Arbres

Ricard Gavaldà

Programació 2

Facultat d'Informàtica de Barcelona, UPC

Primavera 2018

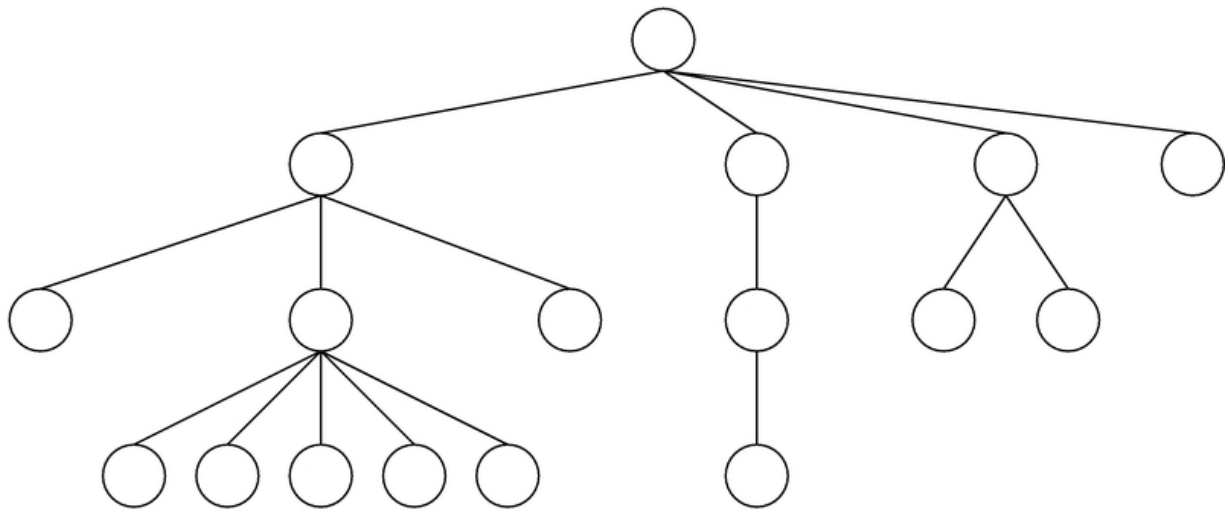
Aquesta presentació no substitueix els apunts

Contingut

Arbres generals i arbres N-aris

Arbres generals; conceptes

- node o nus
- fill, pare
- descendent, ascendent
- germà
- arrel, fulla
- camí
- nivell; alçària



Arbres generals; conceptes, II

Definicions com a graf:

Def. 1: un arbre és un graf dirigit tal que o bé és buit, o bé té un node anomenat arrel tal que hi ha exactament un camí de l'arrel a qualsevol altre node

Def. 2: un arbre és un graf no dirigit, connex, amb un arc menys que nodes i un node distingit anomenat arrel

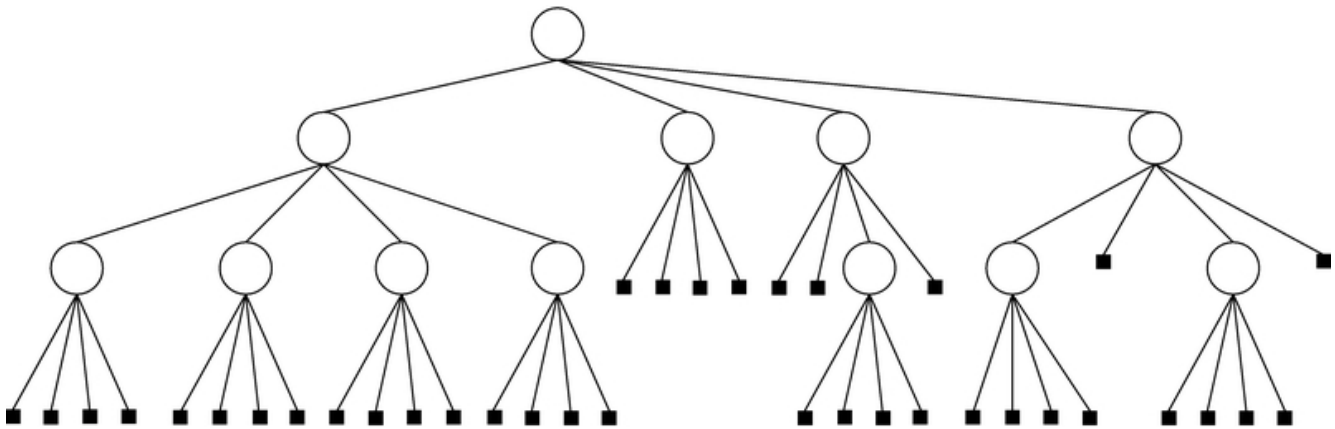
Les dues són poc útils algorísmicament

Arbres generals; conceptes, III

un arbre o bé és l'arbre buit
o bé és un node anomenat arrel amb zero o més
arbres successors anomenats fills o subarbres

