

EXPERIMENT NO.5

Minimum Scanning Tree

Reg. No.: -24141045

Program:-

```
#include <iostream>

#include <vector>

#include <climits>

using namespace std;

int findMinVertex(vector<int>& weights, vector<bool>& visited, int n) {

    int minVertex = -1;

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

        if (!visited[i] && (minVertex == -1 || weights[i] < weights[minVertex]))

            minVertex = i;

    }

    return minVertex;

}

void prims(vector<vector<int> >& graph, int n) {

    vector<int> parent(n);

    vector<int> weights(n, INT_MAX);

    vector<bool> visited(n, false);

    parent[0] = -1;

    weights[0] = 0;
```

```

for (int i = 0; i < n - 1; i++) {
    int minVertex = findMinVertex(weights, visited, n);
    visited[minVertex] = true;
    for (int j = 0; j < n; j++) {
        if (graph[minVertex][j] != 0 && !visited[j]) {
            if (graph[minVertex][j] < weights[j]) {
                weights[j] = graph[minVertex][j];
                parent[j] = minVertex;
            }
        }
    }
}

cout << "\nEdges in MST (Prim's):\n";
int totalWeight = 0;
for (int i = 1; i < n; i++) {
    cout << parent[i] << " - " << i << " (Weight: " << graph[i][parent[i]] << ")\n";
    totalWeight += graph[i][parent[i]];
}

cout << "Total Cost = " << totalWeight << endl;
}

int main() {
    int n;
    cout << "Enter the number of vertices: ";

```

```

cin >> n;

vector< vector<int> >graph(n, vector<int>(n));

cout << "\nEnter the adjacency matrix (enter 0 if no edge):\n";

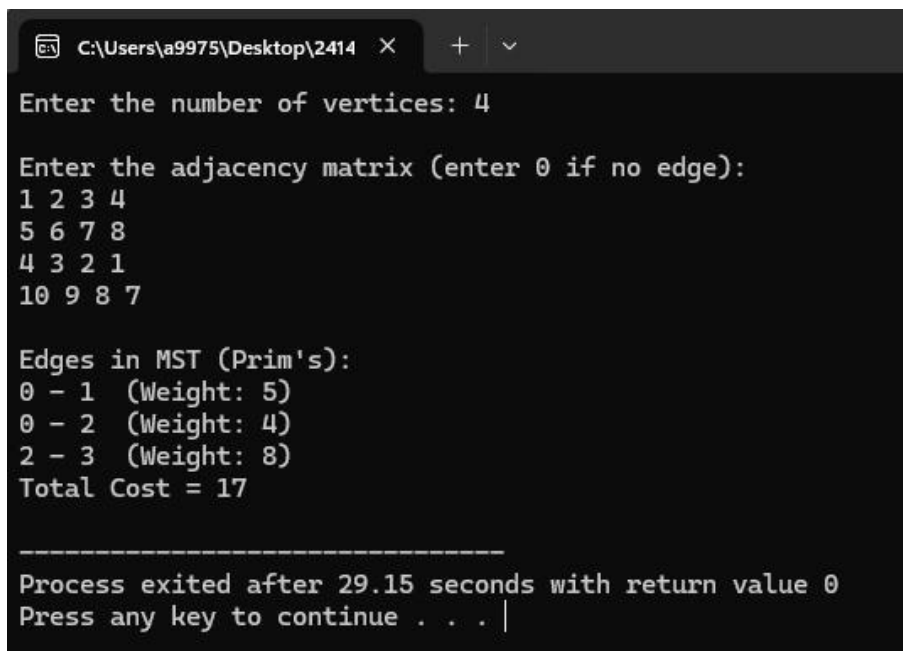
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        cin >> graph[i][j];
    }
}

prims(graph, n);

return 0;
}
}

```

Output:-



```

C:\Users\A9975\Desktop\2414 X + v
Enter the number of vertices: 4

Enter the adjacency matrix (enter 0 if no edge):
1 2 3 4
5 6 7 8
4 3 2 1
10 9 8 7

Edges in MST (Prim's):
0 - 1 (Weight: 5)
0 - 2 (Weight: 4)
2 - 3 (Weight: 8)
Total Cost = 17

-----
Process exited after 29.15 seconds with return value 0
Press any key to continue . . . |

```

Application:-

```

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

int findMinVertex(vector<int>& weights, vector<bool>& visited, int n) {
    int minVertex = -1;
    for (int i = 0; i < n; i++) {
        if (!visited[i] && (minVertex == -1 || weights[i] < weights[minVertex]))
            minVertex = i;
    }
    return minVertex;
}

void prims(vector <vector< int > >& graph, int n) {
    vector<int> parent(n);
    vector<int> weights(n, INT_MAX);
    vector<bool> visited(n, false);
    parent[0] = -1;
    weights[0] = 0;
    for (int i = 0; i < n - 1; i++) {
        int minVertex = findMinVertex(weights, visited, n);
        visited[minVertex] = true;
        for (int j = 0; j < n; j++) {
            if (graph[minVertex][j] != 0 && !visited[j]) {

