Ejercicios Tema 12

Ejercicios Adaptados de Apuntes y Exámenes de EDA
Germán Moltó Martínez
gmolto@dsic.upv.es
Estructuras de Datos y Algoritmos
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Traza de Insertar

- Hacer una traza de insertar el valor 3 sobre el Montículo Binario:

  [0,1,4,8,2,5,6,9,15,7,12,13]

Traza de Insertar en Montículo Vacio

- Hacer una traza de insertar a partir de un Montículo Binario vacío los siguientes valores:

  6,4,15,2,10,11,8,1,1,7,9,12

- Asumir que el array subyacente dispone de espacio adicional para albergar todos los elementos.

Traza de eliminarMin

- Realizar una traza de eliminarMin sobre el Montículo Binario compuesto por los valores:

  [0,1,4,8,2,5,6,9,15,7, 12,13]

Obtener Máximo de Montículo Minimal

- Dada una Colección representada como un Montículo Minimal, se pide escribir un método que obtenga su máximo con el mínimo número de comparaciones.
- Para ello, añadir un nuevo método a la clase MonticuloBinario con la siguiente especificación:

```
public E buscarMax();
```

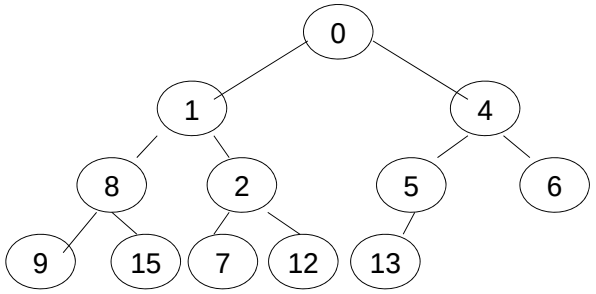

Montículo con Datos de Igual Prioridad

- Una Cola de Prioridad implementada como un Montículo Minimal, ¿Funciona como Cola para datos de igual prioridad?
- Para responder a la cuestión, averiguar el orden en el que saldrían los datos de prioridad 1 del Montículo que resulta de insertar los siguientes pares en el orden mostrado:
  - (1,a), (2,a), (1,b), (3,a), (4,a), (1,c), (6,a), (1,d)
- Nota: La primera componente indica la prioridad del dato que está en la segunda componente.

Búsqueda en un Heap Minimal

- Diseñese un método que compruebe si un dato x está en un Heap Minimal y estúdiese su coste.

```
public boolean buscar(E x);
```



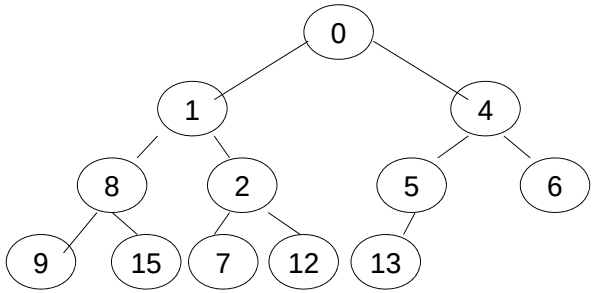
Ventajas e Inconvenientes

- Discutir las ventajas e inconvenientes de las siguientes implementaciones del interfaz ColaPrioridad:
  - Como una LEG Ordenada.
  - Como un array de Colas.
    - Diferentes colas a las que se asigna una determinada prioridad y los objetos almacenados en cada cola reciben la prioridad de la misma.
  - Como un Montículo Binario.

Eliminar de Montículo Minimal

- Diséñese un método que elimine el Dato de la posición k de un Heap Minimal.
- Estúdiese su coste.

public E **borrar**(int k);



Inicialización de Montículo Binario

- Sea h un Montículo Binario Minimal vacío. Realizar la traza del proceso de inicialización de h con los siguientes valores: 32, 26, 65, 68, 19, 31, 21, 13, 13, 6
- El proceso se debe hacer de manera **eficiente**.

Iguales al Mínimo

- Sea la clase *MonticuloBinario*, que implementa la interfaz *ColaDePrioridad* mediante un Montículo Binario Minimal:
  - Se desea añadir un método a *MonticuloBinario* para obtener con el menor coste posible en cada instancia el número de datos del Montículo iguales al mínimo, incluido este.
  - Una posible solución al problema sería hacer un recorrido por niveles, pero su coste no sería el mejor posible en cada instancia ya que no aprovecha la propiedad de ordenación del Montículo Binario.
- Se pide:

1. Diseñar el método **público** *igualesAlMinimo*, que calcula con el menor coste temporal posible el número de datos del Montículo iguales al mínimo
2. Diseñar el método **protected** recursivo *igualesAlMinimo* que se invoca desde el método anterior.
3. Calcula la complejidad temporal asintótica del método.

Comprobación de Propiedad de Orden

- Dado un vector genérico de objetos Comparables, (cuyos datos comienzan en la posición uno), diseña un método que compruebe de la forma más eficiente posible si sus datos cumplen la propiedad de orden de un Montículo Minimal.

Resultado Tras Operaciones

- En un MonticuloBinario<Integer> vacío mB se han insertado, en este orden, los Integer 1, 2, 5, 7, 9 y 14.
- Dibújense el Árbol y Representación Implícita (atributo elArray) correspondientes al estado de mB después de ejecutar cada una de las siguientes instrucciones:

1. mB.insertar(new Integer(0));
2. mB.insertar(new Integer(3));
3. mB.eliminarMin();

Sobre el Método Reflotar (I)

- A continuación se muestra una posible implementación del método reflotar:

```
protected int reflotar(E e, int pos) {
    int res = pos;
    if ( pos != 1 )
       if ( e.compareTo(elArray[pos/2]) < 0 ){
           elArray[pos] = elArray[pos/2]; res = reflotar(e, pos/2);
       }
    return res;
}
```


Sobre el Método Reflotar (II)

1. La talla x del problema que resuelve es:
   a) x = elArray.length
   b) x = elArray.length – pos
   c) x = pos
2. Sobre sus instancias significativas, indíquese cuáles de las siguientes afirmaciones son correctas:
   a) No existen instancias significativas para el coste
   b) Su caso mejor se da cuando pos vale 1 porque no se produce ninguna llamada recursiva
   c) Una instancia del caso peor es $\forall$ i: 1 ≤ i < pos: elArray[i] ≥ e
   d) Una instancia del caso mejor es elArray[pos/2] < e
   e) Una instancia del caso mejor es elArray[pos/2] ≥ e

Sobre el Método Reflotar (III)

- Escribir las Relaciones de Recurrencia que expresan su Complejidad Temporal
- Indicar su coste Temporal Asintótico