Arbres

Programació 2
Facultat d'Informàtica de Barcelona, UPC

Conrado Martínez

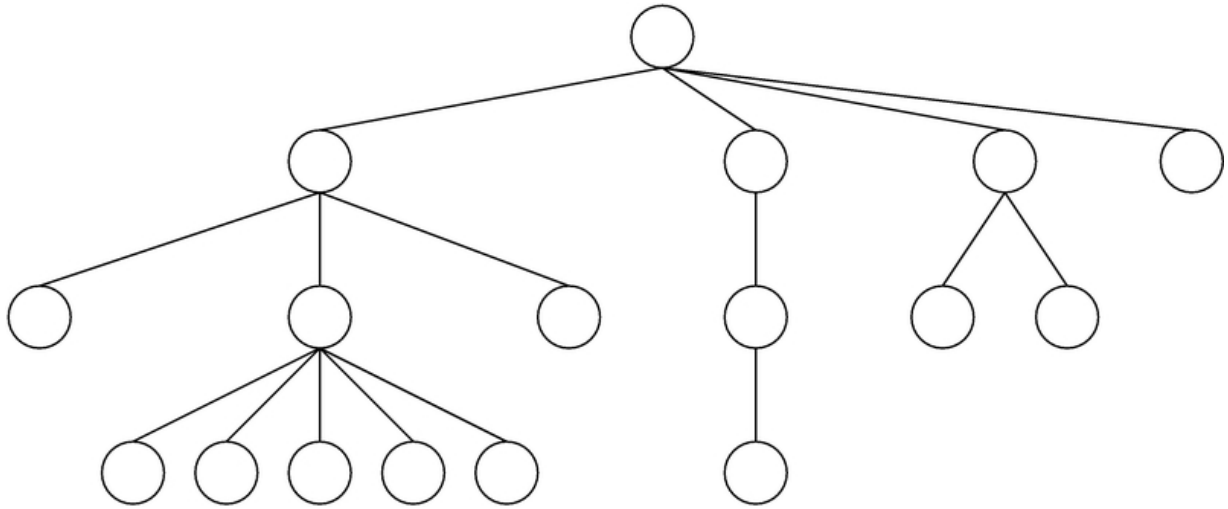
Primavera 2019

- Apunts basats en els d'en Ricard Gavaldà
- Aquestes transparències **no** substitueixen els apunts de l'assignatura, els complementen

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

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

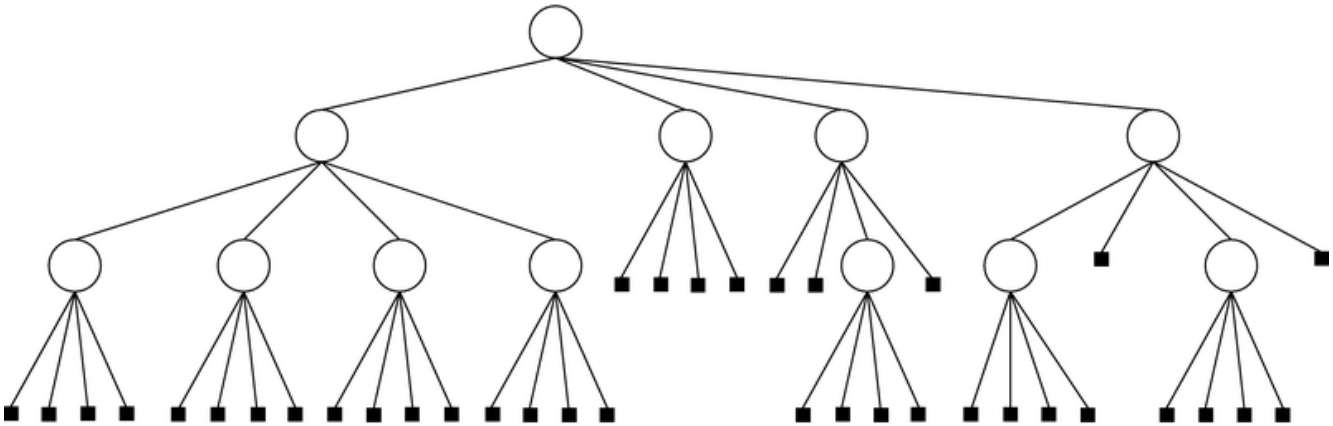
Les dues són poc útils algorísmicament

Arbres generals: conceptes

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
- Tècnicament la definició correspón a arbres arrelats ordenats

