Estructuras de datos no lineales - Parte I
Árboles

Luisa Micó

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante

Index

Index

Objetivos

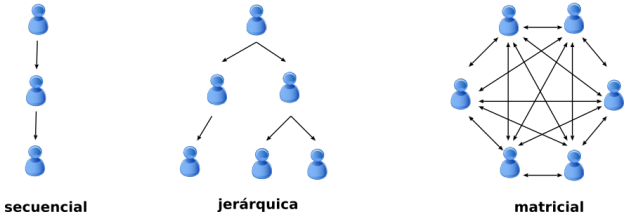
- describir la estructura de dato árbol y sus operaciones
- conocer algunos ámbitos de aplicación de árboles
- conocer diferentes versiones de árboles de búsqueda: binarios de búsqueda, árboles AVL, árboles 2-3-4, árboles rojos-negros
- conocer el coste de las operaciones sobre los diferentes tipos de árboles
- conocer los árboles en `java.util`

Visualizaciones:
`https://www.cs.usfca.edu/~galles/visualization/Algorithms.html`

Index

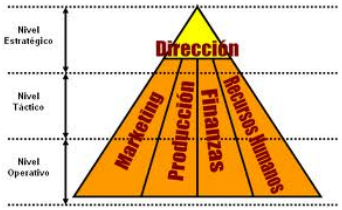
Introducción

Las listas en general no permiten una organización jerárquica de los datos.



Ejemplos:

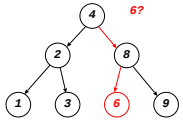
estructura de empresa

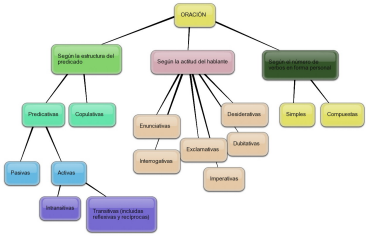


estructura de un documento

Más ejemplos

- Árbol genealógico (familiar, pedigrí de animales, ...)
- Estructura para acelerar búsquedas (árboles, grafos, tablas ...)
- Sistemas de ficheros (directorios, subdirectorios, ficheros)
- Estructura gramatical de oraciones (oración, sintagmas, elementos gramaticales ...)

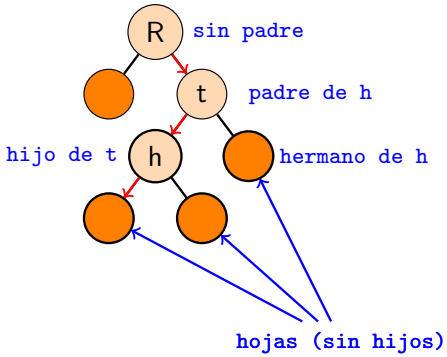




Definición de árbol

Un árbol es un **conjunto de nodos** y un **conjunto de aristas**, de forma que:

- se distingue un nodo R llamado **raíz**
- a cada nodo h, excepto la raíz, le llega una arista de otro nodo t
- para cada nodo hay un **camino** (secuencia de aristas) único desde la raíz.



Definición recursiva de árbol

1. una estructura vacía es un árbol vacío
2. si $t_1, t_2, \ldots, t_k$ son árboles disjuntos, entonces la estructura cuya raíz tiene como sus hijos las raíces de $t_1, t_2, \ldots, t_k$ también es un árbol
3. sólo las estructuras generadas por las reglas 1 y 2 son árboles

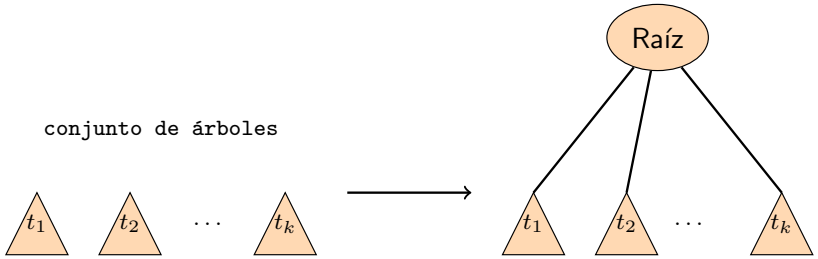
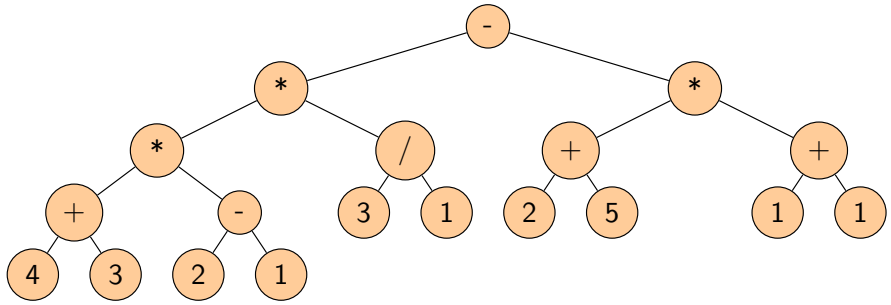


Figure: Obtención de un árbol a partir de un conjunto de árboles

Aplicación: expresión aritmética

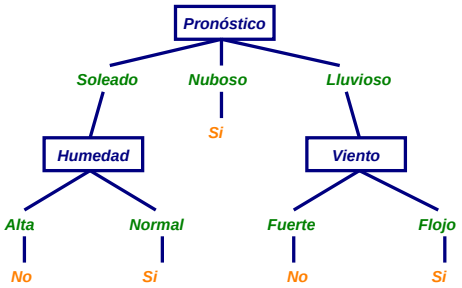
Ejemplo: ((((4+3)*(2-1))*(3/1))-((2+5)*(1+1)))



Aplicación: árbol de decisión (otros)

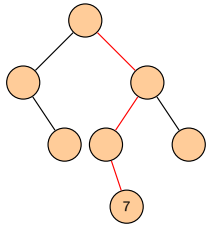
¿Jugamos al tenis?

