Árboles

Carlos Delgado Kloos
Mª Carmen Fernández Panadero
Raquel M. Crespo García
Ingeniería Telemática
Univ. Carlos III de Madrid

cdk@it.uc3m.es

Índice

rcrespo@it.uc3m.es

Cita

- *"The structure of concepts is formally called a **hierarchy** and since ancient times has been a basic structure for all western knowledge. Kingdoms, empires, churches, armies have all been structured into hierarchies. Tables of contents of reference material are so structured, mechanical assemblies, computer software, all scientific and technical knowledge is so structured..."*

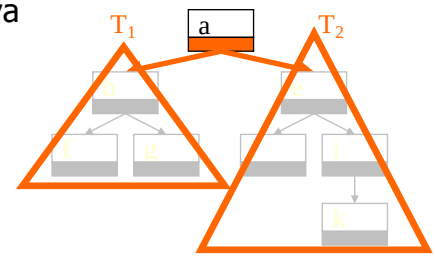
-- Robert M. Pirsig: *Zen and the Art of Motorcycle Maintenance*

cdk@it.uc3m.es

Árboles

¿Qué son?. Características

- Un **árbol** es una estructura de datos **no lineal** que almacena los elementos **jerárquicamente**
- Se puede definir de dos formas:
  - Definición no-recursiva
  - Definición recursiva



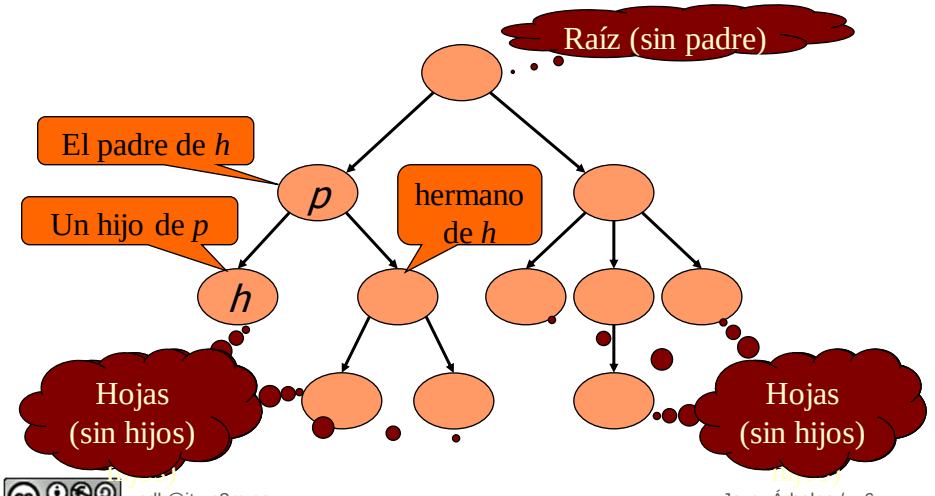
Definición Recursiva

mcfp@it.uc3m.es

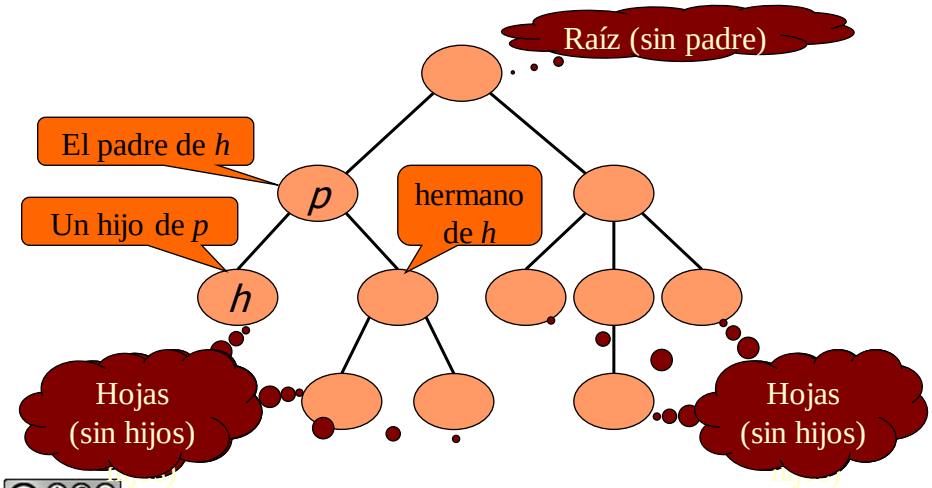
Definición no recursiva

- Un árbol consiste en un conjunto de **nodos** y un conjunto de **aristas**, de forma que:
  - Se distingue un nodo llamado **raíz**
  - A cada nodo *h*, excepto la raíz, le llega una arista de otro nodo *p* (*p* **padre** de *h*, *h* uno de los hijos de *p*)
  - Para cada nodo hay un **camino** (secuencia de aristas) **único** desde la raíz.

