



Marcela Caram      Pedro Veloso      Victor MontMor

#include <confia>

## Contents

<b>1</b>	<b>Grafos</b>	<b>2</b>
1.1	Bfs . . . . .	2
1.2	Dfs . . . . .	2
1.3	Dijkstra . . . . .	2
<b>2</b>	<b>Matematica</b>	<b>2</b>
2.1	Fast Exponentiation . . . . .	2
2.2	Miller-rabin . . . . .	2
2.3	Sieve . . . . .	3
2.4	Sieve Linear . . . . .	3
<b>3</b>	<b>Strings</b>	<b>3</b>
3.1	Kmp . . . . .	3
3.2	Z Function . . . . .	4
<b>4</b>	<b>Template</b>	<b>4</b>
4.1	Template . . . . .	4
<b>5</b>	<b>Trees</b>	<b>4</b>
5.1	Segtree . . . . .	4

# 1 Grafos

## 1.1 Bfs

```
1 // BFS com informacoes adicionais sobre a distancia e o pai de cada
   vertice
2 // Complexidade: O(V + E), onde V eh o numero de vertices e E o numero de
   arestas
3 vector<vector<int>>> adj; // lista de adjacencia
4 int n, s; // n = numero de vertices, s = vertice inicial
5
6 vector<bool> used(n);
7 vector<int> d(n), p(n);
8
9 void bfs(int s) {
10     queue<int> q;
11     q.push(s);
12     used[s] = true;
13     d[s] = 0;
14     p[s] = -1;
15
16     while (!q.empty()) {
17         int v = q.front();
18         q.pop();
19         for (int u : adj[v]) {
20             if (!used[u]) {
21                 used[u] = true;
22                 q.push(u);
23                 d[u] = d[v] + 1;
24                 p[u] = v;
25             }
26         }
27     }
28 }
```

## 1.2 Dfs

```
1 // DFS para percorrer todos os vertices a partir do no inicial u
2 // Complexidade: O(V + E), onde V eh o numero de vertices e E o numero de
   arestas
3 int explore(int u)
4 {
5     visited[u] = true;
6     int c = 1;
7     for (int v : adj[u])
8         if (!visited[v])
9             c += explore(v);
10    return c;
11 }
```

## 1.3 Dijkstra

```
1 vector<vector<pair<int, int>>>> adj;
2 int n, s;
3
```

```
4 vector<int> d(n, LLINF);
5 vector<int> p(n, -1);
6 vector<bool> used(n);
7
8 //Complexidade: O((V + E)logV)
9 void dijkstra(int s) {
10     d[s] = 0;
11     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<
   int, int>>>> q;
12     q.push({0, s});
13     while (!q.empty()) {
14         int v = q.top().second;
15         q.pop();
16         if (used[v]) continue;
17         used[v] = true;
18         for (auto edge : adj[v]) {
19             int to = edge.first, len = edge.second;
20             if (d[v] + len < d[to]) {
21                 d[to] = d[v] + len;
22                 p[to] = v;
23                 q.push({d[to], to});
24             }
25         }
26     }
27 }
28
29 //Complexidade: O(V)
30 vector<int> restorePath(int v) {
31     vector<int> path;
32     for (int u = v; u != -1; u = p[u])
33         path.push_back(u);
34     reverse(path.begin(), path.end());
35     return path;
36 }
```

# 2 Matematica

## 2.1 Fast Exponentiation

```
1 const int mod = 1e9+7;
2 int fexp(int a, int b)
3 {
4     int ans = 1;
5     while (b)
6     {
7         if (b & 1)
8             ans = ans * a % mod;
9         a = a * a % mod;
10        b >>= 1;
11    }
12    return ans;
13 }
```

## 2.2 Miller-rabin

```

1 // Miller-Rabin
2 //
3 // Testa se n eh primo, n <= 3 * 10^18
4 //
5 // O(log(n)), considerando multiplicacao
6 // e exponenciacao constantes
7
8 ll mul(ll a, ll b, ll m) {
9     ll ret = a*b - ll((long double)1/m*a*b+0.5)*m;
10    return ret < 0 ? ret+m : ret;
11 }
12
13 ll pow(ll x, ll y, ll m) {
14     if (!y) return 1;
15     ll ans = pow(mul(x, x, m), y/2, m);
16     return y%2 ? mul(x, ans, m) : ans;
17 }
18
19 bool prime(ll n) {
20     if (n < 2) return 0;
21     if (n <= 3) return 1;
22     if (n % 2 == 0) return 0;
23     ll r = __builtin_ctzll(n - 1), d = n >> r;
24
25     // com esses primos, o teste funciona garantido para n <= 2^64
26     // funciona para n <= 3*10^24 com os primos ate 41
27     for (int a : {2, 325, 9375, 28178, 450775, 9780504, 795265022}) {
28         ll x = pow(a, d, n);
29         if (x == 1 or x == n - 1 or a % n == 0) continue;
30
31         for (int j = 0; j < r - 1; j++) {
32             x = mul(x, x, n);
33             if (x == n - 1) break;
34         }
35         if (x != n - 1) return 0;
36     }
37     return 1;
38 }

```

## 2.3 Sieve

```

1 bool notPrime[ms];
2 int primes[ms], qnt;
3
4 void sieve(int lim) {
5     primes[qnt++] = 1 // se o 1 for valido na questao
6     for(int i = 2; i < ms; i++) { //loop(i,2,ms)
7         if(notPrime[i]) continue;
8         primes[qnt++] = i;
9         for(int j = i + i; j < ms; j += i)
10             notPrime[j] = true;
11     }

