## F002B. Compressed vectors P16175_en

Sometimes you have to use numeric vectors that contain a 0 in most positions. In this cases, you can save memory and time of computation using the *compressed vector* technique, which consists of storing only the values different than zero, together with the position where they appear.

For instance, to represent the vector

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

we use the next compressed vector with four pairs:

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 3;1 | 8;4 | −3;7 | 5;8 |

This compressed vector shows that there is a 3 in the position 1 of $v$, an 8 in the position 4, a $-3$ in the position 7, a 5 in the position 8 and that, in the rest of positions, there is a 0.

Notice that in compressed vectors we *only* store the positions whose value is different than 0, and the table is in increasing order according to the positions.

The next definitions allow us to use compressed vectors of integers:

```
struct Pair {
    int value;                  // Different than zero
    int pos;                    // Greater or equal than zero
};

typedef vector<Pair> Com_Vec;   // Sorted by pos!
```

Using these definitions, you must implement the function

*Com_Vec sum*(**const** *Com_Vec*& *v1*, **const** *Com_Vec*& *v2*);

that returns the sum, component by component, of two given compressed vectors `v1` and `v2`.

You must also implement the procedure

**void** *read*(*Com_Vec*& *v*);

that reads, according to the format of the instances, a compressed vector, and stores it in `v`.

The main program is already done; do not modify it. First, it reads a natural number $k$. Then, it reads $k$ compressed vector pairs, sums them, and prints the result. The procedure that prints compressed vectors is also already done; *Do not modify it*. Notice that in the input and the output there is the number of values different than zero of the vector. Notice also that the original vector length (not compressed) is irrelevant in this problem.

### Observation

Use some efficient method to implement `sum()`. Otherwise, the Judge will reject your solution for being too slow. Inspire yourself in one of the fundamental algorithms seen in class.

## Sample input

```
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
```

## Sample output

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

## Problem information

Author : Professorat de P1
Translator : Carlos Molina
Generation : 2023-07-14 17:44:13