UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Estructuras de datos. Curso 2020-2021
Convocatoria ordinaria de Enero
Grado en Ingeniería Informática.
Doble Grado en Ingeniería Informática y Matemáticas
Doble Grado en Ingeniería Informática y ADE

1. (0.75 puntos) **(1)** Dados los siguientes recorridos en **preorden** = (Z,A,R,H,L,M,N,T,S,W,Q), y **postorden** = (H,L,R,N,M,A,W,Q,S,T,Z) 1**-a:** No hay ningún árbol binario con esos recorridos asociados; 1**-b:** Hay 1 solo árbol binario con esos recorridos asociados: 1**-c:** Hay dos árboles binarios con esos recorridos asociados ; 1**-d:** Hay múltiples árboles binarios con esos recorridos asociados. Razona la respuesta.

   (2) Supongamos que hacemos las 3 siguientes afirmaciones: (a) El valor máximo en un **AVL<int>** siempre se encuentar en una hoja. (b) Si A es una **tabla hash cerrada** con un **40%** de elementos **vacios** y un **40%** de elementos **borrados** es más eficiente a la hora de buscar elementos que otra tabla hash cerrada B con un **60%** de elementos **vacios** y **sin borrados** (c) Un **APO** puede reconstruirse de **de forma unívoca** dado su recorrido en **inorden 2-a:** Las tres son ciertas **2-b:** Dos son ciertas y una falsa **2-c:** Dos son falsas y una cierta; **2-d:** Las tres son falsas. Razona la respuesta.

   (3) Dadas las siguientes 4 afirmaciones:
   (a) Dados A y B dos árboles binarios distintos con etiquetas diferentes, nunca puede ocurrir simultáneamente: Pre (A) = Post (B) y Post (A)=Pre (B)
   (b) Un ABB no puede reconstruirse de forma unívoca dado su recorrido en preorden
   (c) Un APO puede reconstruirse de forma unívoca dado su recorrido en postorden
   (d) Solo hay un APO que tiene como preorden={4,9,24,33,21,74,63}
   (3-a) Todas son falsas (3-b) Hay 2 ciertas y 2 falsas (3-c) hay 3 ciertas y una falsa(3-d) Todas son ciertas. Razona la respuesta.

2. (1.5 puntos) Se define una **matriz sesgada** como una **matriz nxn** en la solo unos pocos elementos son pares Un ejemplo de este tipo de matrices es la matriz que se muestra. Es una matriz **6x6** de valores enteros. De los 36 elementos, solo 2 son pares

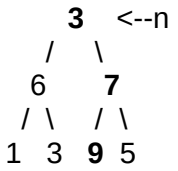
7	9	**4**	5	1	3
5	9	3	7	1	5
1	3	7	3	3	1
5	5	5	5	5	5
3	1	1	**2**	9	9
1	5	1	3	7	7

   Proponer una representación eficiente (basada en el tipo **vector<list>**) para la matriz e implementar: (a) el **operator()** que devuelva el elemento en la fila fil, columna col.
   (b) **un iterador** que itere sobre los elementos pares de la matriz. Han de implementarse (aparte de las de la clase iteradora) las funciones **begin()** y **end()**.

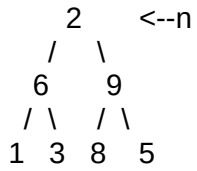
3. (1.5 puntos) Dado un árbol binario A y un nodo n en el mismo, implementar una función:
   bool camino_especial (bintree<int> A, bintree<int> :: node n, int k);

   que devuelva true si existe algún camino desde n a una hoja con las etiquetas de los nodos ordenadas ascendentemente y sumando una cantidad k.

