Estructures de dades i algorismes

Tema 4

Arbres, Arbre Binari i Arbre Binari de Cerca

Curs 2018-2019

Objectius

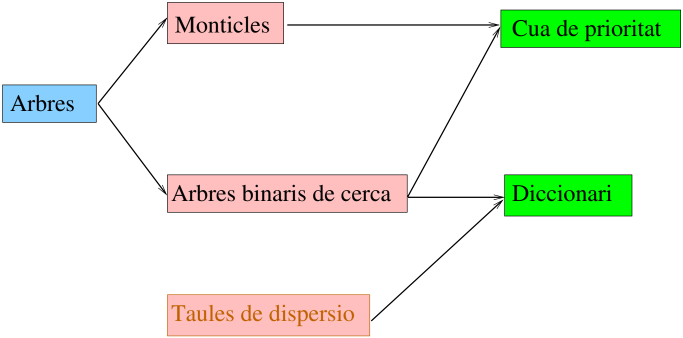
- Aprendre els conceptes bàsics d'arbres
- Aprendre els conceptes bàsics d'arbre binaris
- Aprendre els conceptes bàsics d'arbres binaris de cerca.
- Aprendre els conceptes de recorreguts d'arbres.
- Aprendre les operacions bàsiques sobre arbres binaris de cerca
- Conèixer els arbres equilibrats.

Continguts

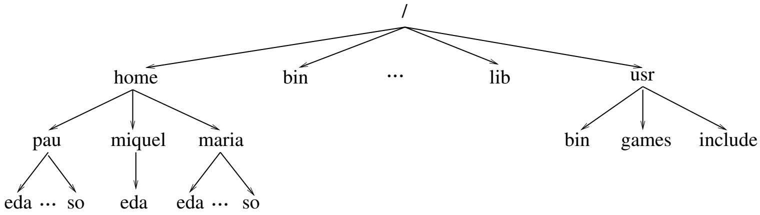
1. Conceptes d'arbres.
2. Arbres generals: representació.
3. Arbres binaris: propietats i representació.
4. Recorreguts d'arbres binaris.
5. Arbres binaris de cerca: representació i operacions bàsiques.
6. Arbres equilibrats.

5.1 Concepes d'arbres

- Relació entre tipus abstractes de dades i representacions



- Exemple de representació jeràrquica d'una col·lecció d'elements



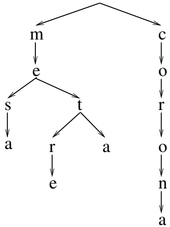
5.1 Conceptes d'arbres

➢ L'estructura en arbre:

- Relacions jeràrquiques entre els elements d'una col·lecció
- Cerques en temps sublineals

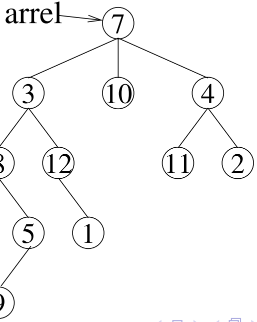
  Exemple: representar el conjunt de paraules
  {*mes*, *mesa*, *meta*, *metre*, *cor*, *corona*}

  - Alternativa 1: usant un vector de cadenes
  - Alternativa 2: usant un arbre

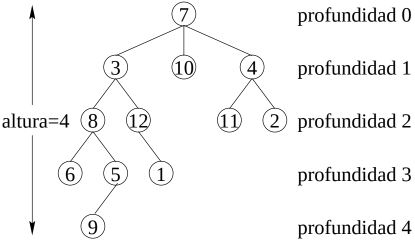


5.1 Conceptes d'arbres

- Un arbre es una estructura jeràrquica que es defineix com:
  - Un conjunt finit de nodes. Un dels nodes es distingix com arrel.
  - Una relació entre nodes representada en forma d'arestes de manera que:
    - Hi ha una aresta de P fins S si el node P és pare del node S. Es diu que S és fill del node P.
    - Cada node té un únic pare excepte l'arrel que no té pare.



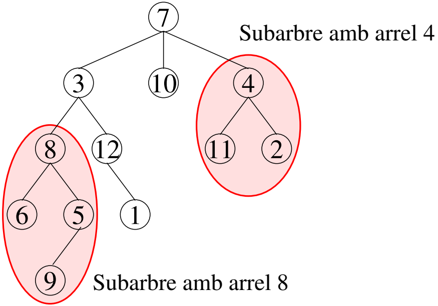
- Un node sense fills s'anomena fulla. La resta s'anomenen nodes interns.
- El grau d'un node és el nombre de fills que té.
- S'anomena profunditat d'un node a la longitud del camí des de l'arrel fins eixe node.
- L'alçària d'un arbre és la profunditat del node més profund.



