UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Estructuras de datos. Curso 2018-2019
Convocatoria de Febrero
Grado en Ingeniería Informática.
Doble Grado en Ingeniería Informática y Matemáticas
Doble Grado en Ingeniería Informática y ADE

1. (1 punto) Implementar una función:

   void juntalista (list<int> &L, int n)

   que dada una lista L, agrupe los elementos de n en n dejando su suma

   P.ej si L={**1,3,2,**4,5,2,**2,3,5**,7,4,3,**2,2**} y **n=3** quedaría L={**6,**11,**10**,14,**4**}

   No pueden usarse estructuras auxiliares. Si n=0 devuelve la misma lista y si L está vacia, devuelve la lista vacia

2. (1 punto) En un hospital quieren implementar un sistema que permita que se pueda atender a usuarios considerando su gravedad. Además se debe poder hacer una búsqueda sobre los datos de cada paciente en función de su DNI o de su nombre completo. Indica una representación adecuada e implementa las operaciones:

   void urgencias::insertar_paciente(string dni, string nombre, string apellidos, int gravedad)

   void urgencias::cambiar_gravedad(string dni, int nueva_gravedad)

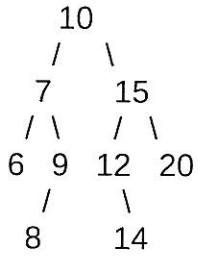
3. (1 punto) Implementar una función

   bool esABB (bintree <int> & A);

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

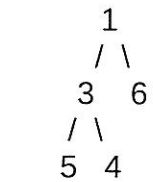
Ejemplos:

A



True

A



False

4. (1 punto) Implementar una función

   bool inall (list<set<int> > & LS, set<int> & S);

   que devuelva true si algún conjunto está incluido en todos los demás y tal conjunto lo devuelva en S

   P.ej: Si LS = [{1,2,3}, {2,3,4}] devuelve FALSE

   Si LS= [{**1,2,3**}, {**1,2,3**,4}, {**1,2,3**}] devuelve TRUE y S={**1,2,3**}

   Si LS= [{**1**,2,3}, {**1**}, {**1**,2}] devuelve TRUE y S={**1**}

5. (1 punto) Detalla cada una de las operaciones siguientes:
   a) Insertar las claves {5, 13, 17, 38, 7, 59, 24, 62, 10, 11} en una **Tabla Hash cerrada** de tamaño 13. A continuación borrar el 10 y el 38 y finalmente insertar el valor 48. Resolver las colisiones usando **rehashing doble**.
   b) Construir un **AVL** insertando, en ese orden, las siguientes claves {98, 27, 69, 80, 46, 37, 99, 20, 15, 48, 56}, especificando los pasos seguidos e indicando cuando sea necesario el tipo de rotación que se usa para equilibrar.
   c) Construir un **APO-min** realizando las siguientes tareas:
      - Insertar, en ese orden, las siguientes claves {11, 6, 13, 14, 7, 4, 10, 5, 8}
      - Borrar dos elementos.

6. (1 punto) Tenemos un contenedor de pares de elementos, {string set<int>} definido como:

```
class contenedor {
    private:
        map<string, set<int> > datos;
    ..............
    ..............
}
```

Implementar un **iterador** que itere sobre los elementos que cumplan la propiedad de que el conjunto asociado solo contenga elementos mayores que 20. Se debe implementar (aparte de las de la clase iteradora) las funciones begin() y end().

