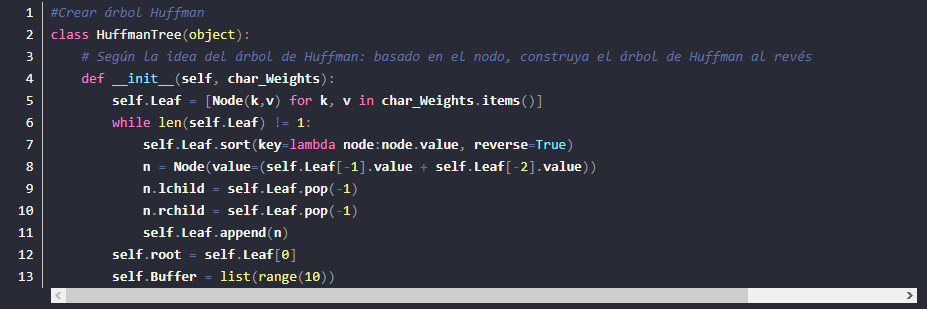
##### 1. Prepara la tabla de probabilidad para cada personaje.



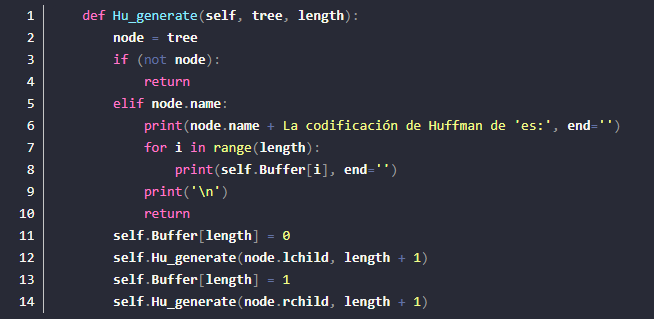
##### 2. Crea una clase de nodo



##### 3. Revertir el establecimiento del árbol de Huffman con nodos



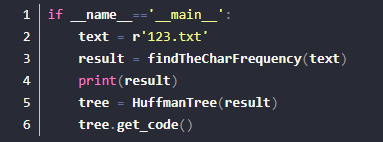
##### Genera códigos con pensamiento recursivo



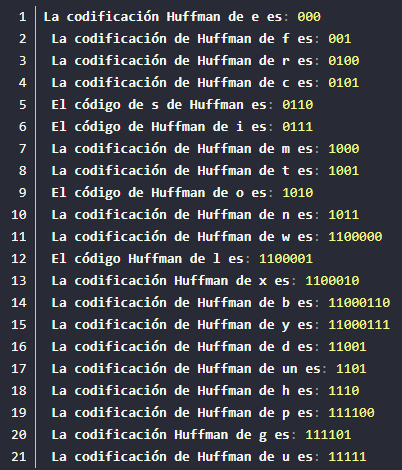
##### Salida de código Huffman



##### Salida



Salida de muestra:



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

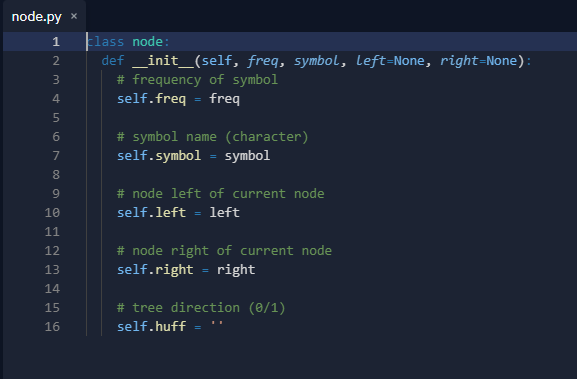
Antes de presentar código, vamos a hacer un ejemplo rápido. Imaginemos que tenemos un archivo binario que contiene "10000101010111111110", y un diccionario que dice {'a':0, 'b':10, 'c':111, Fin:110}. Empezamos:

1. Cargamos el diccionario: dic = {'a':0, 'b':10, 'c':111, Fin:110}
2. Nos saltamos el primer bit. Entrada: "0000101010111111110"
3. Leemos un bit, y checamos en el diccionario. Bits leídos: "0". Entrada: "000101010111111110"
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"

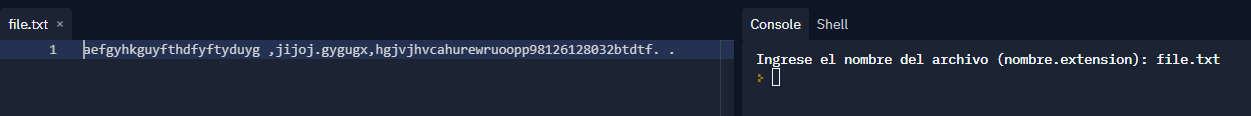
Y de continuar así, es posible ver que los bits que quedan en la entrada "111111110" corresponden a cc Fin.



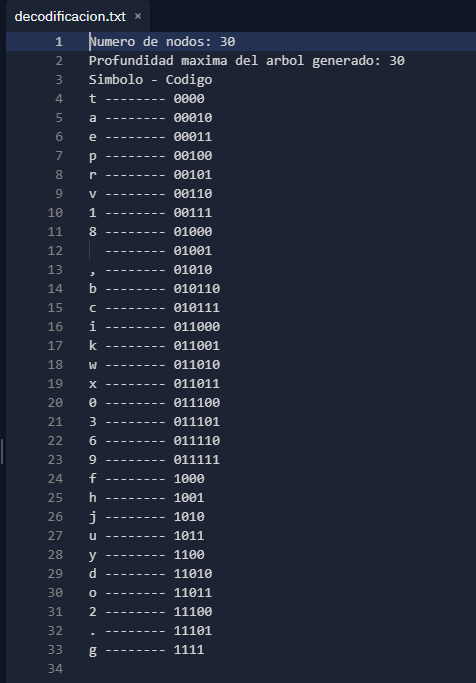
Este es el archivo que usaremos para decodificar un archivo con extensión “.txt”, el cual leerá y calculará el numero de nodos, profundidad del árbol y cada letra/símbolo con su respectivo código.



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.