Codeforces Round #179

Problema A: *Greg and Array*

Prof. Edson Alves – UnB/FGA

Problema

Greg has an array $a = a_1, a_2, \ldots, a_n$ and m operations. Each operation looks as: $l_i, r_i, d_i, (1 \leq l_i \leq r_i \leq n)$. To apply operation i to the array means to increase all array elements with numbers $l_i, l_i + 1, \ldots, r_i$ by value d_i.

Greg wrote down k queries on a piece of paper. Each query has the following form: $x_i, y_i, (1 \leq x_i \leq y_i \leq m)$. That means that one should apply operations with numbers $x_i, x_i + 1, \ldots, y_i$ to the array.

Now Greg is wondering, what the array a will be after all the queries are executed. Help Greg.

Entrada e saída

Input

The first line contains integers $n, m, k (1 \leq n, m, k \leq 10^5)$. The second line contains n integers: $a_1, a_2, \ldots, a_n (0 \leq a_i \leq 10^5)$ – the initial array.

Next m lines contain operations, the operation number i is written as three integers: $l_i, r_i, d_i, (1 \leq l_i \leq r_i \leq n), (0 \leq d_i \leq 10^5)$.

Next k lines contain the queries, the query number i is written as two integers: $x_i, y_i, (1 \leq x_i \leq y_i \leq m)$.

The numbers in the lines are separated by single spaces.

Output

On a single line print n integers $a_1, a_2, \ldots, a_n$ – the array after executing all the queries. Separate the printed numbers by spaces.

Please, do not use the `%lld` specifier to read or write 64-bit integers in C++. It is preferred to use the cin, cout streams of the `%I64d` specifier.

Exemplo de entradas e saídas

Sample Input

```
3 3 3
1 2 3
1 2 1
1 3 2
2 3 4
1 2
1 3
2 3
```

```
1 1 1
1
1 1 1
1 1
```

Sample Output

```
9 18 17
```

```
2
```

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

- A solução tem três partes
- A primeira é acumular o número de vezes que cada operação deverá ser realizada
- Isto pode ser feito com uma árvore de Fenwick com suporte para *range update*
- Em seguida, deve-se acumular o impacto de cada operação no vetor original
- O número de vezes x que a operação i será aplicada pode ser feita com uma *point query* na árvore
- Novamente é necessária uma árvore de Fenwick com suporte para *range update*
- O intervalo $[L, R]$ deve ser atualizado com o valor dx
- Por fim, a cada posição do vetor i deve ser adicionado o valor y obtido pela *point query* do índice i da árvore

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

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

using namespace std;
using ll = long long;
using ii = pair<int, int>;

class BITree {
private:
    vector<ll> ts;
    size_t N;

public:
    BITree(size_t n) : ts(n + 1, 0), N(n) {}

    ll value_at(int i)
    {
        return RSQ(i);
    }
```

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

```
    void range_add(size_t i, size_t j, ll x)
    {
        add(i, x);
        add(j + 1, -x);
    }

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

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

        while (i >= 1) {
            sum += ts[i];
            i -= LSB(i);
        }

        return sum;
    }
```

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

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

struct Op
{
    int L, R;
    ll d;
};

vector<ll>
solve(int N, int M, const vector<int>& as, const vector<Op>& ops, const vector<ii>& qs)
{
    BITree op_tree(M);
```

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

```cpp
    for (const auto& q : qs)
        op_tree.range_add(q.first, q.second, 1);

    BITree ft(N);

    for (int i = 1; i <= M; ++i)
    {
        auto x = op_tree.value_at(i);
        ft.range_add(ops[i].L, ops[i].R, x * ops[i].d);
    }

    vector<ll> ans(N + 1);

    for (int i = 1; i <= N; ++i)
        ans[i] = as[i] + ft.value_at(i);

    return ans;
}
```

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

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

    int N, M, K;
    cin >> N >> M >> K;

    vector<int> as(N + 1);

    for (int i = 1; i <= N; ++i)
        cin >> as[i];

    vector<Op> ops(M + 1);

    for (int i = 1; i <= M; ++i) {
        int L, R, d;
        cin >> L >> R >> d;

        ops[i] = Op { L, R, d };
    }
```

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

```cpp
    vector<ii> qs(K);

    for (int i = 0; i < K; ++i)
        cin >> qs[i].first >> qs[i].second;

    auto ans = solve(N, M, as, ops, qs);

    for (size_t i = 1; i < ans.size(); ++i)
        cout << ans[i] << (i + 1 == ans.size() ? '\n' : ' ');

    return 0;
}
```