Notebook UNosnovatos

Contents

1 C++

1.1 C++ plantilla

```
#include <bits/stdc++.h>
using namespace std;
#define sz(arr) ((int) arr.size())
typedef long long ll;
typedef pair<int, int> ii;
typedef vector<ii> vii;
typedef vector<int> vi;
typedef vector<long long> vl;
const int INF = 1e9;
const ll INFL = 1e18;
const int MOD = 1e9+7;
int dirx[4] = {0,-1,1,0};
int diry[4] = {-1,0,0,1};
int dr[] = {1, 1, 0, -1, -1, -1, 0, 1};
int dc[] = {0, 1, 1, 1, 0, -1, -1, -1};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    // freopen("file.in", "r", stdin);
    // freopen("file.out", "w", stdout);

    return 0;
}
```

2 Estructuras de Datos

2.1 Disjoint Set Union

```
struct dsu{
    vi p, size;
    int num_sets;
    int maxSize;

    dsu(int n){
        p.assign(n, 0);
        size.assign(n, 1);
        num_sets = n;
        for (int i = 0; i<n; i++) p[i] = i;
    }

    int find_set(int i) {return (p[i] == i) ? i : (p[i] =
        find_set(p[i]));}

    bool is_same_set(int i, int j) {return find_set(i) ==
        find_set(j);}

    void unionSet(int i, int j){
            if (!is_same_set(i, j)){
                int a = find_set(i), b = find_set(j);
                if (size[a] < size[b])
                    swap(a, b);
                p[b] = a;
                size[a] += size[b];
                maxSize = max(size[a], maxSize);
                num_sets--;
            }
        }
};
```

2.2 Segment Tree

```
int nullValue = 0;
```

```
struct nodeST{
    nodeST *left, *right;
    int l, r; ll value, lazy, lazy1;

    nodeST(vi &v, int l, int r) : l(l), r(r){
        int m = (l+r)>>1;
        lazy = 0;
        lazy1 = 0;
        if (l!=r){
            left = new nodeST(v, l, m);
            right = new nodeST(v, m+1, r);
            value = opt(left->value, right->value);
        }
        else{
            value = v[l];
        }
    }

    ll opt(ll leftValue, ll rightValue){
        return leftValue + rightValue;
    }

    void propagate(){
        if(lazy1){
            value = lazy1 * (r-l+1);
            if (l != r){
                left->lazy1 = lazy1, right->lazy1 = lazy1
                    ;
                left->lazy = 0, right->lazy = 0;
            }
            lazy1 = 0;
            lazy = 0;
        }
        else{
            value += lazy * (r-l+1);
            if (l != r){
                if(left->lazy1) left->lazy1 += lazy;
                else left->lazy += lazy;
                if(right->lazy1) right->lazy1 += lazy;
                else right->lazy += lazy;
            }
            lazy = 0;
        }
    }

    ll get(int i, int j){
        propagate();
        if (l>=i && r<=j) return value;
        if (l>j  || r<i) return nullValue;

        return opt(left->get(i, j), right->get(i, j));
    }

    void upd(int i, int j, int nv){
        propagate();
        if (l>j  || r<i) return;
        if (l>=i && r<=j){
            lazy += nv;
            propagate();
            // value = nv;
            return;
        }

        left->upd(i, j, nv);
        right->upd(i, j, nv);

        value = opt(left->value, right->value);
    }

    void upd(int k, int nv){
        if (l>k  || r<k) return;
        if (l>=k && r<=k){
            value = nv;
            return;
        }

        left->upd(k, nv);
        right->upd(k, nv);

        value = opt(left->value, right->value);
    }

    void upd1(int i, int j, int nv){
        propagate();
        if (l>j  || r<i) return;
        if (l>=i && r<=j){
            lazy = 0;
            lazy1 = nv;
            propagate();
            return;
        }

        left->upd1(i, j, nv);
        right->upd1(i, j, nv);

        value = opt(left->value, right->value);
    }
};
```

