Implementación de conjuntos sobre ABB en `C++`

Algoritmos y Estructuras de Datos II

1er cuatrimestre de 2023

Introducción

- Vamos a implementar un conjunto en `C++`.
- Usaremos un árbol binario de búsqueda (ABB) como estructura de representación.
- Vamos a usar memoria dinámica. (¿Por qué?)

Árboles binarios de búsqueda (ABB)

Un árbol binario es un ABB si y sólo si

Árboles binarios de búsqueda (ABB)

Un árbol binario es un ABB si y sólo si es nil o

Árboles binarios de búsqueda (ABB)

Un árbol binario es un ABB si y sólo si es nil o satisface todas las siguientes condiciones:

Árboles binarios de búsqueda (ABB)

Un árbol binario es un ABB si y sólo si es nil o satisface todas las siguientes condiciones:

- Los valores en todos los nodos del subárbol izquierdo son menores que el valor en la raíz.

Árboles binarios de búsqueda (ABB)

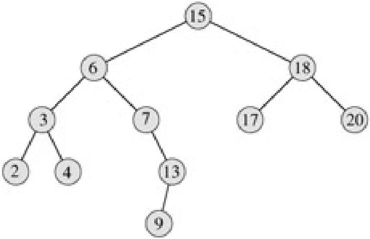
Un árbol binario es un ABB si y sólo si es nil o satisface todas las siguientes condiciones:

- Los valores en todos los nodos del subárbol izquierdo son menores que el valor en la raíz.
- Los valores en todos los nodos del subárbol derecho son mayores que el valor en la raíz.

Árboles binarios de búsqueda (ABB)

Un árbol binario es un ABB si y sólo si es nil o satisface todas las siguientes condiciones:

- Los valores en todos los nodos del subárbol izquierdo son menores que el valor en la raíz.
- Los valores en todos los nodos del subárbol derecho son mayores que el valor en la raíz.
- Los subárboles izquierdo y derecho son ABBs.



Implementación en `C++`

- Vamos a implementar una clase `Conjunto<T>` paramétrica en un tipo `T` con un orden total estricto $<$.
- Primero plantearemos el esquema de la clase.
- Luego la parte pública (interfaz).
- Luego la parte privada (representación e implementación de los métodos).

Interfaz

- Queremos dotar a nuestra clase de una interfaz de conjunto.
- ¿Qué operaciones serán visibles para el usuario?
  En particular, para el taller, nos conformamos con:

Interfaz

- Queremos dotar a nuestra clase de una interfaz de conjunto.
- ¿Qué operaciones serán visibles para el usuario?
  En particular, para el taller, nos conformamos con:
  - Crear un conjunto nuevo (vacío)

Interfaz

- Queremos dotar a nuestra clase de una interfaz de conjunto.
- ¿Qué operaciones serán visibles para el usuario?
  En particular, para el taller, nos conformamos con:
  - Crear un conjunto nuevo (vacío)
  - Insertar un elemento

Interfaz

- Queremos dotar a nuestra clase de una interfaz de conjunto.
- ¿Qué operaciones serán visibles para el usuario?
  En particular, para el taller, nos conformamos con:
  - Crear un conjunto nuevo (vacío)
  - Insertar un elemento
  - Decidir si un elemento pertenece al conjunto

Interfaz

- Queremos dotar a nuestra clase de una interfaz de conjunto.
- ¿Qué operaciones serán visibles para el usuario?
  En particular, para el taller, nos conformamos con:
  - Crear un conjunto nuevo (vacío)
  - Insertar un elemento
  - Decidir si un elemento pertenece al conjunto
  - Eliminar un elemento

Interfaz

- Queremos dotar a nuestra clase de una interfaz de conjunto.
- ¿Qué operaciones serán visibles para el usuario?
  En particular, para el taller, nos conformamos con:
  - Crear un conjunto nuevo (vacío)
  - Insertar un elemento
  - Decidir si un elemento pertenece al conjunto
  - Eliminar un elemento
  - Obtener la cantidad de elementos