   Ejemplo (k=19):



true
El camino <3,7,9> lo cumple



false
Ningún camino lo cumple

4. (1.5 puntos) Dado un **vector de conjuntos** vector<set<int> > V, implementar una función

   void contar(const vector<set<int> > &V, map<int,int> &veces);

   que devuelva, a través del map **veces**, el número de conjuntos en los que aparece cada uno de los elementos presentes en v.
   Por ejemplo, si V={ <1,2,3>, <2,4>, <3,4,5,9>} entonces debe devolver:
   veces= {<1,1>,<2,2>, <3, 2>, <4,2>, <5,1>, <9,1>}

5. (0.75 puntos) Insertar en el orden indicado (detallando los pasos) las siguientes claves en un **AVL**: {53, 34, 55, 69, 54, 77, 35, 30, 27}

   (b) Insertar los elementos anteriores en el orden indicado (detallando los pasos) en un **APO. Borrar un elemento**

Tiempo: 2.30 horas

Preguntas test
---

1. La respuesta correcta es 1.d

May varios posibles árboles con esos recorridos. Cuando se va haciendo la construcción llega un punto en que en varios casos no se puede decidir si un hijo va a la izquierda o a la derecha.

2. La respuesta correcta es la 2.c

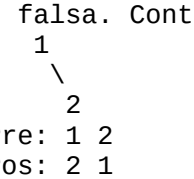
Dos son falsas (a y b) y una cierta (la c). La (b) es falsa porque cara a buscar los borrados se han de tratar como ocupados. La (a) es falsa. Un contraejemplo podría ser:

Es un AVL y el mayor valor (el 12) no se sitúa en una hoja

3.-La respuesta correcta es la 3.b

Dos son ciertas (la c y la d) y dos falsas (la a y la b).

- La a es falsa. Contraejemplo:



son 2 árboles binarios diferentes con etiquetas no todas iguales

- La b es falsa. El preorden junto con la propiedad de ABB permite reconstruirlo de forma unívova

- La c es cierta. Un recorrido y el hecho de que las hojas estén empujadas a la izquierda permite reconstruirlo de forma unívoca

- La d es cierta. Por la misma razon que antes. Con un recorrido (en este caso el preorden) y el hecho de que las hojas estén empujadas a la izquierda, permite reconstruirlo de forma unívoca.

```
#include <iostream>
#include <vector>
#include <list>
#include <cassert>

using namespace std;

class matriz_sesgada{
  private:
      vector<list<int> > datos;

  public:

    //para poder pasar una matriz constante. se puede hacer de otra
    //forma pasando un vector e indicando el numero de filas y columnas
    template <int R,int C>
    matriz_sesgada(int (&dat)[R][C]){
       assert(R==C);
       int n=R;
       datos =vector<list<int> >(n);
       for (int i=0;i<n;i++){
          datos[i]=list<int>(dat[i],dat[i]+n);
       }
    }

    int &operator()(int i,int j){
       assert(i>=0 && i<datos.size() && j>=0 && j<datos.size());
       list<int>::iterator it=datos[i].begin();
       advance(it,j);
       return *it;
    }

    int numfilas(){
      return datos.size();
    }

    int numcols(){
       return datos[0].size();
    }


    class iterator{
       private:
        vector<list<int> >::iterator it_row,final;
        list<int>::iterator it_col;
       public:
          bool operator==(const iterator &i)const{
             return ((it_row==final && i.it_row==final) ||
                     (it_row==i.it_row && it_col==i.it_col));
          }
          bool operator!=(const iterator &i)const{
             return !(*this==i);
          }
          int & operator *(){
              return *it_col;
          }
          iterator & operator++(){
             do{
                ++it_col;
                if (it_col==it_row->end()){
                    ++it_row;
                    if (it_row!=final){
                          it_col=it_row->begin();
                    }
```

```
                }

            }while (it_row!=final && *it_col%2==1);


            return *this;

          }
          friend class matriz_sesgada;
    };
    iterator begin(){
      iterator i;
      i.it_row=datos.begin();
      i.final=datos.end();
      if (i.it_row!=datos.end())
        i.it_col=i.it_row->begin();
        if ((*(i.it_col))%2==1)++i;
      return i;

    }

    iterator end(){
      iterator i;
      i.it_row=datos.end();
      i.final=datos.end();

      return i;

    }
};
int main(){

   int m[6][6]={{7,9,4,6,1,3},
                {5,9,3,7,1,5},
                {1,3,7,3,3,1},
                {5,5,5,5,5,5},
                {3,1,1,2,9,9},
                {1,5,1,3,7,7}};
  matriz_sesgada mimatriz(m);
  cout<<"Salida usando el operator ()..."<<endl;
  for (int i=0;i<mimatriz.numfilas();i++){
        for (int j=0;j<mimatriz.numcols();j++){
          cout<<mimatriz(i,j)<<" ";
        }
        cout<<endl;
    }
  cout<<endl;
  cout<<"Salida de los pares..."<<endl;
  for (auto it = mimatriz.begin();it!=mimatriz.end();++it){
        cout<<*it<<" ";
  }


}
```