Ejemplo

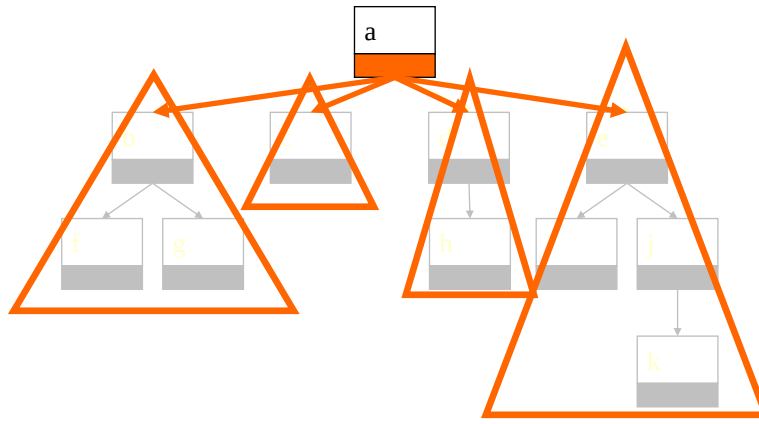


Ejemplo

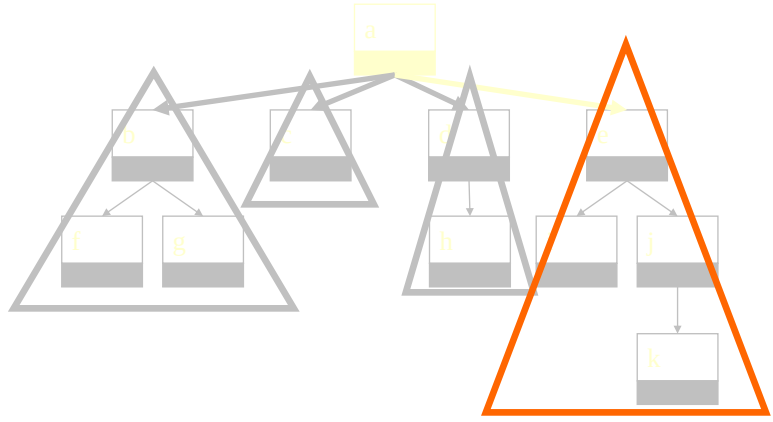


Árboles

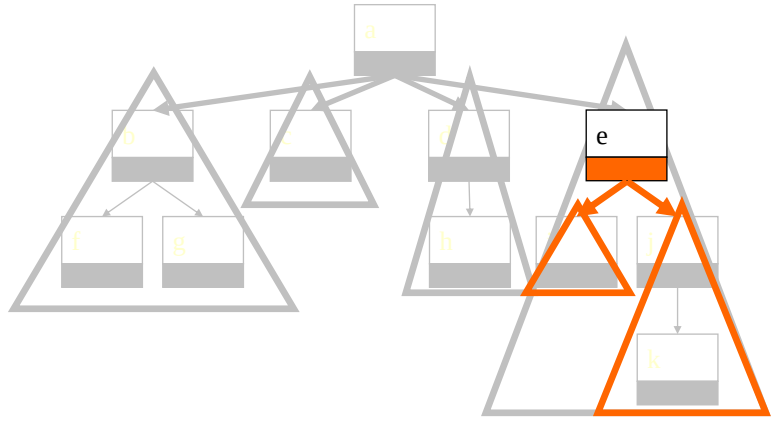
Definición recursiva



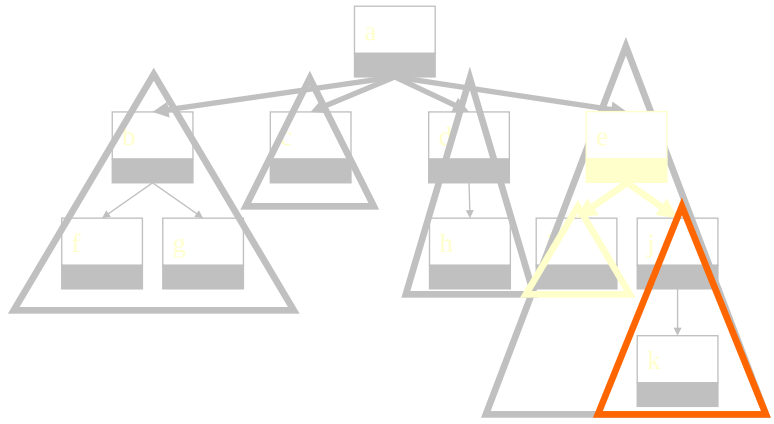
Árboles
Definición recursiva



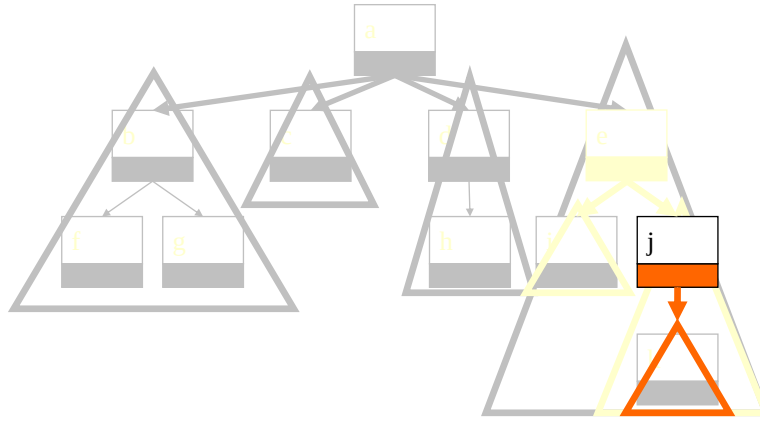
Árboles
Definición recursiva



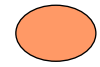
Árboles
Definición recursiva



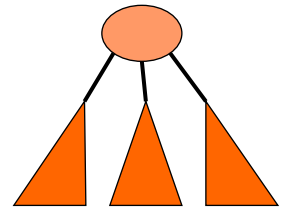
Árboles
Definición recursiva



Definición recursiva (1)

- Un árbol es
  - Un nodo 
  - o un nodo y subárboles conectados con el nodo por medio de una arista a su raíz