Arbres N-aris



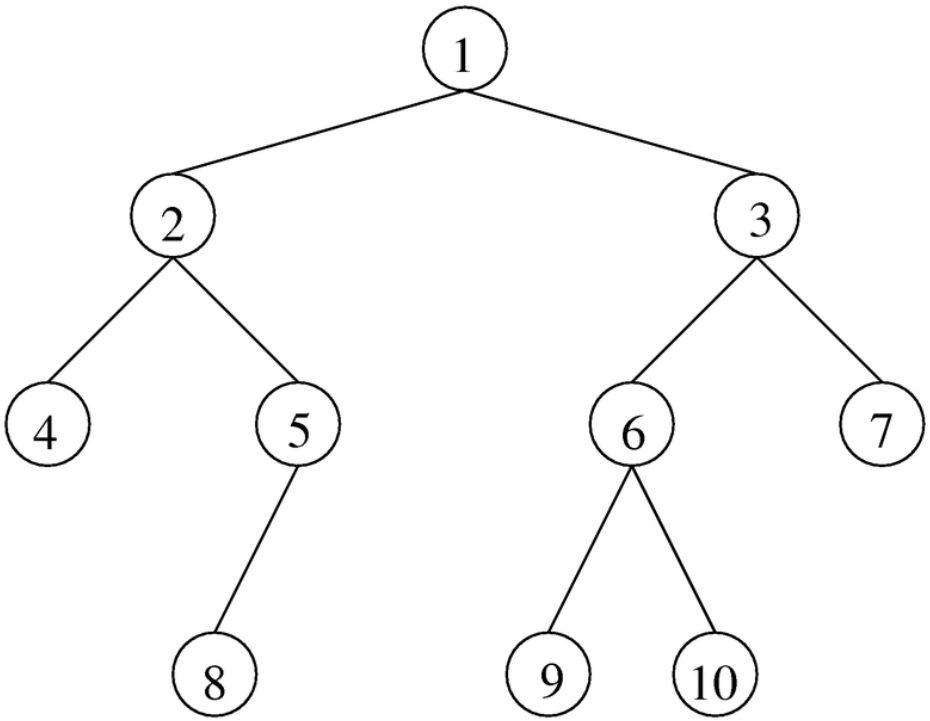
- Def.: Tots els subarbres no buits tenen exactament el mateix nombre de fills, N, que poden ser buits o no
- Exemple: Arbre 4-ari; quadrats negres = arbres buits
- Per claredat convé representar explícitament els arbres buits en els arbres N-aris; típicament els subarbres buits es representen mitjançant un quadrat (negre o blanc)

