SPOJ FENTREE

Fenwick Trees

Prof. Edson Alves – UnB/FGA

Problema

Mr. Fenwick has an array a with many integers, and his children love to do operations on the array with their father. The operations can be a query or an update.

For each query the children say two indices l and r, and their father answers back with the sum of the elements from indices l to r (both included).

When there is an update, the children say an index i and a value x, and Fenwick will add x to a_i (so the new value of a_i is $a_i + x$).

Because indexing the array from zero is too obscure for children, all indices start from 1. Fenwick is now too busy to play games, so he needs your help with a program that plays with his children for him, and he gave you an input/output specification.

Entrada e saída

Input

The first line of the input contains N $(1 \leq N \leq 10^6)$. The second line contains N integers a_i $(-10^9 \leq a_i \leq 10^9)$, the initial values of the array. The third line contains Q $(1 \leq Q \leq 3 \times 10^5)$, the number of operations that will be made. Each of the next Q lines contains an operation. Query operations are of the form "q l r" $(1 \leq l \leq r \leq N)$, while update operations are of the form "u i x" $(1 \leq i \leq N, -10^9 \leq x \leq 10^9)$.

Output

You have to print the answer for every query in a different line, in the same order of the input.

Exemplo de entradas e saídas

Sample Input

```
10
3 2 4 0 42 33 -1 -2 4 4
6
q 3 5
q 1 10
u 5 -2
q 3 5
u 6 7
q 4 7
```

Sample Output

```
46
89
44
79
```

Solução

- A solução *naive* consiste em percorrer cada intervalo a cada consulta, de modo que a complexidade seria igual a $O(QN)$, onde Q é o número de *queries* do tipo q
- Como $Q \leq 3 \times 10^5$ e $N \leq 10^6$, esta solução levaria ao TLE
- O uso de uma árvore de Fenwick permite responder cada uma das *queries* com complexidade $O(\log N)$
- A construção da árvore tem complexidade $O(N \log N)$, de modo que a solução teria complexidade $O((N + Q) \log N)$
- É preciso tomar cuidado com possíveis *overflows*, usando o tipo **`long long`** para armazenar as informações dos nós da árvore

Solução AC com complexidade $O((Q + N) \log N)$

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

using namespace std;
using ll = long long;

struct BITree {
    vector<ll> ts;
    size_t N;

    BITree(size_t n) : ts(n + 1, 0), N(n) {}
    ll LSB(ll n) { return n & (-n); }

    void add(size_t i, ll x)
    {
        while (i <= N)
        {
            ts[i] += x;
            i += LSB(i);
        }
    }
```

Solução AC com complexidade $O((Q + N) \log N)$

```
    ll RSQ(size_t i, size_t j)
    {
        return RSQ(j) - RSQ(i - 1);
    }

    ll RSQ(size_t k)
    {
        ll sum = 0;

        while (k)
        {
            sum += ts[k];
            k -= LSB(k);
        }

        return sum;
    }
};
```

Solução AC com complexidade $O((Q + N) \log N)$

```
int main()
{
    ios::sync_with_stdio(false);

    size_t N;
    cin >> N;

    BITree ft(N);

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

        ft.add(i, a);
    }

    int Q;
    cin >> Q;
```

Solução AC com complexidade $O((Q + N) \log N)$

```
    while (Q--)
    {
        string cmd;
        ll L, R;

        cin >> cmd >> L >> R;

        switch (cmd[0]) {
        case 'q':
            cout << ft.RSQ(L, R) << '\n';
            break;

        default:
            ft.add(L, R);
        }
    }

    return 0;
}
```