5.1 Conceptes d'arbres

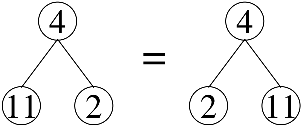
➢ Definició recursiva d'un arbre:
- Un conjunt buit és un arbre.
- Un node arrel P i zero o més subarbres no buits on les seues arrels son fills de P és un arbre.

➢ Cada node d'un arbre defineix un subarbre format per aquell del qual és arrel.

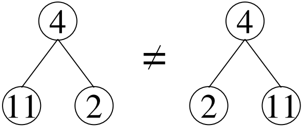


5.1 Conceptes d'arbres

- Arbre no ordenat: l'ordre dels fills no és rellevant.

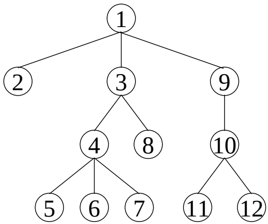


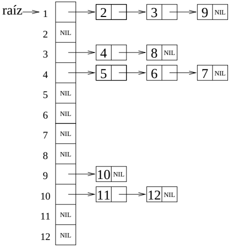
- Arbre ordenat: l'ordre dels fills és rellevant.



5.2 Arbres generals: representació

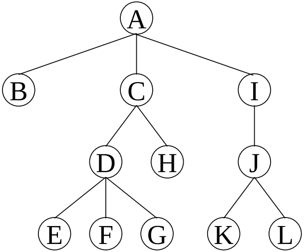
- Arbres generals: nombre de fills sense acotar.
  - Llistes (ordenades) de fills.
  - Fill més a l'esquerra-germà dret: variables dinàmiques o vectors.
  - Amb vectors i apuntant al pare (*mf-sets*).
  - Altres . . .
- Llistes (ordenades) de fills



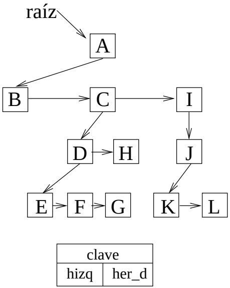


5.2 Arbres generals: representació

➢ Fill més a l'esquerra-germà dret: variables dinàmiques o vectors.

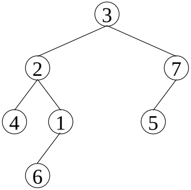


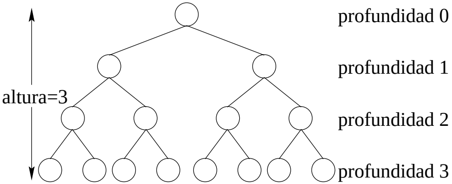
	hizq	clave	her_d
2	3	C	10
3	17	D	9
raíz → 4	8	A	NIL
....			
8	NIL	B	2
9	NIL	H	NIL
10	13	I	NIL
....			
12	NIL	G	NIL
13	14	J	NIL
14	NIL	K	16
15	NIL	F	12
16	NIL	L	7
17	NIL	E	15
....			



5.3 Arbres binaris

➢ Un arbre binari és un arbre en el que el màxim nombre de fills de cada node és 2: fill esquerrà i fill dret.

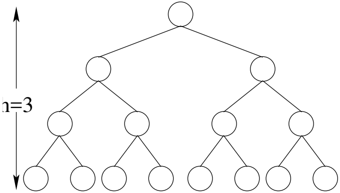




- Un arbre binari es diu que és complet si totes les fulles tenen la mateixa profunditat i tots els nodes interns tenen grau 2.
- Un arbre binari és quasi-complet si l'arbre és complet, a excepció potser en el nivell de les fulles, en el qual totes les fulles estan tan a l'esquerra com siga possible.