Day	Pronóstico	Temperatura	Humedad	Viento	JuegaTenis
D1	Soleado	Calor	Alta	Flojo	No
D2	Soleado	Calor	Alta	Fuerte	No
D3	Nuboso	Calor	Alta	Flojo	Si
D4	Lluvioso	Suave	Alta	Flojo	Si
D5	Lluvioso	Fresco	Normal	Flojo	Si
D6	Lluvioso	Fresco	Normal	Fuerte	No
D7	Nuboso	Fresco	Normal	Fuerte	Si
D8	Soleado	Suave	Alta	Flojo	No
D9	Soleado	Fresco	Normal	Flojo	Si
D10	Lluvioso	Suave	Normal	Flojo	Si
D11	Soleado	Suave	Normal	Fuerte	Si
D12	Nuboso	Suave	Alta	Fuerte	Si
D13	Nuboso	Calor	Normal	Flojo	Si
D14	Lluvioso	Suave	Alta	Fuerte	No



Terminología I

- Camino. Secuencia de arcos desde la raíz a un nodo dado.
- Longitud de un camino. Es el número de arcos en un camino.
- Nivel o profundidad de un nodo. Longitud del camino desde la raíz al nodo más 1.
- Altura de un árbol. Nivel máximo de un nodo en el árbol (no vacío).
- Árbol vacío. Tiene altura 0 por definición (un nodo individual es un árbol de altura 1).
- Grado de un nodo. Es el número de hijos del nodo (las hojas tienen grado 0).
- Grado de un árbol. Grado máximo de los nodos del árbol.

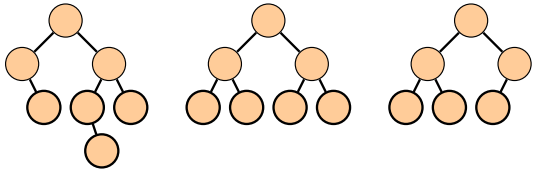


Ejemplo.

- Camino: secuencia en rojo
- Longitud del camino: ??
- Nivel del nodo 7 y altura del árbol: ??
- Grado del nodo 7 y del árbol: ??

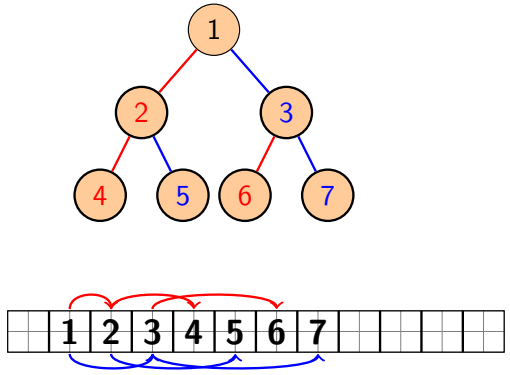
Terminología II

- Árbol general. Puede contener un número indeterminado de subárboles.
- Árbol n-ario. Puede contener un número máximo (n) de subárboles.
- Árbol binario equilibrado. Es aquel en el que la altura de los subárboles izquierdo y derecho de cualquier nodo no difieren en más de una unidad.
- Árbol binario lleno. Árbol 2-ario donde todos los nodos internos tienen dos hijos, excepto las hojas que se encuentran todas en el mismo nivel (totalmente equilibrado).
- Árbol binario completo. Árbol binario que tiene todos los nodos posibles hasta el penúltimo nivel, y donde los elementos del último nivel están colocados de izquierda a derecha sin dejar huecos entre ellos.
- Bosque. Conjunto de dos o más árboles.

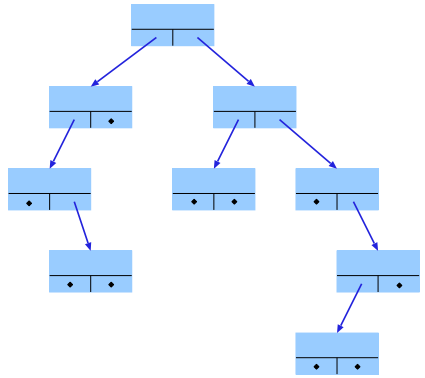


Implementaciones de árboles binarios

Basada en secuencias



Basada en enlaces



$p(x) = 1$
$p(x.hi) = 2 * p(x)$
$p(x.hd) = 2 * p(x) + 1$

Construcción de un árbol binario

```
public class ABBInt {

  private class NodeABBInt {
    int key;
    NodeABBInt hi, hd;

    NodeABBInt() { hi = hd = null; }
    NodeABBInt( int e, NodeABBInt a, NodeABBInt b) {
      key = e; hi = a; hd = b;
    }
  }

  private NodeABBInt raiz;

  public ABBInt() { raiz = null; }

  public void Inorden() { inorden( raiz ); }
  private void inorden ( NodeABBInt p ){ ... }

  public void Inserta( int e ) { ... }

  public int size(){ return size( raiz ); }
  ...
}
```


Algunos algoritmos básicos

- Tamaño (número de nodos)

```
private int size( NodeABBInt p ) {

  if ( p == null ) return 0;

  else return ( 1 + size( p.hi ) + size( p.hd ) );

}

public int size() { return size( raiz ); }
```

- Altura

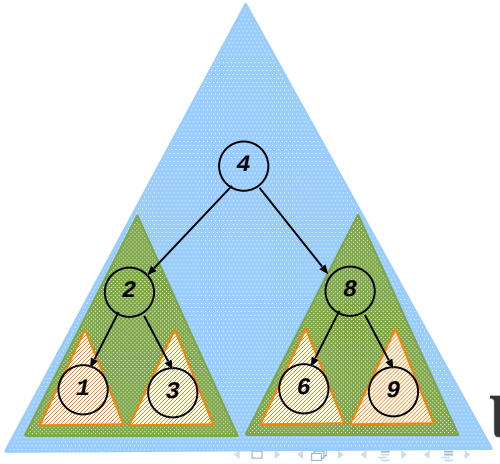
```
private int height (NodeABBInt a){

  if (a==null) return (-1);

  else
  return 1 + Math.max( height( a.hi ), height( a.hd ));

}

public int height (){ return height( raiz ); }
```

Recorridos en árboles

El recorrido de un árbol es el proceso de visitar cada nodo en el árbol exactamente una vez. Al no especificarse el orden de visita, hay tantos recorridos como permutaciones de nodos.

