GRAPH SHEET

1> Dijkstras

```
int maxdis;


    vector <int> dijkstra(int V, vector<vector<int>> adj[], int S)  // adj is
adjancy list with first index to, second index length of edge
    {

        vector<int> temp;
        if(!V)
            return temp;

        maxdis=V*10001+10;   // max distance= no. of nodes*max length of any bridge


        vector<int> ans(V,maxdis);
        ans[S]=0;

        queue<int> q;
        q.push(S);

        while(!q.empty()){
            int t=q.front();

            q.pop();

            for(int i=0;i<(adj)[t].size();i++)
            {
                if(ans[(adj)[t][i][0]] > (adj)[t][i][1]+ans[t])
                 { ans[(adj)[t][i][0]] = (adj)[t][i][1]+ans[t];
                   q.push((adj)[t][i][0]);}


            }
        }
        return ans;
    }
```

2> Bellman Ford, minimum distance in case of negative edges O(VE)

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

```
struct node {
    int u;
    int v;
    int wt;
    node(int first, int second, int weight) {
        u = first;
        v = second;
        wt = weight;
    }
};

int main(){
    int N,m;
    cin >> N >> m;
    vector<node> edges;
    for(int i = 0;i<m;i++) {
        int u, v, wt;
        cin >> u >> v >> wt;
        edges.push_back(node(u, v, wt));
    }

    int src;
    cin >> src;


    int inf = 10000000; //max distance possible
    vector<int> dist(N, inf);

    dist[src] = 0;

    for(int i = 1;i<=N-1;i++) {
        for(auto it: edges) {
            if(dist[it.u] + it.wt < dist[it.v]) {
                dist[it.v] = dist[it.u] + it.wt;
            }
        }
    }

    int fl = 0;
    for(auto it: edges) {
        if(dist[it.u] + it.wt < dist[it.v]) {
            cout << "Negative Cycle";
            fl = 1;
            break;
        }
    }

    if(!fl) {
        for(int i = 0;i<N;i++) {
            cout << i << " " << dist[i] << endl;  //printing distances
```

```
        }
    }


    return 0;
}
```

3> Floyd Warshall, min distance between every pair, O(V3)

```
void floydWarshall(int graph[][V])
{

    int dist[V][V], i, j, k;

    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];  //distance stores minimum dis between i and
j, we need adjancy matrix here


    for (k = 0; k < V; k++) {

        for (i = 0; i < V; i++) {

            for (j = 0; j < V; j++) {

                if (dist[i][j] > (dist[i][k] + dist[k][j])
                    && (dist[k][j] != INF
                        && dist[i][k] != INF))
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }


    printSolution(dist);
}
```

2> Topo sort

```
vector<int> topoSort(int V, vector<int> adj[])
        {

           vector<int> indeg(V,0),res;
         for(int i=0;i<V;i++)
         {
             for(auto s:adj[i])
             {
                 indeg[s]++;

             }
         }

          queue<int> q;
          for(int i=0;i<V;i++)
          {
              if(indeg[i]==0)
                q.push(i);
          }

          while(!q.empty())
          {
              int k=q.front();

              q.pop();
             res.push_back(k);
                  for(auto f:adj[k])
                  {
                      indeg[f]--;
                      if(indeg[f]==0)
                      {
                          q.push(f);
                      }
                  }

          }
          return res;
        }
```

4> Check bipartite

```
int visited[101]; //101 is max no.of nodes, given in question

bool dfs(vector<vector<int>>& graph, int x, int parent){
   if(parent==-1)
       visited[x]=1;
```

```
    else
        visited[x]=1-visited[parent];

    for(int i=0;i<graph[x].size();i++)
    {
        if(visited[graph[x][i]]==-1)
        {
            if(dfs(graph, graph[x][i], x))
                    return 1;
        }
        else
        {
            if(visited[graph[x][i]]==visited[x])
                return 1;
        }
    }
    return 0;

}


    bool isBipartite(vector<vector<int>>& graph) {
        memset(visited,-1,sizeof(visited));

        for(int i=0;i<graph.size();i++)
        {
            if(visited[i]==-1)
                if(dfs(graph, i, -1))
                    return 0;
        }
        return 1;
    }
```

