

# Estructures de dades arborescents

R. Ferrer i Cancho

Universitat Politècnica de Catalunya

PRO2 (curs 2017-2018)  
Versió 0.4

Avís: aquesta presentació no pretén ser un substitut dels apunts  
oficials de l'assignatura.

# On som?

- ▶ Tema 3: Estructures de dades arborescents.
- ▶ 4a sessió

## Avui

- ▶ Mes enllà d'estructures de dades lineals (piles, cues, llistes)
- ▶ Estructures de dades arborescents (diferents menes d'arbres).

Arbres generals

Abres  $N$ -aris

Arbres binaris (Arbre)

Ús d'arbres binaris: operacions de cerca i recorregut

- Alçada d'un arbre

- Cerca d'un valor `int` en un arbre de `int`

- Modificació i generació d'arbres

- Recorreguts típics d'arbres

  - Recorreguts en profunditat

  - Recorreguts en amplada

# Estructures de dades arborescents

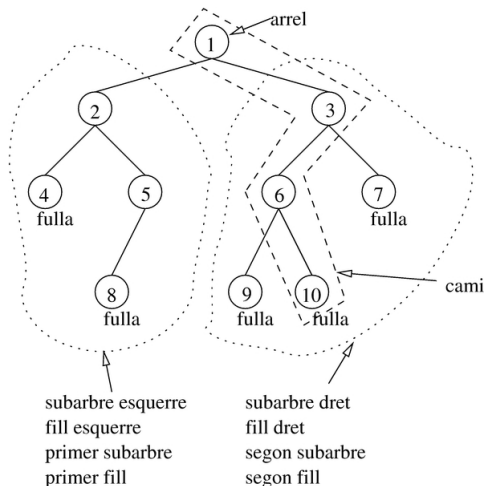
Estructures de dades lineals enfront arborescents:

- ▶ Estructura de dades lineal: un element només pot tenir (pel cap alt) un successor
- ▶ Estructura de dades arborescent: un element pot tenir pot tenir més d'un element successor.

Metàfora de l'arbre (arrel, fulles, alçada).

Diferència amb arbres reals: arrel a dalt / fulles a sota

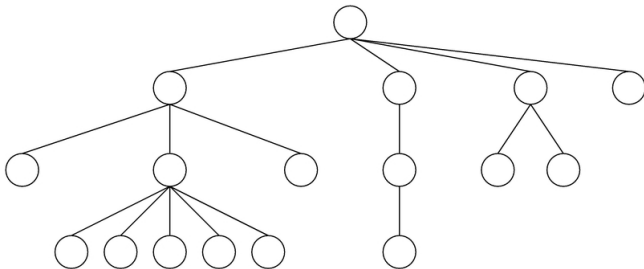
# Conceptes d'arbres I



## Conceptes d'arbres II

- ▶ Node: element d'un arbre
- ▶ Arbre buit: arbre que no té cap element
- ▶ Arrel:
  - ▶ element de tot arbre no buit
  - ▶ únic consultable individualment
  - ▶ node arrel està connectat a zero o més (sub)arbres.
- ▶ Fill: subarbre (consultable individualment).
- ▶ Fulles: nodes sense cap successor
- ▶ Camí: successió de nodes que van de l'arrel a una fulla.
- ▶ Alçada: longitud del camí més llarg.

# Exemple d'un arbre general

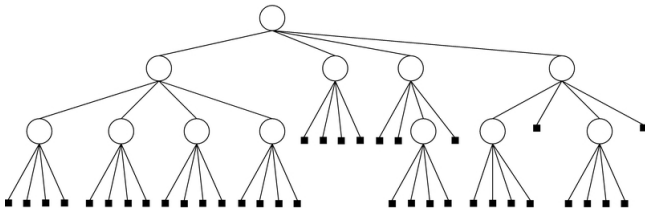


# Arbres $N$ -aris

- ▶ Tipus d'arbre on tots els subarbres no buits tenen exactament el mateix nombre de fills, siguin aquests fills arbres buits o no.
- ▶  $N$ : nombre de fills



## Exemple d'arbre 4-ari

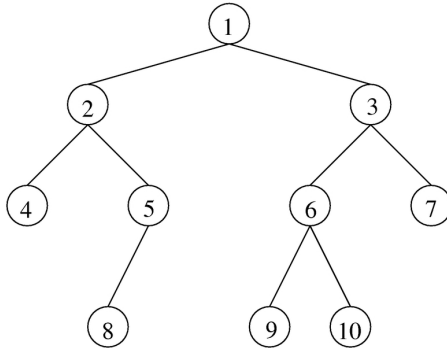


- ▶ Quadrats negres: arbres buits.
- ▶ Per claredat convé representar explícitament els arbres buits en els arbres  $N$ -aris.

# Arbres binaris

- ▶ Cas particular dels arbre  $N$ -aris, on  $N = 2$ .
- ▶ Quan diem arbres sense detallar més, ens referim per defecte a arbres binaris.

