

Graphs

Depth First Search

Q4.

Click submit

Q6.

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

```
using namespace std;
```

```
const int MAXN = 200005; // Maximum number of vertices
```

```
vector<int> adj[MAXN]; // Adjacency list
```

```
vector<bool> visited(MAXN, false); // Visited array
```

```
vector<int> reachable; // Stores reachable nodes
```

```
void dfs(int v) {
```

```
    // Mark the current node as visited
```

```
    visited[v] = true;
```

```
    reachable.push_back(v); // Add the node to reachable list
```

```
    // Iterate over all neighbours of v
```

```
    for (auto u : adj[v]) {
```

```
        // If the neighbour has not been visited, call dfs on it
```

```
        if (!visited[u]) {
```

```
            dfs(u);
```

```
        }
```

```
    }
```

```
}
```

```
int main() {
```

```
    int N, M;
```

```
    cin >> N >> M;
```

```
    // Reading edges
```

```
    for (int i = 0; i < M; ++i) {
```

```
        int A, B;
```

```
        cin >> A >> B;
```

```
        adj[A].push_back(B); // Directed edge from A to B
```

```
    }
```

```

// Perform DFS from node 1
dfs(1);

// Sort the reachable nodes
sort(reachable.begin(), reachable.end());

// Output the reachable nodes
for (int node : reachable) {
    cout << node << " ";
}
cout << endl;

return 0;
}

```

Q7.

```

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

void dfs(int v, vector<vector<int>> &adj, vector<int> &component, int id) {
    component[v] = id; // Mark the component id for this vertex
    for (int u : adj[v]) {
        if (component[u] == -1) { // If the neighbor is not yet visited
            dfs(u, adj, component, id);
        }
    }
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    int T;
    cin >> T;

    while (T--) {
        int N, M;
        cin >> N >> M;

        vector<vector<int>> adj(N); // Adjacency list
    }
}

```

```

for (int i = 0; i < M; ++i) {
    int A, B;
    cin >> A >> B;
    adj[A].push_back(B);
    adj[B].push_back(A); // Since roads are bidirectional
}

vector<int> component(N, -1); // Component id for each vertex
int id = 0;

// Find all connected components
for (int i = 0; i < N; ++i) {
    if (component[i] == -1) {
        dfs(i, adj, component, id);
        ++id;
    }
}

int Q;
cin >> Q;

while (Q--) {
    int X, Y;
    cin >> X >> Y;
    if (component[X] == component[Y]) {
        cout << "YO\n";
    } else {
        cout << "NO\n";
    }
}

return 0;
}

```

Breadth-First Search

Q3.

Click Submit

Q4.

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

```
using namespace std;
```

```
const int N = 2e5 + 1;
```

```
vector<int> adj[N]; // Adjacency list
```

```
vector<bool> vis(N, 0); // Visited array
```

```
void bfs(int v) {
```

```
    queue<int> q;
```

```
    q.push(v);
```

```
    vis[v] = true;
```

```
    while (!q.empty()) {
```

```
        int node = q.front();
```

```
        q.pop();
```

```
        for (int neighbor : adj[node]) {
```

```
            if (!vis[neighbor]) {
```

```
                vis[neighbor] = true;
```

```
                q.push(neighbor);
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

```
int main() {
```

```
    ios_base::sync_with_stdio(false);
```

```
    cin.tie(nullptr);
```

```
    int n, m;
```

```
    cin >> n >> m;
```

```
    // Creating adjacency list
```

```
    for (int i = 0; i < m; ++i) {
```

```
        int a, b;
```

```
        cin >> a >> b;
```

```
        adj[a].push_back(b); // Directed edge
```

```
    }
```

```

bfs(1); // Start bfs from source: 1

vector<int> reachable;
for (int i = 1; i <= n; ++i) {
    if (vis[i]) // A node is reachable if it is marked visited
        reachable.push_back(i);
}

sort(reachable.begin(), reachable.end()); // Sort reachable nodes

for (int node : reachable) {
    cout << node << " ";
}
cout << endl;

return 0;
}

```

Q6.