No incluye al árbol vacío



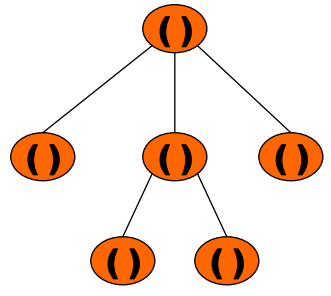
Definición recursiva (2)

- Un árbol es
  - vacío
  - o un nodo y cero o más subárboles no vacíos conectados con el nodo por medio de una arista a su raíz

Ejemplos

- Sistema de ficheros
- Estructura de un libro o de un documento
- Árbol de decisión
- Expresiones aritméticas

Ejemplo
Expresiones



(( ) (( ) ( )) ( ))

Terminología

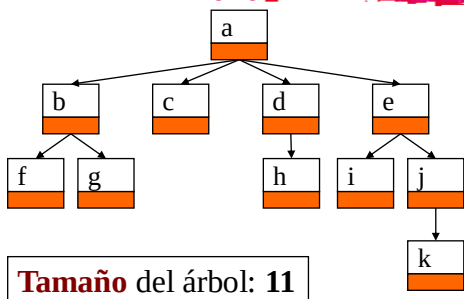
- Un nodo es **externo**, si no tiene hijos (es hoja)
- Un nodo es **interno**, si tiene uno o más hijos
- Un nodo es **ascendiente** de otro, si es padre de él o ascendiente de su padre.
- Un nodo es **descendiente** de otro, si este es ascendiente del primero
- Los descendientes de un nodo determinan un **subárbol** en el que ese nodo hace el papel de raíz

Terminología

- Un **camino** de un nodo a otro, es una secuencia de aristas consecutivas que llevan del primero al segundo.
  - Su *longitud* es el número de aristas que tiene.
- La **profundidad** de un nodo es la longitud del camino de la raíz a ese nodo.
- La **altura** de un árbol es la profundidad del nodo más profundo.
- El **tamaño** de un árbol es el número de nodos que contiene.

Ejemplo

Terminología y propiedades

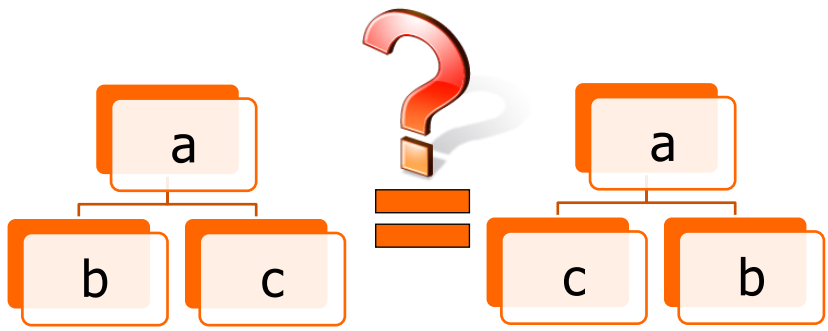


Tamaño del árbol: **11**
Altura del árbol: **3**

Nodo	Altura	Profundidad	Tamaño	Interno / Externo
a	**3**	0	**11**	Interno
b	1	1	3	Interno
c	0	1	1	Externo
d	1	1	2	Interno
e	2	1	4	Interno
f	0	2	1	Externo
g	0	2	1	Externo
h	0	2	1	Externo
i	0	2	1	Externo
j	1	2	2	Interno
k	0	**3**	1	Externo

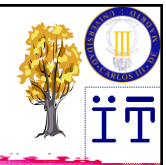
Terminología

Árbol ordenado



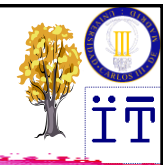
- Un árbol es **ordenado**, si para cada nodo existe un orden lineal para todos sus hijos.

Terminología
Árbol binario



- Un árbol **binario** es un árbol ordenado, en el que cada nodo tiene 0, ~~1~~ ó 2 hijos (el hijo izquierdo y el derecho).

Algoritmos básicos

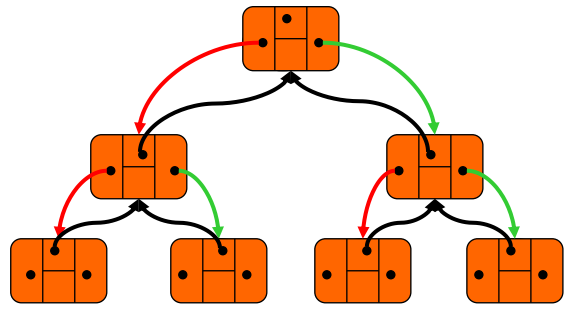


- Tamaño (número de nodos)
- Profundidad de un nodo
- Altura
- Recorridos
  - Euler
  - Pre-, in- y post-orden
- *Suponemos árboles binarios para simplificar*

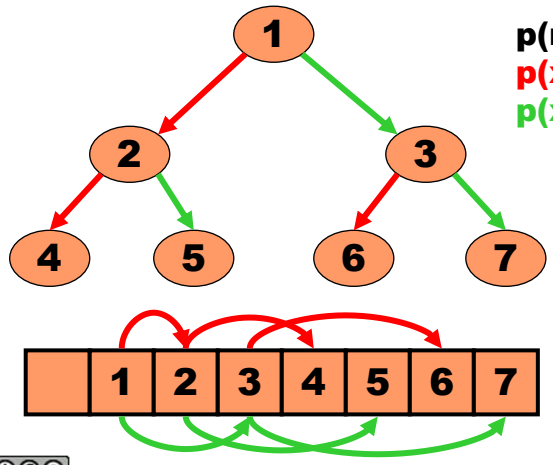
Implementaciones

- Basada en estructura enlazada
- Basada en secuencia

Implementación basada en enlaces



Implementación basada en secuencia



p(raiz)=1
p(x.izq)=2*p(x)
p(x.der)=2*p(x)+1

Clase árbol binario...

