

Heaps (*Montículos*)

- Conceptos
- Representación en memoria
- Ejemplo
- Creación de un heap: *heapify*
- Inserción de un nuevo elemento: *flotar*
- Ejercicios

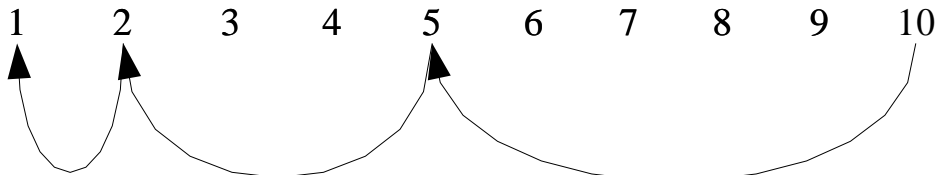
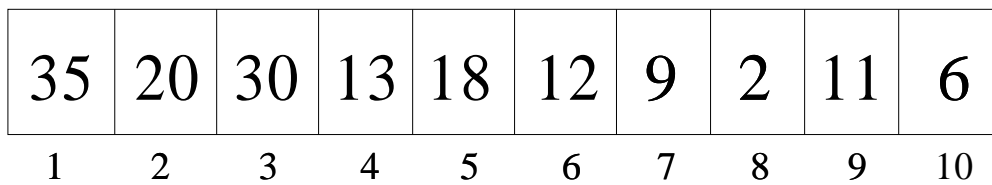
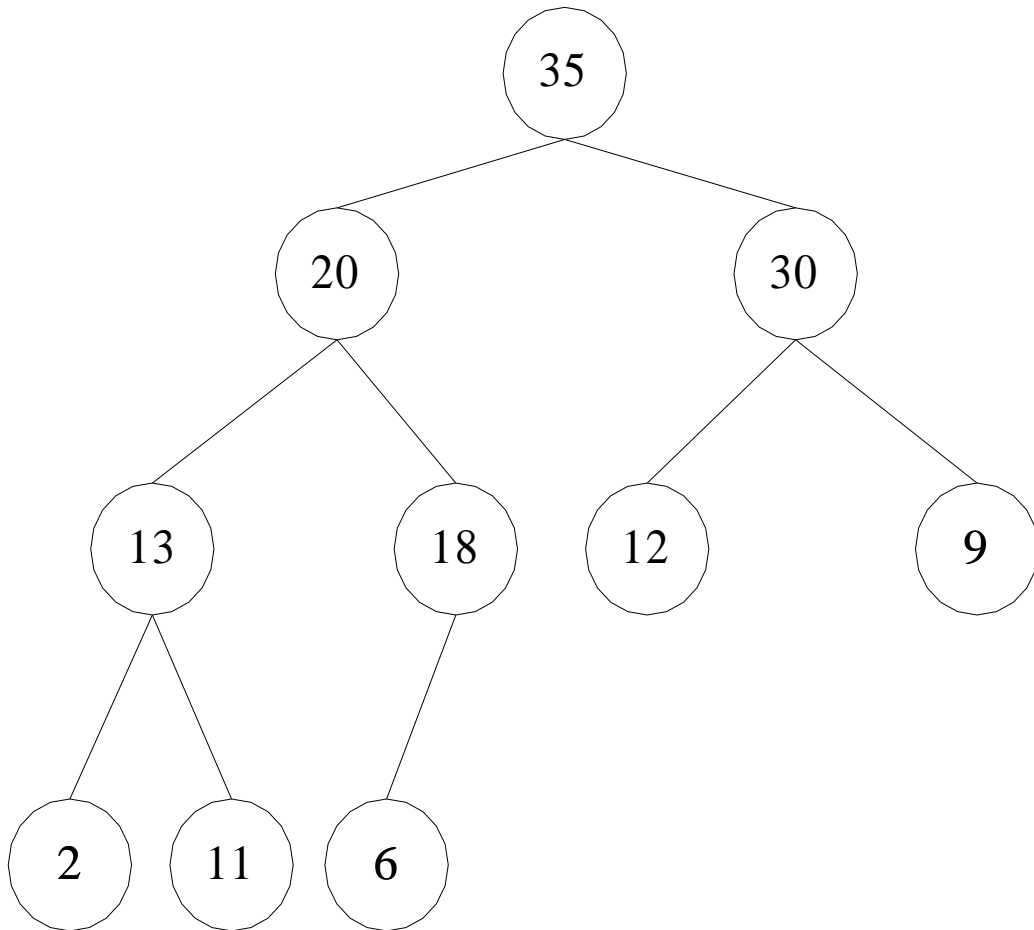
Heaps: conceptos