Tiempo: 3 horas

```cpp
#include <iostream>
#include <list>
using namespace std;
void Imprimir (list<int> & l){
    list<int>::iterator it;
   for (it=l.begin();it!=l.end();++it){
       cout<<*it<<" ";
   }
   cout<<endl;
}


void juntalista(list<int> &L,int n){
   if (n==0 || L.size()==0)
       return ;

   list<int>::iterator it=L.begin();

   while (it!=L.end()){
     list<int>::iterator inicio=it;
     int k=0;
     int cnt=0;
     while (it!=L.end() && k<n){
         cnt+=*it;
         ++it;
         k++;
     }
     it=L.erase(inicio,it);
     it=L.insert(it,cnt);
     ++it;
   }


}

int main(){
   int v[]={1,3,2,4,5,2,2,3,5,7,4,3,2,2};
   list<int>l;
   l.assign(v,v+14);

   juntalista(l,3);
   Imprimir(l);
}
```

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

struct paciente{
     string dni;
     string nombre;
     string apellidos;
     paciente( string d,string n,string a):dni(d),nombre(n),apellidos(a){}
     paciente(){}
};
ostream & operator<<(ostream & os,const paciente &p){
   os<<"Dni:"<<p.dni<<" Nombre: "<<p.nombre<<" Apellidos: "<<p.apellidos<<endl;
   return os;
}

/*bool operator<(const pair<string,int> & p1, const pair<string,int> & p2){
   return p1.first<p2.first;
}
bool operator<(const pair<int,string> & p1, const pair<int,string> & p2){
   return p1.first>p2.first;
}*/
bool operator<(const pair<string,string> & p1, const pair<string,string> & p2){
   return (p1.first<p2.first || (p1.first==p2.first && p1.second<p2.second));
}
bool operator<(const string & p1, const string & p2){
   return (p1<p2);
}



class urgencias{

private:
  map<string,paciente>pacientes; //datos de los pacientes indexados por dni
  multimap<pair<string,string>,string>indice_nomapellidos; //indexacion por nombreapellidos, dni
  multimap<int, string>indice_gravedad; //indexacion por gravedad->dni
  //Si se quiere hacer busquedas rapidas dado el dni del paciente obtener la gravedad
  //deberiamos tener un map<string,int> para acelerar las busquedas dni-->gravedad
  //map<string,int>indice_dni_gravedad;
public:
  void Insertar_Paciente(string dni,string nombre,string apellido,int gravedad){
     map<string,paciente>::iterator it;
     it=pacientes.find(dni);
     if (it==pacientes.end()){
        paciente nuevo (dni,nombre,apellido);
        pair<string,paciente> pn(dni,nuevo);
        pacientes.insert(pn);
        pair<string,string>na(nombre,apellido);
        pair<pair<string,string>,string>pnad(na,dni);
        indice_nomapellidos.insert(pnad);
        pair<int,string>pg(gravedad,dni);
        indice_gravedad.insert(pg);
        //pair<string,int>pdg(dni,gravedad);
        //indice_dni_gravedad.insert(pdg);
     }
  }

  void Cambiar_Gravedad(string dni,int nueva_gravedad){
     map<string,paciente>::iterator it;
     it=pacientes.find(dni);
     if (it!=pacientes.end()){//comprobar que existe el paciente
        //El codigo en el caso de que no tuvieramos indice_dni_gravedad seria:
        multimap<int, string>::iterator itg=indice_gravedad.begin();
        bool find=false;
        while (itg!=indice_gravedad.end() && !find){
           if (itg->second==dni)
              find=true;
           else
              ++itg;
        }
        if (find){
           indice_gravedad.erase(itg);
           pair<int,string> n(nueva_gravedad,dni);
           indice_gravedad.insert(n);

        }

  }
 }
  int getGravedad(string dni){
    map<string,paciente>::iterator it;
    it=pacientes.find(dni);
    if (it!=pacientes.end()){//comprobar que existe el paciente
       //El codigo en el caso de que no tuvieramos indice_dni_gravedad seria:
       multimap<int, string>::iterator itg=indice_gravedad.begin();
```

```
        bool find=false;
        while (itg!=indice_gravedad.end() && !find){
           if (itg->second==dni)
              find=true;
           else
              ++itg;
        }
        if (find)
           return itg->first;

      }
   }

  friend ostream & operator<<(ostream &os, urgencias &U){
     map<string,paciente>::const_iterator it = U.pacientes.begin();
     for(;it!=U.pacientes.end();++it){

        os<<it->second<<" Gravedad:"<<U.getGravedad(it->first) <<endl;