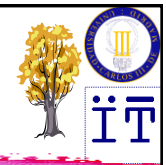
```
public class BTree {
  protected BNode root;

  BTree() {
    root = null;
  }

  BTree(Object info){
    root = new Bnode(info);
  }
```

...Clase árbol binario...

```
public int size() {
  int size = 0;
  if (root != null)
    size=root.size();
  return size;
}
```

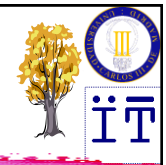
```
public int height(){
  int h = -1;
  if (root != null)
    h=root.height();
  return h;
}
```


... Clase árbol binario



```
public void preorder() {
  if (root!=null) root.preorder();
}
public void inorder() {
  if (root!=null) root.inorder();
}
public void postorder() {
  if (root!=null) root.postorder();
}
}
```

Clase nodo binario...

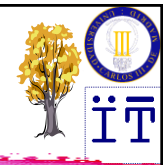


```
class BNode {
  private Object info;
  private BNode left;
  private BNode right;

  BNode() {
    this(null);
  }
  BNode(Object info) {
    this(info,null,null);
  }
  BNode(Object info, BNode l, BNode r) {
    this.info=info;
    left=l;
    right=r;
  }
```


... Clase nodo binario...

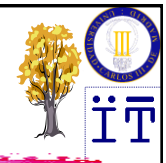
```
  int size (){...;}
  int height (){...;}

  void preorder (){...;}
  void inorder (){...;}
  void postorder (){...;}
}
```

Clase BNode: Size



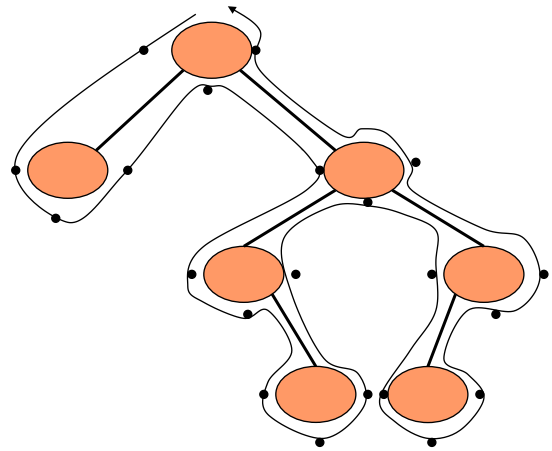
```
int size (){
  int size = 1;
  if (left != null)
    size += left.size();
  if (right != null)
    size += right.size();
  return size;
}
```


Clase BNode: Height



```
int height() {
 int hl = -1;
 int hr = -1;
 if (left !=null)
   hl = left.height();
 if (right !=null)
   hr = right.height();
 return 1 + Math.max(hl, hr);
}
```

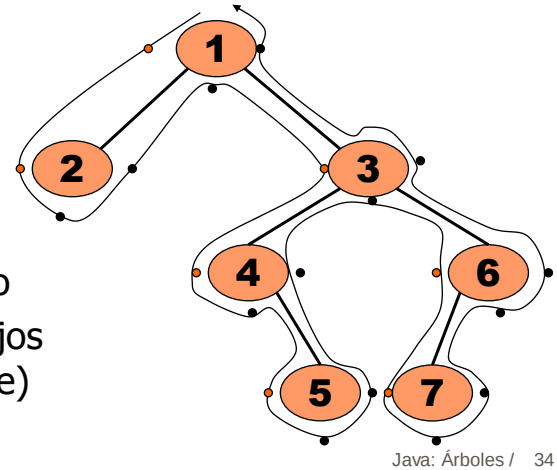
Recorrido de Euler



¡También aplicable a árboles no binarios!

cdk@it.uc3m.es

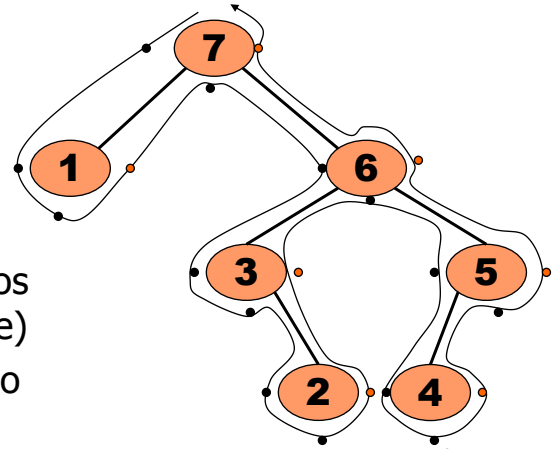
Recorrido preorden



- Primero el nodo
- Después sus hijos (recursivamente)

