Estructuras Arborescentes

- Cada elemento puede tener más de un sucesor simultáneamente
- Árboles generales: cualquier número de sucesores
- Árboles N-arios: menor o igual que N sucesores (N>0)
- Árboles binarios: menor o igual que dos sucesores (es decir, N-ario con N=2)
- No están en la STL

Estructuras Arborescentes

Se usan para representar diversos tipos de datos:

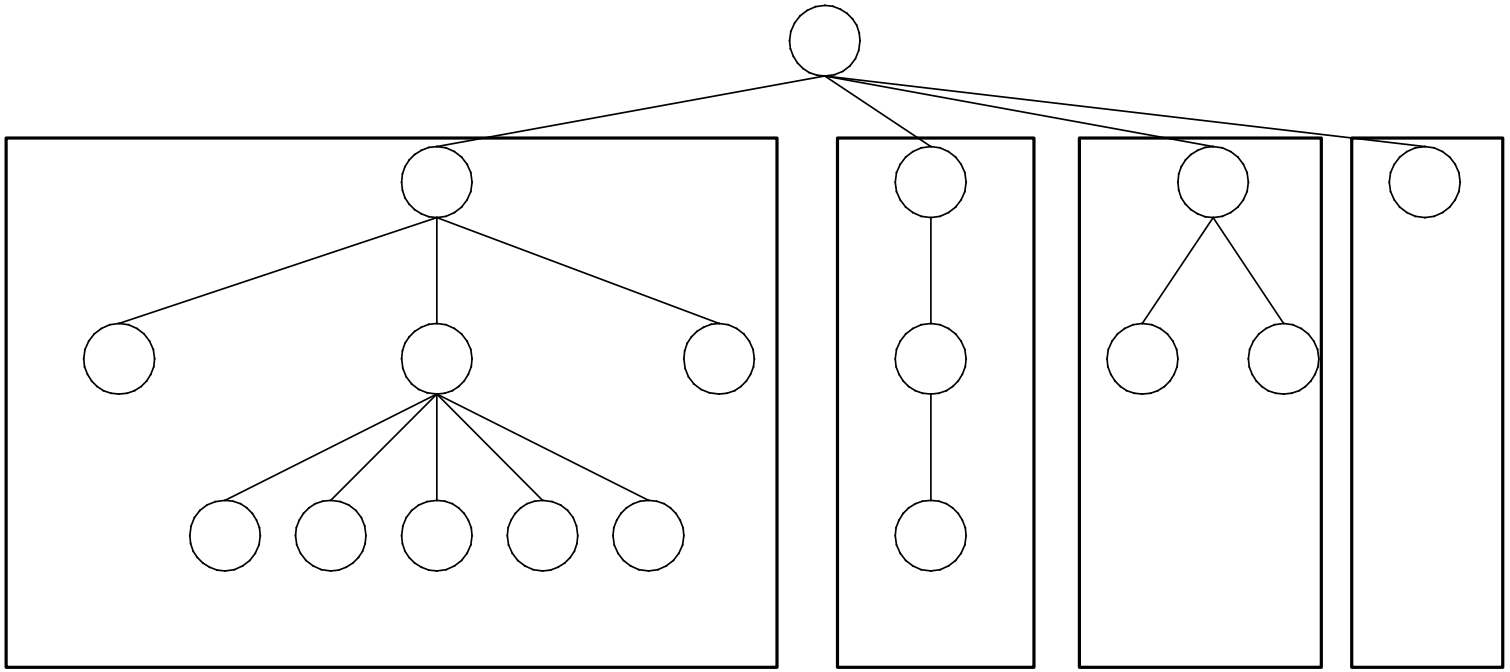
- Jerarquías: presidente, vicepresidentes, etc
- Árboles genealógicos
- Conexiones con un único origen o destino: ríos, trenes de cercanías
- Estructuras ordenadas de acceso rápido: BST, AVL
- Estructuras de carpetas en un sistema operativo

Estructuras Arborescentes

- Un árbol no vacío tiene un elemento especial, la *raíz*, que es el único consultable.
- La raíz está conectada a cero o más subárboles, llamados *hijos*:
  - Árboles generales: cualquier número de subárboles, pero no pueden ser vacíos.
  - Árboles N-arios: exactamente N subárboles, algunos pueden ser vacíos (o todos, o ninguno).
  - Árboles binarios: idem, con N=2.

