Notebook UNosnovatos

Contents

1 C++

1.1 C++ plantilla

```cpp
#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() {
    freopen("file.in", "r", stdin);
    freopen("file.out", "w", stdout);
    ios::sync_with_stdio(false);
    cin.tie(0);

}
```

2 Grafos

2.1 DFS

```cpp
#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]);
        }

    }
}
```

2.2 BFS

```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;
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
            }
        }
    }
}
```

2.3 Orden Topologico

```
#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;
}
```

2.4 Floodfill

```
//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;
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', '.');
}
```

2.5 Dijkstra

```
#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;
}
```

2.6 Floyd Warshall

```
#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]);
}
```

2.7 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 Matematicas

3.1 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);
```


3.2 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";
}
```

3.3 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))
```