```

## 2.4 Sieve Linear

```

1 const int N = 10000000;

```

```

2 vector<int> lp(N+1);
3 vector<int> pr;
4
5 for (int i=2; i <= N; ++i) {
6     if (lp[i] == 0) {
7         lp[i] = i;
8         pr.push_back(i);
9     }
10    for (int j = 0; i * pr[j] <= N; ++j) {
11        lp[i * pr[j]] = pr[j];
12        if (pr[j] == lp[i]) {
13            break;
14        }
15    }
16 }

```

## 3 Strings

### 3.1 Kmp

```

1 // pre() gera um vetor pi com o tamanho da string ne
2 // pi[i] = tamanho do maior prefixo de ne que eh sufixo de ne[0..i]
3 // Complexidade: O(n)
4 vector<int> pre(string ne)
5 {
6     int n = ne.size();
7     vector<int> pi(n, 0);
8     for (int i = 1, j = 0; i < n; i++)
9     {
10         while (j > 0 && ne[i] != ne[j]) j = pi[j - 1];
11         if (ne[i] == ne[j]) j++;
12         pi[i] = j;
13     }
14     return pi;
15 }
16 // search() retorna o numero de ocorrencias de ne em hay
17 // complexidade: O(n+m)
18 int search(string hay, string ne)
19 {
20     vector<int> pi = pre(ne);
21     int c = 0;
22     for (int i = 0, j = 0; i < hay.size(); i++)
23     {
24         while (j > 0 && hay[i] != ne[j]) j = pi[j - 1];
25         if (hay[i] == ne[j]) j++;
26         if (j == ne.size())
27         {
28             c++;
29             // match at (i-j+1)
30             j = pi[j - 1];
31         }
32     }
33     return c;
34 }

```

## 3.2 Z Function

```
1 // a funcao z gera um vetor z com o tamanho da string s
2 //z[i] = tamanho do maior prefixo de s que eh sufixo de s[i..n-1]
3 //Complexidade: O(n)
4
5 vector<int> z_function(string s) {
6     int n = (int) s.length();
7     vector<int> z(n);
8     for (int i = 1, l = 0, r = 0; i < n; ++i) {
9         if (i <= r)
10             z[i] = min(r - i + 1, z[i - l]);
11         while (i + z[i] < n && s[z[i]] == s[i + z[i]])
12             ++z[i];
13         if (i + z[i] - 1 > r)
14             l = i, r = i + z[i] - 1;
15     }
16     return z;
17 }
```

## 4 Template

### 4.1 Template

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 //alias comp='g++ -std=c++17 -g -O2 -Wall -Wconversion -Wshadow -fsanitize=
4     =address,undefined -fno-sanitize-recover -ggdb -o out'
5 #define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
6 //Melhora o desempenho
7 #define int long long //Melhor linha de codigo ja escrita
8 #define endl "\n" //Evita flush
9 #define loop(i,a,n) for(int i=a; i < n; i++)
10 #define input(x) for (auto &it : x) cin >> it
11 #define pb push_back
12 #define all(x) x.begin(), x.end()
13 #define ff first
14 #define ss second
15 #define mp make_pair
16 #define TETO(a, b) ((a) + (b-1))/(b)
17 #define dbg(x) cout << #x << " = " << x << endl
18 #define print(x,y) loop(it,0,y){cout << x[it] << " ";} cout << "\n";
19
20 typedef long long ll;
21 typedef long double ld;
22 typedef vector<int> vi;
23 typedef pair<int,int> pii;
24 typedef priority_queue<int, vector<int>, greater<int>> pqi;
25
26 const ll MOD = 1e9+7;
27 const int MAX = 1e4+5;
28 const ll LLINF = 0x3f3f3f3f3f3f3f3f; //escrevemos 3f 8 vezes
```

```
28 const double PI = acos(-1);
29
30 int32_t main(){ sws;
31
32     return 0;
33 }
34 }
```

## 5 Trees

### 5.1 Segtree

```
1 int v[MAXN]; // input array
2 Tnode seg[4 * MAXN]; // segment tree
3
4 Tnode combine(Tnode left, Tnode right) {
5     // definir como combinar dois nós da árvore
6 }
7
8 Tnode build(int p, int l, int r) { // O(n)
9     if (l == r) return seg[p] = {v[l], {l, r}};
10    int m = (l + r) / 2;
11    Tnode left = build(p * 2, l, m);
12    Tnode right = build(p * 2 + 1, m + 1, r);
13    seg[p] = combine(left, right);
14 }
15
16 Tnode update(int i, int x, int p, int l, int r) { // O(log n)
17    if (i < l || r < i) return seg[p];
18    if (l == r) {
19        seg[p] = ...; // definir o que retornar quando l == r == i
20        return seg[p];
21    }
22    int m = (l + r) / 2;
23    Tnode left = update(i, x, p * 2, l, m);
24    Tnode right = update(i, x, p * 2 + 1, m + 1, r);
25    return seg[p] = combine(left, right);
26 }
27
28 Tnode query(int ql, int qr, int p, int l, int r) { // O(log n)
29    if (qr < l || r < ql) {
30        return ...; // definir o que retornar quando não há interseção
31    }
32    if (ql <= l && r <= qr) {
33        return seg[p];
34    }
35    int m = (l + r) / 2;
36    Tnode left = query(ql, qr, p * 2, l, m);
37    Tnode right = query(ql, qr, p * 2 + 1, m + 1, r);
38    return combine(left, right);
39 }
40
41 }
```