ESTRUCTURAS DE DATOS

TIPOS ABSTRACTOS DE DATOS ARBORESCENTES

Compartición en árboles binarios

Manuel Montenegro Montes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

Definición actual de `TreeNode`

```
struct TreeNode {
  T elem;
  TreeNode *left, *right;

  TreeNode(const TreeNode *left,
           const T &elem,
           const TreeNode *right)
    : elem(elem), left(left),
      right(right) { }
};
```

- Vamos a sustituir los punteros estándar de C++ por *smart pointers*.
- En lugar de

  `TreeNode *`

  utilizamos

  `std::shared_ptr<TreeNode>`

Cambios en TreeNode

```
struct TreeNode {
  T elem;
  std::shared_ptr<TreeNode> left, right;

  TreeNode(const std::shared_ptr<TreeNode> &left,
           const T &elem,
           const std::shared_ptr<TreeNode> &right)
    : elem(elem), left(left),
      right(right) { }
};
```

Cambios en TreeNode

```
using NodePointer = std::shared_ptr<TreeNode>;

struct TreeNode {
  T elem;
  NodePointer left, right;

  TreeNode(const NodePointer &left,
           const T &elem,
           const NodePointer &right)
    : elem(elem), left(left),
      right(right) { }
};
```

Cambios en BinTree

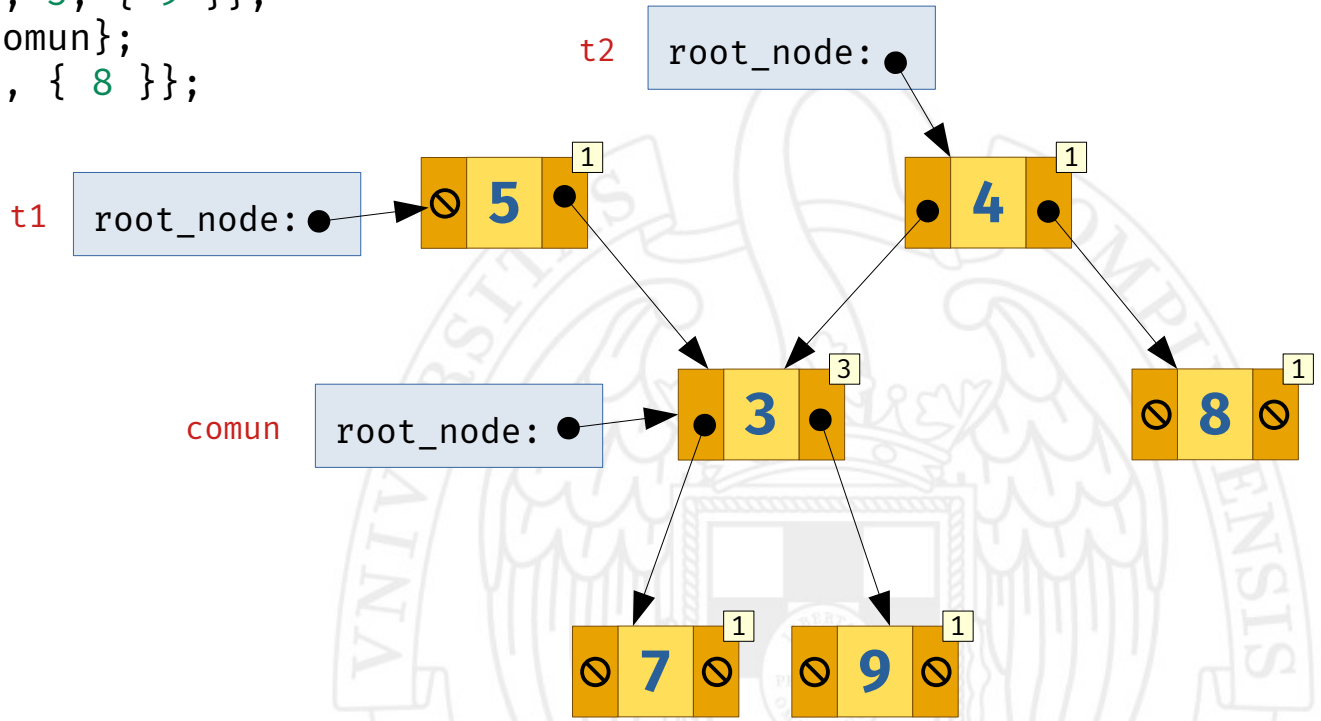
```
template<class T>
class BinTree {
public:

  BinTree(): root_node(nullptr) { }
  BinTree(const T &elem)
    : root_node(std::make_shared<TreeNode>(nullptr, elem, nullptr)) { }
  BinTree(const BinTree &left, const T &elem, const BinTree &right)
    : root_node(std::make_shared<TreeNode>(left.root_node, elem, right.root_node)) { }

  …

private:
  using NodePointer = std::shared_ptr<TreeNode>;
  struct TreeNode { … }

  NodePointer root_node;

  static void display_node(const NodePointer &root, std::ostream &out) { … }
};
```