Arbres binaris: Classe `BinTree`

Arbres binaris

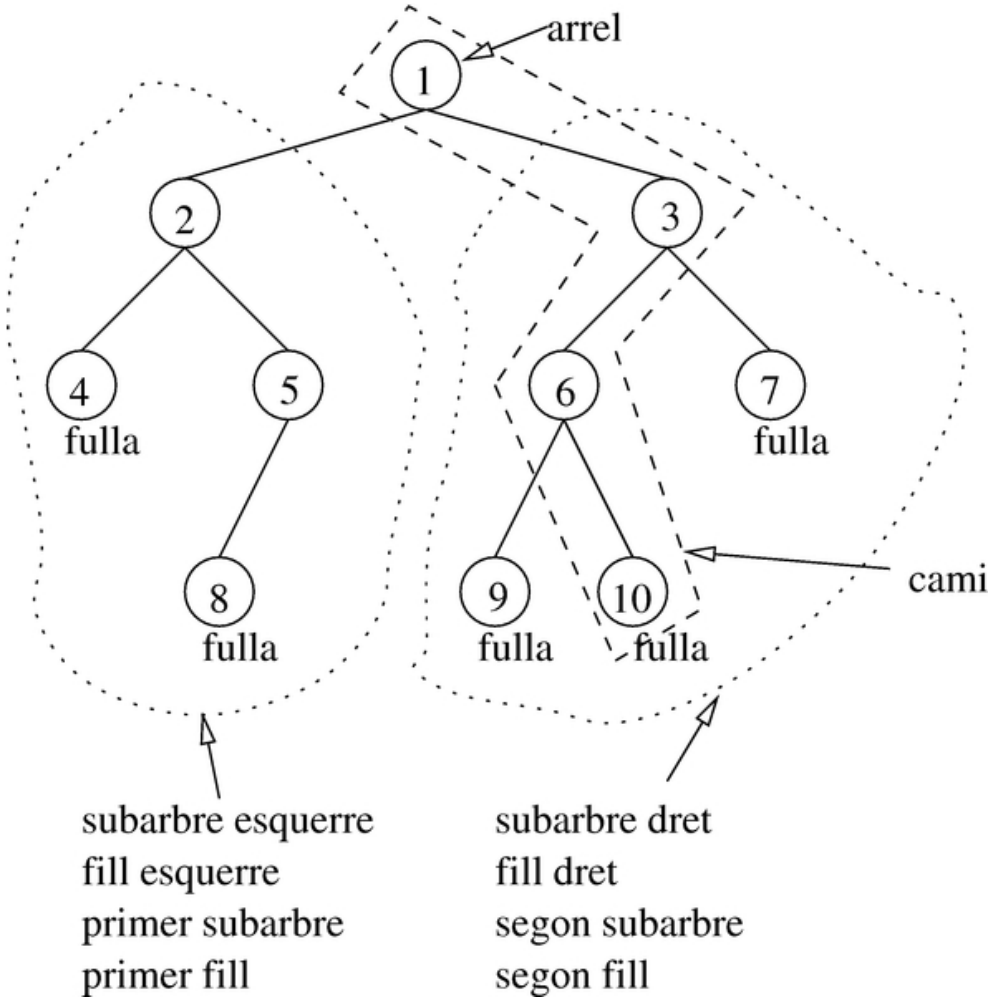
- Cas particular dels arbre N-aris amb $N = 2$
- Quan diem arbres sense detallar més, ens referim per defecte a arbres binaris
- Els dos fills d'un node són anomenats esquerre i dret

Exemple d'arbre binari



No hem dibuixat els subarbres buits amb quadrats negres com en l'exemple d'arbre 4-ari. La inclinació de cada aresta indica si el fill és dret o esquerre

Exemple d'arbre binari



Especificació dels arbres binaris

```
template <typename T> class BinTree {
public:
 BinTree();
 /* Pre: cert */
 /* Post: crea un arbre buit */

 BinTree(const T& x);
 /* Pre:  cert */
 /* Post: crea un arbre binari amb un sol node, l'arrel,
          que conté x, i els seus fills esquerre i dret
          són buits */

 BinTree (const T& x, const BinTree& left, const BinTree& right);
 /* Pre:  cert */
 /* Post: crea un arbre binari amb x a l'arrel,
          i left i right com com a fills esquerre
          i dret, respectivament */
```

Especificació dels arbres binaris

```
// Consultores:
bool empty() const;
/* Pre: cert */
/* Post: retorna cert si i només si
         l'arbre és buit */
BinTree left() const;
/* Pre: L'arbre implícit no és buit */
/* Post: retorna el fill esquerre de l'arbre implícit */

BinTree right() const;
/* Pre: L'arbre implícit no és buit */
/* Post: retorna el fill dret de l'arbre implícit */

const T& value() const;
/* Pre: L'arbre implícit no és buit */
/* Post: retorna el valor de l'arrel de l'arbre */
```

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 requereixen temps constant (excepte la *destructora*)
- Important per al temps constant: tot és `const`, no es fan còpies dels fills
- En l'assignació

```
a1 = a2;
```

requereix temps constant excepte en el cas de que `a1` no "comparteixi" cap subarbre amb cap altre objecte; llavors el temps necessari és proporcional a la mida d'`a1` doncs cal destruir-lo