Es presta a tractaments algorísmics recursius

Arbres N-aris



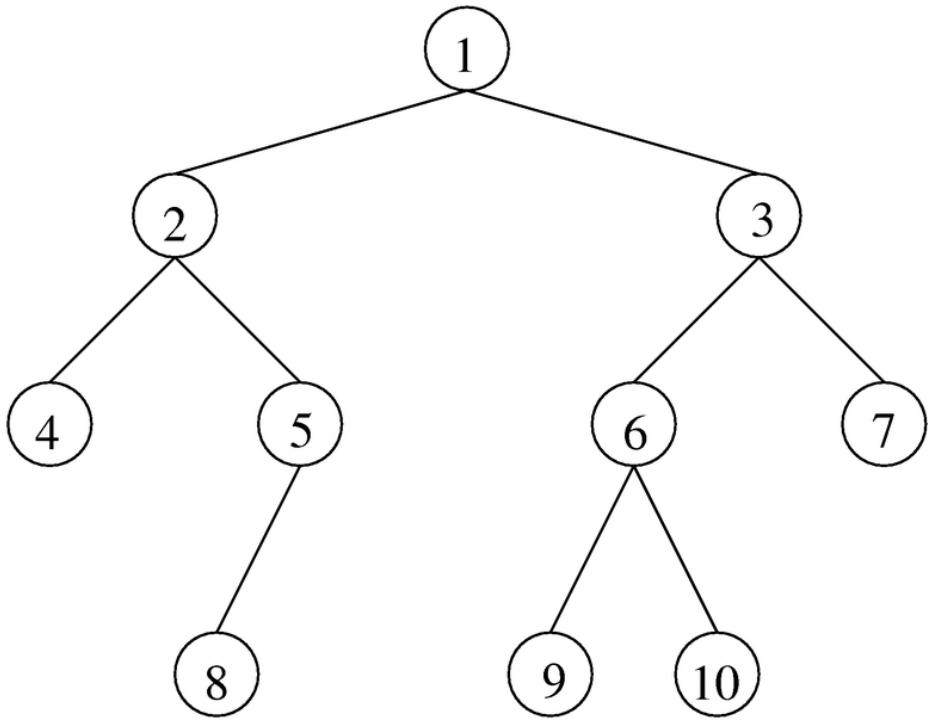
- Def.: Tots els subarbres no buits tenen exactament el mateix nombre de fills, N, que poden ser buits o no
- Exemple: Arbre 4-ari; quadrats negres = arbres buits
- Per claredat convé representar explícitament els arbres buits en els arbres N-aris

