Programación 2
Árboles

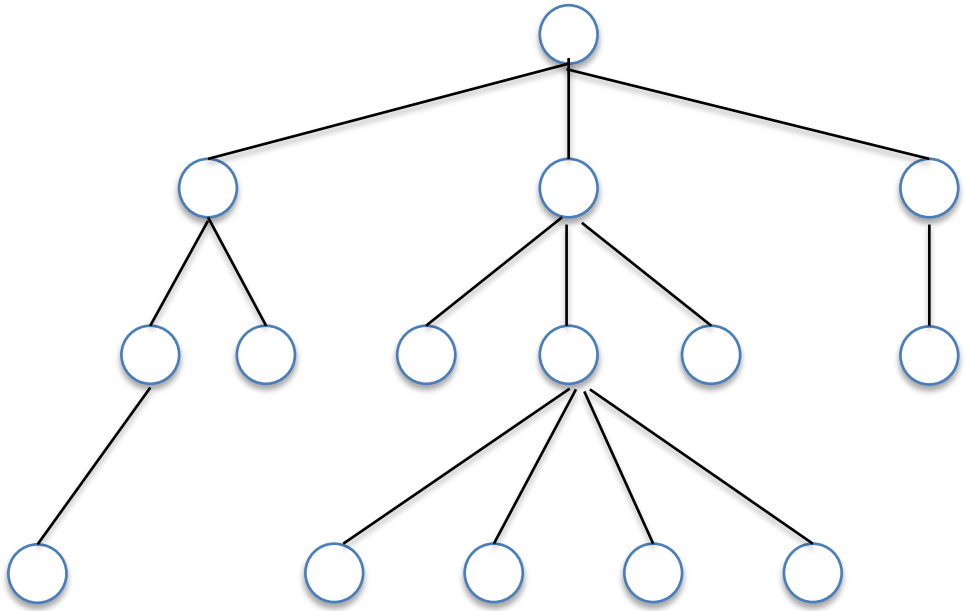
Fernando Orejas

1. Estructuras arborescentes
2. Árboles binarios
3. Recorridos

Estructuras arborescentes

Terminología

- nodo
- padre, hijo
- ascediente
  descendiente
- hermano
- raiz, hoja
- camino
- nivel, altura



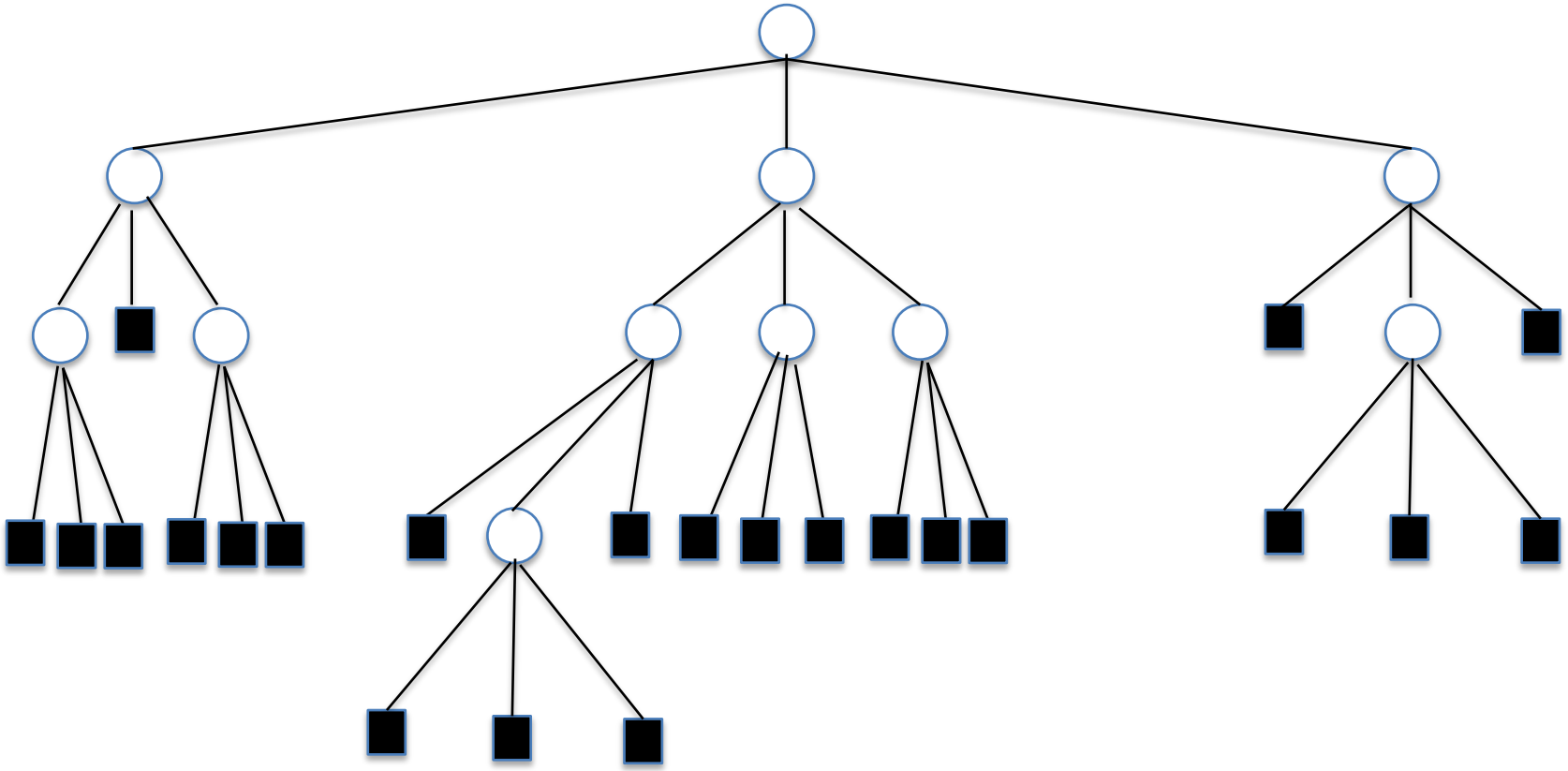
Definiciones

- Grafo dirigido que, o es vacío, o contiene un nodo (la raiz) del que hay un único camino a cada nodo del grafo

- Grafo no dirigido y conexo, con un arco menos que nodos y con un nodo distinguido (la raiz)

- Un árbol es, o bien un árbol vacío, o bien es un nodo con cero o más árboles sucesores

