ED - Reto 4

Javier Gálvez Obispo

8 de diciembre de 2017

Dar un procedimiento para guardar un árbol binario en disco de forma que se recupere la estructura jerárquica de forma unívoca usando el mínimo número de centinelas que posible.

Como ejemplo usaremos el siguiente árbol binario generado aleatoriamente en esta página

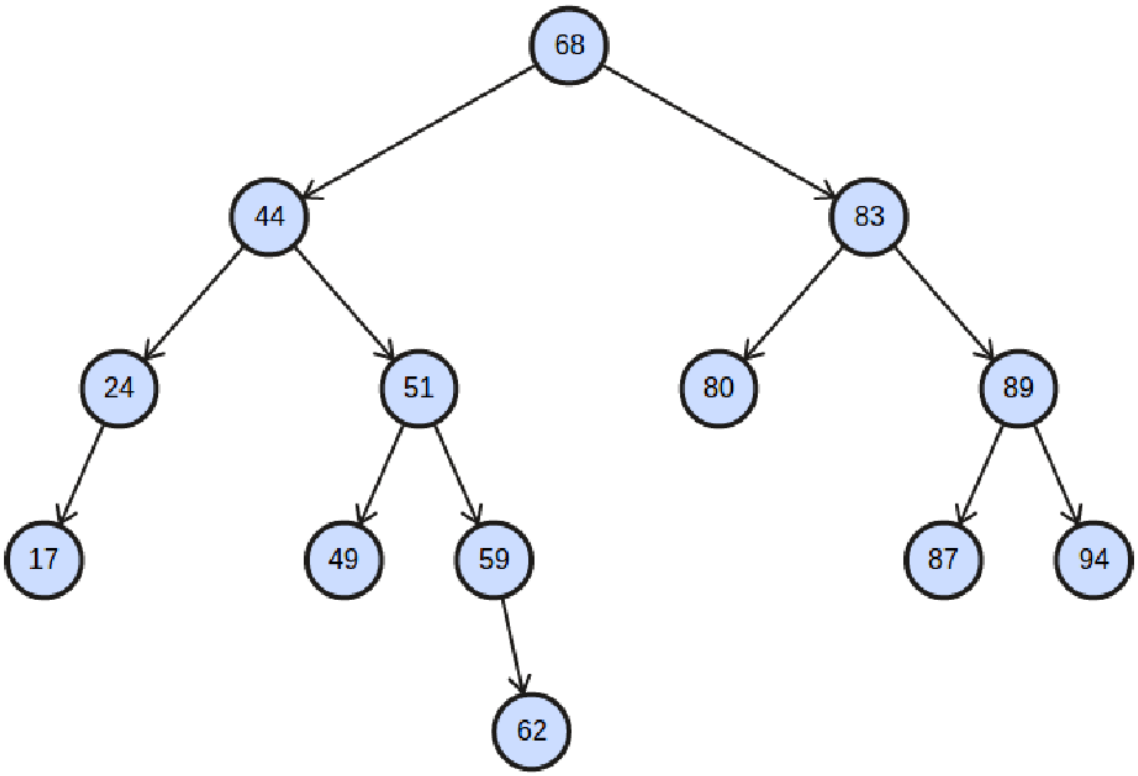

Figura 1: Arbol binario.

Primero definiremos el tipo **Byte** como un **unsigned char**

```
typedef unsigned char Byte;
```

Esto lo hacemos ya que vamos a utilizar bytes como centinelas, es decir, cada bit del centinela significará si el nodo al que corresponde tiene un hijo a su izquierda o derecha dependiendo del caso. Por ejemplo, utilizando el árbol que hemos generado, los bits de la raíz serían 11 mientras que los bits del nodo con valor 24 serían 10 o los del nodo 62 que serían 00. Por cada centinela que utilicemos podremos guardar información de hasta 4 nodos.

Para que no haya ambigüedad guardaremos la información de los nodos por niveles empezando por los que se encuentras más a la izquierda, activando primero los bits más significativos del byte. En nuestro caso el primer centinela almacenaría la información de los nodos 68, 44, 83 y 24. Al nodo 68 le corresponden los 2 bits más significativos y los dos estarían encendidos, al nodo 44 los 2 siguientes y así hasta llegar al último nodo.

Entonces la representación de los centinelas de nuestro árbol binario queda así:

```
Primer centinela:  11111110
Segundo centinela: 11001100
Tercer centinela:  00010000
Cuarto centinela:  00000000
```

Pero en esta representación estamos utilizando 4 bytes (16 nodos) y tenemos sólo 13 nodos. Estamos desperdiciando 6 bits de un centinela, realmente no es tan grande la pérdida que tenemos.

La ventaja de haber definido nuestro tipo **Byte** como **unsigned char** la encontramos al guardar los centinelas en un archivo. Aunque nosotros los utilizemos como bytes no hay que olvidar que son del tipo **unsigned char** y por tanto podemos almacenarlos en un fichero como tal en vez de usar la representación de antes.

Así que a la hora de almacenar el árbol en un fichero primero tendríamos una línea con todos los centinelas y en la segunda línea tendríamos las etiquetas de los nodos.
Para leer el fichero primero guardaríamos en un vector los centinelas leyéndolos uno a uno hasta llegar al final, y una vez leídos todos, guardaríamos las etiquetas en otro vector.
Reconstruir el árbol sería sencillo ya que tenemos tanto los centinelas como las etiquetas almacenados por niveles y sabemos la estructura que tiene el árbol.

Con esto conseguimos reducir el número de datos almacenados a:

$$n + f(n/4)$$

siendo n el número de etiquetas y f la función parte entera. Si n es divisible entre 4, en caso contrario tendríamos que sumar 1 al resultado.
En nuestro caso concreto tenemos:

$$datos = 13 + f(13/4) + 1 = 14 + 3 = 17$$

y efectivamente tenemos 13 etiquetas y 4 centinelas dando un total de 17 datos mientras que de la forma usual de almacenar árboles binarios tendríamos 27.