WUOLAH

QuesoViejo_
www.wuolah.com/student/QuesoViejo_

236782

Practicas-Arboles-Generales-y-otros-Arboles-Resueltas.pdf

EJERCICIOS RESUELTOS PRÁCTICAS

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.

Available on the App Store
GET IT ON Google Play

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

DESPUÉS DE LOS FINALES...
¡RELÁJATE!

- **480** cartas resultado de mucho amor y birra
- **80 cartas especiales** para animar tus fiestas
- **A partir de 16 años**, de 3 a 15 jugadores
- Perfecto para largas noches de risas con amigos

GUA TA FAC

GUA TA FAC

GUA TA FAC

Estructuras de Datos no Lineales

Práctica 3

Problemas de árboles generales

TRABAJO PREVIO

Antes de asistir a la sesión de prácticas es obligatorio:

1. **Implementar y probar** el TAD *árbol general* con las dos representaciones estudiadas, vector de listas de hijos y enlazada.
2. Imprimir copia de este enunciado.
3. Lectura profunda del mismo.
4. Reflexión sobre el contenido de la práctica y generación de la lista de dudas asociada a dicha práctica y a los problemas que la componen.
5. **Esbozo serio de solución** de los problemas en papel (al menos de los que se hayan entendido).

PASOS A SEGUIR

1. Escribir módulos que contengan las implementaciones de los subprogramas demandados en cada problema.
2. Para cada uno de los problemas escribir un programa de prueba, independiente de la representación del TAD elegida, donde se realicen las llamadas a los subprogramas del paso anterior, comprobando el resultado de salida para una batería suficientemente amplia de casos de prueba.

ENTRADA Y SALIDA DE ÁRBOLES GENERALES

Se proporciona la cabecera `agen_E-S.h` que incluye cuatro funciones genéricas para la lectura y escritura de árboles generales a través de flujos de entrada y salida:

template <typename T> void rellenarAgen (Agen<T>& A, const T& fin)
Pre: *A* está vacío.
Post: Rellena el árbol *A* con la estructura y elementos leídos en preorden de la entrada estándar, usando *fin* como elemento especial para introducir nodos nulos.

template <typename T> void rellenarAgen (istream& is, Agen<T>& A)
Pre: *A* está vacío.
Post: Extrae los nodos de *A* del flujo de entrada *is*, que contendrá el elemento especial que denota un nodo nulo seguido de los elementos en preorden, incluyendo los correspondientes a nodos nulos.

template <typename T> void imprimirAgen (const Agen<T>& A)
Post: Muestra los nodos de *A* en la salida estándar.

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

template <typename T>
void imprimirAgen (ostream& os, const Agen<T>& A, const T& fin)
Post: Inserta en el flujo de salida *os* los nodos de *A* en preorden, precedidos del elemento especial usado para denotar un nodo nulo.

Ejemplo:

```
#include <iostream>
#include <fstream>
#include "agenlis.h"
#include "agen_E-S.h"

using namespace std;

typedef char tElto;
const tElto fin = '#';   // fin de lectura

int main ()
{
   Agen<tElto> A(16), B(16);

   cout << "*** Lectura del árbol A ***\n";
   rellenarAgen(A, fin);     // Desde std::cin

   ofstream fs("agen.dat");  // Abrir fichero de salida.
   imprimirAgen(fs, A, fin); // En fichero.
   fs.close();
   cout << "\n*** Árbol A guardado en fichero agen.dat ***\n";

   cout << "\n*** Lectura de árbol B de agen.dat ***\n";
   ifstream fe("agen.dat");  // Abrir fichero de entrada.
   rellenarAgen(fe, B);      // Desde fichero.
   fe.close();

   cout << "\n*** Mostrar árbol B ***\n";
   imprimirAgen(B);          // En std::cout
}
```

Salida del programa: [Usando el ejemplo de la diapositiva 1 de árboles generales]

```
*** Lectura del árbol A ***
Raiz (Fin = #): f
Hijo izqdo. de f (Fin = #): g
Hijo izqdo. de g (Fin = #): i
Hijo izqdo. de i (Fin = #): #
Hermano drcho. de i (Fin = #): p
Hijo izqdo. de p (Fin = #): #
Hermano drcho. de p (Fin = #): h
Hijo izqdo. de h (Fin = #): j
Hijo izqdo. de j (Fin = #): #
Hermano drcho. de j (Fin = #): #
Hermano drcho. de h (Fin = #): a
Hijo izqdo. de a (Fin = #): r
Hijo izqdo. de r (Fin = #): #
Hermano drcho. de r (Fin = #): #
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

```
Hermano drcho. de a (Fin = #): #
Hermano drcho. de g (Fin = #): n
Hijo izqdo. de n (Fin = #): o
Hijo izqdo. de o (Fin = #): #
Hermano drcho. de o (Fin = #): k
Hijo izqdo. de k (Fin = #): #
Hermano drcho. de k (Fin = #): #
Hermano drcho. de n (Fin = #): b
Hijo izqdo. de b (Fin = #): d
Hijo izqdo. de d (Fin = #): e
Hijo izqdo. de e (Fin = #): #
Hermano drcho. de e (Fin = #): c
Hijo izqdo. de c (Fin = #): #
Hermano drcho. de c (Fin = #): l
Hijo izqdo. de l (Fin = #): #
Hermano drcho. de l (Fin = #): #
Hermano drcho. de d (Fin = #): #
Hermano drcho. de b (Fin = #): #

*** Árbol A guardado en fichero agen.dat ***

*** Lectura de árbol B de agen.dat ***

*** Mostrar árbol B ***
Raiz del arbol: f
Hijo izqdo de f: g
Hijo izqdo de g: i
Hermano derecho de i: p
Hermano derecho de p: h
Hijo izqdo de h: j
Hermano derecho de h: a
Hijo izqdo de a: r
Hermano derecho de g: n
Hijo izqdo de n: o
Hermano derecho de o: k
Hermano derecho de n: b
Hijo izqdo de b: d
Hijo izqdo de d: e
Hermano derecho de e: c
Hermano derecho de c: l
```

Fichero `agen.dat`:

```
#
f g i # p # h j # # a r # # # n o # k # # b d e # c # l # # # #
```

PROBLEMAS

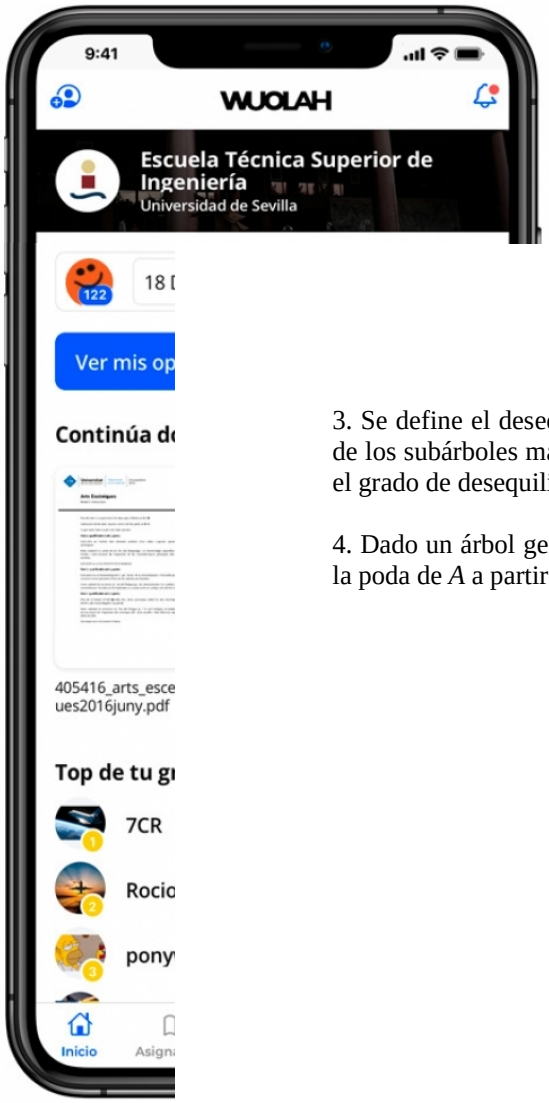
1. Implementa un subprograma que dado un árbol general nos calcule su grado.

2. Implementa un subprograma que dados un árbol y un nodo dentro de dicho árbol determine la profundidad de éste nodo en el árbol.

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play

3. Se define el desequilibrio de un árbol general como la máxima diferencia entre las alturas de los subárboles más bajo y más alto de cada nivel. Implementa un subprograma que calcule el grado de desequilibrio de un árbol general.

4. Dado un árbol general de enteros A y un entero x, implementa un subprograma que realice la poda de A a partir de x. Se asume que no hay elementos repetidos en A.

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

```
/* "AgenVec.h"

IMPLEMENTACIÓN VECTORIAL DEL ÁRBOL GENERAL
*/

#ifndef AGENVEC
#define AVECGEN

#include "ListaDin.h"
#include <cassert>

template <typename T> class Agen{
public:
    typedef int nodo;    //índice

    static const nodo NODO_NULO;     //Inicializado más tarde

    explicit Agen(size_t maxNodos); //Ctor de conversión
    Agen(const Agen<T>& A);
    Agen& operator=(const Agen<T>& A);


    void insertarRaiz(const T& e);
    void insertarHijoIzqdo(nodo n, const T& e);
    void insertarHermDrcho(nodo n, const T& e);

    void eliminarHijoIzqdo(nodo n);
    void eliminarHermDrcho(nodo n);
    void eliminarRaiz();

    bool arbolVacio()const;
    const T& elemento(nodo n)const;
    T& elemento(nodo n);

    nodo raiz()const;
    nodo padre(nodo n)const;
    nodo hijoIzqdo(nodo n)const;
    nodo hermDrcho(nodo n)const;

    ~Agen();
private:
    struct celda{
        T elto;
        nodo padre;
        Lista<nodo> hijos;
    };

    int numNodos;    //Necesario para saber si existe raiz o no
    int maxNodos;    //Tamaño del vector de celdas
    celda* nodos;    //Vector de celdas


};

template <typename T>
const typename Agen<T>::nodo Agen<T>::NODO_NULO(-1);   //Inicializo NODO_NULO a -1

template <typename T>
Agen<T>::Agen(size_t maxNodos): maxNodos(maxNodos), nodos(new celda[maxNodos]),
numNodos(0){
    for(int i = 0; i < maxNodos ; i++){
        nodos[i].padre = NODO_NULO; //Marcamos los padres como nulos(para indicar que la
celda está vacía)
    }
}

template <typename T>
Agen<T>::Agen(const Agen<T>& A) : maxNodos(A.maxNodos), nodos(new celda[A.maxNodos]),
numNodos(A.numNodos){

    for(nodo i = 0 ; i < maxNodos ; i++){
        nodos[i] = A.nodos[i];   //Hay que copiar el vector entero porque no sabemos
cuáles son celdas válidas y cuáles no.
    }
}
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

```
template <typename T>
Agen<T>& Agen<T>::operator=(const Agen<T>& A){

    if(this != &A){ //Para evitar autoasignación
        if(maxNodos != A.maxNodos){
            delete[] nodos;
            maxNodos = A.maxNodos;
            nodos = new celda[A.maxNodos];
        }
        numNodos = A.numNodos;
        for(nodo i = 0 ; i < maxNodos ; i++){
            nodos[i] = A.nodos[i];
        }


    }
    return *this;
}

template <typename T>
void Agen<T>::insertarRaiz(const T& e){
    assert(numNodos == 0);

    nodos[0].elto = e;
    numNodos = 1;
}

template <typename T>
void Agen<T>::insertarHijoIzqdo( Agen<T>::nodo n, const T& e){
    assert(numNodos>0);
    assert(n>=0 && n < maxNodos);     //n es un nodo válido
    assert(n == 0 || nodos[n].padre != NODO_NULO); //Es la raiz o es una celda con valor
    assert(numNodos < maxNodos); //Árbol no lleno

    nodo hizqdo;

    for(hizqdo = 1 ; nodos[hizqdo].padre != NODO_NULO ; hizqdo++);  //hizqdo se para en
la primera posición libre del vector

    nodos[hizqdo].elto = e;
    nodos[hizqdo].padre = n;

    Lista<nodo>& Lh = nodos[n].hijos;
    Lh.insertar(hizqdo,Lh.primera());
    std::cout<<"El primer elemento de la lista es: "<<Lh.elemento(Lh.primera())<<std::endl;
    ++numNodos;
}

template <typename T>
void Agen<T>::insertarHermDrcho( Agen<T>::nodo n, const T& e){

    assert(n>0 && n < maxNodos);     //OJO, la raiz no tiene hermano derecho
    assert(numNodos<maxNodos);  //Árbol no lleno
    assert(nodos[n].padre != NODO_NULO); //Que n sea un nodo válido, no puede ser la raiz

    nodo hedrcho;
    for(hedrcho = 1 ; nodos[hedrcho].padre != NODO_NULO ; hedrcho++);     //Busca la
primera posición libre en el vector

    nodos[hedrcho].elto = e;
    nodos[hedrcho].padre = nodos[n].padre;

    Lista<nodo>& Lhp = nodos[nodos[n].padre].hijos;

    std::cout<<"Es la posicion de n fin? "<<(Lhp.buscar(n)==Lhp.fin())<<std::endl;
    Lhp.insertar(hedrcho,Lhp.siguiente(Lhp.buscar(n)));
    ++numNodos;

}

template <typename T>
void Agen<T>::eliminarHijoIzqdo( Agen<T>::nodo n){

    assert(numNodos > 0);
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

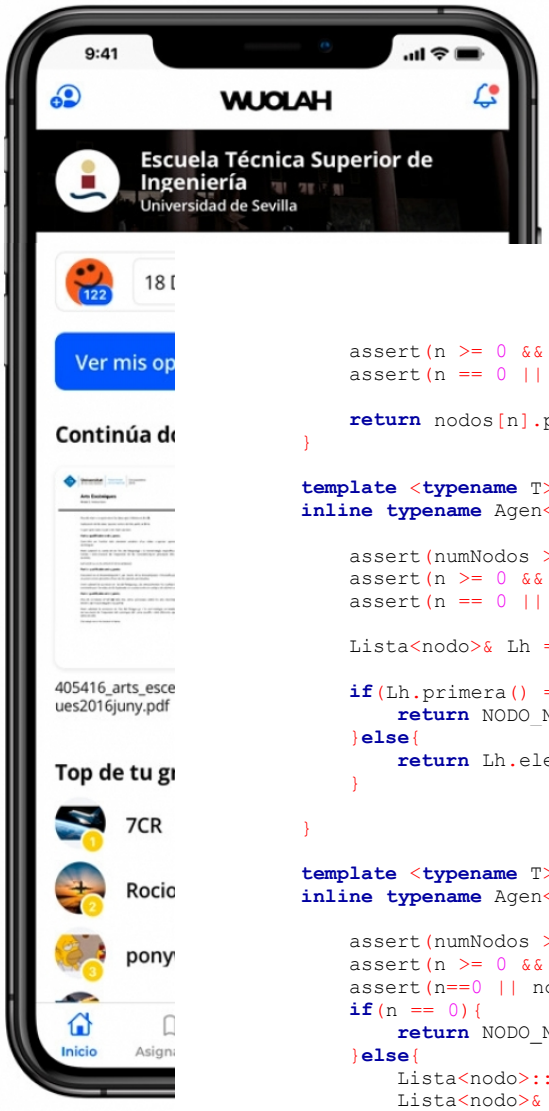
```
    assert(n >= 0 && n < maxNodos);
    assert(n == 0 || nodos[n].padre != NODO_NULO);     //Que n sea un nodo ocupado

    Lista<nodo>& Lh = nodos[n].hijos;
    nodo hizqdo;

    assert(Lh.primera() != Lh.fin());    //Para comprobar que n tiene hijos
    hizqdo = Lh.elemento(Lh.primera());
    assert(nodos[hizqdo].hijos.primera() == nodos[hizqdo].hijos.fin()); //Comprobamos que
el hijoizqdo de n no tiene hijos

    nodos[hizqdo].padre = NODO_NULO;
    Lh.eliminar(Lh.primera());
    --numNodos;
}

template <typename T>
void Agen<T>::eliminarHermDrcho( Agen<T>::nodo n){

    assert(n>0 && n < maxNodos);
    assert(nodos[n].padre != NODO_NULO); //Que n sea un nodo con valor

    nodo hdrcho;
    Lista<nodo>::posicion p;
    Lista<nodo>& Lhp = nodos[nodos[n].padre].hijos;

    p = Lhp.siguiente(Lhp.buscar(n));
    assert(p != Lhp.fin()); //Que p está en la lista de hijos del padre

    hdrcho = Lhp.elemento(p);
    assert(nodos[hdrcho].hijos.primera() == nodos[hdrcho].hijos.fin());
    nodos[hdrcho].padre = NODO_NULO;
    Lhp.eliminar(p);
    --numNodos;
}

template <typename T>
void Agen<T>::eliminarRaiz(){

    assert(numNodos == 1);
    numNodos = 0;
}

template <typename T>
bool Agen<T>::arbolVacio()const{
    return (numNodos == 0);
}

template <typename T>
const T& Agen<T>::elemento( Agen<T>::nodo n)const{
    assert(numNodos>0);
    assert(n>= 0 && n < maxNodos);
    assert(n==0 ||nodos[n].padre != NODO_NULO);


    return nodos[n].elto;
}

template <typename T>
T& Agen<T>::elemento( Agen<T>::nodo n){
    assert(numNodos>0);
    assert(n>= 0 && n < maxNodos);
    assert(n==0 ||nodos[n].padre != NODO_NULO);

    return nodos[n].elto;
}

template <typename T>
inline typename Agen<T>::nodo Agen<T>::raiz()const{
    return (numNodos>0) ? 0 : NODO_NULO;
}

template <typename T>
inline typename Agen<T>::nodo Agen<T>::padre(Agen<T>::nodo n)const{
    assert(numNodos>0);
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play




```
    assert(n >= 0 && n < maxNodos);
    assert(n == 0 || nodos[n].padre != NODO_NULO);

    return nodos[n].padre;  //Si n es la raiz, devuelve NODO_NULO
}

template <typename T>
inline typename Agen<T>::nodo Agen<T>::hijoIzqdo(Agen<T>::nodo n)const{

    assert(numNodos > 0);
    assert(n >= 0 && n < maxNodos);
    assert(n == 0 || nodos[n].padre != NODO_NULO);    //Que n sea un nodo con valor

    Lista<nodo>& Lh = nodos[n].hijos;

    if(Lh.primera() == Lh.fin()){
        return NODO_NULO;
    }else{
        return Lh.elemento(Lh.primera());
    }

}

template <typename T>
inline typename Agen<T>::nodo Agen<T>::hermDrcho(Agen<T>::nodo n)const{

    assert(numNodos > 0);
    assert(n >= 0 && n < maxNodos);
    assert(n==0 || nodos[n].padre != NODO_NULO);
    if(n == 0){
        return NODO_NULO;
    }else{
        Lista<nodo>::posicion p;
        Lista<nodo>& Lhp = nodos[nodos[n].padre].hijos;

        p = Lhp.siguiente(Lhp.buscar(n));

        if(p == Lhp.fin()){
            return NODO_NULO;
        }else{
            return Lhp.elemento(p);
        }
    }
}

template <typename T>
Agen<T>::~Agen(){

    delete[] nodos;
}


#endif // AGENVEC
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