```

```

        if (graph[minVertex][j] < weights[j]) {
            weights[j] = graph[minVertex][j];
            parent[j] = minVertex;
        }
    }
}

cout << "\nRoad connections in MST (Prim's Algorithm):\n";

int totalCost = 0;

for (int i = 1; i < n; i++) {
    cout << "City " << parent[i] << " - City " << i
        << " (Cost: " << graph[i][parent[i]] << ")\n";
    totalCost += graph[i][parent[i]];
}

cout << "Total Construction Cost = " << totalCost << endl;
}

int main() {
    int n;

    cout << "Enter the number of cities: ";

    cin >> n;

    vector< vector< int > > graph(n, vector<int>(n));

    cout << "\nEnter the cost adjacency matrix (enter 0 if no direct road):\n";

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

```

```

        for (int j = 0; j < n; j++) {
            cin >> graph[i][j];
        }
    }

    prims(graph, n);

    return 0;
}

```

Output:-

```

C:\Users\A9975\Desktop\2414 >
Enter the number of cities: 5

Enter the cost adjacency matrix (enter 0 if no direct road):
1 2 3 4 5
6 7 8 9 10
0 3 6 9 12
1 3 5 7 0
2 4 6 8 0

Road connections in MST (Prim's Algorithm):
City 0 - City 1 (Cost: 6)
City 0 - City 2 (Cost: 0)
City 0 - City 3 (Cost: 1)
City 0 - City 4 (Cost: 2)
Total Construction Cost = 9

-----
Process exited after 101.5 seconds with return value 0
Press any key to continue . . .

```

Program:-

```

#include <iostream>

#include <vector>

#include <algorithm>

Using namespace std;

Struct Edge {

```

```

    Int src, dest, weight;
};

Bool compare(Edge a, Edge b) {
    Return a.weight < b.weight;
}

Int findParent(int v, vector<int>& parent) {
    If (parent[v] == v)
        Return v;
    Return parent[v] = findParent(parent[v], parent);
}

Void kruskal(vector<Edge>& edges, int n) {
    Sort(edges.begin(), edges.end(), compare);
    Vector<int> parent(n);
    For (int i = 0; i < n; i++) parent[i] = i;
    Vector<Edge> mst;
    Int i;
    Int totalWeight = 0;
    For (i=0; i<edges.size(); i++) {
        Int srcParent = findParent(edges[i].src, parent);
        Int destParent = findParent(edges[i].dest, parent);
        If (srcParent != destParent) {
            Mst.push_back(edges[i]);
            totalWeight += edges[i].weight;

```

```

        parent[srcParent] = destParent;
    }
}

Cout << "\nEdges in MST (Kruskal's):\n";

For (i=0; i<mst.size(); i++) {
    Cout << edges[i].src << " – " << edges[i].dest << " (Weight: " <<
edges[i].weight << ")\n";
}

Cout << "Total Cost = " << totalWeight << endl;
}

Int main() {
    Int n, e;

    Cout << "Enter the number of vertices: ";

    Cin >> n;

    Cout << "Enter the number of edges: ";

    Cin >> e;

    Vector< Edge > edges(e);

    Cout << "\nEnter each edge as: src dest weight\n";

    Cout << "(Vertices are numbered from 0 to " << n-1 << ")\n";

    For (int i = 0; i < e; i++) {
        Cin >> edges[i].src >> edges[i].dest >> edges[i].weight;
    }

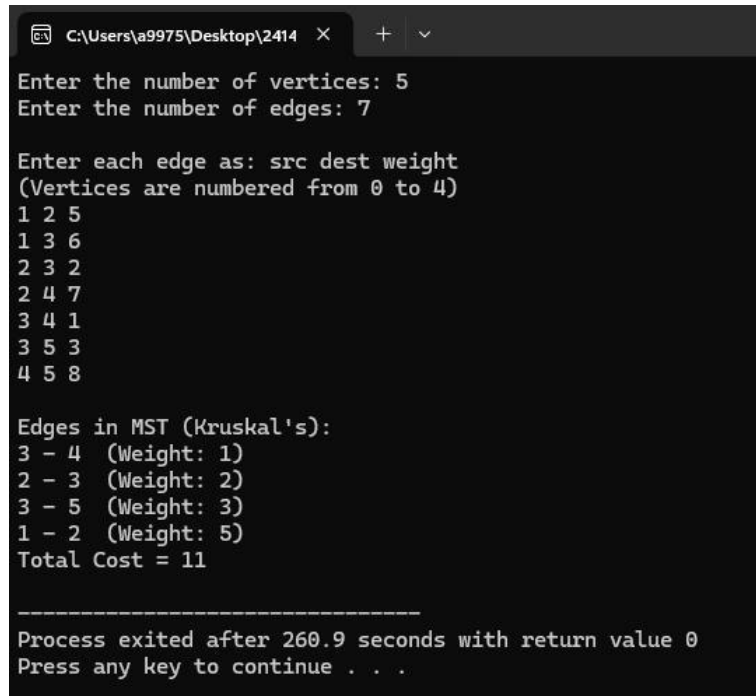
    kruskal(edges, n);
}

