ALGORITMIA

Grado en Ingeniería Informática del Software - Curso 2018-2019

GUIÓN DE LA PRÁCTICA 4

OBJETIVO:

- **Resolución de problemas con algoritmos voraces**

Codificación Huffman

La codificación de Huffman es un algoritmo de compresión de datos sin pérdida. Es decir, que a diferencia de otros algoritmos como MP3, los datos comprimidos utilizando el algoritmo de Huffman no pierden ningún tipo de información.

La idea es asignar códigos de longitud variable a los caracteres de entrada, siendo más cortos los códigos de los caracteres de entrada más frecuentes y más largos los códigos de los caracteres de entrada menos frecuentes.

Los códigos generados son códigos "prefijo". Eso significa que los códigos (secuencias de bits) se asignan de tal forma que el código asignado a un carácter nunca va a ser prefijo del código asignado a ningún otro carácter. Así, nunca habrá ambigüedad cuando se descodifica el flujo de bits generado tras aplicar el algoritmo de Huffman.

Códigos prefijo

Imaginemos que a las siguientes letras les asignamos los códigos de longitud variable indicados a continuación:

- a → 00
- b → 01
- c → 0
- d → 1

Si tuviéramos que descodificar la cadena `0001`, la salida sería ambigua, ya que se podría corresponder con `ab`, `cccd`, `ccb` o `acd`.

Sin embargo, cambiando los códigos asignados a códigos "prefijo", evitaríamos la ambigüedad. Por ejemplo:

- a → 1
- b → 01
- c → 001
- d → 000

Si ahora tuviéramos que descodificar cualquier cadena, como, por ejemplo `1001000011`, no habría ninguna ambigüedad y estaría claro que se trata de la cadena `acdba`.

Algoritmo: ¿cómo obtener los códigos prefijo mediante Huffman?

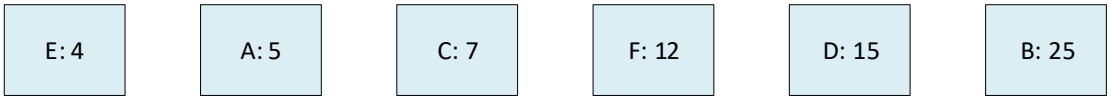
Supongamos que tenemos la siguiente entrada en un fichero de texto:

```
a 5
b 25
c 7
d 15
e 4
f 12
```

Dicha entrada representa la frecuencia con la que aparecen ciertos caracteres (también podrían ser cadenas de mayor tamaño). Por ejemplo, el carácter `a` tiene una frecuencia de 5, mientras que el carácter `b` tiene una frecuencia de 25. Como el carácter `b` se repite mucho más que el carácter `a`, nos conviene que la representación de `b` sea más pequeña que la de `a`, para así comprimir lo máximo posible la representación.

Paso 1:

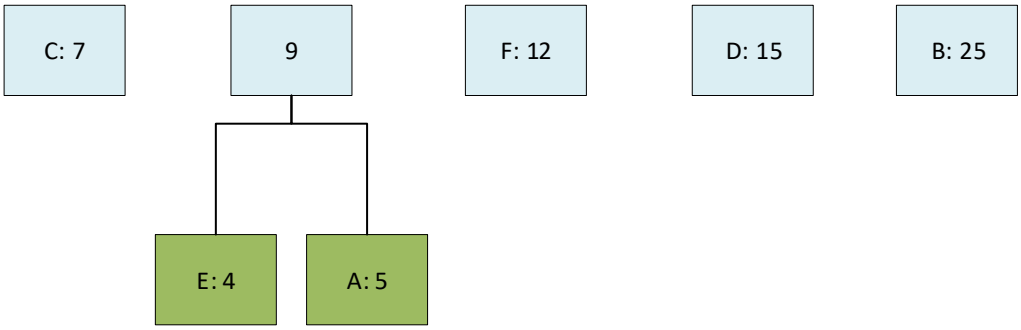
Creamos una cola de prioridades, ordenando a los elementos de menor a mayor frecuencia:



Paso 2:

Sacamos los dos elementos de la cola con menor frecuencia y los unimos en un nuevo elemento cuya frecuencia es la suma de las dos anteriores. El carácter (o la cadena) del nuevo elemento no importa, ya que es una combinación de los dos anteriores.

