***Week – 8 (******06.06.2021 – 12.06.2021)***

***CODES IN PDF***

1. ***All Paths From Source to Target:***

class Solution {

public:

void dfs(vector<vector<int>>& graph, vector<vector<int>>& res, vector<int> path, int node)

{

path.push\_back(node);

if(node == graph.size()-1) res.push\_back(path);

else for(int neigh: graph[node]) dfs(graph, res, path, neigh);

path.pop\_back();

}

vector<vector<int>> allPathsSourceTarget(vector<vector<int>>& graph) {

vector<vector<int>> res;

dfs(graph, res, {}, 0);

return res;

}

};

1. ***Cycle in Directed Graph:***

bool dfs(map<int,vector<int>> &m, int s, vector<bool> &visited,vector<bool> &temp)

{

visited[s]=true;

temp[s]=true;

for(int u:m[s])

{

if(visited[u]==false && dfs(m,u,visited,temp)==true)

return true;

else if(temp[u]==true)

return true;

}

temp[s]=false;

return false;

}

int Solution::solve(int A, vector<vector<int> > &B) {

map<int,vector<int>> m;

vector<bool> visited(A, false);

vector<bool> temp(A, false);

for(auto i : B)

m[i[0]].push\_back(i[1]);

for(int i=0;i<A;i++)

{

if(visited[i]==false)

if(dfs(m,i,visited,temp)==true)

return 1;

}

return 0;

}

1. ***Cycle in Undirected Graph:***

int findP(int check [],int x)

{

while(check[x]!=-1) x=check[x];

return x;

}

int Solution::solve(int A, vector<vector<int> > &B) {

int check[A+1], x, y, i;

memset(check,-1,sizeof(check));

for(i=0;i<B.size();i++)

{

x=findP(check,B[i][0]);

y=findP(check,B[i][1]);

if(x==y) return 1;

check[x]=y;

}

return 0;

}

1. ***Floyd : City of Blinding Lights:***

#include <bits/stdc++.h>

using namespace std;

int main() {

    /\* Enter your code here. Read input from STDIN. Print output to STDOUT \*/

    long long wt[405][405] = {INT\_MAX};

    int n, m;

    cin>>n>>m;

    int i, x, y,k,j, w;

    for(i=0; i<n; i++)

    {

        for(j=0; j<n; j++)

            wt[i][j] = INT\_MAX;

        wt[i][i] = 0;

    }

    for(i=0; i<m; i++)

    {

        cin>>x>>y>>w;

        wt[x-1][y-1] = w;

    }

    long long dist[n+1][n+1];

    for(i=0; i<n; i++)

    {

        for(j=0; j<n; j++)

            dist[i][j] = wt[i][j];

    }

     for(k=0; k<n; k++)

     {

         for(i=0; i<n; i++)

            for(j=0; j<n; j++)

                if(dist[i][k]!=INT\_MAX && dist[k][j] != INT\_MAX && (dist[i][j] > dist[i][k]+dist[k][j]))

                    dist[i][j] = dist[i][k]+dist[k][j];

    }

    int q;

    cin>>q;

    while(q--)

    {

        cin>>x>>y;

        if(dist[x-1][y-1]==INT\_MAX)

            cout<<"-1"<<endl;

        else

            cout<<dist[x-1][y-1]<<endl;

    }

    return 0;

}

1. ***Kruskal (MST): Really Special Subtree:***

#include <bits/stdc++.h>

using namespace std;

const int MAX = 1e6-1;

int root[MAX];

int g\_nodes, g\_edges, g\_from, g\_to, g\_weight, min\_cost;

pair<int,pair<int,int>> p[MAX];

int parent(int a)

{

    while(root[a] != a)

    {

        root[a] = root[root[a]];

        a = root[a];

    }

    return a;

}

void union\_find(int a, int b)

{

    int d = parent(a);

    int e = parent(b);

    root[d] = root[e];

}

int kruskals()

{

    int a, b, cost, mincost=0, i;

    for(i=0; i<g\_edges; i++)

    {

        a = p[i].second.first;

        b = p[i].second.second;

        cost = p[i].first;

        if(parent(a) != parent(b))

        {

            mincost += cost;

            union\_find(a, b);

        }

    }

    return mincost;

}

int main()