```
/* "AgenDin.h"

IMPLEMENTACIÓN DINÁMICA DEL ÁRBOL GENERAL
*/

#ifndef ARBOLDINAMICO
#define ARBOLDINAMICO

#include <cassert>

template <typename T>
class Agen{
    struct celda;//Declaración adelantada privada
public:
    typedef celda* nodo;
    static const nodo NODO_NULO;

    Agen();
    Agen(const Agen<T>& A);
    Agen<T>& operator=(const Agen<T>& A);

    void insertarRaiz(const T& e);
    void insertarHijoIzqdo(nodo n,const T& e);
    void insertarHermDrcho(nodo n,const T& e);

    void eliminarHijoIzqdo(nodo n);
    void eliminarHermDrcho(nodo n);
    void eliminarRaiz();

    bool arbolVacio()const;
    const T& elemento(nodo n)const;
    T& elemento(nodo n);

    nodo raiz()const;
    nodo padre(nodo n)const;
    nodo hijoIzqdo(nodo n)const;
    nodo hermDrcho(nodo n)const;

    ~Agen();
private:
    struct celda{
        T elto;
        nodo padre, hizq, heder;
        celda(const T& e, nodo p = NODO_NULO) : elto(e), padre(p), hizq(NODO_NULO),
heder(NODO_NULO){}
    };

    nodo r;      //Para acceder al árbol

    void destruirNodos(nodo& n);
    nodo copiar(nodo n);

};


template <typename T>
void Agen<T>::destruirNodos(Agen<T>::nodo& n){

    if(n != NODO_NULO){ //Caso Base
        if(n->hizq != NODO_NULO){
            nodo hedrcho = n->hizq->heder;
            while(hedrcho != NODO_NULO){     //Destruye todos los subarboles excepto el
primero
                n->hizq->heder = hedrcho->heder;
                destruirNodos(hedrcho);
                hedrcho = n->hizq->heder;
            }
            destruirNodos(n->hizq); //Destruye el primer subarbol
        }
        delete(n);
        n = NODO_NULO;
    }

}
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Descarga la app de Wuolah desde tu store favorita

```
template <typename T>
typename Agen<T>::nodo Agen<T>::copiar(Agen<T>::nodo n){
    nodo m = NODO_NULO;
    if(n != NODO_NULO){
        m=new celda(n->elto);    //Copio n
        if(n->hizq != NODO_NULO){
            m->hizq = copiar(n->hizq);   //Copio el primer subarbol
            m->hizq->padre = m;

            //Copiar el resto de los subarboles
            nodo hijo = m->hizq;
            nodo hedrcho = n->hizq->heder;

            while(hedrcho != NODO_NULO){
                hijo = hijo->heder = copiar(hedrcho);
                hijo->padre = m;
                hedrcho = hedrcho->heder;

            }
        }

    }
    return m;
}

template <typename T>
const typename Agen<T>::nodo Agen<T>::NODO_NULO(0);

template <typename T>
inline Agen<T>::Agen(): r(NODO_NULO){}

template <typename T>
Agen<T>::Agen(const Agen<T>& A){
    r = copiar(A.r);
}

template <typename T>
Agen<T>& Agen<T>::operator=(const Agen<T>& A){

    if(this != &A){
        this->~Agen();
        copiar(A.r);
    }
    return *this;
}

template <typename T>
void Agen<T>::insertarRaiz(const T& e){

    assert(r == NODO_NULO);
    r = new celda(e);
}

template <typename T>
void Agen<T>::insertarHijoIzqdo(Agen<T>::nodo n, const T& e){

    assert(n != NODO_NULO);

    nodo hizqdo = n->hizq;

    n->hizq = new celda(e,n);
    n->hizq->heder = hizqdo;


}

template <typename T>
void Agen<T>::insertarHermDrcho(Agen<T>::nodo n, const T& e){

    assert(n != NODO_NULO);
    assert(n != r);

    nodo hedrcho = n->heder;
    n->heder = new celda(e,n->padre);
    n->heder->heder = hedrcho;
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

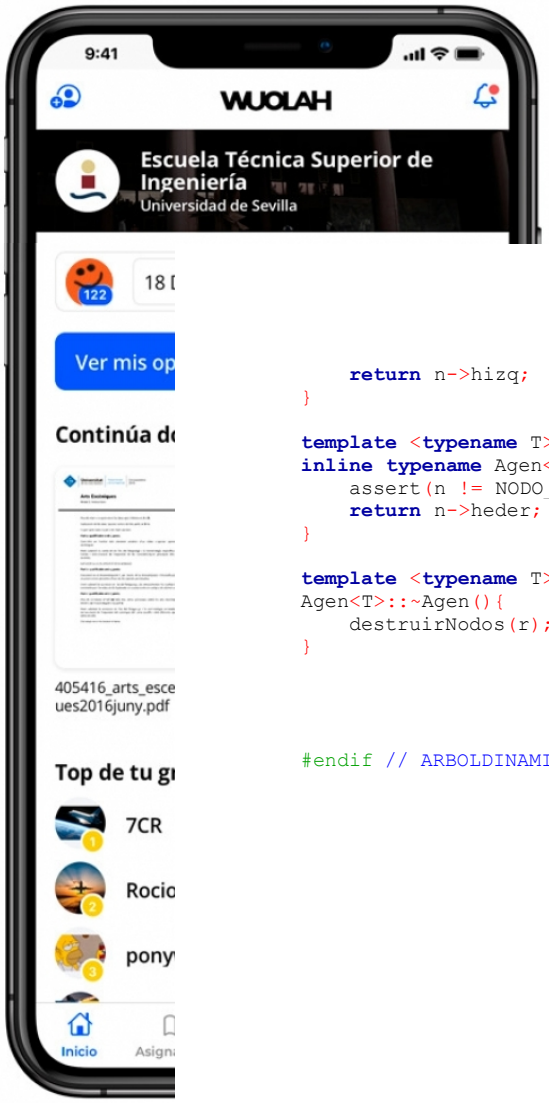
```
}


template <typename T>
void Agen<T>::eliminarRaiz(){

    assert(r != NODO_NULO);
    assert(r->hizq == NODO_NULO);
    delete r;
    r = NODO_NULO;

}

template <typename T>
void Agen<T>::eliminarHijoIzqdo(Agen<T>::nodo n){

    assert(n != NODO_NULO);
    assert(n->hizq != NODO_NULO);
    assert(n->hizq->hizq == NODO_NULO);

    nodo hizqdo = n->hizq;
    n->hizq = hizqdo->heder;
    delete hizqdo;

}

template <typename T>
void Agen<T>::eliminarHermDrcho( Agen<T>::nodo n){
    assert(n != NODO_NULO);
    assert(n->heder != NODO_NULO);
    assert(n->heder->hizq == NODO_NULO);

    nodo hedrcho = n->heder;
    n->heder = hedrcho->heder;
    delete hedrcho;

}

template <typename T>
inline bool Agen<T>::arbolVacio()const{
    return (r == NODO_NULO);
}

template <typename T>
inline const T& Agen<T>::elemento(Agen<T>::nodo n)const{
    assert(n != NODO_NULO);

    return n->elto;
}

template <typename T>
inline T& Agen<T>::elemento(Agen<T>::nodo n){
    assert(n != NODO_NULO);

    return n->elto;
}

template <typename T>
inline typename Agen<T>::nodo Agen<T>::raiz()const{
    return r;
}

template <typename T>
inline typename Agen<T>::nodo Agen<T>::padre(Agen<T>::nodo n)const{
    assert (n != NODO_NULO);

    return n->padre;
}

template <typename T>
inline typename Agen<T>::nodo Agen<T>::hijoIzqdo(Agen<T>::nodo n)const{
    assert(n != NODO_NULO);
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play




```
    return n->hizq;
}

template <typename T>
inline typename Agen<T>::nodo Agen<T>::hermDrcho(Agen<T>::nodo n)const{
    assert(n != NODO_NULO);
    return n->heder;
}

template <typename T>
Agen<T>::~Agen(){
    destruirNodos(r); //Destruye los nodos de r hacia abajo (r incluido)
}



#endif // ARBOLDINAMICO
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.
WUOLAH

```
/* "agen_E-S.h"

Funciones de E/S para árboles generales
*/

#ifndef AGEN_E_S_H
#define AGEN_E_S_H

#include <cassert>
#include <iostream>
#include <fstream>

using std::istream;
using std::ostream;
using std::cin;
using std::cout;
using std::endl;

template <typename T>
void rellenarAgen(Agen<T>& A, const T& fin)
// Pre: A está vacío.
// Post: Rellena el árbol A con la estructura y elementos
//       leídos en preorden de la entrada estándar, usando
//       fin como elemento especial para introducir nodos nulos.
{
   T e;

   assert(A.arbolVacio());
   cout << "Raiz (Fin = " << fin << "): "; cin >> e;
   if (e != fin) {
      A.insertarRaiz(e);
      rellenarDescendientes(A.raiz(), A, fin);
   }
}

template <typename T>
void rellenarDescendientes(typename Agen<T>::nodo n, Agen<T>& A, const T& fin)
// Pre: n es un nodo hoja de A.
// Post: Lee de la entrada estándar y en preorden los descendientes de n,
//       usando fin para introducir nodos nulos.
{
   T ehi, ehd;

   assert(A.hijoIzqdo(n) == Agen<T>::NODO_NULO);
   cout << "Hijo izqdo. de " << A.elemento(n)
        << " (Fin = " << fin << "): ";
   cin >> ehi;
   if (ehi != fin) {
      A.insertarHijoIzqdo(n, ehi);
      rellenarDescendientes(A.hijoIzqdo(n), A, fin);
      typename Agen<T>::nodo hijo = A.hijoIzqdo(n);
      cout << "Hermano drcho. de " << A.elemento(hijo)
           << " (Fin = " << fin << "): ";
      cin >> ehd;
      while (ehd != fin) {
         A.insertarHermDrcho(hijo, ehd);
         rellenarDescendientes(A.hermDrcho(hijo), A, fin);
         hijo = A.hermDrcho(hijo);
         cout << "Hermano drcho. de " << A.elemento(hijo)
              << " (Fin = " << fin << "): ";
         cin >> ehd;
      }
   }
}

template <typename T>
void rellenarAgen(istream& is, Agen<T>& A)
// Pre: A está vacío.
// Post: Extrae los nodos de A del flujo de entrada is,
//       que contendrá el elemento especial que denota
//       un nodo nulo seguido de los elementos en preorden,
//       incluyendo los correspondientes a nodos nulos.
{
   T e, fin;
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

```
   assert(A.arbolVacio());
   if (is >> fin && is >> e && e != fin) {
      A.insertarRaiz(e);
      rellenarDescendientes(is, A.raiz(), A, fin);
   }
}

template <typename T>
void rellenarDescendientes(istream& is, typename Agen<T>::nodo n, Agen<T>& A, const T& fin)
// Pre: n es un nodo hoja de A.
// Post: Lee del flujo de entrada is y en preorden los descendientes de n,
//       usando fin para los nodos nulos.
{
   T ehi, ehd;
   assert(A.hijoIzqdo(n) == Agen<T>::NODO_NULO);
   if (is >> ehi && ehi != fin) {
      A.insertarHijoIzqdo(n, ehi);
      rellenarDescendientes(is, A.hijoIzqdo(n), A, fin);
      typename Agen<T>::nodo hijo = A.hijoIzqdo(n);
      while (is >> ehd && ehd != fin) {
         A.insertarHermDrcho(hijo, ehd);
         rellenarDescendientes(is,A.hermDrcho(hijo), A, fin);
         hijo = A.hermDrcho(hijo);
      }
   }
}

template <typename T>
void imprimirAgen(const Agen<T>& A)
// Post: Muestra los nodos de A en la salida estándar
{
   if (!A.arbolVacio())
   {
      cout << "Raiz del arbol: "
           << A.elemento(A.raiz()) << endl;
      imprimirDescendientes(A.raiz(), A);
   }
   else
      cout << "Arbol vacio\n";
}

template <typename T>
void imprimirDescendientes(typename Agen<T>::nodo n, const Agen<T>& A)
// Post: Muestra los descendientes de n en la salida estándar.
{
   if (A.hijoIzqdo(n) != Agen<T>::NODO_NULO)
   {
      typename Agen<T>::nodo hijo = A.hijoIzqdo(n);
      cout << "Hijo izqdo de " << A.elemento(n) << ": "
           << A.elemento(hijo) << endl;
      while (hijo != Agen<T>::NODO_NULO) {
         imprimirDescendientes(hijo, A);
         if (A.hermDrcho(hijo) != Agen<T>::NODO_NULO)
            cout << "Hermano derecho de " << A.elemento(hijo) << ": "
                 << A.elemento(A.hermDrcho(hijo)) << endl;
         hijo = A.hermDrcho(hijo);
      }
   }
}

template <typename T>
void imprimirAgen(ostream& os, const Agen<T>& A, const T& fin)
// Post: Inserta en el flujo de salida os los nodos de A en preorden,
//       precedidos del elemento especial usado para denotar un nodo nulo.
{
   if (!A.arbolVacio())
   {
      os << fin << endl
         << A.elemento(A.raiz()) << ' ';
      imprimirDescendientes(os, A.raiz(), A, fin);
      os << endl;
   }
}

template <typename T>
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Descarga la app de Wuolah desde tu store favorita

```
void imprimirDescendientes(ostream& os, typename Agen<T>::nodo n, const Agen<T>& A, const
T& fin)
// Post: Inserta en el flujo de salida os y en preorden
//       los descendientes de n, usando fin como nodo nulo.
{
   typename Agen<T>::nodo hijo = A.hijoIzqdo(n);
   while (hijo != Agen<T>::NODO_NULO) {
      os << A.elemento(hijo) << ' ';
      imprimirDescendientes(os, hijo, A, fin);
      hijo = A.hermDrcho(hijo);
   }
   os << fin << ' ';
}

#endif   // AGEN_E_S_H
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play

```
/* "AgenDin.h"

IMPLEMENTACIÓN DINÁMICA DEL TAD COLA
*/

#ifndef COLA_DIN_H
#define COLA_DIN_H
#include <cassert>

template <typename T> class Cola{
public:
    Cola();
    Cola(const Cola<T>& C);
    Cola<T>& operator=(const Cola<T>&);

    bool vacia() const;

    const T& frente() const;
    void push(const T& x);
    void pop();

    ~Cola(); //Destructor


private:
    struct nodo{
        T elto;
        nodo* sig;
        nodo(const T& e, nodo* p = 0): elto(e), sig(p){}     //Ctor
    };

    nodo *inicio, *fin;

    void copiar(const Cola<T>& C);
};

template <typename T>
inline void Cola<T>::copiar(const Cola<T>& C){
    if(C.inicio){//Que no esté vacía
        inicio = fin = new nodo(C.inicio->elto);
        //Copio el primer elemento y hago que inicio y fin apunten ambos a él

        //Ahora copio el resto

        for(nodo* p = C.inicio ->sig ; p ; p = p->sig){ //Cond: mientras p no sea vacío,
va recorriendo la lista
            fin->sig = new nodo(p->elto);
            fin = fin->sig;
        }
    }
}

template <typename T>
inline Cola<T>::Cola(): inicio(0), fin(0){}

template <typename T>
inline Cola<T>::Cola(const Cola<T>& C): inicio(0), fin(0){
    copiar(C);
}

template <typename T>
inline Cola<T>& Cola<T>::operator=(const Cola<T>& C){

    if(this != &C){ //Para evitar la autoasignación
       this->~Cola();    //Destruye la cola actual
        copiar(C);
    }

}

template <typename T>
inline bool Cola<T>::vacia() const{
    return inicio == 0;
}
```

WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

```
template <typename T>
inline void Cola<T>::pop() {
    assert(!vacia());
    nodo* aux = inicio;
    inicio = inicio->sig;
    if(!inicio) fin = 0;     //Si solo había un elemento, ambos apuntaban a el.
Ahorainicio apunta a 0, luego fin también
    delete aux;
}

template <typename T>
inline void Cola<T>::push(const T& x){

    nodo* aux = new nodo(x);
    if(!inicio){
        inicio = fin = aux;
    }else{
        fin = fin->sig = aux;
    }
}

template <typename T>
inline const T& Cola<T>::frente() const{
    assert(!vacia());
    return inicio->elto;
}

template <typename T>
inline Cola<T>::~Cola(){
    nodo* p;

    while(inicio){
        p = inicio;
        inicio = inicio->sig;
        delete p;
    }
    fin = 0;
}


#endif // COLA_DIN_H
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

```
/* "ListaDin.h"

IMPLEMENTACIÓN DINÁMICA DEL TAD LISTA
*/

#ifndef LISTA_ENLA_H
#define LISTA_ENLA_H
#include <cassert>

template <typename T> class Lista{
    struct nodo; //Declaración adelantada privada
public:
    typedef nodo* posicion; // "posicion" es un nodo* que representa la posicion de un
elemento (puntero al nodo anterior)
    Lista();     //Constructor
    Lista(const Lista<T>& l);    //Constructor de copia
    Lista<T>& operator=(const Lista<T>& l);   //Operador de asignación

    void insertar(const T& x, posicion p);
    void eliminar(posicion p);

    const T& elemento(posicion p) const; //Devuelve el elemento de esa posicion. L
    T& elemento(posicion p);    //Devuelve el elemento de esa posición L/E


    posicion buscar(const T& x)const;
    posicion anterior(posicion p)const;
    posicion siguiente(posicion p)const;
    posicion primera() const;
    posicion fin() const; //Posición después del último
    ~Lista(); //Destructor
private:
    struct nodo{
        T elto;
        nodo* sig;
        nodo(const T& e, nodo* p = 0): elto(e), sig(p){} //Constructor. Por defecto el
siguiente es 0 (Cuando se crea lista vacia)
    };

    nodo* L; //Lista enlazada de nodos
    void copiar(const Lista<T>& l); //Simplemente se usa esta función para no copiar en
la implementación varias veces lo mismo

};

//Esta función solo se encarga de copiar. Comprobaciones del tipo l != l, etc... se hacen
antes de llamarla
template <typename T>
void Lista<T>::copiar(const Lista<T>& l){

        L = new nodo(T());   //Se crea el nodo cabecera, llamando a su vez al ctor de T
        nodo* q = L;

        //                      piensa que el nodo al que apunta l es el de cabecera que
lo acabo de crear arriba
        for(Lista<T>::posicion r = l.L->sig; r ; r=r->sig){
            q->sig = new nodo(r->elto);
            q=q->sig;
        }
}

template <typename T>
Lista<T>::Lista(): L(new nodo(T())){};     //Creo solo el nodo de cabecera

template <typename T>
inline Lista<T>::Lista(const Lista<T>& l){
    copiar(l);       //Copia l en la lista que llama al método copiar(this)
}

template <typename T>
Lista<T>& Lista<T>::operator =(const Lista<T>& l){

    if(this != &l){      //Para evitar autoasignación
    this->~Lista();      //Eliminamos la memoria ocupada por la lista que llama al método
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

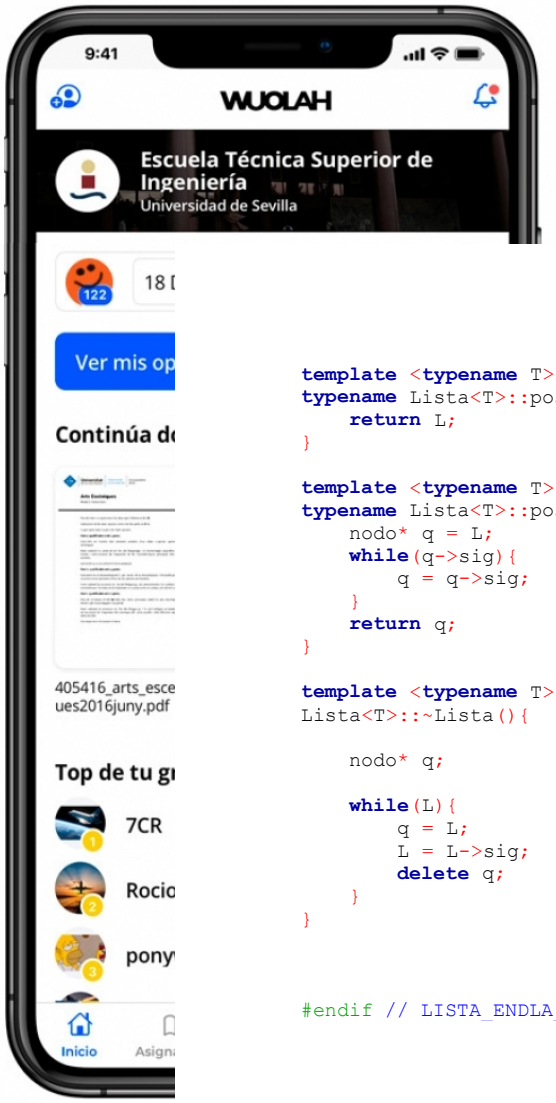
```
    copiar(l);    //Hago uso de la funcion copiar que para algo la he hecho. Copia l en la
lista que lo llama (this)
    }
    return *this;
}

template <typename T>
void Lista<T>::insertar (const T& x, Lista<T>::posicion p){
    p->sig = new nodo(x,p->sig);     //El nodo anterior apunta a un nuevo nodo que
contiene al elemento x y apunta a donde
                                     //antes apuntaba p->sig
}

template <typename T>
void Lista<T>::eliminar (Lista<T>::posicion p){
    assert(p->sig);     //Comprobamos que p no sea fin       (fin: nodo con elto pero cuyo
puntero siguiente es 0)
                        //por definicion p apunta al nodo anterior. Fin no es nodo
anterior a ningun nodo

    nodo* q = p->sig;    //q apunta al nodo que quiero eliminar
    p->sig = q->sig;        //Ahora p->sig apunta no al siguiente inmediato, sino a otro más
    delete q;    //Elimino el nodo que quería eliminar

}


template <typename T>
const T& Lista<T>::elemento(Lista<T>::posicion p)const{       //Primer const para el T que
devuelve, el segundo porque no modifica la Lista
    assert(p->sig); //p no es fin
    return p->sig->elto;

}

template <typename T>
inline T& Lista<T>::elemento(Lista<T>::posicion p){
    assert(p->sig);      //p no es fin

    return p->sig->elto;
}

template <typename T>
typename Lista<T>::posicion Lista<T>::buscar(const T& x)const{
    bool encontrado = false;
    nodo* q = L;

    while( q->sig && !encontrado){
        if(q->sig->elto == x){
            encontrado = true;
        }else{
            q = q->sig;
        }
    }

    return q;

}
template <typename T>
typename Lista<T>::posicion Lista<T>::siguiente(Lista<T>::posicion p)const{
    assert(p->sig); //Que p no sea fin
    return p->sig;
}

template <typename T>
typename Lista<T>::posicion Lista<T>::anterior(Lista<T>::posicion p)const{
    assert(L != p);   //Que p no sea el nodo cabecera

    nodo* q = L;
    while(q->sig != p){
        q = q->sig;
    }

    return q;
}
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Descarga la app de Wuolah desde tu store favorita

Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play




```
template <typename T>
typename Lista<T>::posicion Lista<T>::primera()const{
    return L;
}

template <typename T>
typename Lista<T>::posicion Lista<T>::fin()const {
    nodo* q = L;
    while(q->sig){
        q = q->sig;
    }
    return q;
}

template <typename T>
Lista<T>::~Lista(){

    nodo* q;

    while(L){
        q = L;
        L = L->sig;
        delete q;
    }
}


#endif // LISTA_ENDLA_H
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

```
/*
Ejercicio 1 Práctica 3

QuesoViejo_


Ejemplo de contenido del fichero NUMS.dat:

#
f g i # p # h j # # a r # # # n o # k # # b d e # c # l # # # #

*/

#include <iostream>
#include "AgenDin.h"
#include "agen_E-S.h"
#include "ColaDin.h"


template <typename T>
int numhijos (typename Agen<T>::nodo n, const Agen<T>& A);

template <typename T>
int CalcularGrado(const Agen<T>& A);

template <typename T>
int CalcGrado_rec(typename Agen<T>::nodo n, const Agen<T>& A);

int Maximo(int n1, int n2);

using namespace std;

int main()
{

    Agen<char> A;



    cout << "\n*** Lectura de árbol A desde fichero ***\n";
    ifstream fe("NUMS.dat"); // Abrir fichero de entrada.
    rellenarAgen(fe, A); // Desde fichero.
    fe.close();

    cout << "\n*** Mostrar árbol A ***\n";
    imprimirAgen(A); // En std::cout

    int resultado;
    resultado = CalcularGrado(A);

    cout<<"El resultado es: "<<resultado<<endl;


}

template <typename T>
int CalcularGrado(const Agen<T>& A)
{
    return CalcGrado_rec(A.raiz(), A);
}

template <typename T>
int CalcGrado_rec(typename Agen<T>::nodo n, const Agen<T>& A)
{

    int nhijos,grado_max,resultado;
    typename Agen<T>::nodo hijo;

    if(n == Agen<T>::NODO_NULO)
    {
        return 0;
    }
    else
    {
        grado_max = 0;
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

```
        hijo = A.hijoIzqdo(n);

        while(hijo != Agen<T>::NODO_NULO)
        {
            grado_max = Maximo(grado_max,CalcGrado_rec(hijo,A));
            hijo = A.hermDrcho(hijo);
        }
        nhijos = numhijos(n,A); //Contarlo en el bucle

        return Maximo(grado_max,nhijos);   //El maximo grado de ese subarbol
    }
}

template <typename T>
int numhijos (typename Agen<T>::nodo n, const Agen<T>& A)
{
    int nhijos;
    typename Agen<T>::nodo hijo;

    nhijos = 0;
    hijo = A.hijoIzqdo(n);
    while(hijo != Agen<T>::NODO_NULO)
    {
        nhijos++;
        hijo = A.hermDrcho(hijo);
    }
    return nhijos;

}

int Maximo(int n1, int n2)
{
    return (n1>n2)? n1 : n2;
}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

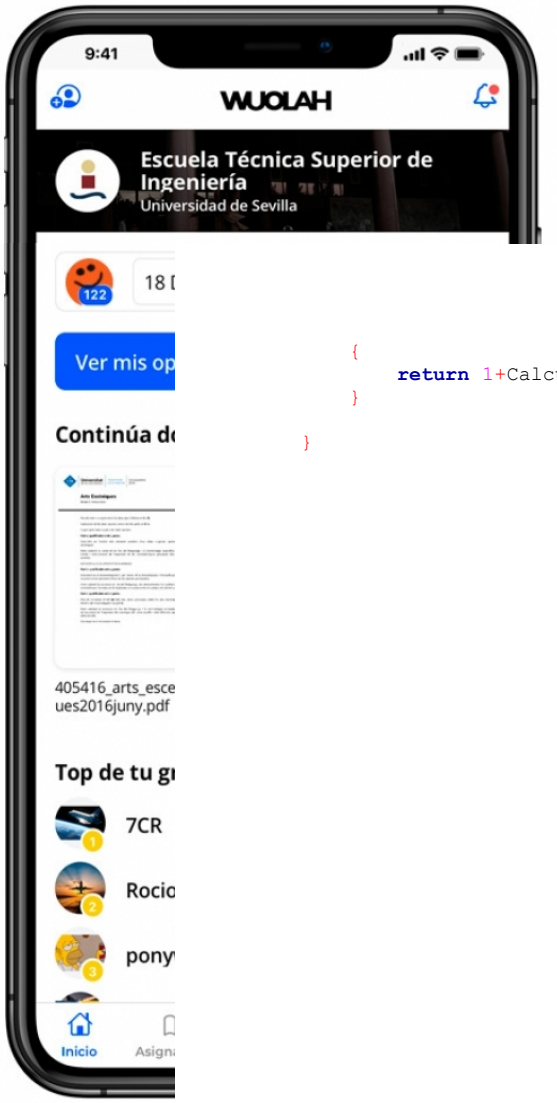
Descarga la app de Wuolah desde tu store favorita

```
/*
Ejercicio 2 Práctica 3

QuesoViejo_


Ejemplo de contenido del fichero NUMS.dat:

#
f g i # p # h j # # a r # # # n o # k # # b d e # c # l # # # #

*/

#include <iostream>
#include "AgenDin.h"
#include "agen_E-S.h"


template <typename T>
int CalcularProfundidad_iter(typename Agen<T>::nodo n, const Agen<T>& A);

template <typename T>
int CalcularProfundidad_rec(typename Agen<T>::nodo n, const Agen<T>& A);

using namespace std;
int main(){

Agen<char> A;


    cout << "\n*** Lectura de árbol A desde fichero ***\n";
    ifstream fe("NUMS.dat"); // Abrir fichero de entrada.
    rellenarAgen(fe, A); // Desde fichero.
    fe.close();

    cout << "\n*** Mostrar árbol A ***\n";
    imprimirAgen(A); // En std::cout

    int resultado;
    resultado = CalcularProfundidad_iter(A.hijoIzqdo(A.hijoIzqdo(A.raiz())),A);


    cout<<"\n[ITER] El resultado es: "<<resultado<<endl;


    resultado = CalcularProfundidad_rec(A.hijoIzqdo(A.hijoIzqdo(A.raiz())),A);


    cout<<"\n[REC] El resultado es: "<<resultado<<endl;


}

template <typename T>
int CalcularProfundidad_iter(typename Agen<T>::nodo n, const Agen<T>& A)
{
    int res = -1;    //Prof NODO_NULO=-1 ; Prof raiz=0 ; ...

    while(n != Agen<T>::NODO_NULO){
        ++res;
        n = A.padre(n);
    }

    return res;

}

template <typename T>
int CalcularProfundidad_rec(typename Agen<T>::nodo n, const Agen<T>& A)
{
    if(n == Agen<T>::NODO_NULO)
    {
        return -1;
    }
    else
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play

```
        {
            return 1+CalcularProfundidad_rec(A.padre(n),A);
        }

    }
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

```
/*
Ejercicio 3 Práctica 3

QuesoViejo_


Ejemplo de contenido del fichero NUMS.dat:

#
f g i # p # h j # # a r # # # n o # k # # b d e # c # l # # # #

*/

#include <iostream>
#include "AgenDin.h"
#include "agen_E-S.h"
#include "ColaDin.h"


template <typename T>
int CalcDeseq(const Agen<T>& A);

template <typename T>
int CalcDeseq_interna(typename Agen<T>::nodo n, const Agen<T>& A);

template <typename T>
int Altura_rec(typename Agen<T>::nodo n, const Agen<T>& A);


using namespace std;

int main(){


    Agen<char> A;

    cout << "\n*** Lectura de árbol A desde fichero ***\n";
    ifstream fe("NUMS.dat"); // Abrir fichero de entrada.
    rellenarAgen(fe, A); // Desde fichero.
    fe.close();

    cout << "\n*** Mostrar árbol A ***\n";
    imprimirAgen(A); // En std::cout

    int resultado = CalcDeseq(A);
    cout<<"El reslutado es: "<<resultado<<endl;
}




template <typename T>
int CalcDeseq(const Agen<T>& A){
    return CalcDeseq_interna(A.raiz(),A);
}
template <typename T>
int CalcDeseq_interna(typename Agen<T>::nodo n, const Agen<T>& A)
{
    Cola<typename Agen<T>::nodo> C1,C2; // cola de nodos de un árbol

    bool primeraIter = true,procesarRaiz = true;      //Está a true al procesar el primer
nodo de cada nivel
    int DESEQ_max= 0,DESEQ_nivel,ALT_max, ALT_min,ALT_nodo;

    typename Agen<T>::nodo h;

    //El deseq del árbol vacío y del de 1 nodo es 0. A partir de ahí, si es otro se
modificará abajo

    //La primera llamada será con el nodo raiz.

    //No se introducen los nodos nulos
    if(n != Agen<T>::NODO_NULO){     //Solo para el caso raiz = NODO_NULO
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Descarga la app de Wuolah desde tu store favorita

```
        C1.push(n);

        do{

        primeraIter = true;
        while(!C1.vacia()){
            n = C1.frente();
            C1.pop();

            h = A.hijoIzqdo(n);

            while(h != Agen<T>::NODO_NULO){      //Meto los hijos del nodo que se está
procesando en C2
                C2.push(h);
                h = A.hermDrcho(h);
            }

            if(!procesarRaiz){   //La raiz no es hoja de ningún nivel superior
                if(primeraIter){
                    ALT_max = ALT_min = ALT_nodo = Altura_rec(n,A);
                    primeraIter = false;
                }else{
                    ALT_nodo = Altura_rec(n,A);
                    //cout<<"NODO: "<<A.elemento(n)<<" Altura: "<<ALT_nodo<<endl;
                    if(ALT_nodo > ALT_max){
                        ALT_max = ALT_nodo;
                    }else if(ALT_nodo < ALT_min){
                        ALT_min = ALT_nodo;
                    }
                }
            }