Arbres binaris: Classe `BinTree`

Arbres binaris

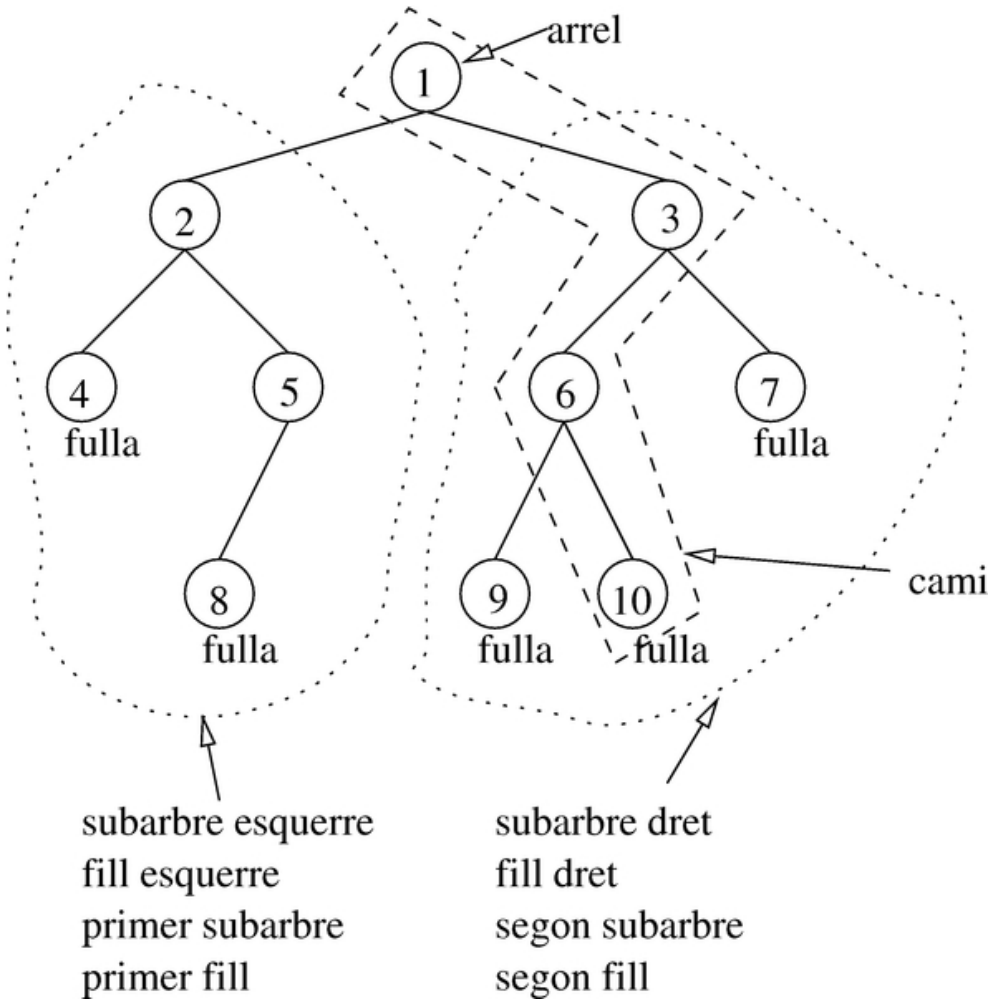
- Cas particular dels arbre N-aris, on $N = 2$
- Quan diem arbres sense detallar més, ens referim per defecte a arbres binaris
- Els dos fills d'un node són anomenats esquerre i dret

Exemple d'arbre binari



No hem dibuixat els subarbres buits amb caixes negres com en l'exemple d'arbre 4-ari. La inclinació diu si és dret o esquerre

Exemple d'arbre binari, II



Especificació dels arbres binaris

Template amb un paràmetre T

Creadores:

```
BinTree();
/* Pre: cert */
/* Post: El resultat és un arbre sense cap element */

BinTree(const T& x);
/* Pre:  cert */
/* Post: El resultat té x com a arrel, i els seus fills
esquerre i dret són buits */

BinTree (const T& x, const BinTree& left, const BinTree& right);
/* Pre:  cert */
/* Post: El paràmetre implícit té x com a arrel,
left com a fill esquerre i right com a fill dret */
```

Especificació dels arbres binaris

Consultores:

```
bool empty() const;
/* Pre: cert */
/* Post: retorna cert si i només si
el paràmetre implícit és buit */

BinTree left() const;
/* Pre: El paràmetre implícit no és buit */
/* Post: El resultat és el fill esquerre del p.i. */

BinTree right() const;
/* Pre: El paràmetre implícit no és buit */
/* Post: El resultat és el fill dret del p.i. */

const T& value() const;
/* Pre: El paràmetre implícit no és buit */
/* Post: El resultat és l'arrel del paràmetre implícit */
```

Especificació dels arbres binaris

cap modificadora! La única manera de modificar un arbre és construir l'arbre modificat i assignar-lo a l'original.

Totes les operacions són en temps constant, excepte la *destructora*

Important per al temps constant: tot és `const`. No cal fer còpies dels fills

En l'assignació

```
a1 = a2;
```

Si l'arbre que contenia `a1` abans de l'assignació forma part d'algun altre arbre, llavors temps constant. Si no, temps proporcional a la mida d'aquell arbre perquè cal destruir-lo.

Operacions amb arbres binaris

Mida d'un arbre

```
/* Pre: cert */
/* Post: El resultat és el nombre de nodes d' a */
int size(const BinTree<int>& a);
```

Mida d'un arbre

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

Alçària d'un arbre

Def.: L'alçària d'un arbre és la longitud del camí (nombre de nodes) més llarg de l'arrel a una fulla

Especificació:

```
/* Pre: cert */
/* Post: El resultat és l'alçària de l'arbre a*/
int alcaria(const BinTree<int>& a);
```

Alçària d'un arbre

Def.: L'alçària d'un arbre és la longitud del camí més llarg de l'arrel a una fulla. Longitud vol dir número de nodes.