Operacions amb arbres binaris

Mida d'un arbre

```
/* Pre: cert */
/* Post: El resultat és el nombre de nodes d' a */
template <typename T>
int size(const BinTree<T>& a);
```

Mida d'un arbre

```
/* Pre: cert */
/* Post: El resultat és el nombre de nodes d'a */
template <typename T>
int size(const BinTree<T>& 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*/
template <typename T>
int alcaria(const BinTree<T>& a);
```

Alçària d'un arbre

- $\mathsf{alcaria}(\square) = 0$
- si a no buit, ...

$$\mathsf{alcaria}(a) = 1 + \max(\mathsf{alcaria}(a.\texttt{left}()), \mathsf{alcaria}(a.\texttt{right}()))$$

Demostració: per inducció!

Alçària d'un arbre

```
/* Pre: cert */
/* Post: El resultat és l'alçària de l'arbre a */
template <typename T>
int alcaria(const BinTree<T>& a) {
  if (a.empty())
    return 0;
  else
    return 1 + max(alcaria(a.left(),alcaria(a.right());
}
```

Cerca d'un valor en un arbre

```
/* Pre: cert */
/* Post: El resultat indica si x és a l'arbre a o no *
template <typename T>
bool cerca(const BinTree<int>& a, const T& x);
```

Cerca d'un valor en un arbre

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

Cerca d'un valor en un arbre

```
/* Pre: cert */
/* Post: El resultat indica si x és a l'arbre a o no */
template <typename T>
bool cerca(const BinTree<T>& a, const T& x) {
    if (a.empty()) return false;
    else return (a.value() == x)
                or cerca(a.left(),x)
                or cerca(a.right(),x);
```

És imprescindible que l'operador d'igualtat estigui definit per elements del tipus `T`: si x y z són de tipus `T` llavors `x == y` ha d'estar definit!
Nota: Eficient perquè `or` és 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):

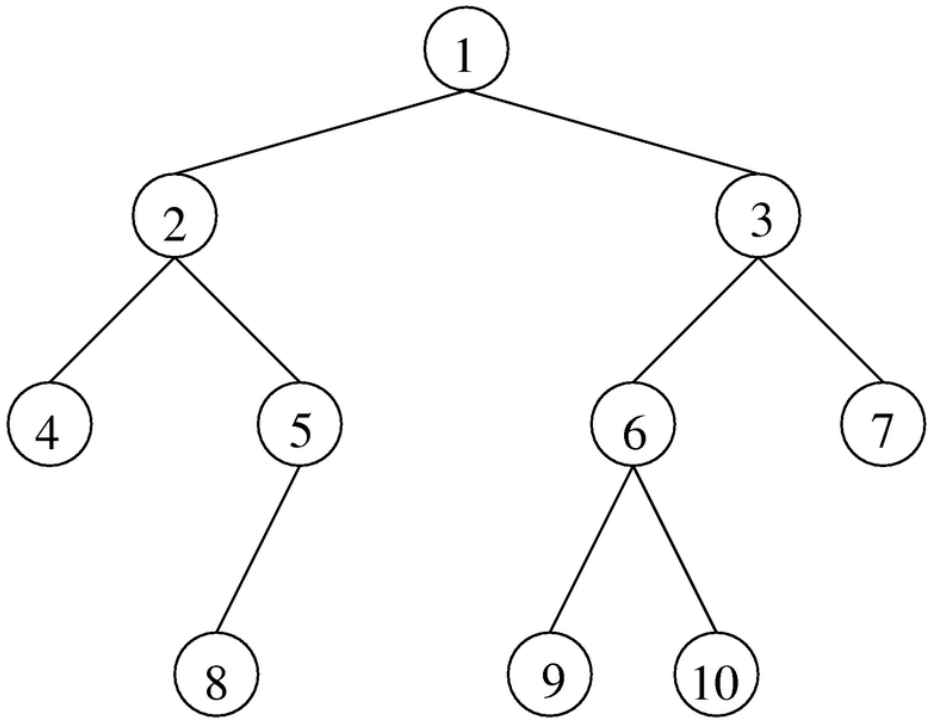
- Recorreguts en profunditat
  - En preordre
  - En inordre
  - En postordre
- Recorregut en amplada o per nivells

