



CAZZ

www.wuolah.com/student/CAZZ

26885

ABB.pdf

ED Práctica 5 - Arboles



2º Estructuras de Datos



Grado en Ingeniería Informática



**Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación
Universidad de Granada**



**Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.**



Exámenes, preguntas, apuntes.

12:48

WUOLAH

Join the student revolution.

MULTI

Conéctate dónde y cómo prefieras.

Guarda tus apuntes en un lugar seguro y ordenado, y accede a ellos desde tu pc, móvil o tablet.

Acceder

Registrarse

GET IT ON
Google Play

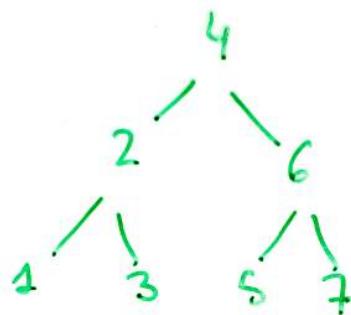
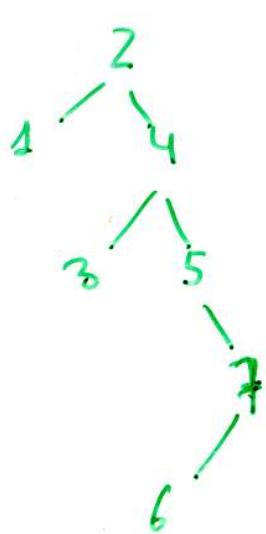
Download on the
App Store

ARBOLÉS BINARIOS DE BÚSQUEDA (ABB)

Definición

Un ABB es un árbol binario con la propiedad de que todos los elementos almacenados en el sub-árbol izquierdo de cualquier nodo x (incluyendo la raíz) son menores (o iguales^(*)) 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

Ejemplos



(*) Habitualmente se tienen claves no repetidas

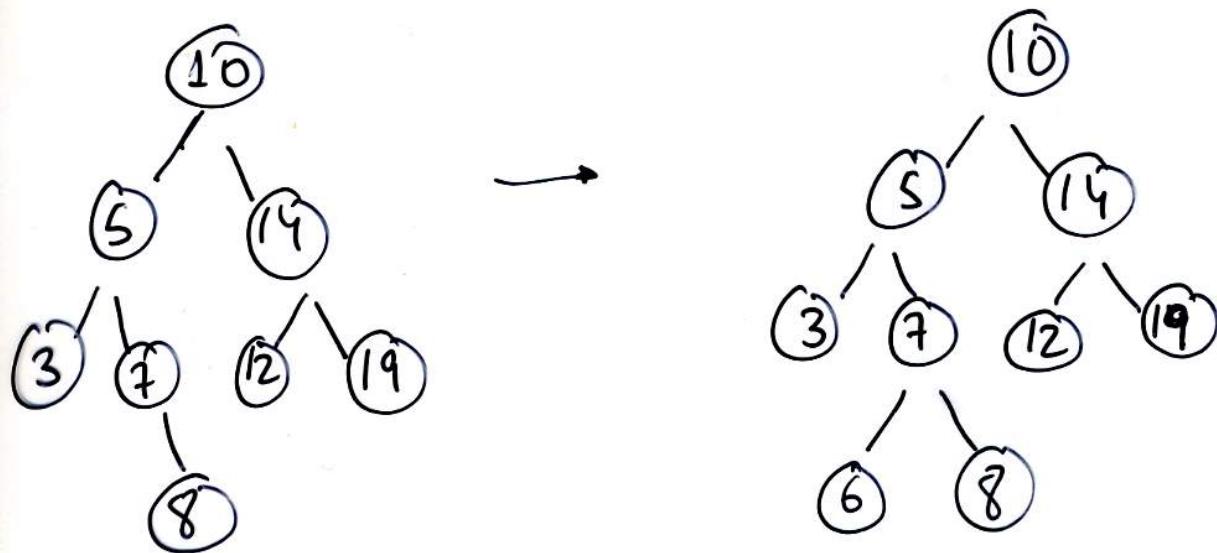
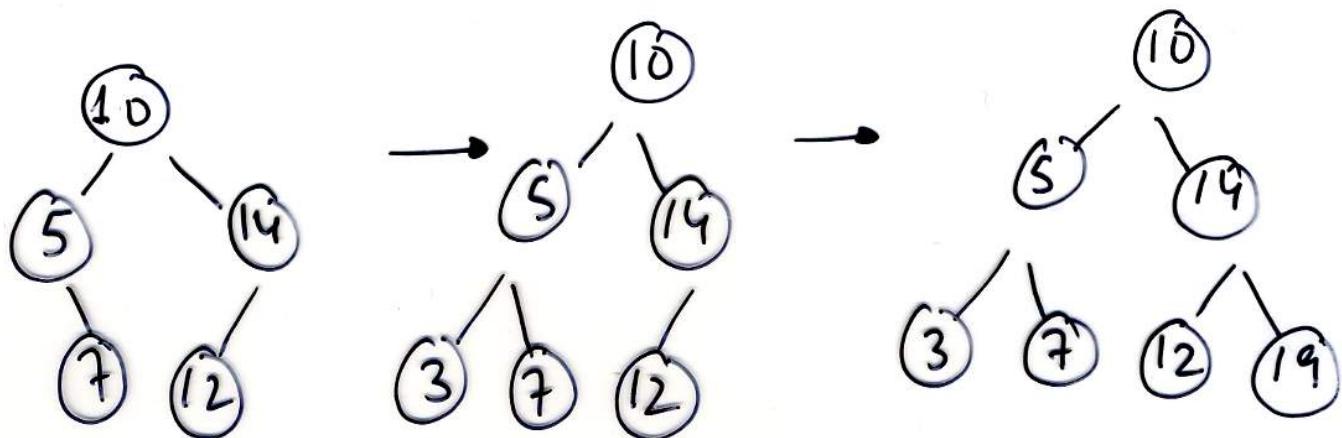
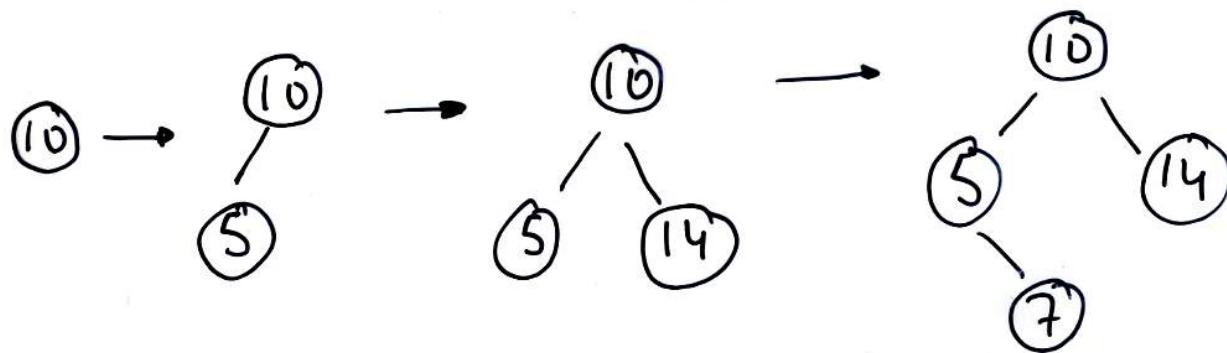
Nos interesan las operaciones de:

- pertenencia
- inserción
- borrado

Ejemplo

Construcción de un ÁRBOL con las claves:

{10, 5, 14, 7, 12, 3, 19, 8}



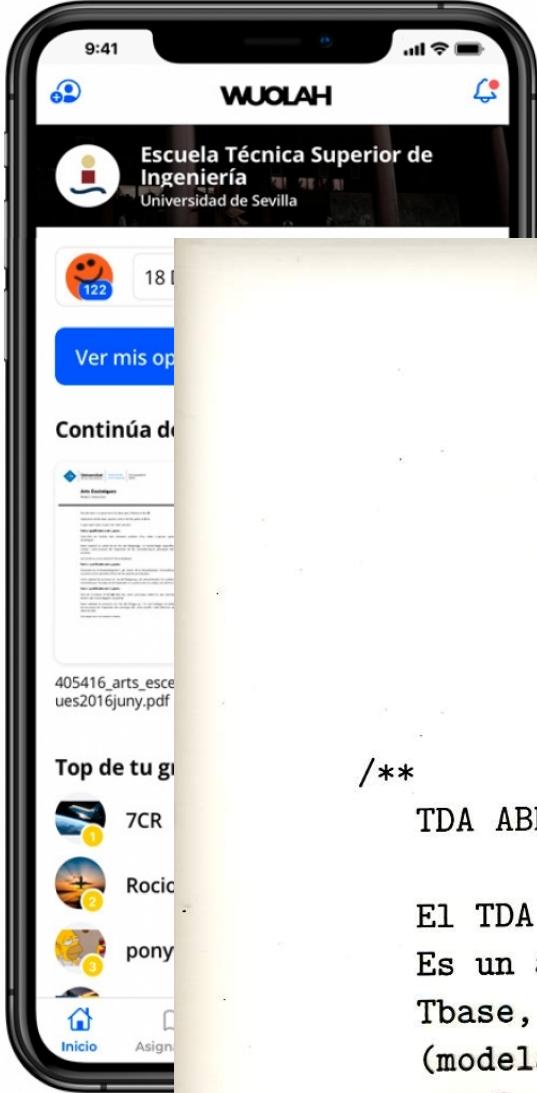
ABB

Motivación: La búsqueda binaria es un proceso rápido de búsqueda de elementos en un vector ordenado ($O(\log_2(n))$). Sin embargo, las inserciones y borrados son muy ineficientes ($O(n)$).

