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
int explore(int u)
{
    visited[u] = true;
    int c = 1;
    for (int v : adj[u])
        if (!visited[v])
           c += explore(v);
    return c;
}
```

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 Fast Exponentiation

```cpp
const int mod = 1e9+7;
int fexp(int a, int b)
{
    int ans = 1;
    while (b)
    {
        if (b & 1)
            ans = ans * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return ans;
}
```

2.2 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;
}
```

2.3 Sieve

```cpp
bool notPrime[ms];
int primes[ms], qnt;

void sieve(int lim) {
  primes[qnt++] = 1 // se o 1 for valido na questao
  for(int i = 2; i < ms; i++) { //loop(i,2,ms)
    if(notPrime[i]) continue;
    primes[qnt++] = i;
    for(int j = i + i; j < ms; j += i)
      notPrime[j] = true;
  }
```

2.4 Sieve Linear

```cpp
// Crivo de óEratstenes para gerar primos éat um limite 'lim'
// Complexidade: O(n log log n), onde n é o limite
const int ms = 1e6 + 5;
bool notPrime[ms]; // notPrime[i] é true se i ãno é um únmero primo
int primes[ms], qnt; // primes[] armazena os únmeros primos e qnt é a
    quantidade de primos encontrados

void sieve(int lim) {
primes[qnt++] = 1; // adiciona 1 como primo se for ávlido no problema
for(int i = 2; i <= lim; i++) {
if(notPrime[i]) continue; // se i ãno é primo, pula
primes[qnt++] = i; // i é primo, adiciona em primes[]
for(int j = i + i; j <= lim; j += i) // marca todos os úmltiplos de i como
     ãno primos
notPrime[j] = true;
}
}
```

3 Outros

3.1 Binaryconvert

```cpp
string decimal_to_binary(int dec) {
    string binary = "";
    while (dec > 0) {
        int bit = dec % 2;
        binary = to_string(bit) + binary;
        dec /= 2;
    }
    return binary;
}

int binary_to_decimal(string binary) {
    int dec = 0;
    int power = 0;
    for (int i = binary.length() - 1; i >= 0; i--) {
        int bit = binary[i] - '0';
        dec += bit * pow(2, power);
        power++;
    }
    return dec;
}
```

3.2 Binarysearch

```cpp
int BinarySearch(<vector>int arr, int x){
    int k = 0;
    int n = arr.size();

    for (int b = n/2; b >= 1; b /= 2) {
        while (k+b < n && arr[k+b] <= x) k += b;
    }
    if (arr[k] == x) {
        return k;
    }
}
```

3.3 Hoursconvert

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

int cts(int h, int m, int s) {
    int total = (h * 3600) + (m * 60) + s;
    return total;
}

tuple<int, int, int> cth(int total_seconds) {
    int h = total_seconds / 3600;
    int m = (total_seconds % 3600) / 60;
    int s = total_seconds % 60;
    return make_tuple(h, m, s);
}
```

3.4 Ispalindrome

```cpp
string isPalindrome(string S){
    string P = S;

    // Reverte P
    reverse(P.begin(), P.end());

    // Se S igual a P
    if (S == P){
        return 1;
    }else{
        return 0;
    }
}
```

3.5 Maxsubarraysum

```cpp
int maxSubarraySum(vector<int> x){

    int best = 0, sum = 0;
    for (int k = 0; k < n; k++) {
        sum = max(x[k],sum+x[k]);
        best = max(best,sum);
    }
    return best;
}
```

4 Strings

4.1 Kmp

```cpp
// 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;
}
```

4.2 Z Function

```cpp
// 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> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}
```

5 Template

5.1 Mini Template

```cpp
#include <bits/stdc++.h>
using namespace std;

```

```cpp
#define sws std::ios::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
#define int long long
#define endl "\n"
#define loop(i,a,n) for(int i=a; i < n; i++)
#define ff first
#define ss second

int32_t main(){ sws;


    return 0;
}
```

5.2 Template

```cpp
#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;

const ll MOD = 1e9+7;
const int MAX = 1e4+5;
const ll LLINF = 0x3f3f3f3f3f3f3f3f; //escrevemos 3f 8 vezes
const double PI = acos(-1);

int32_t main(){ sws;


    return 0;
}
```

6 Trees

6.1 Fenwick Tree

```cpp
// Fenwick Tree (Binary Indexed Tree) para somas de intervalos
// Complexidade:
vector<int> bit; // vetor da arvore
int n;

// Construtor da Fenwick tree usando add()
// O(n log n)
void build(vector<int> &a)
{
    n = a.size() + 1;
    bit.assign(n + 1, 0);
    for (size_t i = 0; i < a.size(); i++)
        add(i, a[i]);
}

// Construtor da Fenwick tree usando prefix sums
// O(n)
void build2(vector<int> &a)
{
    n = a.size() + 1;
    bit.assign(n + 1, 0);
    for (int i = 1; i < n; i++)
        bit[i] += a[i - 1];
    for (int i = 1; i < n; i++)
    {
        int j = i + (i & -i);
        if (j < n)
            bit[j] += bit[i];
    }
}

// Retorna a soma dos valores dos primeiros 'idx + 1' elementos
// O(logn)
int sum(int idx)
{
    int ret = 0;
    for (++idx; idx > 0; idx -= idx & -idx)
        ret += bit[idx];
    return ret;
}

// Retorna a soma dos valores dos elementos no intervalo [l, r]
// O(logn)
int sum(int l, int r)
{
    return sum(r) - sum(l - 1);
}

// Adiciona 'delta' ao valor na posicao 'idx' do vetor
// O(logn)
void add(int idx, int delta)
{
```

```
    for (++idx; idx < n; idx += idx & -idx)
        bit[idx] += delta;
}
```

6.2 Segtree

```
int v[MAXN];  // input array
Tnode seg[4 * MAXN];  // segment tree

Tnode combine(Tnode left, Tnode right) {
    // definir como combinar dois ós da árvore
}

Tnode build(int p, int l, int r) { // O(n)
    if (l == r) return seg[p] = {v[l], {l, r}};
    int m = (l + r) / 2;
    Tnode left = build(p * 2, l, m);
    Tnode right = build(p * 2 + 1, m + 1, r);
    seg[p] = combine(left, right);

}

Tnode update(int i, int x, int p, int l, int r) { // O(log n)
    if (i < l || r < i) return seg[p];
    if (l == r) {
        seg[p] = ...; // definir o que retornar quando l == r == i
        return seg[p];
    }
    int m = (l + r) / 2;
    Tnode left = update(i, x, p * 2, l, m);
    Tnode right = update(i, x, p * 2 + 1, m + 1, r);
    return seg[p] = combine(left, right);

}

Tnode query(int ql, int qr, int p, int l, int r) { // O(log n)
    if (qr < l || r < ql) {
        return ...;  // definir o que retornar quando ãno áh çãinterseo
    }
    if (ql <= l && r <= qr) {
        return seg[p];
    }
    int m = (l + r) / 2;
    Tnode left = query(ql, qr, p * 2, l, m);
    Tnode right = query(ql, qr, p * 2 + 1, m + 1, r);
    return combine(left, right);
}
```