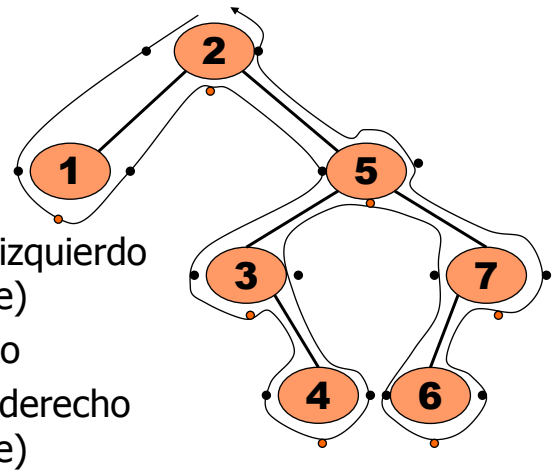
cdk@it.uc3m.es

Recorrido postorden



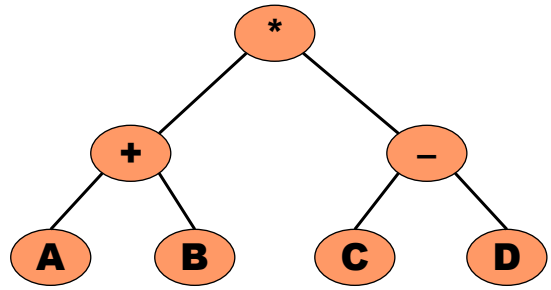
- Primero sus hijos (recursivamente)
- Después el nodo

Recorrido inorden (simétrico)



- Primero el hijo izquierdo (recursivamente)
- Después el nodo
- Primero el hijo derecho (recursivamente)

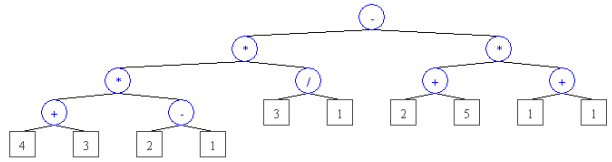
(A+B)*(C–D)



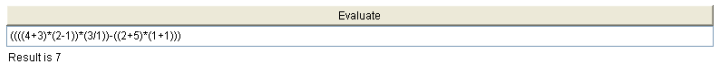
Ejemplo

Infijo	Prefijo	Posfijo
A+B	+AB	AB+
A+B–C	–+ABC	AB+C–
(A+B)*(C–D)	*+AB–CD	AB+CD–*

Actividad

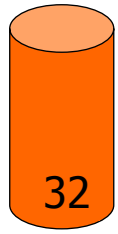


- Visualización de expresiones como árboles
  http://www.cs.jhu.edu/~goodrich/dsa/05trees/Demo1/



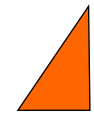
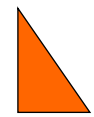
Notación postfijo

- Calculadoras HP
- Pila para almacenar operandos
- Ej.: ~~3 5 + 6 2 - *~~



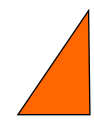
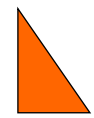
Clase BNode: preorder

```
void preorder (){
  System.out.println(info);
  if (left != null)
    left.preorder();
  if (right != null)
    right.preorder();
}
```

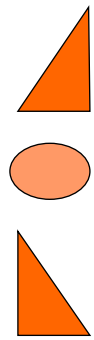
Clase BNode: postorder

```
void postorder (){
  if (left != null)
    left.postorder();
  if (right != null)
    right.postorder();
  System.out.println(info);
}
```


Clase BNode: inorder

```
void inorder (){
  if (left != null)
    left.inorder();
  System.out.println(info);
  if (right != null)
    right.inorder();
}
```



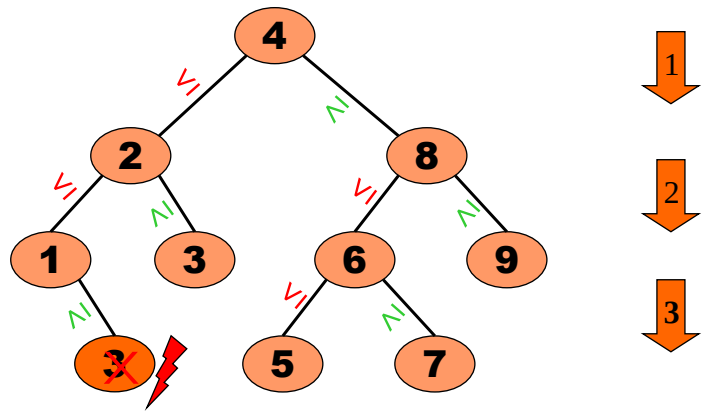
Propiedades de árboles binarios

- Sea
  - E=Número de nodos externos
  - I=Número de nodos internos
  - N=Tamaño=E+I
  - H=Altura
- Se cumple
  - E=I+1
  - $H+1 \leq E \leq 2^H$ $H \leq I \leq 2^H-1$ $2*H+1 \leq N \leq 2^{H+1}-1$
  - $\log_2(N+1)-1 \leq H \leq (N-1)/2$

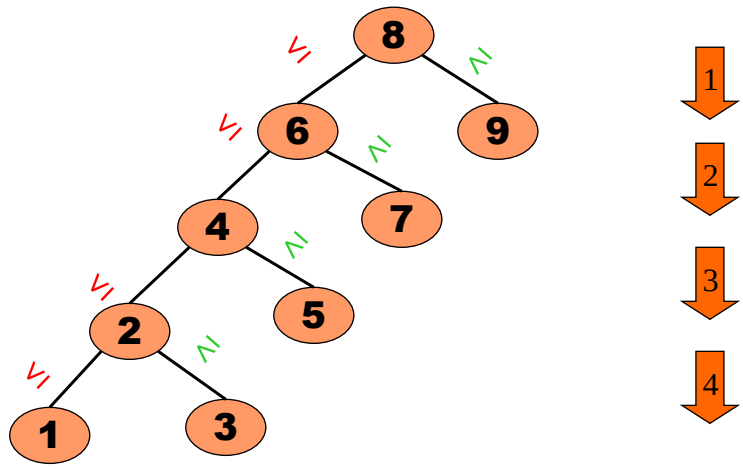
Árboles binarios de búsqueda

- Un **árbol binario de búsqueda** es un árbol binario en el que para cada nodo *n*,
  - todas las claves de los nodos del subárbol **izquierdo** son **menores** que la clave de *n* (o iguales)
  - y todas las del subárbol **derecho** **mayores** (o iguales).

