1.- Dado un vector de conjuntos **vector<set<int> > V,** implementa una función:

void esta_en_conjunto(vector<set<int> > &v,
map<int,list<int> > &recuento);

que devuelva a través del map **recuento** cada uno de los enteros que aparecen en algún conjunto y una lista de las posiciones del vector de conjuntos en las que se pueden encontrar.
Por ejemplo, si v=[{1,20,3}, {20,3,45}, {3,45,5,93}, {20,45,6,8}, {93}, {1,3,5}] entonces debe devolver:
recuento={<1,{0,5}>, <20, {0, 1,3}>, <3, {0,1,2,5}>, ... ,
<5,{2,5}>, ... }

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

void esta_en_conjunto(vector<set<int> > &v,
                      map<int,list<int> > &recuento){
 set<int>dif_elem; //para tener los enteros de forma única

   for (int i=0; i<v.size(); i++)
    for (auto it = v[i].begin(); it!=v[i].end(); ++it)
       dif_elem.insert(*it);
   //recorremos los diferentes elementos y contabilizamos los
  //conjuntos en los que se encuentran
for (auto it = dif_elem.begin(); it!=dif_elem.end(); ++it){
                recuento[*it]=list<int>();
                for (int i=0; i<v.size(); i++)
                     if (v[i].count(*it)>0)
                          recuento[*it].push_back(i);
   }
}
```

2.- Dada un vector de conjuntos **vector< set<int> > v,** implementar una función

bool subconjunto_propio (vector< set<int> > &v);

que devuelve true si los conjuntos de la lista L son subconjuntos propios en forma consecutiva. Es decir, si v = (A0,A1, ...,An−1), debe ocurrir que que Aj ⊂ Aj+1 para j = 0, ..., n − 2. (A ⊂ B indica inclusión propia, es decir A ⊆ B y A != B.)

Ejemplo:

Si v ={ { 1, 2, 3}; {1,2,3,4}; {1,2,3,4,6} } devolvería true;
Si v ={ { 1, 2, 3}; {1,2,3,4}; {1,3,4,6} } devolvería false;

```
#include <set>
#include <vector>

using namespace std;

bool incluido_en(const set<int> &A,const set<int>&B){
    if (A.size()>=B.size()) return false;
    for (auto x :A)
        if (B.find(x)==B.end())
          return false;
    return true;
}

bool subconjunto_propio (vector< set<int> > &V){
     for (int i=0; i<V.size()-1; i++)
        if (!incluido_en(V[i],V[i+1]))
          return false;
     return true;
}
```

3.- Implementar una función
bool contiene_parejas(vector< set<int> > &sw, int n);

que devuelva true si cada par de enteros (j; k) con 0 <= j < k; k < n está contenido en al menos uno de los conjuntos en sw[].

P.ej si: sw[0] = {0; 1; 2; 3; 4}; sw[1] = {0; 1; 5; 6; 7}; sw[2] = {2; 3; 4; 5; 6; 7} contieneparejas(sw,8) debe devolver true.

Por otra parte si tenemos sw[0] = {0; 2; 3; 4}; sw[1] = {0; 1; 5; 7}; sw[2] = {2; 3; 5; 6; 7} entonces los pares (0; 6), (1; 2), (1; 3), (1; 4), (1; 6), (4; 5), (4; 6) y (4; 7) no están en ningún conjunto y contieneparejas(sw,8) debe devolver false.

