Árboles

- Presentar la estructura no lineal más importante en computación
- Mostrar la especificación e implementación de varios tipos de árboles
- Algoritmos de manipulación de árboles

Árboles

Contenido

1. Terminología fundamental
- **1.1. Recorridos de un árbol**

2. Árboles binarios
- **2.1. Definición**
- **2.2. Especificación**
- **2.3. Implementación**

3. Heap

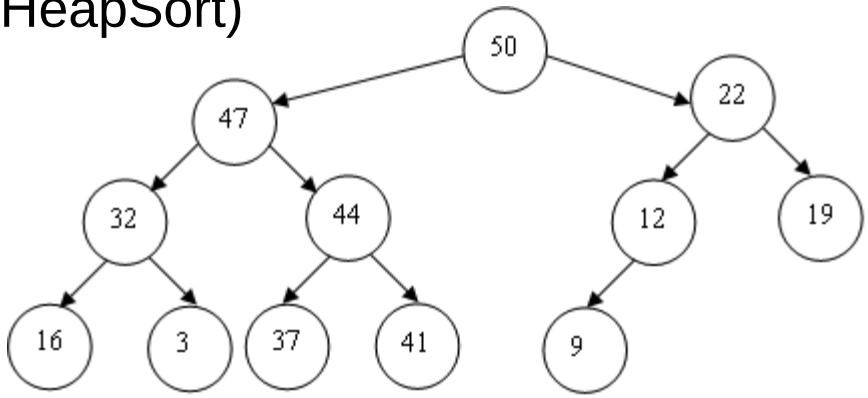
4. Árboles binarios de búsqueda
- 4.1. Definición
- 4.2. Especificación

5. Árboles binarios equilibrados
- 5.1. Árboles AVL

6. Árboles generales
- 6.1. Especificación

Heap

- Heap, montículo binario o montón es un árbol binario que satisface las siguientes condiciones:
  - Es completo, es decir, es un árbol completamente lleno, con la excepción del nivel inferior, que debe llenarse de izquierda a derecha
  - Las hojas están en dos niveles adyacentes
  - Para cada nodo X con padre P, se cumple que el dato en P es mayor o igual que el dato en X
- La propiedad de ordenación permite un rápido acceso al elemento de mayor prioridad, siempre situado en la raíz del árbol
- Se usan para implementar colas de prioridad (permite un rápido acceso al elemento de mayor prioridad) y para algoritmos de ordenación (HeapSort)



Heap

- **Operaciones para trabajar con Heap**

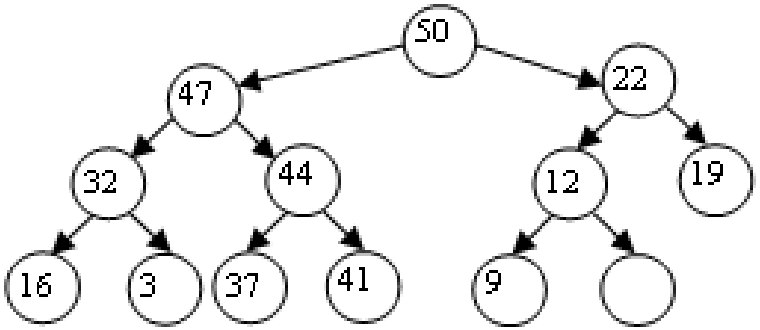
- Insertar: añade un elemento al heap, manteniendo la propiedad estructural y de ordenación.
- Suprimir el máximo: elimina el elemento máximo del montículo.
- Recuperar el máximo: consulta el elemento máximo del montículo.

