Competitive Programming Algorithms and Topics

BFS() not found!

1. Template

1.1. Código do Template

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

bool DEBUG = false;
// #define int long long
#define print if (DEBUG) std::cout <<
#define ff first
#define ss second
#define pii pair<int, int>
#define mp make_pair
#define pb push_back
#define vi vector<int>
#define INF (int)(1e9*2)
#define fori(N) for (int i = 0; i < N; ++i)
#define size(x) (int)x.size()
#define endl '\n'
#define PI 3.14159265358979323846
#define SYNC ios_base::sync_with_stdio(false), cin.tie(NULL), cout.tie(NULL)

using namespace std;

int32_t main() {
  SYNC;
  // Code
    return 0;
}
//troque o int por qualquer outro tipo primitivo
// numeric_limits<int>::max() //retorna limite maximo do tipo int
// numeric_limits<int>::lowest() // retorna limite minimo do tipo int
```

1.2. Script de submissão

```sh
#!/bin/bash

IFS=''
g++ -std=c++11 -o $1.out ./$1/$1.cpp
for i in {1..2};
do
  echo "Teste $i"
  ./$1.out < $1/input/in$i > $1/output/saida$1
  Diff=$(diff $1/output/saida$1 $1/output/out$i)
  if [ "$Diff" ]; then
    echo "$Diff"
  else
    echo "Ok"
  fi
done

read -s -N 1 -p "Submit problem $1?[s/n] " option
if  [[ "$option" == "S" ]] || [[ "$option" == "s" ]] || [[ "$option" == $'\x0a' ]] ; then
  echo "submit" $1;
  # adicionar comando de submissao
else
  echo "done"
fi
```

1.3. Script em python

```sh
#!/usr/bin/python
```

```
import math
print math.pi
```

2. Matemática

2.1. Geometria

```
//Dot product AB  BC
int dot(int[] A, int[] B, int[] C){
    AB = new int[2];
    BC = new int[2];
    AB[0] = B[0]-A[0];
    AB[1] = B[1]-A[1];
    BC[0] = C[0]-B[0];
    BC[1] = C[1]-B[1];
    int dot = AB[0] * BC[0] + AB[1] * BC[1];
    return dot;
}

//Produto vetorial AB x AC
int cross(int[] A, int[] B, int[] C){
    AB = new int[2];
    AC = new int[2];
    AB[0] = B[0]-A[0];
    AB[1] = B[1]-A[1];
    AC[0] = C[0]-A[0];
    AC[1] = C[1]-A[1];
    int cross = AB[0] * AC[1] - AB[1] * AC[0];
    return cross;
}

//Distncia de A para B
double distance(int[] A, int[] B){
    int d1 = A[0] - B[0];
    int d2 = A[1] - B[1];
    return sqrt(d1*d1+d2*d2);
}

//Distncia de C para a reta AB, se isSegment  true, AB  um segmento, no uma
    reta.
double linePointDist(int[] A, int[] B, int[] C, boolean isSegment){
    double dist = cross(A,B,C) / distance(A,B);
    if(isSegment){
        int dot1 = dot(A,B,C);
        if(dot1 > 0)return distance(B,C);
        int dot2 = dot(B,A,C);
        if(dot2 > 0)return distance(A,C);
    }
    return abs(dist);
}

-------------------------------------------------------
//rea de polgonos
int area = 0;
int N = quantidade de pontos no polgono e armazenados em p;
//Triangular o polgono em tringulos com pontos p[0],p[i],p[i+1]

for(int i = 1; i+1<N; i++){
    int x1 = p[i][0] - p[0][0];
    int y1 = p[i][1] - p[0][1];
    int x2 = p[i+1][0] - p[0][0];
    int y2 = p[i+1][1] - p[0][1];
    int cross = x1*y2 - x2*y1;
    area += cross;
}
return abs(cross/2.0);

-------------------------------------------------------
// Interseco de retas Ax + By = C    dados pontos (x1,y1) e (x2,y2)
A = y2-y1
B = x1-x2
C = A*x1+B*y1