{

    int i;

    cin>>g\_nodes>>g\_edges;

    for(i=0; i<MAX; i++)

        root[i] = i;

    for(i=0; i<g\_edges; i++)

    {

        cin>>g\_from>>g\_to>>g\_weight;

        p[i] = make\_pair(g\_weight, make\_pair(g\_from, g\_to));

    }

    sort(p, p+g\_edges);

    min\_cost = kruskals();

    cout<<min\_cost;

}

1. ***Number of Islands:***

class Solution {

public:

int dfs(vector<vector<char>>& grid, int i, int j)

{

return (i >= 0 && i < grid.size() && j >= 0 && j < grid[0].size() && grid[i][j] == '1') ? grid[i][j] = 'v', 1 + dfs(grid, i - 1, j) + dfs(grid, i + 1, j) + dfs(grid, i, j - 1) + dfs(grid, i, j + 1) : 0;

}

int numIslands(vector<vector<char>>& grid) {

if(grid.size() == 0|| grid[0].size() == 0) return 0;

int islands = 0, i, j;

for(i=0; i<grid.size(); i++)

for(j=0; j<grid[0].size(); j++)

if(dfs(grid, i, j) > 0) islands++;

return islands;

}

};

1. ***Shortest Path Visiting All Nodes:***

class Solution {

public:

int result;

void dfs(vector<vector<int>>& graph, vector<int>& cur, vector<int>& visited, int count) {

if (result == graph.size()) return;

if (count == graph.size()) {

result = min(result, int(cur.size()));

return;

}

if (cur.size() >= result) return;

int pos = cur.back();

int minv = INT\_MAX;

for (auto n : graph[pos]) minv = min(minv, visited[n]);

for (auto n : graph[pos]) {

if (visited[n] == minv) {

cur.push\_back(n);

if (!visited[n]) count++;

visited[n]++;

dfs(graph, cur, visited, count);

visited[n]--;

if (!visited[n]) count--;

cur.pop\_back();

}

}

}

int shortestPathLength(vector<vector<int>>& graph) {

result = INT\_MAX;

vector<int> cur;

vector<int> visited(graph.size(), 0);

int count = 0;

for (int i = 0; i < graph.size(); i++) {

cur.push\_back(i);

visited[i]++;

count++;

dfs(graph, cur, visited, count);

count--;

visited[i]--;

cur.pop\_back();

}

return result - 1;

}

};

1. ***Network Delay Time (Dijkstra Algorithm):***

class Solution {

public:

int networkDelayTime(vector<vector<int>>& times, int n, int k) {

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

for(auto t : times) g[t[0]].push\_back({ t[1], t[2] });

set<pair<int, int>> Set;

vector<int> dist(n+1, INT\_MAX);

dist[k] = 0;

Set.insert({ 0, k });

while(!Set.empty()) {

auto [d, u] = \*(begin(Set));

Set.erase(begin(Set));

for(auto [v, w] : g[u]) {

if(dist[v] > d + w) {

if(dist[v] != INT\_MAX)

Set.erase(Set.find({ dist[v], v }));

dist[v] = d + w;

Set.insert({ dist[v], v });

}

}

}

int mx = \*max\_element(begin(dist) + 1, end(dist));

return mx == INT\_MAX ? -1 : mx;

}

};

1. ***Network Delay Time (Bellman Ford):***

class Solution {

public:

int networkDelayTime(vector<vector<int>>& times, int n, int k) {

vector<int> dist(n + 1, INT\_MAX);

dist[k] = 0;

for (int i = 0; i < n; i++) {

for (vector<int> e : times) {

int u = e[0], v = e[1], w = e[2];

if (dist[u] != INT\_MAX && dist[v] > dist[u] + w) {

dist[v] = dist[u] + w;

}

}

}

int maxwait = 0;

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

maxwait = max(maxwait, dist[i]);

return maxwait == INT\_MAX ? -1 : maxwait;

}

};

1. ***Network Delay Time (Warshall Ford):***

class Solution {

public:

int networkDelayTime(vector<vector<int>>& times, int N, int K) {

long long int mat[ N+1] [N+1],i,j,m,n=times.size();

for(i=0;i<=N;i++)

for(j=0;j<=N;j++)

mat[i][j] = INT\_MAX;

for(i=0;i<=N;i++)

mat[i][i] = 0;

for(i=0;i<n;i++)

