

Competitive Programming Algorithms and Topics

BFS() not found!

1 Template

- 1.1 Código do Template
- 1.2 Script de submissão
- 1.3 Script em python

2 Matemática

- 2.1 Geometria
- 2.2 Exponenciação
- 2.3 Divisores

3 Grafos

- 3.1 Componentes fortemente conexas (SCC)
- 3.2 Caminho Euleriano
- 3.3 LCA (Menor Ancestral Comum)
- 3.4 Dijkstra (Caminhos mínimos)
- 3.5 Fluxo

4 Programação dinâmica

- 4.1 Mochila
- 4.2 Moedas
- 4.3 Troco
- 4.4 Segtree

1. Template

1.1. Código do Template

```

1 #include<bits/stdc++.h>
2
3 bool DEBUG = false;
4 // #define int long long
5 #define print if (DEBUG) std::cout <<
6 #define ff first
7 #define ss second
8 #define pii pair<int, int>
9 #define mp make_pair
10 #define pb push_back
11 #define vi vector<int>
12 #define INF (int)(1e9*2)
13 #define fori(N) for (int i = 0; i < N; ++i)
14 #define size(x) (int)x.size()
15 #define endl '\n'
16 #define PI 3.14159265358979323846
17
18 #define SYNC ios_base::sync_with_stdio(false), cin.tie(NULL), cout.tie(NULL)
19
20 using namespace std;
21
22 int32_t main() {
23     SYNC;
24     // Code
25     return 0;
26 }

```

1.2. Script de submissão

```

1 #!/bin/bash
2
3 IFS=' '
4 g++ -std=c++11 -o $1.out ./$1/$1.cpp
5 for i in {1..2};
6 do
7     echo "Teste $i"
8     ./$1.out < $1/input/in$i > $1/output/saida$i
9     Diff=$(diff $1/output/saida$i $1/output/out$i)
10    if [ "$Diff" ]; then
11        echo "$Diff"
12    else
13        echo "Ok"
14    fi
15 done
16
17 read -s -N 1 -p "Submit problem $1?[s/n] " option
18 if [[ "$option" == "S" ]] || [[ "$option" == "s" ]] || [[ "$option" ==
19    $'\x0a' ]]; then
20     echo "submit" $1;
21     # adicionar comando de submissao
22 else
23     echo "done"
24 fi

```

1.3. Script em python

```

1 #!/usr/bin/python
2 import math
3 print math.pi

```

2. Matemática

2.1. Geometria

```

1 //Dot product AB BC
2 int dot(int[] A, int[] B, int[] C){
3     AB = new int[2];
4     BC = new int[2];
5     AB[0] = B[0]-A[0];
6     AB[1] = B[1]-A[1];
7     BC[0] = C[0]-B[0];
8     BC[1] = C[1]-B[1];
9     int dot = AB[0] * BC[0] + AB[1] * BC[1];
10    return dot;
11 }
12
13 //Produto vetorial AB x AC
14 int cross(int[] A, int[] B, int[] C){
15     AB = new int[2];
16     AC = new int[2];
17     AB[0] = B[0]-A[0];
18     AB[1] = B[1]-A[1];
19     AC[0] = C[0]-A[0];
20     AC[1] = C[1]-A[1];
21     int cross = AB[0] * AC[1] - AB[1] * AC[0];
22     return cross;
23 }
24
25 //Distncia de A para B
26 double distance(int[] A, int[] B){
27     int d1 = A[0] - B[0];
28     int d2 = A[1] - B[1];
29     return sqrt(d1*d1+d2*d2);
30 }
31
32 //Distncia de C para a reta AB, se isSegment true, AB um segmento, no uma
33 //reta.
34 double linePointDist(int[] A, int[] B, int[] C, boolean isSegment){
35     double dist = cross(A,B,C) / distance(A,B);
36     if(isSegment){
37         int dot1 = dot(A,B,C);
38         if(dot1 > 0) return distance(B,C);
39         int dot2 = dot(B,A,C);
40         if(dot2 > 0) return distance(A,C);
41     }
42     return abs(dist);
43 }
44
45 -----
46 //rea de polgonos
47 int area = 0;
48 int N = quantidade de pontos no polgono e armazenados em p;
49 //Triangular o polgono em tringulos com pontos p[0],p[i],p[i+1]
50 for(int i = 1; i+1<N; i++){
51     int x1 = p[i][0] - p[0][0];
52     int y1 = p[i][1] - p[0][1];
53     int x2 = p[i+1][0] - p[0][0];
54     int y2 = p[i+1][1] - p[0][1];
55     int cross = x1*y2 - x2*y1;
56     area += cross;
57 }
58 return abs(cross/2.0);
59
60 -----