```
#include <set>
#include <vector>

using namespace std;

bool contiene_parejas(vector< set<int> > &sw, int n){
    for (int i=0; i<n-1; i++)
     for (int j=i+1; j<n; j++){
      bool enc=false;
      for (auto it=sw.begin(); it!=sw.end() && !enc; ++it){
        if (it->find(i)!=it->end() && it->find(j)!=it->end())
            enc=true;
      }
      if (!enc) return false;
    }
    return true;
}
```

4.- Implementar una función
void corta_map(map<int, list<int> > &M, int p, int q);

que elimine todas las claves y todos los elementos de las listas asociadas que no están en el rango [p,q). Aunque una clave no se elimine por estar en el rango, también se deben eliminar los elementos de la lista asociada que no están en el rango, con la particularidad de que si la lista quedara vacía entonces la clave también debe ser eliminada. **No se pueden usar contenedores auxiliares.**
Por ejemplo:
Si M={1->(2,3,4),
5->(6,7,8),
8->(4,5),
3→(1,3,7)},
entonces corta_map(M,1,6) debe dejar
M={1->(2,3,4),
3->(1,3)}.
La clave 5 ha sido eliminada aunque está dentro del rango porque su lista quedaría vacía.

```
#include <map>
#include <list>

using namespace std;

void LimpiaLista(list<int> &L, int p,int q){
    auto it = L.begin();
    while (it!=L.end()){
        if (!(*it>=p && *it<q))
            it=L.erase(it);
        else ++it;
    }
}
```

```
void corta_map(map<int, list<int> > &M, int p, int q){
    auto it = M.begin();
    while (it!=M.end()){
        if (!(it->first>=p && it->first<q)){
            auto it2=it;
            ++it;
            M.erase(it2);
        }
        else{
            LimpiaLista(it->second,p,q);
            if (it->second.empty()){
                auto it2=it;
                ++it;
                M.erase(it2);
            }
            else ++it;
        }
    }
}
```

5.- 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>}

```
#include <set>
#include <vector>
#include <map>

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]++;
       }
     }
}
```

6.- Implementar una función

bool en_todos(vector< set<int> > &V);

que devuelve true si existe al menos un elemento que pertenece a todos los conjuntos V[j]
Ejemplos:
V[0]={0,2,3,4,5}, V[1]={0,1,5,7}, V[2]={2,3,5,6,7} devuelve true (el 5 está en todos)
V[0]={0,2,3,4,5}, V[1]={0,1,7}, V[2]={2,3,5,6,7} devuelve false

```
#include <set>
#include <vector>

using namespace std;


set<int> interseccion(const set<int> & s1, const
                                  set<int> &s2){
    set<int>r;
    for (auto it=s1.begin(); it!=s1.end(); ++it)
        if (s2.find(*it)!=s2.end())
          r.insert(*it);
    return r;
}

bool en_todos(const vector< set<int> > &V){
   set<int>s_i(V[0]);
   for (int i=1; i<V.size(); i++){
       s_i=interseccion(s_i,V[i]);
       if (s_i.size()==0) return false;
   }
   return true;
}
```

7.- Implementar una función

int mas_conectado(vector< set<int> > &VS);

que devuelve el índice **j** tal que el conjunto **VS[j]** es el conjunto que está conectado con un mayor número de otros conjuntos de VS. Decimos que dos conjuntos están conectados si no son disjuntos, es decir, si tienen intersección común no vacía. En el caso de haber varios conjuntos con el mismo número de conexiones debe devolver el primero de ellos.

Ejemplos:

VS=[{0},{1},{2},{0,1,2}], devuelve 3

VS=[{0,1,2},{0},{1},{2}], devuelve 0

VS=[{0,6,9},{5,6,9},{5},{1},{5,9},{5},{1,5,7}], devuelve1

```
#include <vector>
#include <set>

using namespace std;

bool interseccion_isempty(const set<int> & s1,const
                                    set<int> &s2){
   set<int>::const_iterator it;
   for (it=s1.cbegin(); it!=s1.cend();++it){
         if (s2.count(*it)>0)
          return false;
   }
   return true;
}
```