     }
     return os;
  }


};



int main(){
   paciente p1("2343223","Francisco","Lopez Rivas");
   paciente p2("3243227","Monica","Naranjo Limon");
   paciente p3("5555555","Felix","Gato Murillo");
   paciente p4("11111111","Beatriz","Leyva Ruiz");

   urgencias U;
   U.Insertar_Paciente(p1.dni,p1.nombre,p1.apellidos,0);
   U.Insertar_Paciente(p2.dni,p2.nombre,p2.apellidos,2);
   U.Insertar_Paciente(p3.dni,p3.nombre,p3.apellidos,3);
   U.Insertar_Paciente(p4.dni,p4.nombre,p4.apellidos,1);

   cout<<"Datos Originales:" <<endl;
   cout<<"Urgencias: "<<U<<endl;
   cout<<"_____"<<endl;



   U.Cambiar_Gravedad("11111111",5);
   cout<<"Tras cambiar la informacion del paciente con dni 11111111"<<endl;
   cout<<"Urgencias: "<<U<<endl;
}
```

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

     return esAbb(n.hi(),min,*n) && esAbb(n.hd(),*n,max);
   }
}

int main(){
 ArbolBinario<int> a;

 // ej:n5n3n2xxn4xxn8n7xxn9xx se corresponde con el arbol  es ABB

 // ej:n1n2n4xxn8xxn3n6xxn7xx se corresponde con el arbol no es ABB
 // 1
 // |_2
 // | |_4
 // | |_8
 // |_3
 //   |_6
 //   |_7

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

```cpp
#include <iostream>
#include <list>
#include <set>
using namespace std;

bool operator==(const set<int> &S1,const set<int> &S2){
   set<int>::iterator it1;

   for (it1=S1.begin();it1!=S1.end();++it1){
      set<int>::iterator it2=S2.find(*it1);
      if (it2==S2.end())
         return false;

   }
   return true;
}
void Imprimir(set<int > &S){

   for (auto i=S.begin();i!=S.end();++i)
      cout<<*i;
}

bool inall(list<set<int> > & LS,set<int> &S){
   list<set<int> >::iterator itL;

   for (itL=LS.begin();itL!=LS.end();++itL){

      list<set<int> >::iterator it2=LS.begin();
      bool contenido=true;
      while (it2!=LS.end() && contenido){

         if (!(*itL==*it2))
            contenido=false;
         else
            ++it2;
      }
      if (contenido){
          S=*itL;
          return true;
      }
   }
   return false;
}

int main(){
   int v[]={1,2,3};
   set<int>s1(v,v+3);
   int v2[]={1,2,3,4};
   set<int>s2(v2,v2+4);
   int v3[]={1,2,3,4,5};
   set<int>s3(v3,v3+4);
   list<set<int> > LS;
   LS.insert(LS.end(),s1);
   LS.insert(LS.end(),s2);
   LS.insert(LS.end(),s3);
   set<int>sout;

   if (  inall(LS,sout)){
      cout<<"El conjunto incluido en todos es "<<endl;
      Imprimir(sout);
      cout<<endl;
   }
   else
      cout<<"No existe ningun conjunto incluido en todos "<<endl;


}

// Otra opción sería:

bool inall(list<set<int> > & LS,set<int> &S){
   list<set<int> >::iterator itL=LS.begin();
   set<int>menor=*itL;
   ++itL;
   for (;itL!=LS.end();++itL){
      if ((*itL).size()<menor.size())
     menor= *itL;
   }