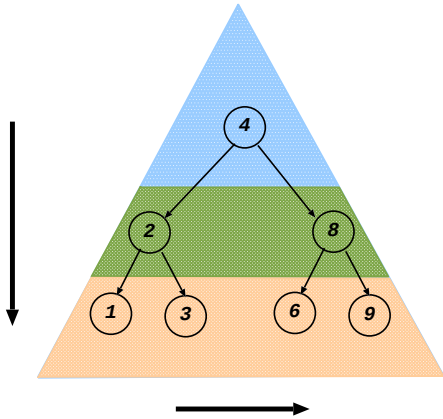
Los recorridos clásicos y útiles:

- **Recorrido en profundidad** (recursivo)
  - recorrido en preorden (Raiz-Iz-De)
  - recorrido en inorden (Iz-Raiz-De)
  - recorrido en postorden (Iz-De-Raiz)
- **Recorrido en anchura** (iterativo)



Recorrido en anchura

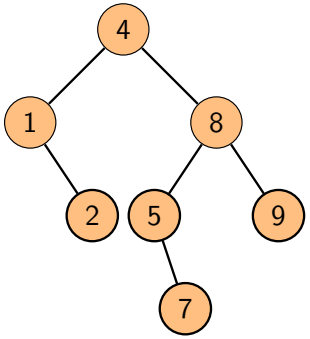
También llamado `recorrido en amplitud` o `recorrido por niveles` consiste en visitar cada nodo empezando por la raíz del árbol y moverse hacia abajo nivel por nivel, visitando nodos en cada nivel de izquierda a derecha.
La implementación es sencilla utilizando una cola.



Algoritmo de recorrido en anchura en un árbol binario

procedure ANCHURA()
  Q: cola;
  encola(raíz, Q);
  while (Q no está vacía) **do**
    v = desencola(Q); visita(v);
    if (hijoIz(v) no es vacío) **then**
      encola(hijoIz(v), Q);
    end if
    if (hijoDe(v) no es vacío) **then**
      encola(hijoDe(v), Q);
    end if
  end while
end procedure

Ejemplos de recorridos



Recorridos:

- Niveles: 4 1 8 2 5 9 7
- Preorden: 4 1 2 8 5 7 9
- Inorden: 1 2 4 5 7 8 9
- Postorden: 2 1 7 5 9 8 4

¿Es posible reconstruir un árbol a partir de un recorrido?
¿Y de dos?
¿Y de tres?

Profundidad y número de nodos en un árbol binario

Si n es el número de nodos en un árbol binario y h es la profundidad, entonces se cumple que $h \geq \lfloor \log(n) \rfloor$

- cada nodo tiene como máximo 2 nodos internos, entonces en cada nivel i existen a lo sumo 2^i nodos
- el número total de nodos está acotado por $n \leq 1 + 2 + 2^2 + 2^3 + \ldots + 2^{h-1} = \sum_{i=0}^{h-1} 2^i = \frac{2^h - 1}{2-1} = 2^h - 1$
- ... que es equivalente a decir que $\lfloor \log(n) \rfloor < h$

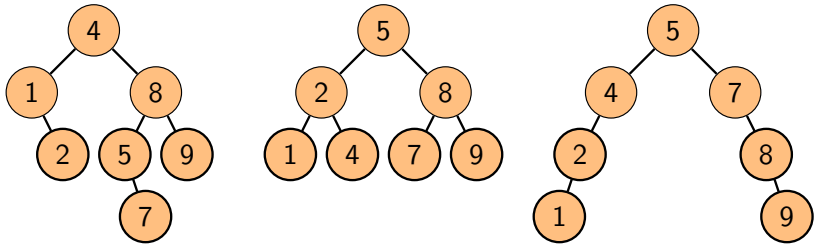
Index

Árbol binario de búsqueda (ABB)

Un ABB es un árbol vacío o cumple las siguientes propiedades:

- el nodo raíz contiene dos subárboles h_0 y h_1 y una etiqueta k_1
- se cumple que
  - toda etiqueta k_i en h_0 es menor que k_1
  - toda etiqueta k_i en h_1 es mayor que k_1
- todo subárbol h_i es un ABB

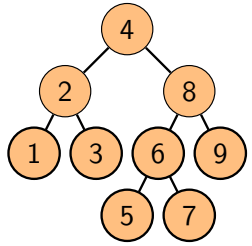
Se obtienen diferentes árboles dependiendo del orden en el que se insertan las etiquetas.



Búsqueda en ABB

Busquemos el "3" en el siguiente árbol:

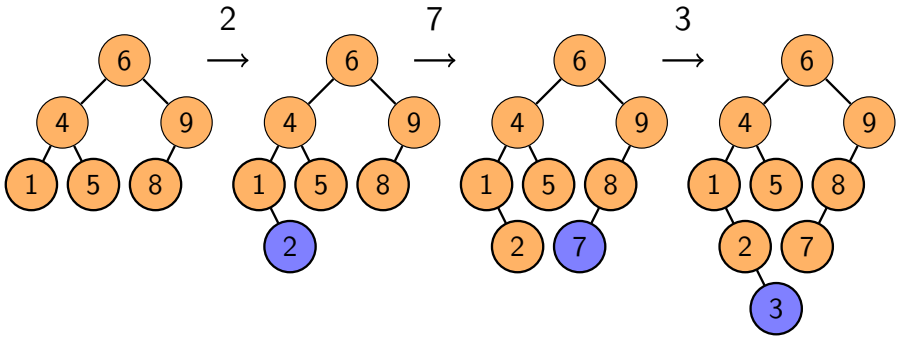
- $3 < 4 \rightarrow$ subárbol izquierdo
- $3 > 2 \rightarrow$ subárbol derecho
- $3 = 3 \rightarrow$ elemento encontrado



`https://www.cs.usfca.edu/~galles/visualization/BST.html`

Inserción en ABB

- las inserciones siempre se realizan en una hoja, aplicando el algoritmo de búsqueda
- si la etiqueta ya existe en el árbol, la relación de orden que se suele aplicar es $\leq$

