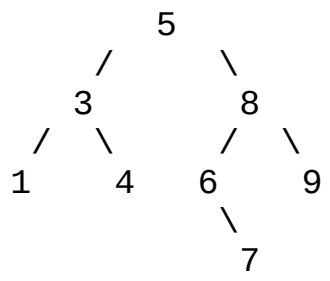
Dado un árbol binario de búsqueda, implementa una función para **imprimir las etiquetas de los nodos en orden de mayor a menor profundidad**. Si tienen la misma profundidad pueden aparecer en cualquier orden. Ejemplo:



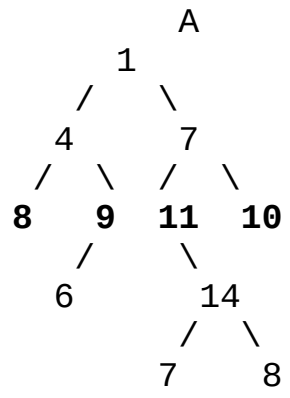
El resultado seria 7,1,4,6,9,3,8,5.

```
#include "bintree.h"
#include <queue>
#include <stack>
#include<iostream>
using namespace std;
//sin usar iteradores

void ImprimeProfundidad(bintree<int> &ab){

    queue<bintree<int>::node> q;

    bintree<int>::node n = ab.root();

    q.push(n);

    stack<bintree<int>::node> p;

    while (!q.empty()){
      n= q.front();
      q.pop();
      p.push(n);//ponemos en la pila
      //ponemos los hijos en la cola
      q.push(n.right());
      q.push(n.left());

    }
    while (!p.empty()){
      n=p.top();
      cout<<*n<<" ";
      p.pop();
    }
}
```

Implementa una función

list<int> Maximo (bintree<int> & ab);

que dado un árbol binario A, devuelva una lista con las etiquetas del nivel que tenga un mayor número de nodos.



L={8,9,11,10}

En caso de que haya más de una lista solución, basta con devolver una de ellas

```
#include "bintree.h"
#include <queue>
#include <list>
#include <iostream>

using namespace std;

//sin usar iteradores
void Imprimir (list<int> & l){
    list<int>::iterator it;
   for (it=l.begin();it!=l.end();++it){
       cout<<*it<<" ";
   }
   cout<<endl;
}


list<int> Maximo(bintree<int> &ab){

    queue<pair<bintree<int>::node,int> > q;

    bintree<int>::node n = ab.root();
    pair<bintree<int>::node,int> p(n,0);

    int level =0;
    list<int>laux;
    list<int>lout;
```

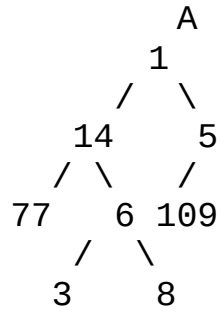
```
    q.push(p);

    while (!q.empty()){
      p= q.front();
      if (p.second==level){
      laux.push_back(*(p.first));
      }
      else{
        if (laux.size()>lout.size()){
          lout=laux;
          level++;
        }
      laux.clear();
      laux.push_back(*(p.first));
      }
      q.pop();
      n=p.first;
      int aux_level =p.second+1;
      if (!n.left().null()){
       p.first=n.left();
       //cout<<"Insertado "<<*(n.left())<<endl;
       p.second=aux_level;
       q.push(p);
      }
      if (!n.right().null()){
         p.first=n.right();
      p.second=aux_level;
         q.push(p);
      }
    }
    if (laux.size()>lout.size()){
          lout=laux;
   }
    return lout;
}

int main(){
      bintree<int> a;
 // ej:n1n4n8xxn9n6xxxn7n11xn14n7xxn8xxn10xx

 cout<<"Introduce un arbol:";

 cin>>a;
 cout<<endl<<"El arbol insertado: "<<endl;
 cout<<a<<endl;
 list<int> l=Maximo(a);
 cout<<"El level de mayor longitud:";
 Imprimir(l);
}
```

Implementar una función

list<int> caminodemenores (const bintree<int> & A);

que dado un árbol binario A, devuelva en L el camino de la raíz a una hoja de forma que la suma de sus etiquetas sea la menor posible. No se pueden usar iteradores.