# Exemple d'un arbre binari I



No hem dibuixat els subarbres buits amb caixes negres com en l'exemple d'arbre 4-ari

## Exemple d'un arbre binari II

- ▶ Arrel de l'arbre: 1.
- ▶ L'arrel té dos fills:
  - ▶ Subarbre esquerre: té com a arrel 2
  - ▶ Subarbre dret: té com a arrel 3.
- ▶ Tots els nodes tenen dos fills.
- ▶ Si els dos fills són arbres buits, els nodes es diuen fulles.
- ▶ Els nodes que contenen 4, 8, 9, 10 i 7 són fulles.
- ▶ Els arbres buits no estan representats.

# Especificació de la classe genèrica arbre binari (BinTree)

Veure document:

*especificacions\_\_estructures\_\_dades\_\_arborescents.pdf*

Instanciació: `BinTree<tipus> nom_arbre;`

► Exemples:

- `BinTree<int> a;`
- `BinTree<Estudiant> b;.`

Atenció: coherència entre el valor de  $T$  a l'arbre del p.i. i el dels paràmetres explícits.

`BinTree(const T& x, const BinTree& left, const BinTree& right);`

# Ús d'arbres binaris

Cassificació en tipus d'algorismes (vista en sessions anterios):

- ▶ *algorismes d'exploració*
- ▶ *algorismes de modificació*
- ▶ *algorismes de creació o generació.*

# Alçada d'un arbre I

- ▶ Exploració d'un arbre.
- ▶ Només es calcula una propietat de la estructura de l'arbre,
- ▶ No es fa servir l'operació `arrel` (no es consulta en cap moment el contingut dels nodes).
- ▶ Especificació

```
int altura(const BinTree<int>& a)
/* Pre: cert */
{
    ...
}
/* Post: El resultat és la longitud del camí més llarg de l'arrel
de l'arbre a */
```

- ▶ Versió recursiva.

## Alçada d'un arbre II

```
int altura(const BinTree<int>& a)
/* Pre: cert */
{
    int x;
    if (a.empty()) x=0;
    else {
        int y = altura(a.left());
        int z = altura(a.right());
        if (y >= z) x = y + 1;
        else x = z + 1;
    }
    return x;
}
/* Post: El resultat és la longitud del camí més llarg de l'arrel a una fulla
de l'arbre a */
```

Què li ha passat al paràmetre "a"?



## Exercici: mida d'un arbre

- ▶ 

```
int mida(const BinTree<int>& a)
/* Pre: cert */
{
    ...
}
/* Post: El resultat és el nombre de nodes de l'arbre a */
```
- ▶ Versió recursiva

## Mida d'un arbre: solució

```
int mida(const BinTree<int>& a)
/* Pre: cert */
{
    int x;
    if (a.empty()) x=0;
    else x= mida(a.left()) + mida(a.right()) + 1;
    return x;
}
/* Post: El resultat és el nombre de nodes de l'arbre a */
```

# Cerca d'un valor int en un arbre de int

Exploració però

- ▶ No cal visitar tots els nodes de l'arbre
- ▶ Cal consultar el contingut dels nodes que visitem.
- ▶ 

```
bool cerca(const BinTree<int>& a, int x)
/* Pre: cert */
{
    ...
}
/* Post: El resultat indica si x és a l'arbre a o no */
```
- ▶ Versió recursiva

# Solució

```
bool cerca(const BinTree<int>& a, int x)
/* Pre: cert */
{
    bool b;

    if (a.empty()) b=false;
    else if (a.value() == x) b=true;
    else {
        b = cerca(a.left(), x);
        if (not b) b = cerca(a.right(), x);
    }
    return b;
}
/* Post: El resultat indica si x és a l'arbre a o no */
```

# Suma d'un valor $k$ a tots els nodes d'un arbre d'`int`.

Dues solucions recursives:

- ▶ Versió acció.
- ▶ Versió funció.

# Versió acció I

- ▶ En el cas base, quan l'arbre és buit, l'arbre modificat és ell mateix i no cal fer res.
- ▶ Per tant, només hi ha el cas recursiu (explícitament en el codi).
- ▶ Especificació

```
void suma(BinTree<int> &a, int k)
/* Pre: a = A */
{
    ...
}
/* Post: El valor de cada node d'a és la suma del valor del
        node corresponent d'A i el valor k */
```

## Versió acció II

```
void suma(BinTree<int> &a, int k)
/* Pre: a = A */
{
    if (not a.empty()) {
        BinTree<int> a1 = a.left();
        BinTree<int> a2 = a.right();
        suma(a1, k);
        suma(a2, k);
        a = BinTree<int>(a.value() + k, a1, a2);
    }
}
/* Post: El valor de cada node d'a és la suma del valor del
        node corresponent d'A i el valor k */
```