Árboles n-arios



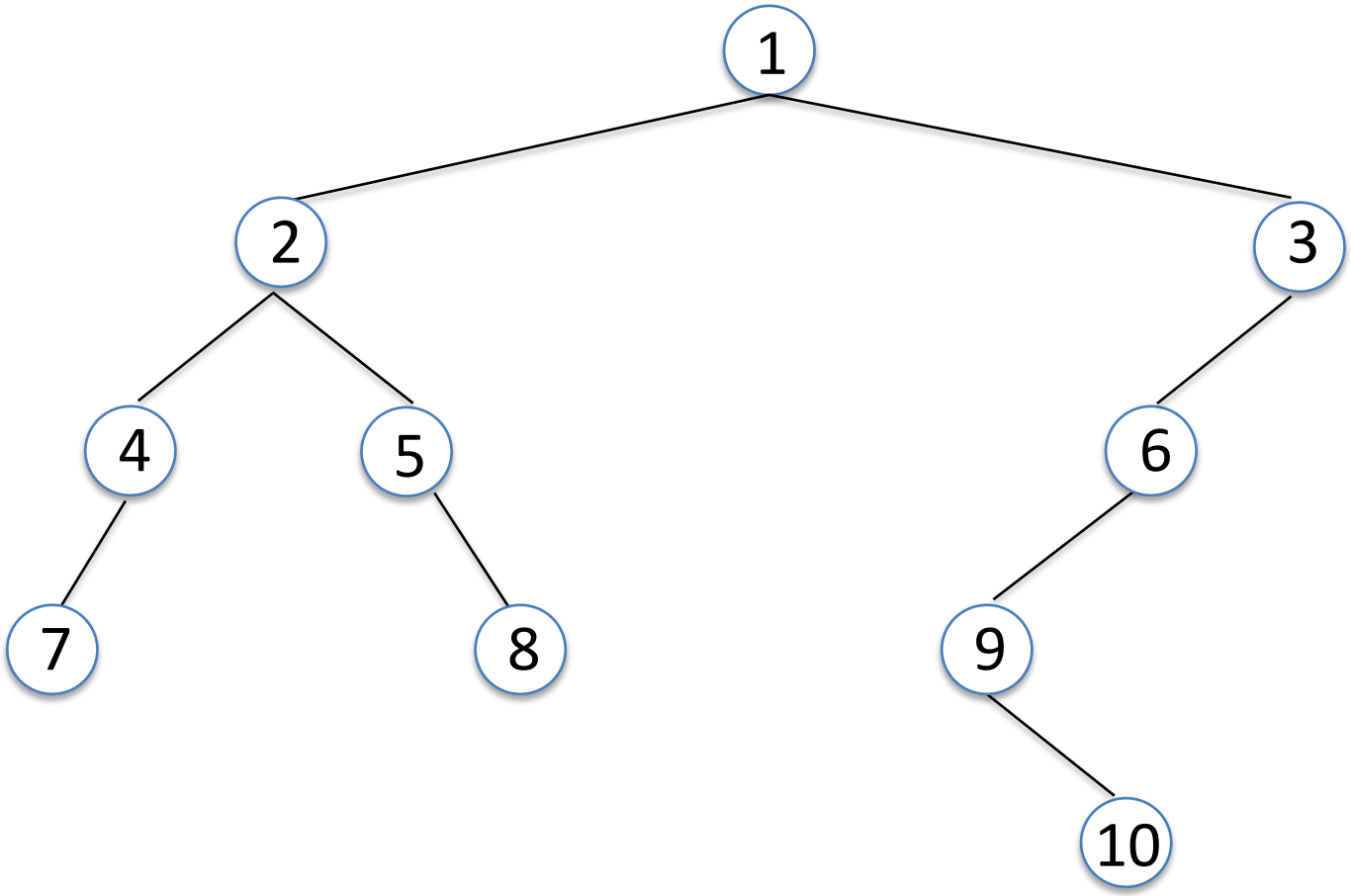
Árboles binarios

Árboles Binarios

- Árboles n-arios con n=2

- Los dos hijos de un nodo son el izquierdo y el derecho

Árboles binarios



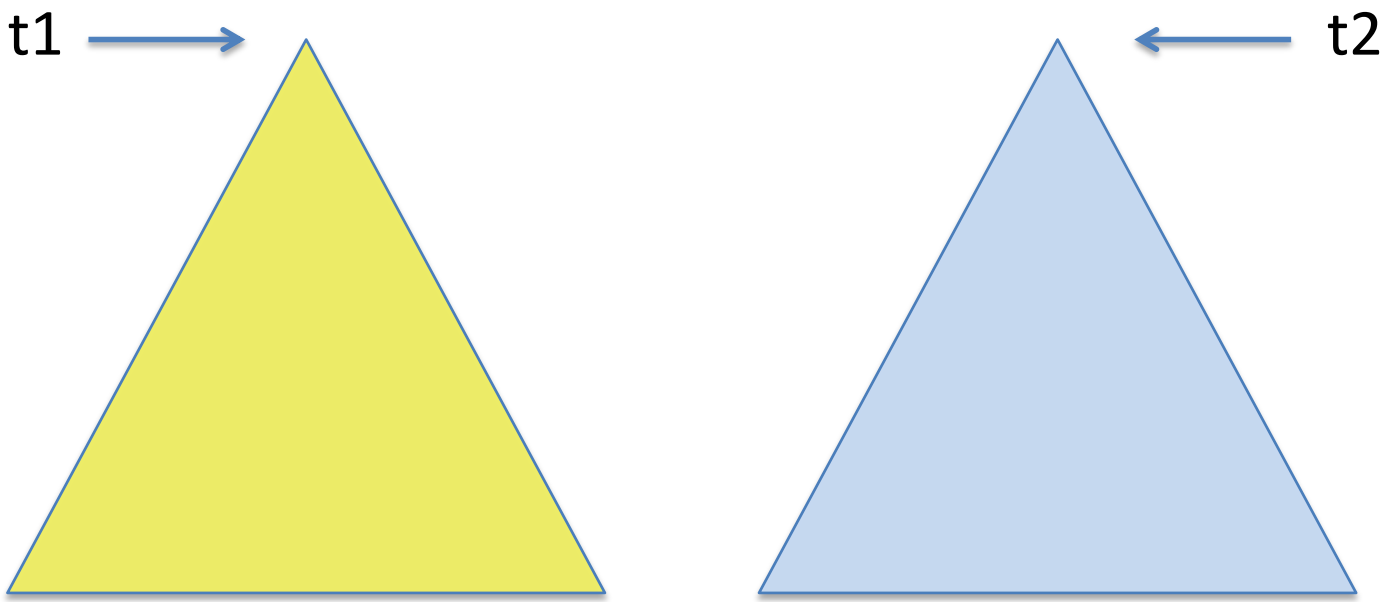
Operaciones de BinTree

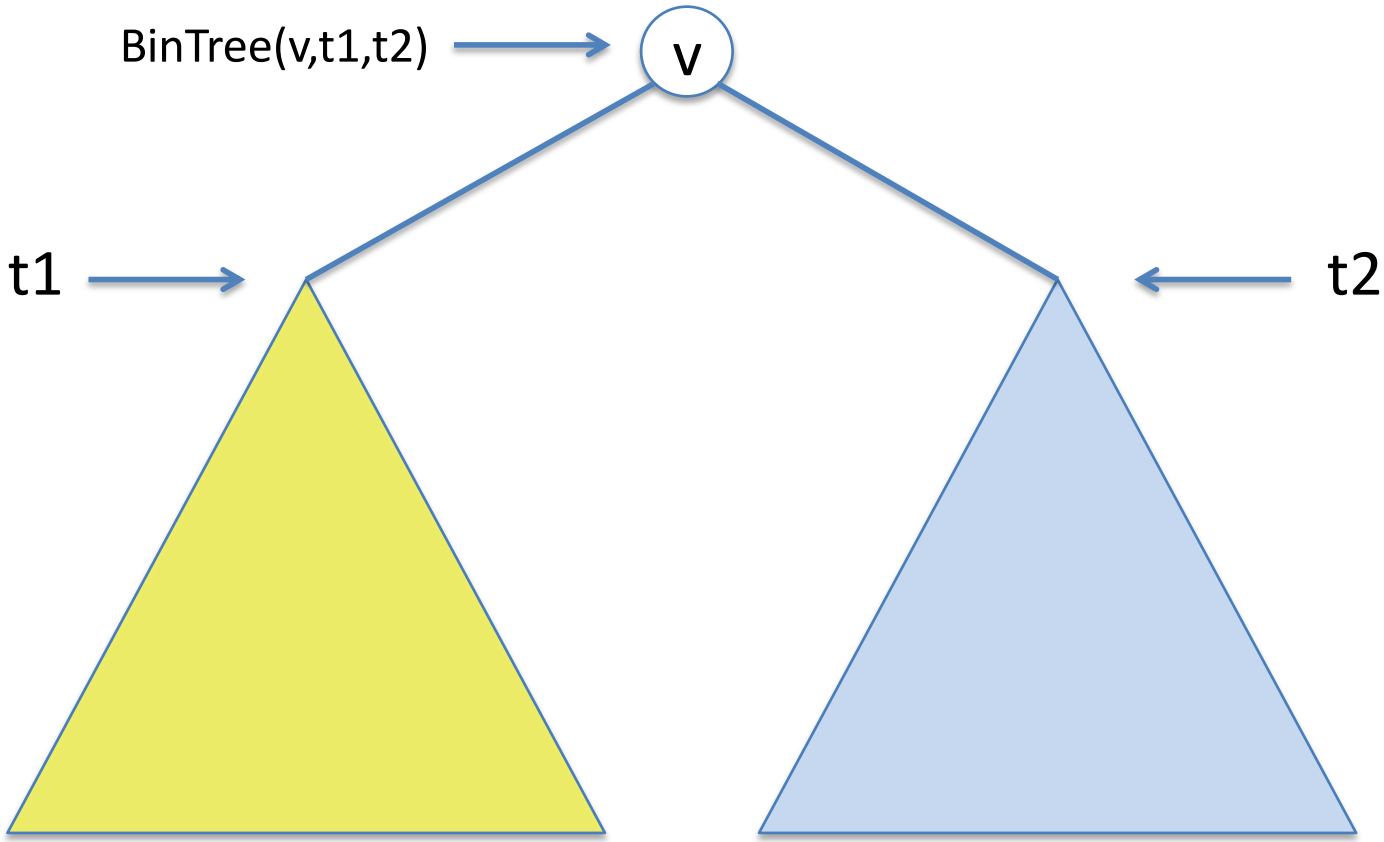
- BinTree no tiene modificadoras: la única manera de modificar un árbol binario es construir un arbol modificado y asignarlo al árbol original.