Ejemplo



Ejemplo



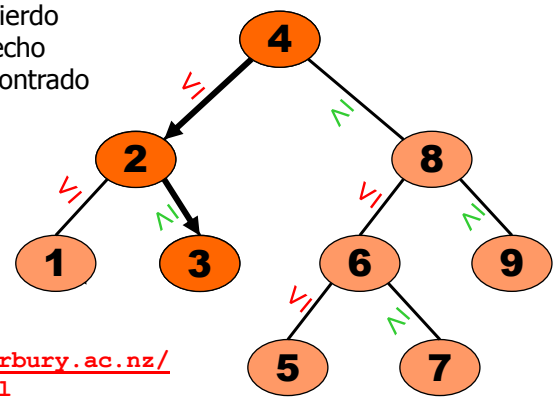
Operaciones

- Búsqueda
- Inserción
- Eliminación

Búsqueda

Buscamos el "3":

- 3<4: Subárbol izquierdo
- 3>2: Subárbol derecho
- 3=3: Elemento encontrado

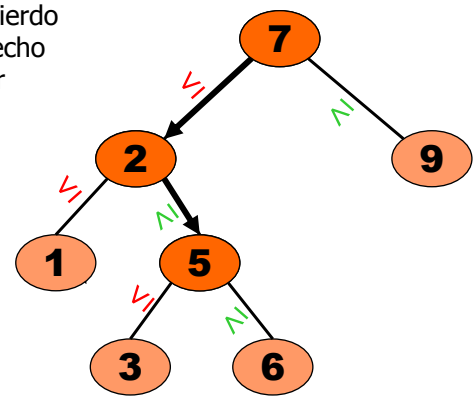


http://www.cosc.canterbury.ac.nz/mukundan/dsal/BST.html

Inserción

Insertar "6":

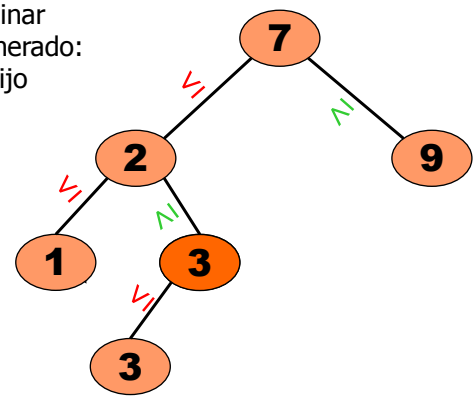
- 6<7: Subárbol izquierdo
- 6>2: Subárbol derecho
- Hueco libre: insertar



Eliminación (1)

Eliminar "5":

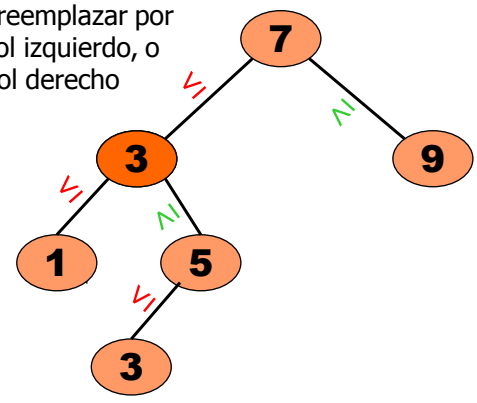
- si es una hoja: eliminar
- si es un nodo degenerado: reemplazar por el hijo



Eliminación (2)

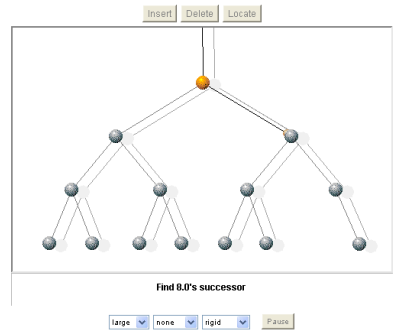
Eliminar "2":

- si el nodo tiene 2 hijos: reemplazar por
  - el mayor del subárbol izquierdo, o
  - el menor del subárbol derecho



Actividad

- Ver animación de árboles binarios de búsqueda
  http://www.ibr.cs.tu-bs.de/courses/ss98/audii/applets/BST/BST-Example.html



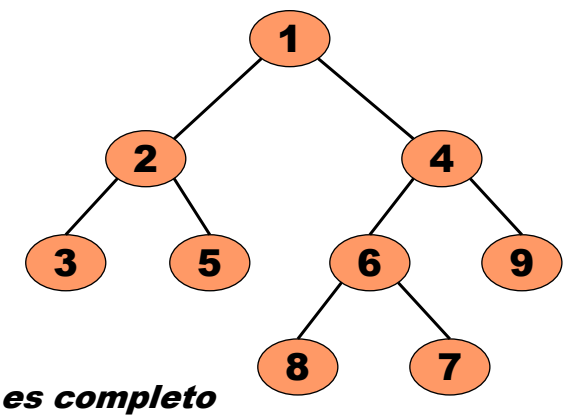
Montículos (Heaps)

