# AHSANIA MISSION UNIVERSITY OF SCIENCE & TECHNOLOGY

# Lab Report-8

**Lab No:** 08

**Course Code:** CSE 2202

**Course Title:** Computer Algorithm Sessional.

## Submitted By:

Md Sakline Hossen
ID: 1012320005101027
1st Batch, 2nd Year, 2nd Semester
Department of Computer science and Engineering,
Ahsania Mission University of Science & Technology

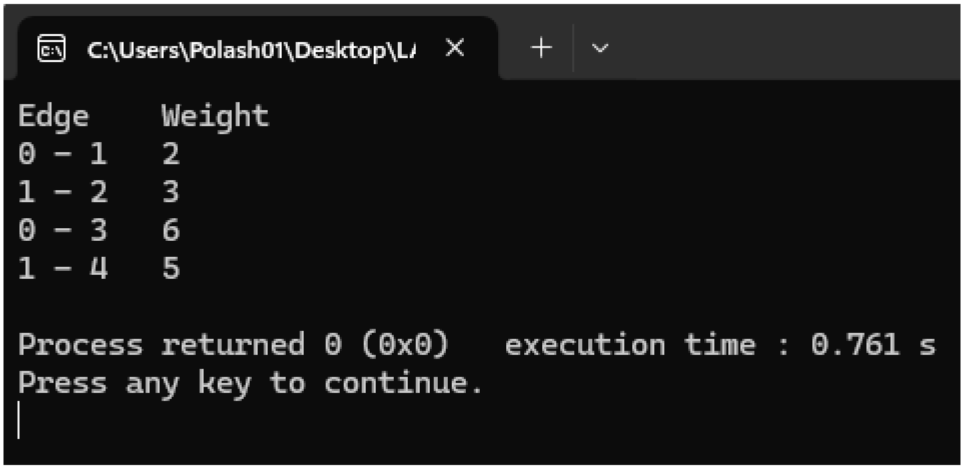
## Submitted To:

Md. Fahim Faisal
Lecturer,
Department of Computer science and Engineering,
Ahsania Mission University of Science & Technology

## Task01: Prim's Algorithm

```
#include <iostream>
#include <limits.h>
using namespace std;

#define V 5

// Find the vertex with the smallest key value
int findMinKey(int key[], bool inMST[]) {
    int min = INT_MAX, index;
    for (int i = 0; i < V; i++) {
        if (!inMST[i] && key[i] < min) {
            min = key[i];
            index = i;
        }
    }
    return index;
}

// Print the MST using parent array
void printMST(int parent[], int graph[V][V]) {
    cout << "Edge\tWeight\n";
    for (int i = 1; i < V; i++) {
        cout << parent[i] << " - " << i << "\t" << graph[i][parent[i]] << "\n";
    }
}

// Prim’s algorithm to construct MST
void primMST(int graph[V][V]) {
    int parent[V];     // Store MST structure
    int key[V];        // Store edge weights
    bool inMST[V];     // Track included vertices

    for (int i = 0; i < V; i++) {
        key[i] = INT_MAX;
        inMST[i] = false;
    }

    key[0] = 0;
    parent[0] = -1;

    for (int i = 0; i < V - 1; i++) {
        int u = findMinKey(key, inMST);
        inMST[u] = true;

        for (int v = 0; v < V; v++) {
            if (graph[u][v] && !inMST[v] && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }

    printMST(parent, graph);
}

int main() {
    int graph[V][V] = {
        {0, 2, 0, 6, 0},
        {2, 0, 3, 8, 5},
        {0, 3, 0, 0, 7},
        {6, 8, 0, 0, 9},
        {0, 5, 7, 9, 0}
    };

    primMST(graph);
    return 0;
}
```

Output:



```
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5

Process returned 0 (0x0)   execution time : 0.761 s
Press any key to continue.
```

