Estructures de dades arborescents

R. Ferrer i Cancho

Universitat Politècnica de Catalunya

PRO2 (curs 2010-2011)
Versió 0.3

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).

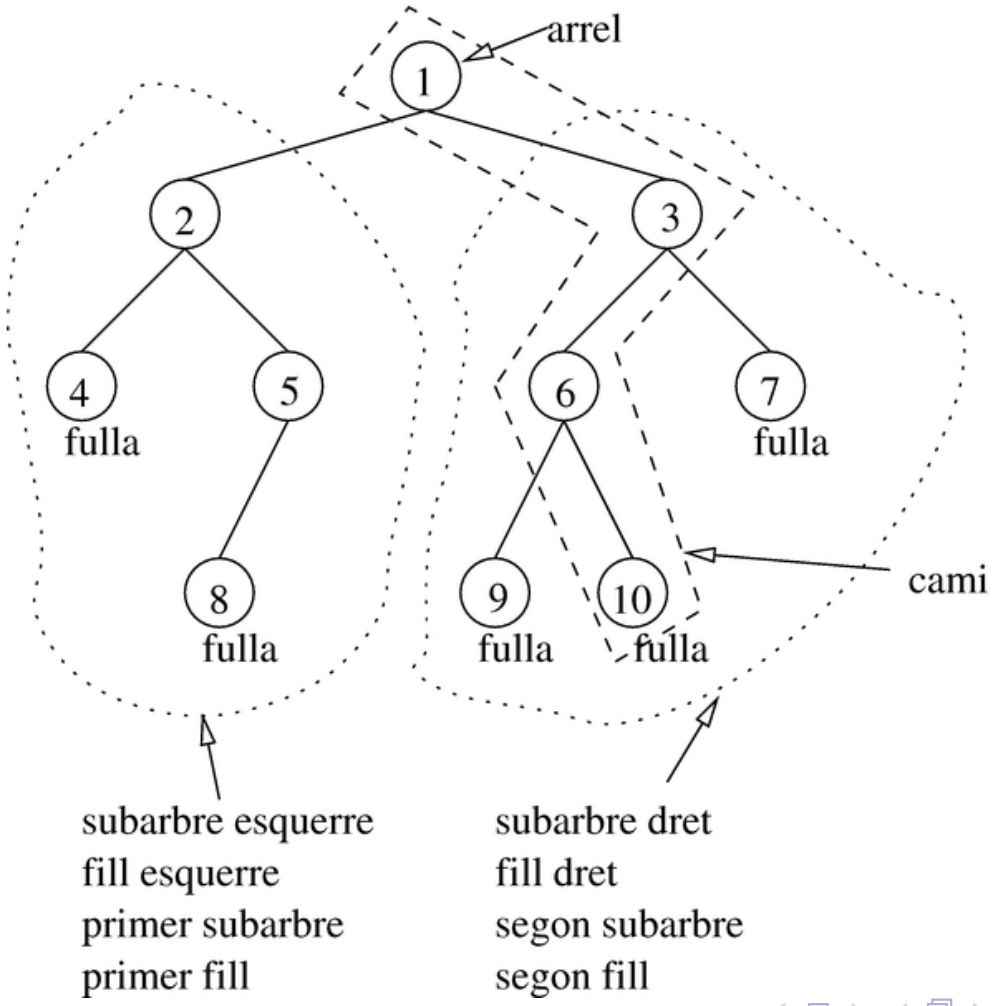
Estructures de dades arborescents

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

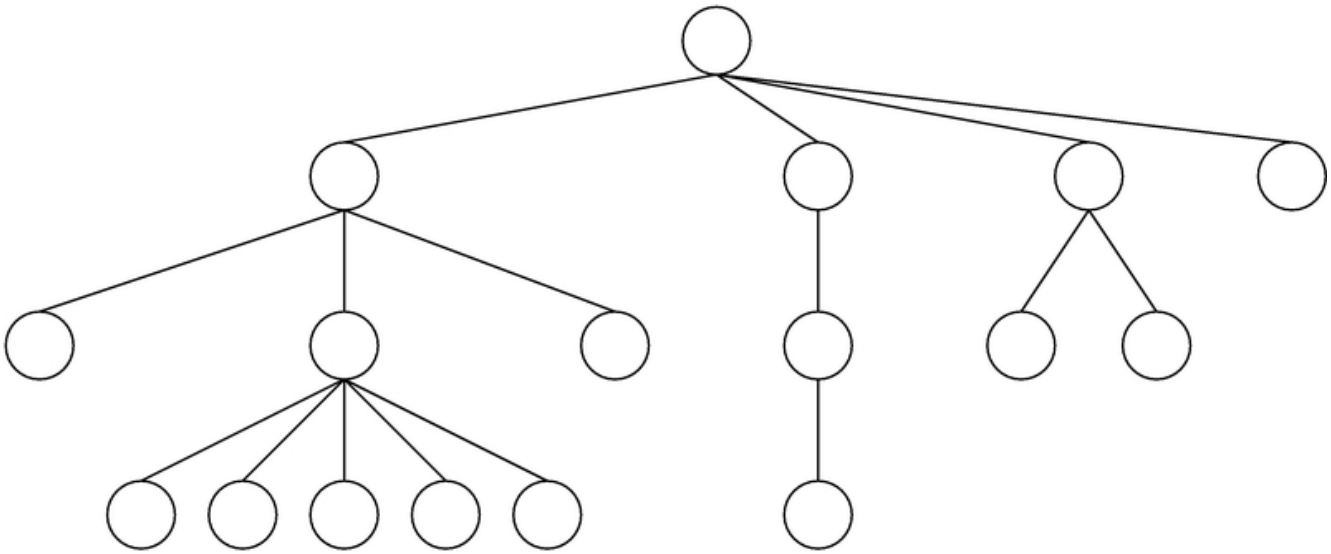
Conceptes d'arbres I



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.