Árbol Binario de Búsqueda (ABB): árbol binario verificando que todos los elementos almacenados en el subárbol izquierdo de cualquier nodo n son menores que el elemento almacenado en n , y todos los elementos almacenados en el subárbol derecho de n son mayores (o iguales) que el elemento almacenado en n .

Propiedades:

- La búsqueda de un elemento en el árbol reproduce la búsqueda binaria: $O(\log_2(n))$.
- El recorrido en InOrden de un ABB produce un listado ordenado de las etiquetas.



Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

TDA ABB

/**

TDA ABB::ABB, Insertar, Existe, Borrar, begin, end, ~ABB.

El TDA ABB modela un Arbol Binario de Búsqueda. Es un árbol binario etiquetado con datos del tipo Tbase, entre los que existe un orden lineal (modelado mediante operator<). Para todo nodo se cumple que las etiquetas de los nodos a su izqda son menores estrictos que la suya, y que las etiquetas de los nodos a su drcha son mayores o iguales que la suya.

Requisitos para el tipo instanciador Tbase:
Tbase debe tener definidas las siguientes operaciones:

- Tbase & operator=(const Tbase & e);
- bool operator!=(const Tbase & e);
- bool operator==(const Tbase & e);
- bool operator<(const Tbase & e);

Son objetos mutables.

Residen en memoria dinámica.

*/

TDA ABB

```
template <class Tbase>
class ABB {
public:

    ABB();
    /**
     * Constructor por defecto.

     @doc
     Crea un Arbol Binario de Búsqueda vacío.
    */

    ABB(const ABB<Tbase> & a);
    /**
     * Constructor de copia.

     @param a: Arbol que se copia.

     @doc
     Crea un Arbol Binario de Búsqueda duplicado de a.
    */

}
```

TDA ABB

```
ABB(const Tbase & e);
```

```
/**
```

Constructor primitivo.

@param e: Elemento a insertar.

@doc

Crea un Arbol Binario de Búsqueda con un sólo nodo, que se etiqueta con el valor "e".

```
*/
```

```
bool Existe(const Tbase & e) const;
```

```
/**
```

Informa de la existencia de un elemento en el ABB.

@param e: elemento que se busca.

@return true, si el elemento e está en el árbol.
false, en otro caso.

```
*/
```

TDA ABB

```
void Insertar(const Tbase & e);
```

```
/**
```

Inserta un elemento en el árbol.

param e: Elemento que se inserta.

doc

Añade al ABB un nuevo nodo etiquetado con e.

```
*/
```

```
void Borrar(const Tbase & e);
```

```
/**
```

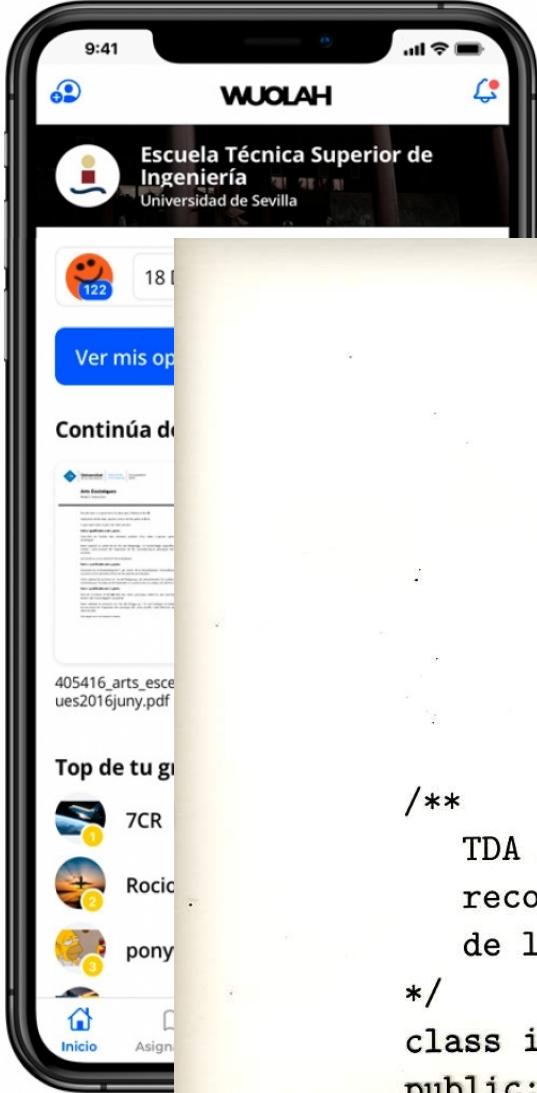
Elimina un elemento.

param e: Elemento a eliminar.

