TAD CONJUNTO MEDIANTE AVL

UNIVERSIDAD
COMPLUTENSE
MADRID

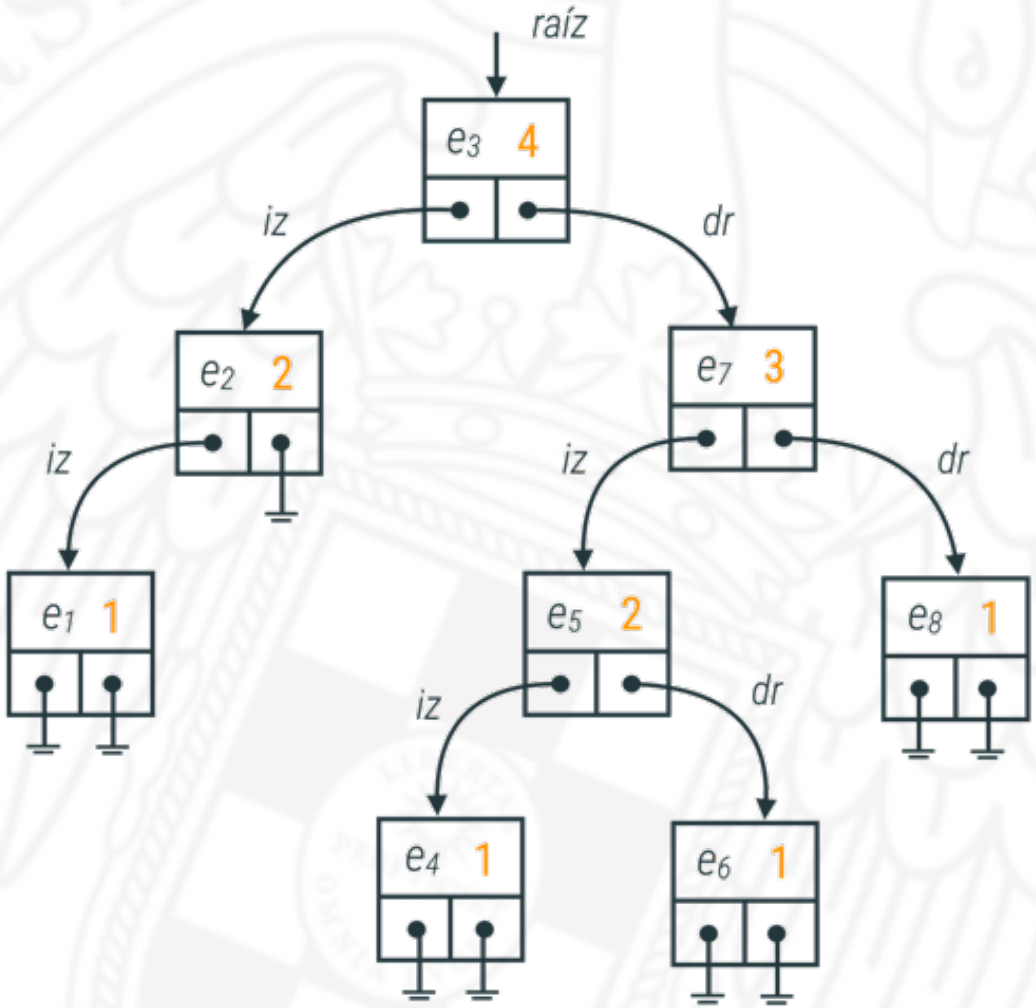
ALBERTO VERDEJO

TAD de conjuntos

El TAD de los conjuntos (de elementos ordenables) cuenta con las siguientes operaciones:

- conjunto vacío, `Set`
- insertar un elemento, `bool insert(T const& e)`
- eliminar un elemento, `bool erase(T const& e)`
- averiguar la pertenencia al conjunto, `bool contains(T const& e) const`
- averiguar si el conjunto es vacío, `bool empty() const`
- obtener el cardinal del conjunto, `int size() const`
- Iteradores que permitan recorrer el conjunto de forma ordenada.

Implementación mediante un árbol AVL



Implementación mediante un AVL

TreeSet_AVL.h

```
template <class T, class Comparator = std::less<T>>
class Set {
protected:

    struct TreeNode;
    using Link = TreeNode *;
    struct TreeNode {
        T elem;
        Link iz, dr;
        int altura;
        TreeNode(T const& e, Link i = nullptr, Link d = nullptr,
                 int alt = 1) : elem(e), iz(i), dr(d), altura(alt) {}
    };
```

Implementación mediante un AVL

TreeSet_AVL.h

```
    // puntero a la raíz de la estructura jerárquica de nodos
    Link raiz;

    // número de elementos (cardinal del conjunto)
    int nelems;

    // objeto función que compara elementos (orden total estricto)
    Comparator menor;


public:
    bool insert(T const& e) {  // O(log N)
        return inserta(e, raiz);
    }
```