   for (itL=LS.begin();itL!=LS.end();++itL){
        if (!(menor==*itL))
        return false;
   }
   S=menor;
   return true;
}
```

5.9 {5, 13, 17, 38, 7, 59, 24, 62, 10, 11}

$h(K) = K \,\%\, 13$

$h_i(K) = [h_{i-1}(K) + h_0(K)] \,\%\, 13$ $\quad h_0(K) = 1 + K \,\%\, (M-2)$

$h_0(K) = 1 + K \,\%\, 11$

K	5	13	17	38	7	59	24	62	10	11	48
h(K)	5	0	4	12	7	7	11	10	10	11	9
$h_0(K)$	6	3	5	6	8	5	3	8	11	1	5

$h(5) = 5 \,\%\, 13 = 5$ $\quad h(13) = 13 \,\%\, 13 = 0$ $\quad h(17) = 17 \,\%\, 13 = 4$

$h(38) = 38 \,\%\, 13 = 12$ $\quad h(7) = 7 \,\%\, 13 = 7$ $\quad h(59) = 59 \,\%\, 13 = 7$ colisión

$h_2(59) = [h_1(59) + h_0(59)] \,\%\, 13 = (7 + 5) \,\%\, 13 = 12$ colisión

$h_3(59) = [h_2(59) + h_0(59)] \,\%\, 13 = (12 + 5) \,\%\, 13 = 4$ colisión

$h_4(59) = [h_3(59) + h_0(59)] \,\%\, 13 = (4 + 5) \,\%\, 13 = 9 \rightarrow h_4(59) = 9$