doc

Si existen uno o más nodos en el receptor con la etiqueta e, elimina uno de ellos.

```
*/
```



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

TDA ABB

```
/**  
 * TDA ABB<Tbase>::iterator permite realizar un  
 * recorrido por orden ascendente (según operator<)  
 * de los elementos de un ABB<Tbase>.  
 */  
class iterator {  
public:  
    iterator();  
    iterator(ArbolBinario<Tbase>::Nodo n);  
    iterator(ArbolBinario<Tbase>::iterator it);  
    bool operator!=(const ABB<Tbase>::iterator & it);  
    bool operator==(const ABB<Tbase>::iterator & it);  
    Tbase operator*();  
    iterator operator++();  
};
```

TDA ABB

```
iterator begin();
/** Posición de inicio del recorrido.

    @return Posición de inicio del recorrido.
 */

iterator end();
/** Posición final del recorrido.

    @return Posición final del recorrido.
 */

iterator begin() const;
/** Posición de inicio del recorrido.

    @return Posición de inicio del recorrido.
 */
```

TDA ABB

```
iterator end() const;  
/**  
    Posición final del recorrido.  
  
    @return Posición final del recorrido.  
*/  
  
~ABB();  
/**  
    Destructor.  
*/
```

Ejemplo de uso del TDA ABB

```
/**  
 * Programa ejemplo de prueba del TDA ABB  
 */  
  
#include <iostream>  
#include "abb.h"  
  
  
template <class Tbase>  
ostream & operator<<(ostream & s,  
                      const ABB<Tbase> & abb)  
{  
    ABB<int>::iterator i = abb.begin();  
    while (i != abb.end())  
    {  
        s << *i << ", ";  
        ++i;  
    }  
    s << endl;  
    return s;  
}
```



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

Ejemplo de uso del TDA ABB

```
int main()
{
    ABB<int> abb;

    cout << "Introduce un entero (<0 para terminar): ";
    int e;
    cin >> e;
    while (e > 0)
    {
        abb.Insertar(e);
        cout << "Introduce un entero (<0 para terminar): ";
        cin >> e;
    }

    ABB<int>::iterator i = abb.begin();
    while (i != abb.end())
    {
        cout << *i << ", ";
        ++i;
    }
    cout << endl;
```

Ejemplo de uso del TDA ABB

```
cout << "Buscando datos" << endl;
cout << "Introduce un entero (<0 para terminar): " ;
cin >> e;
while (e > 0)
{
    if (abb.Buscar(e))
        cout << e << " está en el ABB" << endl;
    else
        cout << e << " NO está en el ABB" << endl;
    cout << "Introduce un entero (<0 para terminar): " ;
    cin >> e;
}

cout << "Borrando elementos del ABB:" << endl;
cout << "Introduce un entero (<0 para terminar): " ;
cin >> e;
while (e > 0)
{
    abb.Borrar(e);
    cout << abb;
    cout << "Introduce un entero (<0 para terminar): " ;
    cin >> e;
}

return 0;
}
```

TDA ABB: Representación

```
template <class Tbase>
class ABB{

    ...

    class iterator {

        ...

    private:
        ArbolBinario<Tbase>::iterator introducir eliterador;
    };

private:
    ArbolBinario<Tbase> arbolb;

    void borrar_nodo(ArbolBinario<Tbase>::Nodo n);
    /**
        Elimina un nodo del árbol.
        @param n: Nodo a eliminar. n != NODO_NULO.

        @doc
        Elimina n del árbol receptor.
    */
};


```

TDA ABB: Representación

/*

Función de abstracción:

Cada objeto del tipo rep $r = \{\text{arbolb}\}$ representa al objeto abstracto arbolb .

Invariante de representación:

Para cada nodo n de $r.\text{arbolb}$ se cumple:

- $\text{arbolb.Etiqueta}(n) > \text{arbolb.Etiqueta}(m)$, con m un nodo a la izqda de n .
- $\text{arbolb.Etiqueta}(n) \leq \text{arbolb.Etiqueta}(m)$, con m un nodo a la drcha de n .

*/



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

TDA ABB: Constructores

```
template <class Tbase>
inline ABB<Tbase>::ABB()
{
}
```

```
template <class Tbase>
inline ABB<Tbase>::ABB(const ABB<Tbase> & a)
: arbolb(a.arbolb)
{
}
```

```
template <class Tbase>
inline ABB<Tbase>::ABB(const Tbase & e)
: arbolb(e)
{
}
```

TDA ABB: Existe

```
template <class Tbase>
bool ABB<Tbase>::Existe(const Tbase & e) const
{
    if (arbolb.Nulo())
        return false;

    ArbolBinario<Tbase>::Nodo n = arbolb.Raiz();
    bool encontrado = false;
    while (!encontrado &&
           (n != ArbolBinario<Tbase>::NODO_NULO))
    {
        if (e == arbolb.Etiqueta(n))
            encontrado = true;
        else if (e < arbolb.Etiqueta(n))
            n = arbolb.HijoIzqda(n);
        else
            n = arbolb.HijoDrcha(n);
    }
    return encontrado;
}
```