5.3 Arbres binaris

- Propietat:



profundidad	num. nodos
0	1=2^0
1	2=2^1
2	4=2^2
3	8=2^3

- Nodes de profunditat $i = 2^i$.
- fulles: 2^h
- nodes interns: $2^0 + 2^1 + \cdots + 2^{h-1} = \sum_{i=0}^{h-1} 2^i = 2^h - 1$

$$\boxed{\text{nodes total: } n = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1}$$

- ...i l'alçària

$$\boxed{n = 2^{h+1} - 1 \qquad \rightarrow \qquad h = \lfloor log_2 n \rfloor}$$

5.3 Arbres binaris

- Propietats:
  - El màxim nombre de nodes de profunditat i és 2^i.
  - El màxim nombre de nodes en un arbre binari d'alçària h és $2^{h+1} - 1$. El màxim nombre de nodes interns és $2^h - 1$. El màxim nombre de fulles és 2^h.
  - L'alçària mínima d'un arbre binari amb n nodes és $\lfloor log_2 n \rfloor$.
  - Per a qualsevol arbre binari no buit, si n_0 és el nombre de fulles i n_2 és el nombre de nodes de grau 2, aleshores $n_0 = n_2 + 1$.

5.3 Arbres binaris

➢ Representació d'arbre binari mitjançant variables dinàmiques.
  - Cada node és un objecte de la classe NodeBinTree.

```
package librerias.estructurasDeDatos.jerarquicos;
public class NodeABC<E> {
  private E e;
  NodeABC<E> esq, dre;
    public NodeABC(E e) {
        this(e, null, null);
    }
    public NodeABC(E e, NodeABC<E> esq, NodeABC<E> dre) {
        this.e = e;
        this.esq = esq;
        this.dre = dre;
    }
    public E valor() {return e;}
}
```

5.3 Arbres binaris

```
package librerias.estructurasDeDatos.jerarquicos;

import librerias.estructurasDeDatos.modelos.*;

public class ABCDinamic<E extends Comparable<E>>
implements ArbreBinariDeCerca<E> {
    protected NodeABC<E> arrel;
    protected int talla;

    public ABCDinamic() {
        arrel = null;
        talla = 0;
    }
```

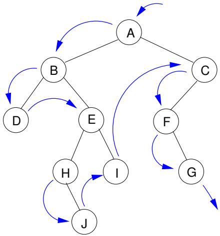
5.4 Recorreguts d'arbres binaris

➢ Problema: donat una arbre binari, es desitja visitar tots els nodes de l'arbre per a realitzar una determinada operació.

- Preordre:
  1. visitar el node
  2. visitar el fill esquerre en preordre
  3. visitar el fill dret en preordre
- Inordre:
  1. visitar el fill esquerre en inordre
  2. visitar el node
  3. visitar el fill dret en inordre
- Postordre:
  1. visitar el fill esquerre en postordre
  2. visitar el fill dret en postordre
  3. visitar el node

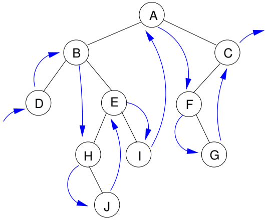
5.4 Recorreguts d'arbres binaris

- Recorregut preordre: l'arrel es processa abans que els subarbres.




```java
private String preOrdre(NodeABC<E> r) {
    if (r==null) return "";
    else return r.valor().toString() + "\n" +
       preOrdre(r.esq) + preOrdre(r.dre);
}
public String preOrdre() {
    if (arrel == null) return "";
    else return preOrdre(arrel);
}
```

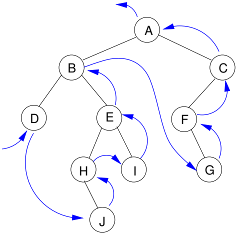
- Recorregut inordre: l'arrel es processa entre el subarbre esquerre i dret.




```java
private String inOrdre(NodeABC<E> r) {
    if (r==null) return "";
    else return inOrdre(r.esq) +
       r.valor().toString() + "\n" + inOrdre(r.dre);
}
public String inOrdre() {
    if (arrel == null) return "";
    else return inOrdre(arrel);
}
```

5.4 Recorreguts d'arbres binaris

- Recorregut postordre: l'arrel es processa després dels subarbres.




```
private String postOrdre(NodeABC<E> r) {
    if (r==null) return "";
    else return postOrdre(r.esq) +
       postOrdre(r.dre) + r.valor().toString() + "\n";
}
public String postOrdre() {
    if (arrel == null) return "";
    else return postOrdre(arrel);
}
```

5.5 Arbre binaris de cerca

- Estructura de dades molt versàtil, serveix per a la implementació de diccionaris i de cues de prioritat.
- És una generalització de la cerca dicotòmica. Suporta eficientment les operacions de cerca, localitzar el mínim, el màxim, el predecessor i el successor. També suporta eficientment les operacions d'inserció i esborrat.
- Un arbre binari és de cerca si:
  - Tots el elements del subarbre esquerre són menors que el valor de l'arrel.
  - Tots el elements del subarbre dret són majors que el valor de l'arrel.
- **Propietat:** Si es llisten els elements segons un recorregut inordre resulta una seqüència ordenada.