Interfaz

- Queremos dotar a nuestra clase de una interfaz de conjunto.
- ¿Qué operaciones serán visibles para el usuario?
  En particular, para el taller, nos conformamos con:
  - Crear un conjunto nuevo (vacío)
  - Insertar un elemento
  - Decidir si un elemento pertenece al conjunto
  - Eliminar un elemento
  - Obtener la cantidad de elementos
  - Mostrar los elementos

Interfaz

- Queremos dotar a nuestra clase de una interfaz de conjunto.
- ¿Qué operaciones serán visibles para el usuario?
  En particular, para el taller, nos conformamos con:
  - Crear un conjunto nuevo (vacío)
  - Insertar un elemento
  - Decidir si un elemento pertenece al conjunto
  - Eliminar un elemento
  - Obtener la cantidad de elementos
  - Mostrar los elementos
- ¿Alguna otra operación que podría resultar útil?
  (dado que `T` tiene orden total estricto)

Interfaz

- Queremos dotar a nuestra clase de una interfaz de conjunto.
- ¿Qué operaciones serán visibles para el usuario?
  En particular, para el taller, nos conformamos con:
  - Crear un conjunto nuevo (vacío)
  - Insertar un elemento
  - Decidir si un elemento pertenece al conjunto
  - Eliminar un elemento
  - Obtener la cantidad de elementos
  - Mostrar los elementos
- ¿Alguna otra operación que podría resultar útil?
  (dado que `T` tiene orden total estricto)
  - Obtener el mínimo

Interfaz

- Queremos dotar a nuestra clase de una interfaz de conjunto.
- ¿Qué operaciones serán visibles para el usuario?
  En particular, para el taller, nos conformamos con:
  - Crear un conjunto nuevo (vacío)
  - Insertar un elemento
  - Decidir si un elemento pertenece al conjunto
  - Eliminar un elemento
  - Obtener la cantidad de elementos
  - Mostrar los elementos
- ¿Alguna otra operación que podría resultar útil?
  (dado que `T` tiene orden total estricto)
  - Obtener el mínimo
  - Obtener el máximo

Interfaz

- Queremos dotar a nuestra clase de una interfaz de conjunto.
- ¿Qué operaciones serán visibles para el usuario?
  En particular, para el taller, nos conformamos con:
  - Crear un conjunto nuevo (vacío)
  - Insertar un elemento
  - Decidir si un elemento pertenece al conjunto
  - Eliminar un elemento
  - Obtener la cantidad de elementos
  - Mostrar los elementos
- ¿Alguna otra operación que podría resultar útil?
  (dado que `T` tiene orden total estricto)
  - Obtener el mínimo
  - Obtener el máximo
  - Obtener el elemento siguiente a otro dado

Interfaz

```
template <class T>
class Conjunto {
    public:
        Conjunto();
        void insertar(const T&);
        bool pertenece(const T&) const;
        void remover(const T&);
        const T& minimo() const;
        const T& maximo() const;
        unsigned int cardinal() const;
        void mostrar(std::ostream&) const;
        const T& siguiente(const T&) const;
    private :
        /*...*/
};
```

Interfaz

```
template <class T>
class Conjunto {
    public:
        Conjunto();
        void insertar(const T&);
        bool pertenece(const T&) const;
        void remover(const T&);
        const T& minimo() const;
        const T& maximo() const;
        unsigned int cardinal() const;
        void mostrar(std::ostream&) const;
        const T& siguiente(const T&) const;
    private :
        /*...*/
};
```

- ¿Por qué mínimo y máximo devuelven const T?

Representación de los nodos

- Definimos una estructura `Nodo` para representar los nodos del ABB.
- La estructura estará en la parte privada de la clase ABB (no queremos exportarla).

Representación de los nodos

- Definimos una estructura `Nodo` para representar los nodos del ABB.
- La estructura estará en la parte privada de la clase ABB (no queremos exportarla).
- La estructura va a contener un valor del tipo `T` y dos punteros: uno al hijo izquierdo y el otro al hijo derecho.
- La estructura tendrá un constructor que recibirá el valor de tipo `T` como único argumento e inicializará los dos punteros a `nullptr`.