TDA ABB: Insertar

```
template <class Tbase>
void ABB<Tbase>::Insertar(const Tbase & e)
{
    if (arbolb.Nulo())
    {
        arbolb = ArbolBinario<Tbase>(e);
        return;
    }

    // Buscar la posición en la que insertar:
    // será un hijo de n
    ArbolBinario<Tbase>::Nodo n = arbolb.Raiz();
    bool posicionEncontrada = false;
    while (!posicionEncontrada)
    {
        if (e < arbolb.Etiqueta(n))
        {
            if (arbolb.HijoIzqda(n) != ArbolBinario<Tbase>::NODO_NULO)
                n = arbolb.HijoIzqda(n);
            else
                posicionEncontrada = true;
        }
        else
        {
```

```

        if (arbolb.HijoDrcha(n) !=  

            ArbolBinario<Tbase>::NODO_NULO)  

            n = arbolb.HijoDrcha(n);  

        else  

            posicionEncontrada = true;  

    }  

}  
  

ArbolBinario<Tbase> a(e);  

if (e < arbolb.Etiqueta(n))  

    arbolb.InsertarHijoIzqda(n, a);  

else  

    arbolb.InsertarHijoDrcha(n, a);  

}

```



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.



TDA ABB: Borrar

```
template <class Tbase>
void ABB<Tbase>::Borrar(const Tbase & e)
{
    if (arbolb.Nulo())
        return;

    // Comprobar que la etiqueta "e" está en el árbol
    const ArbolBinario<Tbase>::Nodo NODO_NULO =
        ArbolBinario<Tbase>::NODO_NULO;
    ArbolBinario<Tbase>::Nodo n = arbolb.Raiz();
    bool encontrado = false;
    while (!encontrado && (n != NODO_NULO))
    {
        if (e == arbolb.Etiqueta(n))
            encontrado = true;
        else if (e < arbolb.Etiqueta(n))
            n = arbolb.HijoIzqda(n);

        else
            n = arbolb.HijoDrcha(n);
    }
    if (!encontrado)
        return;
    else
        borrar_nodo(n);
}
```

TDA ABB: borrar_nodo

```
template <class Tbase>
void ABB<Tbase>::borrar_nodo(ArbolBinario<Tbase>::Nodo n)
{
    const ArbolBinario<Tbase>::Nodo NODO_NULO =
        ArbolBinario<Tbase>::NODO_NULO;

    if (arbolb.HijoIzqda(n) == NODO_NULO)
        if (arbolb.HijoDrcha(n) == NODO_NULO)
            { // Primer caso: el nodo es una hoja
                ArbolBinario<Tbase>::Nodo padre =
                    arbolb.Padre(n);

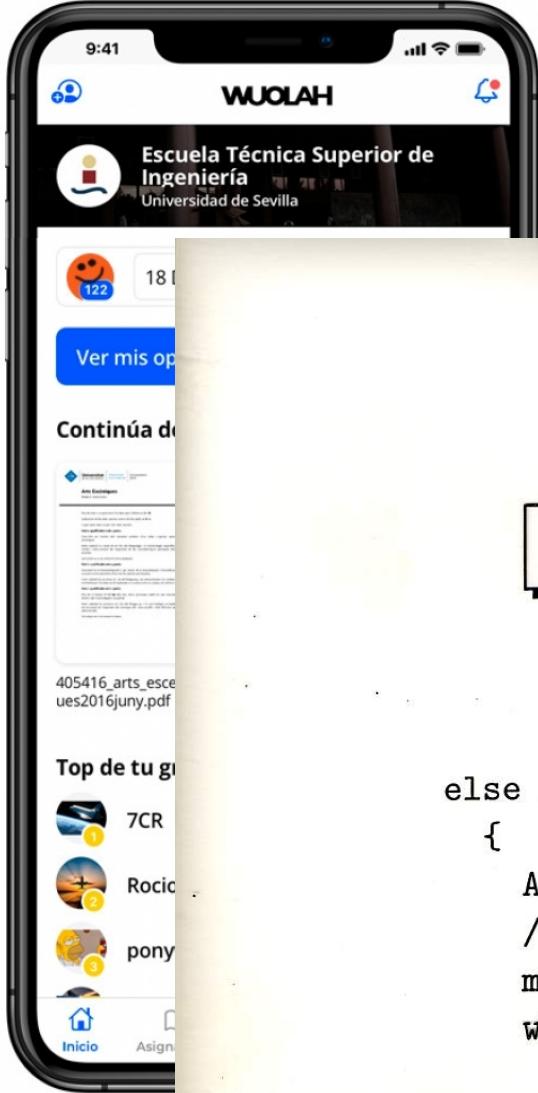
                // Subcaso: el árbol sólo tiene un nodo
                if (padre == NODO_NULO)
                    arbolb = ArbolBinario<Tbase>();
                else if (n == arbolb.HijoIzqda(padre))
                {
                    ArbolBinario<Tbase> a;
                    arbolb.PadarHijoIzqda(padre, a);
                }
                else
                {
                    ArbolBinario<Tbase> a;
                    arbolb.PadarHijoDrcha(padre, a);
                }
            }
        }
}
```

