Curso de Programación en C++.
TPL2-2024. Trabajo Práctico de Laboratorio. [2024-05-29]

Ejercicios

[Ej. 1] **keepset**: [*Video explicativo:* https://youtu.be/PeTKzBR6iAA]
Escribir una funcion
`void keepset(list<set<int> > &LS, list<set<int> > &good,list<set<int> > &bad);`
que elimina de **LS** todos los conjuntos **s** de **LS**, tales que

- Tienen intersección vacía con todos los conjuntos de **good**.
- Tienen intersección no vacía con al menos un conjunto de **bad**.

EJEMPLOS:

```
(1) LS:[[1,2,3,4,5],[1,3,5],[2,4]],bad:[[2,4]],good:[[1,3,5]] => LS:[[1,3,5]]
(2) LS:[[1,2,3,4,5],[1,3,5],[2,4]],bad:[[1,3,5]],good:[[2,4]] => LS:[[2,4]]
```

En el ejemplo (1) el primer conjunto `[1,2,3,4,5]` es eliminado ya que comparte los elementos 2,4 con uno de los conjuntos de **bad**. El conjunto `[2,4]` tampoco queda, ya que no tiene ninguno elemento en común con **good**. El conjunto `[1,3,5]` SI queda ya que no tiene ningún elemento en **bad** y varios en **good**.

RESTRICCION: Para las operaciones de conjuntos deben usarse las operaciones binarias `set_union`, `set_intersection`, `set_difference`. Es decir, NO deben recorrerse los conjuntos con iteradores.

AYUDA: Escribir una funcion predicado
`bool disjoint(set<int> &A,set<int> &B);` Que retorna verdadero si **A** y **B** son disjuntos, es decir si su interseccion es vacia, usando las operaciones binarias.

- Para cada **s** de **LS**, recorrer los conjuntos de **bad** hasta encontrar uno que no sea disjunto, en ese caso descartar ese conjunto.
- Luego recorrer todos los conjuntos de **good** y verificar que NO sea disjunto con al menos uno de ellos. Si es disjunto con todos ellos hay que eliminarlo de LS.
- Al recorrer los elementos de LS tener cuidado al borrarlos, si corresponde. Recordar al borrar que deben refrescar el iterador y recordar que deben eliminar el iterador O avanzar, no las dos cosas al mismo tiempo.

[Ej. 2] **equivlset**: [*Video explicativo:* https://youtu.be/fpi47ueTk0M]
Escribir una funcion
`bool equivlset(list<set<int> > &LSA,list<set<int> > &LSB);`
que determina si los elementos de todos los conjuntos de **LSA** coinciden con los de **LSB**, **con su multiplicidad**. Es decir si un entero **x** está en **n** elementos de **LSA**, entonces debe estar en la misma cantidad de elementos de **LSB**.

EJEMPLOS:

```
(1) LSA:[[1,2],[3,4,5],[1,4]],LSB:[[1,3,4],[5],[1,2,4]]      => true
(2) LSA:[[1,2],[1,3,4,5],[1,4]],LSB:[[1,3,4],[5],[1,2,4]]   => false
(3) LSA:[[1,2],[1,3,4,5],[1,4]],LSB:[[1,3,4],[1,5],[1,2,4]] => true
```

En el ejemplo (1) la función debe retornar **`true`** ya que el 1 está dos veces en **`LSA`** y en **`LSB`**, etc... En cambio el (2) debe retornar false ya que el 1 está 3 veces en **`LSA`** pero sólo 2 veces en **`LSB`**.

AYUDA:

- Usar dos listas auxiliares **`LA`**, **`LB`**. Poner todos los elementos de los conjuntos de **`LSA`** en **`LA`** y los de **`LSB`** en **`LB`**. Ordenar ambos conjuntos (usar el métodos **`sort()`** de la clase **`list<>`**) y comparar ambas listas (se puede usar el operador == para listas).

[Ej. 3] **equivlsetmap**: [*Video explicativo:* **`https://youtu.be/fpi47ueTk0M`**]

Escribir una funcion

`bool equivlsetmap(list<set<int> > &LSA,list<set<int> > &LSB);`

que resuelve el mismo problema que el ejercicio anterior, pero usando un **`map<>`** que cuenta la frequencia de aparición de cada uno de los elementos.

AYUDA: Construir un **`map<int,int> MA`** tal que **`MA[x]`** es la cantidad de conjuntos de **`LSA`** que contienen a **`x`**. Por ejemplo

```
LSA:[[1,2],[3,4,5],[1,4]],LSB:[[1,3,4],[5],[1,2,4]]      => true
```

En este caso debe dar **`MA={1->2,2->1,3->1,4->2,5->1}`**.
Ya que el elemento 1 aparece en los conjuntos **`[1,2]`**, y **`[1,4]`**.

De la misma manera consruir **`MB`** para **`LSB`**. Luego comparar directamete los dos maps **`MA`** y **`MB`**.