5> Articulation points

```
#include <bits/stdc++.h>
using namespace std;
void dfs(int node, int parent, vector<int> &vis, vector<int> &tin, vector<int> &low,
int &timer, vector<int> adj[], vector<int> &isArticulation) {
    vis[node] = 1;
```

```
    tin[node] = low[node] = timer++;
    int child = 0;
    for(auto it: adj[node]) {
        if(it == parent) continue;

        if(!vis[it]) {
            dfs(it, node, vis, tin, low, timer, adj, isArticulation);
            low[node] = min(low[node], low[it]);
            child++;
            if(low[it] >= tin[node] && parent != -1) {
                isArticulation[node] = 1;
            }
        } else {
            low[node] = min(low[node], tin[it]);
        }
    }

    if(parent == -1 && child > 1) {
        isArticulation[node] = 1;
    }
}
int main() {
    int n, m;
    cin >> n >> m;
        vector<int> adj[n];
        for(int i = 0;i<m;i++) {
            int u, v;
            cin >> u >> v;
            adj[u].push_back(v);
            adj[v].push_back(u);
        }

        vector<int> tin(n, -1);
        vector<int> low(n, -1);
        vector<int> vis(n, 0);
        vector<int> isArticulation(n, 0);
        int timer = 0;
        for(int i = 0;i<n;i++) {
            if(!vis[i]) {
                dfs(i, -1, vis, tin, low, timer, adj, isArticulation);
            }
        }

        int ans=0;
        for(int i = 0;i<n;i++) {
            if(isArticulation[i] == 1) {cout << i << endl; ans++;}
        }

        return ans;
}
```

6> Number of bridges and printing them

```
int ans;

void dfs(int node, int parent, vector<int> &vis, vector<int> &tin, vector<int> &low,
int &timer, vector<vector<int>> adj) {
    vis[node] = 1;
    tin[node] = low[node] = timer++;
    for(auto it: adj[node]) {
        if(it == parent) continue;

        if(!vis[it]) {
            dfs(it, node, vis, tin, low, timer, adj);
            low[node] = min(low[node], low[it]);
            if(low[it] > tin[node]) {
                ans++;
                cout << node << " " << it << endl;
            }
        } else {
            low[node] = min(low[node], tin[it]);
        }
    }
}


int solve(vector<vector<int>>& graph) {
    ans=0;
    int n=graph.size();
    vector<int> tin(n, -1);
        vector<int> low(n, -1);
        vector<int> vis(n, 0);
        int timer = 0;
        for(int i = 0;i<n;i++) {
            if(!vis[i]) {
                dfs(i, -1, vis, tin, low, timer, graph);
            }
        }
    return ans;
}
```

7> Shortest path visiting all nodes

```
 struct info
{
  int node; int bitmask;
};

bool hasReach(int x,int v)
{
    int cnt=0;
    for(int i=12;i>=0;i--)
    {
        if(1 & (x>>i))
            cnt++;
    }
    return cnt==v;
}

int shortestPathLength(vector<vector<int>>& graph) {
        int v=graph.size();
        vector<int>adj[v];
        set<pair<int,int>>s; queue<info>q;
        for(int i=0;i<v;i++)
        {
            for(auto it:graph[i])
                adj[i].push_back(it);
        }
        for(int i=0;i<v;i++)
            q.push({i,1<<i});
        int cost=0;
        while(!q.empty())
        {
            int comp=q.size();
            while(comp--)
            {
                info temp=q.front(); q.pop();
                if(hasReach(temp.bitmask,v))
                {
                    cout<<"Sad";
                    return cost;
                }
                s.insert({temp.node,temp.bitmask});
               for(auto it:adj[temp.node])
                {
                   int tbitmask=temp.bitmask | (1<<it);
```

```
                    if(s.find({it,tbitmask})==s.end())
                    {
                        q.push({it,tbitmask});
                        s.insert({it,tbitmask});
                    }
                }
            }
            cost++;
        }
        return -1;
    }
```

8> Krushkal algo for MST

```
#include <iostream>
#include <vector>
#include <utility>
#include <algorithm>

using namespace std;
const int MAX = 1e4 + 5;
int id[MAX], nodes, edges;
pair <long long, pair<int, int> > p[MAX];

void initialize()
{
    for(int i = 0;i < MAX;++i)
        id[i] = i;
}

int root(int x)
{
    while(id[x] != x)
    {
        id[x] = id[id[x]];
        x = id[x];
    }
    return x;
}
```

```
void union1(int x, int y)
{
    int p = root(x);
    int q = root(y);
    id[p] = id[q];
}

long long kruskal(pair<long long, pair<int, int> > p[])
{
    int x, y;
    long long cost, minimumCost = 0;
    for(int i = 0;i < edges;++i)
    {

        x = p[i].second.first;
        y = p[i].second.second;
        cost = p[i].first;

        if(root(x) != root(y))
        {
            minimumCost += cost;
            union1(x, y);
        }
    }
    return minimumCost;
}

int main()
{
    int x, y;
    long long weight, cost, minimumCost;
    initialize();
    cin >> nodes >> edges;
    for(int i = 0;i < edges;++i)
    {
        cin >> x >> y >> weight;
        p[i] = make_pair(weight, make_pair(x, y));
    }

    sort(p, p + edges);
    minimumCost = kruskal(p);
    cout << minimumCost << endl;
    return 0;
}
```