{

int u=times[i][0] , v=times[i][1] , w=times[i][2];

mat[u][v]=w;

}

for(m=1;m<=N;m++)

for(i=1;i<=N;i++)

for(j=1;j<=N;j++)

mat[i][j]=min(mat[i][j],mat[i][m]+mat[m][j]);

m=INT\_MIN;

for(j=1;j<=N;j++)

m=max(mat[K][j],m);

if(m>=INT\_MAX) return -1;

return m;

}

};

1. ***Possible Bipartition:***

class Solution {

public:

bool dfs(vector<vector<int>>& v,vector<int>& vis,int i)

{

int x,j,color;

queue<int> q;

q.push(i);

vis[i]=1;

while(!q.empty())

{

x=q.front();

q.pop();

if(vis[x]==1) color=2;

else color=1;

for(j=0;j<v[x].size();j++)

{

if(vis[v[x][j]]==0)

{

q.push(v[x][j]);

vis[v[x][j]]=color;

}

else if(vis[v[x][j]]==vis[x]) return false;

}

}

return true;

}

bool possibleBipartition(int n, vector<vector<int>>& dislikes) {

int i;

vector<int> vis(n+1,0);

vector<vector<int>> v(n+1);

for(auto x:dislikes)

{

v[x[0]].push\_back(x[1]);

v[x[1]].push\_back(x[0]);

}

for(i=1;i<=n;i++)

if(vis[i]==0)

if(dfs(v,vis,i)==false)

return false;

return true;

}

};

1. ***Is Graph Bipartite?:***

class Solution {

public:

bool isBipartite(vector<vector<int>>& graph) {

int i, cur,size = graph.size();

vector<int> color(size, 0);

for (i=0; i<size; i++)

{

if (color[i] == 0)

{

color[i] = 1;

stack<int> Stack;

Stack.push(i);

while (!Stack.empty())

{

cur = Stack.top(); Stack.pop();

for (const int& nei : graph[cur])

{

if (color[nei] == color[cur]) return false;

if (color[nei] == 0)

{

color[nei] = - color[cur];

Stack.push(nei);

}

}

}

}

}

return true;

}

};

1. ***Graph Connectivity With Threshold:***

class Solution {

public:

vector<int> g;

int find(int i)

{

if(g[i] == i) return i;

g[i] = find(g[i]);

return g[i];

}

void merge(int x, int y)

{

x = find(x);

y = find(y);

g[y] = x;

}

vector<bool> areConnected(int n, int threshold, vector<vector<int>>& queries) {

g.resize(n+1, 0);

int i, k, cur;

for(i=1; i<=n; i++) g[i] = i;

for(k = threshold+1; k \* 2 <= n; k++)

{

cur = k \* 2;

while(cur <= n)

{

merge(k, cur);

cur += k;

}

}

vector<bool> res;

for(auto v : queries) res.push\_back(find(v[0]) == find(v[1]));

return res;

}

};

1. ***Largest Color Value in a Directed Graph:***

class Solution {

public:

bool cycle = false;

void dfs(int v, vector<vector<int>>& edge, string& color, vector<int>& m, vector<vector<int>>& dp)

{

int i;

m[v] = 1;

for(int u : edge[v])

{

if(m[u] == 0)

{

dfs(u, edge, color, m, dp);

for(i = 0; i < 26; i++)

dp[v][i] = max(dp[v][i], dp[u][i]);

}

else if(m[u] == 1) cycle = true;

else if(m[u] == 2)

for(i = 0; i < 26; i++)

dp[v][i] = max(dp[v][i], dp[u][i]);

}

m[v] = 2;

dp[v][color[v] - 'a']++;

}

int largestPathValue(string colors, vector<vector<int>>& edges) {

int n = colors.size(), i, ans, v;

vector<vector<int>> edge(n);

for(vector<int>& p : edges) {

edge[p[0]].push\_back(p[1]);

}

vector<int> m(n);

vector<vector<int>> dp(n, vector<int> (26));

for(i = 0; i < n; i++)

if(m[i] == 0)

dfs(i, edge, colors, m, dp);

if(cycle) return -1;

ans = 0;

for(v = 0; v < n; v++)

for(i = 0; i < 26; i++)

ans = max(ans, dp[v][i]);

return ans;

}

};