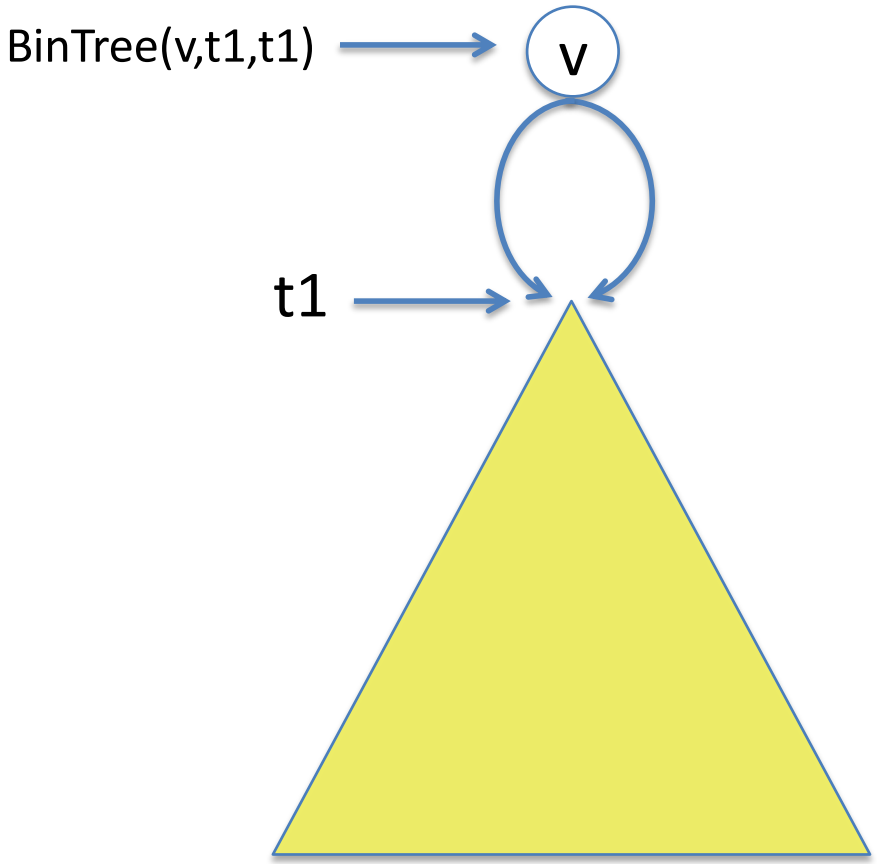
- Todo se hace en tiempo constante (salvo la destrucción)

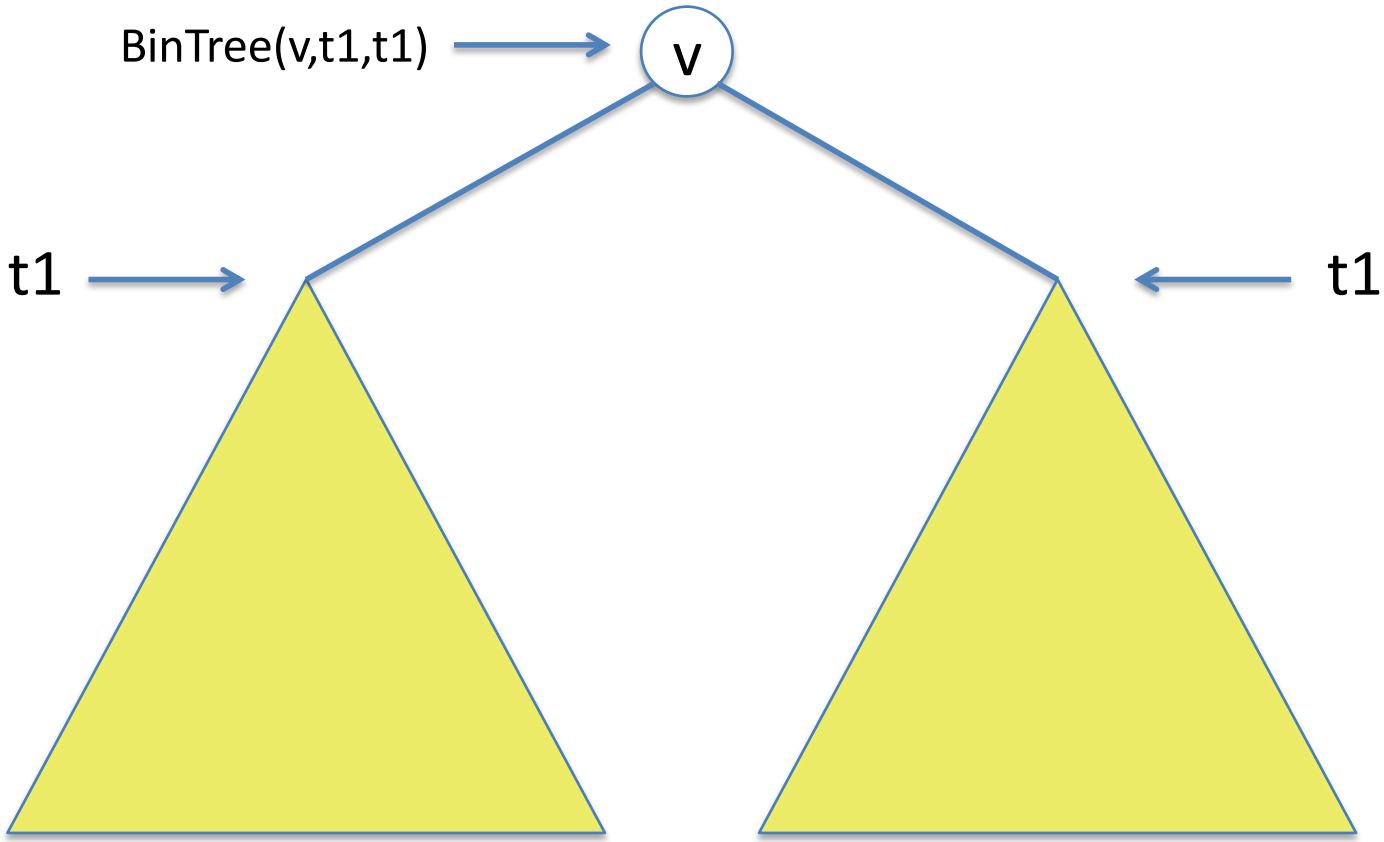
Especificación de la clase BinTree

```
template <class T> class BinTree {
    public:
    // Constructoras
    // Pre: true
    // Post: crea un árbol vacío
    BinTree ();
    // Pre: true
    // Post: crea un árbol con x como raiz, y árboles vacíos
    // como hijos
    BinTree (const T& x);
    // Pre: true
    // Post: crea un árbol con x como raiz, left como hijo
    // izquierdo y right como hijo derecho
    BinTree (const T& x, const BinTree& left, const BinTree&
    right);
```


t1
t2


BinTree(v,t1,t2)
v
t1
t2


BinTree(v,t1,t1)
v
t1


BinTree(v,t1,t1)
v
t1
t1

```
// Destructora
~BinTree();

// Consultoras   // Pre: true
// Post: Retorna true si el árbol y false en caso
// contrario
bool empty ();

// Pre: El parámetro implícito no está vacío
// Post: retorna el hijo izqdo del parámetro implícito
BinTree left ();

// Pre: El parámetro implícito no está vacío
// Post: retorna el hijo dcho del parámetro implícito
BinTree right ();

// Pre: El parámetro implícito no está vacío
// Post: retorna la raiz del parámetro implícito
BinTree value ();
```

Tamaño de un árbol

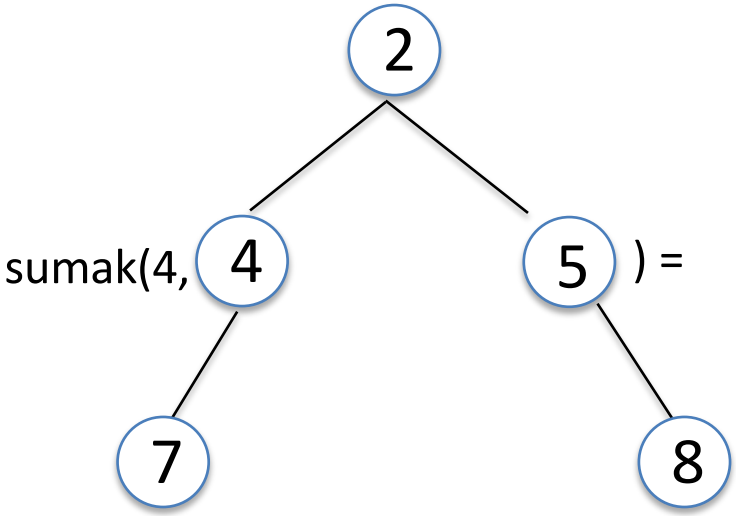
```
/* Pre: true */
/* Post: retorna el número de nodos del árbol t*/
int size(const BinTree <int>& t){
   if (t.empty()) return 0;
   else return 1 + size(t.left()) + size(t.right());
}
```

