TAD Cola de Prioridad

Práctica 15
Heap

Objetivo

El objetivo de esta práctica es que el alumno refuerce sus conocimientos acerca de colas de prioridad, desarrollando algunos métodos para la implementación del TAD colas de prioridad que hace uso de un heap.

Descripción general

La práctica consiste en incluir dentro de la implementación del TAD colas de prioridad los métodos para imprimir el contenido del heap, cambiar el valor del elemento de una posición específica en el heap, borrar el elemento de una posición específica del heap y crear un iterador en orden ascendente sobre los elementos de la cola de prioridad.

Material

El material de esta práctica consta de los siguientes archivos:

- **InterfazColaConPrioridad.class** interfaz para trabajar con colas de prioridad.
- **Heap.java** clase que implementa al TAD colas de prioridad utilizando heaps. En esta clase se incluirán los métodos solicitados.
- **PruebaHeap.class** programa para probar la clase `Heap`.
- **PruebaHeap$ComparaEnteros.class** programa para probar la clase `Heap`.
- Documentación:
    - InterfazColaConPrioridad.html documentación de la clase `InterfazColaConPrioridad`.

Desarrollo

1. Descargar los archivos `InterfazColaConPrioridad.class`, `Heap.java`, `PruebaHeap.class` y `PruebaHeap$ComparaEnteros.class` en el directorio donde se va a desarrollar la práctica.

2. Leer la documentación y revisar los archivos del material de la práctica. Recordar que la firma de los métodos proporcionados y requeridos en el desarrollo de la práctica no podrán ser modificados.

3. Dentro de la clase `Heap` se implementarán los siguientes métodos:

    a. **imprimirHeap**() método para imprimir el contenido del heap como un árbol.
    b. **cambiar**(**int** p**, Object** nuevoValor) método para cambiar el elemento en la posición p con el nuevo valor. Si la posición p es inválida, no hace nada. El método debe garantizar la propiedad de continuar siendo un heap.
    c. **borrar**(**int** p) método para eliminar del heap el elemento en la posición p. Si la posición p es inválida, no hace nada. El método debe asegurarse de preservar la propiedad de ser un heap.
    d. **iterador**() método que devuelve un iterador para recorrer los elementos de la cola de prioridad en orden ascendente. El iterador no debe soportar la operación *remove*.

4. Probar la implementación realizada ejecutando el programa `PruebaHeap`. Si el programa funciona adecuadamente se verán los siguientes mensajes:

```
-Imprimir contenido del heap:
5 15 14 20 16 24 21 34 26 17 35
Tamanio: 11

-Cambiar elemento de la posicion 0 con el nuevo valor 18
Imprimir contenido del heap:
14 15 18 20 16 24 21 34 26 17 35

-Cambiar elemento de la posicion 5 con el nuevo valor 7
Imprimir contenido del heap:
7 14 18 20 15 24 21 34 26 17 35

-Cambiar ultimo elemento con el nuevo valor 12
Imprimir contenido del heap:
7 12 18 20 14 24 21 34 26 17 15

-Tratar de cambiar con posicion inexistente 15
Imprimir contenido del heap:
7 12 18 20 14 24 21 34 26 17 15
```

-Borrar elemento de la posicion 0
Tamanio: 10
Imprimir contenido del heap:
12 14 18 20 15 24 21 34 26 17

-Borrar elemento de la posicion 5
Tamanio: 9
Imprimir contenido del heap:
12 14 17 20 15 18 21 34 26

-Borrar elemento de la ultima posicion
Tamanio: 8
Imprimir contenido del heap:
12 14 17 20 15 18 21 34

-Tratar de borrar posicion inexistente 30
Tamanio: 8
Imprimir contenido del heap:
12 14 17 20 15 18 21 34

-Iterador en orden ascendente:
12
14
15
17
18
20
21
34