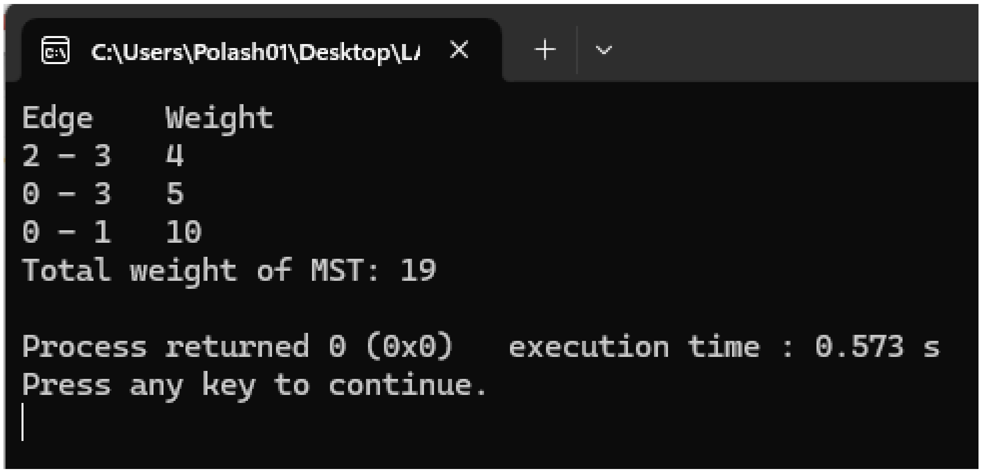
| Test Case | Input Adjacency Matrix | Output MST Edges (Edge : Weight) | Total Cost |
|---|---|---|---|
| **1** | { {0,2,0,6,0}, {2,0,3,8,5}, {0,3,0,0,7}, {6,8,0,0,9}, {0,5,7,9,0} } | 0–1:2, 1–2:3, 0–3:6, 1–4:5 | **16** |
| **2** | { {0,1,2,0}, {1,0,4,6}, {2,4,0,3}, {0,6,3,0} } | 0–1:1, 0–2:2, 2–3:3 | **6** |
| **3** | { {0,10,0,0,5}, {10,0,1,0,2}, {0,1,0,4,0}, {0,0,4,0,3}, {5,2,0,3,0} } | 0–4:5, 4–1:2, 1–2:1, 4–3:3 | **11** |

## Task02: Kruskal's Algorithm

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

struct Edge {
    int u, v, weight;
    bool operator<(const Edge& other) {
        return weight < other.weight;
    }
};

int findSet(int node, vector<int>& parent) {
    if (parent[node] == node)
        return node;
    return parent[node] = findSet(parent[node], parent);
}

void unionSet(int a, int b, vector<int>& parent, vector<int>& rank) {
    a = findSet(a, parent);
    b = findSet(b, parent);

    if (a != b) {
        if (rank[a] < rank[b])
            swap(a, b);
        parent[b] = a;
        if (rank[a] == rank[b])
            rank[a]++;
    }
}
```

```
int main() {
   int V = 4;
   vector<Edge> edges = {
      {0, 1, 10},
      {0, 2, 6},
      {0, 3, 5},
      {1, 3, 15},
      {2, 3, 4}
   };

   sort(edges.begin(), edges.end());

   vector<int> parent(V), rank(V, 0);
   for (int i = 0; i < V; i++)
      parent[i] = i;

   vector<Edge> mst;

   for (Edge e : edges) {
      if (findSet(e.u, parent) != findSet(e.v, parent)) {
         mst.push_back(e);
         unionSet(e.u, e.v, parent, rank);
      }
   }

   cout << "Edge\tWeight\n";
   int total = 0;
   for (Edge e : mst) {
      cout << e.u << " - " << e.v << "\t" << e.weight << "\n";
      total += e.weight;
   }
   cout << "Total weight of MST: " << total << endl;