No necesitamos...

- Destructor
  - Cuando se elimina un objeto `BinTree` se llama automáticamente al destructor de `root_node`.
  - El destructor de `root_node` decrementa el contador de referencias del nodo raíz, y lo libera, en caso de llegar a 0.
- Constructor de copia
  - El constructor de copia por defecto `BinTree` nos sirve, ya que llama al constructor de copia de `root_node`.
  - El constructor de copia de `root_node` incrementa el contador de referencias del nodo raíz.

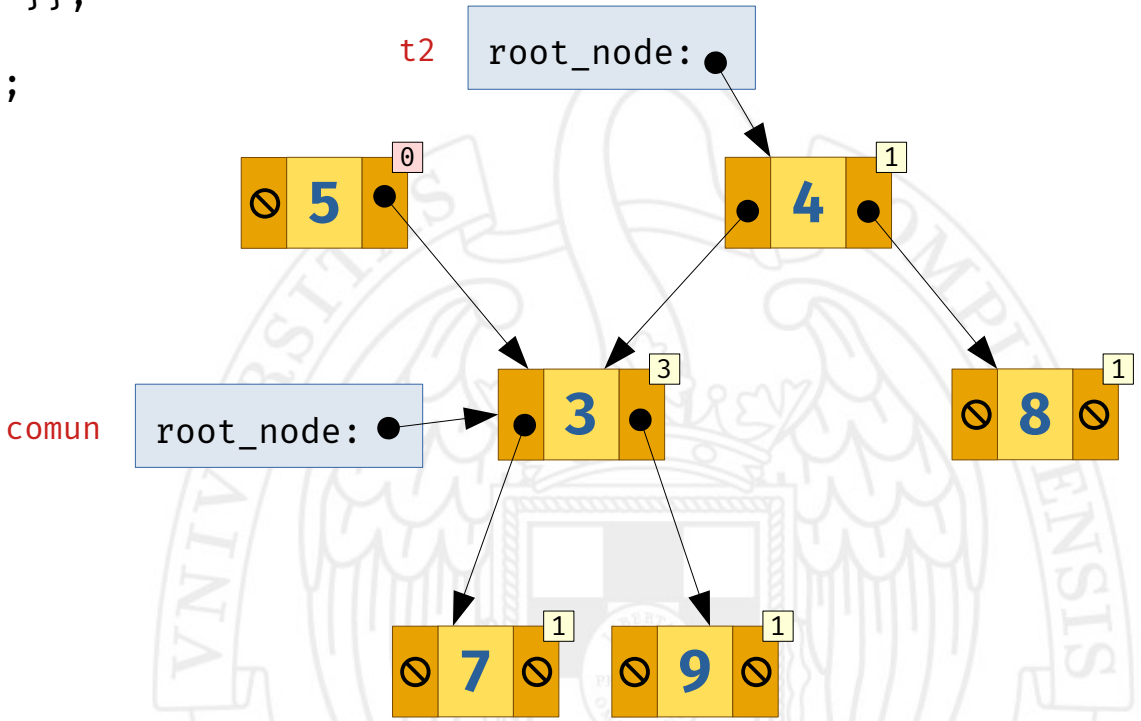
Ejemplo

```
BinTree<int> comun = {{ 7 }, 3, { 9 }};
BinTree<int> t1 = {{}, 5, comun};
BinTree<int> t2 = {comun, 4, { 8 }};
```