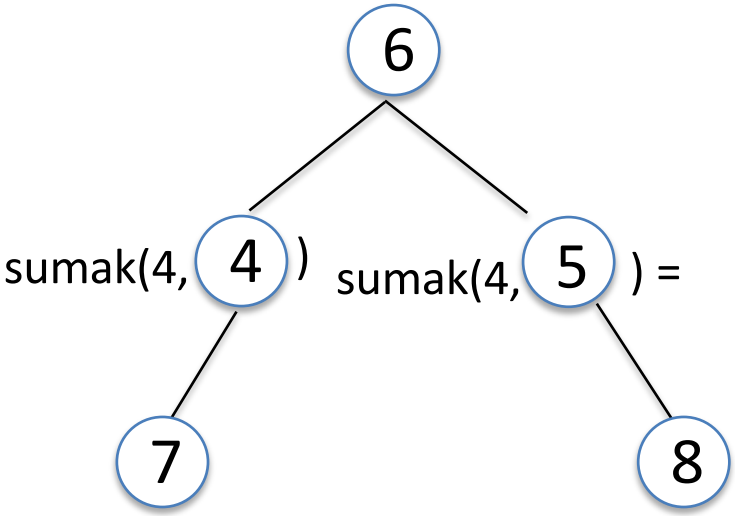
Búsqueda en un árbol

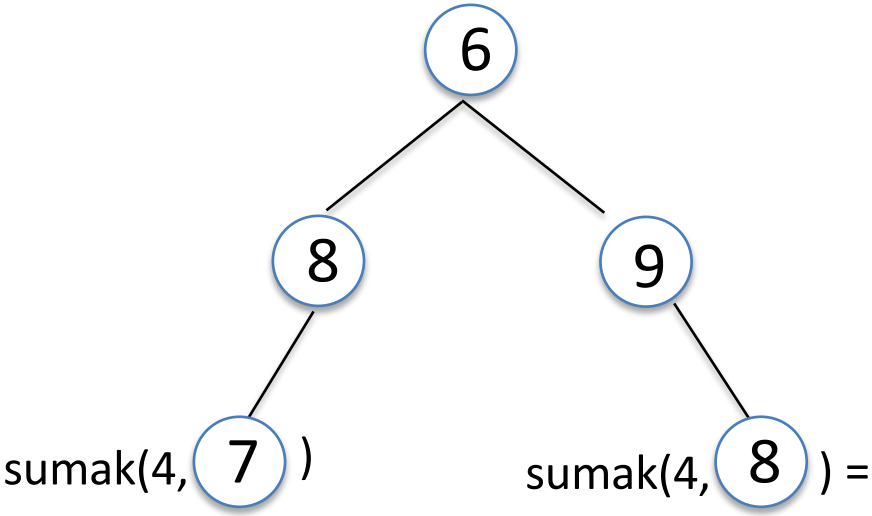
```
/* Pre: true */
/* Post: nos dice si x está en t*/
bool busq(const BinTree <int>& t, int x){
   if (t.empty()) return false;
   else
      return (t.value() == x) or busq(t.left(),x) or
         busq(t.right(),x);
}
```

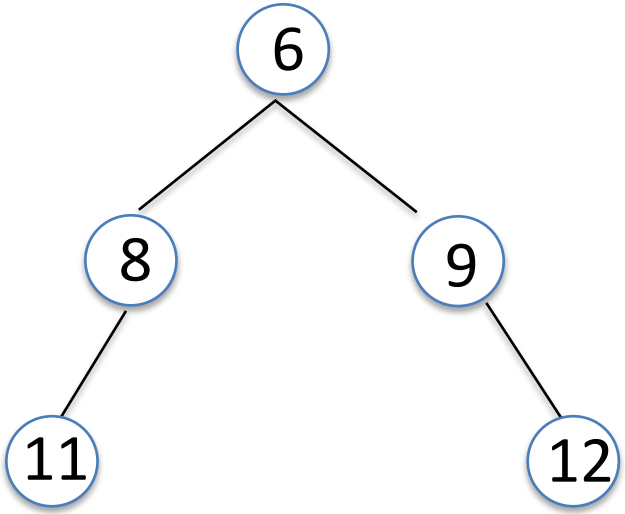
Suma k a todos los valores de un árbol

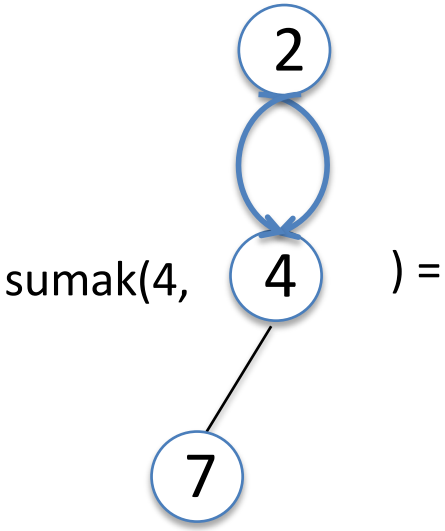
```
/* Pre: true */
/* Post: retorna un arbol t' con la misma forma que t,
tal que  el valor de cada nodo de t' es igual a k + el
valor del nodo correspondiente de t */
BinTree sumak(const BinTree <int>& t, int k){
   if (t.empty()) return t;
   else return BinTree(t.value()+k,
          sumak(t.left(),k),
          sumak(t.right(),k));
}
```

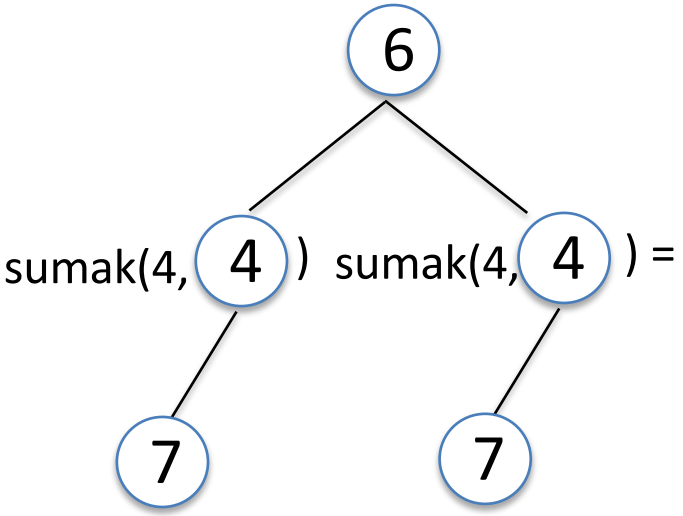

2
sumak(4,
4
5
) =
7
8

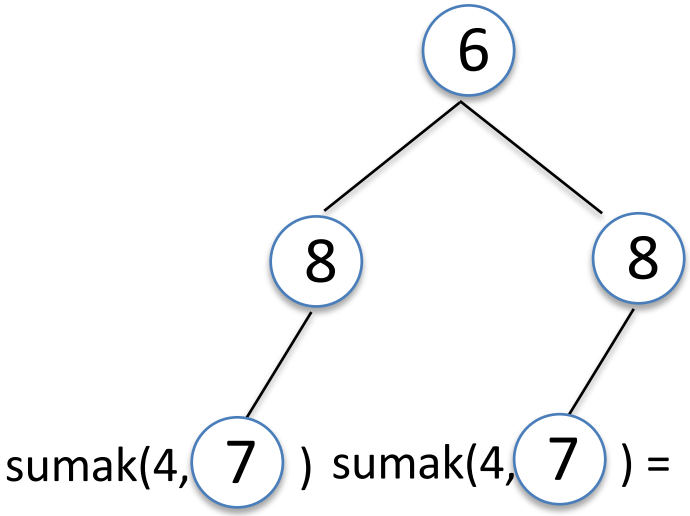

6
sumak(4,
4
)
sumak(4,
5
) =
7
8

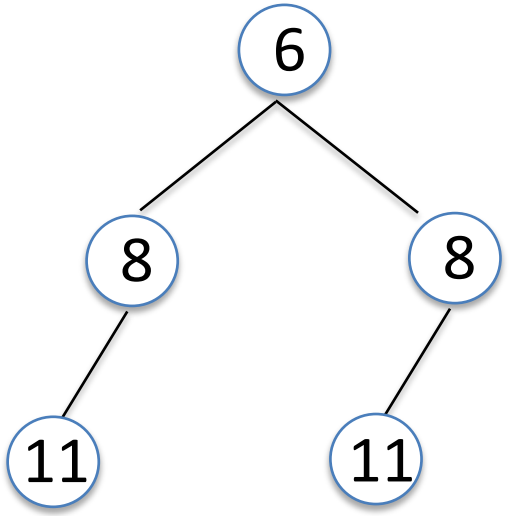

6
8
9
sumak(4,
7
)
sumak(4,
8
) =


6
8
9
11
12


2
sumak(4,
4
) =
7


6
sumak(4,
4
) sumak(4,
4
) =
7
7


6
8
8
sumak(4,
7
) sumak(4,
7
) =


6
8
8
11
11

Suma k a todos los valores de un árbol

```
/* Pre: true */
/* Post: retorna un arbol t' con la misma forma que t,
tal que  el valor de cada nodo de t' es igual a k + el
valor del nodo correspondiente de t */
void sumak(BinTree <int>& t, int k){
   if (not t.empty()){
      BinTree <int> i = t.left();
      BinTree <int> d = t.right();
      sumak(i,k);
      sumak(d,k);
      t = BinTree(t.value()+k, i, d);
   }
}
```