Árboles generales

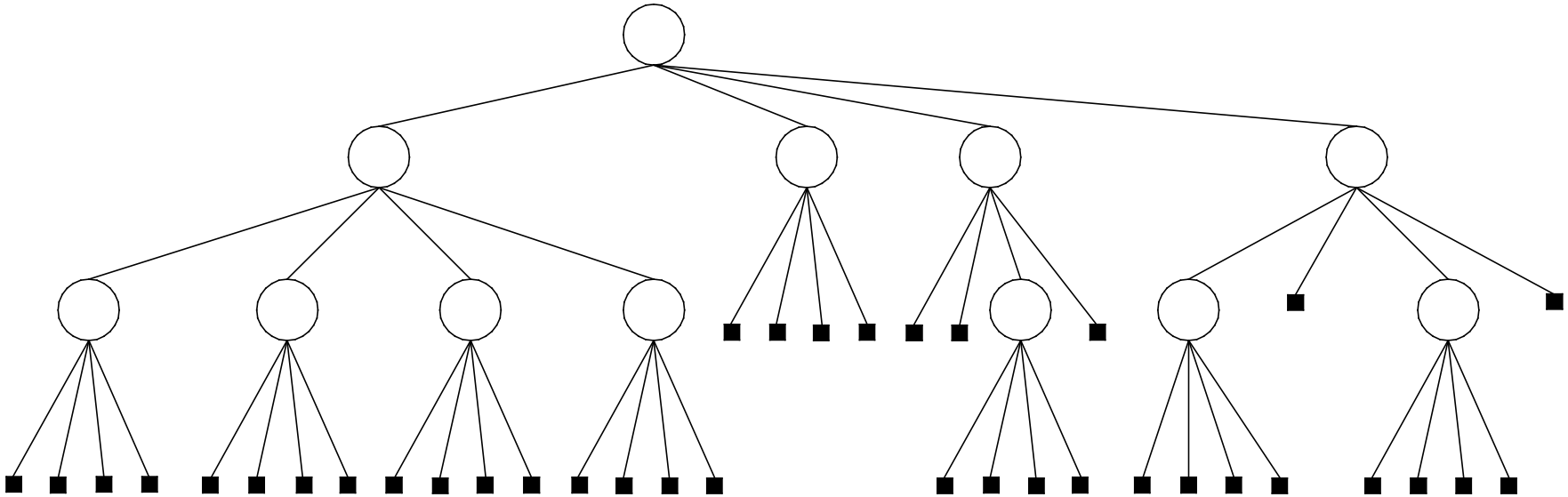
Ejemplo: raíz + cuatro hijos en el primer nivel + diferentes números de hijos en los siguientes niveles



Árboles N-arios

primer elemento (raíz)

Ej: N=4



Árboles binarios

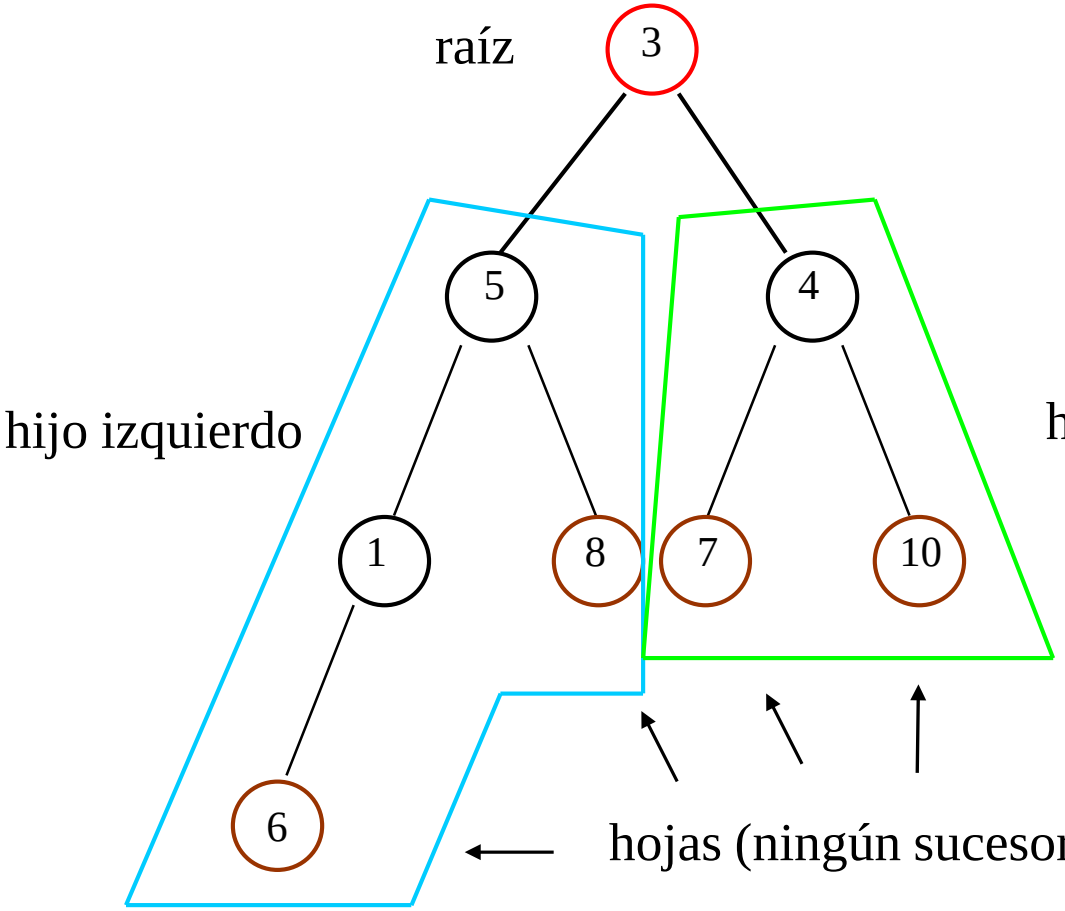
• Estructuras arborescentes en las que cada elemento puede tener un máximo de dos sucesores.