```



```
    return 0;
}
```

Output:-

A screenshot of a Windows command prompt window showing the execution of a C++ program. The window title is 'C:\Users\A9975\Desktop\2414'. The program prompts for the number of vertices (5) and edges (7). It then lists 7 edges with their source, destination, and weight. The output shows the edges selected for the Minimum Spanning Tree (MST) using Kruskal's algorithm, along with the total cost of 11. The program exits after 260.9 seconds with a return value of 0.

```
C:\Users\A9975\Desktop\2414 X + v
Enter the number of vertices: 5
Enter the number of edges: 7

Enter each edge as: src dest weight
(Vertices are numbered from 0 to 4)
1 2 5
1 3 6
2 3 2
2 4 7
3 4 1
3 5 3
4 5 8

Edges in MST (Kruskal's):
3 - 4 (Weight: 1)
2 - 3 (Weight: 2)
3 - 5 (Weight: 3)
1 - 2 (Weight: 5)
Total Cost = 11

-----
Process exited after 260.9 seconds with return value 0
Press any key to continue . . .
```

Application:-

```
#include <iostream>

#include <vector>

#include <algorithm>

Using namespace std;

Struct Edge {

    Int src, dest, weight;

};

Bool compare(Edge a, Edge b) {
```

```

    Return a.weight < b.weight;
}

Int findParent(int v, vector<int>& parent) {
    If (parent[v] == v)
        Return v;
    Return parent[v] = findParent(parent[v], parent);
}

Void kruskal(vector<Edge>& edges, int n) {
    Sort(edges.begin(), edges.end(), compare);
    Vector<int> parent(n);
    For (int i = 0; i < n; i++)
        Parent[i] = i;
    Vector<Edge> mst;
    Int totalCost = 0;
    For (int i = 0; i < edges.size(); i++) {
        Int srcParent = findParent(edges[i].src, parent);
        Int destParent = findParent(edges[i].dest, parent);
        If (srcParent != destParent) {
            Mst.push_back(edges[i]);
            totalCost += edges[i].weight;
            parent[srcParent] = destParent;
        }
    }
}

```

```

    Cout << "\nRoad connections in MST (Kruskal's Algorithm):\n";
    For (int i = 0; i < mst.size(); i++) {
        Cout << "City " << mst[i].src << " – City " << mst[i].dest
            << " (Cost: " << mst[i].weight << ")\n";
    }
    Cout << "Total Construction Cost = " << totalCost << endl;
}

Int main() {
    Int n, e;
    Cout << "Enter the number of cities: ";
    Cin >> n;
    Cout << "Enter the number of possible roads: ";
    Cin >> e;
    Vector<Edge> edges(e);
    Cout << "\nEnter each road as: City1 City2 Cost\n";
    For (int i = 0; i < e; i++) {
        Cin >> edges[i].src >> edges[i].dest >> edges[i].weight;
    }
    Kruskal(edges, n);
    Return 0;
}

```

Output:-

```
C:\Users\A9975\Desktop\2414 X + v
Enter the number of cities: 5
Enter the number of possible roads: 7

Enter each road as: City1 City2 Cost
1 2 4
1 3 6
2 3 8
2 4 10
3 4 5
3 5 12
4 5 9

Road connections in MST (Kruskal's Algorithm):
City 1 - City 2 (Cost: 4)
City 3 - City 4 (Cost: 5)
City 1 - City 3 (Cost: 6)
City 4 - City 5 (Cost: 9)
Total Construction Cost = 24

-----
Process exited after 53.76 seconds with return value 0
Press any key to continue . . .
```

Prim's Algorithm:-

Algorithm Steps:

1. Pick an arbitrary starting vertex.
2. Mark it as visited (add to MST set).
3. From all edges that connect visited to unvisited vertices, choose the minimum-weight edge.
4. Add the chosen edge and the new vertex to the MST set.
5. Repeat steps 3–4 until all vertices are included.

Time Complexity:

With adjacency matrix $\rightarrow O(V^2)$

With min-heap + adjacency list $\rightarrow O(E \log V)$

Kruskal's Algorithm:-

Algorithm Steps:-

1. Sort all edges in non-decreasing order of their weights.
2. Initialize an empty MST (no edges).
3. For each edge (u, v):

If adding (u, v) does not form a cycle (use Disjoint Set/Union-Find to check), then include it in the MST.

4. Stop when you have $(V - 1)$ edges in the MST.

Time Complexity:

Sorting edges $\rightarrow O(E \log E) \approx O(E \log V)$.

Union-Find operations \rightarrow nearly $O(1)$ each (amortized)

List of Applications:-

Applications of Prim's Algorithm

- Network design (like LAN, WAN, or telecommunication)
- Designing least-cost spanning trees
- Electrical grid and circuit design
- Cluster analysis in data science
- Approximation algorithms for NP-hard problems (like TSP)

Applications of Kruskal's Algorithm

- Network and road connectivity design
- Laying cables or pipelines with minimal cost
- Image segmentation in computer vision
- Constructing hierarchical clustering trees
- Designing railway or transportation networks