Inserción de un elemento

TreeSet_AVL.h

```
protected:
   bool inserta(T const& e, Link & a) {  // O(log N)
      bool crece;
      if (a == nullptr) { // se inserta el nuevo elemento e
         a = new TreeNode(e);
         ++nelems;
         crece = true;
      } else if (menor(e, a->elem)) {
         crece = inserta(e, a->iz);
         if (crece) reequilibraDer(a);
      } else if (menor(a->elem, e)) {
         crece = inserta(e, a->dr);
         if (crece) reequilibraIzq(a);
      } else // el elemento e ya está en el árbol
         crece = false;
      return crece;
   }
```

Autoequilibrado

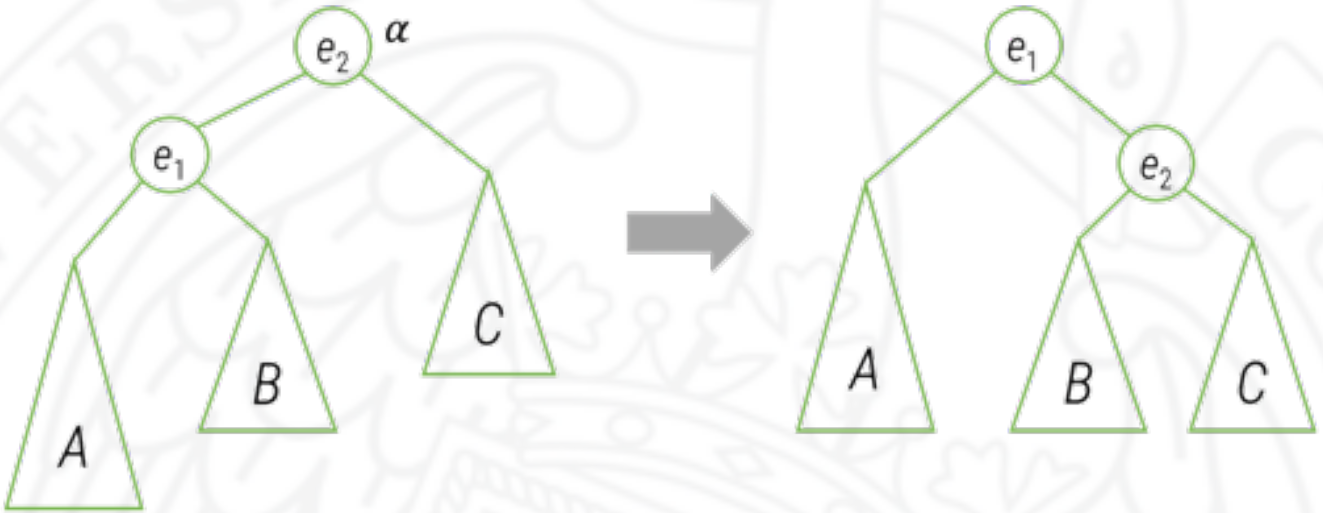
TreeSet_AVL.h

```
void reequilibraDer(Link & a) {  // O(1)
   if (altura(a->iz) - altura(a->dr) > 1) {
      if (altura(a->iz->dr) > altura(a->iz->iz))
         rotaIzqDer(a);
      else rotaDer(a);
   }
   else a->altura = std::max(altura(a->iz), altura(a->dr)) + 1;
}

int altura(Link a) {  // O(1)
   if (a == nullptr) return 0;
   else return a->altura;
}
```

Rotación simple a la derecha

TreeSet_AVL.h




```
void rotaDer(Link & r2) {  // O(1)
    Link r1 = r2->iz;
    r2->iz = r1->dr;
    r1->dr = r2;
    r2->altura = std::max(altura(r2->iz), altura(r2->dr)) + 1;
    r1->altura = std::max(altura(r1->iz), altura(r1->dr)) + 1;
    r2 = r1;
}
```

Rotación doble izquierda-derecha

TreeSet_AVL.h


```
void rotaIzqDer(Link & r3) {  // O(1)
    rotaIzq(r3->iz);
    rotaDer(r3);
}
```

Eliminación de un elemento

TreeSet_AVL.h

```
protected:
   bool borra(T const& e, Link & a) {  // O(log N)
      bool decrece = false;
      if (a != nullptr) {
         if (menor(e, a->elem)) {
            decrece = borra(e, a->iz);
            if (decrece) reequilibraIzq(a);
         }
         else if (menor(a->elem, e)) {
            decrece = borra(e, a->dr);
            if (decrece) reequilibraDer(a);
         }
         else { // e == a->elem
         ...
```