Ejemplo:



L={1,14,6,3}

```
#include "bintree.h"
#include <queue>
#include <list>
#include <limits>
#include <iostream>
using namespace std;
//sin usar iteradores

void Imprimir (list<int> & l){
    list<int>::iterator it;
   for (it=l.begin();it!=l.end();++it){
       cout<<*it<<" ";
   }
   cout<<endl;
}

pair<int,list<int>> caminomen_nodo(bintree<int>::node n){
     if (n.null())
      return
pair<int,list<int>>(numeric_limits<int>::max(),list<int>());
     else
      if (n.left().null() && n.right().null())//Paramos en la
       return pair<int,list<int>>(*n,list<int>(1,*n)); //hoja
     else{
         pair<int,list<int>> li=caminomen_nodo(n.left());
         pair<int,list<int>> ld=caminomen_nodo(n.right());
         if (li.first<ld.first){
             li.second.push_front(*n);
             li.first+=*n;
             return li;
```

```
        }
        else{
              ld.second.push_front(*n);
              ld.first+=*n;
              return ld;
        }
     }

}
list<int> caminodemenores(bintree<int> &ab){
    return caminomen_nodo(ab.root()).second;
}


int main(){

// Creamos el árbol:

  //         3
  //       /  \
  //      6    8
  //      \   / \
  //      8  5   4
  //     /   \
  //   3     4

list<int> listanodos;
bintree<int> Arb(3);
Arb.insert_left(Arb.root(), 6);
Arb.insert_right(Arb.root(), 8);
Arb.insert_right(Arb.root().left(), 8);
Arb.insert_left(Arb.root().left().right(), 3);
Arb.insert_left(Arb.root().right(), 5);
Arb.insert_right(Arb.root().right(),4);
Arb.insert_right(Arb.root().right().left(),4);


cout << "caminosminimos:" <<endl;
listanodos=caminodemenores(Arb);


Imprimir(listanodos);

}
```

Implementar una función

bool esABB (bintree <int> & A);

que devuelva true si el arbol binario A es un ABB y false en caso contrario

False

True

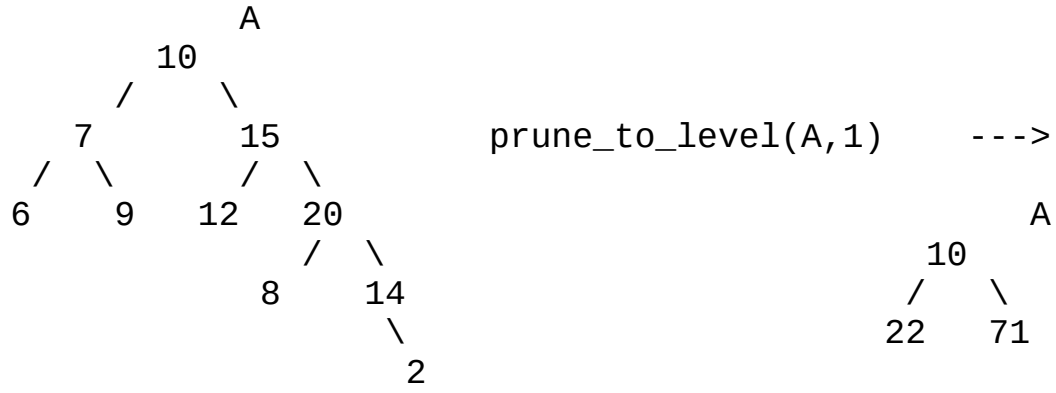
```
#include <iostream>
#include <bintree.h>
#include <limits>
using namespace std;
bool esAbb(bintree<int>::node n,int min,int max){
   if (n.null())
     return true;
   else{
     if ((*n<min || *n>max)) return false;

     return esAbb(n.left(),min,*n) &&
            esAbb(n.right(),*n,max);
   }
}

int main(){
 bintree<int> a;

 // ej:n5n3n2xxn4xxn8n7xxn9xx es un arbol que si es ABB
 // ej:n1n2n4xxn8xxn3n6xxn7xx es un arbol que no es ABB

 cout<<"Introduce un arbol:";
 cin>>a;
 if
(esAbb(a.root(),numeric_limits<int>::min(),numeric_limits<int
>::max()))
  cout<<"Es Abb"<<endl;
 else
    cout<<"No es Abb"<<endl;
}
```