3 Grafos

3.1 DFS

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

int vertices, aristas;

vector<int> dfs_num(vertices+1, -1); //Vector del estado
    de cada vertice (visitado o no visitado)

const int NO_VISITADO = -1;
const int VISITADO = 1;
```

```
vector<vector<int>> adj(vertices + 1); //Lista adjunta
    del grafo

// Complejidad O(V + E)
void dfs(int v){
    dfs_num[v] = VISITADO;
    //Se recorren los vecinos
    for (int i = 0; i < (int) adj[v].size(); i++){

        if (dfs_num[adj[v][i]] == NO_VISITADO){
            dfs(adj[v][i]);
        }

    }
}
```

3.2 BFS

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> ii;
typedef vector<ii> vii;
typedef vector<int> vi;
typedef vector<long long> vl;
vector<vi> adj;

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);
    ll n, m; cin >> n >> m;
    adj.resize(n+1);
    for (int i = 0; i<m; i++){
        int x, y; cin >> x >> y;
        adj[x].push_back(y);
        adj[y].push_back(x);
    }
    //BFS, complejidad O(V + E)
    queue<int> q; q.push(adj[1][0]);     //Origen
    vi d(n+1, INT_MAX); d[adj[1][0]] = 0;   //La
       distancia del vertice a el mismo es cero
    while(!q.empty()){
        int nodo = q.front(); q.pop();
        for (int i = 0; i<(int)adj[nodo].size(); i++){
            if (d[adj[nodo][i]] == INT_MAX){   //Si el
               vecino no visitado y alcanzable
                d[adj[nodo][i]] = d[nodo] + 1;    //Hacer
                    d[adj[u][i]] != INT_MAX para
                   etiquetarlo
                q.push(adj[nodo][i]);          //
                   Anadiendo a la cola para siguiente
                   iteracion
            }
        }
    }
```

```
}
```

3.3 Puntos de articulacion y puentes

```
vi dfs_num, dfs_low, dfs_parent, articulation_vertex;
int dfsNumberCounter, dfsRoot, rootChildren;
vector<vii> adj;
void articulationPointAndBridge(int u) {
    dfs_num[u] = dfsNumberCounter++;
    dfs_low[u] = dfs_num[u]; // dfs_low[u]<=dfs_num[u]
    for (auto &[v, w] : adj[u]) {
        if (dfs_num[v] == -1) { // a tree edge
            dfs_parent[v] = u;
            if (u == dfsRoot) ++rootChildren; // special
               case, root
            articulationPointAndBridge(v);
            if (dfs_low[v] >= dfs_num[u]) // for
               articulation point
                articulation_vertex[u] = 1; // store this
                    info first
            if (dfs_low[v] > dfs_num[u]) // for bridge
                printf(" (%d, %d) is a bridge\n", u, v);
            dfs_low[u] = min(dfs_low[u], dfs_low[v]); //
               subtree, always update
        }
        else if (v != dfs_parent[u]) // if a non-trivial
           cycle
            dfs_low[u] = min(dfs_low[u], dfs_num[v]); //
               then can update
    }
}
int main(){
    dfs_num.assign(V, -1); dfs_low.assign(V, 0);
    dfs_parent.assign(V, -1); articulation_vertex.assign(
       V, 0);
    dfsNumberCounter = 0;
    adj.resize(V);

    printf("Bridges:\n");
    for (int u = 0; u < V; ++u)
        if (dfs_num[u] == -1) {
            dfsRoot = u; rootChildren = 0;
            articulationPointAndBridge(u);
            articulation_vertex[dfsRoot] = (rootChildren
               > 1); // special case
        }
    printf("Articulation Points:\n");
    for (int u = 0; u < V; ++u)
        if (articulation_vertex[u])
            printf(" Vertex %d\n", u);
}
```

3.4 Orden Topologico

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> ii;
typedef vector<ii> vii;
typedef vector<int> vi;
typedef vector<long long> vl;
#define INF 1000000000;

vector<vi> adj;
vi dfs_num;
vi ts;

void dfs(int v){
    dfs_num[v] = 1;
    for (int i = 0; i < (int) adj[v].size(); i++){

        if (dfs_num[adj[v][i]] != 1){
            dfs(adj[v][i]);
        }

    }
    ts.push_back(v);
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);

    int n, m;
    cin >> n >> m;
    adj.resize(n+1);
    dfs_num.resize(n+1);

    for (int i = 0; i<m; i++){
        int x, y;
        cin >> x >> y;
        adj[x].push_back(y);
        adj[y].push_back(x);
    }

    for (int i = 1; i<=n; i++){
        if (dfs_num[i] != 1){
            dfs(i);
        }
    }

    reverse(ts.begin(), ts.end());

    return 0;
}
```

3.5 Algoritmo de Khan

```cpp
int n, m;
vector<vi> adj;
vi grado;
vi orden;

void khan(){
    queue<int> q;

    for (int i = 1; i<=n; i++){
        if (!grado[i]) q.push(i);
    }
    int nodo;