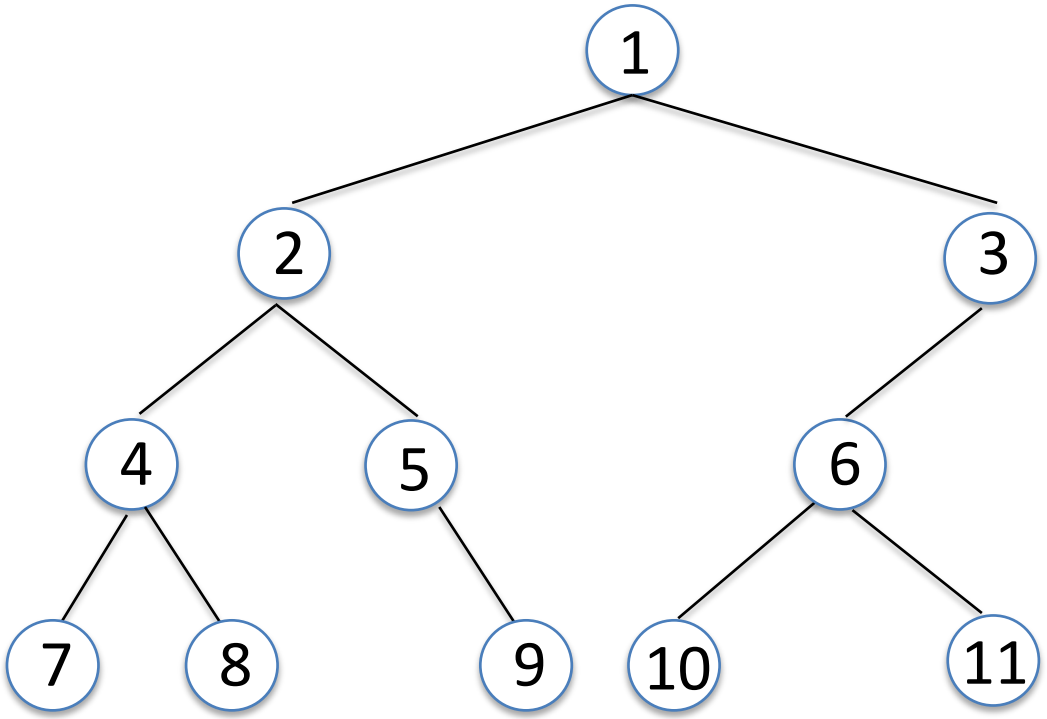
Recorridos

Recorridos de árboles

- En profundidad
  - Preorden
  - Postorden
  - inorden
- En amplitud (o por niveles)