Implementar una función

void prune_to_level(bintree<int> & A, int level);

que pode todos los nodos de un árbol binario A por debajo del nivel level sustituyendo la etiqueta de los nodos del nivel más profundo que quede tras podar por la suma de sus descendientes (incluyendo la etiqueta del propio nodo)

Ejemplo:




```
#include <iostream>
#include "bintree.h"
using namespace std;

int Suma(bintree<int>::node n){
  if (n.null())
    return 0;
  else
    return *n + Suma(n.left()) + Suma(n.right());
}

void prune_to_level(bintree<int> &A,int level){
  queue<pair<bintree<int>::node,int> >micola;
  pair<bintree<int>::node, int> p(A.root(),0);
  pair<bintree<int>::node, int> aux;
  bintree<int>  destino;

  //Insertamos en la cola el pair
  //correspondiente a la raíz (con su nivel, 0)
  micola.push(p);

  //Mientras queden elementos en la cola
  while (!micola.empty()){
    //Tomamos el primer elemento de la cola
    //y lo extraemos
    p = micola.front();
    micola.pop();
```

```
    //Si es del nivel a podar
    if (p.second==level ){
      //Sustituimos su etiqueta por la suma del subárbol
      *(p.first)=Suma(p.first);
      //y podamos sus subárboles izquierdo y derecho
      A.prune_left(p.first,destino);
      A.prune_right(p.first,destino);
    }
    //Si no es del nivel a podar
    else{
      //Si tiene hijo izquierdo
      if (!p.first.left().null()){
        //Insertamos el pair de ese nodo en la cola
        aux.first = p.first.left();
        aux.second = p.second+1;
        micola.push(aux);
      }
      //Si tiene hijo derecho
      if (!p.first.right().null()){
        //Insertamos el pair de ese nodo en la cola
        aux.first = p.first.right();
        aux.second = p.second+1;
        micola.push(aux);
      }
    }
  }
}

int main(){
  /*
                      A
                   10
                 /    \
               7       15          prune_to_level(A,1)    --->
             /  \      /  \
           6    9   12   20                                    A
                        /  \                              10
                     8    14                            /   \
                           \                          22   71
                            2

  n10n7n6xxn9xxn15n12xxn20n8xxn14xn2xx

*/
  bintree<int> arbol(10);
  arbol.insert_left(arbol.root(), 7);
  arbol.insert_right(arbol.root(), 15);
  arbol.insert_left(arbol.root().left(), 6);
  arbol.insert_right(arbol.root().left(), 9);
```

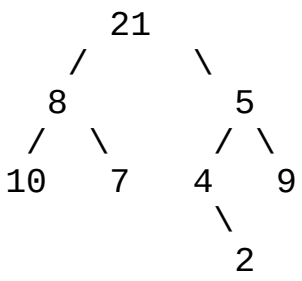
```
  arbol.insert_left(arbol.root().right(), 12);
  arbol.insert_right(arbol.root().right(), 20);
  arbol.insert_left(arbol.root().right().right(), 8);
  arbol.insert_right(arbol.root().right().right(),14);
  arbol.insert_right(arbol.root().right().right().right(),2);

  prune_to_level(arbol,1);

  cout << "Preorden del arbol resultante: ";
  for (bintree<int>::preorder_iterator i =
arbol.begin_preorder(); i!=arbol.end_preorder(); ++i)
    cout << *i << " ";
  cout << endl;

  cout << "Listado por niveles: ";
  for (bintree<int>::level_iterator i = arbol.begin_level();
i!=arbol.end_level(); ++i)
    cout << *i << " ";
  cout << endl;

}
```

Un recorrido guiado sobre un árbol binario de enteros comienza listando la raiz para a continuación en cada iteración, seleccionar el nodo con etiqueta más pequeña de entre los nodos disponibles en ese momento que no hayan sido listados, independientemente de en qué rama se encuentren. Se entiende por nodo disponible aquel cuyo padre ya ha sido procesado (excluyendo la raiz). Implementar una función

void GuideBinario (const bintree<int> & A)

que permita hacer un recorrido guiado en un árbol binario. Pueden usarse estructuras auxiliares.