```

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

const int INF = 1e9;
const int N_MAX = 1e5 + 5;

vector<pair<int, int>> adj[N_MAX]; // adjacency list to store edges

int minEdgesToReverse(int n, int m) {
    vector<int> dist(n + 1, INF); // distance array initialized to INF
    deque<int> dq; // deque for 0-1 BFS

    // Start BFS from node 1
    dq.push_back(1);
    dist[1] = 0;

    while (!dq.empty()) {
        int u = dq.front();
        dq.pop_front();
    }
}

```

```

    for (auto [v, w] : adj[u]) {
        if (dist[u] + w < dist[v]) {
            dist[v] = dist[u] + w;
            if (w == 0) {
                dq.push_front(v); // 0-weight edges are processed first
            } else {
                dq.push_back(v); // 1-weight edges are processed last
            }
        }
    }
}

return (dist[n] == INF) ? -1 : dist[n];
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    int n, m;
    cin >> n >> m;

    for (int i = 0; i < m; ++i) {
        int x, y;
        cin >> x >> y;
        adj[x].emplace_back(y, 0); // original edge
        adj[y].emplace_back(x, 1); // reversed edge
    }

    int result = minEdgesToReverse(n, m);
    cout << result << "\n";

    return 0;
}

```

Q7.

```

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

```

```

int main() {

```

```

ios::sync_with_stdio(false);
cin.tie(nullptr);

string s;
cin >> s;

int n = s.size();
if (n == 1) {
    cout << 0 << "\n";
    return 0;
}

vector<int> adj[10]; // adjacency list for digits 0-9
for (int i = 0; i < n; ++i) {
    adj[s[i] - '0'].push_back(i);
}

vector<int> dist(n, INT_MAX);
dist[0] = 0;

deque<int> dq;
dq.push_back(0);

while (!dq.empty()) {
    int i = dq.front();
    dq.pop_front();

    // Move to i-1
    if (i > 0 && dist[i - 1] > dist[i] + 1) {
        dist[i - 1] = dist[i] + 1;
        dq.push_back(i - 1);
    }

    // Move to i+1
    if (i < n - 1 && dist[i + 1] > dist[i] + 1) {
        dist[i + 1] = dist[i] + 1;
        dq.push_back(i + 1);
    }
}

// Jump to all indices with the same digit

```

```

    int digit = s[i] - '0';
    for (int j : adj[digit]) {
        if (dist[j] > dist[i] + 1) {
            dist[j] = dist[i] + 1;
            dq.push_back(j);
        }
    }

    // Clear the list of current digit to avoid reprocessing
    adj[digit].clear();
}

cout << dist[n - 1] << "\n";
return 0;
}

```

Q8.

Dijkstra

Q3.

Click Submit

Q5.

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

```
using namespace std;
```

```
const int N = 1001; // As per the problem constraints
```

```
const long long INF = 1e18;
```

```
vector<pair<int, int>> adj[N];
```

```
long long d[N];
```

```
bool vis[N];
```

```
void dijkstra(int source, int n) {
```

```
    // Priority queue to store {distance, vertex}
```

```
    priority_queue<pair<long long, int>, vector<pair<long long, int>>, greater<pair<long long, int>>> min_heap;
```

```
    // Initialize distances
```

```
    for (int i = 1; i <= n; ++i) {
```

```
        d[i] = INF;
```

```
        vis[i] = false;
```

```
    }
```



```

d[source] = 0;
min_heap.push({0, source});

while (!min_heap.empty()) {
    int u = min_heap.top().second;
    min_heap.pop();

    if (vis[u]) continue;
    vis[u] = true;

    for (auto edge : adj[u]) {
        int v = edge.first;
        int weight = edge.second;

        if (!vis[v] && d[u] + weight < d[v]) {
            d[v] = d[u] + weight;
            min_heap.push({d[v], v});
        }
    }
}

int main() {
    int n, m;
    cin >> n >> m;

    for (int i = 0; i < m; ++i) {
        int u, v, w;
        cin >> u >> v >> w;
        adj[u].push_back({v, w});
        adj[v].push_back({u, w}); // Undirected graph
    }

    dijkstra(1, n);

    // Output distances
    for (int i = 1; i <= n; ++i) {
        if (d[i] == INF) cout << "-1 ";
        else cout << d[i] << " ";
    }
    cout << "\n";

    return 0;
}

```

Q6.

[Click Submit](#)

Q7.

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

```
const int N = 200001; // Maximum number of vertices
const long long INF = 1e18;
```

