Estructures de dades
Curs 2020-21
Práctica 2 - Códigos de Huffman - Sprint 3

17 d'abril de 2021

La semana anterior creamos tantos árboles de un solo nodo como caracteres aparecían en el texto de entrada y los insertamos en un Heap. Por lo tanto, esta semana nos encontramos en disposición de construir **el árbol de Huffman**.

Los pasos a seguir para crear el árbol de Huffman son:

1. Mientras queden dos o más elementos en el heap:
   (a) Extraer los dos árboles con menos frecuencia (es decir, con más prioridad).
   (b) Crear el árbol unión: el valor de la nueva raíz será la suma de las frecuencias de las raíces de los árboles extraídos y el árbol que tenga la raíz con menos frecuencia corresponderá al hijo izquierdo.
   (c) Insertar el nuevo árbol en el heap.

Por ejemplo, si el contenido de la tabla de frecuencias es el siguiente:

```
a:  5
b:  9
c:  12
d:  13
e:  16
f:  45
```

Para generar la tabla de frecuencias del ejemplo anterior podéis utilizar el archivo de ejemplo *entrada.txt* adjunto en AulaDigital conjuntamente con este enunciado.

El árbol de Huffman asociado sería:

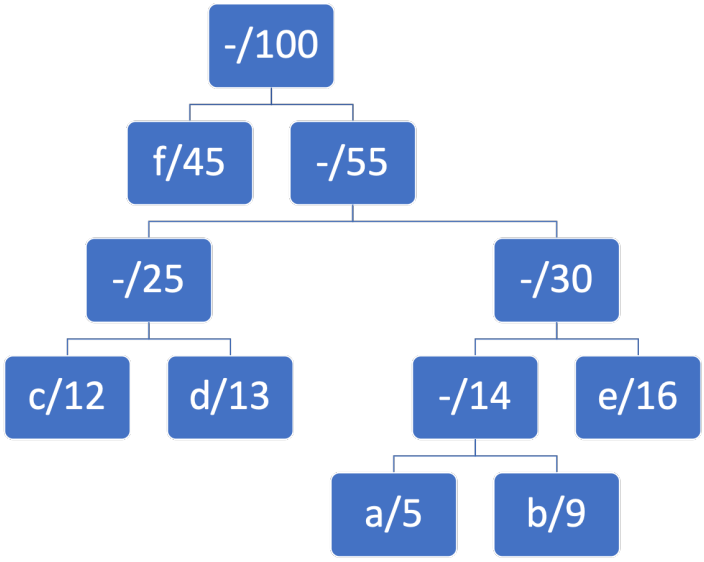

Figura 1: Ejemplo de árbol de Huffman.

1 Tareas a realizar

Las tareas a realizar durante esta semana son:

1. Construcción del **árbol de Huffman** a partir de la tabla de frecuencias.
2. Comprobar que el árbol de Huffman se ha construido correctamente mediante un **recorrido en amplitud**:
   (a) Implementar el recorrido en amplitud del árbol binario utilizando el **TAD cola**.
   (b) La acción *visitar nodo* consiste en mostrar la información del nodo (carácter, frecuencia) por consola.

2 ¿Cómo sabemos si quedan *dos o más elementos en el heap*?

Nuestro TAD cola de prioridad no nos permite acceder a la información del número de elementos que contiene, sólo podemos consultar si se encuentra vacía o no *is_empty()*.

Por lo tanto, nuestro bucle principal para construir el árbol de Huffman, debe basarse en si la cola de prioridad se encuentra vacía y, para comprobar si tiene *dos o más elementos*, volver comprobar si la cola se encuentra vacía una vez se ha extraído el elemento con menos frecuencia:

```
while not is_empty(heap) loop
  -- Extraer el elemento con menos frecuencia
  t1 := get_least(heap);
  delete_least(q);

  if not is_empty(heap) then
    -- Contenia dos elementos (o mas)
    ...
  end if;
end loop;
```

De forma que, una vez se ha salido del bucle principal, la variable ***t1*** contendrá el árbol de Huffman.

3 ¿Qué es un recorrido en amplitud?

El recorrido en amplitud de un árbol consiste en recorrer los nodos por niveles, por lo que primero se visita la raíz (nivel 1), después todos sus hijos (nivel 2), después todos los nodos del nivel 3, y así sucesivamente.

Si consideramos el nodo de partida A (raíz del árbol), primero se visitaría el nodo A, después todos los hijos de A, después se visitarían todos los hijos de los hijos de A, y así sucesivamente.

Por ejemplo, si se realizara el recorrido en amplitud del árbol de la Figura 1, el resultado sería:

```
-:  100
f:  45
-:  55
-:  25
-:  30
c:  12
d:  13
-:  14
e:  16
a:  5
b:  9
```

4 ¿Cómo implementamos un recorrido en amplitud utilizando una cola?

Para implementar el recorrido en amplitud, utilizaremos una cola que contendrá *los nodos pendientes de procesar*. La acción de *procesar un nodo* consistirá en *visitar el nodo* y añadir todos los hijos de estos nodo en la lista de nodos para procesar.

Los pasos a seguir para realizar el recorrido en amplitud de un árbol son:

1. Si el árbol no está vacío:
   (a) Añadir la raíz del árbol a la cola de *nodos pendientes de procesar*.
   (b) Mientras la cola de *nodos pendientes de procesar* no esté vacía:
      i. Coger el primer nodo de la cola.
      ii. Visitar el nodo.
      iii. Añadir a la cola de *nodos pendientes de procesar* su hijo izquierda y su hijo derecho (si tiene hijos).
      iv. Eliminar el nodo de la cola de *nodos pendientes de procesar*.

Recordad que **todos los nodos de un árbol son árboles en sí mismos**, por lo tanto, desde nuestro programa principal realmente vamos a trabajar con una cola de árboles.

Recordad también que no podemos crear una cola de árboles directamente (ya que el tipo árbol, al ser un **TAD**, es un tipo **limited**), sino que debe crearse una **cola de punteros de árboles**.

Cómo estamos trabajando con punteros de árboles (direcciones de memoria), recordad que tenéis que reservar memoria para cada uno de los árboles que deseemos almacenar en la cola.