Poc útil algorísmicament

Observem:

- alcaria(buit) = 0
- si a no buit, ...

  alcaria(a) = 1 + max(alcaria(a.left()),alcaria(a.right()))

Demostració: per inducció!

Alçària d'un arbre

```
/* Pre: cert */
/* Post: El resultat és l'alçària de l'arbre a*/
int alcaria(const BinTree<int>& a) {
  if (a.empty())
    return 0;
  else
    return 1 + max(alcaria(a.left(),alcaria(a.right());
}
```

Exercici: feu-ho (eficientment) sense max, amb un if

Cerca d'un valor `int` en un arbre

```
/* Pre: cert */
/* Post: El resultat indica si x és a l'arbre a o no */
bool cerca(const BinTree<int>& a, int x);
```

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

cerca(buit,x) = fals
cerca(a,x) = cert, si arrel(a) = x
cerca(a,x) = cerca(a.left(),x) or cerca(a.right(),x), si arrel(a) $\neq$ x

Solució

```
/* Pre: cert */
/* Post: El resultat indica si x és a l'arbre a o no */
bool cerca(const BinTree<int>& a, int x) {
    if (a.empty()) return false;
    else return a.value() == x
                or cerca(a.left(),x)
                or cerca(a.right(),x);
}
```

Nota: Eficient perquè or condicional

Sumar un valor k a tots els nodes

```
/* Pre: cert */
/* Post: retorna un arbre amb la mateixa forma que a,
i en el qual cada node val k més el valor del node
corresponent en a */
BinTree suma(const BinTree<int> &a, int k);
```

Sumar un valor k a tots els nodes

```
/* Pre: cert */
/* Post: retorna un arbre amb la mateixa forma que a,
i en el qual cada node val k més el valor del node
corresponent en a */
BinTree suma(const BinTree<int> &a, int k) {
   if (a.empty())
      return BinTree<int>();
   else
      return BinTree<int>(a.value() + k,
                  suma(a.left(),k), suma(a.right(),k));
}
```

Sumar un valor k a tots els nodes

Fem-ho sobre el mateix arbre, com una acció:

```
/* Pre: a = A */
/* Post: deixa en a el resultat de sumar k a l'arbre A */
void suma(BinTree<int> &a, int k) {
   if (not a.empty()) { // si és buit no cal fer res
      a = BinTree<int>(a.value() + k,
                       suma(a.left(),k),
                       suma(a.right(),k));
   }
}
```

Recorreguts canònics d'arbres

Recorreguts d'arbres

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

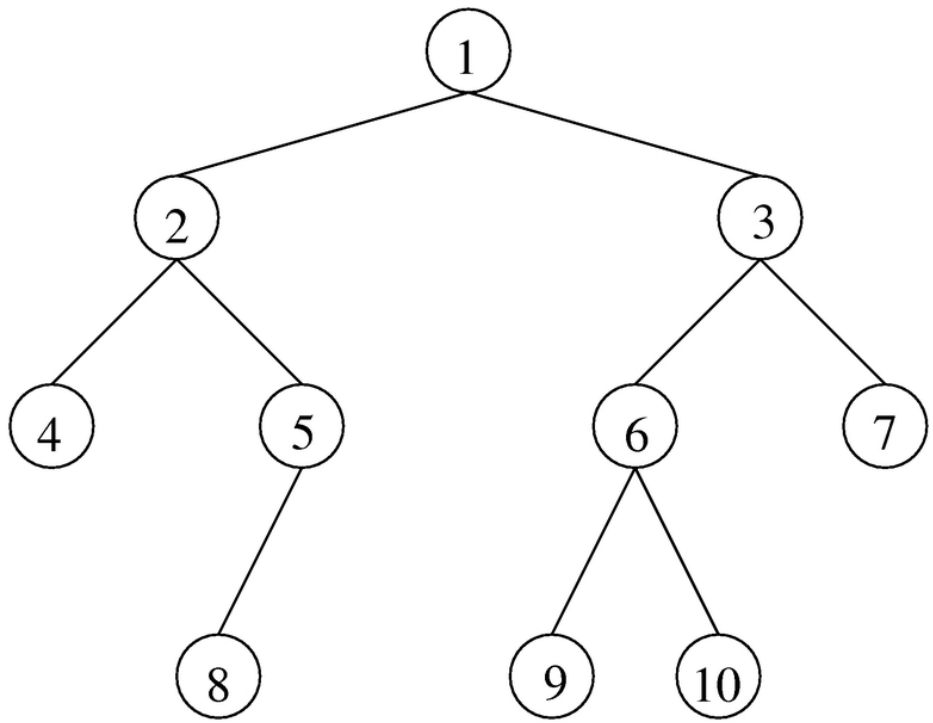
- Recorregut en profunditat
  - En preordre
  - En inordre
  - En postordre
- Recorregut en amplada o per nivells

