

## WEEK-6

**6.1:** Given a (directed/undirected) graph, design an algorithm and implement it using a program to find if a path exists between two given vertices or not. (Hint: use DFS)

### 6.1.1—Pseudo-code

### 6.1.2—Source-code

```
/*Name: Ritesh Singh
Sec: C
Roll No: 52
Student ID: 20011347*/
#include <bits/stdc++.h>
using namespace std;
bool findpath(int src, int dest, vector<vector<int>> &graph)
{
    if (src == dest)
        return true;
    int n = graph.size();
    vector<bool> visited(n, false);
    visited[src] = true;
    stack<int> s;
    s.push(src);
    while (!s.empty())
    {
        int a = s.top();
        s.pop();
        for (int x : graph[a])
        {
            if (x == dest)
                return true;
            if (!visited[x])
            {
                visited[x] = true;
                s.push(x);
            }
        }
    }
    return false;
}
int main()
{
    int n, m;
    cin >> n >> m;
    vector<vector<int>> graph(n);
```

```

for (int i = 0; i < m; i++)
{
    int u, v;
    cin >> u >> v;
    graph[u].push_back(v);
    graph[v].push_back(u);
}
int source, dest;
cin >> source >> dest;
if (findpath(source, dest, graph))
    cout << "Yes Path Exists";
else
    cout << "No Such Path Exists";
return 0;
}

```

### **6.1.3— Output**

```

6 8
0 5
0 1
2 4
2 3
3 1
4 3
1 5
1 4
0 4
Yes Path Exists
PS B:\Desktop\notRitesh\DAA>

```

**6.2:** Given a graph, design an algorithm and implement it using a program to find if a graph is bipartite or not. (Hint: use BFS)

### **6.2.1—Pseudo-code**

### **6.2.2—Source-code**

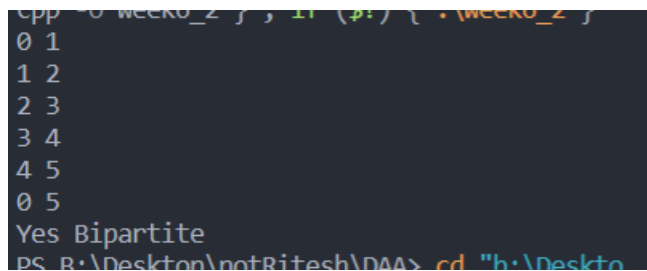
```
/*Name: Ritesh Singh
Sec: C
Roll No: 52
Student ID: 20011347*/
#include <bits/stdc++.h>
using namespace std;
vector<vector<int>> adj;
vector<bool> vis;
vector<int> col;
bool bipart;
//assign color to nodes either 0 or 1
void color(int u, int curr)
{
    if (col[u] != -1 and col[u] != curr)
    {
        bipart = false;
        return;
    }
    col[u] = curr;
    if (vis[u])
        return;
    vis[u] = true;
    for (auto i : adj[u])
    {
        color(i, curr xor 1);
    }
}
int main()
{
    int n, m;
    cin >> n >> m;
    adj = vector<vector<int>>(n);
    vis = vector<bool>(n, false);
    col = vector<int>(n, -1);
    bipart = true;
    for (int i = 0; i < m; i++)
    {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
```

```

        adj[v].push_back(u);
    }
    for (int i = 0; i < n; i++)
    {
        if (!vis[i])
        {
            color(i, 0);
        }
    }
    if (bipart)
        cout << "Yes Bipartite";
    else
        cout << "Not Bipartite";
    return 0;
}

```

### **6.2.3— Output**



```

cpp -o week0_2 } , 11 { #! } . week0_2 }
0 1
1 2
2 3
3 4
4 5
0 5
Yes Bipartite
PS: B:\Desktop\notRitesh\DA4> cd "h:\Deskto

```

**6.3:** Given a directed graph, design an algorithm and implement it using a program to find whether cycle exists in the graph or not.