TDA ABB: borrar_nodo

```
else // Segundo caso: El nodo sólo tiene
      // un hijo a la drcha
{
    ArbolBinario<Tbase>::Nodo padre =
        arbolb.Padre(n);
    if (padre != NODO_NULO)
    {
        ArbolBinario<Tbase> a;
        arbolb.PodarHijoDrcha(n, a);
        if (n == arbolb.HijoIzqda(padre))
            arbolb.InsertarHijoIzqda(padre, a);
        else
            arbolb.InsertarHijoDrcha(padre, a);
    }
    else
        arbolb.AsignarSubarbol(arbolb,
            arbolb.HijoDrcha(n));
}
```

TDA ABB: borrar_nodo

```
else // (arbolb.HijoIzqda(n) != NODO_NULO)
    if (arbolb.HijoDrcha(n) == NODO_NULO)
    { // Tercer caso: El nodo sólo tiene un
        // hijo a la izqda
        ArbolBinario<Tbase>::Nodo padre =
            arbolb.Padre(n);
        if (padre != NODO_NULO)
        {
            ArbolBinario<Tbase> a;
            arbolb.PadarHijoIzqda(n, a);
            if (n == arbolb.HijoIzqda(padre))
                arbolb.InsertarHijoIzqda(padre, a);
            else
                arbolb.InsertarHijoDrcha(padre, a);
        }
    else
        arbolb.AsignarSubarbol(arbolb,
            arbolb.HijoIzqda(n));
    }
```



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

TDA ABB: borrar_nodo

```
else // Cuarto caso: el nodo tiene dos hijos
{
    ArbolBinario<Tbase>::Nodo mhi;
    // Buscar el mayor hijo a la izqda
    mhi = arbolb.HijoIzqda(n);
    while (arbolb.HijoDrcha(mhi) != NODO_NULO)
        mhi = arbolb.HijoDrcha(mhi);
    arbolb.Etiqueta(n) = arbolb.Etiqueta(mhi);
    borrar_nodo(mhi);
}
}
```

TDA ABB: Iterador

```
template <class Tbase>
inline ABB<Tbase>::iterator::iterator()
{
}

template <class Tbase>
inline ABB<Tbase>::iterator::iterator(
    ArbolBinario<Tbase>::Nodo n)
: eliterador(n)
{
}

template <class Tbase>
inline ABB<Tbase>::iterator::iterator(
    ArbolBinario<Tbase>::iterator it)
: eliterador(it)
{
}

template <class Tbase>
inline bool ABB<Tbase>::iterator::operator!=(
    const ABB<Tbase>::iterator & it)
{
    return eliterador != it.eliterador;
}
```

TDA ABB: Iterador

```
template <class Tbase>
inline bool ABB<Tbase>::iterator::operator==(const ABB<Tbase>::iterator & it)
{
    return eliterador == it.eliterador;
}

template <class Tbase>
inline Tbase ABB<Tbase>::iterator::operator*()
{
    return *eliterador;
}

template <class Tbase>
inline ABB<Tbase>::iterator
    ABB<Tbase>::iterator::operator++()
{
    return ++eliterador;
}

template <class Tbase>
ABB<Tbase>::iterator ABB<Tbase>::begin()
{
    return iterator(arbolb.beginInOrden());
}
```