# Versió funció I

- ▶ L'arbre resultat s'ha d'anar generant.
- ▶ Els arbres acabats de crear són buits → no ens cal un cas base explícit.
- ▶ Especificació:

```
BinTree<int> suma(const BinTree<int> &a, int k)
/* Pre: cert */
{
    ...
}
/* Post: El valor de cada node del resultat és la suma del valor
        del node corresponent d'a i el valor k */
```



## Versió funció II

```
BinTree<int> suma(const BinTree<int> &a, int k)
/* Pre: cert */
{
    if (a.empty()) return BinTree<int>();
    else {
        return BinTree<int>(a.value() + k,
                           suma(a.left(), k),
                           suma(a.right(), k)
                           );
    }
}
/* Post: El valor de cada node del resultat és la suma del valor
        del node corresponent d'a i el valor k */
```

# Recorreguts d'arbres

Mètodes més habituals per visitar els nodes d'un arbre (per fer recorreguts o cerques).

- ▶ Recorregut en profunditat.
  - ▶ En preordre.
  - ▶ En inordre.
  - ▶ En postordre.
- ▶ Recorregut en amplada.

Arbre buit: no fer res.

## Recorreguts en profunditat: preordre

1. visitar l'arrel
2. recórrer l'arbre esquerre (en preordre)
3. recórrer l'arbre dret (en preordre)

Exemple: 1, 2, 4, 5, 8, 3, 6, 9, 10 i 7.

# Recorreguts en profunditat: inordre

1. recórrer l'arbre esquerre (en inordre)
2. visitar l'arrel
3. recórrer l'arbre dret (en inordre)

Exemple: 4, 2, 8, 5, 1, 9, 6, 10, 3, i 7.

## Recorreguts en profunditat: postordre

1. recórrer l'arbre esquerre (en postordre)
2. recórrer l'arbre dret (en postordre)
3. visitar l'arrel

Exemple: 4, 8, 5, 2, 9, 10, 6, 7, 3, i 1.

# Exemples

- ▶ Algorismes ja vistos fins ara per a arbres.
- ▶ Altres exemples: funcions que transformen arbres d'enters a llistes d'enters corresponents als recorreguts en preordre, inordre i postordre.  
Ús del mètode `splice` de llistes.

## Exemples recorregut en profunditat: preordre

```
list<int> preordre(const BinTree<int> &a)
/* Pre: cert */
/* Post: El resultat conté els nodes d'a en preordre */
{
    list<int> l;
    if (not a.empty()) {
        l=preordre(a.left());
        l.insert(l.begin(), a.value());
        l.splice(l.end(), preorder(a.right()));
        return l;
    }
}
```

Problemes d'eficiència. On?

# Recorregut en preordre millorat

```
void preordre(const BinTree<int> &a, list<int> &l)
/* Pre: l=L */
/* Post: l conté L seguit dels nodes d'a en preordre */
{
    if (not a.empty()) {
        l.insert(l.end(), a.value());
        preordre(a.left(), l);
        preordre(a.right(), l);
    }
}
```



## Exemples recorregut en profunditat: inordre

```
void inordre(const BinTree<int> &a, list<int> &l)
/* Pre: l=L */
/* Post: l conté L seguit dels nodes d'a en inordre */
{
    if (not a.empty()) {
        inordre(a.left(), l);
        l.insert(l.end(), a.value());
        inordre(a.right(), l);
    }
}
```

## Exemples recorregut en profunditat: postordre

```
void postordre(const BinTree<int> &a, list<int> &l)
/* Pre: l=L */
/* Post: l conté L seguit dels nodes d'a en postordre */
{
    if (not a.empty()) {
        inordre(a.left(), l);
        inordre(a.right(), l);
        l.insert(l.end(), a.value());
    }
}
```

# Recorregut en amplada

- ▶ Recorregut en amplada = recorregut per nivells.
- ▶ Nivell d'un node: distància a l'arrel (en nombre d'enllaços).
- ▶ Passos:
  - ▶ arrel de l'arbre (és a dir, el node de nivell 0),
  - ▶ tots els nodes que estan al nivell 1, d'esquerra a dreta,
  - ▶ tots els nodes que estan a nivell 2, també d'esquerra a dreta, etc.

Exemple: 1, 2, 3, 4, 5, 6, 7, 8, 9, i 10.

# Recorregut en amplada

```
void nivells(const BinTree<int>& a, list<int>& l)
/* Pre: l és buida */
/* Post: l conté els nodes d'a en ordre creixent respecte al nivell on
i els de cada nivell en ordre d'esquerra a dreta */
{
if (not a.empty()) {
    queue<BinTree<int>> c;
    c.push(a);
    while (not c.empty()) {
        BinTree<int> aux(c.front());
        l.insert(l.end(),aux.value());
        if (not aux.left().empty()) c.push(aux.left());
        if (not aux.right().empty()) c.push(aux.right());
        c.pop();
    }
}
```