F002B. Vectors comprimits — P16175_ca

A vegades cal manipular vectors numèrics que tenen un 0 a la majoria de posicions. En aquests casos, es pot estalviar memòria i temps de càlcul usant la tècnica dels *vectors comprimits*, que consisteix a guardar només els valors diferents de 0, juntament amb la posició on es troben.

Per exemple, per representar el vector

$$v = (0, 3, 0, 0, 8, 0, 0, -3, 5, 0, 0, 0, 0)$$

s'utilitza el vector comprimit amb quatre parells següent:

0	1	2	3
3;1	8;4	−3;7	5;8

Aquest vector comprimit indica que hi ha un 3 a la posició 1 del vector v, un 8 a la posició 4, un -3 a la posició 7, un 5 a la posició 8, i que a la resta de posicions hi ha un 0.

Fixeu-vos que en els vectors comprimits *només* es guarden les posicions que tenen un valor diferent de 0, i que la taula es troba ordenada creixentment segons les posicions.

Les definicions següents permeten utilitzar vectors comprimits d'enters:

```
struct Parell {
    int valor;                  // Diferent de zero
    int pos;                    // Més gran o igual que zero
};

typedef vector<Parell> Vec_Com; // Ordenat per pos!
```

Utilitzant aquestes definicions, implementeu la funció

Vec_Com suma(**const** *Vec_Com*& *v1*, **const** *Vec_Com*& *v2*);

que retorna la suma, component a component, de dos vectors comprimits `v1` i `v2` donats.

Implementeu també l'acció

void *llegeix* (*Vec_Com*& *v*);

que llegeix, d'acord amb el format dels exemples, un vector comprimit, i el desa a `v`.

El programa principal ja se us dóna implementat; no el canvieu. Aquest llegeix primer un natural k. Després llegeix k parelles de vectors comprimits, els suma i n'escriu el resultat. L'acció que escriu vectors comprimits també se us dóna implementada; *queda estrictament prohibit cambiar-la*. Fixeu-vos que tant a l'entrada com a la sortida apareix explícitament el nombre de valors diferents de 0 del vector. Fixeu-vos també que la llargada que tenia el vector original (no comprimit) és irrellevant en aquest problema.

Observació

Useu algun mètode eficient per implementar `suma()`. Altrament, el Jutge rebutjarà la vostra solució per ser massa lenta. Inspireu-vos en algun dels algorismes fonamentals vistos a classe.

Exemple d'entrada

```
5

4 3;1 8;4 -3;7 5;8
4 3;1 8;4 -3;7 5;8

3 4;0 8;5 6;6
2 3;0 -6;6

3 2;3 3;18 5;21
3 -2;3 -3;18 -5;21

1 1;1000000000
1 1000000000;1

1 999;666
0
```

Exemple de sortida

```
4 6;1 16;4 -6;7 10;8
2 7;0 8;5
0
2 1000000000;1 1;1000000000
1 999;666
```

Informació del problema

Autor : Professorat de P1
Generació : 2013-09-02 15:08:50