- Un **montículo** binario es un árbol binario completo en el que cada nodo tiene una clave mayor(*) o igual que la de su padre.
  - Normalmente, un montículo se refiere a montículo binario
  - * También podría definirse como menor o igual (el orden es arbitrario)
- Aplicaciones:
  - Colas con prioridad
  - Ordenación

Montículos: propiedades

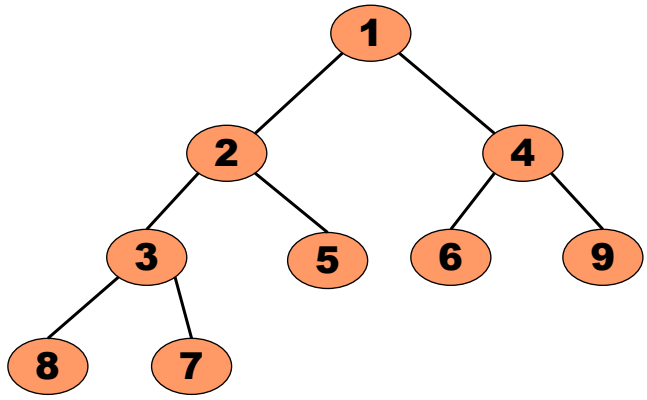
- Un **montículo** cumple dos propiedades
  - Propiedad de montículo: para cada nodo *n* (excepto para el raíz), su clave es mayor o igual que la de su padre.
  - Completo