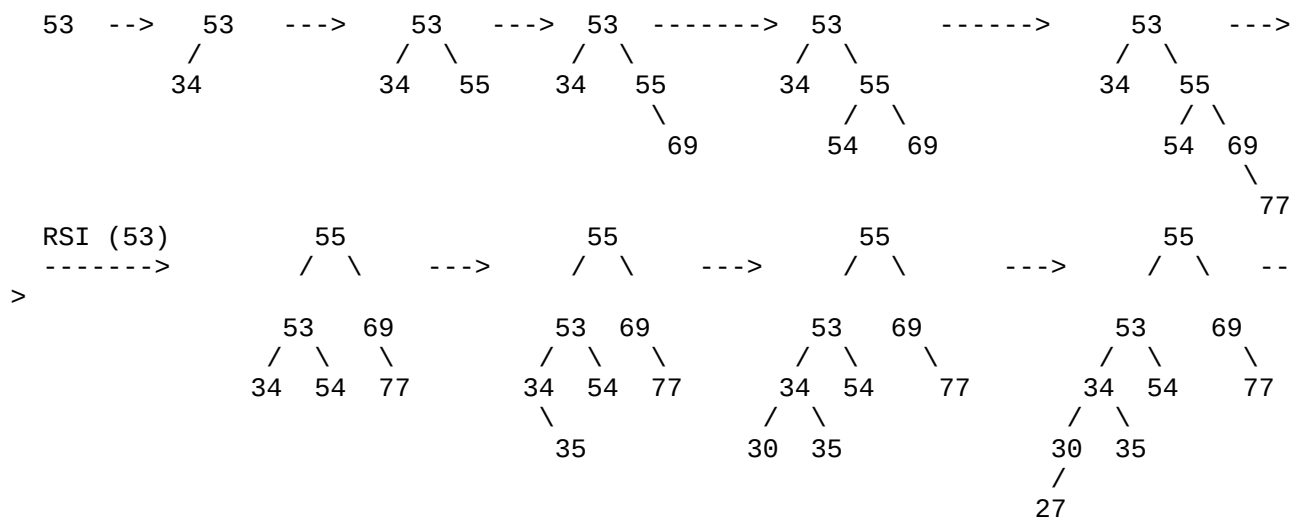
```
#include <iostream>
#include "bintree.h"
#include<queue>
#include<list>
#include <vector>
#include<algorithm>
#include <limits>

using namespace std;


//Esquema de un arbol
template <class T>
void Esquema(const bintree<T>& a,
             const typename bintree<T>::node n, string& pre){
  int i;

  if (n.null())
    cout << pre << "-- x" << endl;
  else {
    cout << pre << "-- " << *n << endl;
    if (n.right()!=0 || n.left()!=0) {// Si no es una hoja
      pre += "   |";
      Esquema(a, n.right(), pre);
      pre.replace(pre.size()-4, 4, "    ");
      Esquema (a, n.left(), pre);
      pre.erase(pre.size()-4, 4);
    }
  }
}

void ImprimirCaminos(const vector<list<int> > &v){
 cout<<"{";
 for (int i=0;i<v.size();i++){
    cout<<"{";
    for (auto it=v[i].begin();it!=v[i].end();++it){
        cout<<*it<<",";

    }
    cout<<"}";
 }
 cout<<"}";

}
vector<list<int> >getCaminos(bintree<int>::node n,list<int> prefijo){
   if (n.null()){
        vector<list<int> > aux;
        aux.push_back(prefijo);
        return aux;

      }
  else{
      vector<list<int> >izq,der;
      if ((n.left().null()) && n.right().null()) {
        izq.push_back(prefijo);
        return izq;
      }

      if (!(n.left().null())){
        list<int> aux=prefijo;
        aux.push_back(*(n.left()));
        izq=getCaminos(n.left(),aux);

      }
```

```
      if (!(n.right().null())){
        list<int> aux=prefijo;
        aux.push_back(*(n.right()));
        der=getCaminos(n.right(),aux);
      }
      for (auto it=izq.begin();it!=izq.end();++it)
        der.push_back(*it);
      //ImprimirCaminos(der); cout<<endl;
      //cin.get();
      return der;

  }


}

//Devuelve la suma y si el camino es ascendente
pair<int,bool> sumalist(const list<int> &l){
  int s=0;
  bool res=true;
  for (auto it=l.begin();it!=l.end() && res;++it){
     s+=*it;
     auto it2=it; ++it2;
     if (it2!=l.end())
        if (*it2<*it) res=false;

  }
  return pair<int,bool>(s,res);


}

bool camino_especial(bintree<int>::node n,int k){
      if (n.null()){
       return false;
      }
      else{
        list<int> aux={*n};
        vector<list<int> >out=getCaminos( n,aux);
        auto it = out.begin();
        while (it!=out.end()){
              pair<int,bool> aux=sumalist(*it);
              if (aux.first==k && aux.second) return true;
              ++it;
        }
        return false;
      }

}
```