Recorrido en preorden

1. Visitamos la raiz
2. Recorremos en preorden el hijo izquierdo
3. Recorremos en preorden el hijo derecho

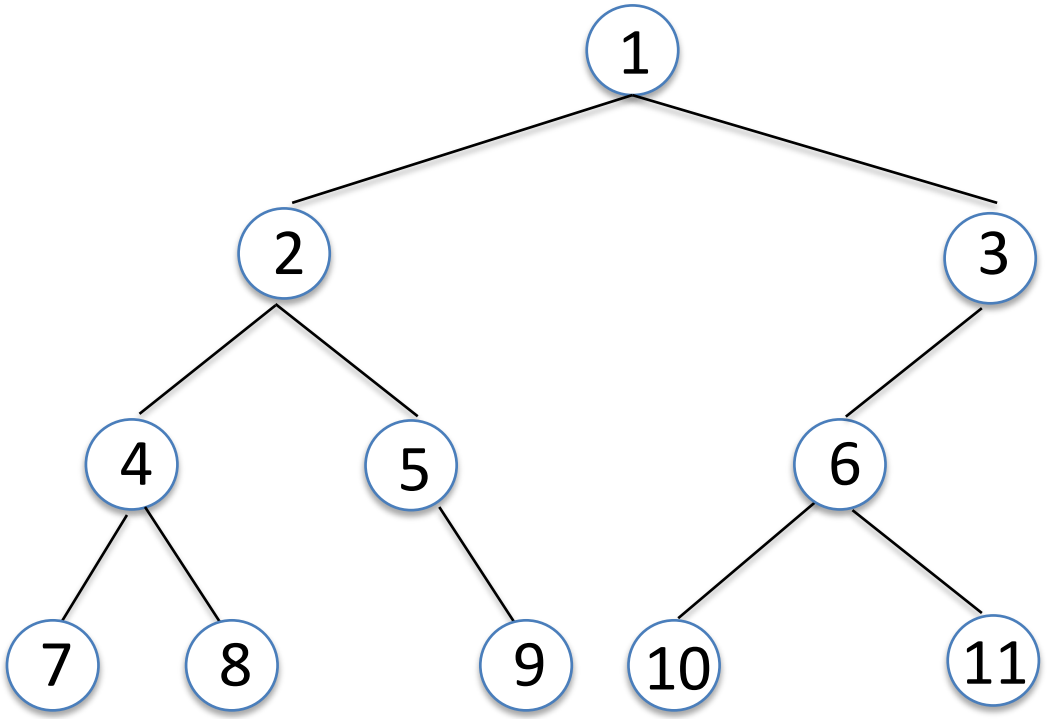


- Recorrido

1 2 4 7 8 5 9 3 6 10 11

Recorrido en postorden

1. Recorremos en postorden el hijo izquierdo
2. Recorremos en postorden el hijo derecho
3. Visitamos la raiz

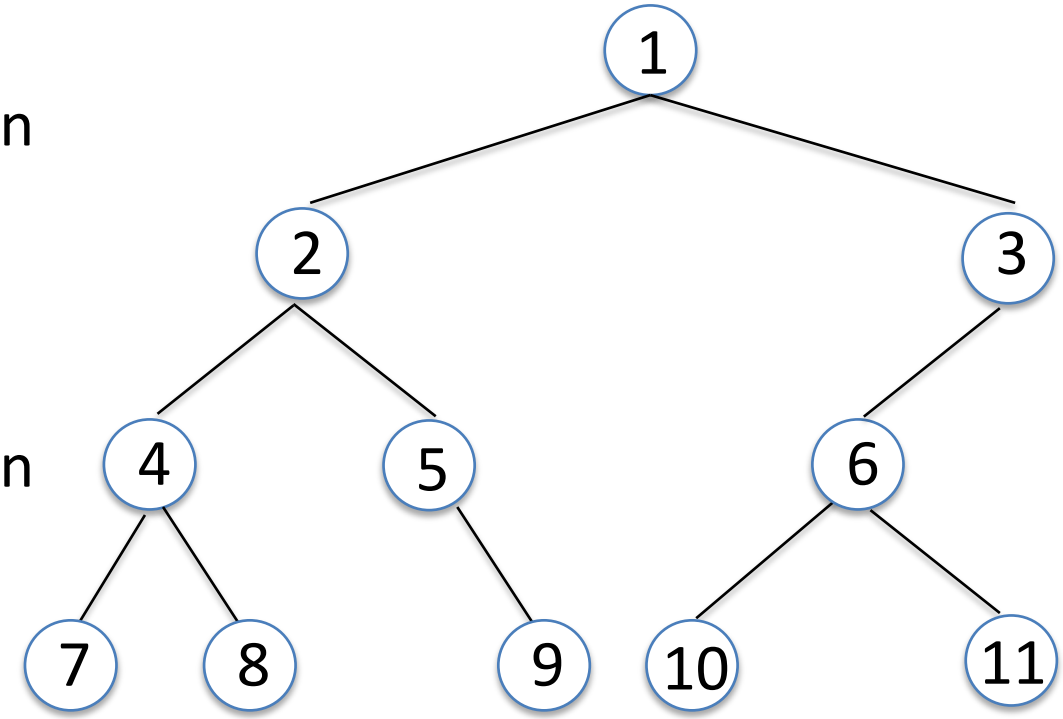


- Recorrido

7 8 4 9 5 2 10 11 6 3 1

Recorrido en inorden

1. Recorremos en inorden el hijo izquierdo
2. Visitamos la raiz
3. Recorremos en inorden el hijo derecho



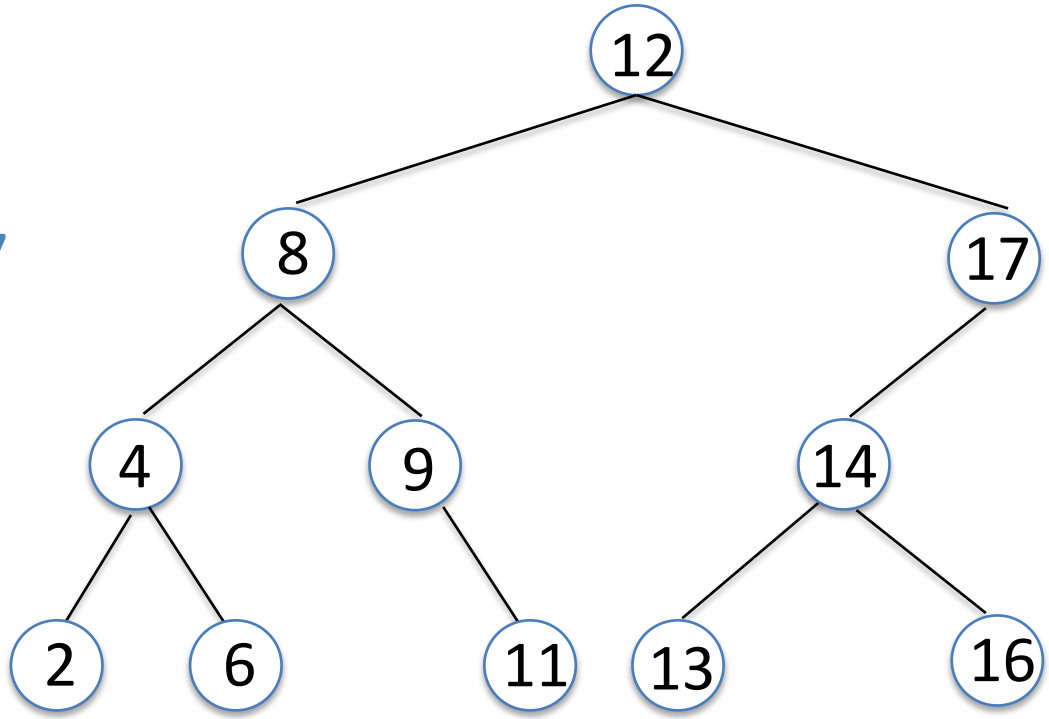
- Recorrido

7 4 8 2 5 9 1 10 6 11 3

Recorrido en inorden

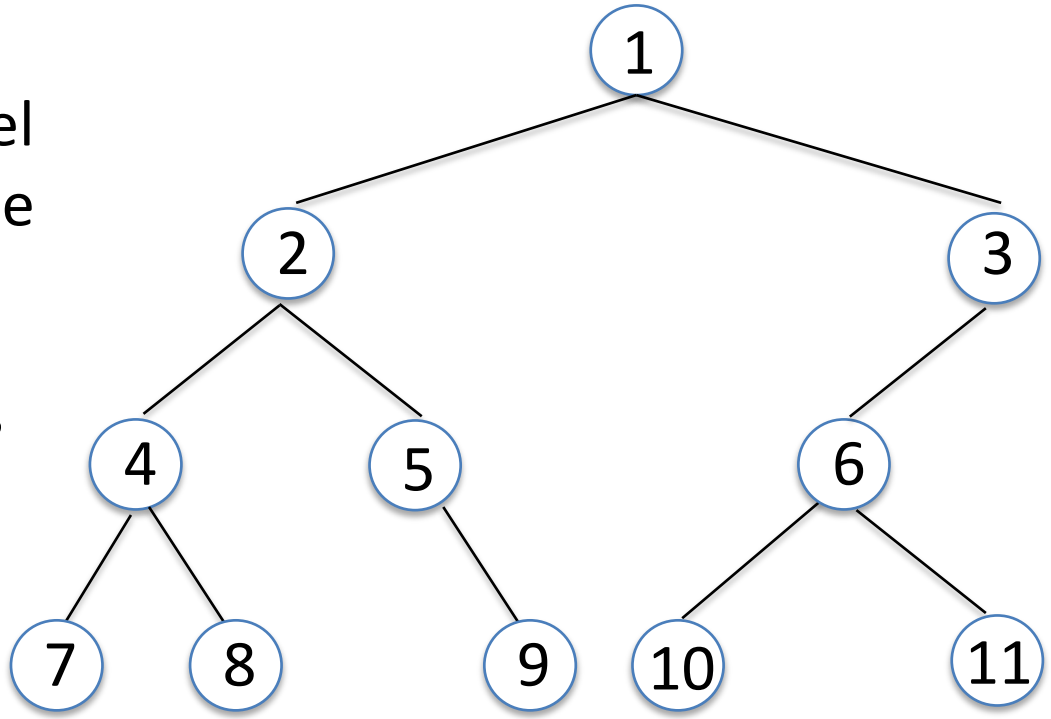
Recorrido

2 4 6 8 9 11 12 13 14 16 17



Recorrido por niveles

- Todos los nodos del nivel k son visitados antes que los del nivel k+1
- En cada nivel, los nodos se visitan de izquierda a derecha