En tots els casos: recórrer l'arbre buit = no fer res

Recorreguts en profunditat: preordre

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

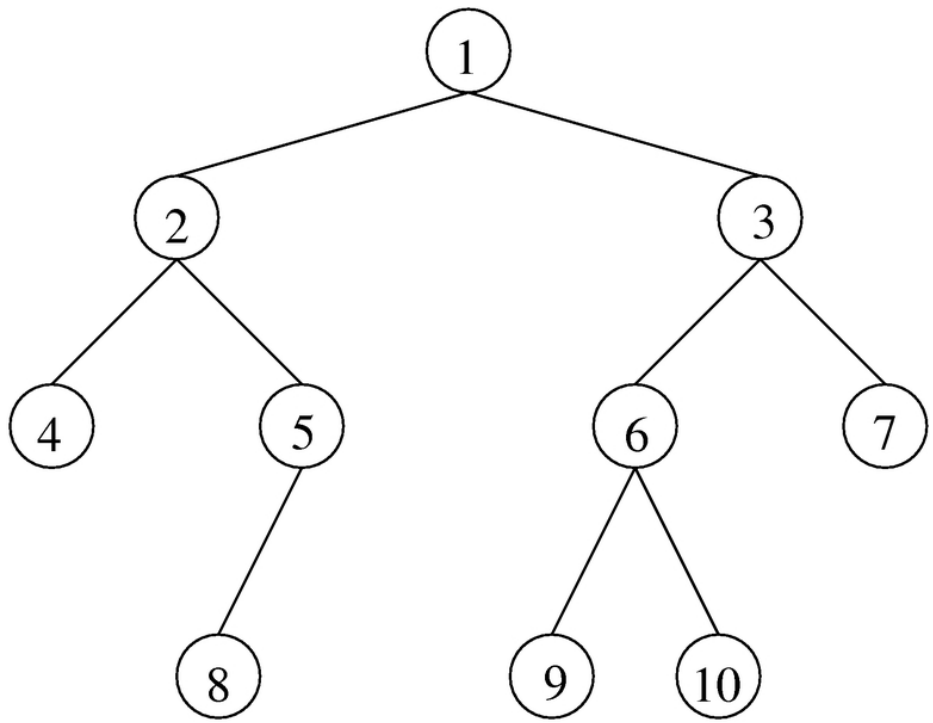
Exemple:
1, 2, 4, 5, 8, 3, 6, 9, 10 i 7



Recorreguts en profunditat: inordre

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

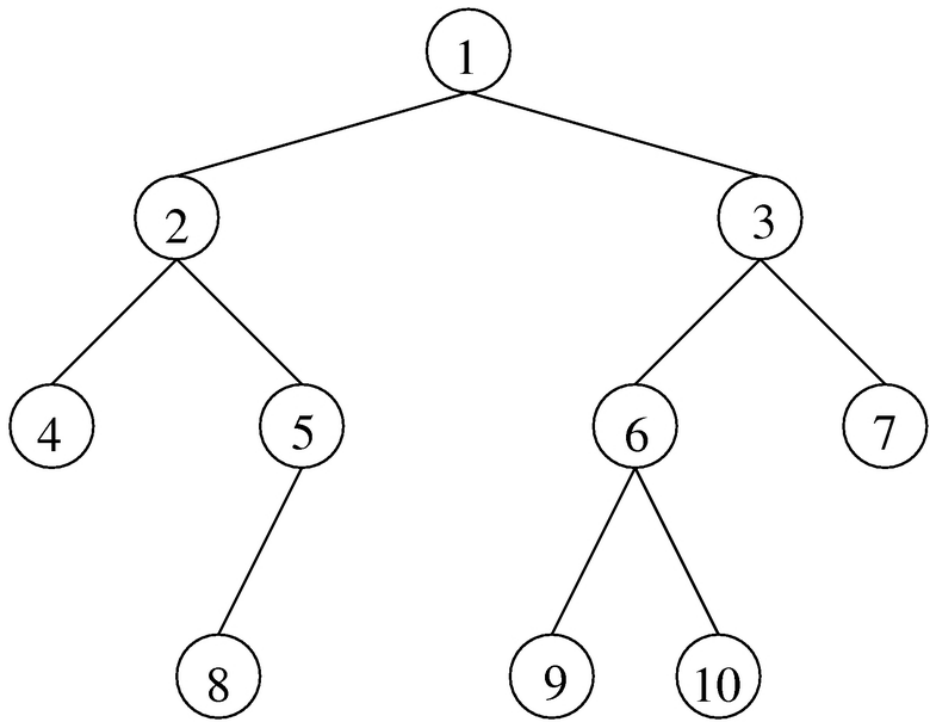
Exemple:
4, 2, 8, 5, 1, 9, 6, 10, 3, i 7