        }
        //Se han acabado de procesar los nodos de cierto nivel. Pasar de la C2 a C1 y
actualizar los valores

        if(!procesarRaiz){
            DESEQ_nivel = ALT_max - ALT_min;

            if(DESEQ_max < DESEQ_nivel) DESEQ_max = DESEQ_nivel;     //DESEQ_max es al
principio 0
        }
        procesarRaiz = false;    //La raiz no es rama de ningún nivel superior

        while(!C2.vacia()){
            C1.push(C2.frente());
            C2.pop();
        }
        //cout<<"Acutalmente nivel: "<<DESEQ_nivel<<endl;

        }while(!C1.vacia());     //Si después de pasar de C2 a C1 sigue vacía, es que se
ha terminado de procesar el árbol
    }
    return DESEQ_max;    //Si es arbol vacío o solo raiz, devolverá el valor inicial que es 0

}


template <typename T>
int Altura_rec(typename Agen<T>::nodo n, const Agen<T>& A)
{
    if(n == Agen<T>::NODO_NULO){
        return -1;
    }
    else
    {
        int maximo = -1, valor;

        typename Agen<T>::nodo h = A.hijoIzqdo(n);

        while(h != Agen<T>::NODO_NULO){
            valor = Altura_rec(h,A);   //Valor tiene las alturas de los nodos hijos
            if(valor > maximo){
                maximo = valor;
            }
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

```
            h = A.hermDrcho(h);
        }

        return 1+maximo;     //Devuelve la altura del nodo n

    }
}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play

```
/*
Ejercicio 4 Práctica 3

QuesoViejo

Ejemplo de contenido del fichero NUMS.dat:

#
f g i # p # h j # # a r # # # n o # k # # b d e # c # l # # # #

*/

#include <iostream>
#include "AgenDin.h"
#include "agen_E-S.h"


template <typename T>
void Poda(const T& x, Agen<T>& A);

template <typename T>
void EliminarDescendencia(typename Agen<T>::nodo n,Agen<T>& A);

template <typename T>
typename Agen<T>::nodo Buscar(const T& x, const Agen<T>& A);

template <typename T>
typename Agen<T>::nodo Buscar_rec(typename Agen<T>:: nodo n, const T& x, const Agen<T>& A);


using namespace std;

int main(){


    Agen<char> A;

    cout << "\n*** Lectura de árbol A desde fichero ***\n";
    ifstream fe("NUMS.dat"); // Abrir fichero de entrada.
    rellenarAgen(fe, A); // Desde fichero.
    fe.close();
    cout<<"Hasta aqui"<<endl;
    cout << "\n*** Mostrar árbol A ***\n";
    imprimirAgen(A); // En std::cout

    Poda('n',A);

    cout << "\n*** Mostrar árbol A eliminando elemento 'n' ***\n";
    imprimirAgen(A); // En std::cout


}

template <typename T>
void Poda(const T& x, Agen<T>& A)
{
    typename Agen<T>::nodo n = Buscar(x,A);
    typename Agen<T>::nodo p, h;

    //Suponemos que siempre se encuentra el elemento x.

    EliminarDescendencia(n,A);
    if(n == A.raiz()){
        A.eliminarRaiz();
    }else{
        p = A.padre(n);
        h = A.hijoIzqdo(p);

        if(h == n){
            A.eliminarHijoIzqdo(n);
        }else{
            while(A.hermDrcho(h) != n){
                h = A.hermDrcho(n);
            }
```

WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

```
            A.eliminarHermDrcho(h);
        }
    }
    //Para eliminar el nodo que contiene a x

}

template <typename T>
void EliminarDescendencia(typename Agen<T>::nodo n,Agen<T>& A)
{
    if(n != Agen<T>::NODO_NULO){
        typename Agen<T>::nodo h = A.hijoIzqdo(n);

        while(h != Agen<T>::NODO_NULO){
            EliminarDescendencia(h,A);
            h = A.hermDrcho(h);
            A.eliminarHijoIzqdo(n); //Elimino el h antiguo y ahora el nuevo h es el hijo
Izqdo de n
        }
    }

}

template <typename T>
typename Agen<T>::nodo Buscar(const T& x, const Agen<T>& A)
{
    return Buscar_rec(A.raiz(),x,A);
}


template <typename T>
typename Agen<T>::nodo Buscar_rec(typename Agen<T>:: nodo n, const T& x, const Agen<T>& A)
{

    if(n == Agen<T>::NODO_NULO){   //Ojo, no se puede hacer A.elemento(NODO_NULO)
        return n;
    }
    else if(A.elemento(n)  == x){
        return n;
    }
    else
    {
        typename Agen<T>::nodo h = A.hijoIzqdo(n);
        typename Agen<T>::nodo maximo,resultadoActual;

        maximo = Buscar_rec(h,x,A);

        if(h != Agen<T>::NODO_NULO){
            h = A.hermDrcho(h); //Por el assert
        }

        while(h != Agen<T>::NODO_NULO && maximo == Agen<T>::NODO_NULO){

            resultadoActual = Buscar_rec(h,x,A);
            if(resultadoActual != Agen<T>::NODO_NULO){
                maximo = resultadoActual;
            }
            h = A.hermDrcho(h);
        }

        return maximo;   //Será NODO_NULO si no lo han encontrado o el nodo en cuestión si
lo han encontrado

    }
}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

```
/*
Ejercicio extra de árboles generales

QuesoViejo_


Ejemplo de contenido del fichero Palindromo.dat:

#
a b d # d # # c # b d # d # # #

*/

#include <iostream>
#include "AgenDin.h"
#include "agen_E-S.h"
#include "ColaDin.h"
#include "ListaDin.h"
#include <iostream>


template <typename T>
bool Simetrico(const Agen<T>& A);


using namespace std;



int main()
{

    Agen<char> A;

    cout << "\n*** Lectura de árbol A desde fichero ***\n";
    ifstream fe("Palindromo.dat"); // Abrir fichero de entrada.
    rellenarAgen(fe, A); // Desde fichero.
    fe.close();

    cout << "\n*** Mostrar árbol A ***\n";
    imprimirAgen(A); // En std::cout

    cout<<"\n\nResultado: "<<endl;
    if(Simetrico(A)){
        cout<<"Es simetrico"<<endl;
    }else{
        cout<<"NO es simetrico"<<endl;
    }
}

template <typename T>
bool Simetrico(const Agen<T>& A)
{
    Cola<typename Agen<T>::nodo> C1,C2;
    Lista<T> valores;
    int numvalores;
    C1.push(A.raiz());
    typename Agen<T>::nodo actual;
    bool iguales = true;
    do{
        numvalores = 0;
        while(!C1.vacia()){
            actual = C1.frente();
            C1.pop();

            typename Agen<T>::nodo hijo = A.hijoIzqdo(actual);

            while(hijo != Agen<T>::NODO_NULO){
                valores.insertar(A.elemento(hijo),valores.fin());
                numvalores++;
                C2.push(hijo);
                hijo = A.hermDrcho(hijo);
            }
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Descarga la app de Wuolah desde tu store favorita

```
        }

        //Aquí comprobar si es palíndromo
        if(numvalores != 0){
            typename Lista<T>::posicion p1 = valores.primera();
            typename Lista<T>::posicion p2 = valores.anterior(valores.fin());
            iguales = true;
            while(numvalores > 0 && iguales){
                if(valores.elemento(p1) == valores.elemento(p2)){
                    numvalores-=2;
                    //Para que solo calcule las posiciones siguientes si va a haber otra
iteración
                    if(numvalores > 0){
                        p1 = valores.siguiente(p1);
                        p2 = valores.anterior(p2);
                    }
                }else{
                    iguales = false;
                }
            }
        }
        C1 = C2;
        //Paso de C2 a C1 y C2 la dejo vacía
        while(!C2.vacia())C2.pop();

    }while(!C2.vacia() && iguales);

    return iguales;

}
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play

Estructuras de Datos no Lineales

Práctica 4

Problemas de árboles binarios de búsqueda

TRABAJO PREVIO

Antes de asistir a la sesión de prácticas es obligatorio:

1. **Implementar y probar** el TAD *Abb* representado mediante una estructura dinámica recursiva.
2. Imprimir copia de este enunciado.
3. Lectura profunda del mismo.
4. Reflexión sobre el contenido de la práctica y generación de la lista de dudas asociada a dicha práctica y a los problemas que la componen.
5. **Esbozo serio de solución** de los problemas en papel (al menos de los que se hayan entendido).

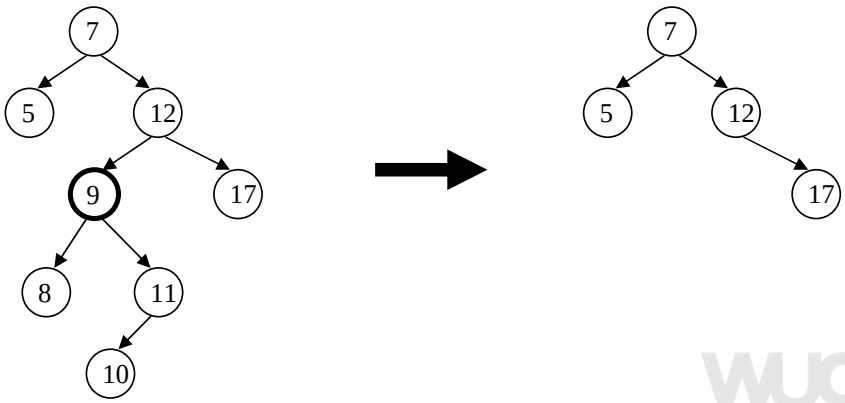
PASOS A SEGUIR

1. Escribir módulos que contengan las implementaciones de los subprogramas demandados en cada problema.
2. Para cada uno de los problemas escribir un programa de prueba, independiente de la representación del TAD elegida, donde se realicen las llamadas a los subprogramas del paso anterior, comprobando el resultado de salida para una batería suficientemente amplia de casos de prueba.

PROBLEMAS

1. Implementa una nueva operación del TAD *Abb* que tomando un elemento del mismo elimine al completo el subárbol que cuelga de él.

Ejemplo: Para el árbol binario de búsqueda de la figura se muestra la transformación si la entrada fuera el valor 9.



Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.
WUOLAH

2. Añade al TAD *Abb* un operador de conversión para obtener un árbol binario a partir de un ABB, *template <typename T> Abb<T>::operator Abin<T>() const;*. Es necesario declararlo como amigo de la clase *Abin*. Este operador nos permitirá obtener una copia de un ABB y tratarlo como un árbol binario, por ejemplo para realizar un recorrido del mismo.

3. Un árbol binario de búsqueda se puede equilibrar realizando el recorrido en inorden del árbol para obtener el listado ordenado de sus elementos y a continuación, repartir equitativamente los elementos a izquierda y derecha colocando la mediana en la raíz y construyendo recursivamente los subárboles izquierdo y derecho de cada nodo. Implementa este algoritmo para equilibrar un ABB.

4. Dados dos conjuntos representados mediante árboles binarios de búsqueda, implementa la operación *unión* de dos conjuntos que devuelva como resultado otro conjunto que sea la unión de ambos, representado por un ABB equilibrado.

5. Dados dos conjuntos representados mediante árboles binarios de búsqueda, implementa la operación *intersección* de dos conjuntos, que devuelva como resultado otro conjunto que sea la intersección de ambos. El resultado debe quedar en un árbol equilibrado.

6. Implementa el operador ♦ para conjuntos definido como $A \blacklozenge B = (A \cup B) - (A \cap B)$. La implementación del operador ♦ debe realizarse utilizando obligatoriamente la operación $\in$, que nos indica si un elemento dado pertenece o no a un conjunto. La representación del tipo *Conjunto* debe ser tal que la operación de pertenencia esté en el caso promedio en *O(log n)*.

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

```
/* "Abb.h"

Implementación del Árbol Binario de Búsqueda

*/

#ifndef ABB_H
#define ABB_H
#include <cassert>
#include "AbinDin.h"
template <typename T> class Abb {
public:
    Abb(); // constructor
    Abb(const Abb& A); // ctor. de copia
    Abb& operator =(const Abb& A); // asig. árboles

    const Abb& buscar(const T& e) const;

    void insertar(const T& e);
    void eliminar(const T& e);

    bool vacio() const;
    const T& elemento() const;

    const Abb& izqdo() const;
    const Abb& drcho() const;

    void eliminarSubAbb(const T& e);

    operator Abin<T>()const;
    void Introducir_rec(typename Abin<T>::nodo n, const Abb<T>& Bb, Abin<T>& A)const;

    ~Abb();

private:
    struct arbol {
        T elto;
        Abb izq, der;
        arbol(const T& e): elto(e) {}
    };

arbol* r; // raíz del árbol

T borrarMin();
void copiar(const Abb& A);




};

template <typename T>
inline Abb<T>::Abb() : r(0) {}

template <typename T>
inline bool Abb<T>::vacio() const
{
    return r == 0;
}
template <typename T>
const Abb<T>& Abb<T>::buscar(const T& e) const
{
    if (r == 0) // árbol vacío, e no encontrado
    return *this;

    else if (e < r->elto) // buscar en subárbol izqdo.
    return r->izq.buscar(e);

    else if (r->elto < e) // buscar en subárbol drcho.
    return r->der.buscar(e);

    else // encontrado e en la raíz
    return *this;
}
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

```
template <typename T>
void Abb<T>::insertar(const T& e)
{
    if (r == 0) // árbol vacío
    r = new arbol(e);

    else if (e < r->elto) // insertar en el subárbol izqdo.
    r->izq.insertar(e);

    else if (r->elto < e) // insertar en el subárbol drcho.
    r->der.insertar(e);
}

template <typename T>
void Abb<T>::eliminar(const T& e)
{
    if (r != 0) { // árbol no vacío
        if (e < r->elto) // quitar e del subárbol izqdo.
        r->izq.eliminar(e);
        else if (r->elto < e) // quitar e del subárbol drcho.
        r->der.eliminar(e);

        else // quitar e de la raíz
        if (r->izq.r == 0 && r->der.r == 0) { // 1. Raíz es hoja
            delete(r);
            r = 0; // el árbol queda vacío
        }
        else if (r->der.r == 0) { // 2. Raíz sólo tiene hijo izqdo.
            arbol* a = r->izq.r;
            r->izq.r = 0; // impide destruir el subárbol izqdo.
            delete(r);
            r = a;
        }
        else if (r->izq.r == 0) { // 3. Raíz sólo tiene hijo drcho.
            arbol* a = r->der.r;
            r->der.r = 0; // impide destruir el subárbol drcho.
            delete(r);
            r = a;
        }
        else // 4. Raíz tiene dos hijos
        // Eliminar el mínimo del subárbol derecho y sustituir
        // el elemento de la raíz por éste.
        r->elto = r->der.borrarMin();
    }
}

// Método privado
template <typename T>
T Abb<T>::borrarMin()
// Elimina el nodo que almacena el menor elemento
// del árbol. Devuelve el elemento del nodo eliminado.
{
    if (r->izq.r == 0) { // subárbol izquierdo vacío
        T e = r->elto;
        arbol* hd = r->der.r;
        r->der.r = 0; // impide destruir el subárbol drcho.
        delete(r);
        r = hd; // sustituir r por el subárbol drcho.
        return e;
    }
    else
        return r->izq.borrarMin();
}

template <typename T>
inline const T& Abb<T>::elemento() const
{
    assert(r != 0);
    return r->elto;
}
template <typename T>
inline const Abb<T>& Abb<T>::izqdo() const
{
    assert(r != 0);
    return r->izq;
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Descarga la app de Wuolah desde tu store favorita

Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play

```
}
template <typename T>
inline const Abb<T>& Abb<T>::drcho() const
{
    assert(r != 0);
    return r->der;
}

template <typename T>
inline Abb<T>::Abb(const Abb<T>& A): r(0)
{
    copiar(A);
}
template <typename T>
Abb<T>& Abb<T>::operator =(const Abb<T>& A)
{
    if (this != &A) { // evitar autoasignación
        this->~Abb(); // vaciar el árbol
        copiar(A);
    }
    return *this;
}

template <typename T>
Abb<T>::~Abb()
{
    if (r != 0) { // árbol no vacío
        delete r; // llama a r->izq.~Abb() y a r->der.~Abb()
        r = 0; // el árbol queda vacío
    }
}
// Método privado
template <typename T>
void Abb<T>::copiar(const Abb<T>& A)
// Copia el árbol a en *this
{
    if (A.r != 0) { // árbol no vacío
        r = new arbol(A.r->elto); // copiar raíz
        r->izq.copiar(A.r->izq); // copiar subárbol izqdo.
        r->der.copiar(A.r->der); // copiar subárbol drcho.
    }
}


//Ejercicio 1
template  <typename T>
void Abb<T>::eliminarSubAbb(const T& e){

    const Abb<int>& E = buscar(e);
    E.~Abb();
}

//Ejercicio2
template <typename T>
Abb<T>::operator Abin<T>()const{

    Abin<T> A;

    A.insertarRaizB(r->elto);

    //Esta función me asegura introducir los elementos de la raiz hacia abajo
    Introducir_rec(A.raizB(),*this,A); //Introduce los elementos del subarbol izq de r



    return A;
}

template <typename T>
void Abb<T>::Introducir_rec(typename Abin<T>::nodo n,const Abb<T>& Bb, Abin<T>& A)const{

    Abb<T> BbIzq;// = Bb.r->izq();
    BbIzq = Bb.izqdo();
    Abb<T> BbDer;// = Bb.r->der();
    BbDer = Bb.drcho();
```

WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

```
    if(!BbIzq.vacio()){
        A.insertarHijoIzqdoB(n,BbIzq.elemento());
        //Esta función me asegura introducir los elementos desde Hizqdo(n) hacia abajo
        Introducir_rec(A.hijoIzqdoB(n),BbIzq,A);
    }

    if(!BbDer.vacio()){
        A.insertarHijoDrchoB(n,BbDer.elemento());
        //Esta función me asegura introducir los elementos desde HDrcho(n) hacia abajo
        Introducir_rec(A.hijoDrchoB(n),BbDer,A);
    }


}

#endif // ABB_H
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

```
/* "Abb_E-S.h"

Funciones de E/S del Árbol Binario de Búsqueda

*/

#include <iostream>
#include "ABB.h"

using namespace std;

template <typename T>
void LeerAbb(Abb<T>& A);

template <typename T>
void MostrarAbb(const Abb<T>& A);

template <typename T>
void LeerAbb(Abb<T>& A){

    int elto;
    cout<<"Lecutra del arbol ABB (fin: '-1')"<<endl;

    do{
        cout<<"Introduce el elemento a insertar: ";
        cin >> elto;
        if(elto != -1) A.insertar(elto);
    }while(elto != -1);


}

template <typename T>
void MostrarAbb(const Abb<T>& A)
{
    cout<<A.elemento()<<endl;
    if(!A.izqdo().vacio())
    {
        cout<<"Hijo izq de "<<A.elemento()<<" : ";
        MostrarAbb(A.izqdo());
    }
    if(!A.drcho().vacio())
    {
        cout<<"Hijo der de "<<A.elemento()<<" : ";
        MostrarAbb(A.drcho());
    }
}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

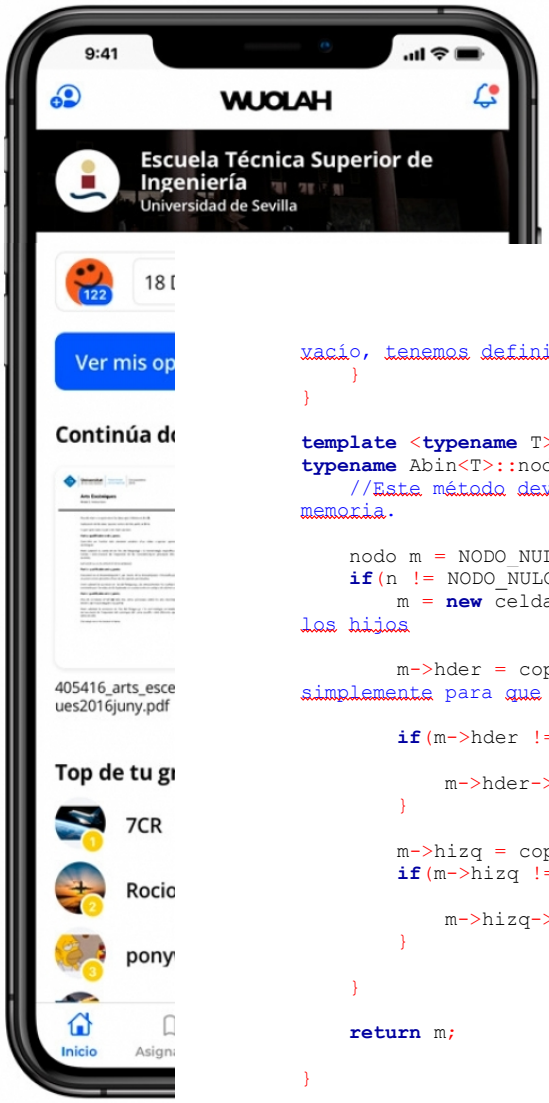
Descarga la app de Wuolah desde tu store favorita

```
/* "AbinDin.h"

Implementación dinámica del Árbol Binario

*/

#ifndef AbinDin
#define AbinDin

#include <cassert>

template <typename T>
class Abin{
    struct celda;
public:
    typedef celda* nodo;

    static const nodo NODO_NULO;    //Se declara  la constante NODO_NULO que después se
inicializará

    Abin(); //Ctor vacío
    Abin(const Abin<T>& a);
    Abin<T>& operator=(const Abin<T>& a);


    void insertarRaizB(const T& e);
    void insertarHijoIzqdoB(nodo n, const T& e);
    void insertarHijoDrchoB(nodo n, const T& e);
    void eliminarHijoIzqdoB(nodo n);
    void eliminarHijoDrchoB(nodo n);
    void eliminarRaizB();


    bool arbolVacioB()const ;
    const T& elemento(nodo n)const;
    T& elemento(nodo n);
    nodo raizB() const;
    nodo padreB(nodo n)const;
    nodo hijoIzqdoB(nodo n)const;
    nodo hijoDrchoB(nodo n)const;



    ~Abin();


private:
    struct celda{
        T elto;
        nodo padre,hizq,hder;
        celda(const T& e, nodo p = NODO_NULO): elto(e), padre(p), hizq(NODO_NULO),
hder(NODO_NULO) {}
    };
    nodo r; //A partir de él accedemos al árbol

    void destruirNodos(nodo& n);    //Se carga el nodo n y todos sus descendientes de
manera recursiva
    nodo copiar(nodo n);        //Copia el nodo n y todos sus descendientes de manera
recursiva

};


template <typename T>
const typename Abin<T>::nodo Abin<T>::NODO_NULO(0);


template <typename T>
void Abin<T>::destruirNodos(Abin<T>::nodo& n){

    if(n != NODO_NULO){
        destruirNodos(n->hizq);
        destruirNodos(n->hder);
        delete n;
        n = NODO_NULO;  //Delete lo deja en estado indefinido. Para identificar qeu está
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play

```
vacío, tenemos definido NODO_NULO
    }
}

template <typename T>
typename Abin<T>::nodo Abin<T>::copiar(Abin<T>::nodo n){
    //Este método devuelve una copia del nodo n, es decir, se copia en otra posición de
memoria.

    nodo m = NODO_NULO;
    if(n != NODO_NULO){
        m = new celda(n->elto); //m->padre por defecto inicializado a nulo, al igual que
los hijos

        m->hder = copiar(n->hder);  //Esto se hace fuera en vez de dentro del if
simplemente para que la llamada recursiva pare en
                                    //el caso base definido.
        if(m->hder != NODO_NULO){

            m->hder->padre = m;
        }

        m->hizq = copiar(n->hizq);
        if(m->hizq != NODO_NULO){

            m->hizq->padre = m;
        }

    }

    return m;

}

template <typename T>
inline Abin<T>::Abin() : r(NODO_NULO){}

template <typename T>
Abin<T>::Abin(const Abin<T>& a){
    r = copiar(a.r);
}

template <typename T>
Abin<T>& Abin<T>::operator=(const Abin<T>& a){

    if(this != &a){ //Para evitar autoasignación
        this->~Abin();  //Se destruye el árbol actual
        r = copiar(a.r);
    }

    return *this;
}

template <typename T>
inline void Abin<T>::insertarRaizB(const T& e){
    assert (r == NODO_NULO);
    r = new celda(e);
}

template <typename T>
inline void Abin<T>::insertarHijoIzqdoB(Abin<T>::nodo n, const T& e){
    assert(n != NODO_NULO);
    assert (n->hizq == NODO_NULO);

    n->hizq = new celda(e,n);
}

template <typename T>
inline void Abin<T>::insertarHijoDrchoB(Abin<T>::nodo n, const T& e){
    assert(n != NODO_NULO);
    assert (n->hder == NODO_NULO);

    n->hder = new celda(e,n);
}
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

```
template <typename T>
inline void Abin<T>::eliminarHijoIzqdoB(Abin<T>::nodo n){
    assert(n != NODO_NULO)
    assert( n->hizq != NODO_NULO);
    assert(n->hizq->hizq == NODO_NULO && n->hizq->hder == NODO_NULO);
    delete n->hizq;
    n->hizq = NODO_NULO;
}

template <typename T>
inline void Abin<T>::eliminarHijoDrchoB(Abin<T>::nodo n){
    assert(n != NODO_NULO)
    assert( n->hder != NODO_NULO);
    assert(n->hder->hizq == NODO_NULO && n->hder->hder == NODO_NULO);
    delete n->hder;
    n->hder = NODO_NULO;
}

template <typename T>
inline void Abin<T>::eliminarRaizB(){
    assert(r != NODO_NULO);
    assert(r->hizq == NODO_NULO && r->hder == NODO_NULO);
    delete r;
    r = NODO_NULO;
}

template <typename T>
inline bool Abin<T>::arbolVacioB()const{
    return (r == NODO_NULO);
}

template <typename T>
inline const T& Abin<T>::elemento(Abin<T>::nodo n)const{
    assert(n != NODO_NULO);
    return n->elto;
}

template <typename T>
inline T& Abin<T>::elemento(Abin<T>::nodo n){
    assert(n != NODO_NULO);
    return n->elto;
}

template <typename T>
typename Abin<T>::nodo Abin<T>::raizB()const{
    return r;    //Si no existe devuelve NODO_NULO que según la precondición
}

template <typename T>
typename Abin<T>::nodo Abin<T>::padreB(Abin<T>::nodo n)const{
    assert(n != NODO_NULO);
    return n->padre;    //Si no existe es NODO_NULO que era la precondición
}

template <typename T>
typename Abin<T>::nodo Abin<T>::hijoIzqdoB(Abin<T>::nodo n)const{
    assert(n != NODO_NULO);
    return n->hizq;
}

template <typename T>
typename Abin<T>::nodo Abin<T>::hijoDrchoB(Abin<T>::nodo n)const{
    assert(n != NODO_NULO);
    return n->hder;
}

template <typename T>
Abin<T>::~Abin(){
    destruirNodos(r);    //El trabajo lo hace la otra función
}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

```
#endif // AbinDin
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

```
/* "Abin_E-S.h"
Funciones de E/S del Árbol Binario

*/

#ifndef ABIN_E_S_H
#define ABIN_E_S_H

#include <cassert>
#include <iostream>
#include <fstream>

using std::istream;
using std::ostream;
using std::cin;
using std::cout;
using std::endl;

template <typename T>
void rellenarAbin(Abin<T>& A, const T& fin)
// Pre: A está vacío.
// Post: Rellena el árbol A con la estructura y elementos
//       leídos en preorden de la entrada estándar, usando
//       fin como elemento especial para introducir nodos nulos.
{
   T e;

   assert(A.arbolVacioB());
   cout << "Raiz (Fin = " << fin << "): "; cin >> e;
   if (e != fin) {
      A.insertarRaizB(e);
      rellenarDescendientes(A.raizB(), A, fin);
   }
}

template <typename T>
void rellenarDescendientes(typename Abin<T>::nodo r, Abin<T>& A, const T& fin)
// Pre: r es un nodo hoja de A.
// Post: Lee de la entrada estándar y en preorden los descendientes de r,
//       usando fin para introducir nodos nulos.
{
   T ehi, ehd;

   assert(A.hijoIzqdoB(r) == Abin<T>::NODO_NULO &&
          A.hijoDrchoB(r) == Abin<T>::NODO_NULO);
   cout << "Hijo izqdo. de " << A.elemento(r) <<
                " (Fin = " << fin << "): ";
   cin >> ehi;
   if (ehi != fin) {
      A.insertarHijoIzqdoB(r, ehi);
      rellenarDescendientes(A.hijoIzqdoB(r), A, fin);
   }
   cout << "Hijo drcho. de " << A.elemento(r) <<
                " (Fin = " << fin << "): ";
   cin >> ehd;
   if (ehd != fin) {
      A.insertarHijoDrchoB(r, ehd);
      rellenarDescendientes(A.hijoDrchoB(r), A, fin);
   }
}

template <typename T>
void rellenarAbin(istream& is, Abin<T>& A)
// Pre: A está vacío.
// Post: Extrae los nodos de A del flujo de entrada is,
//       que contendrá el elemento especial que denota
//       un nodo nulo seguido de los elementos en preorden,
//       incluyendo los correspondientes a nodos nulos.
{
   T e, fin;

   assert(A.arbolVacioB());
   if (is >> fin && is >> e && e != fin) {
      A.insertarRaizB(e);
      rellenarDescendientes(is, A.raizB(), A, fin);
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play

```
   }
}

template <typename T>
void rellenarDescendientes(istream& is, typename Abin<T>::nodo r, Abin<T>& A, const T& fin)
// Pre: r es un nodo hoja de A.
// Post: Lee del flujo de entrada is y en preorden los descendientes de r,
//       usando fin para los nodos nulos.
{
   T ehi, ehd;

   assert(A.hijoIzqdoB(r) == Abin<T>::NODO_NULO &&
          A.hijoDrchoB(r) == Abin<T>::NODO_NULO);
   if (is >> ehi && ehi != fin) {
      A.insertarHijoIzqdoB(r, ehi);
      rellenarDescendientes(is, A.hijoIzqdoB(r), A, fin);
   }
   if (is >> ehd && ehd != fin) {
      A.insertarHijoDrchoB(r, ehd);
      rellenarDescendientes(is, A.hijoDrchoB(r), A, fin);
   }
}

template <typename T>
void imprimirAbin (const Abin<T>& A)
// Post: Muestra los nodos de A en la salida estándar
{
   if (!A.arbolVacioB())
   {
      cout << "Raíz del árbol: "
               << A.elemento(A.raizB()) << endl;
      imprimirDescendientes(A.raizB(), A);
   }
   else
      cout << "Árbol vacío\n";
}

template <typename T>
void imprimirDescendientes (typename Abin<T>::nodo r, const Abin<T>& A)
// Post: Muestra los descendientes de r en la salida estándar.
{
   if (A.hijoIzqdoB(r) != Abin<T>::NODO_NULO)
   {
      cout << "Hijo izqdo de " << A.elemento(r) << ": "
               << A.elemento(A.hijoIzqdoB(r)) << endl;
      imprimirDescendientes(A.hijoIzqdoB(r), A);
   }
   if (A.hijoDrchoB(r) != Abin<T>::NODO_NULO)
   {
      cout << "Hijo derecho de " << A.elemento(r) << ": "
               << A.elemento(A.hijoDrchoB(r)) << endl;
      imprimirDescendientes(A.hijoDrchoB(r), A);
   }
}

template <typename T>
void imprimirAbin (ostream& os, const Abin<T>& A, const T& fin)
// Post: Inserta en el flujo de salida os los nodos de A en preorden,
//       precedidos del elemento especial usado para denotar un nodo nulo.
{
   if (!A.arbolVacioB())
   {
      os << fin << endl
         << A.elemento(A.raizB()) << ' ';
      imprimirDescendientes(os, A.raizB(), A, fin);
      os << endl;
   }
}

template <typename T>
void imprimirDescendientes (ostream& os, typename Abin<T>::nodo r, const Abin<T>& A,
const T& fin)
// Post: Inserta en el flujo de salida os y en preorden
//       los descendientes de r, usando fin como nodo nulo.
{
```

WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

```
    if (A.hijoIzqdoB(r) != Abin<T>::NODO_NULO)
    {
       os << A.elemento(A.hijoIzqdoB(r)) << ' ';
       imprimirDescendientes(os, A.hijoIzqdoB(r), A, fin);
    }
    else
       os << fin << ' ';
    if (A.hijoDrchoB(r) != Abin<T>::NODO_NULO)
    {
       os << A.elemento(A.hijoDrchoB(r)) << ' ';
       imprimirDescendientes(os, A.hijoDrchoB(r), A, fin);
    }
    else
       os << fin << ' ';
}

#endif   // ABIN_E_S_H
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

```
/*
Ejercicio 1 Práctica 4

QuesoViejo_

*/

#include "Abb.h"
#include "Abb_E-S.h"
#include <iostream>


using namespace std;
int main(){

    Abb<int> A;
    LeerAbb(A);
    MostrarAbb(A);

    int valor;
    cout<<"Introduce el valor a eliminar: ";
    cin >> valor;

    A.eliminarSubAbb(valor);
    cout<<"El ABB queda:" <<endl;
    MostrarAbb(A);

}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

```
/*
Ejercicio 2 Práctica 4

QuesoViejo_

*/

#include "Abb.h"
#include "Abb_E-S.h"
#include <iostream>
#include "abin_E-S.h"


using namespace std;
int main(){

    Abb<int> A;
    LeerAbb(A);
    MostrarAbb(A);

    cout<<"Conversión a ABIN"<<endl;
    Abin<int> Ab = A;
    imprimirAbin(Ab);
    cout<<"Valor por ejemplo del hizq de raiz: "<<Ab.elemento(Ab.hijoIzqdoB(Ab.raizB()));

}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play

```
/*
Ejercicio 3 Práctica 4

QuesoViejo

*/

#include "Abb.h"
#include "Abb_E-S.h"
#include <iostream>
#include <cstdio>  //Para el size_t

template <typename T>
void Equilibrar(Abb<T>& A);


using namespace std;
int main(){

    Abb<int> A;
    LeerAbb(A);
    cout<<"Arbol Introducido: "<<endl;
    MostrarAbb(A);

    Equilibrar(A);
    cout<<"\n\nArbol equilibrado: "<<endl;
    MostrarAbb(A);

}

template <typename T>
void Equilibrar(Abb<T>& A){

    size_t tam = ContarNodos(A);
    T vec[tam];

    int indice = 0;
    RellenarVector(vec,indice,A);

    //Hasta aquí correcto
    A.~Abb();
    InsertarElementos(0,tam-1,vec,A);

}

template <typename T>
size_t ContarNodos(const Abb<T>& A){


    if(A.vacio()){
        return 0;
    }else{
        return 1+ContarNodos(A.izqdo()) + ContarNodos(A.drcho());
    }
}

template <typename T>
void RellenarVector(T* vec,int& i,const Abb<T>& A){

    if(!A.vacio())
    {
        RellenarVector(vec, i, A.izqdo());
        //i se habrá modificado si se han metido elementos en el vector
        vec[i] = A.elemento();
        i++;
        RellenarVector(vec,i,A.drcho());
    }
}

template <typename T>
void InsertarElementos(int inf, int sup,T* vec, Abb<T>& A)
{
    if(sup == inf){
        A.insertar(vec[inf]);
    }else if(sup == inf+1){
```

WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

```
        A.insertar(vec[inf]);
        A.insertar(vec[sup]);
    }else{
        int centro = inf + (sup-inf)/2;
        A.insertar(vec[centro]);
        InsertarElementos(inf,centro-1,vec,A);
        InsertarElementos(centro+1,sup,vec,A);
    }
}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

```
/*
Ejercicio 4 Práctica 4

QuesoViejo_

*/

#include "Abb.h"
#include "Abb_E-S.h"
#include <iostream>



template <typename T>
Abb<T> UnionConjunto(const Abb<T>& A,Abb<T> B);


using namespace std;
int main(){

    cout<<"Abb A:"<<endl;
    Abb<int> A;
    LeerAbb(A);
    MostrarAbb(A);

    cout<<"Abb B:"<<endl;
    Abb<int> B;
    LeerAbb(B);
    MostrarAbb(B);

    cout<<"La unión queda: " <<endl;
    MostrarAbb(UnionConjunto(A,B));

}


template <typename T>
Abb<T> UnionConjunto(const Abb<T>& A, Abb<T> B){     //B es una copia del original

    Abb<T> C{A};

    while(!B.vacio()){
        if(A.buscar(B.elemento()).vacio()){ //Si en A no está el elemento de la raiz de B
            C.insertar(B.elemento());
        }
        B.eliminar(B.elemento());
    }
    Equilibrar(C);
    return C;
}

template <typename T>
void Equilibrar(Abb<T>& A){

    size_t tam = ContarNodos(A);
    T vec[tam];

    int indice = 0;
    RellenarVector(vec,indice,A);

    //Hasta aquí correcto
    A.~Abb();
    InsertarElementos(0,tam-1,vec,A);

}

template <typename T>
size_t ContarNodos(const Abb<T>& A){


    if(A.vacio()){
        return 0;
    }else{
        return 1+ContarNodos(A.izqdo()) + ContarNodos(A.drcho());
    }
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

```
}

template <typename T>
void RellenarVector(T* vec,int& i,const Abb<T>& A){

    if(!A.vacio())
    {
        RellenarVector(vec, i, A.izqdo());
        //i se habrá modificado si se han metido elementos en el vector
        vec[i] = A.elemento();
        i++;
        RellenarVector(vec,i,A.drcho());
    }
}

template <typename T>
void InsertarElementos(int inf, int sup,T* vec, Abb<T>& A)
{
    if(sup == inf){
        A.insertar(vec[inf]);
    }else if(sup == inf+1){
        A.insertar(vec[inf]);
        A.insertar(vec[sup]);
    }else{
        int centro = inf + (sup-inf)/2;
        A.insertar(vec[centro]);
        InsertarElementos(inf,centro-1,vec,A);
        InsertarElementos(centro+1,sup,vec,A);
    }
}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play

```
/*
Ejercicio 5 Práctica 4

QuesoViejo

*/

#include "Abb.h"
#include "Abb_E-S.h"
#include <iostream>



template <typename T>
Abb<T> InterseccionConjunto(const Abb<T>& A, Abb<T> B);


using namespace std;
int main(){

    cout<<"Abb A:"<<endl;
    Abb<int> A;
    LeerAbb(A);
    MostrarAbb(A);

    cout<<"Abb B:"<<endl;
    Abb<int> B;
    LeerAbb(B);
    MostrarAbb(B);

    cout<<"\n\nLa intersección queda: " <<endl;
    MostrarAbb(InterseccionConjunto(A,B));

}



template <typename T>
Abb<T> InterseccionConjunto(const Abb<T>& A, Abb<T> B){     //B es una copia del original

    Abb<T> C{};

    while(!B.vacio()){
        if(!A.buscar(B.elemento()).vacio()){ //Si en A no está el elemento de la raiz de B
            C.insertar(B.elemento());
        }
        B.eliminar(B.elemento());
    }
    Equilibrar(C);
    return C;
}

template <typename T>
void Equilibrar(Abb<T>& A){

    size_t tam = ContarNodos(A);
    T vec[tam];

    int indice = 0;
    RellenarVector(vec,indice,A);

    //Hasta aquí correcto
    A.~Abb();
    InsertarElementos(0,tam-1,vec,A);

}

template <typename T>
size_t ContarNodos(const Abb<T>& A){


    if(A.vacio()){
        return 0;
```

WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

```
    }else{
        return 1+ContarNodos(A.izqdo()) + ContarNodos(A.drcho());
    }
}

template <typename T>
void RellenarVector(T* vec,int& i,const Abb<T>& A){

    if(!A.vacio())
    {
        RellenarVector(vec, i, A.izqdo());
        //i se habrá modificado si se han metido elementos en el vector
        vec[i] = A.elemento();
        i++;
        RellenarVector(vec,i,A.drcho());
    }
}

template <typename T>
void InsertarElementos(int inf, int sup,T* vec, Abb<T>& A)
{
    if(sup == inf){
        A.insertar(vec[inf]);
    }else if(sup == inf+1){
        A.insertar(vec[inf]);
        A.insertar(vec[sup]);
    }else{
        int centro = inf + (sup-inf)/2;
        A.insertar(vec[centro]);
        InsertarElementos(inf,centro-1,vec,A);
        InsertarElementos(centro+1,sup,vec,A);
    }
}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

```
/*
Ejercicio 6 Práctica 4

QuesoViejo_

*/

#include "Abb.h"
#include "Abb_E-S.h"
#include <iostream>


/**
    Yo no he hecho TAD CONJUNTO, pero si lo hiciera, después de cada inserción y
eliminación tendría que equilibrar
    el árbol y de esa forma la operación pertenece siempre será de en el caso promedio
O(log n).

*/

template <typename T>
Abb<T> UnionConjunto(const Abb<T>& A,Abb<T> B);


template <typename T>
Abb<T> InterseccionConjunto(const Abb<T>& A, Abb<T> B);

template <typename T>
Abb<T> OperadorRombo(const Abb<T>& A, Abb<T> B);

using namespace std;
int main(){

    cout<<"Abb A:"<<endl;
    Abb<int> A;
    LeerAbb(A);
    MostrarAbb(A);

    cout<<"Abb B:"<<endl;
    Abb<int> B;
    LeerAbb(B);
    MostrarAbb(B);

    cout<<"La (Union - Interseccion) queda: " <<endl;
    MostrarAbb(OperadorRombo(A,B));

}



template <typename T>
Abb<T> OperadorRombo(const Abb<T>& A, Abb<T> B){     //B es una copia del original

    Abb<T> Union{UnionConjunto(A,B)};
    Abb<T> Interseccion{InterseccionConjunto(A,B)};
    Abb<T> Diferencia{Union};

    while(!Interseccion.vacio()){
        if(pertenece(Interseccion.elemento(),Union)){ //Si se encuentra el elemento en la
interseccion, se quita de la diferencia
            Diferencia.eliminar(Interseccion.elemento());
        }
        Interseccion.eliminar(Interseccion.elemento());
        Equilibrar(Interseccion); //Para que "pertenece" sea O(log n)
    }
    Equilibrar(Diferencia);
    return Diferencia;
}

template <typename T>
bool pertenece(const T& e, const Abb<T>& A){
    return !A.buscar(e).vacio();
}

template <typename T>
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

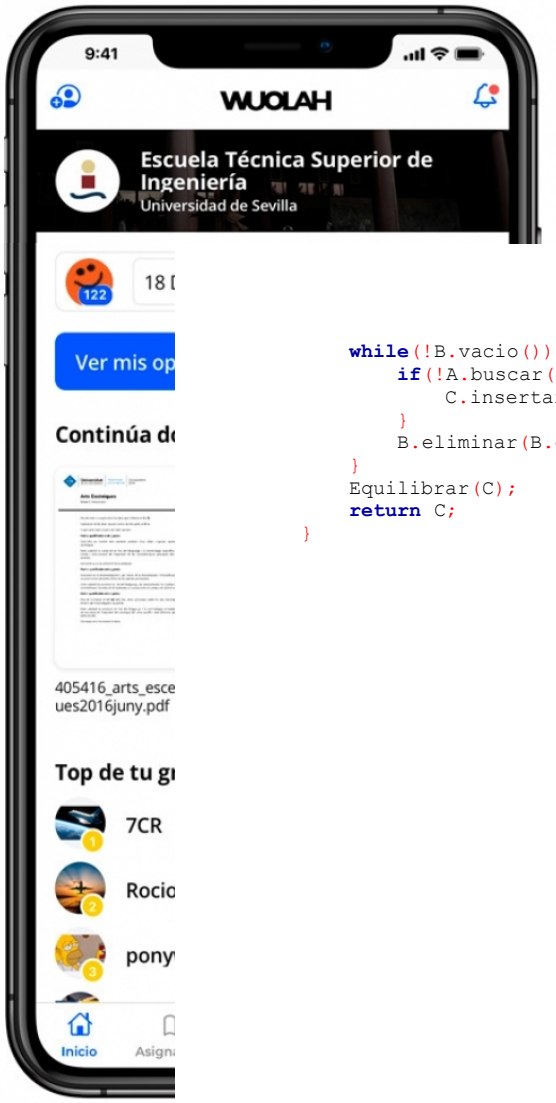
Descarga la app de Wuolah desde tu store favorita

```
Abb<T> UnionConjunto(const Abb<T>& A, Abb<T> B){     //B es una copia del original

    Abb<T> C{A};

    while(!B.vacio()){
        if(A.buscar(B.elemento()).vacio()){ //Si en A no está el elemento de la raiz de B
            C.insertar(B.elemento());
        }
        B.eliminar(B.elemento());
    }
    Equilibrar(C);
    return C;
}

template <typename T>
void Equilibrar(Abb<T>& A){
    if(!A.vacio()){
        size_t tam = ContarNodos(A);
        T vec[tam];

        int indice = 0;
        RellenarVector(vec,indice,A);

        //Hasta aquí correcto
        A.~Abb();
        InsertarElementos(0,tam-1,vec,A);
    }

}

template <typename T>
size_t ContarNodos(const Abb<T>& A){


    if(A.vacio()){
        return 0;
    }else{
        return 1+ContarNodos(A.izqdo()) + ContarNodos(A.drcho());
    }
}

template <typename T>
void RellenarVector(T* vec,int& i,const Abb<T>& A){

    if(!A.vacio())
    {
        RellenarVector(vec, i, A.izqdo());
        //i se habrá modificado si se han metido elementos en el vector
        vec[i] = A.elemento();
        i++;
        RellenarVector(vec,i,A.drcho());
    }
}

template <typename T>
void InsertarElementos(int inf, int sup,T* vec, Abb<T>& A)
{
    if(sup == inf){
        A.insertar(vec[inf]);
    }else if(sup == inf+1){
        A.insertar(vec[inf]);
        A.insertar(vec[sup]);
    }else{
        int centro = inf + (sup-inf)/2;
        A.insertar(vec[centro]);
        InsertarElementos(inf,centro-1,vec,A);
        InsertarElementos(centro+1,sup,vec,A);
    }
}

template <typename T>
Abb<T> InterseccionConjunto(const Abb<T>& A, Abb<T> B){     //B es una copia del original

    Abb<T> C{};
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com



Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play

```
        while(!B.vacio()){
            if(!A.buscar(B.elemento()).vacio()){ //Si en A no está el elemento de la raiz de B
                C.insertar(B.elemento());
            }
            B.eliminar(B.elemento());
        }
        Equilibrar(C);
        return C;
    }
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Estructuras de Datos no Lineales

Práctica 5

Problemas de árboles parcialmente ordenados y otros árboles

TRABAJO PREVIO

Antes de asistir a la sesión de prácticas es obligatorio:

1. **Implementar y probar** los TAD que sean necesarios para resolver los diferentes problemas.
2. Imprimir copia de este enunciado.
3. Lectura profunda del mismo.
4. Reflexión sobre el contenido de la práctica y generación de la lista de dudas asociada a dicha práctica y a los problemas que la componen.
5. **Esbozo serio de solución** de los problemas en papel (al menos de los que se hayan entendido).

PASOS A SEGUIR

1. Escribir módulos que contengan las implementaciones de los subprogramas demandados en cada problema.
2. Para cada uno de los problemas escribir un programa de prueba, independiente de la representación del TAD elegida, donde se realicen las llamadas a los subprogramas del paso anterior, comprobando el resultado de salida para una batería suficientemente amplia de casos de prueba.

PROBLEMAS

1. Dado un árbol binario de enteros donde el valor de cada nodo es menor que el de sus hijos, implementa un subprograma para eliminar un valor del mismo preservando la propiedad de orden establecida. Explica razonadamente la elección de la estructura de datos.

Nota: Se supone que en el árbol no hay elementos repetidos, y que el número de nodos del mismo no está acotado

2. Un montículo *min-max* tiene una estructura similar a la de un montículo ordinario (árbol parcialmente ordenado), pero la ordenación parcial consiste en que los elementos que se encuentran en un nivel par (0, 2, 4,...) son menores o iguales que sus elementos descendientes, mientras que los elementos que se encuentran en un nivel impar (1, 3,

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

5,...) son mayores o iguales que sus descendientes. Esto quiere decir que para cualquier elemento *e* de un nivel par se cumple $abuelo \leq e \leq padre$ y para cualquier elemento *e* de un nivel impar $padre \leq e \leq abuelo$.

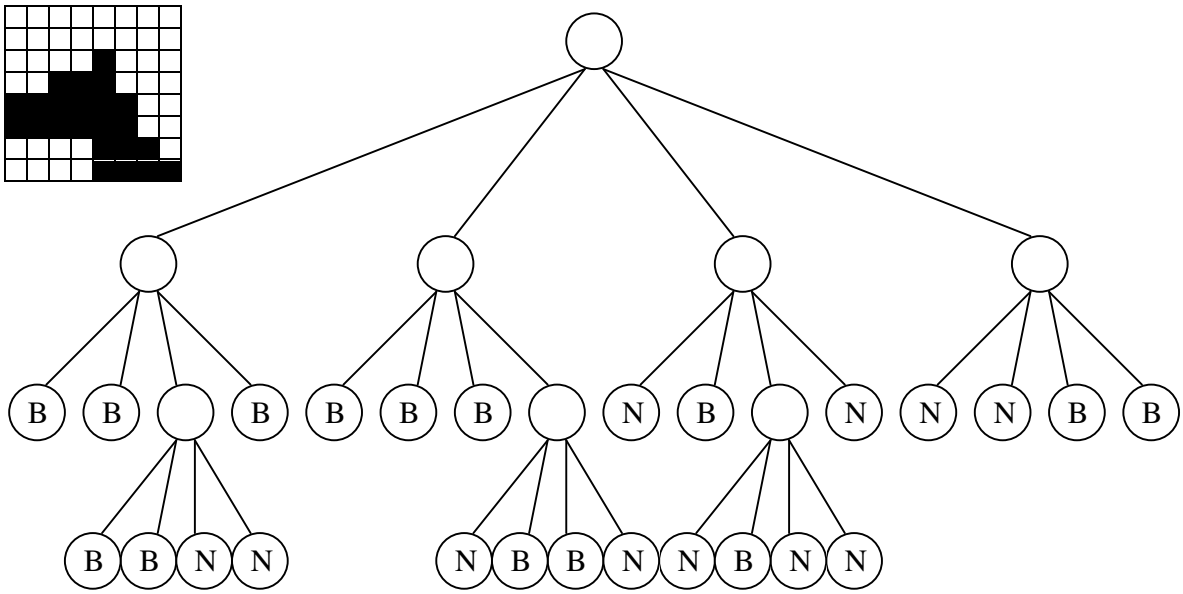
Implementa una operación de orden logarítmico para añadir un elemento a un montículo *min-max* almacenado en un vector de posiciones relativas.

3. Implementa una operación de orden logarítmico para eliminar el elemento máximo de un montículo *min-max* definido como en el problema anterior.

4. Un árbol es estrictamente ternario si todos sus nodos son hojas o tienen tres hijos. Escribe una función que, dado un árbol de grado arbitrario, nos indique si es o no estrictamente ternario.

5. Una forma de representar una figura plana en blanco y negro consiste en utilizar un árbol cuaternario en el que cada nodo o tiene exactamente cuatro hijos, o bien es una hoja. Un nodo hoja puede ser blanco o negro y un nodo interno no tiene color.

Una figura dibujada dentro de un cuadrado de lado 2^k se representa de la forma siguiente: Se divide el cuadrado en cuatro cuadrantes y cada uno se representa como un hijo del nodo raíz. Si un cuadrante está completamente negro corresponde a una hoja negra; si, por el contrario, el cuadrante está completamente blanco, éste corresponde a una hoja blanca; y si un cuadrante está parcialmente ocupado por negro y blanco, entonces corresponde a un nodo interno del árbol y este cuadrante se representa siguiendo el mismo método subdividiéndolo en otros cuatro cuadrantes. Como ejemplo se muestra una figura en blanco y negro y su árbol asociado, tomando los cuadrantes en el sentido de las agujas del reloj a partir del cuadrante superior izquierdo.



Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

Implementa una función que dado un árbol de esta clase, con $k+1$ niveles, devuelva la figura asociada, representada como una matriz cuadrada de tamaño 2^k en la que cada celda representa un punto blanco o negro.

Nota: Por simplificar el problema, se asume que en cada nodo del árbol se incluyen las coordenadas de la esquina superior izquierda y de la esquina inferior derecha del cuadrante que representa.

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play