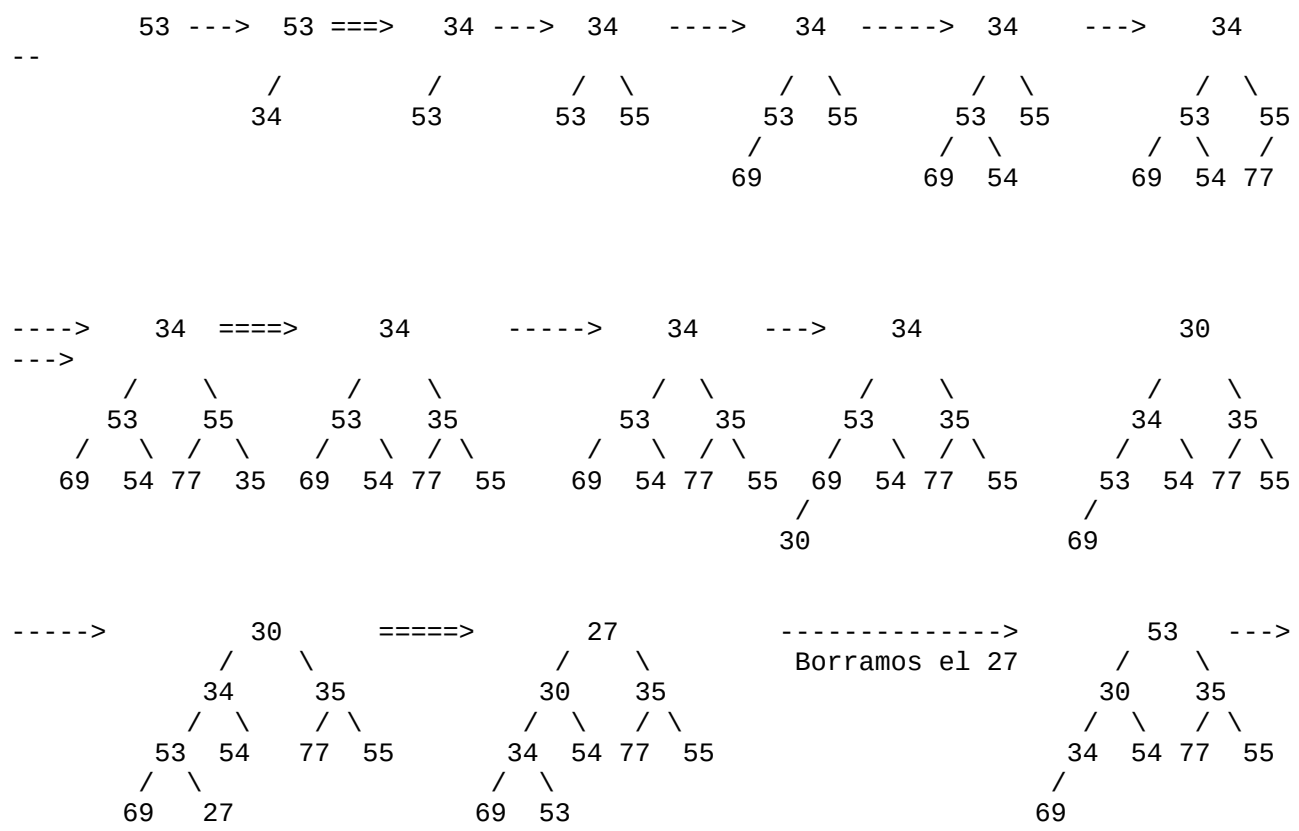
```
int main()
{
// Creamos el árbol Arb1:
  //        3
  //     /    \
  //    6      7
  //  /  \    /  \
  // 1    3  9   5

bintree<int> Arb1(3);
Arb1.insert_left(Arb1.root(), 6);
Arb1.insert_right(Arb1.root(), 7);
Arb1.insert_left(Arb1.root().left(), 1);
Arb1.insert_right(Arb1.root().left(), 3);
Arb1.insert_left(Arb1.root().right(), 9);
Arb1.insert_right(Arb1.root().right(),5);

string pre1= " ";
Esquema(Arb1,Arb1.root(),pre1);


if (camino_especial(Arb1.root(),19)){
    cout<<"Si existe un camino especial"<<endl;
 }
 else cout<<"No existe un camino especial"<<endl;



// Creamos el árbol Arb2:
  //        2
  //     /    \
  //    6      9
  //  /  \    /  \
  // 1    3  8   5



bintree<int> Arb2(2);
Arb2.insert_left(Arb2.root(), 6);
Arb2.insert_right(Arb2.root(), 9);
Arb2.insert_left(Arb2.root().left(), 1);
Arb2.insert_right(Arb2.root().left(), 3);
Arb2.insert_left(Arb2.root().right(), 8);
Arb2.insert_right(Arb2.root().right(),5);

string pre2= " ";
Esquema(Arb2,Arb2.root(),pre2);


if (camino_especial(Arb2.root(),19)){
    cout<<"Si existe un camino especial"<<endl;
 }
 else cout<<"No existe un camino especial"<<endl;



  return 0;



}
```