### **6.3.1—Pseudo-code**

### **6.3.2—Source-code**

/\*Name: Ritesh Singh

Sec: C

Roll No: 52

Student ID: 20011347\*/

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

```
using namespace std;
```

```
bool iscycle(int src, vector<vector<int>> &adj, vector<bool> &vis, int parent)
```

```
{
```

```
    vis[src] = true;
```

```
    for (auto i : adj[src])
```

```
    {
```

```
        if (i != parent)
```

```
        {
```

```
            if (vis[i])
```

```
                return true;
```

```
            if (!vis[i] and iscycle(i, adj, vis, src))
```

```
            {
```

```
                return true;
```

```
            }
```

```
        }
```

```
    }
```

```
    return false;
```

```
}
```

```
int main()
```

```
{
```

```
    int n, m;
```

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

```
    vector<vector<int>> adj(n);
```

```
    vector<bool> vis(n, false);
```

```
    bool cycle = false;
```

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

```
    {
```

```
        int u, v;
```

```
        cin >> u >> v;
```

```
        adj[u].push_back(v);
```

```
        adj[v].push_back(u);
```

```
    }
```

```
    // set node 0 as visited
```

```
    vis[0] = true;
```

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

```
    {
```

```

        if (!vis[i] and iscycle(i, adj, vis, -1))
        {
            cycle = true;
            break;
        }
    }
    if (cycle)
        cout << "Yes Cycle Exists";
    else
        cout << "Cycle Not Present";
    return 0;
}

```

### **6.3.3— Output**

```

PS B:\Desktop\notRitesh\DAA> cd "b:\Desktop\notRitesh\DAA"
5 6
0 1
1 3
0 3
2 4
0 2
2 4
Yes Cycle Exists

```

## **WEEK-7**

**7.1:**After end term examination, Akshay wants to party with his friends. All his friends are living as paying guest and it has been decided to first gather at Akshay's house and then move towards party location. The problem is that no one knows the exact address of his house in the city. Akshay as a computer science wizard knows how to apply his theory subjects in his real life and came up with an amazing idea to help his friends. He draws a graph by looking in to location of his house and his friends' location (as a node in the graph) on a map. He wishes to find out shortest distance and path covering that distance from each of his friend's location to his house and then whatsapp them this path so that they can reach his house in minimum time. Akshay has developed the program that implements Dijkstra's algorithm but not sure about correctness of results. Can you also implement the same algorithm and verify the correctness of Akshay's results? (Hint: Print shortest path and distance from friends' location to Akshay's house)

### **7.1.1—Pseudo-code**

### **7.1.2—Source-code**

```
/*Name: Ritesh Singh
Sec: C
Roll No: 52
Student ID: 20011347*/
#include <bits/stdc++.h>
using namespace std;
void path(vector<int> &parent, int j)
{
    if (parent[j] == -1)
    {
        cout << j;
        return;
    }
    printf("%d ", j);
    path(parent, parent[j]);
}
int main()
{
    int n, e;
    cin >> n >> e;
    vector<vector<pair<int, int>>> graph(n + 1);
    for (int i = 0; i < e; i++)
    {
        int s, d, w;
        cin >> s >> d >> w;
        graph[s].push_back({d, w});
        graph[d].push_back({s, w});
    }
    vector<int> dist(n + 1, INT_MAX);
```

```

set<pair<int, int>> s;
int source;
cin >> source;
dist[source] = 0;
s.insert({0, source});
vector<int> parent(n + 1, -1);
while (!s.empty())
{
    auto x = *(s.begin());
    s.erase(x);
    for (auto it : graph[x.second])
    {
        if (dist[it.first] > dist[x.second] + it.second)
        {
            s.erase({ dist[it.first], it.first });
            dist[it.first] = dist[x.second] + it.second;
            s.insert({ dist[it.first], it.first });
            parent[it.first] = x.second;
        }
    }
}
for (int i = 1; i < n + 1; i++)
{
    path(parent, i);
    cout << " : " << dist[i] << endl;
}
return 0;
}

```

### **7.1.3—Output**

```

-o week/_1 } ; if ($?) { .\week/_1 }
5 6
1 2 4
3 1 1
2 5 4
3 4 4
4 5 4
1
1 : 0
2 3 1 : 3
3 1 : 1
4 3 1 : 5
5 2 3 1 : 7

```



**7.2:** Design an algorithm and implement it using a program to solve previous question's problem using Bellman- Ford's shortest path algorithm.

