Estrutura de dados

Fenwick Tree

Todas as implementações abaixo são indexadas em 1 portanto **não deve ser feito a consulta do índice 1**

Range Sum Query em O(log n) e atualização com soma pontual

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

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

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

        while(i>=1){
            sum+=ts[i];
            i-=LSB(i);
        }
        return sum;
    }
public:
    BITree(size_t n) : ts(n+1, 0), N(n) {};

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

    void add(size_t i, const long long& x){
        if(i==0) return;
        while(i<=N){
            ts[i]+=x;
            i+=LSB(i);
        }
    }
};
```

Range update em O(log n) query pontual

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

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

                void add(size_t i, ll x){
                        while (i <= N) {
                                ts[i] += x;
                                i += LSB(i);
                        }
                }
                long long RSQ(int i){
                        long long sum = 0;

                        while(i>=1){
                                sum+=ts[i];
                                i-=LSB(i);
                        }
                        return sum;
                }
        public:
                BITree(size_t n) : ts(n+1, 0), N(n) {};

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

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

Query bidimencional em O(log n.m) update com soma pontual

```
class BITree2D{
private:
    size_t rows;
    size_t columns;
    vector<vector<long long>> ft;

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

    long long RSQ(int y, int x){
        long long sum = 0;

        for(int i=y; i>0; i-=LSB(i))
            for(int j=x; j>0; j-=LSB(j))
                sum+=ft[i][j];

        return sum;
    }
public:
    BITree2D(size_t y, size_t x) : ft(y+1, vector<long long>(x+1, 0)),
rows(y), columns(x) {}

    // Y e X sao as coordenadas do ponto superior
    // x e y sao as coordenadas do ponto inferior
    // RSQ vai retornar a soma do quadrado formado pelos pontos
    long long RSQ(int y, int x, int Y, int X){
        return RSQ(Y, X) - RSQ(Y, x-1) - RSQ(y-1, X) + RSQ(y-1, x-1);
    }

    void add(int y, int x, long long v){
        for(int i=y; i<=rows; i+=LSB(i))
            for(int j=x; j<=columns; j+=LSB(j))
                ft[i][j] += v;
    }

};
```

Range Product Query em O(log n) e update com multiplicação pontual

```
class BITree{
private:
    size_t N;
    vector<long long> ft;
    vector<int> sz; // 1 se o indice i for 0

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

    long long RPQ(int i){
        long long prod = 1;
        while(i){
            prod*=ft[i];
            i-=LSB(i);
        }
        return prod;
    }

    // Funções rsq para o vetor com zeros
    int RSQ(int i){
        int sum = 0;
        while(i){
            sum+=sz[i];
            i-=LSB(i);
        }
        return sum;
    }

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

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

    void add(int i, int k){
        while(i <= N){
            sz[i] += k;
            i+=LSB(i);
        }
    }

public:

    BITree(int n) : N(n), ft(n+1, 1), sz(n+1, 0) { }
```

```
    long long RPQ(int i, int j){
        long long p = RPQ(j)/RPQ(i-1);
        int z = RSQ(i, j);

        if(z) return 0;
        else return p;
    }

    void update(int i, const long long& k){
        if(k)
            multiply(i, k);
        else if (RSQ(i, i)==0)
            add(i, 1);
    }

};
```