```
int mas_conectado(vector< set<int> > &vs){

    vector<unsigned int> acumuladores(vs.size(),0);

    for (unsigned int i=0; i<vs.size()-1; ++i){
            for (unsigned int j=i+1; j<vs.size(); ++j){
                  if (!interseccion_isempty(vs[i],vs[j])){
                    acumuladores[i]++;
                    acumuladores[j]++;
                  }
            }
    }
    int pos_best=-1;
    int conect_best=-1;
    for (unsigned int i=0;i<acumuladores.size();++i){
            if (conect_best<(int)acumuladores[i]){
                  pos_best=i;
                  conect_best=acumuladores[i];
          }
    }
    return pos_best;
}
```

8.- Implementar una función:

void solo_en_2(vector<set<int> > &VS,set<int> &S1);

que dado un vector de conjuntos enteros VS, encuentre el conjunto S1 de todos aquellos elementos que están exactamente en dos de ellos.
Por ejemplo,si VS=[{**0**,1,**2**,3}, {1,3,**4**,5}, {1,3,6,7}, {**2**,**4**,7,**9**}, {**0**,7,8,**9**}], entonces S1={**0,2,4,9**}

```
#include <vector>
#include <set>
#include <map>
using namespace std;
void solo_en_2(vector<set<int> > &VS,set<int> &S1){
//map con los elementos diferentes y sus frecuencias
   map<int,int> freq;
   for (int i=0; i<VS.size(); i++){
      for (auto it=VS[i].begin(); it!=VS[i].end(); ++it){
         auto pos= freq.find(*it);
         if (pos!=freq.end())
            pos->second++;
         else
            freq.emplace(*it,1);
      }
   }
 for (auto it = freq.begin();it!=freq.end();++it) //frec  2
      if (it->second==2)
         S1.insert(it->first);
}
```

9.- Dados dos map, M1 y M2, definidos como:

map<string,int> M1,M2;

con el primer campo representando el nombre de una **persona** (string) y el segundo campo su **numero de seguidores** (int) en una red social, **implementar una función:**

map<string,int> Union (const map<string,int> &M1, const map<string,int> &M2);

que obtenga el map correspondiente a la unión de los dos map de entrada, en el que el numero de seguidores será la suma de los seguidores en M1 y los seguidores en M2 para la misma persona que aparece en M1 y M2. En el caso que solamente aparezca en uno de los dos se queda tal cual en el map resultado

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

 map<string,int>  Union(map<string,int> &M1,
                                map<string,int> &M2){
     map<string,int>::iterator it2;
     map<string,int>::iterator it;
     map<string,int> M3(M1); //construimos M3 como
                                 //copia de M1
     for (it2=M2.begin();it2!=M2.end();it2++){
        it = M3.find(it2->first);
        if (it!=M3.end())
           it->second += it2->second;
       else
           M3.insert(*it2);
     }
     return M3;
}
```

10.- Implementar una función:

bool tiene_suma_constante (set<int> & s, int M);

que dado un conjunto de enteros **s** y un entero **M**, devuelva true si existe un subconjunto de elementos enteros de s cuyos valores sumen exactamente M. P.ej si. s={**1,2,3,4,5**,6};

tiene_suma_constante(s,**15**) devolvería **true**

tiene_suma_constante(s,**22**) devolvería **false**

Una idea: Si hay un elemento en S cuyo valor es M, fin. Si todos los elementos de S tienen valores mayores que M, fin. Si no, probar para cada uno de los elementos x de S con valor x < M para ver si S-{x} tiene un subconjunto S' con suma M-x. Si esto ocurre, entonces S'+ {x} tiene suma M.

Puede resolverse fácilmente de forma recursiva:

```
#include <set>
bool tiene_suma_constante (set<int> s, int M){
 if (M==0) return true;
  else{
    if (s.size()==0 ) return false;
    for (auto it = s.begin(); it!=s.end(); ++it){
        if (*it<=M){
            set<int> aux = s;
            aux.erase(*it);
            if (tiene_suma_constante(aux,M-*it)) return true;
                    }
}
return false;
    }
}
```