Tema 5

Cola de Prioridad y Montículo Binario. Heapsort

Objetivos

1. Presentar una implementación eficiente del modelo Cola de Prioridad
2. Representación contigua (implícita) de los datos mediante un *array*: Montículo Binario (Binary Heap)
3. Diseño del método genérico de ordenación HeapSort sobre la base de esta implementación

Tema 5

Cola de Prioridad y Montículo Binario. Heapsort

Contenidos

1. El modelo Cola de Prioridad: definición, coste estimado y ejemplos de uso
2. La Implementación de Cola de Prioridad: definición, propiedades y operaciones del Montículo Binario
3. Esquema básico de la clase Java `MonticuloBinario`
4. *Heap Sort*

Tema 5

Cola de Prioridad y Montículo Binario. Heapsort

Bibliografía

- Weiss, M.A. ”Estructuras de datos en Java”, Adisson-Wesley, 2000 (apartados del 1 al 5 del capítulo 20)
- Sahni, S. Data Structures, Algorithms, and Applications in Java. McGraw-Hill, 2000. Capítulo 13, secciones 1 a 4.
- Michael T. Goodrich and Roberto Tamassia. Data Structures and Algorithms in Java (4th edition). John Wiley & Sons, Inc., 2010. Capítulo 8, secciones 1 a 3.

Tema 5

Cola de Prioridad y Montículo Binario. Heapsort

Algunos enlaces interesantes…

- Algorithms with Attitude
  - Vídeo: Heaps 1: Introduction (en inglés)
  - Vídeo: Heaps2: Operations (en inglés)
  - Vídeo: Heaps3: BuildHeap (en inglés)
  - Vídeo: Heaps4: HeapSort (en inglés)

- Vídeo: Heaps: Curso MIT (en inglés)

1. Cola de Prioridad

Una Cola de Prioridad es una colección homogénea de datos en la cual las operaciones características son aquellas que permiten acceder al dato de máxima prioridad.

```
package librerias.estructurasDeDatos.modelos;

public interface ColaPrioridad<E extends Comparable <E>> {
    // Añade e a la cola de prioridad
    void insertar(E e);
    // SII !esVacia() devuelve el dato de máxima prioridad
    E recuperarMin();
    // SII !esVacia() devuelve y elimina el dato máx prioridad
    E eliminarMin();
    // Devuelve true si la cola está vacía
    boolean esVacia();
}
```

1. Es indispensable que cualquier dato de la colección, *d1*, sea `Comparable` con otro, *d2*, en base a su prioridad:

   d1* < *d2* SII la prioridad de *d1* es mayor estricta que la de *d2

2. El dato de **mayor prioridad** es el mínimo (o máximo).
3. A igualdad de prioridades, acceso FIFO.

1. Cola de Prioridad - *Ejemplos de uso*

Existen múltiples aplicaciones que usan una Cola de Prioridad:

- Gestión de procesos de un SO
- Simulación dirigida por eventos discretos, como la de las urgencias de un hospital
- Implementación de la lista de espera de una compañía aérea
- Problemas de optimización basados en algoritmos voraces, como la obtención de Caminos Mínimos y Árbol de Recubrimiento Mínimo de un Grafo

2. Montículo (*Heap*) Binario - *Definición*

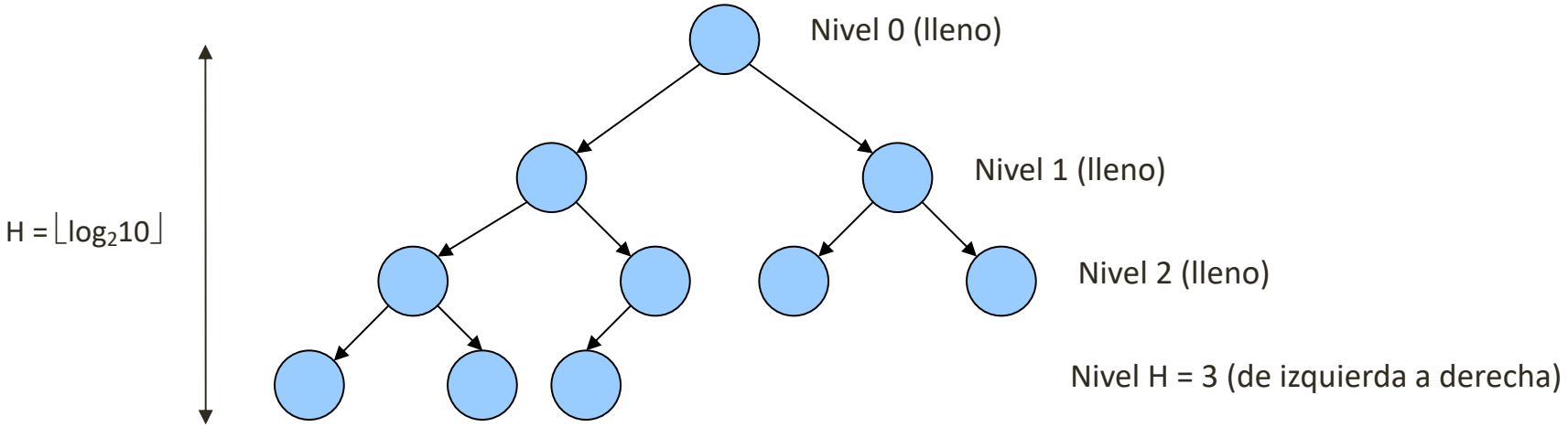
Un Montículo Binario es un Árbol Binario (AB) tal que

