# Ejercicios con Set y Map

En los siguientes ejercicios que se proponen se quiere trabajar con set (multiset) y/o map (multimap) pero puede aparecer Tdas como vector o list

1.-Implementar una función:

```
bool buscaparejas(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 conjunto 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}
  buscaparejas(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 buscaparejas(sw,8) debe devolver false.

2.- Dada una lista de conjuntos de caracteres list<set<char> > V, implementar la siguiente función:

```
void contabilizarChar(const list<set<char> > &V, map<char,int> &veces);
```

que devuelve, a través del map ***veces***, el número de conjuntos en los que aparece un carácter.

P.ej si:

- V={ <a,b,c>, <a,d>, <d,l,n>} el mapa veces contendría
  veces{a:2,b:1,c:1,d:2,l:1,n:1}

3.-Implementar una función

```
bool contiene_unvalor_in_k_set(list<set<int> > &sw, int n, int k);
```

que devuelva true si el entero n está contenido en al menos k conjuntos de sw.

4.- Implementar una función

```
set<int> in_only1(const list<set<int> > & conjuntos)
```

que dado una serie de conjuntos de enteros obtiene el conjunto de enteros tal que cada entero aparece solamente en uno de los conjuntos. Se puede usar para resolver un mapa.

5.- Implementar una función

```
map<int,list<char>> cambiaClave(const map<char,int> & m)
```

que dado un mapa con clave char e información asociada un entero. Obtiene un map ahora siendo la clave el entero y la información asociada todos los caracteres con el mismo entero.
P.e
m={<a,3>,<b,9>,<c,3>,<d,9>} el mapa de salida seria out={<3,<a,c>>,<9,<b,d>>}

6.- Implementar una función

```
multimap<string,string>      BestFriend(const   multimap<string,pair<string,int>   >   &
amigos,const string &nombre,int k)
```

que dado un mapa con clave el nombre de una persona se almacena el nivel de afinidad (int) con otra persona (string). Se pide dado un nombre de una persona obtener los k personas más afines a ella. Si el numero de personas afines es menor que k se devuelven todas.

7.- Implementar una función

```
list<set<int> > conjuntosContenidos(const vector<set<int> > &v)
```

obtenga una lista de conjuntos (de mayor número) $s_1$ ,$s_2$ ,…,$s_n$ tal que $s_1$ C $s_2$ .... C $s_n$
P.ej
v={<1,2>,<3,4>,<1,2,3>,<3,4,9>,<1,2,3,4>} el resultado debería ser
out={<1,2>,<1,2,3>,<1,2,3,4>}

8.- Implementar una función

```
multiset<int> diferencia(const multiset<int> &mA, const multiset<int> &mB)
```

que obtiene la diferencia del multiset mA y mB. Asi si por ejemplo mA tiene 5 instancias del 4 y mB tiene 2 instancias del 4 el resultado tendrán 3 instancias del 2.

9.- Implementar una funcion

```
map<char,set<string>::const_iterator> getIndex(const set<string> &s)
```

que dado un conjunto de string obtenga un mapa que indexa la primera palabra en el set que se inicia con cada letra del abecedario. Si no existe una palabra para una letra del abecedario no se pone nada en el map.
P.e
si s={“amanecer”, “aurora”,”boleto”, “iglesia”,”infinito”, “montaña”,”niña,”niño”,”sal”, “sol”,”zigzag”}
en el mapa de salida tendriamos
out={‘a’:<iterador a la palabra amanecer>,’b’:<iterador a la palabra boleto>,’i’:<iterador a la palabra iglesia>,’m’:<iterador a la palabra montaña>,’n’:<iterador a la palabra niña>,’s’:<iterador a la palabra sal>,’z’:<iterador a la plabra zigzag>}

10.- Implementar una función

```
string PalabraMasLarga(const set<string> &s, const multiset<char> &letras)
```

que dado un conjunto de palabras s y un conjunto de letras obtenga la palabra de mayor longitud que se puede construir con esas letras y esta palabra esta en el conjunto s.