Algorítmica II
Ingeniería Informática en Sistemas de Información
EPD-9: HEAPS

Introducción

En esta EPD se explorarán los conceptos fundamentales de los montículos (o del inglés *heaps*). En concreto se profundizará en el uso de una implementación basada en listas.

Montículos

Un montículo (heap en inglés) es una estructura de datos del tipo árbol con información perteneciente a un conjunto ordenado. Los montículos máximos tienen la característica de que cada nodo padre tiene un valor mayor que el de todos sus nodos hijos, mientras que en los montículos mínimos, el valor del nodo padre es siempre menor al de sus nodos hijos.

Un árbol cumple la condición de montículo si satisface dicha condición y además es un árbol binario completo. Un árbol binario es completo cuando todos los niveles están llenos, con la excepción del último, que se llena desde la izquierda hacia la derecha.

En un montículo de prioridad, el mayor elemento (o el menor, dependiendo de la relación de orden escogida) está siempre en el nodo raíz. Por esta razón, los montículos son útiles para implementar colas de prioridad (ver problema 2). Una ventaja que poseen los montículos es que, por ser árboles completos, se pueden implementar usando arreglos, lo cual simplifica su codificación y libera al programador del uso de punteros (ver problema 1).

Por tanto, un montículo es un árbol binario que tiene, además, dos propiedades añadidas:

1. Es un árbol completo.
2. Cada nodo puede ser igual, mayor o menor que sus hijos.
3. El nodo raíz siempre debe ser el menor número de todos. En caso contrario, se realizan las operaciones necesarias (explicadas en clase de teoría) para solucionar dicho problema.

Este tipo de TDA hereda todas las operaciones de los árboles binarios (ver Figura 1) pero además añade tres nuevas, cuya definición se puede encontrar en la Tabla 1. Por ello, se recomienda al alumno refrescar el contenido de las EPD anteriores en caso de que no esté totalmente familiarizado con el uso de árboles binarios.

Tabla 1. Operaciones particulares de los heaps.

Operación	Descripción
`addElement`	Añade un elemento dado al heap
`removeMin`	Elimina el elemento menor del heap
`findMin`	Encuentra el valor menor del heap

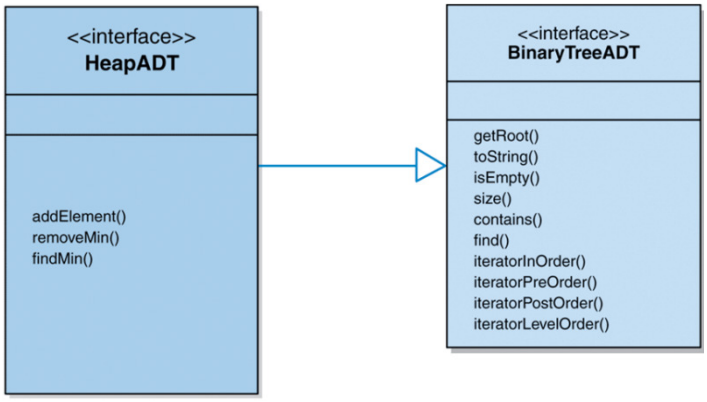

Figura 1. TAD Heap.

En esta EPD se proporciona, pues la interfaz de los heaps, HeapADT.java, junto con una implementación basada en listas enlazadas, LinkedHeap.java. Además, se da una clase, HeapNode.java encargada de gestionar cada uno de los nodos que forman los heaps.

Preparación de la práctica

Todo el material necesario para resolver los experimentos o problemas se encuentra disponible en WebCT. En la carpeta de esta EPD podrá encontrar:

- Enunciado de esta práctica en PDF.
- Fichero **heap.jar**
- Fichero con los códigos fuentes de árboles binarios: **heap.zip**

Cuando cree un proyecto para resolver un determinado experimento o problema, deberá importar la librería heap.jar si necesita trabajar con heaps.

No olvide añadir donde corresponda la cláusula de importación para hacer uso de los TAD y sus implementaciones:

```
import heap.*;
```

Experimentos

Exp1. (15 minutos) Realice un programa principal que construya un heap de 10 elementos que no estén ordenados (puede pedirlos por teclado o inicializarlos manualmente, como prefiera). Una vez construido, convierta ese heap en otro que tenga la siguiente propiedad: el nodo raíz debe ser el elemento menor, el siguiente más pequeño será el hijo izquierdo de la raíz, el siguiente será el hijo derecho del raíz… y así sucesivamente.

Ejercicios

EJ1. (40 minutos) Implemente una clase llamada HeapSort que se encargue de ordenar un array. Esta clase estará compuesta por un método que recibirá un array desordenado, el valor del elemento menor y el valor del elemento mayor. La ordenación se hará sobre el propio array, por lo que dicho método devolverá *void.* Cree una clase principal para comprobar el correcto funcionamiento de la ordenación.

EJ2. (40 minutos) Realice un programa que, dado un heap (que usted mismo creará mediante el método *addElement*), recorra todos sus elementos e imprima por pantalla los hijos izquierdo y/o derecho que tienen todos los elementos que componen dicho heap.

Problemas (240 minutos).

PROB1 (180 minutos). En esta práctica se ha utilizado una implementación de *heaps* basada en listas enlazadas. Cree una clase que implemente el interfaz *HeapsADT* haciendo uso de *arrays.* Use para ello, si lo estima conveniente, las clases referentes a *arrays* utilizadas en prácticas anteriores. Compare ambas implementaciones (basada en listas y en *arrays*) y averigüe cuál es el coste de cada una de ellas.

PROB2 (60 minutos). Realice un programa capaz de gestionar colas de prioridades (priority queue) basadas en heaps. Para ello, se recomienda usar los ficheros priorityqueu.java y priorityqueunode.java de los autores Lewis y Chase (los mismos utilizados durante toda la asignatura) y que están disponibles en internet. Genere, finalmente, un .jar exportable para su uso en futuras aplicaciones.