    while(!q.empty()){
        nodo = q.front(); q.pop();
        orden.push_back(nodo);
        for (int v : adj[nodo]){
            grado[v]--;
            if (grado[v] == 0) q.push(v);
        }
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin >> n >> m;
    adj.resize(n+1);
    grado.resize(n+1);

    for (int i = 0; i<m; i++){
        int x, y; cin >> x >> y;
        adj[x].push_back(y);
        grado[y]++;
    }

    khan();

    if (orden.size() == n){
        for (int i : orden) cout << i;
    }
    else{
        cout << "No DAG"; //No es un grafo aciclico
            dirigido (tiene un ciclo)
    }
}
```

3.6 Floodfill

```cpp
//Relleno por difusion-etiquetado/coloreado de
    componentes conexos
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> ii;
typedef vector<ii> vii;
```

```cpp
typedef vector<int> vi;
typedef vector<long long> vl;
#define INF 1000000000;
int dr[] = {1, 1, 0, -1, -1, -1, 0, 1};      //Truco para
    explorar rejilla 2d
int dc[] = {0, 1, 1, 1, 0, -1, -1, -1};      // vecinos S,
    SE,E,NE,N,NO,O,SO

vector<string> grid;

int R, C, ans;

int floodfill(int r, int c, char c1, char c2){
    //Devuelve tamano de CC
    if (r < 0 || r >= R || c< 0 || c >= C) return 0;
        //fuera de la rejilla
    if (grid[r][c] != c1) return 0;
        //No tiene color c1
    int ans = 1;                    //suma 1 a ans porque el
         vertice (r, c) tiene color c1
    grid[r][c] = c2;                //Colorea el vertice (r,
         c) a c2 para evitar ciclos
    for (int d = 0; d < 8; d++){
        ans += floodfill(r + dr[d], c + dc[d], c1, c2);
    }
    return ans;          //El codigo es limpio porque
        usamos dr[] y dc[]
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin >> R; cin >> C;
    cout << floodfill(2, 1, 'W', '.');
}
```

3.7 Algoritmo Kosajaru

```cpp
void Kosaraju(int u, int pass) {
    dfs_num[u] = 1;
    vii &neighbor = (pass == 1) ? AL[u] : AL_T[u];
    for (auto &[v, w] : neighbor)
        if (dfs_num[v] == UNVISITED)
            Kosaraju(v, pass);
    S.push_back(u);
}

int main(){
    S.clear();
    dfs_num.assign(N, UNVISITED);
    for (int u = 0; u < N; ++u)
        if (dfs_num[u] == UNVISITED)
            Kosaraju(u, 1);
    numSCC = 0;
    dfs_num.assign(N, UNVISITED);
    for (int i = N-1; i >= 0; --i)
        if (dfs_num[S[i]] == UNVISITED)
            ++numSCC, Kosaraju(S[i], 2);
    printf("There are %d SCCs\n", numSCC);
}
```

3.8 Dijkstra

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> ii;
typedef vector<ii> vii;
typedef vector<int> vi;
typedef vector<long long> vl;

vi dijkstra(vector<vii> &adj, int s, int V){
    vi dist(V+1, INT_MAX); dist[s] = 0;
    priority_queue<ii, vii, greater<ii> > pq; pq.push(ii
        (0, s));
    while(!pq.empty()){
        ii front = pq.top(); pq.pop();
        int d = front.first, u = front.second;
        if (d > dist[u]) continue;

        for (int j = 0; j < (int)adj[u].size(); j++){
            ii v = adj[u][j];
            if (dist[u] + v.second < dist[v.first]){
                dist[v.first] = dist[u] + v.second;
                pq.push(ii(dist[v.first], v.first));
            }
        }
    }
    return dist;
}
```

3.9 Bellman Ford

```cpp
void bellman_ford(){
    vi dist(V, INF); dist[s] = 0;
    for (int i = 0; i < V-1; ++i)
        for (int u = 0; u < V; ++u)
            if (dist[u] != INF)
                for (auto &[v, w] : adj[u])
                    dist[v] = min(dist[v], dist[u]+w);
}
```