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)

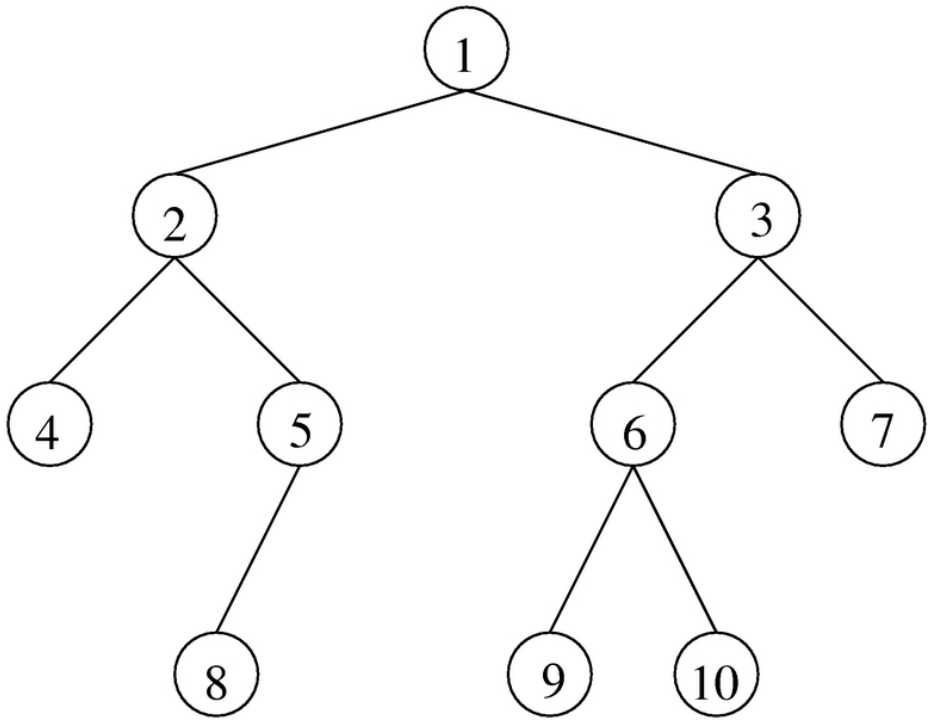
Exemple:
1, 2, 4, 5, 8, 3, 6, 9, 10 i 7



Recorreguts en profunditat: inordre

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

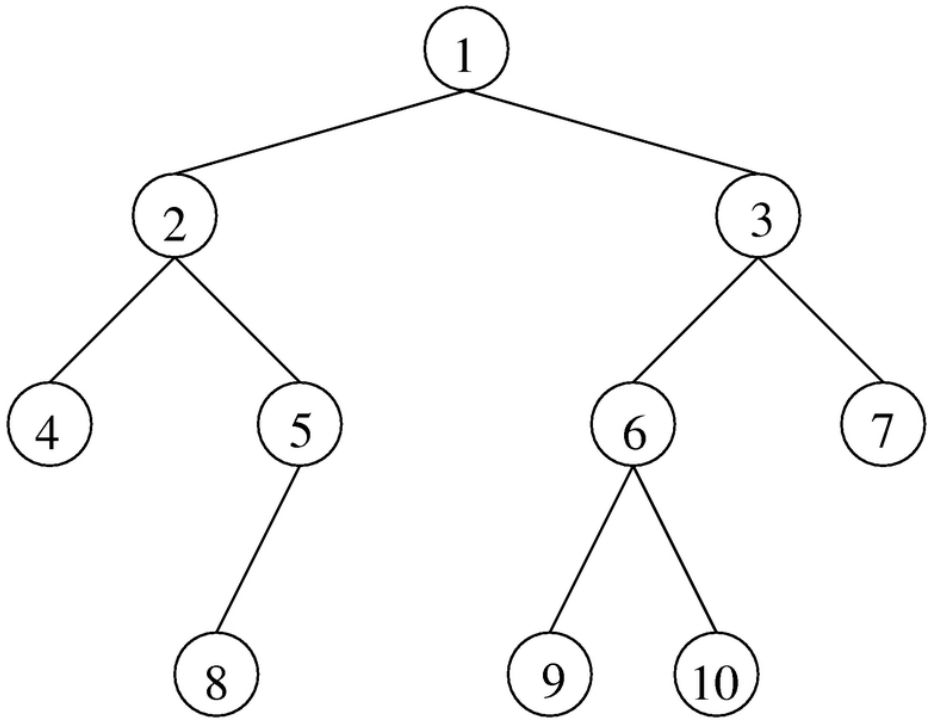
Exemple:
4, 2, 8, 5, 1, 9, 6, 10, 3, i 7