//Retas definidas pelas equaes:
A1x + B1y = C1
A2x + B2y = C2

//Encontrar x e y resolvendo o sistema
double det = A1*B2 - A2*B1;
if(det == 0){
    //Lines are parallel
}else{
    double x = (B2*C1 - B1*C2)/det;
    double y = (A1*C2 - A2*C1)/det;
}
```

2.2. Exponenciação

```
// Exponenciacao

int exp(int value1, int value2){
  if (value2 == 0) return 1;
  if (value2 == 1) return value1;
  int result = exp(value1, value2 / 2);
  result = result * result;
  if (value2 % 2 == 1) result = result * value1;
  return result;
}
```

2.3. Divisores

```
void factorize(unsigned long long n){
  int raiz=sqrt(n);
  for(int i =2; i<=n && i<=raiz; i++){
    if(n%i==0){
      while(n%i == 0)
        n/=i;
      cout << i << endl;
    }
  }
  if(n>1)
    cout << n << endl;
}
```

2.4. Permutações

```
void permutations() {
  std::vector<int> vec = {1, 2, 3, 4, 5};
  int n = vec.size();
  int cont = 0;
  do{
    for (int i = 0; i < n; ++i)
      cout << vec[i] << ' ';
    cout << endl;
    cont++;
  } while (next_permutation(vec.begin(), vec.end()));
```

```
  cout << cont << endl;
  return 0;
}
```

3. Grafos

3.1. Componentes fortemente conexas (SCC)

```
void function() {
  // code
}
```

3.2. Caminho Euleriano

```
list<int> cyc;
std::vector<pib > adj[MAX];
void euler_tour(list<int>::iterator it, int u) {
    for (int j = 0; j < (int)adj[u].size(); j++) {
        pib v = adj[u][j];
        if (v.not_visited) {
            adj[u][j].not_visited = false;
            for (int k = 0; k < (int)adj[v.ff].size(); k++) {
                pib uu = adj[v.ff][k];
                if (uu.ff == u && uu.not_visited) {
                    adj[v.ff][k].not_visited = false;
                    break;
                }
            }
            euler_tour(cyc.insert(it, u), v.ff);
        }
    }
}
```

3.3. LCA (Menor Ancestral Comum)

```
#define MAXN 1001 // N
#define MAXL 10 // (N / 0.3) -> log_2(N)

int ancestral[MAXN][MAXL];
void monta_tabela() {
    // Inicializa todos com -1
    for (int i = 0; i < MAXN; i++)
        for (int j = 0; j < MAXL; j++)
            ancestral[i][j] = -1;

    for (int i = 1; i <= N; i++)
        ancestral[i][0] = 0;

    // Montamos o restante da tabela com programacao dinamica
    for (int j = 1; j < MAXL; j++)
        for (int i = 1; i <= N; i++)
            if (ancestral[i][j - 1] != -1) {
                int k = ancestral[i][j - 1];
                ancestral[i][j] = ancestral[k][j - 1];
            }
}

int lca(int u, int v) {
    if (nivel[u] < nivel[v]) swap(u, v);
    // Agora vamos fazer o nivel[u] ser igual ao nivel[v], subindo pelos
    ancestrais de u
    for (int i = MAXL - 1; i >= 0; i--)
        if (nivel[u] - (1<<i) >= nivel[v]) u = ancestral[u][i];
        // Agora eles estao no mesmo nvel
    if(u == v) return u;

    for (int i = MAXL - 1; i >= 0; i--) {
        if (ancestral[u][i] != -1 && ancestral[u][i] != ancestral[v][i]) {
            u = ancestral[u][i];
            v = ancestral[v][i];
        }
    }
    // Como subimos o maximo possivel sabemos que u != v e
    // que pai[u] == pai[v], logo LCA(u, v) == pai[u] == pai[v]
    return ancestral[u][0];
}
```

3.4. Dijkstra (Caminhos mínimos)