- Un *heap* o montículo es un árbol binario semicompleto en el que el valor clave almacenado en cualquier nodo es menor (mayor) o igual que los valores clave de sus hijos.
- En un árbol *binario* cada nodo tiene a lo sumo dos hijos.
- Es clave la propiedad de ordenación parcial, es decir, el hecho de tener un árbol parcialmente ordenado: para cualquier camino directo desde una hoja del árbol hasta la raíz, los valores clave deben aparecer en orden no decreciente (no creciente).
- Un heap se va llenando por niveles, primero el nivel 0, luego el nivel 1, luego el nivel 2, etc. Todos los niveles estarán llenos salvo el último. Un cierto nivel i lleno tendrá 2^i nodos.
- *Semicompleto* significa que el último nivel puede no estar lleno, es decir, podemos encontrar hojas en dos niveles distintos.

Heaps: representación en memoria

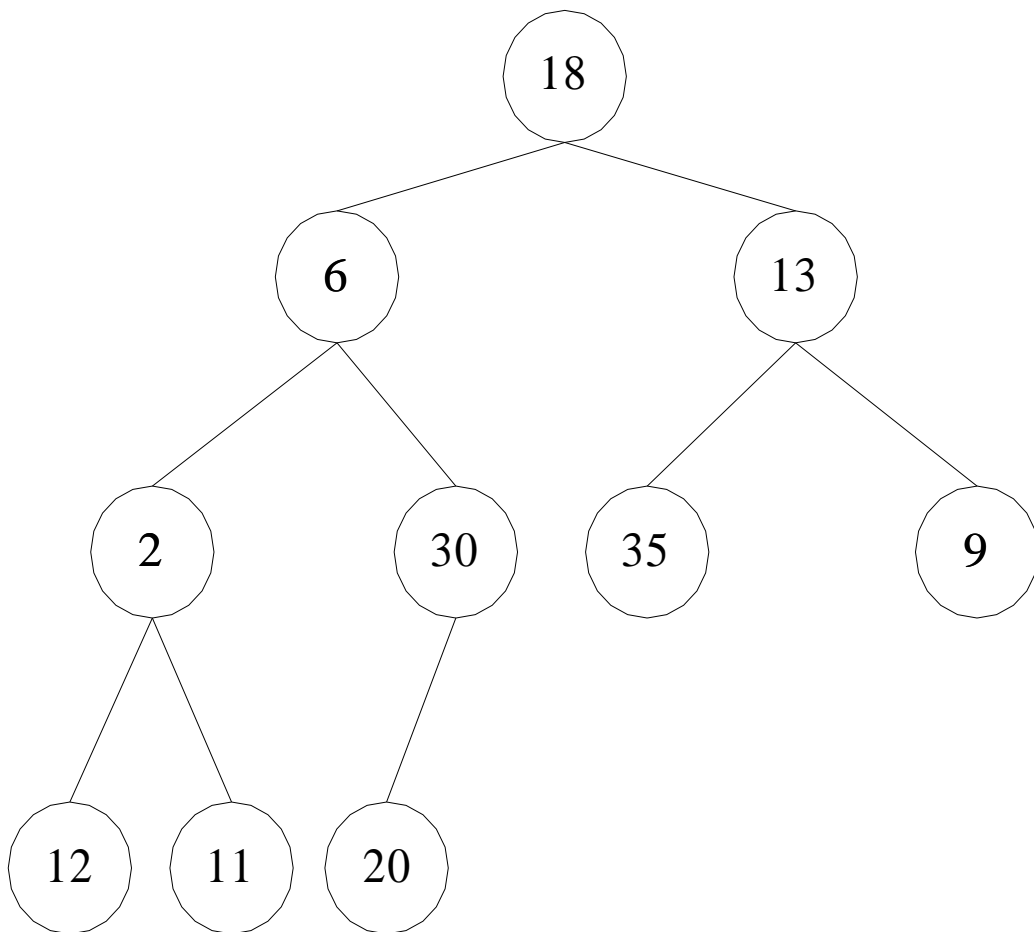
- Un heap se representa habitualmente mediante un array, por sencillez y por rapidez.
- El elemento que ocupa la posición 1 es el de clave más pequeña (o más grande).
- Los índices del array sirven para localizar cada nodo del heap: dado un cierto nodo i , su hijo izquierdo (si existe) ocupará la posición $2i$, y su hijo derecho (si existe) la posición $2i + 1$. Al mismo tiempo, el índice del nodo padre será $i \text{ div } 2$.
- Si, como en lenguaje C, disponemos de una posición 0, las alternativas son:
 1. no utilizarla para nada
 2. guardar en ella el número de nodos del heap

