

ÁRBOLES BINARIOS DE BÚSQUEDA (ABB)

INTRODUCCIÓN

- Un árbol binario de búsqueda (ABB) es un árbol binario con la propiedad de que todos los elementos almacenados en el subárbol izquierdo de cualquier nodo x son menores que el elemento almacenado en x , y todos los elementos almacenados en el subárbol derecho de x son mayores que el elemento almacenado en x .

- Una interesante propiedad es que si se listan los nodos del ABB en inorden nos da la lista de nodos ordenada. Esta propiedad define un método de ordenación similar al Quicksort, con el nodo raíz jugando un papel similar al del elemento de partición del Quicksort aunque con los ABB hay un gasto extra de memoria mayor debido a los punteros. La propiedad de ABB hace que sea muy simple diseñar un procedimiento para realizar la búsqueda.

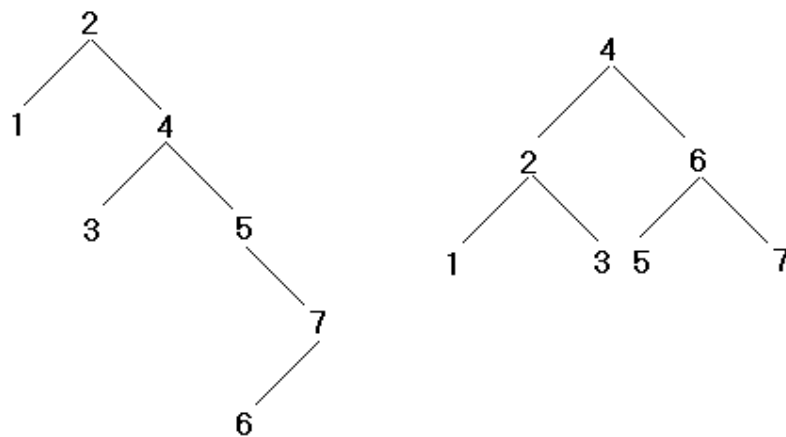


Figura 1: Dos ABB con los mismos elementos.

- El espacio requerido para el almacenamiento es $O(n)$. Donde n es el número de nodos del árbol.

LA CLASE ABSTRACTA

```
Class ABB
{
    Public:
        typedef struct nodo *Iterador;

        ABB();
        ABB (const ABB<Tbase>& v);
        ABB<Tbase>& operator = (const ABB<Tbase> &v);
        Iterador primero() const;
        Iterador siguiente(Iterador i) const;
        Iterador final() const;
        const Tbase& etiqueta(const Iterador n) const;
        Iterador buscar(const Tbase& e) const;
        bool insertar(const Tbase& e);
        Iterador borrar(const Iterador p);
        void equilibrar();
        void clear();
        int size() const;
        bool empty() const;
        bool operator == (const ABB<Tbase>& v) const;
        bool operator != (const ABB<Tbase>& v) const;
        ~ABB();

    Private:
```

```

struct nodo
{
    Tbase etiqueta;
    struct nodo *izqda;
    struct nodo *drcha;
    struct nodo *padre;
};

struct nodo *laraiz;
int nelementos;

void destruir(nodo * n);
void copiar(nodo *& dest, nodo * orig);
void enganchar(nodo *&r, nodo **m, int n);
}

```

Implementación

```

template <class Tbase>
inline ABB<Tbase>::ABB()
{
    laraiz=0;
    nelementos=0;
}

```

```

template <class Tbase>
ABB<Tbase>::ABB (const ABB<Tbase>& v)
{
    copiar (laraiz,v.laraiz);
}