### **7.2.1—Pseudo-code**

### **7.2.2—Source-code**

```
/*Name: Ritesh Singh
Sec: C
Roll No: 52
Student ID: 20011347*/
#include <bits/stdc++.h>
using namespace std;
void path(vector<int> &parent, int j)
{
    if (parent[j] == -1)
    {
        cout << j;
        return;
    }
    printf("%d ", j);
    path(parent, parent[j]);
}
int main()
{
    int n, e;
    cin >> n >> e;
    vector<vector<int>> edges;
    for (int i = 0; i < e; i++)
    {
        int u, v, w;
        cin >> u >> v >> w;
        edges.push_back({u, v, w});
        edges.push_back({v, u, w});
    }

    vector<int> parent(n + 1, -1);
    //initialize distance array
    vector<int> dist(n + 1, 1e9);
    int src;
    cin >> src; //input source
    dist[src] = 0; // initialize source distance to 0
    //iterate n-1 times to relax each edge
    bool negative_cycle;
    for (int i = 1; i < n; i++)
    {
        negative_cycle = false; //to detect -ve cycle
```

```

for (auto it : edges)
{
    int u, v, w;
    u = it[0];
    v = it[1];
    w = it[2];
    if (dist[v] > dist[u] + w)
    {
        dist[v] = dist[u] + w;
        parent[v] = u;
        negative_cycle = true;
    }
}
}
if (negative_cycle)
    cout << "negative cycle present";
else
{
    for (int i = 1; i < n + 1; i++)
    {
        path(parent, i);
        cout << " : " << dist[i] << endl;
    }
}
return 0;
}

```

### **7.2.3— Output**

```

-0 week7_2 } ; 11 ($!) { . week7_2 }
5 6
1 2 4
3 1 1
2 5 4
3 4 4
4 5 4
1
1 : 0
2 3 1 : 3
3 1 : 1
4 3 1 : 5
5 2 3 1 : 7
PS B:\Desktop\notRitesh\DAA> cd "b:\Desktop\r

```

**7.3:** Given a directed graph with two vertices ( source and destination). Design an algorithm and implement it using a program to find the weight of the shortest path from source to destination with exactly k edges on the path.

### **7.3.1—Pseudo-code**

### **7.3.2—Source-code**

```
/*Name: Ritesh Singh
Sec: C
Roll No: 52
Student ID: 20011347*/
#include <bits/stdc++.h>
using namespace std;
#define V 100
#define INF INT_MAX
int arr[100][100];
int shortestpath(int arr[][V], int u, int v, int k, int n)
{
    if (k == 0 && u == v)
        return 0;
    if (k == 1 && arr[u][v] != INF)
        return arr[u][v];
    if (k <= 0)
        return INF;
    int res = INF;
    for (int i = 0; i < n; i++)
    {
        if (arr[u][i] != INF && u != i && v != i)
        {
            int rec_res = shortestpath(arr, i, v, k - 1, n);
            if (rec_res != INF)
                res = min(res, arr[u][i] + rec_res);
        }
    }
    return res;
}
int main()
{
    int n;
    cout << "for values INF enter -1" << endl;
    cin >> n;
    int a;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> a;
```

```

        if (a < 0)
        {
            arr[i][j] = INF;
        }
        else
            arr[i][j] = a;
    }
}
int u, v, k;
cin >> u >> v >> k;
cout << "weight of the shortest path is " << shortestpath(arr, u - 1, v, k, n) << endl; // 0
indexing is followed
return 0;
}

```

### **7.3.3— Output**

```

PS C:\Desktop\notRitesh\DA4> cd C:\Desktop\notRitesh\DA4
for values INF enter -1
4
0 10 3 2
-1 -1 0 7
0 -1 -1 6
-1 -1 -1 0
1 4 2
weight of the shortest path is 2
PS C:\Desktop\notRitesh\DA4>