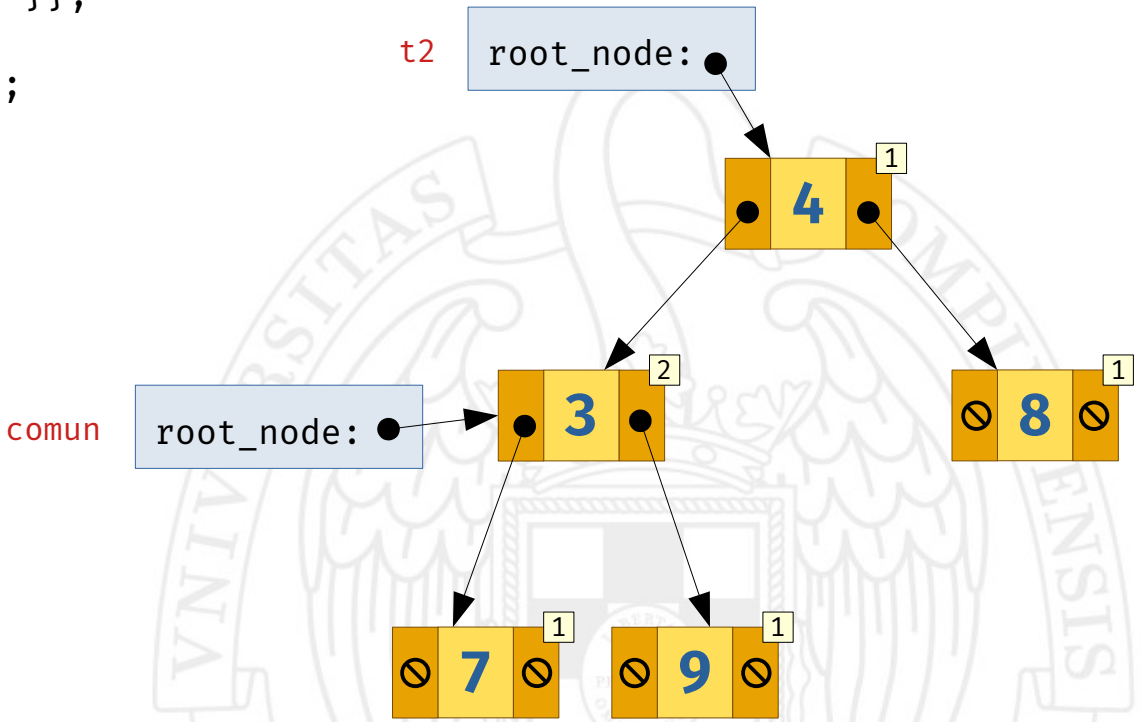
Ejemplo

```
BinTree<int> comun = {{ 7 }, 3, { 9 }};
BinTree<int> t1 = {{}, 5, comun};
BinTree<int> t2 = {comun, 4, { 8 }};
```



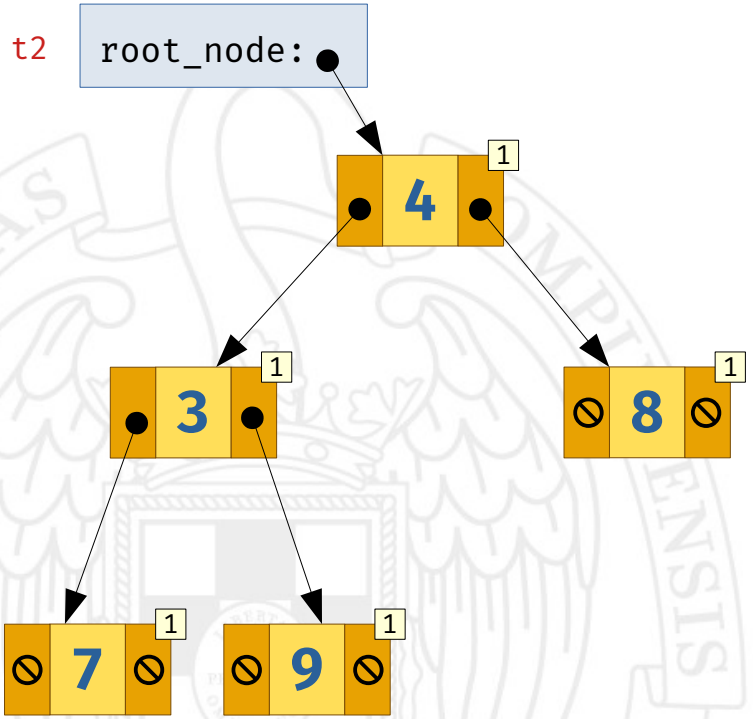
Ejemplo

```
BinTree<int> comun = {{ 7 }, 3, { 9 }};
BinTree<int> t1 = {{}, 5, comun};
BinTree<int> t2 = {comun, 4, { 8 }};
```