• Un árbol no vacío tiene exactamente dos subárboles, llamados *izquierdo* y *derecho*, aunque alguno de ellos sea vacío .

• Los elementos también se llaman *nodos*

• Se pueden parametrizar mediante `templates`

Componentes de un árbol binario



Tamaño (número de elementos) = 8

Altura (máximo número de elementos entre la raíz y una hoja) = 4

Especificación de árboles binarios

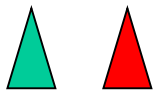
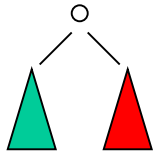
Ver fichero `arbin_espec.hh`

Declaración de árboles nuevos:

`Arbre<int> a;` <- árbol vacío de enteros
`Arbre<int> a1(a);` <- árbol de enteros, copia de `a`

`Arbre<Estudiant> ae;` <- árbol vacío de Estudiant

Plantar, arrel y fills:

a.plantar(∘ ) => a = 

x = a.arrel () => x = ∘

a.fills(a1,a2) =>

a1 =  a2 =  a = árbol vacío

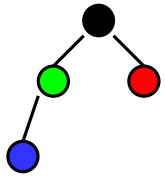
Ejemplos de uso de árboles

- Altura, tamaño, búsquedas, modificar árbol: ver fichero `ejemplos_arbin.cc`
- Recorridos en profundidad y por niveles: ver fichero `recorridos_arbin.cc`

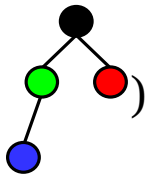
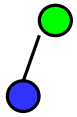
```
int mida(Arbre<int>& a)
/* Pre: a=A */
/* Post: El resultat és el nombre de nodes de l'arbre A */
```

```
// versió expandida
{
  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;
}
```

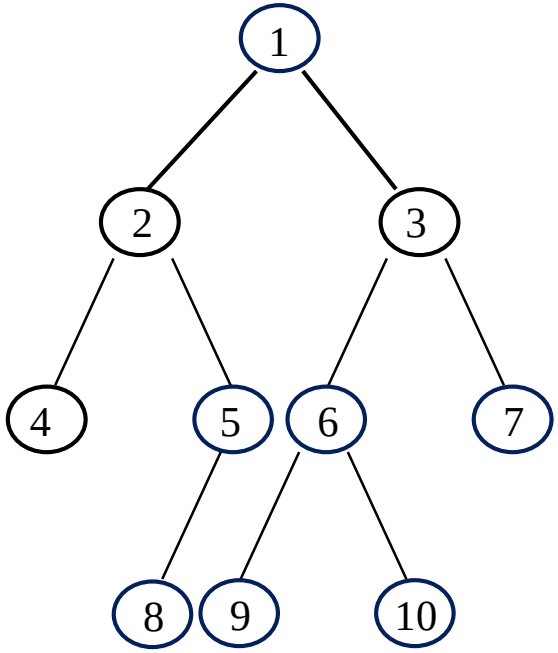
```
// versió compacta
 {
   if (a.es_buit()) return 0;
   Arbre<int> a1,a2;
   a.fills(a1,a2);
   return 1+mida(a1)+mida(a2);
 }
```

a = 

mida (a) = 4

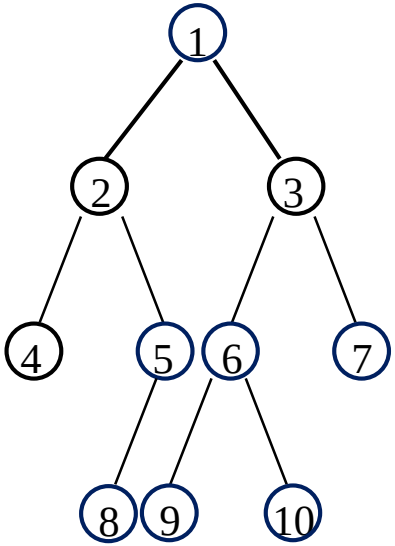
mida() = 1+ mida() + mida( ) = 1+1 + mida( ) + 1 = 4