```

## **WEEK-8**

**8.1:** Assume that a project of road construction to connect some cities is given to your friend. Map of these cities and roads which will connect them (after construction) is provided to him in the form of a graph. Certain amount of rupees is associated with construction of each road. Your friend has to calculate the minimum budget required for this project. The budget should be designed in such a way that the cost of connecting the cities should be minimum and number of roads required to connect all the cities should be minimum (if there are N cities then only N-1 roads need to be constructed). He asks you for help. Now, you have to help your friend by designing an algorithm which will find minimum cost required to connect these cities. (use Prim's algorithm)

### **8.1.1—Pseudo-code**

### **8.1.2—Source-code**

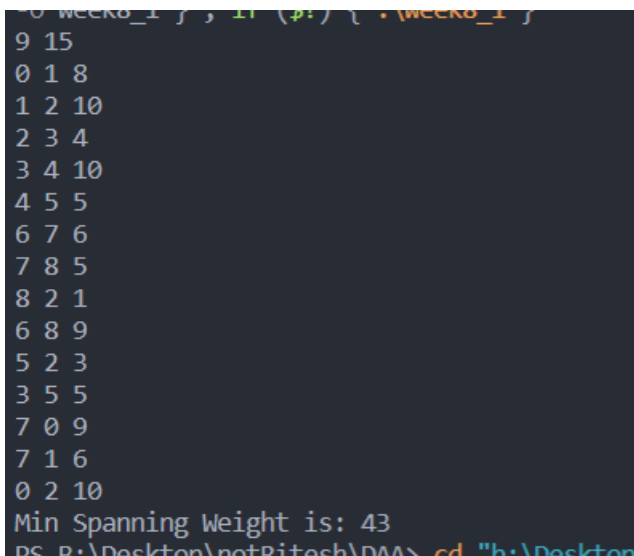
```
/*Name: Ritesh Singh
Sec: C
Roll No: 52
Student ID: 20011347*/
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int nodes, edges;
    cin >> nodes;
    cin >> edges;
    vector<pair<int, int>> graph[nodes];
    int source, destination, weight;
    for (int i = 0; i < edges; i++)
    {
        cin >> source >> destination >> weight;
        graph[source].push_back(make_pair(destination, weight));
        graph[destination].push_back(make_pair(source, weight));
    }
    int key[nodes]; //to select the min weight
    int parent[nodes]; // to store the parent node
    bool mst[nodes]; // to construct the path
    for (int i = 0; i < nodes; i++)
    {
        key[i] = INT_MAX;
        mst[i] = false;
        parent[i] = -1;
    }
    // priority queue APPROACH
    priority_queue<pair<int, int>, vector<pair<int, int>>,
        greater<pair<int, int>>>
        pq;
```

```

key[0] = 0; //select a node to start from
parent[0] = 0;
pq.push({0, 0}); //{weight, index of starting node}
for (int i = 0; i < nodes - 1; i++)
{
    int u = pq.top().second; //get the index of top node
    pq.pop();                //remove the node from queue
    mst[u] = true;           //set mst as true for the node u
    for (auto it : graph[i])
    {
        int dest = it.first;
        int wt = it.second;
        if (mst[dest] == false && wt < key[dest]) //check if the parent array needs to be changed
        {
            parent[dest] = u;
            pq.push({key[dest], dest});
            key[dest] = wt;
        }
    }
}
int mstwt = 0;
// to print the list with minimum weight
for (int i = 0; i < nodes; i++)
    mstwt += key[i];
cout << "Min Spanning Weight is: " << mstwt;
return 0;
}

```

### 8.1.3— Output



```

0 1 8
1 2 10
2 3 4
3 4 10
4 5 5
6 7 6
7 8 5
8 2 1
6 8 9
5 2 3
3 5 5
7 0 9
7 1 6
0 2 10
Min Spanning Weight is: 43
PS: B:\Desktop\notRitesh\DMA> cd "h:\Desktop"