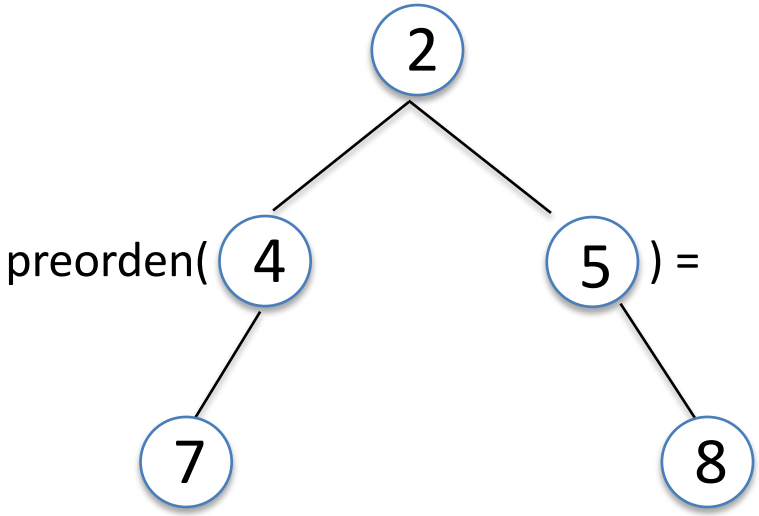


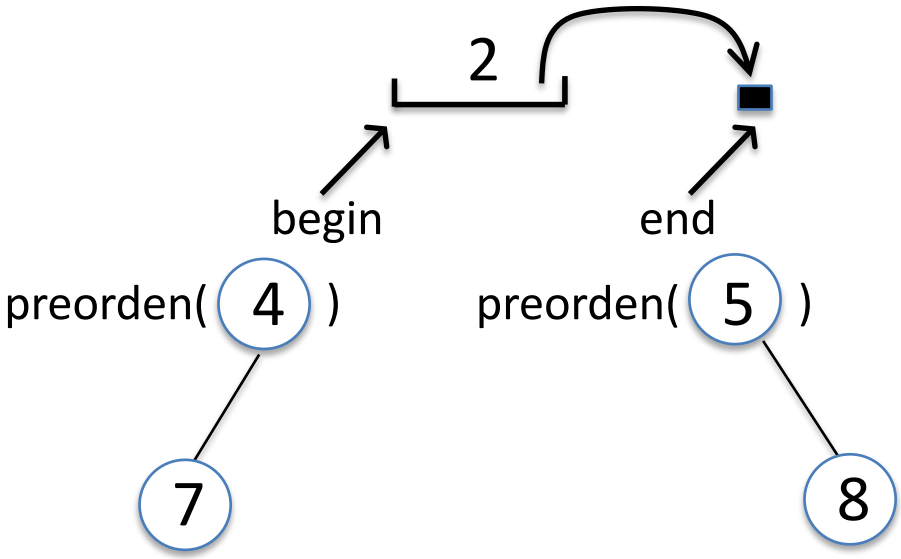
- Recorrido

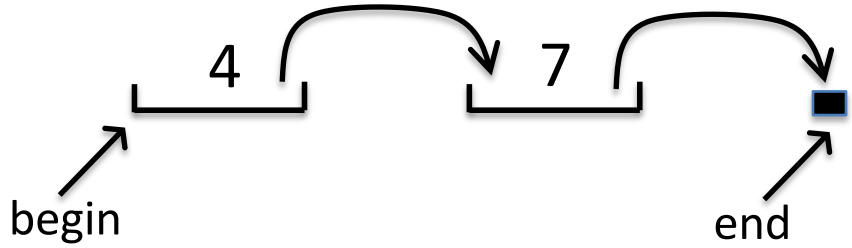
1 2 3 4 5 6 7 8 9 10 11

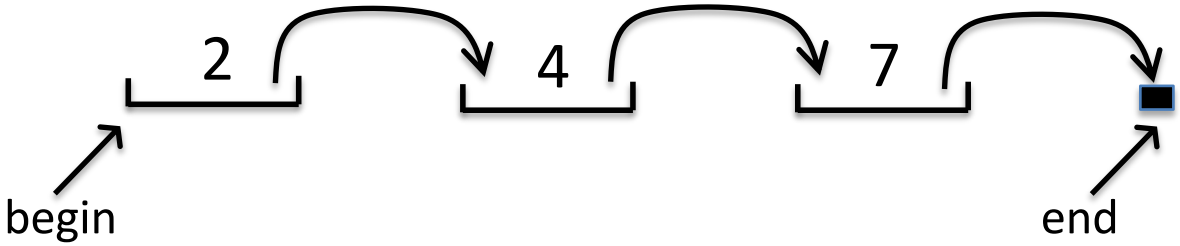
Recorrido en preorden

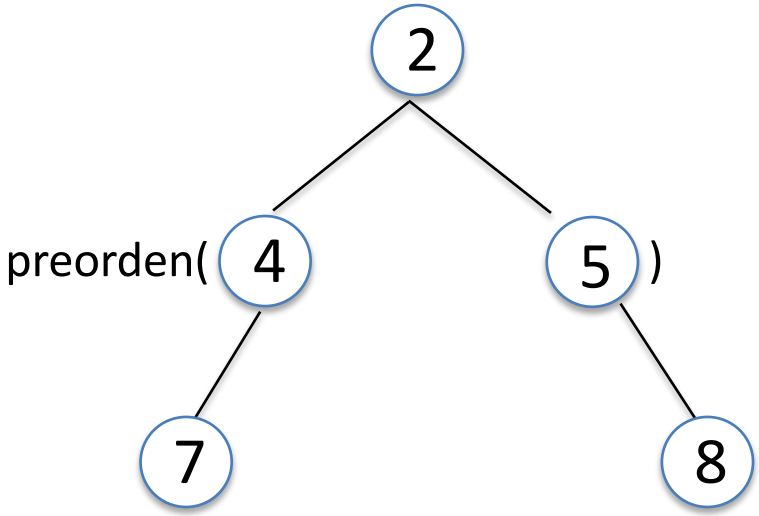
```
/* Pre: true */
/* Post: El resultado es la lista en preorden de los
elementos de t */
list<int> preorden(const BinTree <int>& t,) {
    list<int> L;
    if (not t.empty()) {
        L.insert(L.begin(), t.value()),
        L.splice(L.end(), preorden(t.left())),
        L.splice(L.end(), preorden(t.right()));
    return L;
}
```