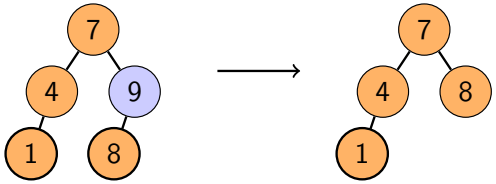


Implementación de un ABB

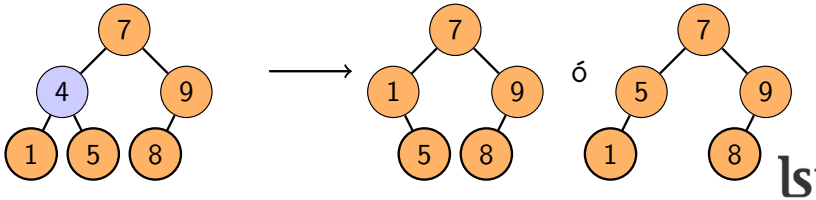
```
//inserción en un árbol binario de búsqueda
public void inserta( int e ) {
  NodeABBInt p = raiz, ant = null;
  while ( p != null ) { //encuentra un lugar para insertar
    ant = p;
    if ( p.key < e )
      p = p.hd;
    else
      p = p.hi;
  }
  if ( raiz == null )  //el árbol está vacío
    raiz = new NodeABBInt( e );
  else if ( ant.key < e )
    ant.hd = new NodeABBInt( e );
  else
    ant.hi = new NodeABBInt( e );
}

//recorrido en inorden en un árbol binario de búsqueda
public void Inorden() { inorden( raiz ); }

private void inorden ( NodeABBInt p ){
  if ( p != null ) {
    inorden( p.hi );
    visita( p );
    inorden( p.hd );
  }
}
```


Borrado en ABB

- si la etiqueta está en una hoja, se borra la hoja
- si la etiqueta está en la raíz de un subárbol con un único hijo, se sustituye el nodo a eliminar por su hijo



- si la etiqueta está en la raíz de un subárbol con dos hijos, se sustituye el nodo por la etiqueta mayor en el subárbol izquierdo o la menor en el subárbol derecho



Ejercicios

1. dibuja y da los recorridos en inorden, preorden, postorden y por niveles sobre un árbol binario de búsqueda inicialmente vacío en el que se han realizado las inserciones en el orden en que se dan a continuación. ¿Qué tipo de árbol se obtiene en cada caso?
   - 4,8,1,2,9,6,5,0,3,7
   - 0,9,3,8,4,6,5
   - 5,3,2,4,7,6
   - 5,2,1,7,3,6,8
2. dibuja y da los recorridos en inorden, preorden, postorden y por niveles de los árboles binarios de búsqueda obtenidos en el apartado anterior, tras realizar el borrado de la etiqueta 3
3. idem con la etiqueta 6
4. idem con la etiqueta 5

Coste de las operaciones sobre un ABB

Suponiendo que la altura de un ABB es a:

- ¿coste de búsqueda en el mejor y peor caso?
- ¿coste de inserción en el mejor y peor caso?
- ¿coste de borrado en el mejor y peor caso?

Suponiendo que el número de nodos de un ABB es n:

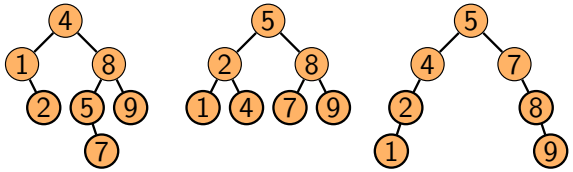
- ¿coste de búsqueda en el mejor y peor caso?
- ¿coste de inserción en el mejor y peor caso?
- ¿coste de borrado en el mejor y peor caso?

Index

Árboles equilibrados: AVL

- Árbol completamente equilibrado. Diferencia entre el número de nodos entre los subárboles izquierdo y derecho es como máximo uno.
  - Ventajas: búsqueda muy rápida
  - Desventajas: mantenimiento del árbol
- Árbol equilibrado (AVL). Diferencia en la altura de los dos subárboles de cualquier nodo en el árbol es cero o uno.
  - Ventajas: búsqueda rápida y mantenimiento asequible
  - Desventajas: mantenimiento del árbol en el peor caso ($\log_2(n)$ operaciones, siendo $n=$ número de nodos)

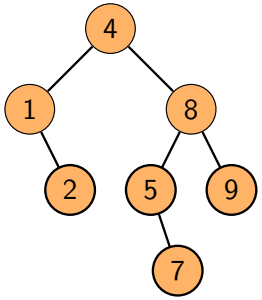
En los siguientes ejemplos, ¿existe algún árbol AVL, completamente equilibrado o no equilibrado?



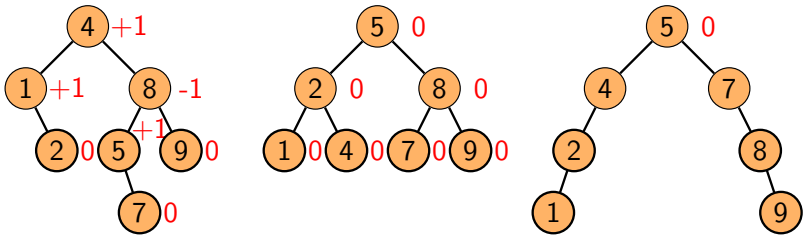
Árboles equilibrados: AVL

Factor de equilibrio de un nodo n. Dado un nodo n, y siendo a_i y a_d las alturas de sus subárboles h_i y h_d, se define FE $= a_d$ - a_i.

¿Qué valor tendrá el factor de equilibrio en los nodos de un árbol AVL?



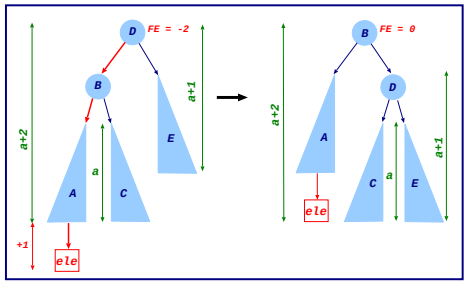
Factores de equilibrio en el ejemplo



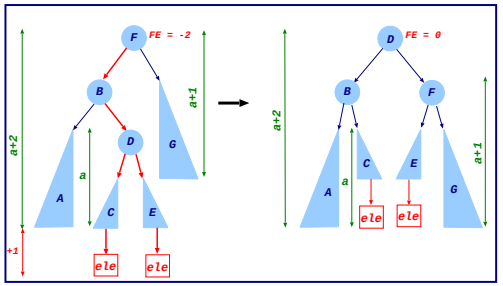
Inserciones en AVL

Cuando después de una inserción en una hoja de un AVL el factor de equilibrio es +2 o -2 se procede al reequilibrado: 4 tipos de rotaciones