Representación de los nodos

```
private:
    struct Nodo {
        T valor;
        Nodo* izq;
        Nodo* der;
        Nodo(const T& v) :
          valor(v), izq(nullptr), der(nullptr) {
        }
    };
    /*...*/
```

Representación de los nodos

```
private:
    struct Nodo {
        T valor;
        Nodo* izq;
        Nodo* der;
        Nodo(const T& v) :
          valor(v), izq(nullptr), der(nullptr) {
        }
    };
    Nodo* raiz;
```

Observar que `raiz` es la única variable de instancia y apunta al nodo raíz del ABB, o es `nullptr` si el conjunto es vacío.

Representación de los nodos

¿En qué se diferencia con la estructura de la lista doblemente enlazada?

```
private:
    struct Nodo {
        T valor;
        Nodo* prev;
        Nodo* sig;
        Nodo(const T& v) :
          valor(v), prev(nullptr), sig(nullptr) {
        }
    };
    Nodo* primero;
```

¿Representan lo mismo? ¿Se comportan igual?

Pertenencia de un elemento

Pertenencia de un elemento

- Empezamos en la raíz

Pertenencia de un elemento

- Empezamos en la raíz, si existe — si no, devolver `False`.

Pertenencia de un elemento

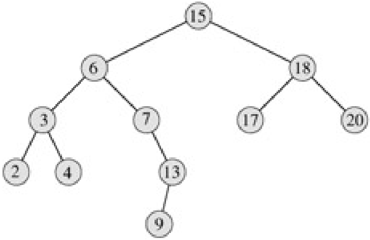
- Empezamos en la raíz, si existe — si no, devolver `False`.
- Si el elemento está en la raíz, devolvemos `True`.

Pertenencia de un elemento

- Empezamos en la raíz, si existe — si no, devolver `False`.
- Si el elemento está en la raíz, devolvemos `True`.
- Si no, decidimos en qué nodo continuar en base a $<$
  - Consideramos a este nodo como la raíz del subárbol correspondiente y repetimos.

Pertenencia de un elemento

- Empezamos en la raíz, si existe — si no, devolver `False`.
- Si el elemento está en la raíz, devolvemos `True`.
- Si no, decidimos en qué nodo continuar en base a $<$
  - Consideramos a este nodo como la raíz del subárbol correspondiente y repetimos.



Insertar un elemento

Insertar un elemento

- Buscamos en qué lugar del árbol debe ir el nuevo elemento.

Insertar un elemento

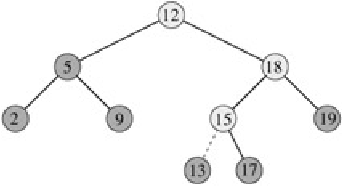
- Buscamos en qué lugar del árbol debe ir el nuevo elemento.
- Para ello hacemos una búsqueda del elemento en el árbol.

Insertar un elemento

- Buscamos en qué lugar del árbol debe ir el nuevo elemento.
- Para ello hacemos una búsqueda del elemento en el árbol.
- Si la búsqueda es exitosa, el elemento ya pertenece al conjunto y no hacemos nada.

Insertar un elemento

- Buscamos en qué lugar del árbol debe ir el nuevo elemento.
- Para ello hacemos una búsqueda del elemento en el árbol.
- Si la búsqueda es exitosa, el elemento ya pertenece al conjunto y no hacemos nada.
- Si la búsqueda fracasa, se debe insertar un nuevo nodo como hijo del último nodo de la búsqueda.



Borrar un elemento

Borrar un elemento

- Buscamos el nodo que tenemos que borrar.
- Tenemos 3 casos:

Borrar un elemento

- Buscamos el nodo que tenemos que borrar.
- Tenemos 3 casos:
  - El nodo que tenemos que borrar es una hoja.
    $\rightarrow$ Lo borramos.

Borrar un elemento

- Buscamos el nodo que tenemos que borrar.
- Tenemos 3 casos:
  - El nodo que tenemos que borrar es una hoja.
    $\rightarrow$ Lo borramos.
  - El nodo que tenemos que borrar tiene un solo hijo.
    $\rightarrow$ El hijo pasa a ocupar el lugar del padre.

