Marcela Caram Pedro Veloso Victor MontMor

#include <confia>

Contents

1 Grafos

1.1 Bfs

```cpp
// BFS com informacoes adicionais sobre a distancia e o pai de cada vertice
// Complexidade: O(V + E), onde V eh o numero de vertices e E o numero de arestas
vector<vector<int>> adj; // lista de adjacencia
int n, s; // n = numero de vertices, s = vertice inicial

vector<bool> used(n);
vector<int> d(n), p(n);

void bfs(int s) {
    queue<int> q;
    q.push(s);
    used[s] = true;
    d[s] = 0;
    p[s] = -1;

    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (int u : adj[v]) {
            if (!used[u]) {
                used[u] = true;
                q.push(u);
                d[u] = d[v] + 1;
                p[u] = v;
            }
        }
    }
}
```

1.2 Dfs

```cpp
// DFS para percorrer todos os vertices a partir do no inicial u
// Complexidade: O(V + E), onde V eh o numero de vertices e E o numero de arestas
vector<bool> visited;

void dfs(int u, vector<vector<int>> &adj) {
    visited[u] = true;
    for (int v : adj[u]) {
        if (!visited[v]) {
            dfs(v, adj);
        }
    }
}
```

1.3 Dijkstra

```cpp
vector<vector<pair<int, int>>> adj;
int n, s;

vector<int> d(n, LLINF);
vector<int> p(n, -1);
vector<bool> used(n);

//Complexidade: O((V + E)logV)
void dijkstra(int s) {
    d[s] = 0;
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> q;
    q.push({0, s});
    while (!q.empty()) {
        int v = q.top().second;
        q.pop();
        if (used[v]) continue;
        used[v] = true;
        for (auto edge : adj[v]) {
            int to = edge.first, len = edge.second;
            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                p[to] = v;
                q.push({d[to], to});
            }
        }
    }
}

//Complexidade: O(V)
vector<int> restorePath(int v) {
    vector<int> path;
    for (int u = v; u != -1; u = p[u])
        path.push_back(u);
    reverse(path.begin(), path.end());
    return path;
}
```

2 Matematica

2.1 Miller-rabin

```cpp
// Miller-Rabin
//
// Testa se n eh primo, n <= 3 * 10^18
//
// O(log(n)), considerando multiplicacao
// e exponenciacao constantes

ll mul(ll a, ll b, ll m) {
    ll ret = a*b - ll((long double)1/m*a*b+0.5)*m;
    return ret < 0 ? ret+m : ret;
}

ll pow(ll x, ll y, ll m) {
    if (!y) return 1;
    ll ans = pow(mul(x, x, m), y/2, m);
    return y%2 ? mul(x, ans, m) : ans;
```

```
}

bool prime(ll n) {
    if (n < 2) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0) return 0;
    ll r = __builtin_ctzll(n - 1), d = n >> r;

    // com esses primos, o teste funciona garantido para n <= 2^64
    // funciona para n <= 3*10^24 com os primos ate 41
    for (int a : {2, 325, 9375, 28178, 450775, 9780504, 795265022}) {
        ll x = pow(a, d, n);
        if (x == 1 or x == n - 1 or a % n == 0) continue;

        for (int j = 0; j < r - 1; j++) {
            x = mul(x, x, n);
            if (x == n - 1) break;
        }
        if (x != n - 1) return 0;
    }
    return 1;
}
```

3 Strings

3.1 Kmp

```
// pre() gera um vetor pi com o tamanho da string ne
// pi[i] = tamanho do maior prefixo de ne que eh sufixo de ne[0..i]
// Complexidade: O(n)
vector<int> pre(string ne)
{
    int n = ne.size();
    vector<int> pi(n, 0);
    for (int i = 1, j = 0; i < n; i++)
    {
        while (j > 0 && ne[i] != ne[j]) j = pi[j - 1];
        if (ne[i] == ne[j]) j++;
        pi[i] = j;
    }
    return pi;
}
// search() retorna o numero de ocorrencias de ne em hay
// complexidade: O(n+m)
int search(string hay, string ne)
{
    vector<int> pi = pre(ne);
    int c = 0;
    for (int i = 0, j = 0; i < hay.size(); i++)
    {
        while (j > 0 && hay[i] != ne[j]) j = pi[j - 1];
        if (hay[i] == ne[j]) j++;
        if (j == ne.size())
        {
            c++;
            // match at (i-j+1)
            j = pi[j - 1];
        }
    }
    return c;
}
```

3.2 Z Function

```
// a funcao z gera um vetor z com o tamanho da string s
//z[i] = tamanho do maior prefixo de s que eh sufixo de s[i..n-1]
//Complexidade: O(n)

vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

4 Template

4.1 Template

```
#include <bits/stdc++.h>
using namespace std;
//alias comp='g++ -std=c++17 -g -O2 -Wall -Wconversion -Wshadow -fsanitize
    =address,undefined -fno-sanitize-recover -ggdb -o out'

#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL
    ); //Melhora o desempenho
#define int long long //Melhor linha de codigo ja escrita
#define endl "\n" //Evita flush
#define loop(i,a,n) for(int i=a; i < n; i++)
#define input(x) for (auto &it : x) cin >> it
#define pb push_back
#define all(x) x.begin(), x.end()
#define ff first
#define ss second
#define mp make_pair
#define TETO(a, b) ((a) + (b-1))/(b)
#define dbg(x) cout << #x << " = " << x << endl
#define print(x,y) loop(it,0,y){cout << x[it] << " ";} cout << "\n";

typedef long long ll;
typedef long double ld;
typedef vector<int> vi;
typedef pair<int,int> pii;
typedef priority_queue<int, vector<int>, greater<int>> pqi;
```

```
24
25 const ll MOD = 1e9+7;
26 const int MAX = 1e4+5;
27 const ll LLINF = 0x3f3f3f3f3f3f3f3f; //escrevemos 3f 8 vezes
28 const double PI = acos(-1);
29
30 int32_t main(){ sws;
31
32
33     return 0;
34 }
```

5 Trees