```

```

61 // Interseco de retas Ax + By = C    dados pontos (x1,y1) e (x2,y2)
62 A = y2-y1
63 B = x1-x2
64 C = A*x1+B*y1
65
66 //Retas definidas pelas equaes:
67 A1x + B1y = C1
68 A2x + B2y = C2
69
70 //Encontrar x e y resolvendo o sistema
71 double det = A1*B2 - A2*B1;
72 if(det == 0){
73     //Lines are parallel
74 }else{
75     double x = (B2*C1 - B1*C2)/det;
76     double y = (A1*C2 - A2*C1)/det;
77 }

```

2.2. Exponenciação

```

1 // Exponenciacao
2
3 int exp(int value1, int value2){
4     if (value2 == 0) return 1;
5     if (value2 == 1) return value1;
6     int result = exp(value1, value2 / 2);
7     result = result * result;
8     if (value2 % 2 == 1) result = result * value1;
9     return result;
10 }

```

2.3. Divisores

```

1 void factorize(unsigned long long n){
2     int raiz=sqrt(n);
3     for(int i =2; i<=n && i<=raiz; i++){
4         if(n%i==0){
5             while(n%i == 0)
6                 n/=i;
7             cout << i << endl;
8         }
9     }
10     if(n>1)
11         cout << n << endl;
12 }

```

3. Grafos

3.1. Componentes fortemente conexas (SCC)

```

1 void function() {
2     // code
3 }

```

3.2. Caminho Euleriano

```

1 list<int> cyc;
2 std::vector<pib > adj[MAX];
3 void euler_tour(list<int>::iterator it, int u) {
4     for (int j = 0; j < (int)adj[u].size(); j++) {
5         pib v = adj[u][j];

```

```

6     if (v.not_visited) {
7         adj[u][j].not_visited = false;
8         for (int k = 0; k < (int)adj[v.ff].size(); k++) {
9             pib uu = adj[v.ff][k];
10            if (uu.ff == u && uu.not_visited) {
11                adj[v.ff][k].not_visited = false;
12                break;
13            }
14        }
15        euler_tour(cyc.insert(it, u), v.ff);
16    }
17 }
18 }

```

3.3. LCA (Menor Ancestral Comum)

```

1 #define MAXN 1001 // N
2 #define MAXL 10 // (N / 0.3) -> log_2(N)
3
4 int ancestral[MAXN][MAXL];
5 void monta_tabela() {
6     // Inicializa todos com -1
7     for (int i = 0; i < MAXN; i++)
8         for (int j = 0; j < MAXL; j++)
9             ancestral[i][j] = -1;
10
11     for (int i = 1; i <= N; i++)
12         ancestral[i][0] = 0;
13
14     // Montamos o restante da tabela com programacao dinamica
15     for (int j = 1; j < MAXL; j++)
16         for (int i = 1; i <= N; i++)
17             if (ancestral[i][j-1] != -1) {
18                 int k = ancestral[i][j-1];
19                 ancestral[i][j] = ancestral[k][j-1];
20             }
21 }
22
23 int lca(int u, int v) {
24     if (nivel[u] < nivel[v]) swap(u, v);
25     // Agora vamos fazer o nivel[u] ser igual ao nivel[v], subindo pelos
26     // ancestrais de u
27     for (int i = MAXL - 1; i >= 0; i--)
28         if (nivel[u] - (1<<i) >= nivel[v]) u = ancestral[u][i];
29     // Agora eles estao no mesmo nvel
30     if (u == v) return u;
31
32     for (int i = MAXL - 1; i >= 0; i--) {
33         if (ancestral[u][i] != -1 && ancestral[u][i] != ancestral[v][i]) {
34             u = ancestral[u][i];
35             v = ancestral[v][i];
36         }
37     }
38     // Como subimos o maximo possivel sabemos que u != v e
39     // que pai[u] == pai[v], logo LCA(u, v) == pai[u] == pai[v]
40     return ancestral[u][0];
41 }