Borrar un elemento

- Buscamos el nodo que tenemos que borrar.
- Tenemos 3 casos:
  - El nodo que tenemos que borrar es una hoja.
    $\rightarrow$ Lo borramos.
  - El nodo que tenemos que borrar tiene un solo hijo.
    $\rightarrow$ El hijo pasa a ocupar el lugar del padre.
  - El nodo que tenemos que borrar tiene dos hijos.

Borrar un elemento

- Buscamos el nodo que tenemos que borrar.
- Tenemos 3 casos:
  - El nodo que tenemos que borrar es una hoja.
    $\rightarrow$ Lo borramos.
  - El nodo que tenemos que borrar tiene un solo hijo.
    $\rightarrow$ El hijo pasa a ocupar el lugar del padre.
  - El nodo que tenemos que borrar tiene dos hijos.
    - ¿Qué nodos pueden ocupar su lugar?

Borrar un elemento

- Buscamos el nodo que tenemos que borrar.
- Tenemos 3 casos:
  - El nodo que tenemos que borrar es una hoja.
    $\rightarrow$ Lo borramos.
  - El nodo que tenemos que borrar tiene un solo hijo.
    $\rightarrow$ El hijo pasa a ocupar el lugar del padre.
  - El nodo que tenemos que borrar tiene dos hijos.
    - ¿Qué nodos pueden ocupar su lugar?
    - El inmediato sucesor. ¿Dónde está?

Borrar un elemento

- Buscamos el nodo que tenemos que borrar.
- Tenemos 3 casos:
  - El nodo que tenemos que borrar es una hoja.
    $\rightarrow$ Lo borramos.
  - El nodo que tenemos que borrar tiene un solo hijo.
    $\rightarrow$ El hijo pasa a ocupar el lugar del padre.
  - El nodo que tenemos que borrar tiene dos hijos.
    - ¿Qué nodos pueden ocupar su lugar?
    - El inmediato sucesor. ¿Dónde está?
    - El inmediato predecesor. ¿Dónde está?

Discusión

- ¿Qué complejidades en peor caso tienen las siguientes operaciones?
  - Pertenece

Discusión

- ¿Qué complejidades en peor caso tienen las siguientes operaciones?
    - Pertenece

Discusión

- ¿Qué complejidades en peor caso tienen las siguientes operaciones?
  - Pertenece $\rightarrow \mathcal{O}(N)$
  - Insertar

Discusión

- ¿Qué complejidades en peor caso tienen las siguientes operaciones?
  - Pertenece $\rightarrow \mathcal{O}(N)$
  - Insertar

Discusión

- ¿Qué complejidades en peor caso tienen las siguientes operaciones?
  - Pertenece $\rightarrow \mathcal{O}(N)$
  - Insertar $\rightarrow \mathcal{O}(N)$
  - Borrar

Discusión

- ¿Qué complejidades en peor caso tienen las siguientes operaciones?
  - Pertenece $\rightarrow \mathcal{O}(N)$
  - Insertar $\rightarrow \mathcal{O}(N)$
  - Borrar

Discusión

- ¿Qué complejidades en peor caso tienen las siguientes operaciones?
  - Pertenece $\rightarrow \mathcal{O}(N)$
  - Insertar $\rightarrow \mathcal{O}(N)$
  - Borrar $\rightarrow \mathcal{O}(N)$
  - Mínimo/Máximo

Discusión

- ¿Qué complejidades en peor caso tienen las siguientes operaciones?
  - Pertenece $\rightarrow \mathcal{O}(N)$
  - Insertar $\rightarrow \mathcal{O}(N)$
  - Borrar $\rightarrow \mathcal{O}(N)$
  - Mínimo/Máximo

Discusión

- ¿Qué complejidades en peor caso tienen las siguientes operaciones?
  - Pertenece $\rightarrow \mathcal{O}(N)$
  - Insertar $\rightarrow \mathcal{O}(N)$
  - Borrar $\rightarrow \mathcal{O}(N)$
  - Mínimo/Máximo $\rightarrow \mathcal{O}(N)$

  donde N es la cantidad de elementos que tiene el conjunto.

