Estructura de Datos 20/21

Práctica 5

Heap (montículo): una cola de prioridad

Objetivos

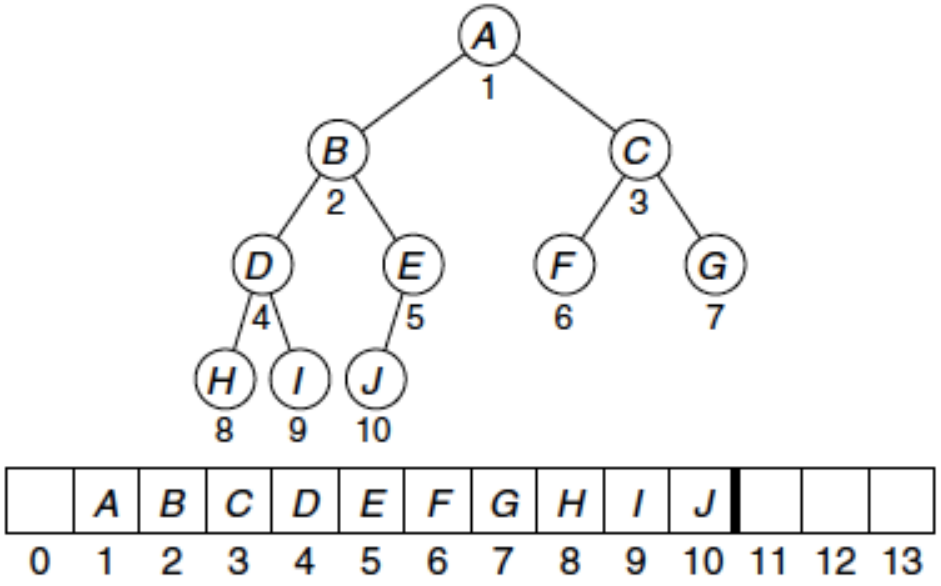
- Implementar la operación de insertar en un heap
- Implementar la operación de eliminar en un heap

Introducción

Una cola de prioridad es una estructura de datos fundamental que nos permite acceder muy rápidamente al mínimo elemento (aquel que tiene mayor prioridad).

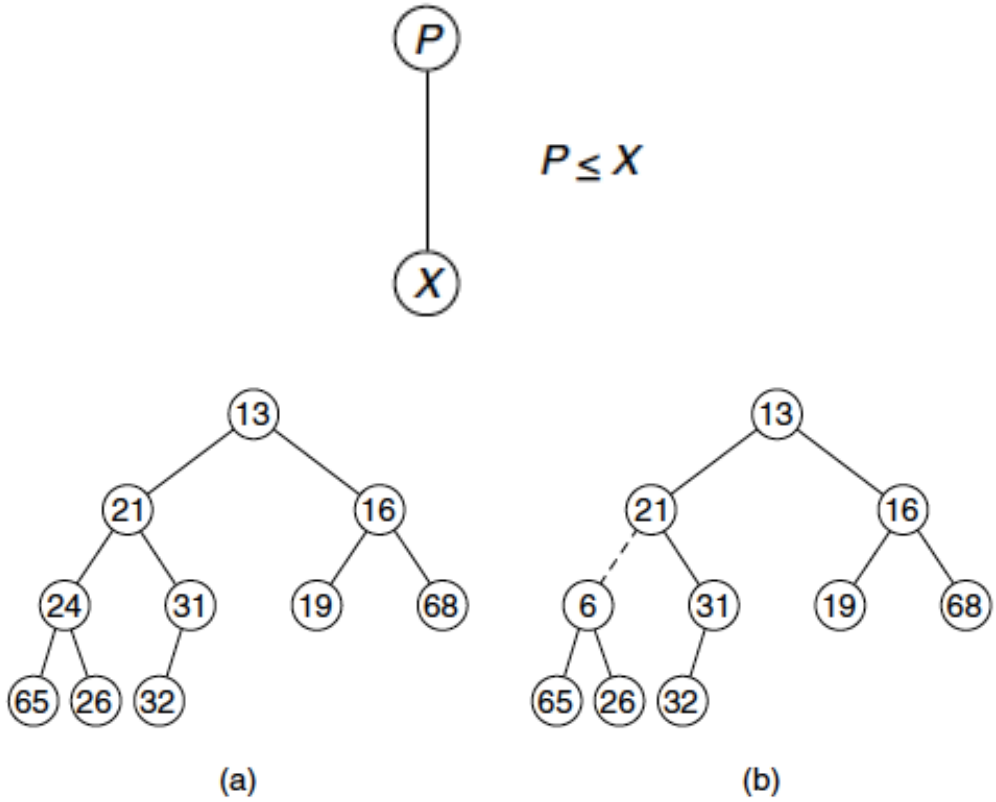
Un heap se puede implementar de diferentes maneras. Se podría utilizar una lista enlazada. Insertar un elemento al principio de la lista es muy eficiente, de tiempo constante. Sin embargo, encontrar el mínimo, requiere búsqueda lineal. También se podría implementar utilizando un BST. Sin embargo, la entrada en un BST no es típicamente aleatoria.