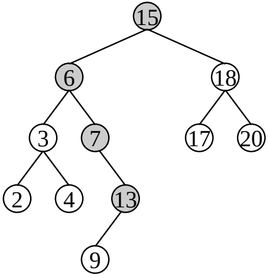
5.5 Arbre binaris de cerca

```java
package librerias.estructurasDeDatos.modelos;
public interface ArbreBinariDeCerca<E extends Comparable <E>> {
    void inserir(E e);
    void esborrar(E e);
    boolean cercar(E e);
    boolean esBuit();
    String preOrdre();
    String inOrdre();
    String postOrdre();
    E max();
    E min();
    E successor(E e);
    //E predecessor(E e);
}
```

5.5 Cerca d'una clau en un ABC

```java
public boolean cercar(E e) {
    if (arrel == null)
        return false;
    else {
        NodeABC<E> r = arrel;
        while (r!=null &&
               e.compareTo(r.valor()) != 0)
          if (e.compareTo(r.valor()) < 0)
            r = r.esq;
          else
            r = r.dre;
        return r!=null;
    }
};
```

Exemple: Cerca del valor 13



Cost $O(h)$, on h és l'alçària de l'arbre.

Exercicis

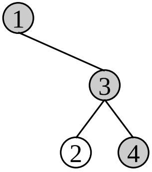
Exercici. Suposem que tinguem el nombre de l'1 fins el 1000 en un ABC i volem cercar el 363. Quina de les següents seqüències de nodes no pot ser la seqüència de nodes examinada? Escriu un programa que responga a la pregunta.

- 2, 252, 401, 398, 330, 344, 397, 363
- 924, 220, 911, 244, 898, 258, 362, 363
- 925, 202, 911, 240, 912, 245, 363
- 2, 399, 387, 219, 266, 382, 381, 278, 363
- 935, 278, 347, 621, 299, 392, 358, 363

Exercicis

Exercici. Demostra que la següent propietat no és certa: Suposem que la cerca d'un element de clau k en un ABC acaba en una fulla. Considera tres conjunts: A, les claus a l'esquerre del camí de cerca; B, les claus del camí de cerca; i C, les claus a la dreta del camí de cerca. Troba un contraexemple per a demostrar que no s'acompleix que per a qualsevol $a \in A$, $b \in B$ i $c \in C$, $a \leq b \leq c$.

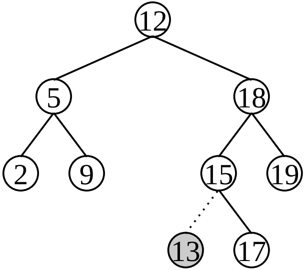
Solució falsa propietat



$$A = \{2\};\ B = \{1, 3, 4\};\ C = \{\} \Rightarrow 2 > 1$$

5.5 Inserir en un ABC

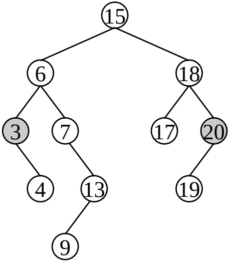
```java
public void insertar(E e) {
    if (arrel == null) {
        arrel = new NodeABC<E>(e);
        talla++; }
    else {
        NodeABC<E> r = arrel, ant=null;
        while (r!=null  &&
               e.compareTo(r.valor()) != 0) {
            ant = r;
            if (e.compareTo(r.valor()) < 0)
          r = r.esq;
            else
          r = r.dre;
        }
        if (r == null) {
            if (e.compareTo(ant.valor()) < 0)
          ant.esq = new NodeABC<E>(e);
            else
          ant.dre = new NodeABC<E>(e);
            talla++:
        }
    }
};
```



5.5 Mínim i màxim en un ABC

- Element mínim en un ABC: node que no té cap fill a l'esquerre i no pertany a cap subarbre dret de cap node.