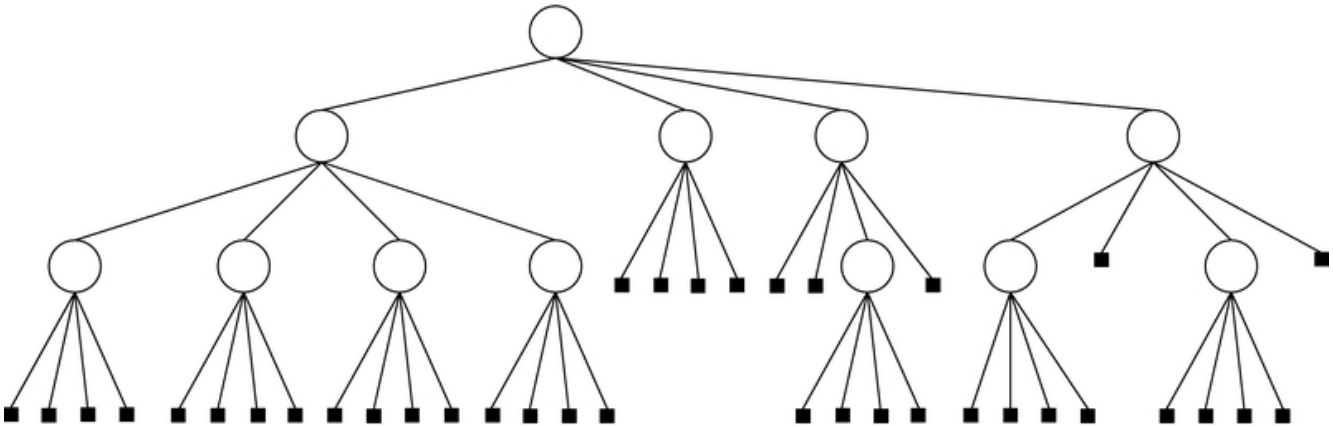
Exemple d'un arbre general



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

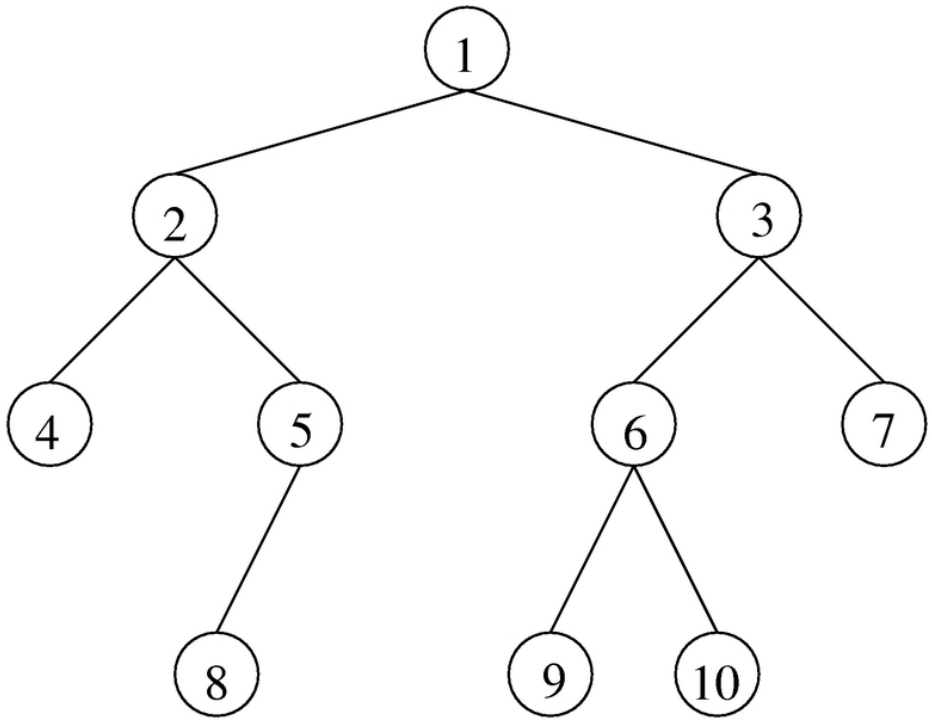


- Quadrats negres: arbres buits.
- Per claredat convé representar explícitament els arbres buits en els arbres N-aris.

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 (Arbre)

Veure document:
especificacions_estructures_dades_arborescents.pdf

Instanciació: `Arbre<tipus> nom_arbre;`

- Exemples:
  - `Arbre<int> a;`
  - `Arbre<Estudiant> b;`.

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

```
void plantar(const T& x, Arbre& a1, Arbre& a2);
```

Ús d'arbres binaris

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

- *algorismes d'exploració (no operació* `plantar`*)*
- *algorismes de modificació (operació* `plantar`*)*
- *algorismes de creació o generació (operació* `plantar`*).*

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(Arbre<int>& a)
/* Pre: a = A */
{
  ...
}
/* 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(Arbre<int>& a)
/* Pre: a=A */
{
  int x;
  if (a.es_buit()) x = 0;
  else{
        Arbre<int> a1, a2;
        a.fills(a1, a2);
        int y = altura(a1);
        int z = altura(a2);
        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(Arbre<int>& a)
  /* Pre: a = A */
