***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;

}