En esta práctica, implementaremos un HEAP como si fuera un árbol binario completo y equilibrado, y lo haremos en forma de vector. Un árbol binario nos da tiempo logarítmico. Un vector es una estructura muy sencilla. Si a esto le sumamos que representamos un árbol binario completo, entonces no necesitamos punteros izquierda / derecha. Dado un nodo en posición i, su hijo izquierdo está en 2i, y su hijo derecho, en la posición 2i + 1. El padre del nodo en posición i está en la posición (división entera) i / 2.



La propiedad que permite a las operaciones del montículo realizarse rápidamente es la **propiedad del orden del montículo**. En un montículo, el mínimo elemento (con mayor prioridad), siempre está en la raíz. Si aplicamos la recursividad, en un

montículo, cualquier nodo tiene que ser más pequeño que cualquiera de sus descendientes.

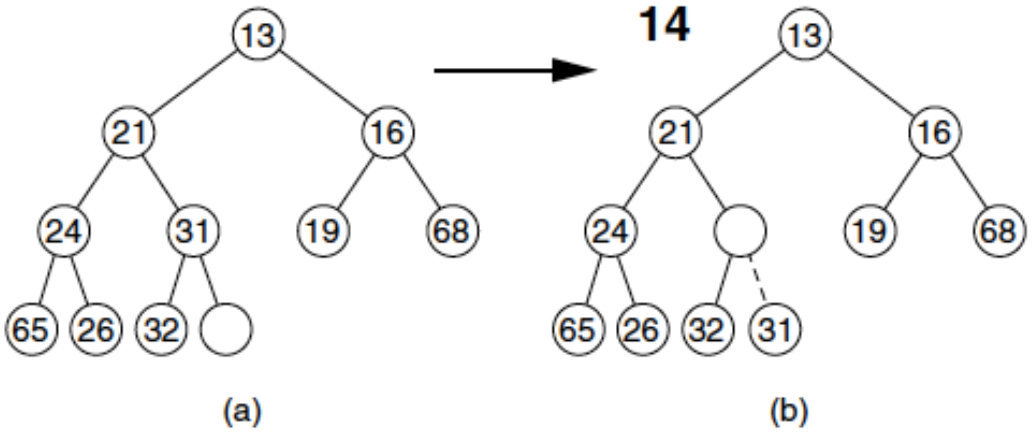


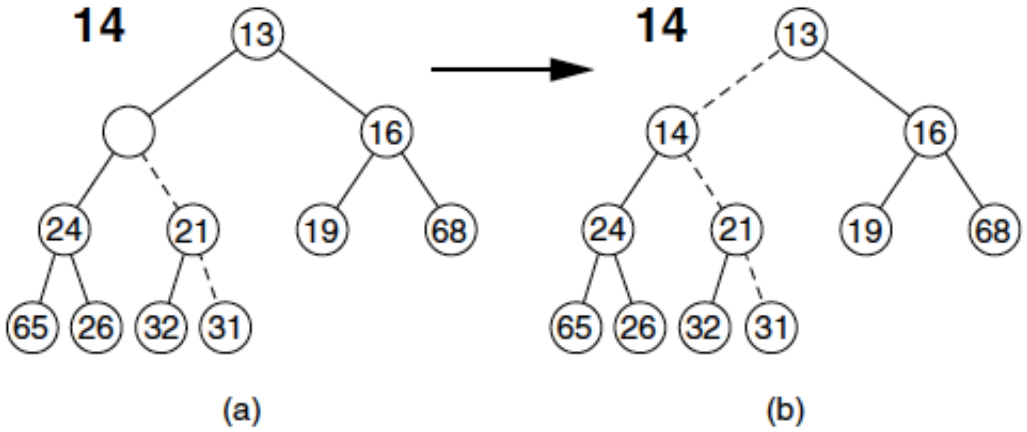
a) es un HEAP, b) no es un HEAP

Operaciones