rotación II

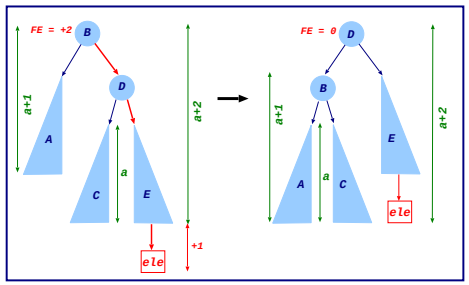


rotación ID

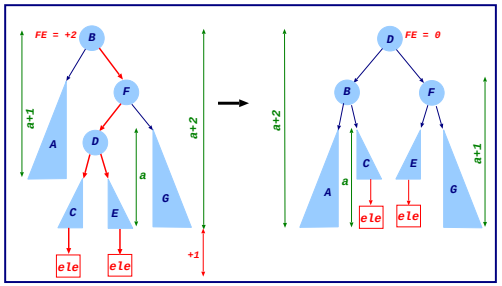


Inserciones en AVL

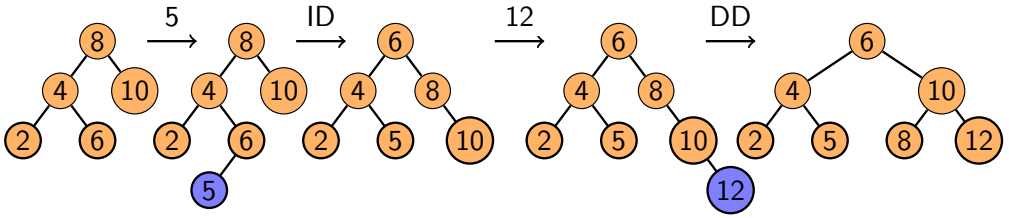
rotación DD



rotación DI



Ejemplo de inserción en AVL



Con los borrados habrá que reestructurar de la misma forma los árboles.

Características de los árboles AVL:

- la altura de los árboles es $\mathrm{O}(\log_2(n))$
- es preferible a otros si $|N_{busqueda}| >> |N_{insercion}|$ ó $|N_{borrado}|$
- al realizar una inserción se realiza una única rotación (como máximo)
- al realizar un borrado se realizan $\log_2(n)$ rotaciones (como máximo)
- empíricamente, se produce una rotación cada 2 inserciones o cada 5 borrados

Index

Montículos (heaps)

Son estructuras de datos de tipo árbol que mantienen información de un conjunto ordenado.

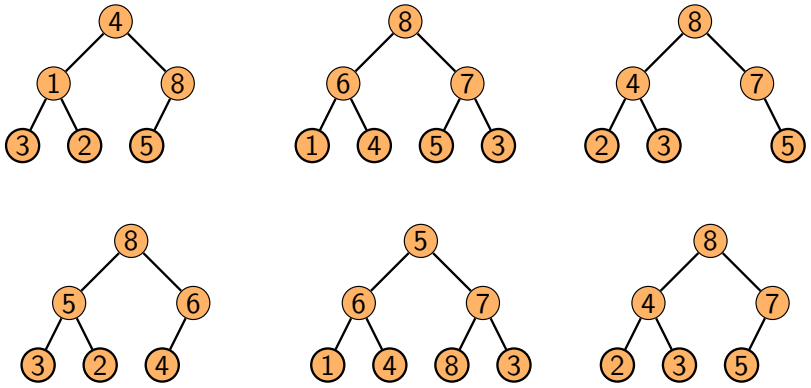
Se caracterizan porque

- en un **montículo máximo**:
  - el valor de cada nodo es mayor o igual a los valores almacenados en cada hijo (la raíz contiene el máximo)
- en un **montículo mínimo**:
  - el valor de cada nodo es menor o igual a los valores almacenados en cada hijo (la raíz contiene el mínimo)
- el árbol está perfectamente equilibrado y las hojas del último nivel se completan siempre de izquierda a derecha (arbol completo)

¿Cuál es el número de niveles en un montículo, siendo n el número de elementos?

Ejemplos

¿Cuáles de los siguientes árboles son montículos?



Montículos como colas de prioridad

Un montículo es una forma excelente de implementar una cola de prioridad.

En la vida real:

- sala de urgencias de un hospital

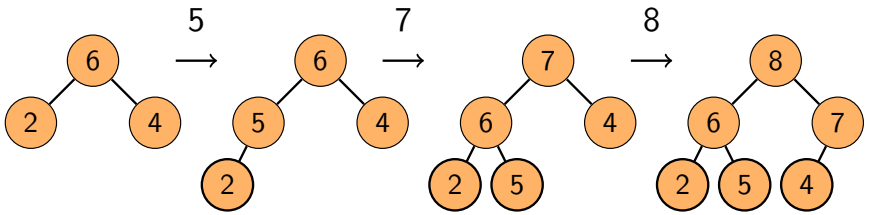
En informática:

- correo prioritario
- gestión de procesos en un sistema operativo
- implementación de algoritmos voraces
- algoritmos sobre grafos (caminos, árboles de expansión ...)

Inserción en una cola de prioridad máxima

El algoritmo para **insertar** un elemento **ele** en un montículo es muy sencillo:

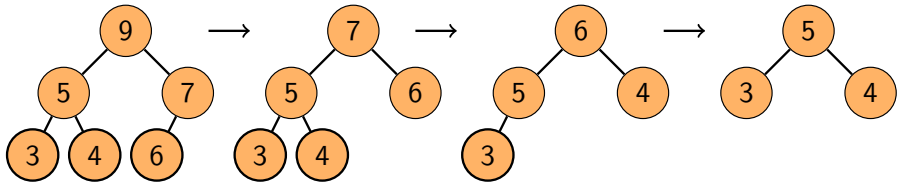
1. se pone **ele** al final del montículo
2. mientras (**ele** no está en la raíz) y (**ele** $>$ **padre(ele)**) hay que intercambiar **ele** con **padre(ele)**



Borrado en una cola de prioridad máxima

La **extracción** del primer elemento del montículo (máximo o mínimo) consiste en eliminar el elemento en la raíz.