  {
    ...
  }
  /* Post: El resultat és el nombre de nodes de l'arbre A */
  ```
- Versió recursiva
  ```

Mida d'un arbre: solució

```
int mida(Arbre<int>& a)
/* Pre: a=A */
{
  int x;
  if (a.es_buit()) x = 0;
  else{
    Arbre<int> a1, a2;
    a.fills(a1, a2);
    int y = mida(a1);
    int z = mida(a2);
    x = y + z + 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(Arbre<int>& a, int x)
  /* Pre: a = A */
  {
    ...
  }
  /* Post: El resultat indica si x és a l'arbre A o no */
  ```
- Versió recursiva
  ```

Solució

```
bool cerca(Arbre<int>& a, int x)
/* Pre: a = A */
{
  bool b;
  if (a.es_buit()) b=false;
  else if (a.arrel()==x) b=true;
  else{
    Arbre<int> a1, a2;
    a.fills(a1, a2);
    b = cerca(a1, x);
    if (not b) b=cerca(a2, 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(Arbre<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(Arbre<int> &a, int k)
/* Pre: a = A */
{
  if (not a.es_buit()) {
    int n = a.arrel() + k;
    Arbre<int> a1, a2;
    a.fills(a1, a2);
    suma(a1, k);
    suma(a2, k);
    a.plantar(n, 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 $\rightarrow$ no ens cal un cas base explícit.
- Especificació:

```
Arbre<int> suma(Arbre<int> &a, int k)
/* Pre: a=A */
{
   ...
}
/* 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

