**Problem "Catching fish"**

The main topics problem "Catching fish" are: data structures, double-linked lists, root optimization. Possible partial solutions are based on the use of modeling, segment trees.

The main difficulty in solving this problem is to find the company with which the event occurs. Let's describe the General approach to storing information about companies used in solving all subtasks, and then in the description of the solution of each subtask we will describe the data structure that allows you to solve this subtask.

We will store information about the company in the form of a doubly linked list. As new companies appear, we will assign global numbers to them, and for the company with the number *v*, we will store the value of the value next[*v*] – the number of the next company and the value of prev [v] is the number of the previous company along the river.

In case of bankruptcy of the company the corresponding element is removed from the list, and in case of separation this element is removed and the numbers of two companies are inserted in the corresponding place of the list.

Below is a fragment of the program in the language C++, which handles an event of type *e* under the assumption that the global company number *v* with which the event occurs was somehow derived from the query. The counter in the variable *t* is used to assign global numbers.

            if (e == 1) {

                if (prev[v] == -1) {

                    int u = next[v];

                    ans -= a[u] \* a[u];

                    a[u] += a[v];

                    ans += a[u] \* a[u];

                    prev[u] = -1;

                } else if (next[v] == -1) {

                    int w = prev[v];

                    ans -= a[w] \* a[w];

                    a[w] += a[v];

                    ans += a[w] \* a[w];

                    next[w] = -1;

                } else {

                    int w = prev[v];

                    int u = next[v];

                    ans -= a[u] \* a[u];

                    ans -= a[w] \* a[w];

                    a[w] += a[v] / 2;

                    a[u] += (a[v] + 1) / 2;

                    ans += a[w] \* a[w];

                    ans += a[u] \* a[u];

                    next[w] = u;

                    prev[u] = w;

                }

                a[v] = 0;

            } else { // e = 2

                int w = prev[v];

                int u = next[v];

                a[t] = a[v] / 2;

                a[t + 1] = (a[v] + 1) / 2;

                ans += a[t] \* a[t];

                ans += a[t + 1] \* a[t + 1];

                if (w != -1) {

                    next[w] = t;

                }

                next[t] = t + 1;

                next[t + 1] = u;

                prev[t] = w;

                prev[t + 1] = t;

                if (u != -1) {

                    prev[u] = t + 1;

                }

                t += 2;

            }

Now, to learn the answer, it remains to choose a data structure that allows you to search for the global number of the company by its serial number from the source of the river.

To solve subtask 1, it is enough to implement the actions described in the condition in any way. For example, you can simply store information about all companies in the array in the order in which they are located along the river and shift part of the array when changes occur. Or you can apply the described idea with a list and directly run the pointer each time from the beginning of the list to the corresponding position.

When solving subtask 2, it is already necessary to implement a two-linked list. At the same time, thanks to an additional condition, the companies with which events occur are not far from each other in the list. Therefore, by maintaining a pointer to the last company that the event happened to, you can quickly navigate along the list to the next company.

In subtask 3, when only bankruptcies occur with companies, no new companies appear, and the number of companies only decreases. Therefore, we can assume that all companies have global numbers equal to the original number along the river. When solving this subtask, you can use the segment tree data structure for the sum operation to find the original company number.

Let's create a tree of segments for the *sum operation* on *n* elements and assign a value equal to 1 to all elements. When deleting a company, we will set the corresponding element to 0 and update it in the segment tree. To find the *i*-th enterprise from the beginning of the river, we will go down the tree segments. Being in the next vertex, check: if the sum in the left subtree is greater than or equal to *i*, then go to the left son of the current vertex, otherwise-subtract from *i* the sum of the values in the left subtree and go to the right son. The leaf in which as a result of descent there will be a pointer, corresponds to the *i*-th company from the beginning of the river. Both operations are performed in O(log *n*), therefore, the overall complexity of the solution will be O((*n* + *k*) log *n*).

Note that the solution for subtask 3 does not solve subtask 1 and 2, so to get 80 points, you need to implement both the list idea for subtask 1 and 2 and the segment tree for subtask 3. For the convenience of participants who implement several solutions for different subtasks in their program, the number of the subtask is indicated in the input data.

In subtask 4, unlike deleting companies, adding new companies to the middle of the list cannot simply be done using the segment tree. Here, a data structure that supports the following operations can come to the rescue:

the receipt of the item number;

delete an item at a specified position;

add an item to a specified position.Есть две достаточно распространенные структуры, которые удовлетворяют этим требованиям.

In the first case, an approach based on the use of "root decomposition"is used. The essence of it is as follows. Let's divide the set of elements into consecutive segments of length *B*. Inside each of these segments we will solve a problem similar to the first subtask-to store a sequence of elements in an array, shifting the elements if necessary. If, in the process of adding elements, the length of the segment becomes greater than *2B*, then we divide it into two. We will also store these segments in the array, shifting the end of the array if necessary. With this implementation, each operation is performed in time O(*B* + (*n* + *k*) / *B*). Choosing B of the order of the square root of (*n + k*), we obtain the asymptotics O(*k*∙sqrt(*n* + *k*))

In the second case, the approach is used, which is to choose to store the Cartesian tree data by an implicit key. In this case, the operations of getting the item by number, deleting and inserting are performed on average for O(log *n*), the total time of the solution is obtained O(*k*∙log n).

Note also that in contrast to subtask 3, the solution to subproblem 4 solves all of the previous subtasks.