Ejemplo: propiedad de montículo

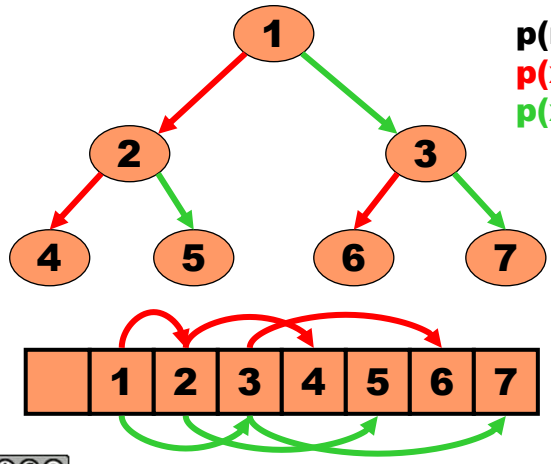


Pero no es completo

Ejemplo: Montículo completo

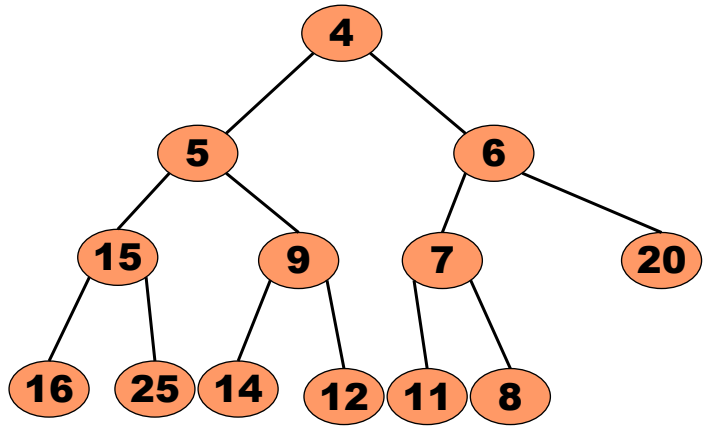


Implementación basada en secuencias

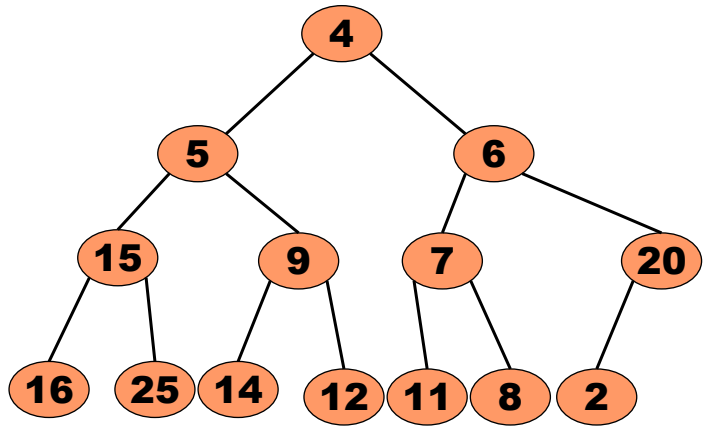


p(root)=1
p(x.left)=2*p(x)
p(x.right)=2*p(x)+1

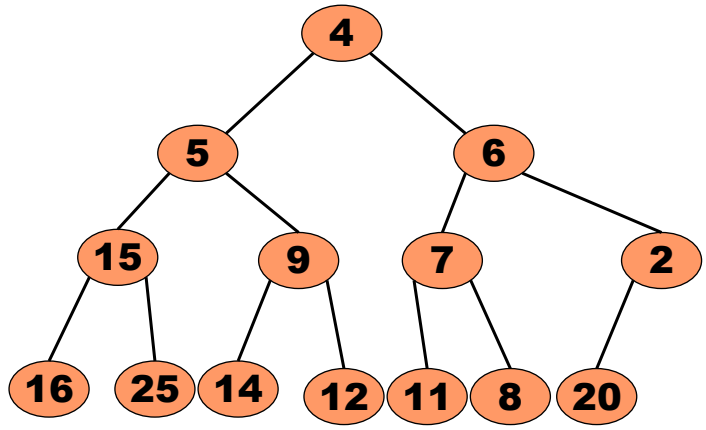
Insertar



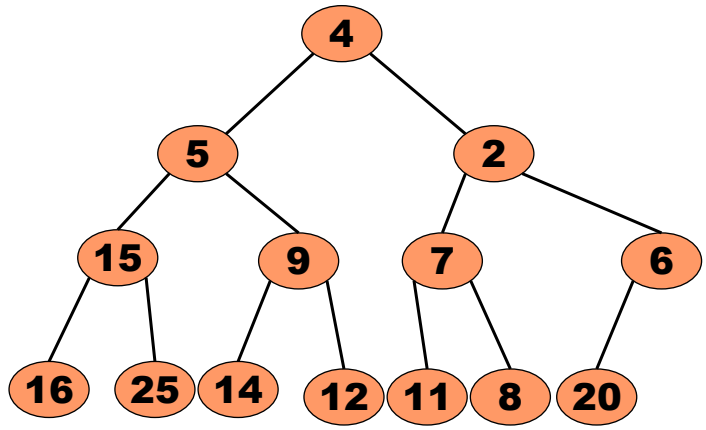
Insertar



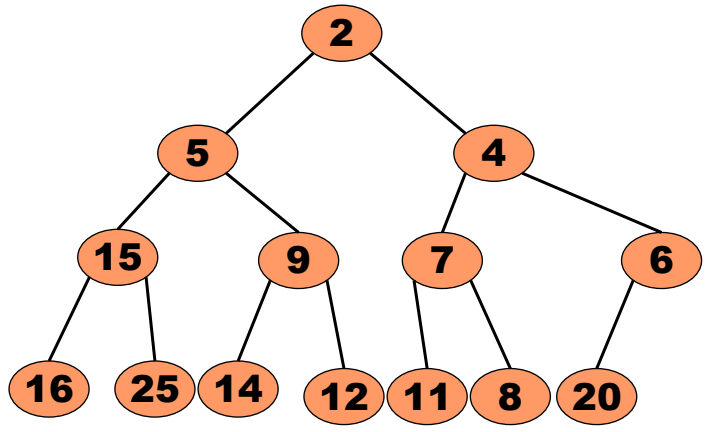
Insertar



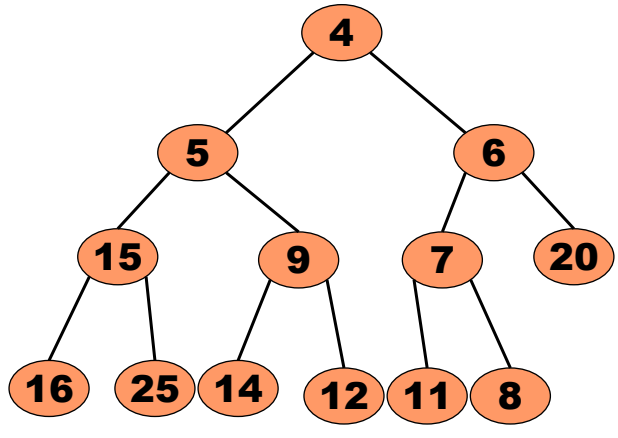
Insertar



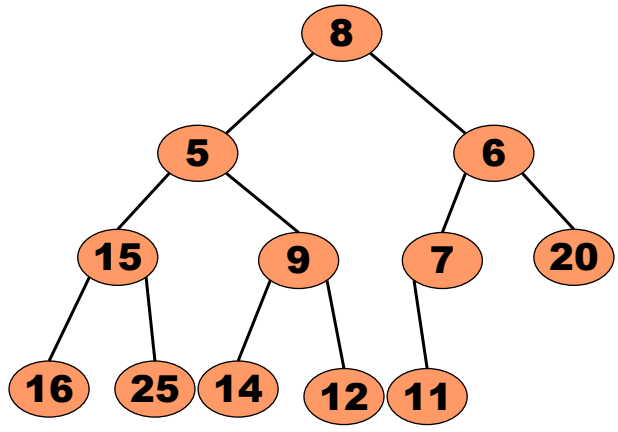
Insertar



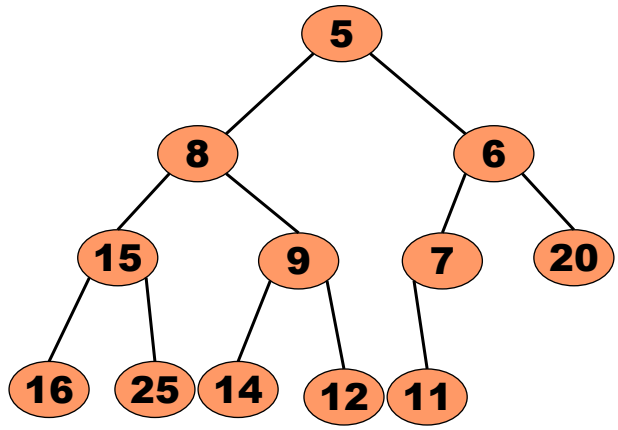
Eliminar



Eliminar

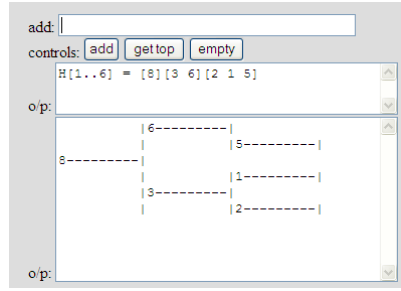


Eliminar



Actividad

- Probar el formulario de
  http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Priority-Q/



Actividad

- Probar el applet de
  http://www.cosc.canterbury.ac.nz/mukundan/dsal/MinHeapAppl.html