3.10 Floyd Warshall

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

```
typedef long long ll;
typedef pair<int, int> ii;
typedef vector<ii> vii;
typedef vector<int> vi;
typedef vector<long long> vl;
int dr[] = {1, 1, 0, -1, -1, -1, 0, 1};
int dc[] = {0, 1, 1, 1, 0, -1, -1, -1};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int V; cin >> V;
    vector<vi> adjMat(V+1, vi(V+1));
    //Condicion previa: adjMat[i][j] contiene peso de la
       arista (i, j)
    //o INF si no existe esa arista
    for (int k = 0; k<V; k++)
        for (int i = 0; i<V; i++)
            for (int j = 0; j<V; j++)
                adjMat[i][j] = min(adjMat[i][j], adjMat[i
                   ][k] + adjMat[k][j]);
}
```

3.11 MST Kruskal

```
#include <bits/stdc++.h>
using namespace std;
#define sz(arr) ((int) arr.size())
typedef long long ll;
typedef pair<int, int> ii;
typedef vector<ii> vii;
typedef vector<int> vi;
typedef vector<long long> vl;
const int INF = 1e9;
const ll INFL = 1e18;
const int MOD = 1e9+7;
int dirx[4] = {0,-1,1,0};
int diry[4] = {-1,0,0,1};
int dr[] = {1, 1, 0, -1, -1, -1, 0, 1};
int dc[] = {0, 1, 1, 1, 0, -1, -1, -1};

class UnionFind{
    private: vi p, rank;
    public:
        UnionFind(int N){
            rank.assign(N, 0);
            p.assign(N, 0);
            for (int i = 0; i<N; i++) p[i] = i;
        }
        int findSet(int i) {return (p[i] == i) ? i : (p[i
           ] = findSet(p[i]));}
        bool isSameSet(int i, int j) {return findSet(i)
           == findSet(j);}
        void unionSet(int i, int j){
            if (!isSameSet(i, j)){
                int x = findSet(i), y = findSet(j);
                if (rank[x] > rank[y]) p[y] = x;
                else {p[x] = y;
                if (rank[x] == rank[y]) rank[y]++;
                }
            }
        }
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    ios::sync_with_stdio(false);
    cin.tie(0);

    int n, m;
    cin >> n >> m;
    vector<pair<int, ii>> adj;

    for (int i = 0; i<m; i++){
        int x, y, w; cin >> x >> y >> w;
        adj.push_back(make_pair(w, ii(x, y)));
    }

    sort(adj.begin(), adj.end());

    int mst_costo = 0, tomados = 0;
    UnionFind UF(n);
    for (int i = 0; i<m && tomados < n-1; i++){
        pair<int, ii> front = adj[i];
        if (!UF.isSameSet(front.second.first, front.
           second.second)){
            tomados++;
            mst_costo += front.first;
            UF.unionSet(front.second.first, front.second.
               second);
        }
    }
    cout << mst_costo;
}
```

3.12 Shortest Path Faster Algorithm

```
ll spfa(vector<vii>& adj, ll s, ll n) {
    vl d(n+1, INFL);
    vector<bool> inqueue(n, false);
    queue<ll> q;

    d[s] = 0;
    q.push(s);
    inqueue[s] = true;
    while (!q.empty()) {
        ll v = q.front();
        q.pop();
```

```
            inqueue[v] = false;
            for (auto edge : adj[v]) {
                ll to = edge.first;
                ll len = edge.second;
                if (d[v] + len < d[to]) {
                    d[to] = d[v] + len;
                    if (!inqueue[to]) {
                        q.push(to);
                        inqueue[to] = true;
                    }
                }
            }
        }
        return d[n];
    }
```

4 Matematicas

4.1 Descomposicion primos

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> ii;
typedef vector<ii> vii;
typedef vector<int> vi;
typedef vector<long long> vl;

ll _sieve_size;
bitset<10000010> bs;
vl p;
void sieve(ll upperbound) {
    _sieve_size = upperbound+1;
    bs.set();
    bs[0] = bs[1] = 0;
    for (ll i = 2; i < _sieve_size; ++i) if (bs[i]) {
        for (ll j = i*i; j < _sieve_size; j += i) bs[j] =
            0;
        p.push_back(i);
    }
}

vl primeFactors(ll N) {
    vl factors;
    for (int i = 0; (i < (int)p.size()) && (p[i]*p[i] <=
       N); ++i)
        while (N%p[i] == 0) {
            N /= p[i];
            factors.push_back(p[i]);
        }
    if (N != 1) factors.push_back(N);
    return factors;
}
int main(){
    sieve(10000000);
    vl r;
    r = primeFactors((1LL<<31)-1);
    for (auto &pf : r) cout << "> " << pf << "\n";
    cout << "\n";
    r = primeFactors(136117223861LL);
    for (auto &pf : r) cout << "> " << pf << "\n";
    cout << "\n";
    r = primeFactors(5000000035LL);
    for (auto &pf : r) cout << "> " << pf << "\n";
    cout << "\n";
    r = primeFactors(142391208960LL);
    for (auto &pf : r) cout << "> " << pf << "\n";
    cout << "\n";
    r = primeFactors(100000380000361LL);
    for (auto &pf : r) cout << "> " << pf << "\n";
}