9> Prims algo for MST

```
#include <iostream>
#include <vector>
#include <queue>
#include <functional>
#include <utility>

using namespace std;
const int MAX = 1e4 + 5;  // MAX is no. of nodes
typedef pair<long long, int> PII;
bool marked[MAX];
vector <PII> adj[MAX];

long long prim(int x)
{
    priority_queue<PII, vector<PII>, greater<PII> > Q;
    int y;
    long long minimumCost = 0;
    PII p;
    Q.push(make_pair(0, x));
    while(!Q.empty())
    {
        p = Q.top();
        Q.pop();
        x = p.second;

        if(marked[x] == true)
            continue;
        minimumCost += p.first;
        marked[x] = true;
        for(int i = 0;i < adj[x].size();++i)
        {
            y = adj[x][i].second;
            if(marked[y] == false)
                Q.push(adj[x][i]);
        }
    }
    return minimumCost;
}

int main()
{
    int nodes, edges, x, y;
    long long weight, minimumCost;
```

```
    cin >> nodes >> edges;
    for(int i = 0;i < edges;++i)
    {
        cin >> x >> y >> weight;
        adj[x].push_back(make_pair(weight, y));
        adj[y].push_back(make_pair(weight, x));
    }

    minimumCost = prim(1);
    cout << minimumCost << endl;
    return 0;
}
```

10> Travelling salesman problem, find shortest length hamiltonian cycle

11> Finding hamiltonian cycle if any (a cycle that visits each node only once and last node is connected to first node, so a cycle. THere can be multiple solutions

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

// Number of vertices in the graph
#define V 5

void printSolution(int path[]);


bool isSafe(int v, bool graph[V][V],
            int path[], int pos)
{

    if (graph [path[pos - 1]][ v ] == 0)
        return false;


    for (int i = 0; i < pos; i++)
        if (path[i] == v)
```

```
            return false;

    return true;
}


bool hamCycleUtil(bool graph[V][V],
                  int path[], int pos)
{

    if (pos == V)
    {

        if (graph[path[pos - 1]][path[0]] == 1)
            return true;
        else
            return false;
    }


    for (int v = 1; v < V; v++)
    {

        if (isSafe(v, graph, path, pos))
        {
            path[pos] = v;

            if (hamCycleUtil (graph, path, pos + 1) == true)
                return true;


            path[pos] = -1;
        }
    }


    return false;
}

bool hamCycle(bool graph[V][V])
{
    int *path = new int[V];
    for (int i = 0; i < V; i++)
        path[i] = -1;


    path[0] = 0;
    if (hamCycleUtil(graph, path, 1) == false )
    {
        cout << "\nSolution does not exist";
```

```
        return false;
    }

    printSolution(path);
    return true;
}


void printSolution(int path[])
{
    cout << "Solution Exists:"
            " Following is one Hamiltonian Cycle \n";
    for (int i = 0; i < V; i++)
        cout << path[i] << " ";


    cout << path[0] << " ";
    cout << endl;
}

// Driver Code
int main()
{

    bool graph1[V][V] = {{0, 1, 0, 1, 0},
                         {1, 0, 1, 1, 1},
                         {0, 1, 0, 0, 1},
                         {1, 1, 0, 0, 1},
                         {0, 1, 1, 1, 0}};

    // Print the solution
    hamCycle(graph1);


    bool graph2[V][V] = {{0, 1, 0, 1, 0},
                         {1, 0, 1, 1, 1},
                         {0, 1, 0, 0, 1},
                         {1, 1, 0, 0, 0},
                         {0, 1, 1, 0, 0}};

    // Print the solution
    hamCycle(graph2);

    return 0;
}
```

12> Kosaraju algo for number of strongly connected components O(v+E)

```
#include <bits/stdc++.h>
using namespace std;
void dfs(int node, stack<int> &st, vector<int> &vis, vector<int> adj[]) {
    vis[node] = 1;
    for(auto it: adj[node]) {
        if(!vis[it]) {
            dfs(it, st, vis, adj);
        }
    }

    st.push(node);
}
void revDfs(int node, vector<int> &vis, vector<int> transpose[]) {
    cout << node << " ";
    vis[node] = 1;
    for(auto it: transpose[node]) {
        if(!vis[it]) {
            revDfs(it, vis, transpose);
        }
    }
}
int main() {
    int n, m;
    cin >> n >> m;
        vector<int> adj[n];
        for(int i = 0;i<m;i++) {
            int u, v;
            cin >> u >> v;
            adj[u].push_back(v);
        }

        stack<int> st;
        vector<int> vis(n, 0);
        for(int i = 0;i<n;i++) {
            if(!vis[i]) {
                dfs(i, st, vis, adj);
            }
        }

        vector<int> transpose[n];

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

```
            vis[i] = 0;
            for(auto it: adj[i]) {
                transpose[it].push_back(i);
            }
        }

        int ans=0;

        while(!st.empty()) {
            int node = st.top();
            st.pop();
            if(!vis[node]) {
                ans++;
                cout << "SCC: ";
                revDfs(node, vis, transpose);
                cout << endl;
            }
        }


        return ans;
}
```