Recorreguts en profunditat: postordre

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

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



Recorregut en preordre

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

Exercici: Com canviem les instruccions del mig per obtenir els recorreguts en inordre i en postordre?

Recorregut en inordre

Manera alternativa: afegir a una llista donada
Obtenim el recorregut fent una crida inicial amb la llista buida

```
list<int> rec;
inorder(a,rec);

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

Recorregut en amplada o per nivells

Una llista on:

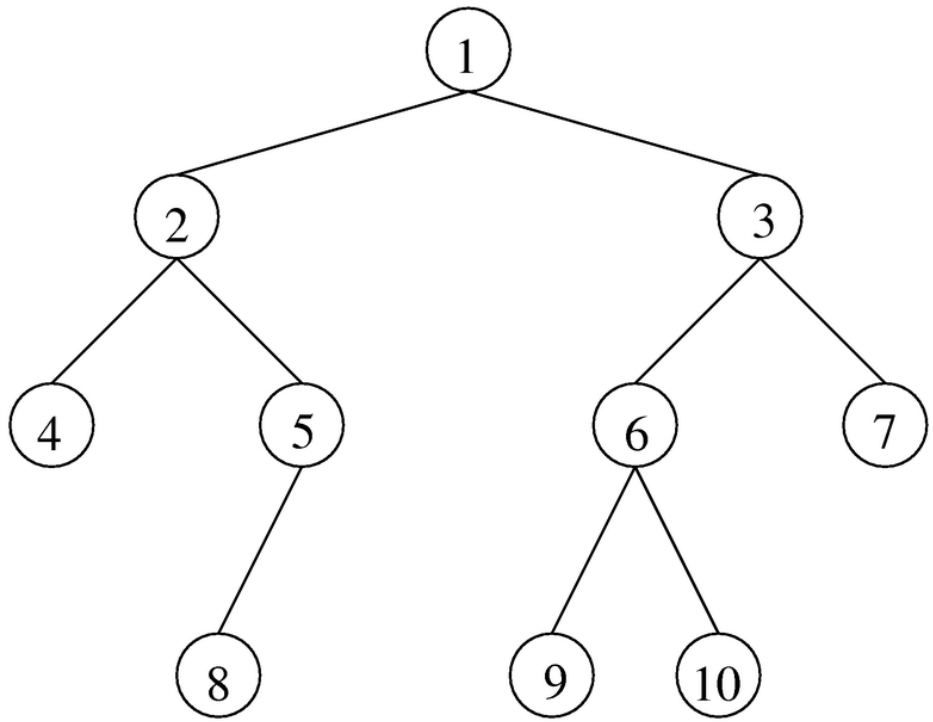
- tots els nodes del nivell i apareixen en la llista abans que els del nivell $i + 1$
- i, dins de cada nivell, els nodes apareixen d'esquerra a dreta

Recorregut en amplada o per nivells

Es fa amb una cua
Repetir:

- agafar primer arbre de la cua;
- llistar la seva arrel;
- ficar els seus dos fills a la cua;

Invariant: la cua conté alguns nodes del nivell k seguits dels fills dels nodes de nivell k que no són a la cua



Recorregut en amplada

```
/* Pre: cert */
/* Post: El resultat conté el recorregut d'a en amplada */
list<int> nivells(const BinTree<int>& a) {
    list<int> l;  // inicialment, buida
    if (not a.es_buit()) {
        queue<BinTree<int> > c;
        c.push(a);
        while (not c.empty()) {
           aux = c.front();
           c.pop();
           l.insert(l.end(),aux.value());
           if (not aux.left().empty()) c.push(aux.left());
           if (not aux.right().empty()) c.push(aux.right());
        }
    }
    return l;
}
```