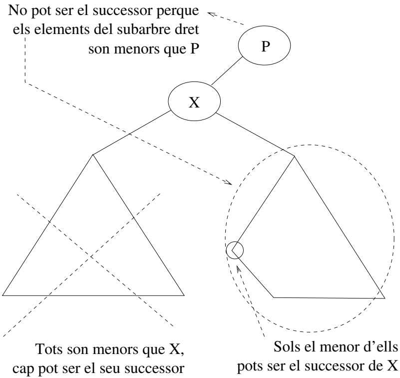
```
public E min() {
    if (arrel==null)
        return null;
    NodeABC<E> r = arrel;
    while (r.esq != null)
        r = r.esq;
    return r.valor();
}
```

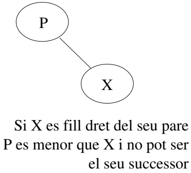


- Element màxim: Cas simètric substituint esq per dret.

Cost $O(h)$.

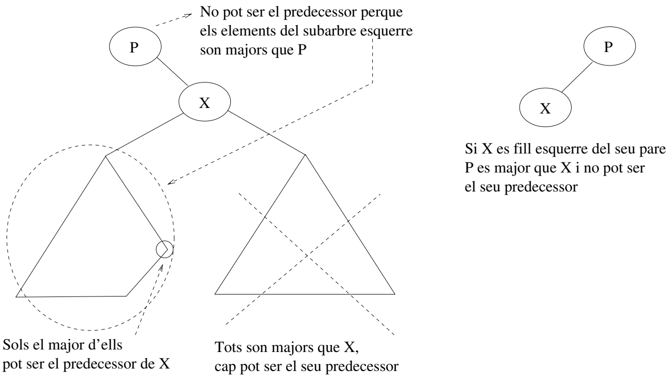
Successor de X: cas fàcil quan hi ha fill dret.





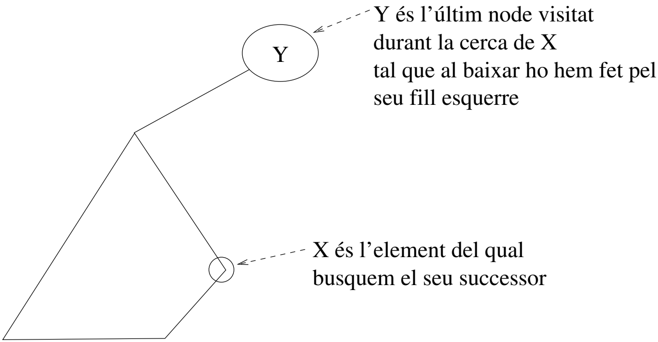
5.5 Successor i predecessor en un ABC

Predecessor de X: cas fàcil quan hi ha fill esquerre.



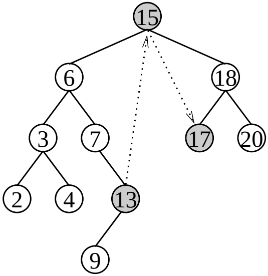
Successor de X: cas menys fàcil quan NO hi ha fill dret.

NOTA: Y és successor de X $\leftrightarrow$ X és predecessor de Y



5.5 Successor i predecessor en un ABC

- Successor en *inordre* de x: node amb menor clau dels majors que la clau de x
- Successor del 15: mínim element en el subarbre dret del node $\rightarrow$ 17
- Successor del 13: no té subarbre dret, amb la qual cosa el seu successor és el menor antecessor el fill esquerre del qual siga també un antecessor $\rightarrow$ 15



2, 3, 4, 6, 7, 9, 13, 15, 17, 18, 20

5.5 Successor i predecessor en un ABC

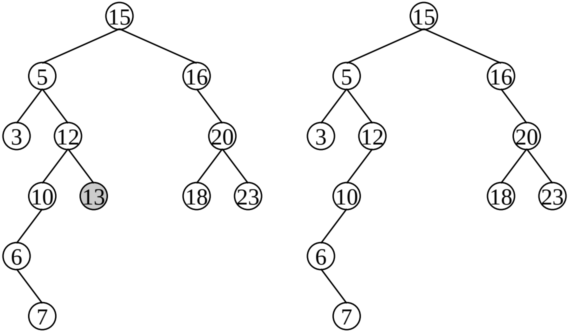
```
public E successor(E e) {
    NodeABC<E> r = arrel, qmin = null;
    while (r != null && r.valor() != e)
        if (e.compareTo(r.valor()) < 0) {
            qmin = r;
            r = r.esq;
        }
        else
            r = r.dre;
    if (r != null && r.dre != null) {
        r = r.dre;
        while (r.esq != null)
            r = r.esq;
        r.valor();
    }
    return qmin.valor();
}
```

Cost $O(h)$ **Exercici.** Escriu l'algorisme Predecessor.