```
class Grafo {
private:
    int n; // Nmero de vertices
    list<pii > * adj;
public:
    Grafo (int n) {
        this->n = n; // Quantidade de vertices
        adj = new list<pii >[n]; // Cria lista de adjacencia
    }

    // Adiciona aresta ao Grafo
    void addAresta(int v1, int v2, int custo) {
        adj[v1].pb(mp(v2, custo));
    }

    int dijkstra(int orig, int dest) {
        // Vetor de distancias
        int dist[n];
        bool vis[n];
        priority_queue<pii, vector<pii >, greater<pii > > pq;
        // Inicia vetores de distancias e visitados
        for (int i = 0; i < n; i++) {
            dist[i] = INFINITO;
            vis[i] = false;
        }
        dist[orig] = 0;
        pq.push(mp(dist[orig], orig));

        while (!pq.empty()) {
            pii p = pq.top(); pq.pop();
            int u = p.second;
            // verifica se o vertice nao foi expandido
            if (vis[u] == false) {
                // marca visitados
                vis[u] = true;
                // Percorre os vrtices "v" adjacentes a "u"
                for (list<pii >::iterator it = adj[u].begin(); it !=
    adj[u].end(); it++) {
                    // Obtm o vrtice adjacente e o custo da aresta
                    int v = it->first;
                    int custo_aresta = it->second;
                    // relaxamento(u, v)
                    if (dist[v] > (dist[u] + custo_aresta)) {
                        dist[v] = (dist[u] + custo_aresta); // Atualiza a
    dist de "v"
                        pq.push(mp(dist[v], v));
                    }
```

```
                }
            }
        }
        return dist[dest];
    }
};
```

3.5. Fluxo

```
int res[MAX][MAX], mf, f, s, t; // Nao esqueca de zerar a matriz
vi p; // p stores the BFS spanning tree from s

void augment(int v, int minEdge) { // Transverse BFS spanning from s->t
    if (v == s) {
        f = minEdge; // Record a minEdge in a global var f
        return;
    }
    else if(p[v] != -1) {
        augment(p[v], min(minEdge, res[p[v]][v]));
        res[p[v]][v] -= f;
        res[v][p[v]] += f;
    }
}

void flow() {
    mf = 0;
    while(1){
        f = 0;
        vi dist(mn, INF); dist[s] = 0;
        std::vector<bool> vis(mn, false);
        queue<int> q; q.push(s);
        p.assign(mn, -1);  // Record the BFS spanning tree, from s to t
        while(!q.empty()){
            int u = q.front(); q.pop();
            if (u == t) break; // Immediatly stop BFS if we already reach
    sink t
            // for(int v = 0; v < mn; v++) {
            if(vis[u]) continue;
            vis[u] = true;
             for(int i = 0; i < (int)adj[u].size(); i++) {
                int v = adj[u][i];
                if(vis[v] || !res[u][v]) continue;

                if (res[u][v] > 0 && dist[v] == INF) {
                    dist[v] = dist[u] + 1;
                    q.push(v);
                    p[v] = u;
                }
            }
        }
        augment(t, INF); // Find the min edge weight 'f in this path, if any
        if (f == 0) break; // We cannot send any more flow ('f' == 0),
    terminate
        mf += f; // We can still send a flow, increase teh max flow
    }
}
```

3.6. DisjointSet

```
class DisjointSet{
    public: vector<int> parent, component_size;
    DisjointSet(int n): parent(n) {
        for(int i=0; i<n; i++) {
            parent[i] = i;
            component_size.pb(1);
        }
    }
    void join(int a, int b) {
        if(!check(a,b)) {
            int s = component_size[find(b)];
            parent[find(b)] = find(a);
            component_size[find(a)] += s;
        }
    }
    int find(int a){
        return a == parent[a] ? a : parent[a] = find(parent[a]);
    }
    bool check(int a, int b) {
        return find(a) == find(b);
    }
    int getsize(int a) {
        return component_size[find(a)];
    }
};
```

4. Programação dinâmica

4.1. Mochila