```
vector<pair<int, int>> adj[N];
long long d[N];
bool vis[N];
```

```
void dijkstra(int source, int n) {
    // Priority queue to store {distance, vertex}
    priority_queue<pair<long long, int>, vector<pair<long long, int>>, greater<pair<long
long, int>>> min_heap;
```

```
    // Initialize distances
    fill(d, d + n + 1, INF);
    fill(vis, vis + n + 1, false);
    d[source] = 0;
    min_heap.push({0, source});
```

```
    while (!min_heap.empty()) {
        int u = min_heap.top().second;
        min_heap.pop();
```

```
        if (vis[u]) continue;
        vis[u] = true;
```

```
        for (auto& edge : adj[u]) {
            int v = edge.first;
            int weight = edge.second;
```

```
            if (!vis[v] && d[u] + weight < d[v]) {
                d[v] = d[u] + weight;
                min_heap.push({d[v], v});
            }
        }
    }
```

```
}
```

```
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
```

```
    int n, m;
    cin >> n >> m;
```

```

for (int i = 0; i < m; ++i) {
    int u, v, w;
    cin >> u >> v >> w;
    adj[u].push_back({v, w});
    adj[v].push_back({u, w}); // Undirected graph
}

dijkstra(1, n);

// Output distances
for (int i = 1; i <= n; ++i) {
    if (d[i] == INF) cout << "-1 ";
    else cout << d[i] << " ";
}
cout << "\n";

return 0;
}

```

Q8.(Chnage the lang to python)

<https://codefile.io/f/3ykheD1Trl>

Q9.(Chnage the lang to python)

<https://codefile.io/f/1Pnhj6f0Un>

Bellman Ford

Q5.

```

#include <iostream>
#include <vector>
#include <queue>
#include <limits>
using namespace std;

const long long INF = numeric_limits<long long>::max();

void dijkstra(int source, int n, vector<vector<pair<int, int>>>& adj) {
    vector<long long> dist(n + 1, INF);
    priority_queue<pair<long long, int>, vector<pair<long long, int>>, greater<pair<long long, int>>> min_heap;

    dist[source] = 0;
    min_heap.push({0, source});

    while (!min_heap.empty()) {
        int u = min_heap.top().second;
        long long u_dist = min_heap.top().first;
        min_heap.pop();
    }
}

```

```

        if (u_dist > dist[u]) continue;

        for (auto& edge : adj[u]) {
            int v = edge.first;
            int weight = edge.second;

            if (dist[u] + weight < dist[v]) {
                dist[v] = dist[u] + weight;
                min_heap.push({dist[v], v});
            }
        }
    }

    for (int i = 1; i <= n; ++i) {
        if (dist[i] == INF)
            cout << "-1 ";
        else
            cout << dist[i] << " ";
    }
    cout << endl;
}

int main() {
    int n, m;
    cin >> n >> m;

    vector<vector<pair<int, int>>> adj(n + 1);

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

    dijkstra(1, n, adj);

    return 0;
}

```

DSU

Q5.

```

#include <iostream>
#include <vector>
using namespace std;

```

```

struct DSU {

```

```

vector<int> parent;
vector<int> size;

DSU(int n) {
    parent.resize(n + 1);
    size.resize(n + 1);
    for (int i = 1; i <= n; ++i) {
        parent[i] = i;
        size[i] = 1;
    }
}

int find_parent(int v) {
    if (parent[v] == v)
        return v;
    return parent[v] = find_parent(parent[v]); // Path compression
}

void merge_set(int a, int b) {
    a = find_parent(a);
    b = find_parent(b);
    if (a != b) {
        if (size[b] > size[a])
            swap(a, b);
        parent[b] = a;
        size[a] += size[b];
    }
}

bool same_set(int a, int b) {
    return find_parent(a) == find_parent(b);
}

};

int main() {
    int n, q;
    cin >> n >> q;

    DSU dsu(n);

    for (int i = 0; i < q; ++i) {
        int t, a, b;
        cin >> t >> a >> b;
        if (t == 1) {
            dsu.merge_set(a, b);
        } else if (t == 2) {
            if (dsu.same_set(a, b))
                cout << "YES\n";

```

```
    else
        cout << "NO\n";
    }
}

return 0;
}
```

Q7.change the prgrm language to python

<https://codefile.io/f/3zquSZNCq1>