```

```
if (laraiz!=0)
    laraiz->padre= 0;
nelementos=v.nelementos;
}
```

```
template <class Tbase>
void ABB<Tbase>::destruir(nodo * n)
{
    if (n!=0)
    {
        destruir(n->izqda);
        destruir(n->drcha);
        delete n;
    }
}
```

```
template <class Tbase>
void ABB<Tbase>::copiar(nodo *& dest, nodo * orig)
{
    if (orig==0)
        dest= 0;
    else {
        dest= new nodo;
        dest->etiqueta= orig->etiqueta;
        copiar (dest->izqda,orig->izqda);
        copiar (dest->drcha,orig->drcha);
        if (dest->izqda!=0)
            dest->izqda->padre= dest;
        if (dest->drcha!=0)
            dest->drcha->padre= dest;
    }
}
```

```
    }  
}
```

```
template <class Tbase>  
void ABB<Tbase>::enganchar (nodo* & r, nodo **m, int n)  
{  
    int i;  
    i=n/2;  
    r=m[i];  
    if (i>0)  
    {  
        enganchar(r->izqda,m,i);  
        r->izqda->padre=r;  
    }  
    else r->izqda= 0;  
    if (n-i-1>0)  
    {  
        enganchar(r->drcha,m+i+1,n-i-1);  
        r->drcha->padre=r;  
    }  
    else r->drcha=0;  
}
```

```
template <class Tbase>  
ABB<Tbase>& ABB<Tbase>::operator=(const ABB<Tbase>& v)  
{  
    if (this!=&v)  
    {  
        destruir(laraiz);  
        copiar (laraiz,v.laraiz);  
    }  
}
```

```

        if (laraiz!=0)
            laraiz->padre= 0;
        nelementos=v.nelementos;
    }
    return *this;
}

```

```

template <class Tbase>
ABB<Tbase>::Iterador  ABB<Tbase>::primero() const
{
    nodo *p;
    if (laraiz==0)
        return 0;
    else {
        p=laraiz;
        while (p->izqda!=0)
            p= p->izqda;
        return p;
    }
}

```

```

template <class Tbase>
ABB<Tbase>::Iterador  ABB<Tbase>::siguiente(Iterador  i)
const
{
    nodo *padre;
    bool subir;
    assert(i!=0);
    if (i->drcha!=0)
    {

```

```

        i=i->drcha;
        while (i->izqda!=0)
            i=i->izqda;
    }
    else {
        subir=true;
        while (subir)
        {
            padre= i->padre;
            if (padre==0)
            {
                i=0;
                subir=false;
            }
            else if (padre->drcha!=i)
            {
                i= padre;
                subir=false;
            }
            else i= padre;
        }
    }
    return i;
}

```

```

template <class Tbase>
inline ABB<Tbase>::Iterador  ABB<Tbase>::final() const
{
    return 0;
}

```

```
template <class Tbase>
inline const Tbase& ABB<Tbase>::etiqueta(const Iterador p)
const
{
    assert(p!=0);
    return (p->etiqueta);
}
```

```
template <class Tbase>
ABB<Tbase>::Iterador ABB<Tbase>::buscar(const Tbase& e)
const
{
    Iterador i;
    i=laraiz;
    while (i!=0)
    {
        if (i->etiqueta<e)
            i=i->drcha;
        else if (e<i->etiqueta)
            i=i->izqda;
        else return i;
    }
    return final();
}
```

```
template <class Tbase>
inline bool ABB<Tbase>::insertar(const Tbase& e)
{
    bool fin,dev;
    nodo *p;
```



```

if (laraiz==0)
{
laraiz = new nodo;
laraiz->padre= laraiz->izqda= laraiz->drcha= 0;
laraiz->etiqueta= e;
nelementos++;
return true;
}
else {
p= laraiz;
fin=false;
while (!fin)
{
if (e<p->etiqueta)
{
if (p->izqda==0)
{
p->izqda= new nodo;
p->izqda->padre=p;
p=p->izqda;
p->drcha= p->izqda= 0;
fin=true;
dev= true;
}
else p= p->izqda;
}
else if (p->etiqueta<e)
{
if (p->drcha==0)
{
p->drcha= new nodo;
p->drcha->padre=p;

```

```

        p=p->drcha;
        p->drcha= p->izqda= 0;
        fin=true;
        dev= true;
    }
    else p= p->drcha;
}
else {
    fin=true;
    dev=false;
}
}
p->etiqueta= e;
if (dev)
    nelementos++;
return dev;
}

```

```

template <class Tbase>
ABB<Tbase>::Iterador ABB<Tbase>::borrar(const Iterador p)
{
    nodo *q, *aux, *dev;
    assert(p!=final());
    dev= siguiente(p);
    nelementos--;
    if (p->izqda==0 && p->drcha==0)
    {
        if (p==laraiz)
        {
            delete p;
            laraiz=0;
        }
    }
}