Discusión

- ¿Qué complejidades en peor caso tienen las siguientes operaciones?
  - Pertenece $\rightarrow \mathcal{O}(N)$
  - Insertar $\rightarrow \mathcal{O}(N)$
  - Borrar $\rightarrow \mathcal{O}(N)$
  - Mínimo/Máximo $\rightarrow \mathcal{O}(N)$

  donde N es la cantidad de elementos que tiene el conjunto.

Las complejidades dependen del orden en el que se hayan ingresado los datos.

Iteración

Problema

Dar un algoritmo para recorrer todos los nodos de un árbol,

- en tiempo lineal (i.e. en $\mathcal{O}(n)$),
- iterativo
  - ¿Por qué, si ya conocemos recorridos recursivos?
    Para (después) poder implementar iteradores sobre árboles.

Recorridos de árboles

Repaso

`preorder(Bin(`i`, `r`, `d`))` $\equiv$

Recorridos de árboles

Repaso

$\texttt{preorder(Bin(}i\texttt{, }r\texttt{, }d\texttt{))} \equiv \langle r \rangle \texttt{ \& preorder(}i\texttt{) \& preorder(}d\texttt{)}$

$\texttt{inorder(Bin(}i\texttt{, }r\texttt{, }d\texttt{))} \equiv$

Recorridos de árboles

Repaso

$\texttt{preorder(Bin(}i\texttt{, }r\texttt{, }d\texttt{))} \equiv \langle r \rangle \texttt{ \& preorder(}i\texttt{) \& preorder(}d\texttt{)}$

$\texttt{inorder(Bin(}i\texttt{, }r\texttt{, }d\texttt{))} \equiv \texttt{inorder(}i\texttt{) \& } \langle r \rangle \texttt{ \& inorder(}d\texttt{)}$

$\texttt{postorder(Bin(}i\texttt{, }r\texttt{, }d\texttt{))} \equiv$

Recorridos de árboles

Repaso

`preorder(Bin(`i`, `r`, `d`))` $\equiv$ $\langle r \rangle$ `& preorder(`i`) & preorder(`d`)`

`inorder(Bin(`i`, `r`, `d`))` $\equiv$ `inorder(`i`) &` $\langle r \rangle$ `& inorder(`d`)`

`postorder(Bin(`i`, `r`, `d`))` $\equiv$ `postorder(`i`) & postorder(`d`) &` $\langle r \rangle$

Observación

Si el árbol es un ABB, el recorrido inorder está

Recorridos de árboles

Repaso

$\texttt{preorder(Bin(}i\texttt{, }r\texttt{, }d\texttt{))} \equiv \langle r \rangle \ \texttt{\&}\ \texttt{preorder(}i\texttt{)}\ \texttt{\&}\ \texttt{preorder(}d\texttt{)}$

$\texttt{inorder(Bin(}i\texttt{, }r\texttt{, }d\texttt{))} \equiv \texttt{inorder(}i\texttt{)}\ \texttt{\&}\ \langle r \rangle\ \texttt{\&}\ \texttt{inorder(}d\texttt{)}$

$\texttt{postorder(Bin(}i\texttt{, }r\texttt{, }d\texttt{))} \equiv \texttt{postorder(}i\texttt{)}\ \texttt{\&}\ \texttt{postorder(}d\texttt{)}\ \&\ \langle r \rangle$

Observación

Si el árbol es un ABB, el recorrido inorder está ordenado.

Recorridos de árboles

Repaso

$\texttt{preorder(Bin(}i\texttt{, }r\texttt{, }d\texttt{))} \equiv \langle r\rangle \ \texttt{\&}\ \texttt{preorder(}i\texttt{)}\ \texttt{\&}\ \texttt{preorder(}d\texttt{)}$

$\texttt{inorder(Bin(}i\texttt{, }r\texttt{, }d\texttt{))} \equiv \texttt{inorder(}i\texttt{)}\ \texttt{\&}\ \langle r\rangle\ \texttt{\&}\ \texttt{inorder(}d\texttt{)}$

$\texttt{postorder(Bin(}i\texttt{, }r\texttt{, }d\texttt{))} \equiv \texttt{postorder(}i\texttt{)}\ \texttt{\&}\ \texttt{postorder(}d\texttt{)}\ \&\ \langle r\rangle$

Observación

Si el árbol es un ABB, el recorrido inorder está ordenado.

