LAB 9

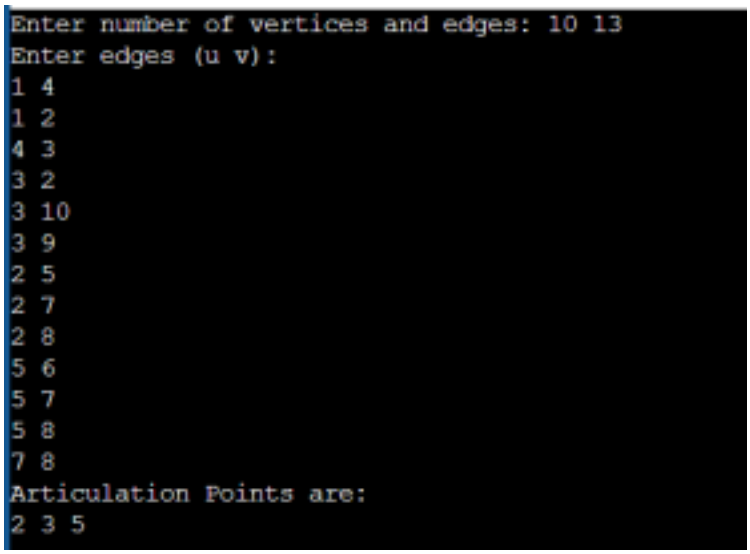
Tarjan’s Algorithm :

PROGRAM :

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

const int MAX = 100;

vector<int> adj[MAX]; // Adjacency list
int dfn[MAX]; // Discovery time
int low[MAX]; // Lowest reachable discovery time bool
visited[MAX];
bool isAP[MAX]; // Marks articulation points int
timer;

void tarjanAP(int u, int parent) {
    visited[u] = true;
    dfn[u] = low[u] = ++timer;
    int children = 0;

    for (int v : adj[u]) {

        if (!visited[v]) {
            children++;
            tarjanAP(v, u);

            // Update low value
            low[u] = min(low[u], low[v]);

            // Case 1: Root node with more than 1 child
            if (parent == -1 && children > 1)
                isAP[u] = true;

            // Case 2: Non-root node
            if (parent != -1 && low[v] >= dfn[u])
                isAP[u] = true;
        }
```

```
        else if (v != parent) {
            // Back edge
            low[u] = min(low[u], dfn[v]);
        }
    }
}

int main() {
    int V, E;
    cout << "Enter number of vertices and edges: ";
    cin >> V >> E;

    cout << "Enter edges (u v):\n";
    for (int i = 0; i < E; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u); // Undirected graph }

    timer = 0;

    for (int i = 0; i < V; i++) {
        if (!visited[i])
            tarjanAP(i, -1);
    }

    cout << "Articulation Points are:\n";
    for (int i = 0; i < V; i++) {
        if (isAP[i])
            cout << i << " ";
    }

    return 0;
}
```

OUTPUT :



Bellman Ford algorithm :

PROGRAM :

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

struct Edge {
    int u, v, weight;
};

int main() {
    int V, E;
    cin >> V >> E;

    vector<Edge> edges(E);

    for (int i = 0; i < E; i++) {
        cin >> edges[i].u >> edges[i].v >> edges[i].weight;
    }

    int source;
    cin >> source;
    const int INF = 1000000000;
    vector<int> dist(V, INF);
```

```
    dist[source] = 0;

    for (int i = 1; i <= V - 1; i++) {
        for (int j = 0; j < E; j++) {
            int u = edges[j].u;
            int v = edges[j].v;
            int w = edges[j].weight;

            if (dist[u] != INF && dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
            }
        }
    }

    for (int j = 0; j < E; j++) {
        int u = edges[j].u;
        int v = edges[j].v;
        int w = edges[j].weight;

        if (dist[u] != INF && dist[u] + w < dist[v]) {
            cout << "Negative weight cycle detected\n";
            return 0;
        }
    }

    cout << "Shortest distances from source " << source << ":\n";
    for (int i = 0; i < V; i++) {
        cout << "Vertex " << i << " : " << dist[i] << endl;
    }

    return 0;
}
```

OUTPUT :

```
5 8
0 1 -1
0 2 4
1 2 3
1 3 2
1 4 2
3 2 5
3 1 1
4 3 -3
0
Shortest distances from source 0:
Vertex 0 : 0
Vertex 1 : -1
Vertex 2 : 2
Vertex 3 : -2
Vertex 4 : 1
```