$h(24) = 24 \,\%\, 13 = 11$ $\quad h(62) = 62 \,\%\, 13 = 10$ $\quad h(10) = 10 \,\%\, 13 = 10$ colisión

$h_2(10) = [h_1(10) + h_0(10)] \,\%\, 13 = [10 + 11] \,\%\, 13 = 8 \rightarrow h_2(10) = 8$

$h(11) = 11 \,\%\, 13 = 11$ colisión

$h_2(11) = [h_1(11) + h_0(11)] \,\%\, 13 = [11 + 1] \,\%\, 13 = 12$ colisión

$h_3(11) = [h_2(11) + h_0(11)] \,\%\, 13 = [12 + 1] \,\%\, 13 = 0$ colisión

$h_4(11) = [h_3(11) + h_0(11)] \,\%\, 13 =$

$= [0 + 1] \,\%\, 13 = 1 \rightarrow h_4(11) = 1$

$h(48) = 48 \,\%\, 13 = 9$ colisión

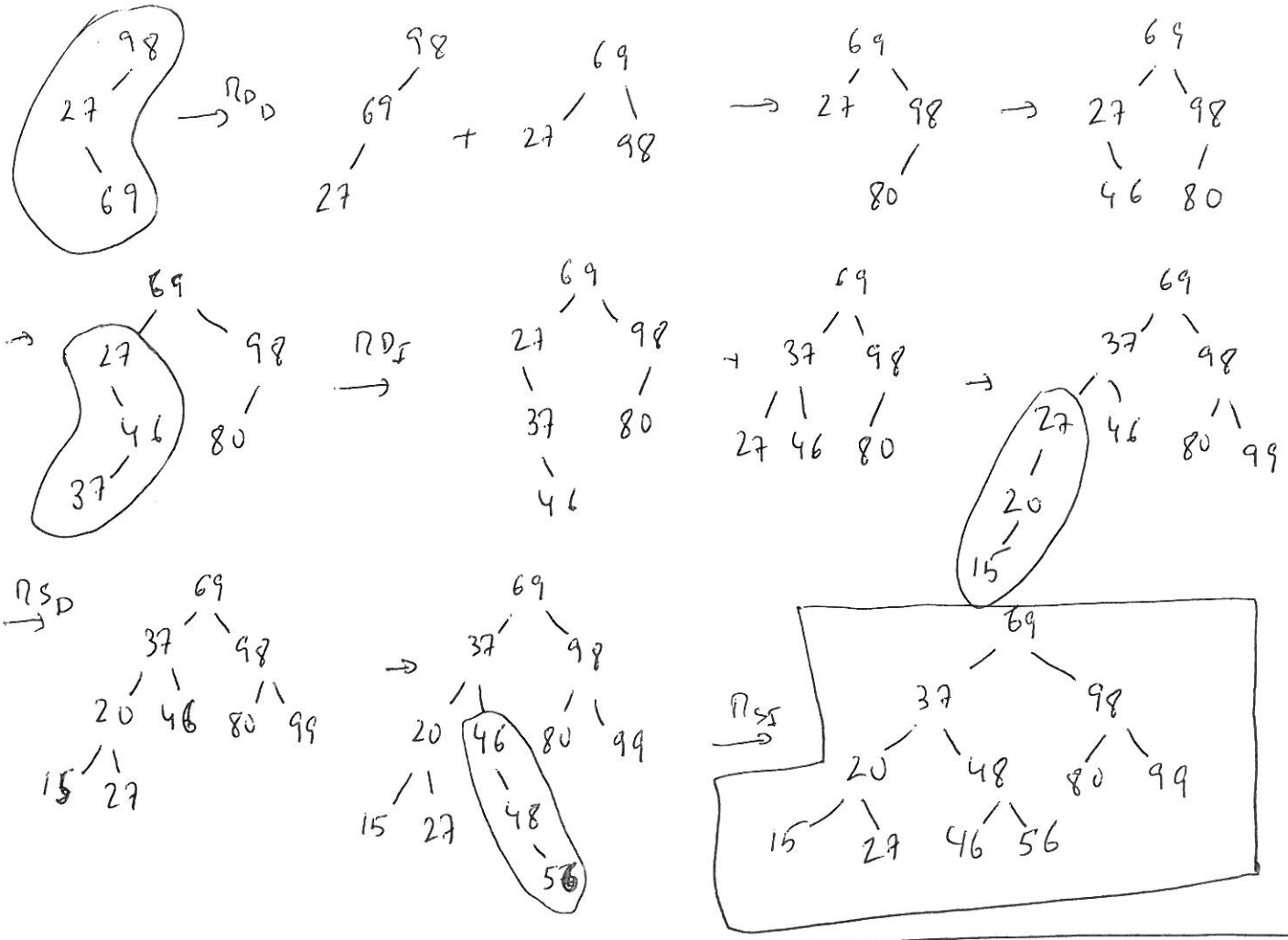
$h_2(48) = [h_1(48) + h_0(48)] \,\%\, 13 = [9 + 5] \,\%\, 13 = 1$ colisión

$h_3(48) = [h_2(48) + h_1(48)] \,\%\, 13 = [1 + 5] \,\%\, 13 = 6$

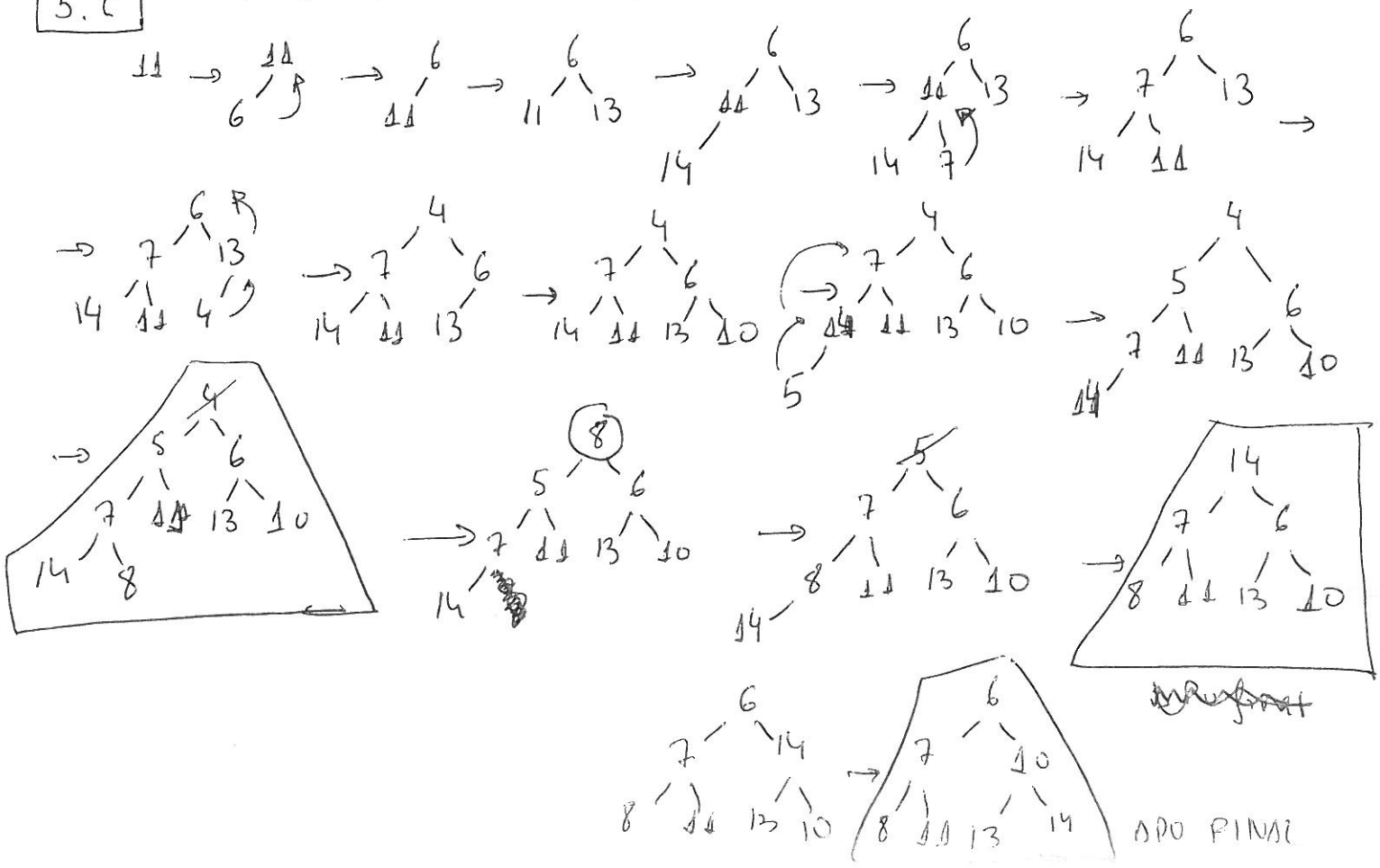
$h_3(48) = 6$

0	13
1	11
2	
3	
4	17
5	5
6	48
7	7
8	10
9	59
10	62
11	24
12	38

5.b {98, 27, 69, 80, 46, 37, 99, 20, 15, 48, 56}



5.c {11, 6, 13, 14, 7, 4, 10, 5, 8}