Ejemplo:

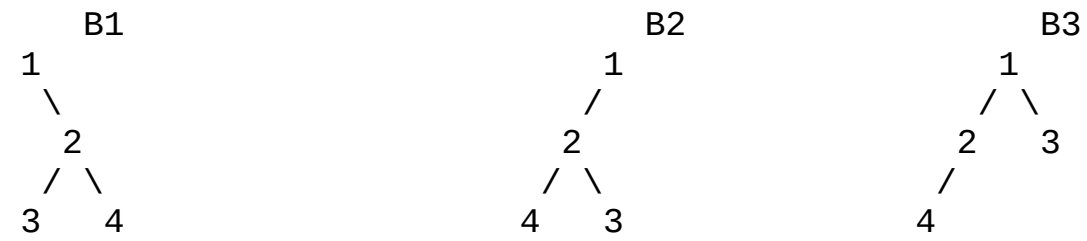


Recorrido guiado: 21, 5, 4, 2, 8, 7, 9, 10

```
#include "bintree.h"
#include <queue>
#include <iostream>
using namespace std;

bool operator<(const bintree<int>::node n1,
               const bintree<int>::node n2){
   return *n1>*n2;

}
void recorrido_guiado(bintree<int> &a){
   priority_queue<bintree<int>::node> mipq;
   mipq.push(a.root());
   cout<<endl;
   cout<<"Recorrido guiado: ";
   while (!mipq.empty()){
    bintree<int>::node n=mipq.top();
    mipq.pop();
    cout<<*n<<" ";
    if (!n.left().null())
        mipq.push(n.left());
    if (!n.right().null())
        mipq.push(n.right());
   }
   cout<<endl;
}
```

```
int main(){
 bintree<int> a;

//n21n8xxn5n4xn2xxn7xx
 cout<<"Introduce un arbol:";

 cin>>a;
 recorrido_guiado(a);
}
```

Dos árboles binarios B1, B2 son isomorfos si se pueden aplicar una serie de inversiones entre los hijos derechos e izquierdos de los nodos de B2 de manera que quede un árbol semejante a B1, es decir que tiene la misma estructura.
Por ejemplo



B1 es isomorfo a B2 pero no a B3. En particular si dos árboles son isomorfos la cantidad de nodos que tienen una profundidad y altura dadas es la misma. Por ejemplo en el caso anterior B1 y B2 tienen 4 nodos, de los cuales uno a profundidad 1, 2 a profundidad 2 y por supuesto 1 a profundidad 0 (raíz).

Concretamente B1 es isomorfo a B2 si:
(a) Ambos son vacios
b) Los subárboles de los hijos izquierdos de B1 y B2 son isomorfos entre sí y los derechos también ó el subárbol del hijo izquierdo de B1 es isomorfo al derecho de B2 y el derecho de B1 es isomorfo al izquierdo de B2.

Implementar una función
bool btisomorph(bintree<int> &B1, bintree<int> &B2);

que devuelva true si B1 y B2 son isomorfos y false en caso contrario
Nota: Sólo se mira la estructura del árbol, no los valores de las etiquetas de los nodos.

```
#include <iostream>
#include <string>
#include "bintree.h"
using namespace std;

bool btisomorph_nodo( bintree<int>::node &n1,
                      bintree<int>::node &n2){
    if (n1.null() && n2.null())
      return true;
    else
     if (n1.null() || n2.null())
       return false;
```

```
     else{
      if (btisomorph_nodo(n1.left(),n2.left()) &&
          btisomorph_nodo(n1.right(),n2.right()))
        return true;
      else{
        //comprobamos si reflejando coinciden en estructura
        if (btisomorph_nodo(n1.left(),n2.right()) &&
            btisomorph_nodo(n1.right(),n2.left()))
          return true;
        else
          return false;
       }
    }
}

bool btisomorph(const bintree<int> &B1,const bintree<int>
&B2){
    return btisomorph_node(B1.root(),B2.root());
}

int main(){
  //n5n3n2xxn4xxn8n7xxn9xn10xx
  bintree<int> miarbol1;
  cout<<"Introduce el arbol 1 (preorden con n y x)..."<<endl;
  cin>>miarbol1;
  string s;
  getline(cin,s);
  //Ejenplo de isomoros
  //n5n3n2xxn4xxn8n9xxn7xn10xx
  //Otro ejemplo de isomoro
  //n5n3n4xxn2xxn8n9n10xxxn7xx
  //Ejemplo no es isomoro
  //n6n4xxn2xxn4xxn8n9xxn7xn10xx

  bintree<int> miarbol2;
  cout<<"Introduce el arbol 2 (preorden con n y x)..."<<endl;
  cin>>miarbol2;
  getline(cin,s);
  if (btisomorph(miarbol1,miarbol2))
    cout<<"Los arboles son isomorfos "<<endl;
  else
    cout<<"Los arboles no son isomorfos "<<endl;
}
```