Heaps: ejemplo



Creación de un heap: *heapify*

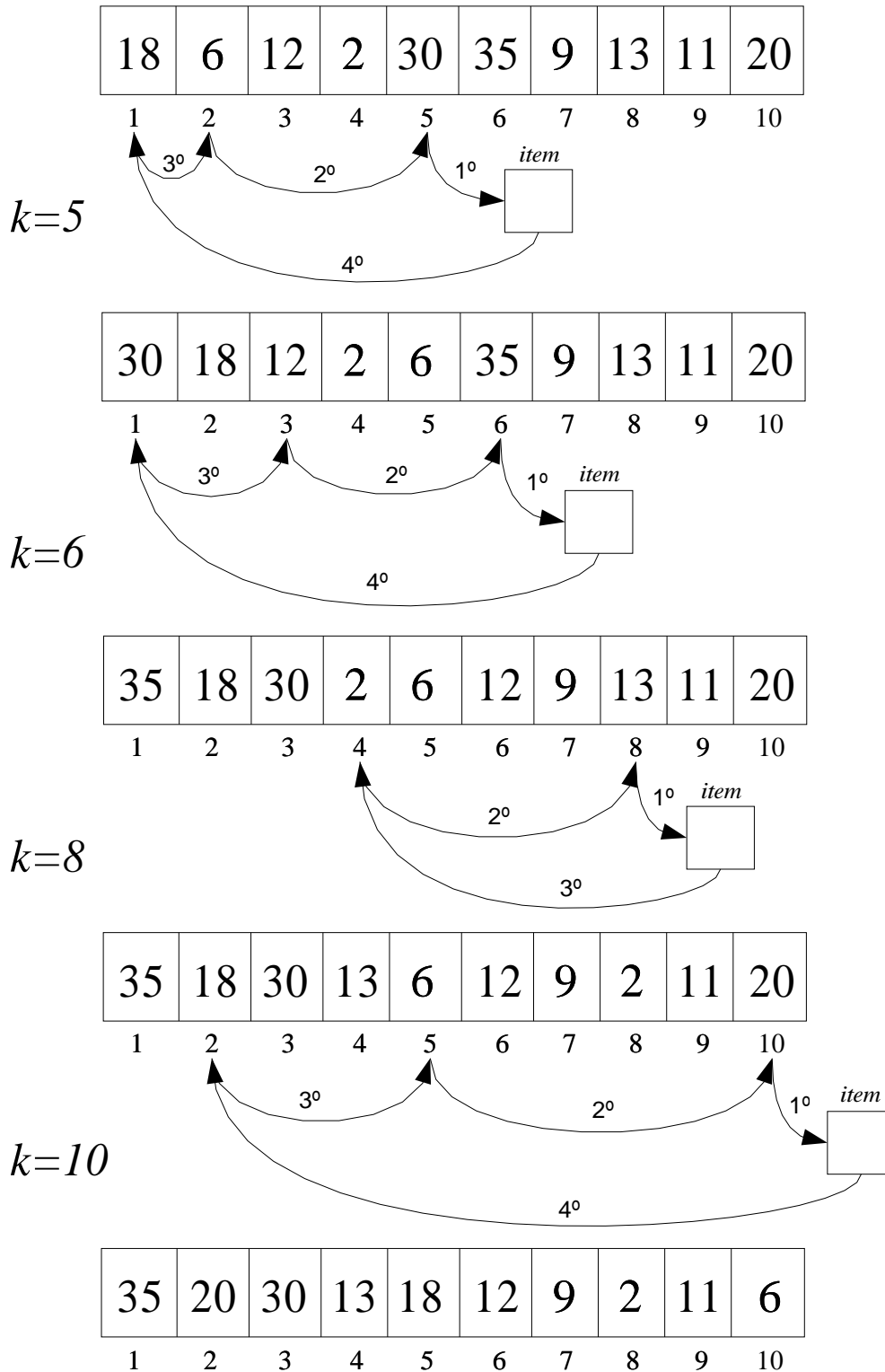
Estado inicial de la estructura



Heaps

Creación de un heap: *heapify*

Proceso de creación del heap



Procedimiento *heapify*

constantes

$MAX = 100$

fin_constantes

tipos

HEAP=registro

tamaño: **entero**

valor: **array**[1.. MAX] **de entero**

fin_registro

fin_tipos

procedimiento *heapify* (**ref** h: HEAP)