```

3.4. Dijkstra (Caminhos mínimos)

```

1 class Grafo {
2 private:
3     int n; // Nmero de vertices

```

```

4     list<pii> * adj;
5 public:
6     Grafo(int n) {
7         this->n = n; // Quantidade de vertices
8         adj = new list<pii>[n]; // Cria lista de adjacencia
9     }
10
11     // Adiciona aresta ao Grafo
12     void addAresta(int v1, int v2, int custo) {
13         adj[v1].pb(mp(v2, custo));
14     }
15
16     int dijkstra(int orig, int dest) {
17         // Vetor de distancias
18         int dist[n];
19         bool vis[n];
20         priority_queue<pii, vector<pii>, greater<pii>> pq;
21         // Inicia vetores de distancias e visitados
22         for (int i = 0; i < n; i++) {
23             dist[i] = INFINITO;
24             vis[i] = false;
25         }
26         dist[orig] = 0;
27         pq.push(mp(dist[orig], orig));
28
29         while (!pq.empty()) {
30             pii p = pq.top(); pq.pop();
31             int u = p.second;
32             // verifica se o vertice nao foi expandido
33             if (vis[u] == false) {
34                 // marca visitados
35                 vis[u] = true;
36                 // Percorre os vrtices "v" adjacentes a "u"
37                 for (list<pii>::iterator it = adj[u].begin(); it !=
38                     adj[u].end(); it++) {
39                     // Obtm o vrtice adjacente e o custo da aresta
40                     int v = it->first;
41                     int custo_aresta = it->second;
42                     // relaxamento(u, v)
43                     if (dist[v] > (dist[u] + custo_aresta)) {
44                         dist[v] = (dist[u] + custo_aresta); // Atualiza a
45                         // dist de "v"
46                         pq.push(mp(dist[v], v));
47                     }
48                 }
49             }
50             return dist[dest];
51 }

```

3.5. Fluxo

```

1 int res[MAX][MAX], mf, f, s, t; // Nao esqueca de zerar a matriz
2 vi p; // p stores the BFS spanning tree from s
3
4 void augment(int v, int minEdge) { // Transverse BFS spanning from s->t
5     if (v == s) {
6         f = minEdge; // Record a minEdge in a global var f
7         return;
8     }
9     else if (p[v] != -1) {
10        augment(p[v], min(minEdge, res[p[v]][v]));
11        res[p[v]][v] -= f;

```

```

12     res[v][p[v]] += f;
13 }
14 }
15
16 void flow() {
17     mf = 0;
18     while(1){
19         f = 0;
20         vi dist(mn, INF); dist[s] = 0;
21         std::vector<bool> vis(mn, false);
22         queue<int> q; q.push(s);
23         p.assign(mn, -1); // Record the BFS spanning tree, from s to t
24         while(!q.empty()){
25             int u = q.front(); q.pop();
26             if (u == t) break; // Immediately stop BFS if we already reach
27
28             // for(int v = 0; v < mn; v++) {
29             if(vis[u]) continue;
30             vis[u] = true;
31             for(int i = 0; i < (int)adj[u].size(); i++) {
32                 int v = adj[u][i];
33                 if(vis[v] || !res[u][v]) continue;
34
35                 if (res[u][v] > 0 && dist[v] == INF) {
36                     dist[v] = dist[u] + 1;
37                     q.push(v);
38                     p[v] = u;
39                 }
40             }
41             augment(t, INF); // Find the min edge weight 'f' in this path, if any
42             if (f == 0) break; // We cannot send any more flow ('f' == 0),
43             terminate
44             mf += f; // We can still send a flow, increase teh max flow
45 }