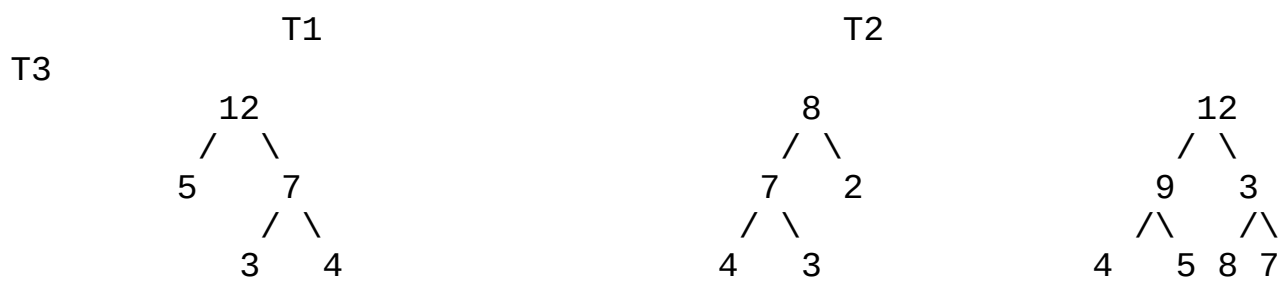
Implementar una función

bool is_recurrent_tree(bintree<int> &T);

que devuelva true si T es un árbol recurrente.

Un árbol binario se considera recurrente si es completo y cada uno de sus nodos interiores es la suma de sus 2 nodos hijos.

Por ejemplo



T1 -----> true
T2 -----> false (la raíz no es la suma de sus hijos)
T3 -----> false (el nodo con etq 3 no es la suma de sus hijos

```
#include <iostream>
#include <string>
#include "bintree.h"
using namespace std;

bool is_recurrent_tree_nodo(bintree<int>::node n){
    if (n.null())
        return true;
    else{
        //si es hoja
        if (n.left().null() && n.right().null())
            return true;
        else{
          //si tiene solo un hijo
          if (n.left().null() || n.right().null())
            return false;
          //comprobamos si la etiqueta del nodo
          //es la suma de la etiquetas de los hijos
          if (!(*n == *(n.left())+*(n.right())))
            return false;
          //comprobamos que se cumple para los hijos
          return is_recurrent_tree_nodo(n.left()) &&
                 is_recurrent_tree_nodo(n.right());
        }
    }
}
```

```
bool is_recurrent_tree(bintree<int> &T){
    return is_recurrent_tree_nodo(T.root());
}


int main(){
  //Ejemplo B1 true
  //n12n5xxn7n3xxn4xx
  //Ejemplo B2 false
  //n8n7n3xxn4xxn2xx
  //Ejemplo B3 false
  //n12n9n4xxn5xxn3n8xxn6xx
  bintree<int> miarbol;
  cout<<"Introduce un arbol (preorden con n y x)..."<<endl;
  cin>>miarbol;
  string s;
  getline(cin,s);
  if (is_recurrent_tree(miarbol))
    cout<<"El arbol es recurrente"<<endl;
  else
    cout<<"El arbol no es recurrente"<<endl;
}
```

Sea T un árbol binario de enteros con n nodos. Se define un **k-nodo** como un nodo **y** que cumple la condición de que el número de descendientes en el subárbol izquierdo de **v** difiere del número de descendientes del subárbol derecho en al menos k. Implementar una función:

list<int> knodos (bintree<int> & A, int k);