1. se sustituye la raíz por el último elemento en el montículo (**last**)
2. mientras (**last** no es una hoja) y (**last** $<$ **hijo(ele)**) hay que intercambiar **ele** con **hijo(ele)** (el más grande)

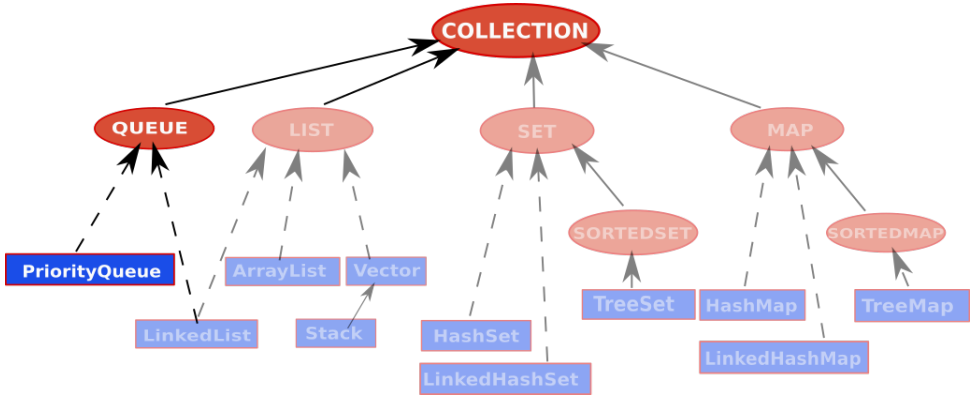


Implementación y coste de operaciones de colas de prioridad

- **Listas enlazadas**
  - coste de consultar el máximo (o mínimo)
  - coste de insertar un elemento
  - coste de borrar un elemento
- **Montículos**
  - coste de consultar el máximo (o mínimo)
  - coste de insertar un elemento
  - coste de borrar un elemento

Colas de prioridad en **java.util**

basada en el uso de un montículo con un array dinámico. El coste de las operaciones de inserción y borrado es $O(\log_2(n))$. La cola implementada es mínima. El coste de consultar el mínimo es constante.

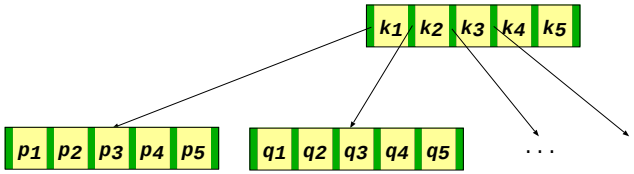


Index

Árbol multi-camino de búsqueda

Un árbol multicamino de búsqueda T es un árbol n-ario vacío o que cumple las siguientes propiedades:

- la raíz contiene $h_0, h_1 \dots h_{n-1}$ subárboles y $k_1 \dots k_{n-1}$ etiquetas
- se cumple que $k_i < k_{i+1}, 1 \leq i < n-1$
- todas las etiquetas del subárbol h_i son
  - menores que $k_{i+1}, 0 \leq i < n-1$
  - mayores que $k_i, 0 < i \leq n-1$
- los subárboles $h_i, 0 \leq i \leq n-1$ son también árboles AMB

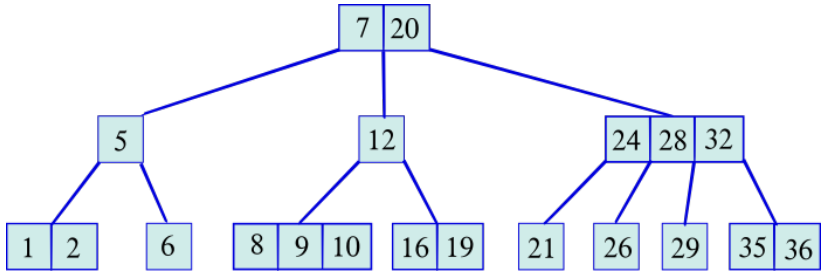


Index

Árbol 2-3-4

Un árbol 2-3-4 es un árbol vacío o cumple las siguientes propiedades:

- los nodos pueden tener 2, 3 ó 4 hijos (2-nodo, 3-nodo ó 4-nodo)
- cumple las propiedades de árbol multicamino de búsqueda
- todas las hojas están en el mismo nivel



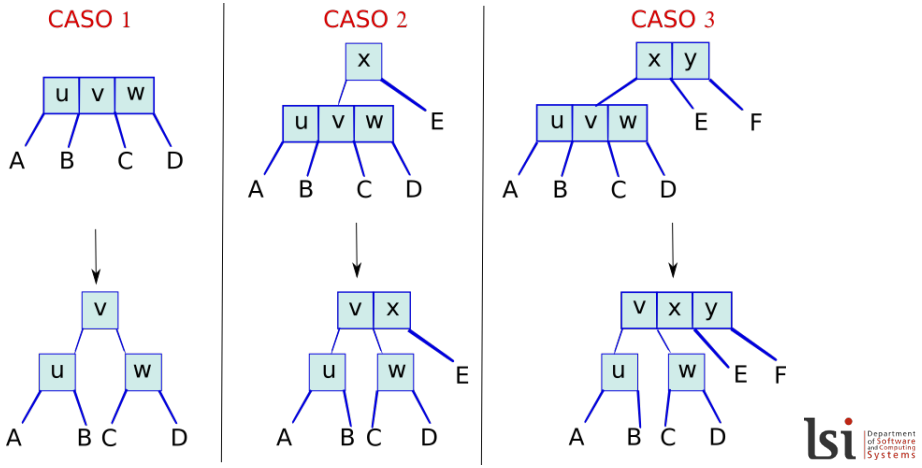
Árbol 2-3-4

- Operaciones básicas:
  - búsqueda, similar a los árboles multicamino de búsqueda
  - inserción, se realiza en las hojas (reestructuración asociada)
  - borrado, se realiza en las hojas (reestructuración asociada)
- Propiedades:
  - en un árbol 2-3-4 de altura h tenemos:
    - $2^h - 1$ elementos si todos los nodos son del tipo 2-nodo
    - $4^h - 1$ elementos si todos los nodos son del tipo 4-nodo