1. **PROPIEDAD ESTRUCTURAL:** es un AB **Completo**
   - **Su altura es, a lo sumo,** $\lfloor \log_2 N \rfloor$
   - Se asegura entonces un coste logarítmico en el peor caso si los algoritmos implican la exploración de una rama entera
   - Los árboles binarios completos permiten una representación implícita sobre un array
2. **PROPIEDAD DE ORDEN:** es un AB en cuya Raíz se sitúa, siempre el dato de máxima prioridad
   - En un minHeap, el dato de un nodo es siempre menor o igual que el de sus hijos
   - En un maxHeap, es siempre mayor o igual

2. Montículo (*Heap*) Binario – *Representación*

Un AB Completo es aquel que tiene todos sus niveles llenos a excepción, quizás, del último, que en tal caso debe de tener situados todos sus nodos tan a la izquierda como sea posible

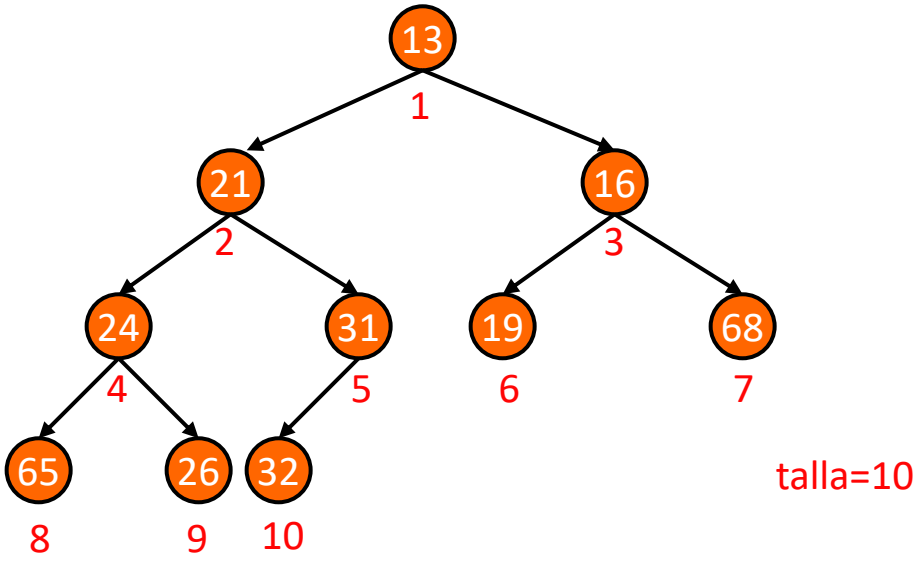
La altura de un Heap de N elementos es $\lfloor \log_2 N \rfloor$



2. Montículo (*Heap*) Binario-*Representación (cont.)*

Un árbol binario completo admite una representación implícita sobre un array

- Se almacena los nodos en un array según su recorrido por niveles
- **La raíz se guarda en la posición 1 del array**; el Hijo Izquierdo de la Raíz en la posición 2; el el Hijo Derecho de la Raíz en la posición 3; etc. (La posición 0 se deja libre, lo que facilita el cálculo de los hijos de un nodo)

