

Estrutura de dados

Fenwick Tree

Todas as implementações abaixo são indexadas em 1 portanto **não deve ser feito a consulta do índice 0**

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

```
const int neutral = 0;
#define comp(a, b) ((a)+(b))

class BITree {
private:
    vector<int> ft;
public:
    BITree(int n) { ft.assign(n+1, 0); }
    int rsq(int i){
        int sum = neutral;
        for(; i; i-= (i&-i))
            sum = comp(sum, ft[i]);
        return sum;
    }
    int rsq(int i, int j){
        return rsq(j)-rsq(i-1);
    }
    void update(int i, int v){
        for(; i<(int) ft.size(); i+= (i&-i))
            ft[i] = comp(v, ft[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 bidimensional 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);
}

};
```

Segment Tree

Segment Tree 1D

Construção em $O(n)$. Queries e updates em $O(\log n)$. memória $O(n)$, Indexado em 0.

```
const int neutral = 0;
#define comp(a, b) ((a)+(b))

class SegTree{
    vi a; //using vi = vector<int>
    int n;
public:
    SegTree(vi st){
        int sz = int(st.size());
        for(n = 1; n<sz; n<=1 );
        a.assign(n<1, neutral);
        for(int i=0; i<sz; ++i) a[i+n] = st[i];
        for(int i=n+sz-1; i>1; --i)
            a[i>>1] = comp(a[i>>1], a[i]);
    }
    void update(int i, int x){
        a[i+=n] = x; //substitui
        for( i>=1; i ; i>=1)
            a[i] = comp(a[i<<1], a[1+(i<<1)]);
    }
    int query(int l, int r){
        int ans = neutral;
        for(l+=n, r+=n+1; l<r; l>=1, r>=1){
            if(l&1) ans=comp(ans, a[l++]);
            if(r&1) ans=comp(ans, a[--r]);
        }
        return ans;
    }
};
```