   return 0;
}
```

Output:



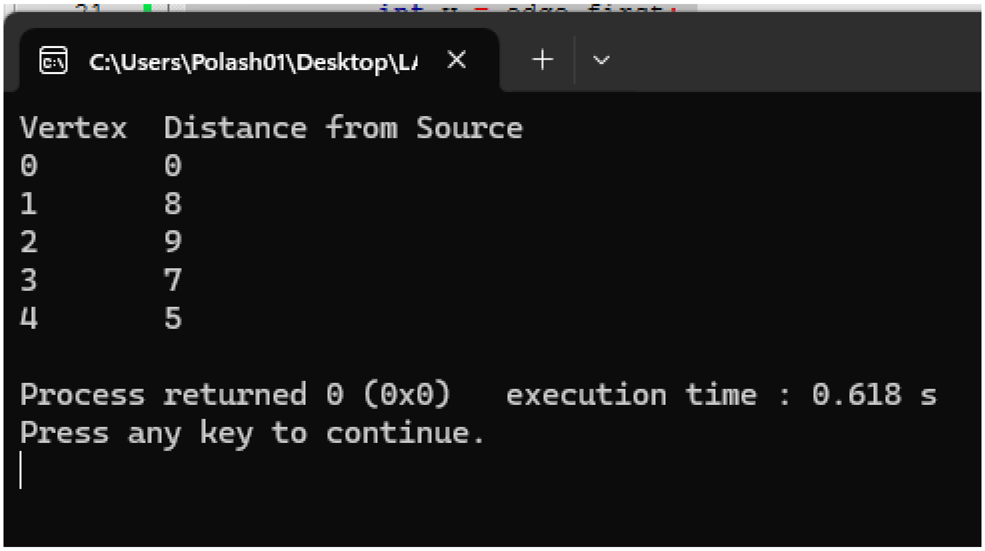
```
Edge    Weight
2 - 3   4
0 - 3   5
0 - 1   10
Total weight of MST: 19

Process returned 0 (0x0)   execution time : 0.573 s
Press any key to continue.
```

| Test Case | Input Edges (u, v, weight) | Output MST Edges (Edge : Weight) | Total Cost |
|---|---|---|---|
| **1** | (0,1,10), (0,2,6), (0,3,5), (1,3,15), (2,3,4) | 2–3:4, 0–3:5, 0–1:10 | **19** |
| **2** | (0,1,1), (0,2,3), (1,2,2), (1,3,4), (2,3,5) | 0–1:1, 1–2:2, 1–3:4 | **7** |
| **3** | (0,1,2), (0,3,6), (1,2,3), (1,3,8), (2,3,5) | 0–1:2, 1–2:3, 2–3:5 | **10** |

## Task03: Dijkstra's Algorithm

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

typedef pair<int, int> pii;

void dijkstra(int V, vector<pii> adj[], int source) {
    vector<int> dist(V, INT_MAX);
    dist[source] = 0;

    priority_queue<pii, vector<pii>, greater<pii>> pq;
    pq.push({0, source});

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();

        for (auto& edge : adj[u]) {
            int v = edge.first;
            int weight = edge.second;

            if (dist[v] > dist[u] + weight) {
                dist[v] = dist[u] + weight;
                pq.push({dist[v], v});
            }
        }
    }
    cout << "Vertex\tDistance from Source\n";
    for (int i = 0; i < V; ++i)
        cout << i << "\t" << dist[i] << "\n";
}

int main() {
    int V = 5;
    vector<pii> adj[V];

    adj[0].push_back({1, 10});
    adj[0].push_back({4, 5});
    adj[1].push_back({2, 1});
    adj[1].push_back({4, 2});
    adj[2].push_back({3, 4});
    adj[3].push_back({2, 6});
    adj[3].push_back({0, 7});
    adj[4].push_back({1, 3});
    adj[4].push_back({2, 9});
    adj[4].push_back({3, 2});

    dijkstra(V, adj, 0);
    return 0;
}
```

Output:



```
Vertex  Distance from Source
0       0
1       8
2       9
3       7
4       5

Process returned 0 (0x0)   execution time : 0.618 s
Press any key to continue.
```

| **Test Case** | **Input Edges (Adjacency List)** | **Source Vertex** | **Output (Vertex : Distance from Source)** |
|---|---|---|---|
| **1** | 0→(1,10),(4,5); 1→(2,1),(4,2); 2→(3,4); 3→(2,6),(0,7); 4→(1,3),(2,9),(3,2) | 0 | 0:0, 1:8, 2:9, 3:7, 4:5 |
| **2** | 0→(1,4),(2,1); 1→(3,1); 2→(1,2),(3,5); 3→() | 0 | 0:0, 1:3, 2:1, 3:4 |
| **3** | 0→(1,2),(2,4); 1→(2,1),(3,7); 2→(4,3); 3→(5,1); 4→(3,2); 5→() | 0 | 0:0, 1:2, 2:3, 3:8, 4:6, 5:9 |