```

## **8.2:** Implement the previous problem using Kruskal's algorithm

### **8.2.1**—Pseudo-code

### **8.2.2**—Source-code

```
/*Name: Ritesh Singh
Sec: C
Roll No: 52
Student ID: 20011347*/
#include <bits/stdc++.h>
using namespace std;
vector<int> parent(100);
vector<int> sz(100);
void make_set(int v)
{
    parent[v] = v;
    sz[v] = 1;
}
int find_set(int v)
{
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}
void union_set(int a, int b)
{
    a = find_set(a);
    b = find_set(b);
    if (a != b)
    { //dont belong to same set
        if (sz[a] < sz[b])
            swap(a, b);
        parent[b] = a;
        sz[a] += sz[b];
    }
}
int main()
{
    int n, e;
    cin >> n >> e;
    for (int i = 0; i < n; i++)
        make_set(i);
    vector<vector<int>> graph;
    for (int i = 0; i < e; i++)
    {
        int u, v, w;
        cin >> u >> v >> w;
```

```

        graph.push_back({ w, u, v });
        graph.push_back({ w, v, u });
    }
    sort(graph.begin(), graph.end()); //sort according to weight
    int total_weight = 0;
    for (auto i : graph)
    {
        int w = i[0];
        int u = i[1];
        int v = i[2];
        int x = find_set(u);
        int y = find_set(v);
        if (x == y)
        {
            continue;
        }
        else
        {
            total_weight += w;
            union_set(u, v); //add to set
        }
    }
    cout << "Minimum Spanning Weight is: " << total_weight;
    return 0;
}

```

### **8.2.3— Output**

```

9 15
0 1 8
1 2 10
2 3 4
3 4 10
4 5 5
5 6 1
6 7 6
7 8 5
8 2 1
6 8 9
5 2 3
3 5 5
7 0 9
7 1 6
0 2 10
Minimum Spanning Weight is: 33

```



**8.3:** Assume that same road construction project is given to another person. The amount he will earn from this project is directly proportional to the budget of the project. This person is greedy, so he decided to maximize the budget by constructing those roads who have highest construction cost. Design an algorithm and implement it using a program to find the maximum budget required for the project.

### **8.3.1—Pseudo-code**

### **8.3.2—Source-code**

```
/*Name: Ritesh Singh
Sec: C
Roll No: 52
Student ID: 20011347*/
#include <bits/stdc++.h>
using namespace std;
bool compare(const pair<int, int> &a, const pair<int, int> &b)
{
    return b.first > a.first;
}
vector<int> parent(100);
vector<int> sz(100);
void make_set(int v)
{
    parent[v] = v;
    sz[v] = 1;
}
int find_set(int v)
{
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}
void union_set(int a, int b)
{
    a = find_set(a);
    b = find_set(b);
    if (a != b)
    { //dont belong to same set
        if (sz[a] < sz[b])
            swap(a, b);
        parent[b] = a;
        sz[a] += sz[b];
    }
}
int main()
{
    int n, e;
```

```

cin >> n >> e;
for (int i = 0; i < n; i++)
    make_set(i);
vector<vector<int>> graph;
for (int i = 0; i < e; i++)
{
    int u, v, w;
    cin >> u >> v >> w;
    graph.push_back({w, u, v});
    graph.push_back({w, v, u});
}
sort(graph.rbegin(), graph.rend()); //sort according to weight
int total_weight = 0;
for (auto i : graph)
{
    int w = i[0];
    int u = i[1];
    int v = i[2];
    int x = find_set(u);
    int y = find_set(v);
    if (x == y)
    {
        continue;
    }
    else
    {
        total_weight += w;
        union_set(u, v); //add to set
    }
}
cout << "Maximum Spanning Weight is: " << total_weight;
return 0;
}

```

### **8.3.3— Output**

```

9 15
0 1 8
1 2 10
2 3 4
3 4 10
4 5 5
5 6 1
6 7 6
7 8 5
8 2 1
6 8 9
5 2 3
3 5 5
7 0 9
7 1 6
0 2 10
Maximum Spanning Weight is: 63
PS: B:\Desktop\netRitesh\DA4>

```