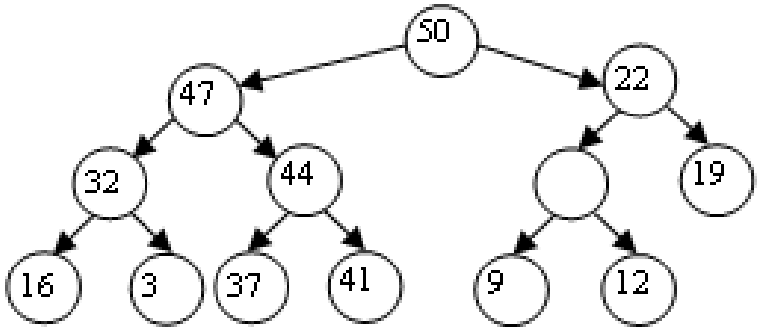
Heap

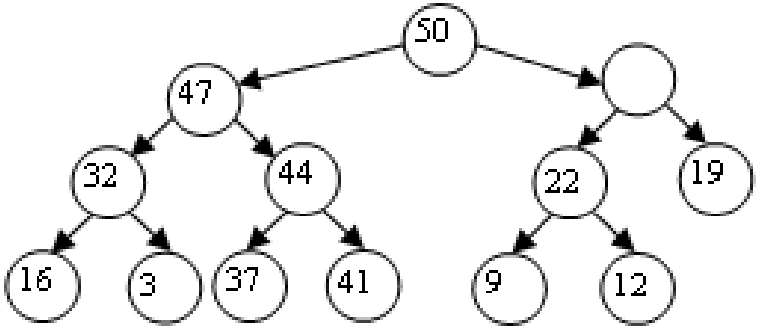
- **Algoritmo Insertar**
- Se crea un hueco en la siguiente posición disponible del vector
- Si el elemento se puede colocar en ese hueco sin violar la propiedad de ordenación del heap, se coloca y termina la inserción
- Sino, se desplaza el elemento situado en el padre del nodo a dicho hueco, moviéndolo así hacia la raíz
- Se continúa con el proceso hasta que se pueda colocar el elemento en el hueco

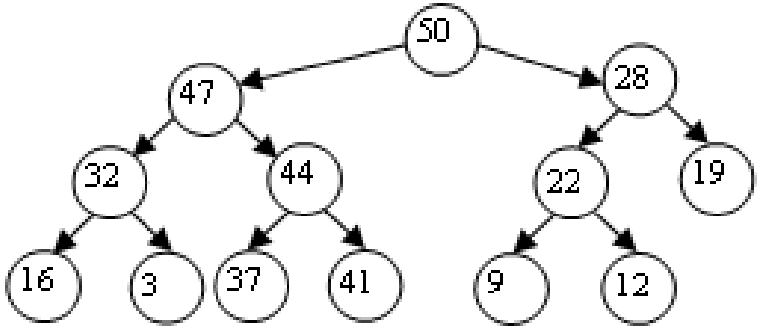
Heap

- **Ejemplo:** inserción de un objeto con prioridad 28







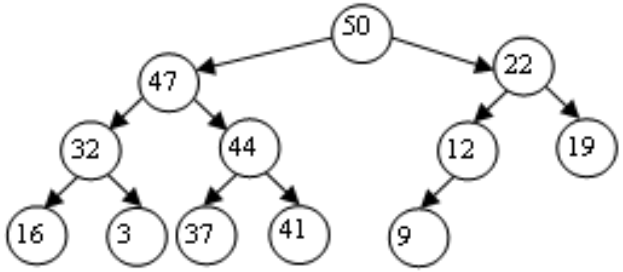


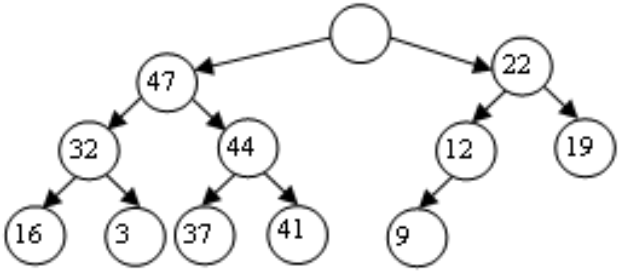
Heap

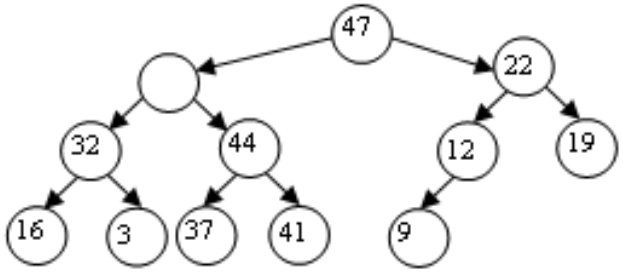
- **Algoritmo Suprimir**
  - Eliminar la raíz y se crea un hueco en la raíz
  - Buscamos el hijo del hueco con la prioridad mayor
  - Si ese hijo es mayor que el último nodo, movemos el hijo al hueco, empujando el hueco un nivel hacia abajo
  - Repetimos el proceso hasta que el último nodo se pueda colocar en el hueco