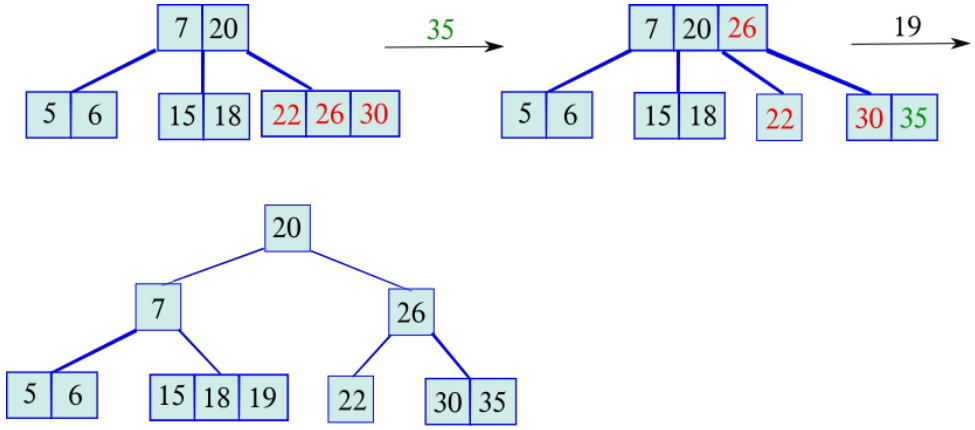
    por lo que la altura de un árbol 2-3-4 con n elementos se encuentra entre los límites: $\log_4(n+1)$ y $\log_2(n+1)$
  - las reestructuraciones se realizan desde la raíz hacia las hojas

Árbol 2-3-4: inserción

La inserción empieza **en la raíz** buscando iterativamente la hoja en la que debería insertarse el nodo (según el algoritmo de búsqueda de AMB). Cada vez que en esta búsqueda nos encontramos un 4-nodo hay que hacer reestructuración (3 posibilidades):



Ejemplo de inserción en árbol 2-3-4



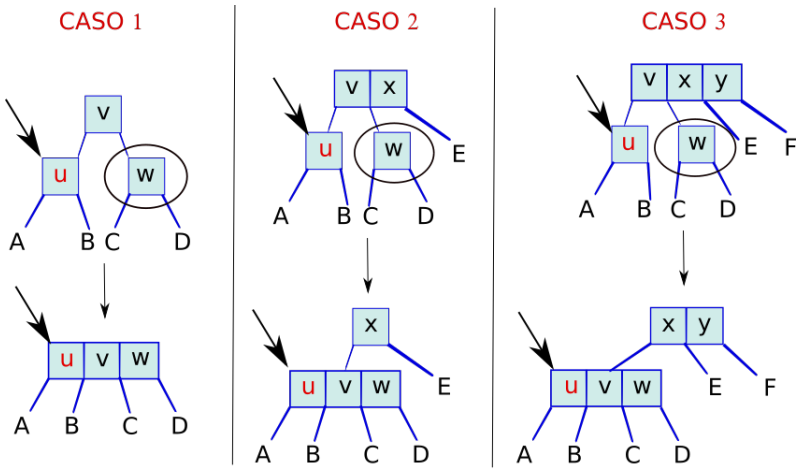
Árbol 2-3-4: borrado

El borrado se reduce a realizar un borrado en una hoja (como en los ABB):

1. si el elemento no está en una hoja se busca el anterior o siguiente (y se intercambia con él)*;
2. si el nodo por el que se pasa es 2-nodo hay que comprobar el grado de un nodo hermano:
   - **FUSIÓN:** si el nodo hermano también es un 2-nodo hay que fusionar los dos nodos con la etiqueta padre de ambos (el grado del nodo padre se decrementa, ver casos 2 y 3 y en el caso 1 decrece la altura del árbol);
   - **ROTACIÓN:** si el nodo hermando es 3-nodo o 4-nodo, sólo es necesario rotar los elementos implicados (ver casos 4 y 5);
3. el paso 2 se repite hasta llegar a las hojas.

*al igual que se hace en los árboles binarios de búsqueda

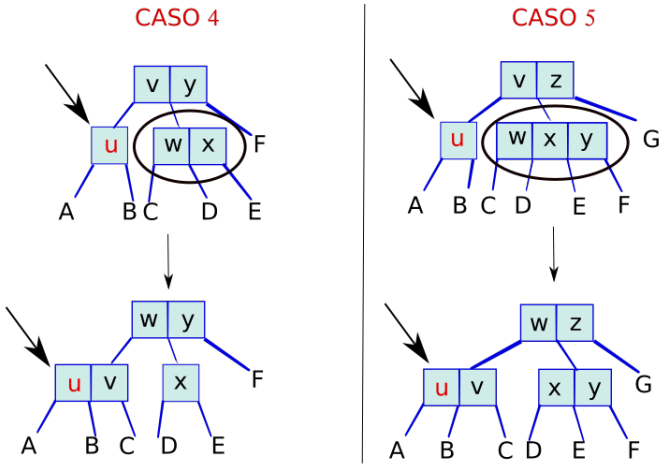
Borrado en 2-3-4: hermano de grado 2



la etiqueta 'u' está en un 2-nodo y su hermano es 2-nodo
FUSIÓN

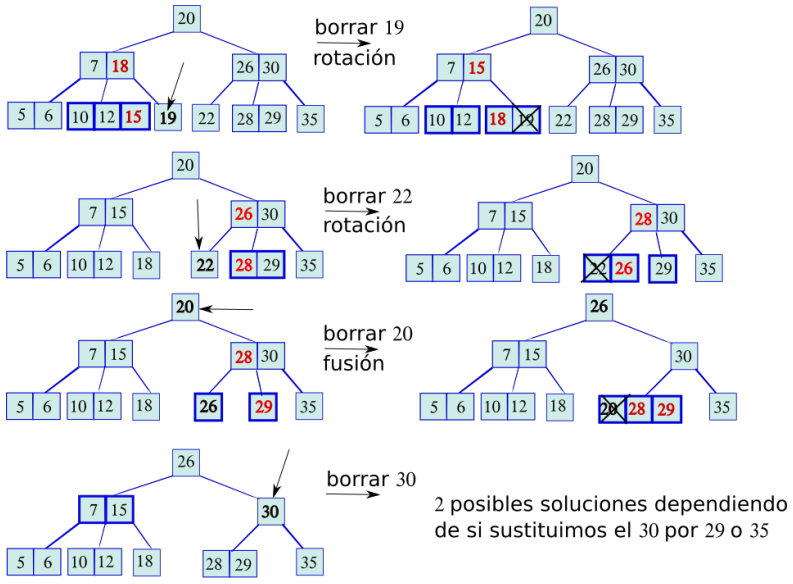
Borrado en 2-3-4: hermano de grado 3 o 4