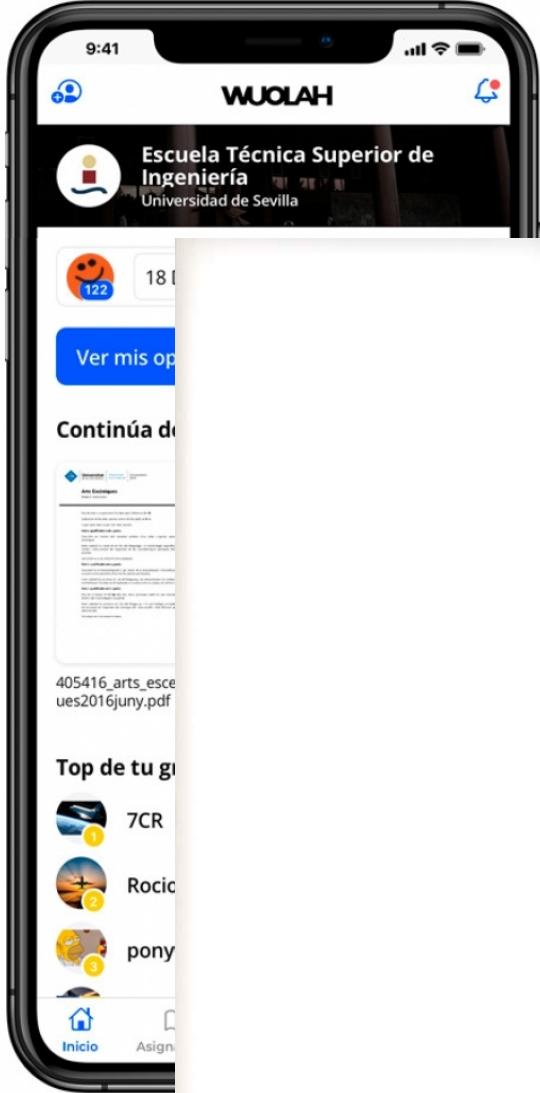
TDA ABB: Iterador

```
template <class Tbase>
ABB<Tbase>::iterator ABB<Tbase>::begin() const
{
    return iterator(arbolb.beginInOrden());
}

template <class Tbase>
ABB<Tbase>::iterator ABB<Tbase>::end()
{
    return iterator(arbolb.endInOrden());
}

template <class Tbase>
ABB<Tbase>::iterator ABB<Tbase>::end() const
{
    return iterator(arbolb.endInOrden());
}

template <class Tbase>
ABB<Tbase>::~ABB()
{
```



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

```
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;
    }
}

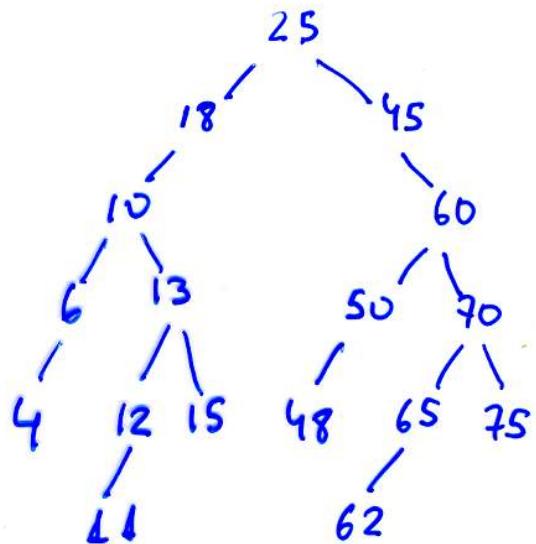
/*----- */
```

```

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){ //La raiz!
i=0; // final!
subir=false;
        } Fin de if (padre==0)
    }
}

```

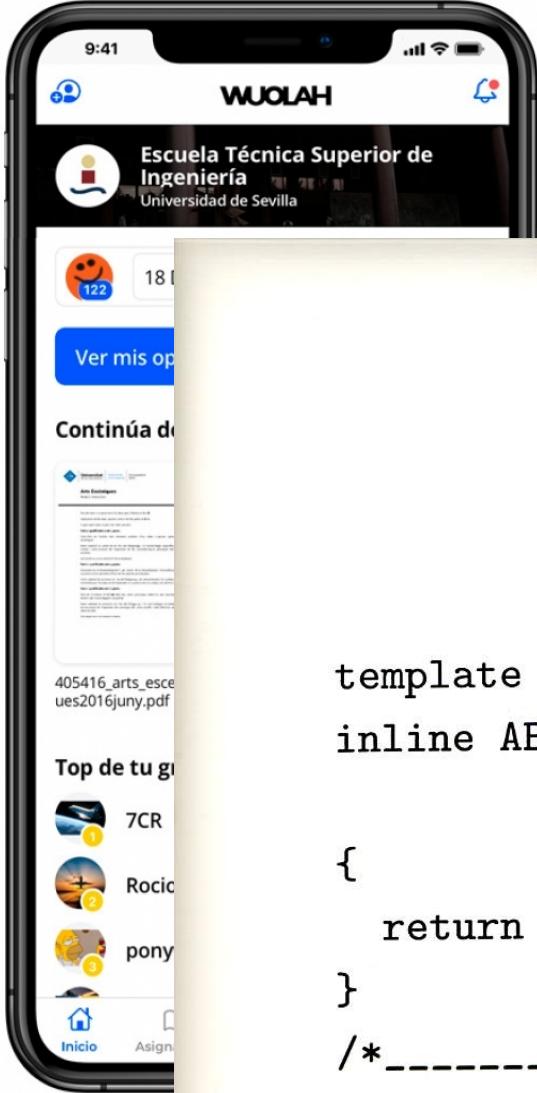


```

    else if (padre->drcha!=i) {
i= padre; // ya se ha recorrido izq
                        de padre
subir=false;
    } // fin de else if (padre->drcha!=i)
    else i= padre; // sigue iterando
    } // fin de while(subir)
} // fin de else { ... }
return i;
} // fin de la función

```

```
/*----- */
```



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.