5.5 Esborrar en un ABC

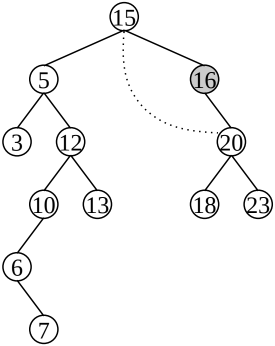
Cas 1. L'element és una fulla: s'esborra i no es fa res més. .

Exemple: esborrem el 13.



Cas 2. L'element té un fill: s'esborra i el seu fill ocupa la seua posició.

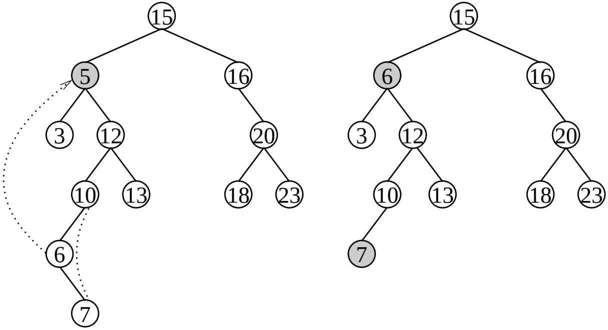
Exemple: esborrem el 16.



5.5 Esborrar en un ABC

Cas 3. L'element té dos fills: s'esborra i el seu lloc és ocupat pel menor fill del subarbre dret (o el major fill del subarbre esquerre).

Exemple: esborrem el 5.



5.5 Esborrar en un ABC

```
public void borrar(E e) {
    NodeABC<E> ant = null, r = arrel;
    while (r != null &&  e.compareTo(r.valor()) != 0) {
        ant = r;
        if (e.compareTo(r.valor()) < 0)
            r = r.esq;
        else
            r = r.dre;
    }
    if (r == null)                                      // Element no trobat
        return;
    if (r.esq == null && r.dre == null) {                     // No te fills
        if (ant == null)                   // L'element a eliminar es l'arrel
            arrel = null;
        else
            if (e.compareTo(ant.valor()) < 0)
                ant.esq = null; // El node a eliminar es fill esquerre
            else
          ant.dre = null;     // El node a eliminar es fill dret
        return;
    }
```

5.5 Esborrar en un ABC

```
if (r.esq != null && r.dre != null) {              // Te dos fills
    NodeABC<E> min = r.dre, ant_min = r;
                                 // Localitzem el minim del fill dret
    while (min.esq != null ) {ant_min = min; min=min.esq;}
    if (ant_min == r)            // El minim es l'arrel del fill dret
        min.esq = r.esq;
    else {
        ant_min.esq = min.dre;
        min.dre = r.dre;
        min.esq = r.esq;
    }
    if (ant == null)               // L'element a eliminar es l'arrel
        arrel = min;
    else
        if (e.compareTo(ant.valor()) < 0)
      ant.esq = min;  // El node a eliminar es fill esquerre
        else
      ant.dre = min;      // El node a eliminar es fill dret
    return;
}
```

5.5 Esborrar en un ABC

```
    if (ant == null)          // Sols hi ha un fill i eliminem l'arrel
        if (r.dre != null)
            arrel = r.dre;
        else
            arrel = r.esq;
    else
        if (r.esq == null)                        // Sols hi ha un fill
            if (e.compareTo(ant.valor()) < 0)
          ant.esq = r.dre;
            else
          ant.dre = r.dre;
        else
            if (e.compareTo(ant.valor()) < 0)
          ant.esq = r.esq;
            else
          ant.dre = r.esq;
    return;
}
```

5.5 Cost de les operacions amb ABC

Cost operacions: $O(h)$, essent h l'alçària de l'ABC ($h \in O(logn)$; $h \in O(n)$)

L'alçària h està entre $O(logn)$ i $\Omega(n)$, on n és el número d'elements.

S'inserim dades **aleatòriament** s'aconsegueix una alçària $O(logn)$. Existeixen estructures de dades basades en arbre que inclouen informació i/o operacions addicionals per a aconseguir un equilibri en els arbres. Són els **arbres balancejats** i es solen basar en:

- Efectuar "rotacions". S'intercanvien nodes i subarbres en un arbre binari de cerca para aconseguir un altre equivalent (Exemple: arbres AVL, arbres roig-negre).
- Efectuar una "reconstrucció parcial". Un recorregut en *inordre* d'un ABC produeix una seqüència ordenada que permet un arbre perfectament balancejat amb un cost lineal.