```
#include <map>
#include <set>
#include<iostream>
using namespace std;
//sin usar iteradores
void Imprimir (set<int> & l){
    set<int>::iterator it;
   for (it=l.begin();it!=l.end();++it){
       cout<<*it<<" ";
   }
   cout<<endl;
}
bool SoloMayores(const set<int> & l,int v){
   set<int>::const_iterator it;
   for (it=l.cbegin(); it!=l.cend();++it)
      if (*it<=v) return false;
   return true;
}



class contenedor{
  private:
   map <string, set<int> > datos;
  public:


    class iterator{
     private:
      map <string, set<int> >::iterator it;
      map <string, set<int> >::iterator f;
     public:
      iterator (){}

      bool operator==(const iterator &i)const{
         return i.it==it;
      }
      bool operator!=(const iterator &i)const{
         return i.it!=it;
      }
      pair<const string,set<int> > & operator*(){
         return *it;
      }
      iterator &operator++(){

         do{
            ++it;

         }while (it!=f && !SoloMayores((*it).second,20));
         return *this;
      }
      friend class contenedor;
   };

   iterator begin(){
      iterator i;
      i.it=datos.begin();
      i.f = datos.end();
      if (!SoloMayores((*(i.it)).second,20))
         ++i;
      return i;
   }
   iterator end(){
      iterator i;
      i.it=datos.end();
      i.f = datos.end();
      return i;
   }
};


int main(){
       contenedor c;
       contenedor::iterator i;
}
```