//Variantes del algoritmo

//Contar el numero de factores primos de N
int numPF(ll N) {
    int ans = 0;
    for (int i = 0; (i < (int)p.size()) && (p[i]*p[i] <=
       N); ++i)
        while (N%p[i] == 0) { N /= p[i]; ++ans; }
    return ans + (N != 1);
}

//Contar el numero de divisores de N
int numDiv(ll N) {
    int ans = 1; // start from ans = 1
    for (int i = 0; (i < (int)p.size()) && (p[i]*p[i] <=
       N); ++i) {
        int power = 0; // count the power
        while (N%p[i] == 0) { N /= p[i]; ++power; }
        ans *= power+1; // follow the formula
    }
    return (N != 1) ? 2*ans : ans; // last factor = N^1
}

//Suma de los divisores de N
ll sumDiv(ll N) {
    ll ans = 1; // start from ans = 1
    for (int i = 0; (i < (int)p.size()) && (p[i]*p[i] <=
       N); ++i) {
        ll multiplier = p[i], total = 1;
        while (N%p[i] == 0) {
            N /= p[i];
            total += multiplier;
            multiplier *= p[i];
    } // total for
    ans *= total; // this prime factor
    }
```

```
    if (N != 1) ans *= (N+1); // N^2-1/N-1 = N+1
    return ans;
}

//EulerPhi(N): contar el numero de enteros positivos < N
   que son primos relativos a N.
ll EulerPhi(ll N) {
    ll ans = N; // start from ans = N
    for (int i = 0; (i < (int)p.size()) && (p[i]*p[i] <=
       N); ++i) {
        if (N%p[i] == 0) ans -= ans/p[i]; // count unique
        while (N%p[i] == 0) N /= p[i]; // prime factor
    }
    if (N != 1) ans -= ans/N; // last factor
    return ans;
}

//Criba modificada
/*
Si hay que determinar el numero de factores primos para
   muchos (o un rango) de enteros.
La mejor solucion es el algoritmo de criba modificada O(N
    log log N)
*/
int numDiffPFarr[MAX_N+10] = {0}; // e.g., MAX_N = 10^7
for (int i = 2; i <= MAX_N; ++i)
    if (numDiffPFarr[i] == 0) // i is a prime number
        for (int j = i; j <= MAX_N; j += i)
            ++numDiffPFarr[j]; // j is a multiple of i

//Similar para EulerPhi
int EulerPhi[MAX_N+10];
for (int i = 1; i <= MAX_N; ++i) EulerPhi[i] = i;
for (int i = 2; i <= MAX_N; ++i)
    if (EulerPhi[i] == i) // i is a prime number
        for (int j = i; j <= MAX_N; j += i)
            EulerPhi[j] = (EulerPhi[j]/i) * (i-1);
```

4.2 Comprobar primos

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> ii;
typedef vector<ii> vii;
typedef vector<int> vi;
typedef vector<long long> vl;

ll _sieve_size;
bitset<10000010> bs;
vl primos;
void sieve(ll upperbound) {
    _sieve_size = upperbound+1;
    bs.set();
    bs[0] = bs[1] = 0;
    for (ll i = 2; i < _sieve_size; ++i) if (bs[i]) {
        for (ll j = i*i; j < _sieve_size; j += i) bs[j] =
            0;
        primos.push_back(i);
    }
}
bool isPrime(ll N) {
    if (N < _sieve_size) return bs[N]; // O(1)
    for (int i = 0; i < (int)primos.size() && primos[i]*
       primos[i] <= N; ++i)
        if (N%primos[i] == 0)
            return false;
    return true;
}
int main(){
    sieve(10000000);
    cout << isPrime(2147483647) << "\n";
    cout << isPrime(136117223861LL) << "\n";
    cout << isPrime(1e9 + 7) << "\n";
}
```

4.3 GCD y LCM

```
//O(log10 n) n == max(a, b)
int gcd(int a, int b) { return b == 0 ? a : gcd(b, a%b);
   }
int lcm(int a, int b) { return a / gcd(a, b) * b; }
//gcd(a, b, c) = gcd(a, gcd(b, c))
```