Inserción. Para insertar un elemento X en el montículo, primero debemos añadir un nodo en el árbol. Esto lo haremos creando una posición en el vector, que será la siguiente posición a ocupar. Si no hacemos esto, el árbol no será completo. Si el elemento se puede insertar satisfaciendo la propiedad de orden, el padre es más pequeño que cualquiera de sus hijos, hemos acabado. De lo contrario, movemos el padre del nuevo nodo a dicho nodo, y repetimos el proceso – vamos subiendo hacia arriba - hasta que colocamos el nuevo nodo. El hecho de subir nos obliga a situar el elemento más pequeño lo más arriba posible.

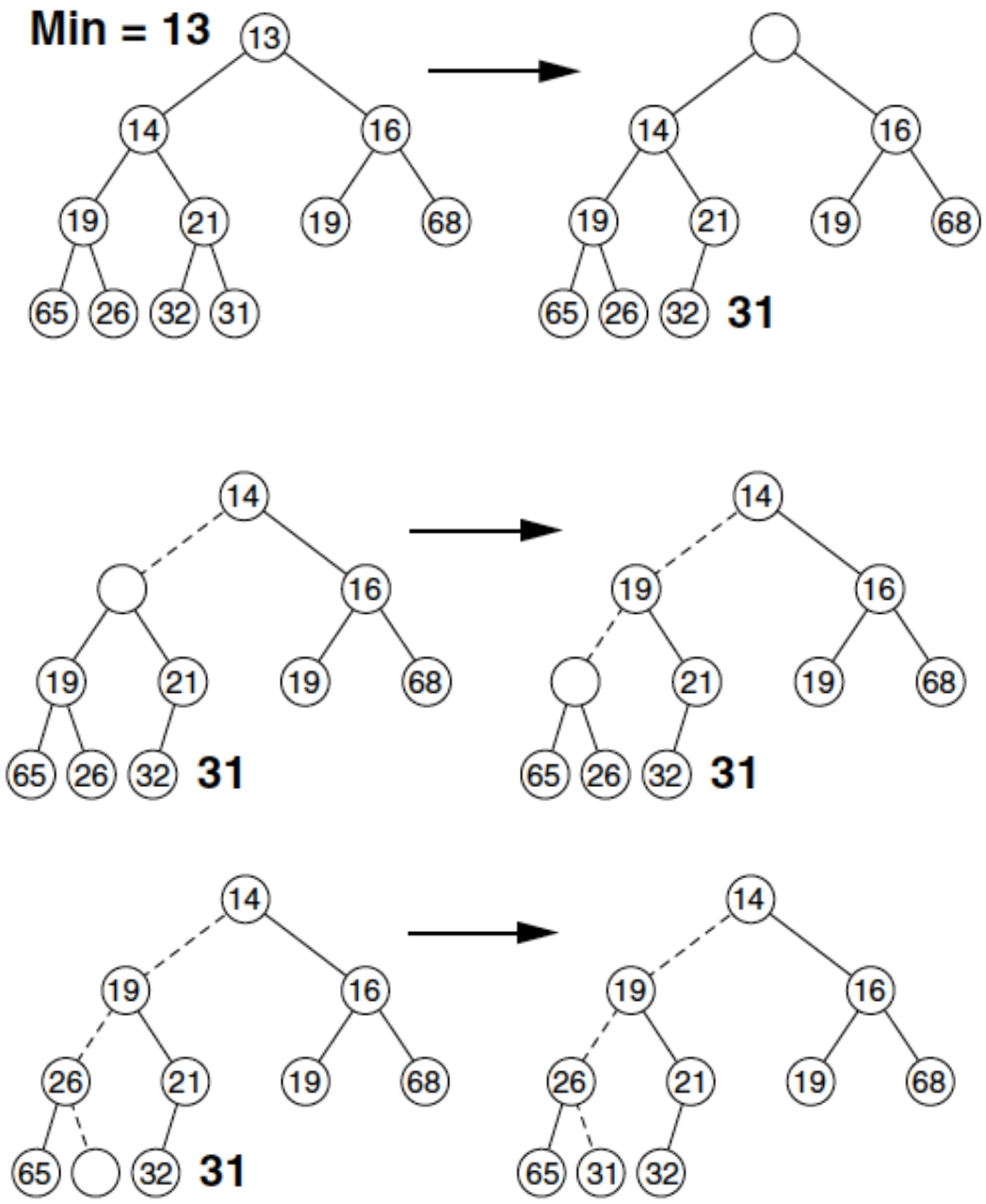
En el siguiente montículo, queremos insertar el 14. Creamos un nuevo nodo, que tiene ser hijo derecho de 31 para que el árbol sea completo. El nodo 14 no lo podemos insertar en el nuevo nodo, porque su padre, 31, no es más pequeño que 14. Por tanto, bajamos 31 al nuevo nodo, y seguimos. El nodo 14 no lo podemos insertar como hijo derecho de 21, porque su padre, 21, sería más grande que uno de sus descendientes. Por tanto, bajamos 21. Ahora si podemos insertar 14, porque 13, su padre, es más pequeño.