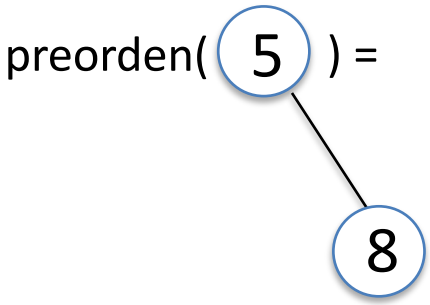

2
preorden(
4
5
) =
7
8

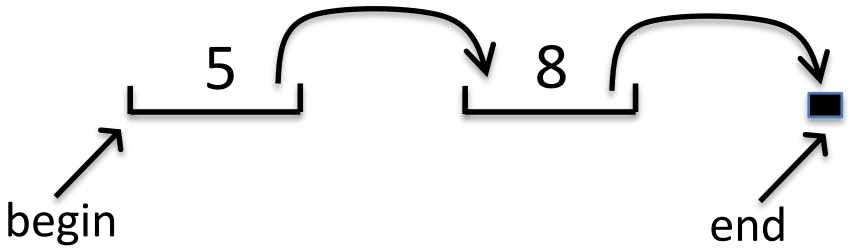

2
begin
end
preorden(
4
)
preorden(
5
)
7
8

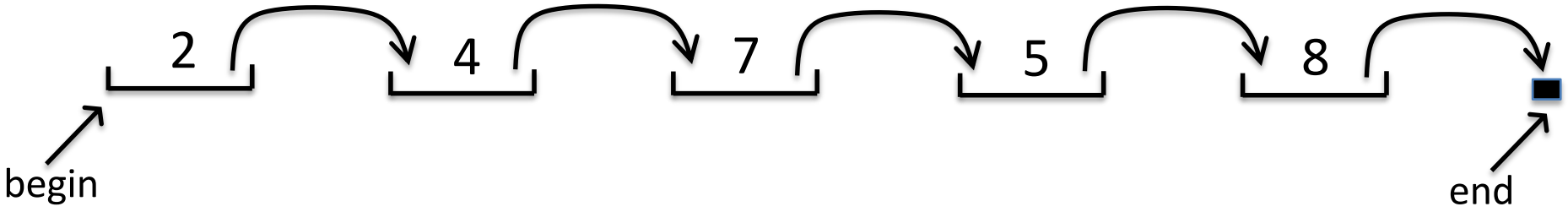

4
7
begin
end


2
4
7
begin
end


2
preorden(4
5)
7
8


preorden(5) =
8

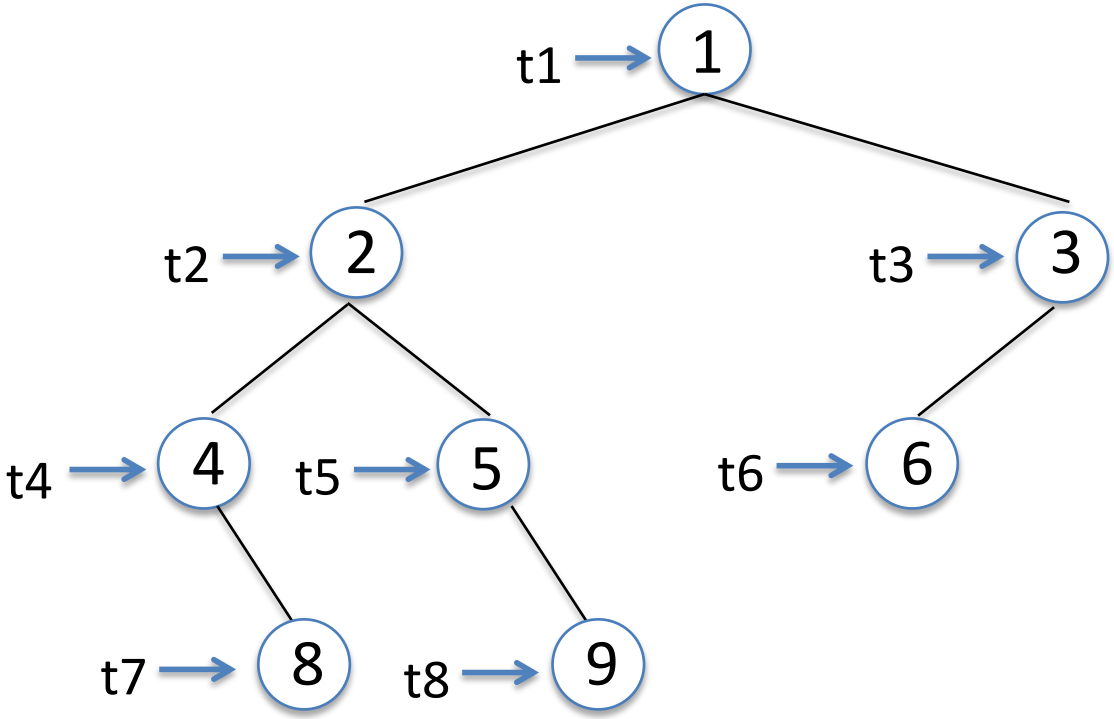

5
8
begin
end

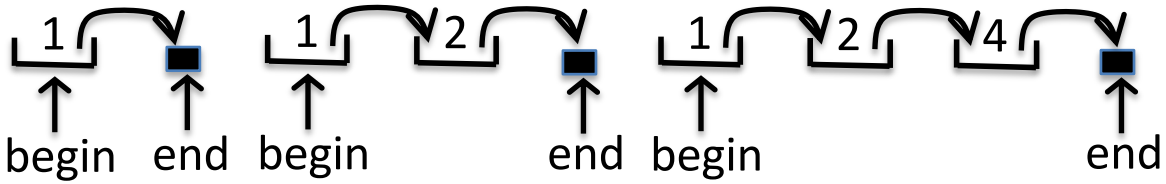

2
4
7
5
8
begin
end

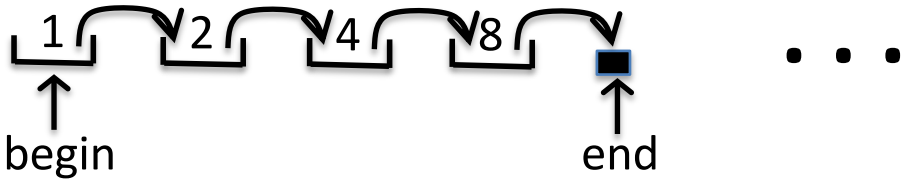
Recorrido en preorden (iterativo)

```
/* Pre: true */
/* Post: El resultado es la lista de los elementos de t
recorridos en preorden */
list<int> preorden (const BinTree <int>& t,) {
    list <int> L;
    if (not t.empty()) {
        stack <BinTree <int>> s; s.push(t);
        while (not s.empty()) {
            BinTree <int> aux = s.top(); s.pop();
            L.insert(L.end(), aux.value()),
            if (not aux.right().empty()) s.push(aux.right()),
            if (not aux.left().empty()) s.push(aux.left());
        }
    }
    return L;
}
```

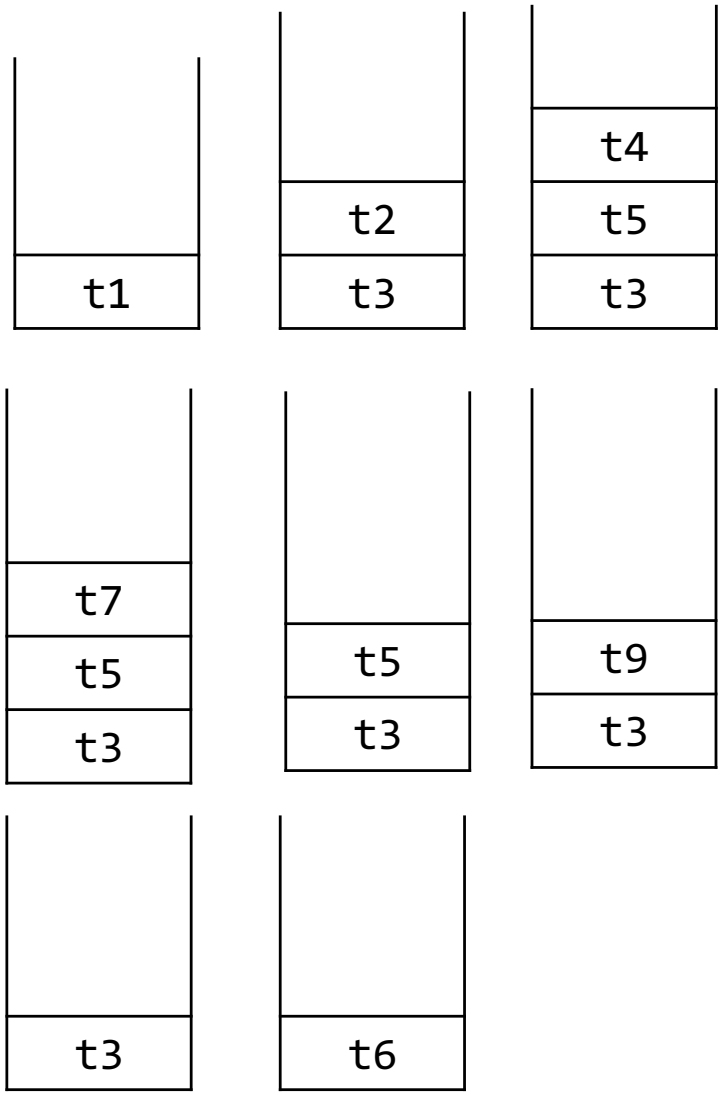
Preorden iterativo usando una pila







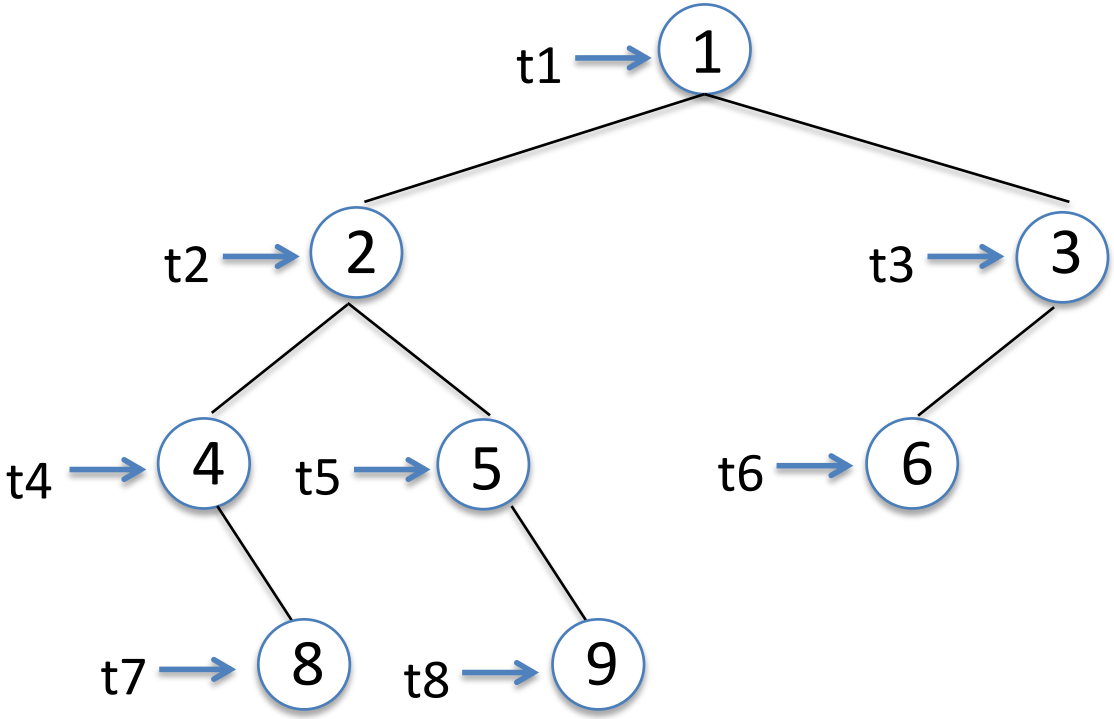
• • •

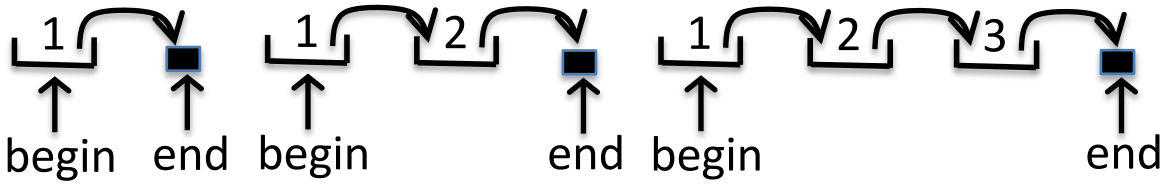


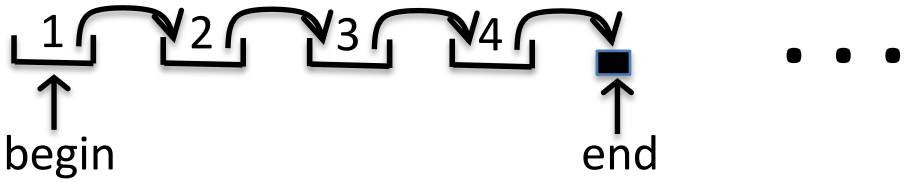
Recorrido por niveles

```
/* Pre: true */
/* Post: El resultado es la lista de los elementos de t
recorridos por niveles */
list<int> niveles (const BinTree <int>& t,) {
    list <int> L;
    if (not t.empty()) {
        queue <BinTree <int>> q; q.push(t);
        while (not q.empty()) {
            BinTree <int> aux = q.front(); q.pop();
            L.insert(L.end(), aux.value()),
            if (not aux.left().empty()) q.push(aux.left()),
            if (not aux.right().empty()) q.push(aux.right());
        }
    }
    return L;
}
```

Recorrido por niveles







• • •

t1			
t2	t3		
t3	t4	t5	
t4	t5	t6	
t5	t6	t7	
t6	t7	t8	
t7	t8		
t9			