```
/* "APO.h"

Implementación del Árbol Parcialmente Ordenado

*/

#ifndef APO_H
#define APO_H
#include <cassert>
#include <cstdlib>    //Para el size_t
#include <iostream>
template <typename T> class Apo {
public:
    explicit Apo(size_t maxNodos); // constructor
    void insertar(const T& e);
    void suprimir();
    const T& cima() const;
    bool vacio() const;
    Apo(const Apo<T>& A); // ctor. de copia
    Apo<T>& operator =(const Apo<T>& A); // asignación de apo
    ~Apo(); // destructor

    void Mostrar();
private:
    typedef int nodo; // índice del vector entre 0 y maxNodos-1

    T* nodos; // vector de nodos
    int maxNodos; // tamaño del vector

    nodo ultimo; // último nodo del árbol
    nodo padre(nodo i) const { return (i-1)/2; }
    nodo hIzq(nodo i) const { return 2*i+1; }
    nodo hDer(nodo i) const { return 2*i+2; }

    void flotar(nodo i);
    void hundir(nodo i);
    void Mostrar_descendencia(nodo i);
};

template <typename T>
Apo<T>::Apo(size_t maxNodos) : nodos(new T[maxNodos]), maxNodos(maxNodos), ultimo(-1){}
// Apo vacío

template <typename T>
inline const T& Apo<T>::cima() const{
    assert(ultimo > -1); // Apo no vacío
    return nodos[0];
}

template <typename T>
inline bool Apo<T>::vacio() const{
    return (ultimo == -1);
}


template <typename T>
inline void Apo<T>::insertar(const T& e){
    assert(ultimo < maxNodos-1); // Apo no lleno
    nodos[++ultimo] = e;
    if (ultimo > 0)
    flotar(ultimo); // Reordenar
}

template <typename T>
void Apo<T>::flotar(nodo i)
{
    T e = nodos[i];
    while (i > 0 && e < nodos[padre(i)]){
        nodos[i] = nodos[padre(i)];
        i = padre(i);
    }
    nodos[i] = e;
}

template <typename T>
```

WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

```
inline void Apo<T>::suprimir()
{
    assert(ultimo > -1); // Apo no vacío

    if (--ultimo > -1){ // Apo no queda vacío
        nodos[0] = nodos[ultimo+1];

        if (ultimo > 0) hundir(0); // Quedan dos o más elementos. =>  Reordenar
    }
}


//Este algoritmo va a intercambiar i con el menor de sus dos hijos
template <typename T>
void Apo<T>::hundir(nodo i){
    bool fin = false;
    T e = nodos[i];
    while (hIzq(i) <= ultimo && !fin){ // hundir e
        nodo hMin; // hijo menor del nodo i
        if (hIzq(i) < ultimo && nodos[hDer(i)] < nodos[hIzq(i)])
            hMin = hDer(i);
        else
            hMin = hIzq(i);

        if (nodos[hMin] < e) { // subir el hijo menor
            nodos[i] = nodos[hMin];
            i = hMin;
        }else fin = true; // e <= hijos

    }
    nodos[i] = e; // colocar e
}

template <typename T>
inline Apo<T>::~Apo()
{
    delete[] nodos;
}


template <typename T>
Apo<T>::Apo(const Apo<T>& A) : nodos(new T[A.maxNodos]), maxNodos(A.maxNodos),
ultimo(A.ultimo){
    // copiar el vector
    for (nodo n = 0; n <= ultimo; n++)
        nodos[n] = A.nodos[n];
}

template <typename T>
Apo<T>& Apo<T>::operator =(const Apo<T>& A)
{
    if (this != &A) { // evitar autoasignación
        // Destruir el vector y crear uno nuevo si es necesario
        if (maxNodos != A.maxNodos) {
            delete[] nodos;
            maxNodos = A.maxNodos;
            nodos = new T[maxNodos];
        }
        ultimo = A.ultimo;

        // Copiar el vector
        for (nodo n = 0; n <= ultimo; n++)
            nodos[n] = A.nodos[n];
    }
    return *this;
}


//Funciones de QuesoViejo_ para mostrarlo por pantalla
template <typename T>
void Apo<T>::Mostrar(){
    using namespace std;
    if(ultimo != -1){
        cout<<"Raiz: "<<nodos[0]<<endl;
        Mostrar_descendencia(0);
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

```
    }
}

template <typename T>
void Apo<T>::Mostrar_descendencia(Apo<T>::nodo i){
    //Si i ya no pertenece al arbol o alguno de sus hijos, termina él solito.
    using namespace std;

    if(hIzq(i) <= ultimo){
        cout<<"Hijo izqdo de "<<nodos[i]<<": "<<nodos[hIzq(i)]<<endl;
        Mostrar_descendencia(hIzq(i));
    }
    if(hDer(i) <= ultimo){
        cout<<"Hijo drcho de "<<nodos[i]<<": "<<nodos[hDer(i)]<<endl;
        Mostrar_descendencia(hDer(i));
    }
}
#endif // APO_H
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

```
/* "APO_E-S.h"

Funciones para Lectura del APO

QuesoViejo_

*/

#include "APO.h"    //Ya incluye iostream

void LeerAPO(Apo<int>& A);
void LeerAPOmM(ApomM<int>& A);

void LeerAPO(Apo<int>& A){
    using namespace std;
    int e;
    do{
        cout<<"Introduce elemento [FIN -1]: ";
        cin >> e;
        if(e != -1) A.insertar(e);
    }while(e != -1);
}

void LeerAPOmM(ApomM<int>& A){
    using namespace std;
    int e;
    do{
        cout<<"Introduce elemento [FIN -1]: ";
        cin >> e;
        if(e != -1) A.insertar(e);
    }while(e != -1);
}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play

```
/* "APOmM.h"

Implementación del APO min y Max

QuesoViejo_

*/


#ifndef ApomM_H
#define ApomM_H
#include <cassert>
#include <cstdlib>    //Para el size_t
#include <iostream>
#include <cmath>    //Para el log2
template <typename T> class ApomM {
public:
    explicit ApomM(size_t maxNodos); // constructor
    void insertar(const T& e);
    void suprimir();
    void suprimirMax();
    const T& cima() const;
    bool vacio() const;
    ApomM(const ApomM<T>& A); // ctor. de copia
    ApomM<T>& operator =(const ApomM<T>& A); // asignación de ApomM
    ~ApomM(); // destructor

    void Mostrar();
private:
    typedef int nodo; // índice del vector entre 0 y maxNodos-1

    T* nodos; // vector de nodos
    int maxNodos; // tamaño del vector

    nodo ultimo; // último nodo del árbol
    nodo padre(nodo i) const { return (i==0) ? -1 : (i-1)/2; }  //Devuelve -1 si i es la
raiz
    nodo hIzq(nodo i) const { return 2*i+1; }
    nodo hDer(nodo i) const { return 2*i+2; }

    void flotar(nodo i);
    void hundir(nodo i);
    void Mostrar_descendencia(nodo i);
    void Intercambio(T& a, T& b);

    nodo nodoMinimo(nodo n1, nodo n2, nodo n3);
    nodo nodoMaximo(nodo n1, nodo n2, nodo n3);


};

template <typename T>
ApomM<T>::ApomM(size_t maxNodos) : nodos(new T[maxNodos]), maxNodos(maxNodos),
ultimo(-1){}  // ApomM vacío

template <typename T>
inline const T& ApomM<T>::cima() const{
    assert(ultimo > -1); // ApomM no vacío
    return nodos[0];
}

template <typename T>
inline bool ApomM<T>::vacio() const{
    return (ultimo == -1);
}


template <typename T>
inline void ApomM<T>::insertar(const T& e){
    /*  Lo insertaré en la posición del último y lo iré flotando hasta que quede en la
posición correcta.
        Lo único que realmente tengo que cambiar es la función flotar, ya que aquí solo se
inserta en el último
    */
```

WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

```
    assert(ultimo < maxNodos-1); // ApomM no lleno
    nodos[++ultimo] = e;
    if (ultimo > 0){
    flotar(ultimo); // Reordenar
    }
}

template <typename T>
void ApomM<T>::flotar(nodo i)
{
    //i es el nodo recién insertado
    int profundidad = log2(i+1);

    T e = nodos[i];
    T aux;

    nodo abuelo;
    //i tiene padre, ya que si fuera raiz no se llamaría a flotar
    if(profundidad%2 == 0){ //Nivel par
            if(nodos[padre(i)] < nodos[i]){ //Que haya que reordenar respecto a los
niveles impares
                Intercambio(nodos[i], nodos[padre(i)]);
                i = padre(i);
                profundidad--;
                //Ahora tengo que seguir flotandolo pero solo con los niveles impares, ya
que con los pares estará ordenado
                abuelo = padre(padre(i));
                while(abuelo >= 0 && nodos[abuelo] < nodos[i]){
                    Intercambio(nodos[i],nodos[abuelo]);
                    i = abuelo;
                    abuelo = padre(padre(i));
                    profundidad-=2;
                }
            }else{  //Ahora comprobamos si hay que reordenar respecto a los niveles pares
                //Como en este else siempre compararemos con los abuelos, puedo hacerlo
todo en un while.
                //En el if de arriba primero era con el padre y luego con el abuelo, así
que lo hice en 2 partes
                abuelo = padre(padre(i));
                while(abuelo >= 0 && nodos[abuelo] > nodos[i]){   //Hay que reordenar
respecto a los niveles pares
                    Intercambio(nodos[abuelo],nodos[i]);
                    i = abuelo;
                    abuelo = padre(padre(i));
                    profundidad-=2;
                }
            }


    }else{//Nivel impar
        if(padre(i) >= 0 && nodos[padre(i)] > nodos[i]){ //Que haya que reordenar
respecto a los niveles pares
            Intercambio(nodos[i], nodos[padre(i)]);
            i = padre(i);
            profundidad--;
            //Ahora tengo que seguir flotandolo pero solo con los niveles pares, ya que
con los impares estará ordenado

            abuelo=padre(padre(i));
            while(abuelo >= 0 && nodos[abuelo] > nodos[i]){
                Intercambio(nodos[i],nodos[abuelo]);
                i = abuelo;
                abuelo = padre(padre(i));
                profundidad-=2;
            }
        }else{  //Ahora comprobamos si hay que reordenar respecto a los niveles impares

            //Como en este else siempre compararemos con los abuelos, puedo hacerlo todo
en un while.
            //En el if de arriba primero era con el padre y luego con el abuelo, así que
lo hice en 2 partes
            abuelo = padre(padre(i));
            while(abuelo >= 0 && nodos[abuelo] < nodos[i]){   //Hay que reordenar
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Descarga la app de Wuolah desde tu store favorita

```
respecto a los niveles pares
                Intercambio(nodos[abuelo],nodos[i]);
                i = abuelo;
                abuelo=padre(padre(i));
                profundidad-=2;
            }
        }
    }
}

template <typename T>
void ApomM<T>::Intercambio(T& a, T& b){
    T aux = a;
    a = b;
    b = aux;
}
template <typename T>
inline void ApomM<T>::suprimir()
{
    assert(ultimo > -1); // ApomM no vacío

    if (--ultimo > -1){ // ApomM no queda vacío
        nodos[0] = nodos[ultimo+1];
        std::cout<<"Va a llamar a hundir"<<std::endl;
        if (ultimo > 0) hundir(0); // Quedan dos o más elementos. =>  Reordenar
    }
}

template <typename T>
inline void ApomM<T>::suprimirMax(){
    assert(ultimo > -1);

    if(--ultimo > -1){    //APOmM no queda vacio

        if(ultimo >= 2){ //TIENE 2 HIJOS y al menos 1 nieto. El maximo está en uno de los
2 hijos
            if(nodos[1] > nodos[2]){
                nodos[1] = nodos[ultimo+1];
                hundir(1);
            }else{
                nodos[2] = nodos[ultimo+1];
                hundir(2);
            }
        }else if(ultimo == 1){ //Tiene 2 hijos y ningun nieto. Se elimina el hijo derecho
y en el izquiero queda el menor
            nodos[1] = std::min(nodos[1],nodos[2]);
        }//else Tiene 1 solo hijo, por lo que se elimina el hijo (que es el maximo y ya
está)
    }//else El ApomM era solo la raiz, que al eliminarla se queda vacío
}


//Este algoritmo va a intercambiar i con el menor de sus dos hijos


//DESDE SUPRIMIR SOLO SE LE VA A LLAMAR CUANDO SE ELIMINAR LA RAIZ, PERO PUEDE QUE QUIERA
ELIMINAR OTRO NODO
//DE MANERA QUE SOLAMENTE TENGO QUE PASARLE EL NODO A HUNDIR
template <typename T>
void ApomM<T>::hundir(nodo i){
    //Ahora hay que determinar profundidad de i, buscar el mayor(nivel impar) o
menor(nivel par) nieto y reordenar esos 3

    nodo nietos[4], reordenar[3];
    nodo nietoInter,hijoInter;     //El que finalmente se va a intercambiar
    nietos[0] = hIzq(hIzq(i));
    nietos[1] = hDer(hIzq(i));
    nietos[2] = hIzq(hDer(i));
    nietos[3] = hDer(hDer(i));
    std::cout<<"En hundir"<<std::endl;
    int profundidad = log2(i+1);
    if(nietos[0] <=ultimo){
        if(profundidad%2==0){    //Voy a buscar el mínimo de los nietos
            std::cout<<"i está en nivel par"<<std::endl;
            nietoInter = nietos[0];
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

```
            if(nietos[1] <= ultimo){
                if(nodos[nietos[1]]<nodos[nietoInter])nietoInter=nietos[1];
                if(nietos[2] <=ultimo){
                    if(nodos[nietos[2]]<nodos[nietoInter])nietoInter=nietos[2];
                    if(nietos[3] <=ultimo){
                        if(nodos[nietos[3]]<nodos[nietoInter])nietoInter=nietos[3];
                    }
                }
            }

            hijoInter = padre(nietoInter);
            std::cout<<"Para i= "<<i<<" el nieto a intercambiar es: "<<nietoInter<<" y su
padre: "<<hijoInter<<std::endl;

            //Tengo que reordenar a 3 los nodos i, hijoInter, nietoInter

            reordenar[0] = nodoMinimo(i,hijoInter,nietoInter);
            reordenar[1] = nodoMaximo(i,hijoInter,nietoInter);
            if(i != reordenar[0] && i != reordenar[1])reordenar[2] = i;
            else if(hijoInter != reordenar[0] && hijoInter != reordenar[1])reordenar[2] =
hijoInter;
            else if(nietoInter != reordenar[0] && nietoInter != reordenar[1])reordenar[2]
= nietoInter;
            std::cout<<"El orden al final será: "<<nodos[reordenar[0]]<<"
"<<nodos[reordenar[1]]<<"  "<<nodos[reordenar[2]]<<std::endl;
            T valores[3];
            valores[0] = nodos[reordenar[0]];
            valores[1] = nodos[reordenar[1]];
            valores[2] = nodos[reordenar[2]];
            nodos[i] = valores[0];
            nodos[hijoInter] = valores[1];
            nodos[nietoInter] = valores[2];

            hundir(nietoInter); //Sigo reordenando a partir de nietoInter hacia abajo

        }else{  //i está en nivel impar
            std::cout<<"i está en nivel impar"<<std::endl;

            nietoInter = nietos[0];
            if(nietos[1] <= ultimo){
                if(nodos[nietos[1]]>nodos[nietoInter])nietoInter=nietos[1];
                if(nietos[2] <=ultimo){
                    if(nodos[nietos[2]]>nodos[nietoInter])nietoInter=nietos[2];
                    if(nietos[3] <=ultimo){
                        if(nodos[nietos[3]]>nodos[nietoInter])nietoInter=nietos[3];
                    }
                }
            }
            std::cout<<"Para i= "<<i<<" el nieto a intercambiar es:
"<<nietoInter<<std::endl;


            hijoInter = padre(nietoInter);
            //Tengo que reordenar a 3 los nodos i, hijoInter, nietoInter

            reordenar[0] = nodoMaximo(i,hijoInter,nietoInter);
            reordenar[1] = nodoMinimo(i,hijoInter,nietoInter);
            if(i != reordenar[0] && i != reordenar[1])reordenar[2] = i;
            else if(hijoInter != reordenar[0] && hijoInter != reordenar[1])reordenar[2] =
hijoInter;
            else if(nietoInter != reordenar[0] && nietoInter != reordenar[1])reordenar[2]
= nietoInter;
            std::cout<<"El orden al final será: "<<nodos[reordenar[0]]<<"
"<<nodos[reordenar[1]]<<"  "<<nodos[reordenar[2]]<<std::endl;
            T valores[3];
            valores[0] = nodos[reordenar[0]];
            valores[1] = nodos[reordenar[1]];
            valores[2] = nodos[reordenar[2]];
            nodos[i] = valores[0];
            nodos[hijoInter] = valores[1];
            nodos[nietoInter] = valores[2];

            hundir(nietoInter); //Sigo reordenando a partir de nietoInter hacia abajo

        }
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Descarga la app de Wuolah desde tu store favorita

Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play