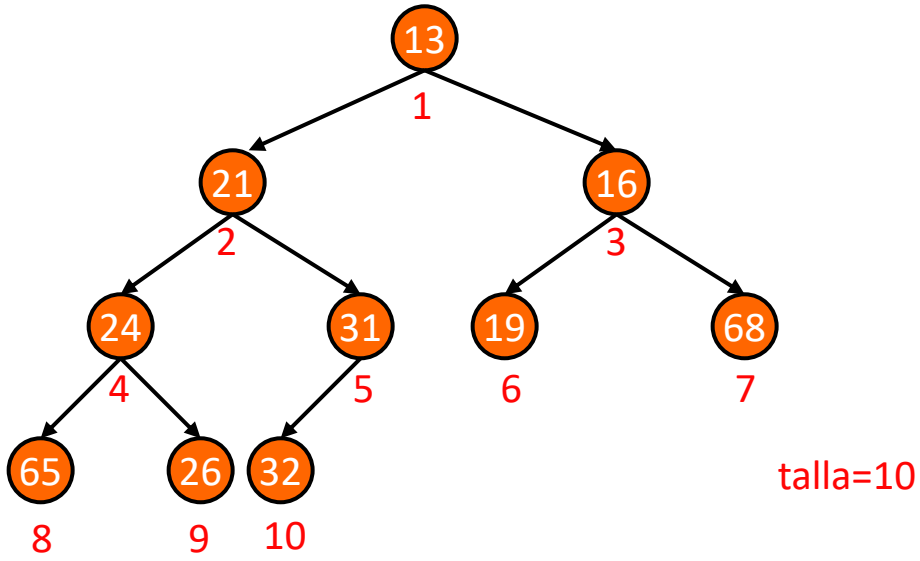


1	2	3	4	5	6	7	8	9	10	
	13	21	16	24	31	19	68	65	26	32	....

* E[] elArray
* int talla

2. Montículo (*Heap*) Binario-*Representación (cont.)*

- elArray[1] representa a su Nodo Raíz
- si elArray[i] representa a su i-ésimo Nodo (Por Niveles)
  - Su Hijo Izquierdo es elArray[2i], si $2i \leq$ talla
  - Su Hijo Derecho es elArray[2i+1], si $2i + 1 \leq$ talla
  - Su Padre es elArray[i/2], excepto para i = 1



1	2	3	4	5	6	7	8	9	10	
	13	21	16	24	31	19	68	65	26	32	....

* E[] elArray
* int talla

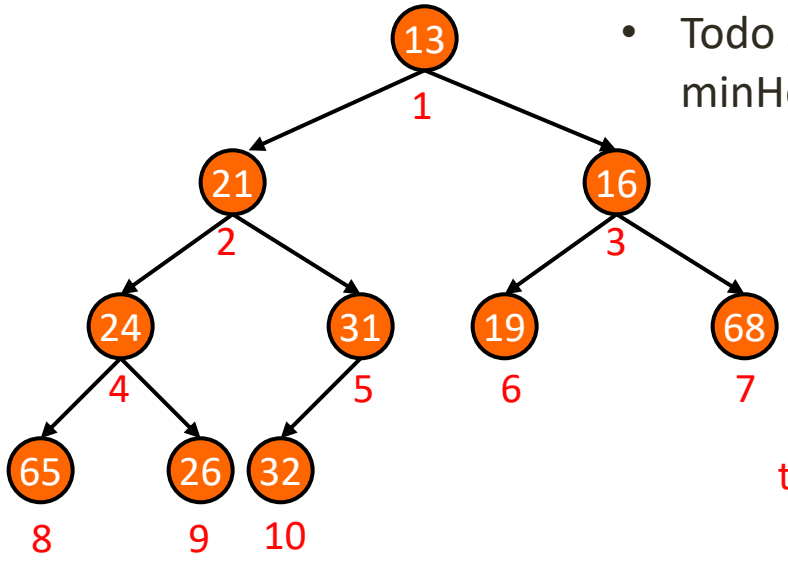
2. Montículo (*Heap*) Binario-*Representación minHeap (cont.)*

- Propiedad de orden en un minHeap:

elArray[padre(i)] <= elArray[i], 2<=i<=talla

Propiedades:

- Todo camino desde la raíz a una hoja es una secuencia ordenada (13, 21, 31, 32; 13, 16, 68)
- La raíz es el nodo de valor mínimo
- Todo subárbol de un minHeap es también un minHeap



talla=10

1	2	3	4	5	6	7	8	9	10	
	13	21	16	24	31	19	68	65	26	32	....

* E[] elArray
* int talla

2. Montículo (*Heap*) Binario-*Ejercicios*

1. Suponiendo que no hay elementos repetidos y que estamos hablando de un minHeap:
    a) ¿Dónde estará el mínimo?
    b) ¿Dónde estará el máximo?
    c) ¿Cualquier elemento de una hoja será mayor que los elementos de los nodos internos?
    d) ¿Un minHeap es un vector ordenado de forma creciente?
    e) ¿Es un minHeap la siguiente secuencia: {1, 5, 12, 7, 17, 14, 13, 28, 6, 18}

2. Suponiendo que no hay elementos repetidos y que estamos hablando de un maxHeap:
    a) ¿Dónde estará el máximo?
    b) ¿Dónde estará el mínimo?
    c) ¿Cualquier elemento de una hoja será menor que los elementos de los nodos internos?
    d) ¿Un maxHeap es un vector ordenado de forma decreciente?
    e) ¿Es un maxHeap la siguiente secuencia: {23, 17, 14, 6, 13, 10, 1, 5, 7, 12}