OJ 12532

Interval Product

Prof. Edson Alves – UnB/FCTE

Problema

It's normal to feel worried and tense the day before a programming contest. To relax, you went out for a drink with some friends in a nearby pub. To keep your mind sharp for the next day, you decided to play the following game. To start, your friends will give you a sequence of N integers $X_1, X_2, \ldots, X_N$. Then, there will be K rounds; at each round, your friends will issue a command, which can be:

- a change command, when your friends want to change one of the values in the sequence; or
- a product command, when your friends give you two values I, J and ask you if the product $X_I \times X_{I+1} \times \ldots \times X_{J-1} \times X_J$ is positive, negative or zero.

Problema

Since you are at a pub, it was decided that the penalty for a wrong answer is to drink a pint of beer. You are worried this could affect you negatively at the next day's contest, and you don't want to check if Ballmer's peak theory is correct. Fortunately, your friends gave you the right to use your notebook. Since you trust more your coding skills than your math, you decided to write a program to help you in the game.

Entrada e saída

Input

Each test case is described using several lines. The first line contains two integers N and K, indicating respectively the number of elements in the sequence and the number of rounds of the game $(1 \leq N, K \leq 10^5)$. The second line contains N integers X_i that represent the initial values of the sequence $(-100 \leq X_i \leq 100)$ for $i = 1, 2, \ldots, N)$. Each of the next K lines describes a command and starts with an uppercase letter that is either 'C' or 'P'. If the letter is 'C', the line describes a change command, and the letter is followed by two integers I and V indicating that X_I must receive the value V $(1 \leq I \leq N$ and $-100 \leq V \leq 100)$. If the letter is 'P', the line describes a product command, and the letter is followed by two integers I and J indicating that the product from X_I to X_J, inclusive must be calculated $(1 \leq I \leq J \leq N)$. Within each test case there is at least one product command.

Output

For each test case output a line with a string representing the result of all the product commands in the test case. The i-th character of the string represents the result of the i-th product command. If the result of the command is positive the character must be '`+`' (plus); if the result is negative the character must be '`-`' (minus); if the result is zero the character must be '`0`' (zero)

Exemplo de entradas e saídas

Sample Input

```
4 6
-2 6 0 -1
C 1 10
P 1 4
C 3 7
P 2 2
C 4 -5
P 1 4
5 9
1 5 -2 4 3
P 1 2
P 1 5
C 4 -5
P 1 5
P 4 5
C 3 0
P 1 5
C 4 -5
C 4 -5
```

Sample Output

```
0+-
+-+-0
```

Solução com complexidade $O(T(N+K)\log(N+K))$

- O problema pode ser resolvido adaptando-se uma árvore de Fenwick para manter o registro dos produtos
- É preciso, entretanto, tratar o zero à parte, uma vez que ele não é invertível na operação de multiplicação
- A cada elemento não-negativo, deve ser registrado apenas seu sinal (1 ou -1)
- Cada zero deve ser tratado em uma árvore à parte
- Assim,

$$RPQ(i,j) = \begin{cases} RPQ(j)/RPQ(i-j), & \text{se } RSQ(i,j) = 0 \\ 0, & \text{caso contrário} \end{cases}$$

onde $RSQ(i,j)$ se refere ao vetor z_k de zeros: $z_i = 1$ se $x_i = 0$; $z_i = 0$, caso contrário.

Solução com complexidade $O(T(N+K)\log(N+K))$

```cpp
#include <bits/stdc++.h>

using namespace std;

class BITree
{
private:
    int N;
    vector<int> ft, zs;

    int LSB(int n) { return n & -n; }

public:
    BITree(int n) : N(n), ft(N + 1, 1), zs(N + 1, 0) { }

    int RPQ(int i, int j) {
        auto p = RPQ(j) / RPQ(i - 1);
        auto z = RSQ(i, j);
        return z ? 0 : p;
    }
```

Solução com complexidade $O(T(N+K)\log(N+K))$

```cpp
    void update(int i, int v)
    {
        // Remove o elemento
        auto x = RPQ(i, i);
        x ? multiply(i, x/abs(x)) : add(i, -1);

        // Insere o novo elemento
        v ? multiply(i, v/abs(v)) : add(i, 1);
    }

private:
    int RPQ(int i)
    {
        int prod = 1;

        while (i)
        {
            prod *= ft[i];
            i -= LSB(i);
        }
```

Solução com complexidade $O(T(N+K)\log(N+K))$

```
        return prod;
    }

    int RSQ(int i, int j)
    {
        return RSQ(j) - RSQ(i - 1);
    }

    int RSQ(int i)
    {
        int sum = 0;

        while (i)
        {
            sum += zs[i];
            i -= LSB(i);
        }

        return sum;
    }
```

Solução com complexidade $O(T(N+K)\log(N+K))$

```
    void multiply(int i, int v)
    {
        while (i <= N)
        {
            ft[i] *= v;
            i += LSB(i);
        }
    }

    void add(int i, int v)
    {
        while (i <= N)
        {
            zs[i] += v;
            i += LSB(i);
        }
    }
};
```

Solução com complexidade $O(T(N+K)\log(N+K))$

```
struct Query {
    char c;
    int i, j;
};

string solve(BITree& ft, const vector<Query>& qs)
{
    ostringstream os;

    for (const auto& q : qs) {
        switch (q.c) {
        case 'C':
            ft.update(q.i, q.j);
            break;

        default:
            auto p = ft.RPQ(q.i, q.j);
            os << (p ? (p > 0 ? '+' : '-') : '0');
        }
    }
```

Solução com complexidade $O(T(N + K) \log(N + K))$

```
    return os.str();
}

int main()
{
    ios::sync_with_stdio(false);
    int N, K;

    while (cin >> N >> K)
    {
        BITree ft(N);

        for (int i = 1; i <= N; ++i)
        {
            int x;
            cin >> x;

            ft.update(i, x);
        }
```

Solução com complexidade $O(T(N+K)\log(N+K))$

```cpp
        vector<Query> qs;

        while (K--)
        {
            string c;
            int i, j;

            cin >> c >> i >> j;

            qs.push_back({ c[0], i, j });
        }

        auto ans = solve(ft, qs);

        cout << ans << '\n';
    }

    return 0;
}
```