Eliminación. Eliminaremos siempre el mínimo, que está en la raíz del árbol. Se implementa de forma similar a la operación de inserción. Cuando se elimina el mínimo, se crea un agujero en el montículo. Tenemos un elemento menos en el árbol, y esto significa que el último elemento se tiene que eliminar – es decir, poner en algún otro sitio. Para tapar el agujero, y reducir en 1 el número de nodos en el árbol, si pudiéramos poner el último elemento arriba del todo, ya estaríamos. Sin embargo, esto es imposible, porque se espera que los nodos en niveles inferiores sean más grandes que los superiores, pero lo tenemos que poner en algún sitio! Entonces lo que hacemos es poner algún elemento en el agujero, y nos movemos hacemos abajo. Concretamente, lo que hacemos es buscar el hijo más pequeño del nodo agujero. Si dicho nodo es más pequeño que el nodo que queremos poner en el agujero (el último), entonces movemos dicho hijo al nodo agujero. Esto crea un nuevo agujero, en un nivel inferior, y repetimos el proceso, hasta que el último nodo del árbol se encuentre en una posición correcta.

En el siguiente montículo, eliminamos el 13. Esto crea un agujero. Fijamos el último nodo (31) para ver dónde ubicarlo. Buscamos el hijo menor de 13. Es 14. Como 14 < 31, lo colocamos en el agujero, y creamos el agujero en el nivel inferior. Buscamos hijo menor. Es 19. Como 19 < 31, lo colocamos en el agujero, y creamos un nuevo agujero, en un nivel inferior. Buscamos hijo menor. Es 26. Como 26 < 31, lo subimos, y tenemos el agujero en nodo hoja. Como no tiene hijos, 31 pasa a ser hijo derecho de 26.



Trabajo a realizar

Implementa el método insertar y eliminar de un HEAP definido de la siguiente manera:

```
public class MyPriorityQueue<T> {
    private static final int DEFAULT_CAPACITY = 20;
    private int currentSize = 0;
    private T[] vector;
    //private Comparator<? super T> cmp;

    public MyPriorityQueue()
    {
        currentSize = 0;
```

```
        vector = (T[]) new Object [DEFAULT_CAPACITY+1];
    }
```

Los métodos están declarados de la siguiente manera:

```
public boolean add (T x)
```

Del método eliminar, te pedimos que implementes el método **percolateDown**.

```
public T remove()
{
    T minItem = element();
    vector[1] = vector [currentSize--];
    percolateDown(1);
    return minItem;
}
private void percolateDown(int hole)
```

Entrega

Fecha: ver tarea correspondiente en el campus virtual

Material: Un fichero ZIP con el proyecto IntelliJ IDEA y un documento RTF con la estrategia / algoritmo implementado.

Criterios (generales) de evaluación

- Práctica no entregada o entregada fuera de plazo = 0
- Entrega parcial, <= 3
- Entrega completa, entre 4 y 10. Se valorará el diseño y claridad del código, comentarios, etc.