```

```

    }
    else {
        q=p->padre;
        delete p;
        if (q->izqda==p)
            q->izqda=0;
        else q->drcha=0;
    }
}
else if (p->izqda==0)
{
    if (p==laraiz)
    {
        laraiz= p->drcha;
        delete p;
        laraiz->padre= 0;
    }
    else {
        q=p->padre;
        if (q->izqda==p)
        {
            q->izqda=p->drcha;
            p->drcha->padre=q;
            delete p;
        }
        else {
            q->drcha=p->drcha;
            p->drcha->padre=q;
        }
    }
    delete p;
}
}
}

```

```

else if (p->drcha==0)
{
    if (p==laraiz)
    {
        laraiz= p->izqda;
        delete p;
        laraiz->padre= 0;
    }
    else {
        q=p->padre;
        if (q->drcha==p)
        {
            q->drcha=p->izqda;
            p->izqda->padre=q;
            delete p;
        }
        else {
            q->izqda=p->izqda;
            p->izqda->padre=q;
            delete p;
        }
    }
}
else {
    q=p->drcha;
    while (q->izqda!=0)
        q=q->izqda;
    aux= q->padre;
    if (q->drcha==0)
    {
        if (aux->izqda==q)
            aux->izqda= 0;
    }
}

```

```

        else aux->drcha=0;
    }
    else {
        if (aux->izqda==q)
            aux->izqda=q->drcha;
        else aux->drcha= q->drcha;
        q->drcha->padre=aux;
    }
    q->izqda= p->izqda;
    q->drcha= p->drcha;
    q->padre= p->padre;
    if (q->padre!=0)
    {
        aux= q->padre;
        if (aux->izqda==p)
            aux->izqda= q;
        else aux->drcha= q;
    }
    else laraiz=q;
    if (q->izqda!=0)
        q->izqda->padre=q;
        if (q->drcha!=0)
            q->drcha->padre=q;
    delete p;
}

return dev;
}

```

```

template <class Tbase>
void ABB<Tbase>::equilibrar()
{

```

```

int i;
nodo **m;
nodo *p;
if (nelementos>1)
{
    m= new nodo*[nelementos];
    for (p=primero(),i=0;p!=final();p=siguiente(p),++i)
        m[i]=p;
    enganchar(laraiz,m,nelementos);
    laraiz->padre= 0;
    delete [] m;
}
}

```

```

template <class Tbase>
inline void ABB<Tbase>::clear()
{
    destruir(laraiz);
    laraiz= 0;
    nelementos= 0;
}

```

```

template <class Tbase>
inline int ABB<Tbase>::size() const
{
    return nelementos;
}

```

```

template <class Tbase>

```

```
inline bool ABB<Tbase>::empty() const
{
    return laraiz==0;
}
```

```
template <class Tbase>
inline bool ABB<Tbase>::operator==(const ABB<Tbase>& v) const
{
    Iterador p,q;
    if (nelementos!=v.nelementos)
        return false;
    for (p=this->primero(),q=v.primero();
         p!=final();
         p=this->siguiente(p),q=v.siguiete(q))
        if (this->etiqueta(p)!=v.etiqueta(q))
            return false;
    return true;
}
```

```
template <class Tbase>
inline bool ABB<Tbase>::operator!=(const ABB<Tbase>& v) const
{
    return !(*this==v);
}
```

```
template <class Tbase>
inline ABB<Tbase>::~~ABB()
{
    destruir(laraiz);
}
```



- Una búsqueda o una inserción en un ABB requiere $O(\log_2 n)$ operaciones en el caso medio, en un árbol construido a partir de n claves aleatorias, y en el peor caso una búsqueda en un ABB con n claves puede implicar revisar las n claves, o sea, es $O(n)$.

- Si el elemento a borrar está en una hoja bastaría eliminarla, pero si el elemento está en un nodo interior, eliminándolo podríamos desconectar el árbol, para evitar que esto suceda se sigue el siguiente procedimiento:

- * Si el nodo a borrar u tiene sólo un hijo se sustituye u por ese hijo y el ABB quedará construido.
- * Si tiene dos hijos, se encuentra el menor elemento de los descendientes del hijo derecho (o el mayor de los descendientes del hijo izquierdo) y se coloca en lugar de u , de forma que se continúe manteniendo la propiedad de ABB.