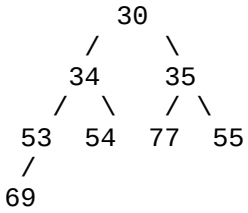
```
#include <set>
#include <vector>
#include <map>
#include <iostream>
using namespace std;

void contar(const vector<set<int> > &V, map<int,int> &veces){

      for (int i=0;i<V.size();i++){
        for (auto it=V[i].begin(); it!=V[i].end();++it){
             map<int,int>::iterator it_m;
             it_m = veces.find(*it);
             if (it_m==veces.end()){
                veces[*it]=1;

             }
             else veces[*it]++;

        }


      }
}

int main(){
  vector<set<int> >miv;
  set<int> s1={1,2};
  set<int> s2={1,2,3,5};
  set<int> s3={1,2,3,5,10,12};
  miv.push_back(s1);
  miv.push_back(s2);
  miv.push_back(s3);
  map<int,int>veces;
  contar(miv,veces);
  for (auto it=veces.begin();it!=veces.end();++it){
    cout<<it->first<<" frecuencia "<<it->second<<endl;

  }

}
```

AVL


53 --> 53 ---> 53 ---> 53 -------> 53 ------> 53 --->
34 34 55 34 55 34 55 34 55
69 54 69 54 69
77
RSI (53) -------> 55 ---> 55 ---> 55 ---> 55 -->
53 69 53 69 53 69 53 69
34 54 77 34 54 77 34 54 77 34 54 77
35 30 35 30 35
27
RSD (53) ---------> 55
34 69
30 53 77
27 35 54

APO


53 ---> 53 ===> 34 ---> 34 ----> 34 -----> 34 ---> 34
34 53 53 55 53 55 53 55 53 55
69 69 54 69 54 77
----> 34 ====> 34 -----> 34 ---> 34 30
53 55 53 35 53 35 53 35 34 35
69 54 77 35 69 54 77 55 69 54 77 55 69 54 77 55 53 54 77 55
30 69
-----> 30 =====> 27 --------------> 53 --->
34 35 30 35 Borramos el 27 30 35
53 54 77 55 34 54 77 55 34 54 77 55
69 27 69 53 69


30
34
35
53
54
77
55
69