```

4. Programação dinâmica

4.1. Mochila

```

1 const int N = 2005;
2 int p[N], v[N];
3 int memo[N][N]; //memset(memo, -1, sizeof memo);
4 int mochila(int i, int j) {
5     if(i == 0) return 0;
6     if(memo[i][j] != -1) return memo[i][j];
7
8     // no colocar o item => mochila(i-1, j)
9     // colocar o item => mochila(i-1, j - p[i]) + v[i]
10    int res = mochila(i-1, j);
11    if(p[i] <= j) {
12        res = max(res, mochila(i-1, j - p[i]) + v[i]);
13    }
14
15    return memo[i][j] = res;
16 }

```

4.2. Moedas

```

1 void moedas(int argc, char const *argv[]){
2     int m, n;
3     cin >> m >> n;

```

```

4 while(m){
5     vector<int> array(m+1, 50001);
6     array[0] = 0;
7     for (int i = 0; i < n; ++i){
8         int valor;
9         cin >> valor;
10        for (int j = 0; j < m; ++j){
11            if(array[j] != 50001 && j + valor <= m)
12                if(array[j+valor] > array[j] + 1)
13                    array[j+valor] = array[j]+1;
14        }
15    }
16    if(array[m] < 50001){
17        cout << array[m] << endl;
18    }
19    else
20        cout << "Impossivel" << endl;
21    cin >> m >> n;
22 }
23 }

```

4.3. Troco

```

1 void troco(){
2     int v, m;
3     cin >> v >> m;
4     vector<int> moedas(v+1);
5     vector<int> entrada(m);
6     moedas[0] = 0;
7     for (int i = 1; i <= v; ++i) moedas[i] = -1;
8     for (int i = 0; i < m; ++i)
9         cin >> entrada[i];
10    for (int j = 0; j < m; ++j){
11        int a = entrada.back();
12        entrada.pop_back();
13        for (int i = v; i >= 0; --i){
14            if(moedas[i] >= 0 && (i + a) <= v){
15                if(moedas[i + a] == -1)
16                    moedas[i + a] = 1;
17                else
18                    moedas[i + a]++;
19            }
20        }
21    }
22    if(moedas[v] > 0)
23        cout << "S\n";
24    else
25        cout << "N\n";
26 }
27 }

```

4.4. Segtree

```

1 const int MAX_N = 10000;
2 int v[MAX_N + 1];
3 int tree[MAX_N * 4];
4
5 void build(int no, int esq, int dir) {
6     if (esq == dir) {
7         tree[no] = v[esq];
8     } else {
9         int meio = (esq + dir) / 2;
10        // intervalo esquerda: [esq, meio]

```

```
11 // intervalo direita: [meio+1, dir]
12 build(2 * no + 1, esq, meio);
13 build(2 * no + 2, meio + 1, dir);
14 tree[no] = tree[2 * no + 1] + tree[2 * no + 2];
15 }
16 }
17
18 void update(int no, int esq, int dir, int i) {
19     if (esq == dir) {
20         tree[no] = v[i];
21     } else {
22         int meio = (esq + dir) / 2;
23         if (i <= meio)
24             update(2 * no + 1, esq, meio, i);
25         else
26             update(2 * no + 2, meio + 1, dir, i);
27         tree[no] = tree[2 * no + 1] + tree[2 * no + 2];
28     }
29 }
30
31 int query(int no, int esq, int dir, int i, int j) {
32     if (esq == i && dir == j) {
33         return tree[no];
34     } else {
35         int meio = (esq + dir) / 2;
36         if (j <= meio)
37             return query(2 * no + 1, esq, meio, i, j);
38         else if (i > meio)
39             return query(2 * no + 2, meio + 1, dir, i, j);
40         else
41             return min(query(2 * no + 1, esq, meio, i, meio),
42                        query(2 * no + 2, meio + 1, dir, meio + 1, j));
43     }
44 }
45
46 int main() {
47     int n, m;
48     cin >> n >> m;
49     for (int i = 1; i <= n; ++i)
50         cin >> v[i];
51     build(0, 1, n);
52     for (int consulta = 1; consulta <= m; ++consulta) {
53         int tipo;
54         cin >> tipo;
55         if (tipo == 1) {
56             int i, x;
57             cin >> i >> x;
58             v[i] += x;
59             update(0, 1, n, i);
60         } else {
61             int l, r;
62             cin >> l >> r;
63             cout << query(0, 1, n, l, r) << endl;
64         }
65     }
66 }
```