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 */
template <typename T>
list<T> preorder(const BinTree<T>& a) {
    list<T> l;    // inicialment, buida
    if (not a.empty()) {
        l.push_back(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

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

// Ús:
BinTree<int> a;
...
list<int> rec;
inorder(a,rec);
```

Recorregut en amplada o per nivells

Visita d'els nodes d'un arbre donat de manera que:

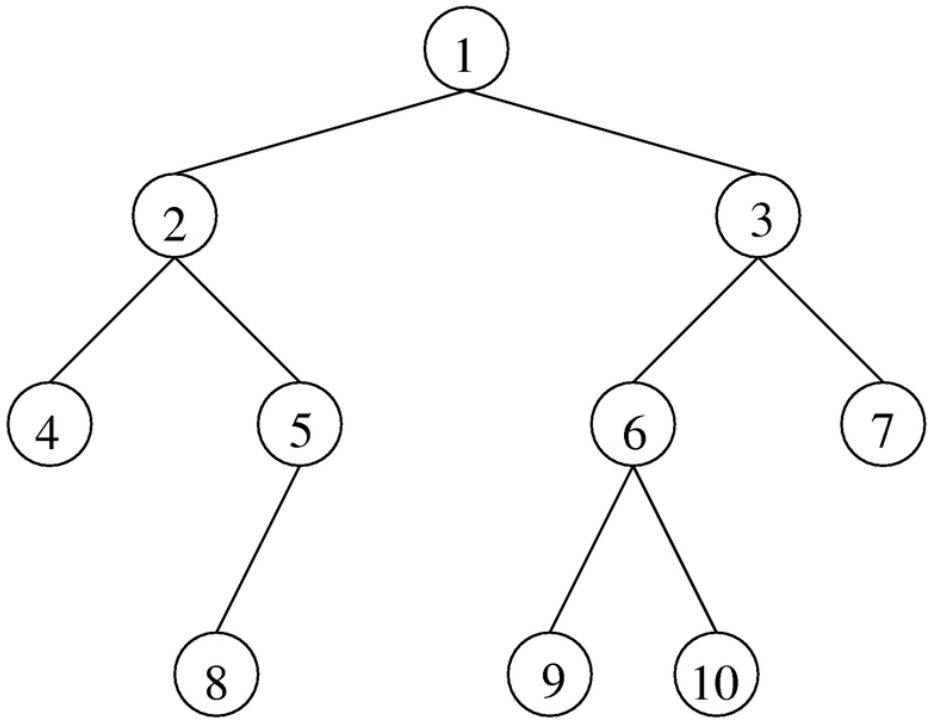
- tots els nodes del nivell i s'han visitat abans que els del nivell $i+1$
- dins de cada nivell, els nodes es visiten d'esquerra a dreta

Recorregut en amplada o per nivells

Es fa amb una cua
Repetir:

- agafar primer arbre de la cua;
- visitar 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 */
template <typename T>
list<T> nivells(const BinTree<T>& a) {
    list<T> l;  // inicialment, buida
    if (not a.empty()) {
        queue< BinTree<T> > c;
        c.push(a);
        while (not c.empty()) {
            BinTree<T> aux = c.front();
            c.pop();
            l.push_back(aux.value());
            if (not aux.left().empty()) c.push(aux.left());
            if (not aux.right().empty()) c.push(aux.right());
        }
    }
    return l;
}
```