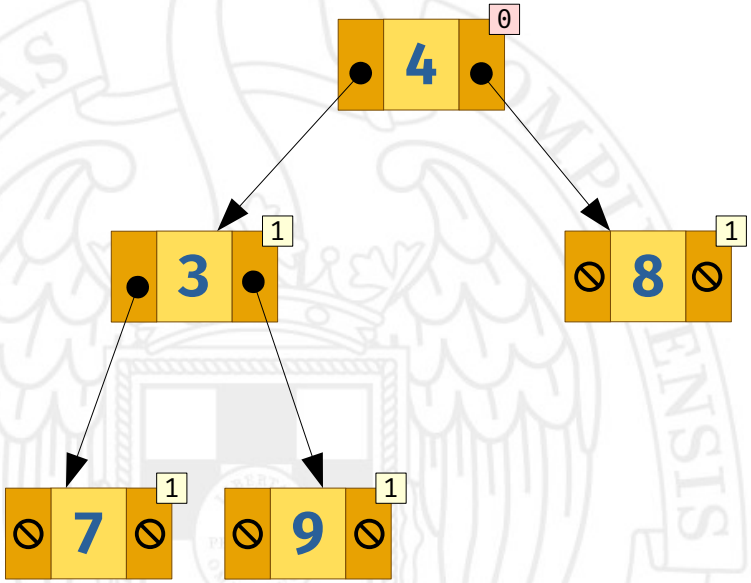
Ejemplo

```
BinTree<int> comun = {{ 7 }, 3, { 9 }};
BinTree<int> t1 = {{}, 5, comun};
BinTree<int> t2 = {comun, 4, { 8 }};
```



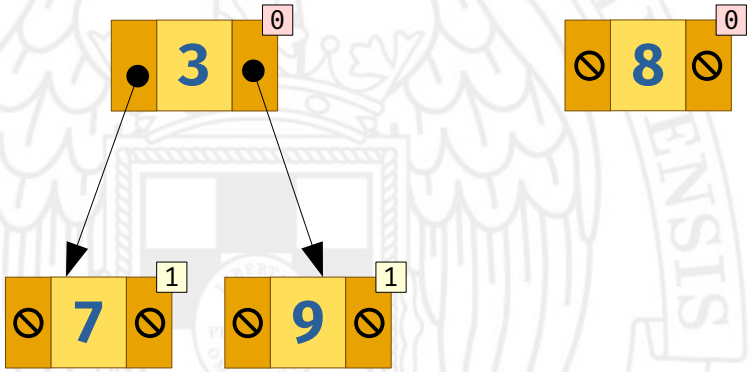
Ejemplo

```
BinTree<int> comun = {{ 7 }, 3, { 9 }};
BinTree<int> t1 = {{}, 5, comun};
BinTree<int> t2 = {comun, 4, { 8 }};
```



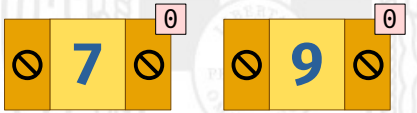
Ejemplo

```
BinTree<int> comun = {{ 7 }, 3, { 9 }};
BinTree<int> t1 = {{}, 5, comun};
BinTree<int> t2 = {comun, 4, { 8 }};
```



Ejemplo

```
BinTree<int> comun = {{ 7 }, 3, { 9 }};
BinTree<int> t1 = {{}, 5, comun};
BinTree<int> t2 = {comun, 4, { 8 }};
```



Ejemplo

```
BinTree<int> comun = {{ 7 }, 3, { 9 }};
BinTree<int> t1 = {{}, 5, comun};
BinTree<int> t2 = {comun, 4, { 8 }};
```