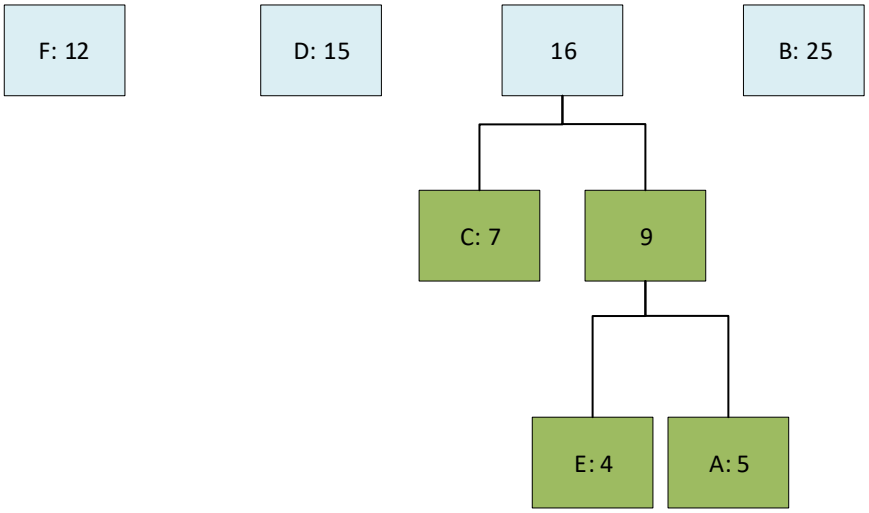
El nuevo elemento se introduce en la cola de prioridades y apunta, por su lado izquierdo al primero de los elementos que se sacó de la cola de prioridades, y por su lado derecho, al segundo de los elementos que se sacó de la cola de prioridades. Estamos comenzando a crear el llamado "árbol de Huffman".



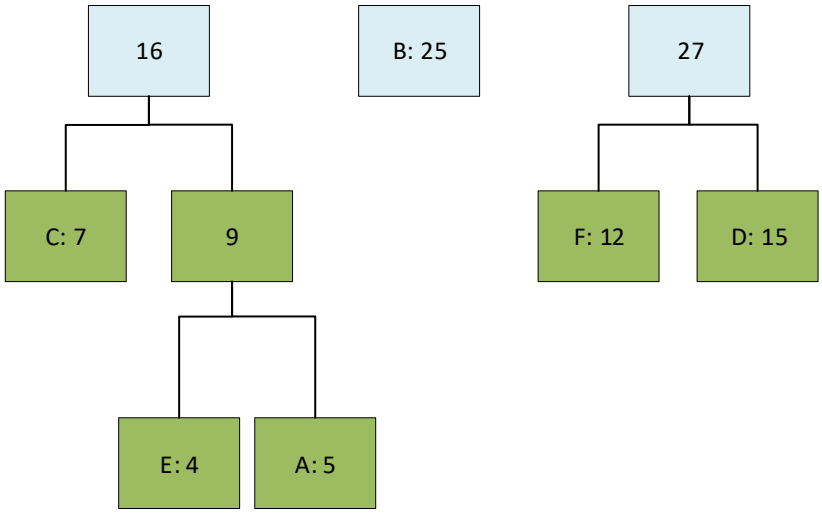
Paso 3:

Repetimos el paso 2 hasta que solo haya un elemento en la cola de prioridades:

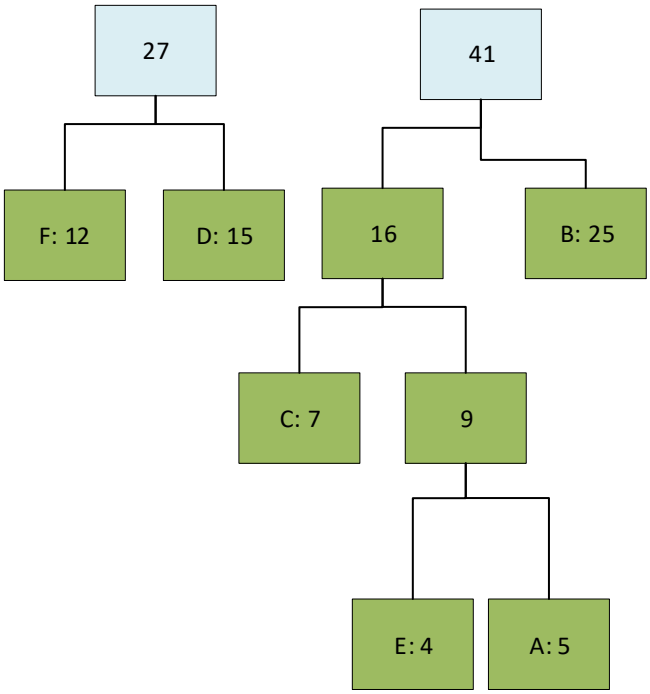
- Iteración 2:



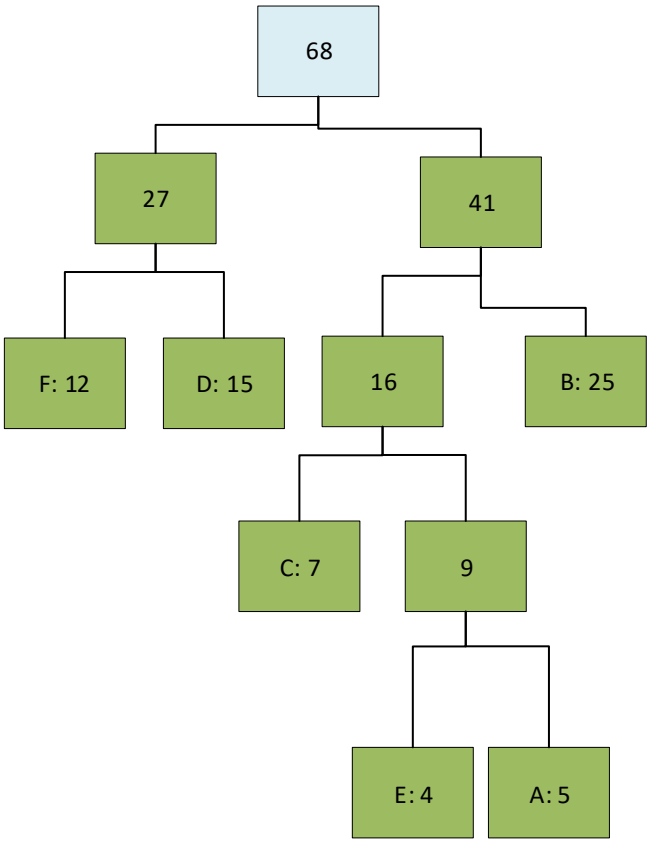
- Iteración 3:



- Iteración 4:

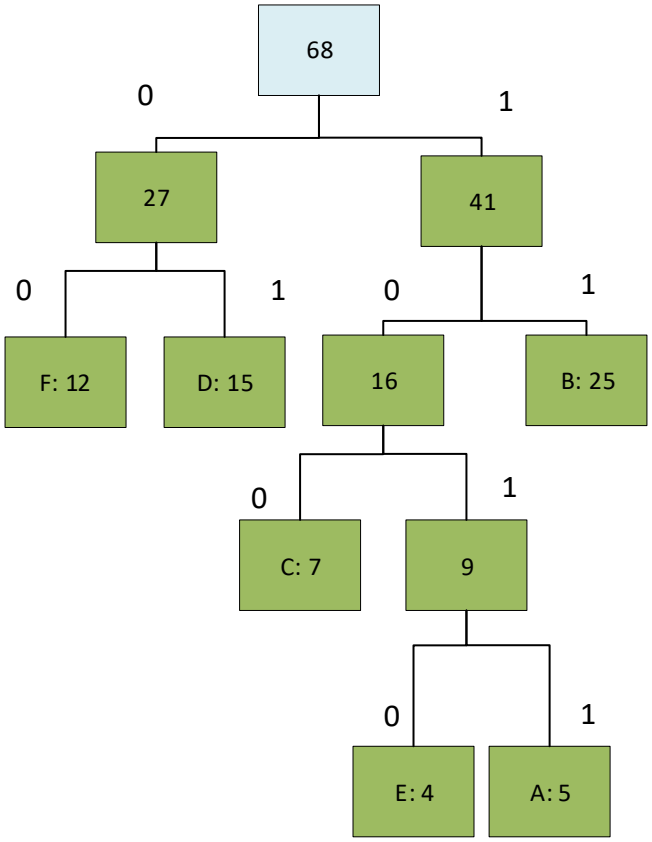


- Iteración 5:



Paso 4:

Ya tenemos el árbol de Huffman. Hay que recorrerlo, asignando el valor 0 a todos los caminos por la izquierda, y el valor 1 a todos los caminos por la derecha:



Paso 5:

Solo queda recorrer el árbol hasta cada uno de los caracteres (o cadenas) concatenando los valores 0 o 1 (según el camino). Se puede observar que los elementos de entrada han quedado situados justamente en las hojas del árbol. Así quedaría:

`a` → `1011`

`b` → `11`

`c` → `100`

`d` → `01`

`e` → `1010`

`f` → `00`

En http://huffman.ooz.ie/ se puede encontrar un generador de árboles Huffman, que puede utilizarse para ver cómo funciona y comprobar si las salidas obtenidas son correctas. Hay mucha más información disponible sobre Huffman en Internet como la mostrada en la Web de la Wikipedia: https://es.wikipedia.org/wiki/Codificaci%C3%B3n_Huffman

Ejercicio pedido

Tarea 1

Se pide realizar un programa **Huffman** en Java que se ejecute con la siguiente sintaxis:

```
Huffman [FICHERO_ENTRADA]
```

Dicho programa recibirá el nombre de un fichero de prueba como parámetro de entrada con la sintaxis que se ha explicado previamente en este documento.

La salida esperada del programa será la codificación Huffman para cada uno de los caracteres de entrada del fichero en función de su frecuencia.

Tarea 2

Posteriormente, se deberá crear un programa **HuffmanTiempos** para que se pueda rellenar una tabla como la siguiente con los tiempos de ejecución para diferentes tamaños del problema:

N	*tiempo*
10000	…..
20000	…..
40000	…..
80000	…..
160000	…..
320000	…..
640000	…..
1280000	…..
…..	…..
Hasta “heap overflow” o casque	

En un documento se deberá incluir la tabla anterior junto con la respuesta a las siguientes preguntas:

- Una breve explicación sobre el algoritmo de **Huffman** utilizado y su complejidad.
- ¿Tienen sentido los tiempos obtenidos si los comparamos con la complejidad del algoritmo? Justifícalo.

Ayuda con la toma de tiempos

Para facilitar la creación de la tabla mediante la generación de problemas aleatorios de gran tamaño se proporciona la clase **Generator**. Dicha clase tiene únicamente un método **run()** que espera los siguientes parámetros:

1. Nombre del fichero de prueba que se genera.
2. Número de palabras diferentes que se generan.
3. Número de caracteres máximo que puede tener cada palabra.
4. Número de caracteres mínimo que puede tener cada palabra.
5. Frecuencia máxima que puede tener cada palabra (las veces que aparece).
6. Frecuencia mínima que puede tener cada palabra.

A continuación, se muestra un ejemplo de utilización en el que se generan 100000 palabras en un fichero llamado **ejemplo.txt**. Las palabras podrán tener un número máximo de 15 caracteres y un mínimo de 3 y la frecuencia estará comprendida entre 1000 y 1 repeticiones:

```
ejemplo.txt 100000 15 3 1000 1
```

Si por ejemplo queremos 10 palabras, todas de 5 caracteres y todas con una frecuencia de 8 repeticiones utilizaríamos los siguientes parámetros:

```
ejemplo.txt 10 5 5 8 8
```

Para implementar **HuffmanTiempos** se deberá utilizar tanto las clases **Huffman** como **Generator**.

*Si utiliza **Eclipse**, se creará el proyecto **prac04_voraz<UOpropio>** con todas las clases necesarias.*

*Si utiliza **JDK**, cree los paquetes necesarios para esta práctica.*

*Las clases necesarias se crearán dentro del paquete **algdnipropio.p4***

Se entregará:

- *Los ficheros fuente de las clases, que se hayan programado, dentro del paquete o del proyecto Eclipse.*
- *Un documento Word con una pequeña explicación de los algoritmos utilizados en ambos programas y su complejidad.*

Se habilitará una tarea en el campus virtual para realizar la entrega. El plazo de entrega será un día antes de la siguiente sesión.