Available on the
App Store

GET IT ON
Google Play

```
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>
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>
void ABB<Tbase>::enganchar (nodo* & r, nodo **m,
                             int n)
{
    int i;

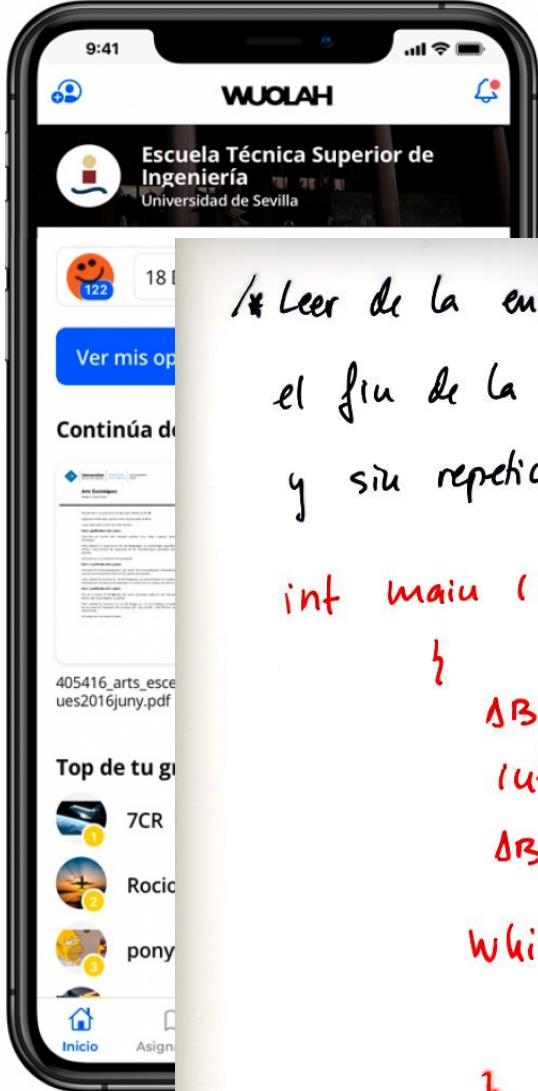
    i=n/2;
    r=m[i];
    if (i>0) {
        enganchar(r->izqda,m,i);
        r->izqda->padre=r;
    }
    else r->izqda= 0;
    if (n-i-1>0) {
        enganchar(r->drcha,m+i+1,n-i-1);
        r->drcha->padre=r;
    }
    else r->drcha=0;
}
/*----- */

```

```

template <class Tbase>
ABB<Tbase>::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();
}
/*----- */
```



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.



/* Leer de la entrada estandar un conjunto de enteros hasta
el fin de la entrada. Despues listarlos todos ordenados
y sin repeticiones */

int main (int argc, char *argv[])

{

 vector<int> a;

 int i;

 vector<int>::iterator p;

 while ((cin >> i))

 a.insertar (i);

}

 for (p = a.primero(); p != a.fin() ; p = a.siguiente(p))

 cout << a.etiqueta(p) << ' ';

 return 0;

}

for (p = a.begin(); p != a.end(); ++p)
 cout << *p << ' ';

/* Leer de la entrada estandar dos conjuntos de enteros
(leer primero el numero de elementos y a continuacion
dichos elementos). Insertarlos en los arboles. Listarlos.
Finalmente unirlos ~~y~~ y listar el resultado de la unión */

```
void listar_abb (const DBB<iut> & a)  
{  
    DBB<iut>::Iterador p;  
    cout << "Arbol con " << a.size() << "elementos" << endl;  
    for (p = a.primer(); p != a.fin(); p = a.siguiente(p))  
        cout << a.etiqueta(p) << ' ';  
    cout << endl;  
}
```

```
int main (iut argc, char * argv [])  
{  
    DBB<iut> a, b;  
    DBB<iut>::Iterador p;  
    iut i, e;  
    cout << "Número de elementos del primer árbol : " << endl;  
    cin >> i;  
    while (i != 0) {  
        cin >> e;  
        a.insertar (e);  
        i --;  
    }
```

```
cout << "Número de elementos del segundo árbol: " << endl;
cin >> i;
while (i != 0) {
    cin >> e;
    b.insertar(e);
    i--;
}
cout << "Primer Árbol: " << endl;
listar_abb(a);
cout << "segundo Árbol: " << endl;
listar_abb(b);
for (p = b.primer(); p != b.fin(); p = b.siguiente(p))
    a.insertar(b.etiqueta(p));
cout << "UNION?" << endl;
listar_abb(a);
return 0;
}
```