```
const int N = 2005;
int p[N], v[N];
int memo[N][N]; //memset(memo, -1, sizeof memo);
int mochila(int i, int j) {
  if(i == 0) return 0;
  if(memo[i][j] != -1) return memo[i][j];

  // no colocar o item => mochila(i-1. j)
  // colocar o item => mochila(i-1, j - p[i]) + v[i]
  int res = mochila(i-1, j);
  if(p[i] <= j) {
    res = max(res, mochila(i-1, j - p[i]) + v[i]);
  }

  return memo[i][j] = res;
}
```

4.2. Moedas

```
void moedas(int argc, char const *argv[]){
  int m, n;
  cin >> m >> n;
  while(m){
    vector<int> array(m+1, 50001);
    array[0] = 0;
    for (int i = 0; i < n; ++i){
      int valor;
      cin >> valor;
      for (int j = 0; j < m; ++j){
        if(array[j] != 50001 && j + valor <= m)
          if(array[j+valor] > array[j] + 1)
            array[j+valor] = array[j]+1;
      }
    }
    if(array[m] < 50001){
      cout << array[m] << endl;
    }
```

```
    else
      cout << "Impossivel" << endl;
    cin >> m >> n;
  }
}
```

4.3. Troco

```
void troco(){
  int v, m;
  cin >> v >> m;
  vector<int> moedas(v+1);
  vector<int> entrada(m);
  moedas[0] = 0;
  for (int i = 1; i <= v; ++i) moedas[i] = -1;
  for (int i = 0; i < m; ++i)
    cin >> entrada[i];
  for (int j = 0; j < m; ++j){
    int a = entrada.back();
    entrada.pop_back();
    for (int i = v; i >= 0; --i){
      if(moedas[i] >= 0 && (i + a) <= v){
        if(moedas[i + a] == -1)
          moedas[i + a] = 1;
        else
          moedas[i + a]++;
      }
    }

  }
  if(moedas[v] > 0)
    cout << "S\n";
  else
    cout << "N\n";
}
```

4.4. Segtree

```
const int MAX_N = 10000;
int v[MAX_N + 1];
int tree[MAX_N * 4];

void build(int no, int esq, int dir) {
    if (esq == dir) {
        tree[no] = v[esq];
    } else {
        int meio = (esq + dir) / 2;
        // intervalo esquerda: [esq, meio]
        // intervalo direita: [meio+1, dir]
        build(2 * no + 1, esq, meio);
        build(2 * no + 2, meio + 1, dir);
        tree[no] = tree[2 * no + 1] + tree[2 * no + 2];
    }
}

void update(int no, int esq, int dir, int i) {
    if (esq == dir) {
        tree[no] = v[i];
    } else {
        int meio = (esq + dir) / 2;
        if (i <= meio)
            update(2 * no + 1, esq, meio, i);
        else
            update(2 * no + 2, meio + 1, dir, i);
        tree[no] = tree[2 * no + 1] + tree[2 * no + 2];
    }
}

int query(int no, int esq, int dir, int i, int j) {
    if (esq == i && dir == j) {
        return tree[no];
    } else {
        int meio = (esq + dir) / 2;
        if (j <= meio)
            return query(2 * no + 1, esq, meio, i, j);
        else if (i > meio)
            return query(2 * no + 2, meio + 1, dir, i, j);
        else
            return min(query(2 * no + 1, esq, meio, i, meio),
              query(2 * no + 2, meio + 1, dir, meio + 1, j));
    }
}

int main() {
    int n, m;
    cin>>n>>m;
    for (int i = 1; i <= n; ++i)
        cin>>v[i];
    build(0, 1, n);
    for (int consulta = 1; consulta <= m; ++consulta) {
        int tipo;
        cin>>tipo;
        if (tipo == 1) {
            int i, x;
            cin>>i>>x;
            v[i] += x;
            update(0, 1, n, i);
        } else {
            int l, r;
            cin>>l>>r;
            cout<<query(0, 1, n, l, r)<<endl;
        }
    }
}
```