```
Arbre<int> suma(Arbre<int> &a, int k)
/* Pre: a=A */
{
  Arbre<int> b;
  if (not a.es_buit()){
    int n = a.arrel() + k;
    Arbre<int> a1, a2;
    a.fills(a1, a2);
    a1 = suma(a1, k);
    a2 = suma(a2, k);
    b.plantar(n, a1, a2);
  }
  return b;
}
/* 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(Arbre<int> &a)
/* Pre: a = A */
/* Post: El resultat conté els nodes d'A en preordre */
{
  list<int> l;
  if (not a.es_buit()) {
    int node = a.arrel();
    Arbre<int> a1, a2;
    a.fills(a1, a2);
    l = preordre(a1);
    list<int> m;
    m = preordre(a2);
    l.insert(l.begin(), node);
    l.splice(l.end(), m);
  }
  return l;
}
```

Problemes d'eficiència. On?

Recorregut en preordre millorat

```
void preordre(Arbre<int> &a, list<int> & l)
/* Pre: a=A, l=L */
/* Post: l conté L seguit dels nodes d'A en preordre */
{
  if (not a.es_buit()) {
     int node = a.arrel();
     Arbre<int> a1, a2;
     a.fills(a1, a2);
     l.insert(l.end(), node);
     preordre(a1, l);
     preordre(a2, l);
  }
}
```

Exemples recorregut en profunditat: inordre

```
void inordre(Arbre<int> &a, list<int> & l)
/* Pre: a=A, l=L */
/* Post: l conté L seguit dels nodes d'A en inordre */
{
if (not a.es_buit()) {
   int node=a.arrel();
   Arbre<int> a1, a2;
   a.fills(a1, a2);
   inordre(a1, l);
   l.insert(l.end(),node);
   inordre(a2, l);
   }
}
```

Exemples recorregut en profunditat: postordre

```
void postordre(Arbre<int> &a, list<int> & l)
/* Pre: a=A, l=L */
/* Post: l conté L seguit dels nodes d'A en postordre */
{
if (not a.es_buit()) {
   int node=a.arrel();
   Arbre<int> a1;
   Arbre<int> a2;
   a.fills(a1, a2);
   postordre(a1, l);
   postordre(a2, l);
   l.insert(l.end(),node);
   }
}
```

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: versió iterativa

```
list<int> nivells(Arbre<int> &a)
/* Pre: a=A */
/* Post: El resultat conté els nodes d'A en ordre creixent respecte al nivell
         al qual es troben, i els de cada nivell en ordre d'esquerra a dreta */
{
  list<int> l;
  if (not a.es_buit()){
     queue<Arbre<int> > c; // compte: queue<Arbre<int>> c no compila!
     int node;
     c.push(a);
     while (not c.empty()){
       a = c.front();
       node = a.arrel();
       l.insert(l.end(), node);
       Arbre<int> a1, a2;
       a.fills(a1,a2);
       if (not a1.es_buit()) c.push(a1);
       if (not a2.es_buit()) c.push(a2);
       c.pop();
     }
  }
  return l;
}
```

Problema d'eficiència. On?

Recorregut en amplada: versió recursiva I

```
void nivells(Arbre<int> &a, list<int> & l)
/* Pre: a=A, l és buida */
/* Post: l conté els nodes d'A en ordre creixent respecte al nivell
al qual es troben, i els de cada nivell en ordre d'esquerra a dreta */
{
if (not a.es_buit()) {
   queue <Arbre<int> > c;
   c.push(a);
   nivells_rec_aux(c,l);
   }
}
```

Recorregut en amplada: versió recursiva II

```
void nivells_aux(queue<Arbre<int> > &c, list<int> & l)
/* Pre: c=C, C no conté cap arbre buit , l=L */
/* Post: l és L seguit dels nodes dels arbres de C en ordre creixent
respecte al nivell al qual es troben; els nodes del mateix nivell, si són
d'arbres diferents, apareixen a l'ordre al qual hi són a C, si són d'un mateix
arbre apareixen en ordre d'esquerra a dreta */
{
if (not c.empty()) {
   Arbre<int> a;
   a = c.front();
   c.pop();
   int node = a.arrel();
   Arbre<int> a1, a2;
   a.fills(a1,a2);
   if (not a1.es_buit()) c.push(a1);
   if (not a2.es_buit()) c.push(a2);
   l.insert(l.end(), node);
   nivells_rec_aux(c, l);
   }
}
```

Post informal no del tot precisa

Per què la llista amb el recurregut és un paràmetre per referència i no un retorn de funció?