Si el hermano es 3-nodo o 4-nodo, sólo es necesario rotar los elementos implicados.



la etiqueta 'u' está en un 2-nodo y su hermano en 3-nodo o 4-nodo
ROTACIÓN

Ejemplo de borrado en árbol 2-3-4



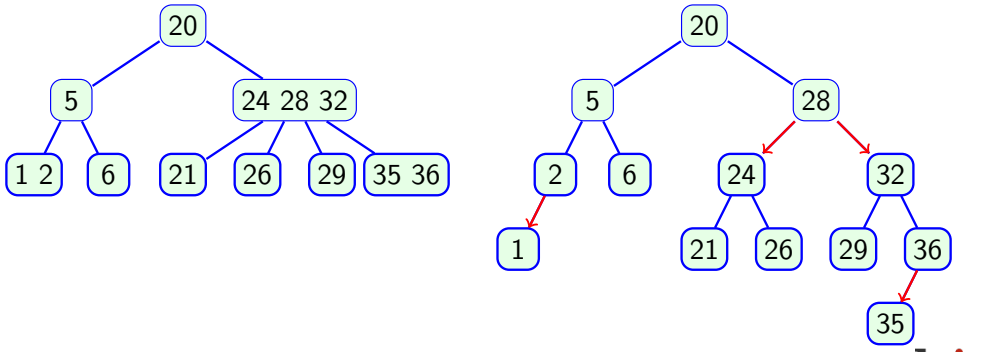
2 posibles soluciones dependiendo
de si sustituimos el 30 por 29 o 35

Index

Árbol rojo-negro

Son una representación en árbol binario de un árbol 2-3-4. Los hijos de un nodo en un árbol rojo-negro son de dos tipos: rojo y negro.

- si el hijo ya existía como tal en un árbol 2-3-4 original, entonces será negro
- si el hijo no existía en un árbol 2-3-4, entonces será rojo



Árbol rojo-negro

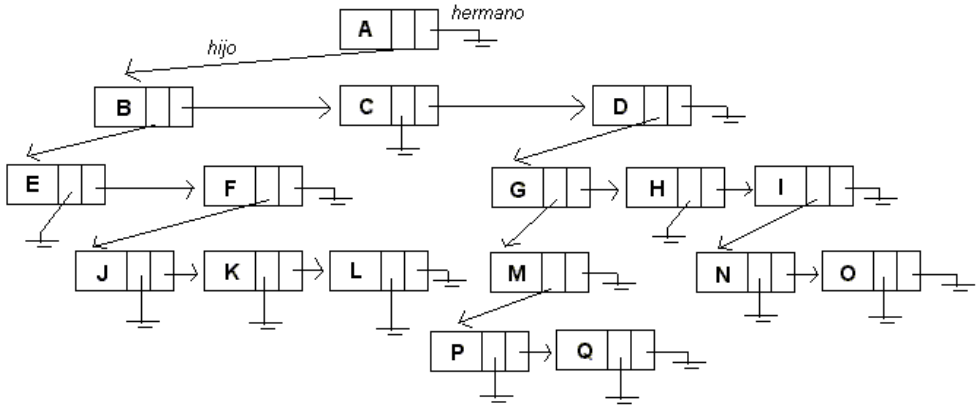
Propiedades:

- un árbol rojo-negro es un árbol binario de búsqueda
- la raíz es siempre negra
- cada camino desde la raíz hasta las hojas tiene el mismo número de hijos negros
- ningún camino desde la raíz tiene dos o más hijos rojos consecutivos
- el camino más largo desde la raíz hasta una hoja no es más largo que 2 veces el camino más corto desde la raíz del árbol a una hoja en dicho árbol
- pueden buscar, insertar y borrar en un tiempo $\mathrm{O}(\log_2(n))$, donde n es el número de elementos del árbol.

Uso:

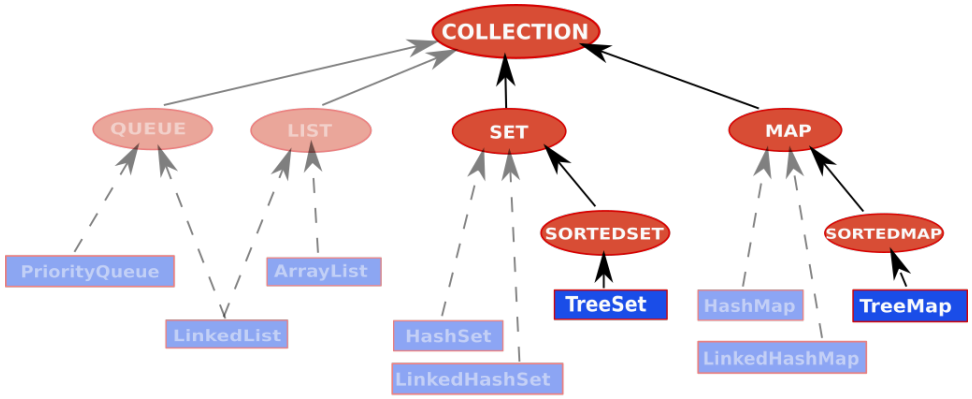
- muchas estructuras de datos usadas en geometría computacional pueden basarse en árboles rojo-negro
- se utilizan como estructura de datos interna en `java.util`

¿Cómo representamos un árbol general?



Index

Árboles en java.util



TreeSet y TreeMap

- **TreeSet** es una implementación para el tipo de datos "conjunto". Están implementados usando un **árbol rojo-negro**, por lo que la inserción y el borrado se realiza en tiempo $\mathrm{O}(\log_2(n))$.
- **TreeMap** es una implementción para los mapas (generalización de arrays, donde los arrays se pueden indexar con cadenas u otros tipos de datos). Están implementados usando un **árbol rojo-negro**.

Profundizaremos en su funcionamiento en la segunda parte del tema.