ESTRUCTURA DE DATOS

Práctica Laboratorio 4

Grado en Ingeniería Informática

Jordi Lazo Florensa

Alejandro Clavera Poza

Universitat de Lleida
Escola Politècnica Superior

Implementación Heaps y Colas con prioridad

El propósito de esta práctica es implementar una cola de prioridad. Esta cola tendrá implementadas las siguientes operaciones: add, remove, element, size.

Para realizar esta cola utilizaremos un árbol binario el cual será representado por un ArrayList que almacenará un trío de valor, la prioridad y el tiempo de llegada, estos dos últimos nos permitirán obtener la prioridad del elemento que tiene en dicha cola.

En cuanto a la operación **Add** se realiza mediante la inserción de un nuevo elemento al final del árbol , tras ello vamos subiendo ese elemento por el árbol hasta encontrarse con un elemento que tiene mayor prioridad que él o bien que este en la posición raíz. Para esto utilizaremos una función recursiva la cual compara el elemento de la posición que le hemos indicado con el elemento padre de dicho índice, si este es mayor que el padre se realizará dicho intercambio. Tras este intercambio se volverá a llamar a la misma función utilizando esta vez el índice del padre ya que el elemento que hemos añadido ahora se encuentra en la posición de su padre. Esta acción se repetirá hasta que llegue a alguna de las condiciones que hemos nombrado anteriormente.

En cuanto a la operación **Remove,** inicia comprobando si hay elementos en la cola, en caso contrario lanzará una excepción. Por otro lado, si encuentra algún elemento en la cola eliminaremos y devolveremos el elemento que se encuentra en la raíz. Una vez realizado la eliminación de ese elemento nos encontraremos que el árbol estará inconsistente ya que hemos eliminado el elemento raíz, para solucionar esto colocaremos como raíz el elemento con menos prioridad de la cola ya que este se encuentra al final del árbol, tras ello mediante el intercambio de posiciones con los hijos de mayor prioridad, iremos bajando dicho elemento hasta llegar a la posición que le corresponde. Por otro lado esta la operación **Element** que al igual que la operación **remove** primero comprobaremos si hay elementos en la cola, si esto no ocurre devolveremos una excepción. Por el contrario, devolveremos el elemento raíz, el cual es el elemento con mayor prioridad.

Por ultimo la operación **Size** la cual devuelve el tamaño de la cola. Para esto nos valemos de la operación size de ArrayList.

Para realizar las operaciones nombradas anteriormente hemos utilizado una serie de métodos auxiliares los cuales nos permiten, dado un índice, saber cual es la posición de su padre, de su hijo derecho y izquierdo. Para esto hemos tenido en cuenta que la posición del hijo izquierdo esta a dos veces la posición del padre más uno y la del hijo derecho esta a una posición más que hijo izquierdo. Por otro lado, hemos podido observar que la posición del hijo derecho siempre se encuentra en una posición par y el izquierdo en una posición impar.

Para finalizar cabe destacar los principales problemas que nos hemos encontrado a la hora de realizar la solución. El primero de estos ocurre cuando se realizaba la operación **add** con prioridades iguales, que en vez de dar más prioridad al que llevaba mas tiempo daba más prioridad al que llevaba menos, esto ocurría porque en el momento que realizabamos la comparación del atributo timeStamp del trio daba mas prioridad al que tenia mayor valor y realmente el que tiene mas prioridad es el que tiene menor valor. Por otro lado en la operación **remove** había casos que cuando realizábamos el arreglo del árbol tras la eliminación del árbol raíz nos lanzaba una excepción, esto ocurría porque no tuvimos en cuenta el caso de que sólo quedará un elemento cuando se realizaba la eliminación de la raíz.