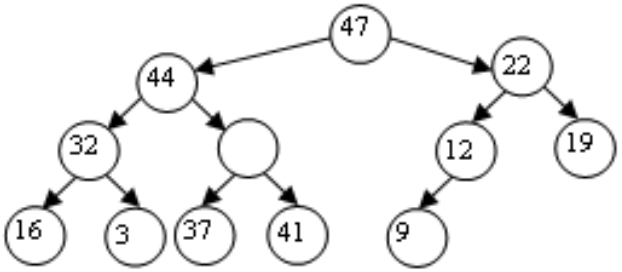
Heap

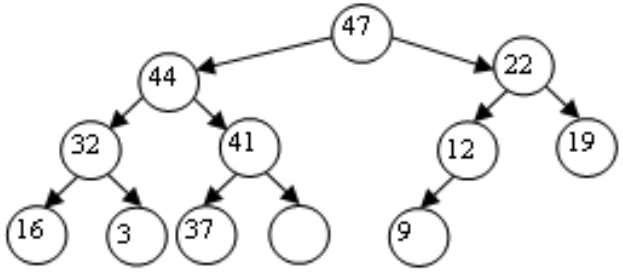
- **Ejemplo:** suprimir

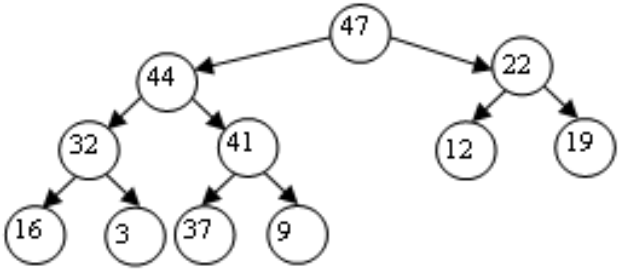












Heap

Ejercicios:

- Dibuja paso a paso el heap que resulta a partir de la siguiente entrada de datos: 4, 7, 12, 15, 3, 5, 14, 18, 16, 17, 2, 8. Una vez dibujado el heap elimina 3 elementos (raíz), mostrando el árbol que resulta en cada caso.
- Dibuja paso a paso el heap que resulta a partir de la siguiente entrada de datos: 14, 6, 5, 8, 1, 3, 12, 9, 7, 13 y 2. Una vez dibujado el heap elimina 3 elementos (raíz), mostrando el árbol que resulta en cada caso.

Heap

Especificación

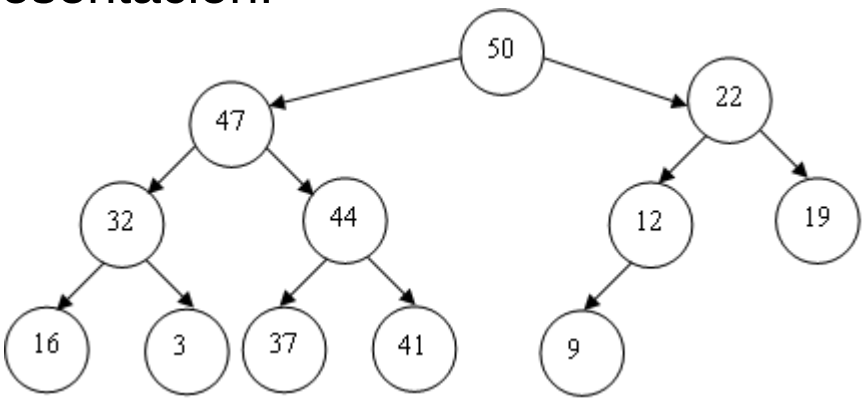
```
public class Heap<E>{
    // Declaración de tipos: Heap, Comparable
    // Características: un árbol binario completamente lleno, con la excepción del nivel inferior, y en donde cada nodo X con padre P, se cumple que P >=X
public Heap()
        // Produce: un objeto heap vacío
public boolean esVacio();
        // Produce: cierto si this está vacío. Falso en caso contrario.
public E recuperarMax() throws HeapVacioExcepcion;
        //Produce: si this está vacío lanza la excepción HeapVacioExcepcion, sino devuelve el objeto de más prioridad.
        Si varios objetos tienen igual prioridad devuelve el objeto que más tiempo lleva en el heap.
public E suprimirMax() throws HeapVacioExcepcion;
        // Modifica: this
        // Produce: si this está vacío lanza la excepción HeapVacioExcepcion, sino devuelve el objeto de más prioridad
        y lo suprime.
public void insertar(E e) throws IllegalArgumentException;
        //Modifica: this
        //Produce: añade el objeto e al heap.
public void anular();
        //Modifica: this
        //Produce: elimina todos los elementos de this, quedando vacío.
```

Heap

Implementación

- Un árbol binario completo se puede representar usando un array, colocando la raíz en la posición 1 y almacenando su recorrido por niveles en el vector. Dado un elemento en la posición i del vector:
  - Hijo izquierdo: posición 2i,
  - Hijo derecho: posición 2i +1,
  - Padre: posición i/2

- Ejemplo de representación:



	50	47	22	32	44	12	19	16	3	37	41	9		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Heap

- Todos los nodos excepto la raíz tienen padre:
  - Se deja sin elemento la posición 0
  - En esa posición se coloca un elemento falso que sirva como padre de la raíz $\Rightarrow$ simplificación de algunas operaciones
- Se necesita mantener un entero que indique cuántos nodos hay actualmente en el árbol.

- ¿Cómo comparar los elementos de los nodos?
  Solución: los elementos deben ser instancias de una clase que implemente la interface Comparable<E> existente en java.

```
public interface Comparable<E>{
        public int compareTo(E e);
}
```

Heap

```
public class Heap <E extends Comparable<E>>{
        private static final int CAPACIDAD = valor;
        private E [] vector;            // el vector del montículo
        private int numElem;            // número de elementos del montículo.
        …
        //implementación de las operaciones (actividad 4)
```