```
    }else{//Fin if que comprueba que hay al menos un nieto
        //En este caso no hay nietos, por lo que hay que reordenar solo con uno de los 2
hijos
        if(profundidad%2==0){
            nodo hijomin;
            if(hIzq(i) <=ultimo){
                hijomin = hIzq(i);
                if(hDer(i)<=ultimo && nodos[hDer(i)] < nodos[hijomin])hijomin = hDer(i);

                if(nodos[hijomin] < nodos[i])Intercambio(nodos[hijomin],nodos[i]);
                //i = hijomin;
                //Ya no se hacen más llamadas recursivas porque no hay más hijos por debajo
            }

        }else{
            nodo hijomax;
            if(hIzq(i) <=ultimo){
                hijomax = hIzq(i);
                if(hDer(i)<=ultimo && nodos[hDer(i)] > nodos[hijomax])hijomax = hDer(i);

                if(nodos[hijomax] > nodos[i])Intercambio(nodos[hijomax],nodos[i]);
                //i = hijomax;
                //Ya no se hacen más llamadas recursivas porque no hay más hijos por debajo
            }
        }
    }
    //Si no hay ni hijos ni nietos, no se hacen llamadas recursivas
}

template <typename T>
typename ApomM<T>::nodo ApomM<T>::nodoMinimo(typename ApomM<T>::nodo n1, typename
ApomM<T>::nodo n2, typename ApomM<T>::nodo n3){
    if(nodos[n1]<nodos[n2]){
        return (nodos[n1]<nodos[n3]) ? n1 : n3;
    }else{ //n2<n1      ( los 3 nodos deberían ser distintos)
        return (nodos[n2]<nodos[n3]) ? n2 : n3;
    }
}

template <typename T>
typename ApomM<T>::nodo ApomM<T>::nodoMaximo(typename ApomM<T>::nodo n1, typename
ApomM<T>::nodo n2, typename ApomM<T>::nodo n3){
    if(nodos[n1]>nodos[n2]){
        return (nodos[n1]>nodos[n3]) ? n1 : n3;
    }else{ //n2>n1      ( los 3 nodos deberían ser distintos)
        return (nodos[n2]>nodos[n3]) ? n2 : n3;
    }
}

template <typename T>
inline ApomM<T>::~ApomM()
{
    delete[] nodos;
}


template <typename T>
ApomM<T>::ApomM(const ApomM<T>& A) : nodos(new T[A.maxNodos]), maxNodos(A.maxNodos),
ultimo(A.ultimo){
    // copiar el vector
    for (nodo n = 0; n <= ultimo; n++)
        nodos[n] = A.nodos[n];
}

template <typename T>
ApomM<T>& ApomM<T>::operator =(const ApomM<T>& A)
{
    if (this != &A) { // evitar autoasignación
        // Destruir el vector y crear uno nuevo si es necesario
        if (maxNodos != A.maxNodos) {
            delete[] nodos;
            maxNodos = A.maxNodos;
            nodos = new T[maxNodos];
        }
        ultimo = A.ultimo;
```

WUOLAH
Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

```
        // Copiar el vector
        for (nodo n = 0; n <= ultimo; n++)
            nodos[n] = A.nodos[n];
    }
    return *this;
}

//Funciones de QuesoViejo_ para mostrarlo por pantalla
template <typename T>
void ApomM<T>::Mostrar(){
    using namespace std;
    if(ultimo != -1){
        cout<<"Raiz: "<<nodos[0]<<endl;
        Mostrar_descendencia(0);
    }
}

template <typename T>
void ApomM<T>::Mostrar_descendencia(ApomM<T>::nodo i){
    //Si i ya no pertenece al arbol o alguno de sus hijos, termina él solito.
    using namespace std;

    if(hIzq(i) <= ultimo){
        cout<<"Hijo izqdo de "<<nodos[i]<<": "<<nodos[hIzq(i)]<<endl;
        Mostrar_descendencia(hIzq(i));
    }
    if(hDer(i) <= ultimo){
        cout<<"Hijo drcho de "<<nodos[i]<<": "<<nodos[hDer(i)]<<endl;
        Mostrar_descendencia(hDer(i));
    }
}
#endif // ApomM_H
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

```
/*
Ejercicio 1 Práctica 5

QuesoViejo_

Ejemplo del contenido del fichero NUMS.dat:

-1
2 -1 3 4 -1 -1 5 -1 -1

*/

#include "AbinDin.h"
#include "abin_E-S.h"
#include <iostream>
using namespace std;

/*
    Este ejercicio se tiene que hacer con un Abin ya que un APO impone más restricciones
(como que el árbol sea completo)
    que no nos interesan para el problema
*/

template <typename T>
void eliminarElemento(const T& e, Abin<T>& A);

template <typename T>
typename Abin<T>::nodo BuscarEnAbin(const T& e, typename Abin<T>::nodo actual, const
Abin<T>& A);

int main(){
    Abin<int> A;

    cout << "\n*** Lectura de árbol binario fichero de datos ***\n";
    ifstream fe("NUMS.dat"); // abrir fichero de entrada
    rellenarAbin(fe, A); // desde fichero
    fe.close();
    cout << "\n*** Mostrar árbol binario A ***\n";
    imprimirAbin(A); // en std::cout

    int e;
    cout<<"Introduce el elemento a eliminar: ";
    cin >> e;
    eliminarElemento(e,A);

    cout<<"El arbol queda: \n";
    imprimirAbin(A); // en std::cout
}

//Para el ejercicio 1, supondremos que se ha introducido el árbol de manera correcta

//EL tipo T tiene que tener definido operator<
template <typename T>
void eliminarElemento(const T& e, Abin<T>& A){

    typename Abin<T>::nodo p = BuscarEnAbin(e,A.raizB(),A);
    if(p != Abin<T>::NODO_NULO){
        //Hay que hundir p a una hoja para poder eliminarlo (POR AQUÍ LO DEJO HOY)

        typename Abin<T>::nodo hIzq,hDer,intercambio;
        hIzq = A.hijoIzqdoB(p);
        hDer = A.hijoDrchoB(p);

        T aux;
        while(hIzq != Abin<T>::NODO_NULO || hDer != Abin<T>::NODO_NULO){//Mientras que no
sea hoja
            if(hIzq != Abin<T>::NODO_NULO && hDer != Abin<T>::NODO_NULO){
                if(A.elemento(hIzq) < A.elemento(hDer)){
                    intercambio = hIzq;
                }else{
                    intercambio=hDer;
                }
            }else if(hIzq != Abin<T>::NODO_NULO){
                intercambio = hIzq;
            }else{
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Descarga la app de Wuolah desde tu store favorita

```
                intercambio = hDer;
            }

            //Ahora que ya sé con que nodo intercambiarlo, intercambio los elementos
            aux = A.elemento(p);
            A.elemento(p) = A.elemento(intercambio);
            A.elemento(intercambio) = aux;

            //Actualizo el nodo p donde está el elemento a eliminar:
            p = intercambio;

            //Actualizo los nodos hijos:
            hIzq = A.hijoIzqdoB(p);
            hDer = A.hijoDrchoB(p);
        }

        //Cuando salga es que el nodo p con el elemento a eliminar es una hoja, compruebo
si tiene padre o si es raiz:

        if(A.raizB() == p){
            A.eliminarRaizB();
        }else{
            if(A.hijoIzqdoB(A.padreB(p)) == p){
                A.eliminarHijoIzqdoB(A.padreB(p));
            }else{
                A.eliminarHijoDrchoB(A.padreB(p));
            }
        }

    }
}

template <typename T>
typename Abin<T>::nodo BuscarEnAbin(const T& e, typename Abin<T>::nodo actual, const
Abin<T>& A){
    if(actual != Abin<T>::NODO_NULO){
        if(A.elemento(actual) == e){
            return actual;
        }else{
            if(BuscarEnAbin(e,A.hijoIzqdoB(actual),A) != Abin<T>::NODO_NULO){
                return BuscarEnAbin(e,A.hijoIzqdoB(actual),A);  //Si lo encuentra no hace
la llamada recursiva derecha
            }else{
                return BuscarEnAbin(e,A.hijoDrchoB(actual),A);
            }
        }
    }else return Abin<T>::NODO_NULO;
}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play

```
/*
Ejercicio 2 Práctica 5

QuesoViejo
*/

#include "APOmM.h"
#include "APO_E-S.h"
#include <iostream>


/**
    Da igual cómo introduzca el APOmM, que la función insertar lo va dejando correcto
*/
using namespace std;


int main(){

    ApomM<int> A(31);  //APO de 4 niveles
    LeerAPOmM(A);
    cout<<"\nEL APOmM actual es: ";
    A.Mostrar();

    int valor;
    cout<<"Introduce el valor a insertar: ";
    cin >> valor;
    A.insertar(valor);

    cout<<"\nEL APOmM actual es: ";
    A.Mostrar();

    cout<<"Introduce el valor a insertar: ";
    cin >> valor;
    A.insertar(valor);

    cout<<"\nEL APOmM actual es: ";
    A.Mostrar();


}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.
WUOLAH

```
/*
Ejercicio 3 Práctica 5

QuesoViejo_
*/

#include "APOmM.h"
#include "APO_E-S.h"
#include <iostream>



/**
    Da igual cómo introduzca el APOmM, que la función insertar lo va dejando correcto
*/
using namespace std;


int main(){

    ApomM<int> A(31);   //APO de 4 niveles
    LeerAPOmM(A);
    cout<<"\nEL APOmM actual es: ";
    A.Mostrar();
    cout<<"FIN DE MOSTRAR"<<endl;

    cout<<"Se va a eliminar el máximo"<<endl;
    A.suprimirMax();
    cout<<"\nEL APOmM actual es: ";
    A.Mostrar();
    cout<<"FIN DE MOSTRAR"<<endl;

    cout<<"Se va a eliminar el máximo"<<endl;
    A.suprimirMax();
    cout<<"\nEL APOmM actual es: ";
    A.Mostrar();
    cout<<"FIN DE MOSTRAR"<<endl;

    cout<<"Se va a eliminar el máximo"<<endl;
    A.suprimirMax();
    cout<<"\nEL APOmM actual es: ";
    A.Mostrar();
    cout<<"FIN DE MOSTRAR"<<endl;



}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

```
/*
Ejercicio 4 Práctica 5

QuesoViejo_


Ejemplo de contenido para el finchero EJ4.dat:

#
a b x # y # z # # c x # y # z # # d # #

*/

#include "AgenDin.h"
#include "Agen_E-S.h"
#include <iostream>


template <typename T>
bool esTernario(const Agen<T>& A);

template <typename T>
bool esTernario_rec(typename Agen<T>::nodo n, const Agen<T>& A);

template <typename T>
int numHijos(typename Agen<T>::nodo n, const Agen<T>& A);

using namespace std;

int main()
{

    Agen<char> A;

    cout << "*** Lectura del árbol A ***\n";

    cout << "\n*** Lectura de árbol A desde fichero ***\n";
    ifstream fe("EJ4.dat"); // Abrir fichero de entrada.
    rellenarAgen(fe, A); // Desde fichero.
    fe.close();

    cout << "\n*** Mostrar árbol A ***\n";
    imprimirAgen(A); // En std::cout

    cout<<"\n\nResultado:"<<endl;
    if(esTernario(A)){
        cout<<"Es ternario"<<endl;
    }else{
        cout<<"No es ternario"<<endl;
    }

}

template <typename T>
bool esTernario(const Agen<T>& A){
    return esTernario_rec(A.raiz(),A);
}

template <typename T>
bool esTernario_rec(typename Agen<T>::nodo n, const Agen<T>& A){

    int numeroHijos = numHijos(n,A);

    if(numeroHijos == 0){
        return true;
    }else if (numeroHijos == 3){
        typename Agen<T>::nodo hijo = A.hijoIzqdo(n);
        bool resultado = esTernario_rec(hijo,A);
        hijo = A.hermDrcho(hijo);

        while(hijo != Agen<T>::NODO_NULO && resultado){
            resultado = resultado && esTernario_rec(hijo,A);
            hijo = A.hermDrcho(hijo);
        }
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com

```
        return resultado;
    }else{
        return false;
    }

}

template <typename T>
int numHijos(typename Agen<T>::nodo n, const Agen<T>& A){
    int contador;
    typename Agen<T>::nodo hijo;
    hijo = A.hijoIzqdo(n);

    contador = 0;
    while(hijo != Agen<T>::NODO_NULO){
        contador++;
        hijo = A.hermDrcho(hijo);
    }
    return contador;
}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Descarga la app de Wuolah desde tu store favorita

Descarga la APP de Wuolah.
Ya disponible para el móvil y la tablet.
Available on the App Store
GET IT ON Google Play

```
/*
Ejercicio 5 Práctica 5

QuesoViejo

Ejemplo de contenido para el finchero EJ5.dat (el árbol del enunciado):

#
V V B # B # V B # B # N # N # # B # # V B # B # B # V N # B # B # N # # # V N # B # V N #
B # N # N # # N # # V N # N # B # B # # #


*/

#include "AgenDin.h"
#include "Agen_E-S.h"
#include <iostream>
#include <cmath>


template <typename T>
int Altura(const Agen<T>& A);

template <typename T>
int Altura_nodo(typename Agen<T>::nodo n, const Agen<T>& A);

template <typename T>
char** FuncionEvaluarArbol(const Agen<T>& A);

template <typename T>
void fun_rec(typename Agen<T>::nodo n, int p, int h, int Fini, int Ffin, int Cini, int
Cfin, char** matriz, const Agen<T>& A);

using namespace std;

int main()
{

    Agen<char> A;

    cout << "*** Lectura del árbol A ***\n";

    cout << "\n*** Lectura de árbol A desde fichero ***\n";
    ifstream fe("EJ5.dat"); // Abrir fichero de entrada.
    rellenarAgen(fe, A); // Desde fichero.
    fe.close();

    cout << "\n*** Mostrar árbol A ***\n";
    imprimirAgen(A); // En std::cout

    int h = Altura(A);
    cout<<"Altura del arbol: "<<h<<endl;

    char**matriz;
    matriz = FuncionEvaluarArbol(A);

    cout<<"\n\nLA MATRIZ ES: \n"<<endl;
    for(int i = 0 ; i < pow(2,h) ; i++){
        for(int j = 0; j < pow(2,h) ; j++){
            cout<<matriz[i][j]<<" ";
        }
        cout<<endl;
    }


    //Debido a que la matriz se creó con memoria dinámica:
    for(int i = 0 ; i < pow(2,h) ; i++){
        delete[] matriz[i];
    }
    delete[] matriz;
}
```

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.
WUOLAH

```
//Suponemos que como mínimo tiene 1 nodo (raiz)
template <typename T>
char** FuncionEvaluarArbol(const Agen<T>& A){
    int h = Altura(A);
    int p = 0;

    int tam = pow(2,h);
    T** matriz = new T*[tam]; //Lo hago dinámico para que no se destruya al devolverlo

    for(int i = 0; i < tam ; i++){
        matriz[i] = new T[tam];
    }

    fun_rec(A.raiz(),0,h,0,tam,0,tam,matriz,A);
    return matriz;
}


//Ffin y Cfin son uno más de la última con valor, es decir, si es de 0 a 3 de las que se
encarga, entonces
//ini = 0 y fin = 4
//AMBAS SE PODRÍAN HABER CALCULADO A PARTIR DE FINI SUMANDO UN DESPLAZAMIENTO PERO ASÍ
QUEDA MÁS CLARO
template <typename T>
void fun_rec(typename Agen<T>::nodo n, int p, int h, int Fini, int Ffin, int Cini, int
Cfin, char** matriz, const Agen<T>& A){
    if(A.elemento(n) != 'V'){
        for(int i = Fini; i < Ffin ; i++){
            for(int j = Cini; j < Cfin ; j++){
                matriz[i][j] = A.elemento(n);
            }
        }
    }else{
            typename Agen<T>::nodo hijo = A.hijoIzqdo(n);
            fun_rec(hijo,p+1,h,     Fini+0,Fini+pow(2,h-(p+1)),
Cini+0,Cini+pow(2,h-p),       matriz,A);
            hijo = A.hermDrcho(hijo);
            fun_rec(hijo,p+1,h,     Fini+0,Fini+pow(2,h-(p+1)),
Cini+pow(2,h-(p+1)),Cini+pow(2,h-p),        matriz,A);
            hijo = A.hermDrcho(hijo);
            fun_rec(hijo,p+1,h,     Fini+pow(2,h-(p+1)),Fini+pow(2,h-p),
Cini+pow(2,h-(p+1)),Cini+pow(2,h-p),       matriz,A);
            hijo = A.hermDrcho(hijo);
            fun_rec(hijo,p+1,h,     Fini+pow(2,h-(p+1)),Fini+pow(2,h-p),
Cini+0,Cini+pow(2,h-(p+1)),           matriz,A);
    }
}




template <typename T>
int Altura(const Agen<T>& A){
    return Altura_nodo(A.raiz(),A);
}

template <typename T>
int Altura_nodo(typename Agen<T>::nodo n, const Agen<T>& A){
    if(n == Agen<T>::NODO_NULO){
        return -1;
    }else{
        typename Agen<T>::nodo hijo = A.hijoIzqdo(n);
        int maxaltura = Altura_nodo(hijo,A);

        if(hijo != Agen<T>::NODO_NULO)hijo = A.hermDrcho(hijo);

        while(hijo != Agen<T>::NODO_NULO){
            maxaltura = std::max(maxaltura, Altura_nodo(hijo,A));
            hijo = A.hermDrcho(hijo);
        }
        return 1+maxaltura;
    }
}
```

WUOLAH

Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Prueba de nivel y primera clase de inglés por teléfono sin coste: ringteacher.com