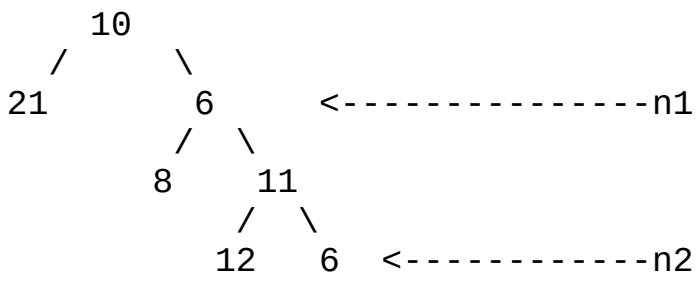
que tenga como entrada un árbol binario de enteros y como salida la lista de las etiquetas de los k-nodos que posee.

```
#include <iostream>
#include <string>
#include "bintree.h"
#include <list>
using namespace std;

template <typename T>
void Imprimir(T comienzo,T fin){
   for (auto it =comienzo; it!=fin; ++it){
      cout<<*it<<" ";
   }
}
int contabiliza (bintree<int>::node n){
    if (n.null())
        return 0;
    else{
      return 1+contabiliza(n.left())+contabiliza(n.right());
    }
}
list<int> knodos_n (bintree<int>::node n, int k){
     if (n.null())
        return list<int>();
     else{
        int ci=contabiliza(n.left());
        int cd=contabiliza(n.right());

        list<int> lout,li,ld;
        if (abs(ci-cd)>=k)
         lout.push_back(*n);
        li=knodos_n(n.left(),k);
        ld=knodos_n(n.right(),k);
        lout.merge(li);
        lout.merge(ld);
        return lout;
     }
}
list<int> knodos (bintree<int> & A, int k){
    return knodos_n(A.root(),k);
```

```
}
int main(){
  //n5n3n2xxn4xxn8n7xxn9xn10xx
  bintree<int> miarbol;
  cout<<"Introduce el arbol  (preorden con n y x)..."<<endl;
  cin>>miarbol;
  string s;
  getline(cin,s);
  list<int>l=knodos(miarbol,1);
  cout<<"Los subarboles son:";
  Imprimir(l.begin(),l.end());
}
```

Implementar una función que devuelva en un vector las etiquetas de los nodos de un arbol binario de enteros que estén entre dos nivele} dados n1 y n2 (0<=n1<n2) ambos inclusive

vector<int> labelinterlevel (bintree<int> &A,int n1, int n2);



v={21,6,8,11,12,6}

```
#include <iostream>
#include <string>
#include "bintree.h"
#include <vector>
#include <queue>

using namespace std;

template <typename T>
void Imprimir(T comienzo,T fin){
   for (auto it =comienzo; it!=fin; ++it){
      cout<<*it<<" ";
   }
}
vector<int> labelinterlevel (bintree<int> &A, int n1, int n2)
{
  typedef pair<bintree<int>::node,int> mipair;

  queue<mipair>micola;
  mipair a(A.root(),0);
  micola.push(a);

  vector<int>vout;
  bool seguir=true;

  while (!micola.empty() && seguir){
    a=micola.front();
    micola.pop();
    if (a.second>=n1 && a.second<=n2)
        vout.push_back(*(a.first));
    if (a.second>n2) seguir=false;
```

```
    else{
        mipair h;
        if (!a.first.left().null()){
            h.first=a.first.left();
            h.second=a.second+1;
            micola.push(h);
        }
        if (!a.first.right().null()){
            h.first=a.first.right();
            h.second=a.second+1;
            micola.push(h);
        }
    }
  }
  return vout;
}
int main(){
  //n5n3n2xxn4xxn8n7xxn9xn10xx
  bintree<int>miarbol;
  cout<<"Introduce el arbol  (preorden con n y x)..."<<endl;
  cin>>miarbol;
  string s;
  getline(cin,s);
  vector<int> v=labelinterlevel(miarbol,2,3);
  cout<<"Las etiquetas de los nodos entre los niveles 2-3
son:"<<endl;
  Imprimir(v.begin(),v.end());
}
```