Recorridos de árboles



Preorden (raiz, hi, hd): 1, 2, 4, 5, 8, 3, 6, 9, 10, 7.
Inorden (hi, raiz, hd): 4, 2, 8, 5, 1, 9, 6, 10, 3, 7.
Postorden (hi, hd, raiz): 4, 8, 5, 2, 9, 10, 6, 7, 3, 1.
Niveles: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Recorridos de árboles - niveles

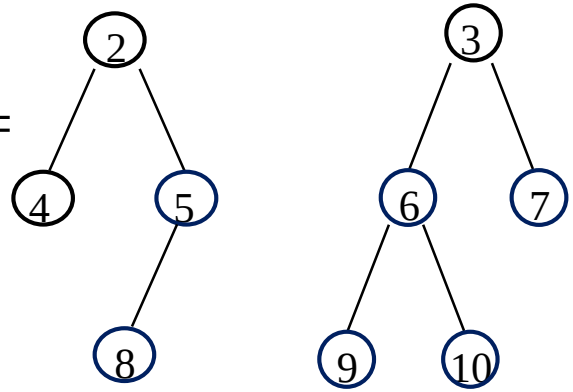


Resultado: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Primera iteración

L = 1

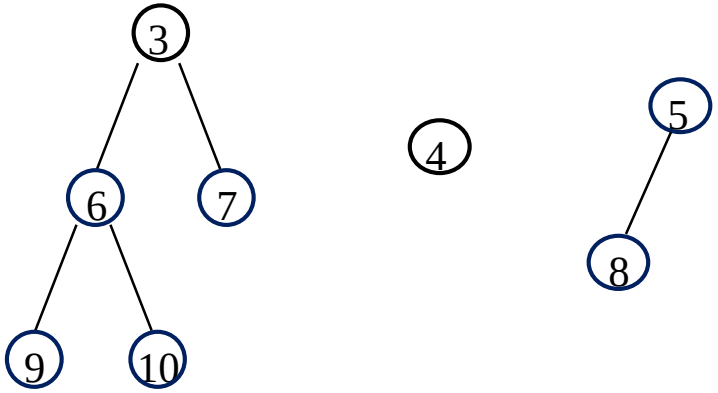
C =



Recorridos de árboles - niveles

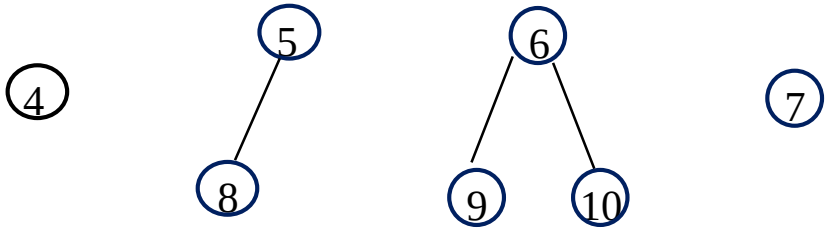
Segunda iteración

L =1 2 C =



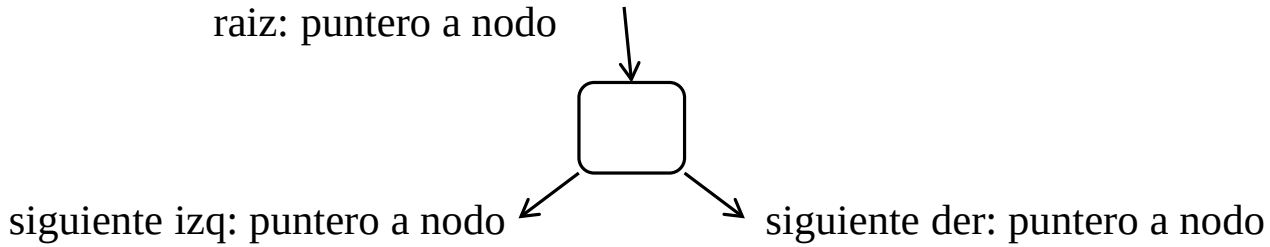
Tercera iteración (etc…)

L = 1 2 3 C =



Implementación de árboles

- Normalmente se implementan con punteros



- En el caso de los binarios, si se sabe que van a ser completos o casi, pueden implementarse con vectores (p.ej. *heaps*)

```
arrel(fe(v[i])) = v[2*i]
arrel(fd(v[i])) = v[2*i+1]
```

Especificación alternativa

• Todo árbol tiene un elemento consultable y modificable (punto de interés)
• Inicialmente, el punto de interés es la raíz
• Hay operaciones para mover el punto de interés hacia el hijo izquierdo, hacia el hijo derecho o hacia el padre
• También hay operaciones para preguntar si el punto de interés es una hoja, etc.