Para hacer un recorrido inorder:

- El primer elemento que tenemos que visitar es

Recorridos de árboles

Repaso

`preorder(Bin(`i`, `r`, `d`))` $\equiv$ $\langle r \rangle$ `& preorder(`i`) & preorder(`d`)`

`inorder(Bin(`i`, `r`, `d`))` $\equiv$ `inorder(`i`) &` $\langle r \rangle$ `& inorder(`d`)`

`postorder(Bin(`i`, `r`, `d`))` $\equiv$ `postorder(`i`) & postorder(`d`)` & $\langle r \rangle$

Observación

Si el árbol es un ABB, el recorrido inorder está ordenado.

Para hacer un recorrido inorder:

- El primer elemento que tenemos que visitar es el mínimo. Sabemos cómo encontrarlo: yendo siempre hacia la izquierda.

Recorridos de árboles

Repaso

`preorder(Bin(`i`, `r`, `d`))` $\equiv$ $\langle r \rangle$ `& preorder(`i`) & preorder(`d`)`

`inorder(Bin(`i`, `r`, `d`))` $\equiv$ `inorder(`i`) &` $\langle r \rangle$ `& inorder(`d`)`

`postorder(Bin(`i`, `r`, `d`))` $\equiv$ `postorder(`i`) & postorder(`d`) &` $\langle r \rangle$

Observación

Si el árbol es un ABB, el recorrido inorder está ordenado.

Para hacer un recorrido inorder:

- El primer elemento que tenemos que visitar es el mínimo.
  Sabemos cómo encontrarlo: yendo siempre hacia la izquierda.
- Tenemos que hallar el sucesor del mínimo.

Recorridos de árboles

Repaso

`preorder(Bin(`i`, `r`, `d`))` $\equiv$ $\langle r \rangle$ `& preorder(`i`) & preorder(`d`)`

`inorder(Bin(`i`, `r`, `d`))` $\equiv$ `inorder(`i`) &` $\langle r \rangle$ `& inorder(`d`)`

`postorder(Bin(`i`, `r`, `d`))` $\equiv$ `postorder(`i`) & postorder(`d`)` & $\langle r \rangle$

Observación

Si el árbol es un ABB, el recorrido inorder está ordenado.

Para hacer un recorrido inorder:

- El primer elemento que tenemos que visitar es el mínimo. Sabemos cómo encontrarlo: yendo siempre hacia la izquierda.
- Tenemos que hallar el sucesor del mínimo.
- Más en general, tenemos que poder hallar el sucesor de un elemento arbitrario del árbol.

Sucesor de un elemento

Supongamos que queremos encontrar el sucesor de un nodo X en el árbol:

1. Recorremos el ABB hasta encontrar el nodo X.
   1.1 Si el nodo X tiene hijo derecho, el sucesor es el mínimo del hijo derecho.
   1.2 Sino, recorremos el ABB comenzando por la raíz hasta llegar al nodo X considerando lo siguiente:
      1.2.1 Si el nodo actual es $>$ que el nodo X, entonces tomamos al nodo actual como sucesor y continuamos por el subárbol izquierdo.
      1.2.2 De lo contrario continuamos por el subárbol derecho.

Sucesor de un elemento, según el Cormen

```
TREE-SUCCESSOR(x)
if right[x] ≠ NIL
    then return TREE-MINIMUM(right[x])
y ← p[x]
while y ≠ NIL and x = right[y]
      do x ← y
         y ← p[y]
return y
```

Pueden encontrar los algoritmos para árboles binarios de búsqueda (BST en inglés) en el capítulo 12 del Cormen.

¡A programar!

En `Conjunto.hpp` está la declaración de la clase, su parte pública y la definición de `Nodo`.