var

n, i, j, k, item: **entero**

fin_var

principio

paratodo $k \in [2..h.tamaño]$ **hacer**

$j \leftarrow k$

$i \leftarrow k \text{ div } 2$

$item \leftarrow h.valor[k]$

mientras $i > 0$ **AND** $h.valor[i] < item$ **hacer**

$h.valor[j] \leftarrow h.valor[i]$

$j \leftarrow i$

$i \leftarrow i \text{ div } 2$

fin_mientras

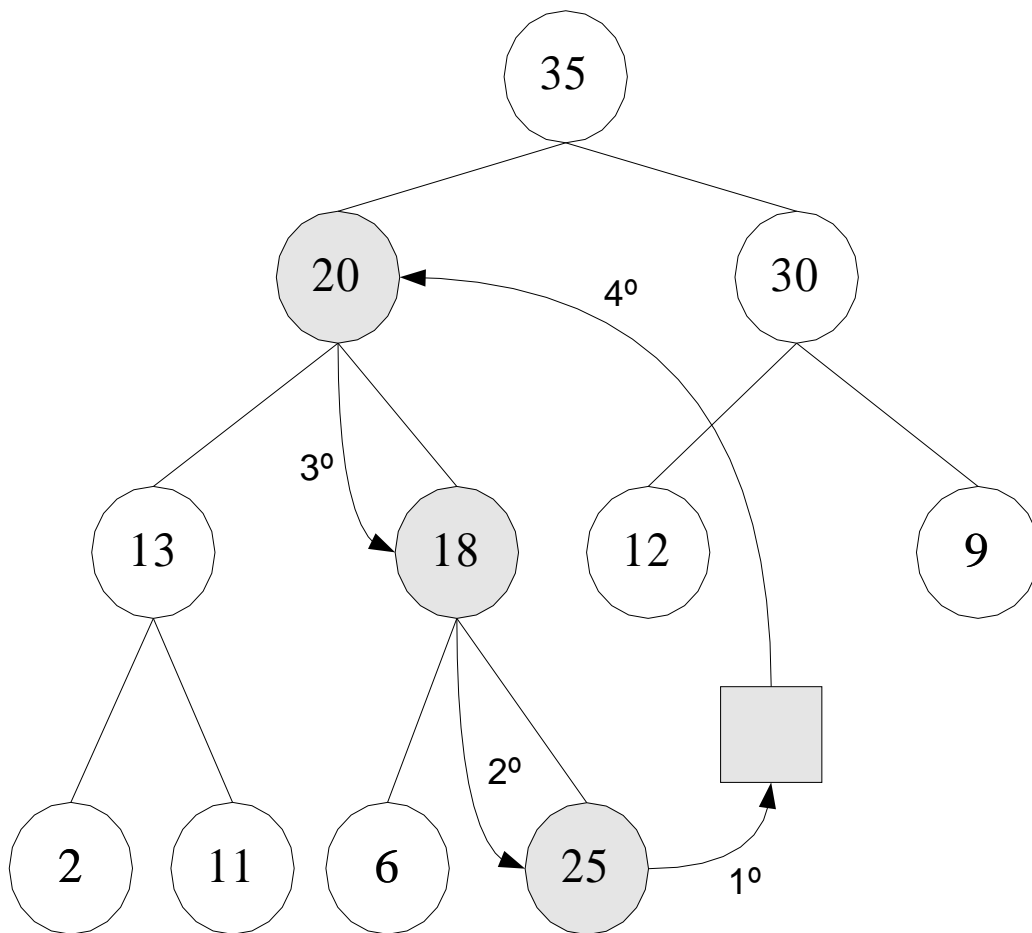
$h.valor[j] \leftarrow item$

fin_paratodo

fin /* *heapify* */

Heaps

Inserción de un nuevo elemento: *flotar*



Ejercicios

1. Suponiendo una implementación basada en arrays, escribir el algoritmo para insertar (*flotar*) un nuevo elemento en un heap. Estudiar su complejidad temporal.
2. Suponiendo una implementación basada en arrays, escribir un algoritmo para eliminar el elemento más pequeño (o más grande) del heap. Estudiar su complejidad temporal.
3. Describir una representación en memoria basada en listas ligadas. Indicar sus ventajas y desventajas con respecto a la representación basada en arrays.
 - a) Escribir un algoritmo equivalente a *heapify* suponiendo que los heaps se representan en memoria mediante listas ligadas.
 - b) Repetir los ejercicios 1 y 2 suponiendo que se está utilizando una representación en memoria basada en listas ligadas.