Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2024-25

Class:	**SE**	**Semester:**	**IV**
Course Code:	**CSL401**	**Course Name:**	**Analysis of Algorithm Lab**

Name of Student:	**Shravani Sandeep Raut**
Roll No. :	**48**
Experiment No.:	**6**
Title of the Experiment:	**Prim's Algorithm**
Date of Performance:	**13/02/2025**
Date of Submission:	**20/02/2025**

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Mrs. Sneha Yadav

Signature :

Date:

Experiment No. 6

Title: Prim's Algorithm.

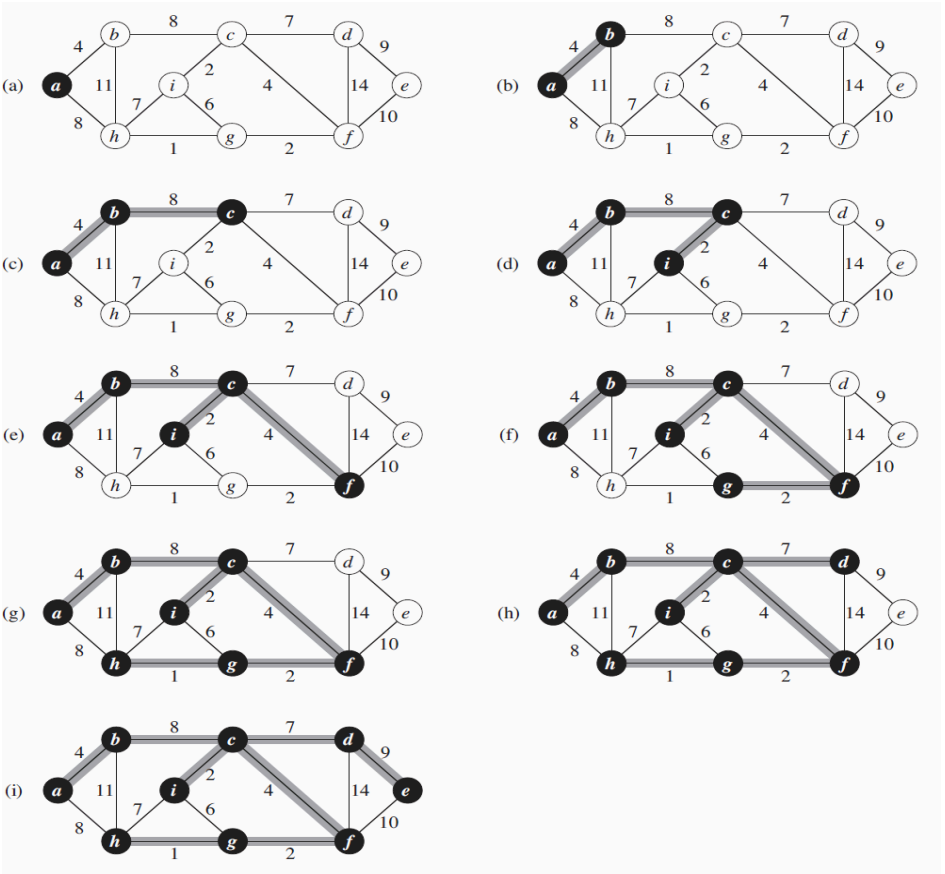
Aim: To study and implement Prim's Minimum Cost Spanning Tree Algorithm.

Objective: To introduce Greedy based algorithms

Theory:

Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The algorithm operates by building this tree one vertex at a time, from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex.

Example:



Algorithm and Complexity:

```
1   Algorithm Prim(E, cost, n, t)
2   // E is the set of edges in G. cost[1 : n, 1 : n] is the cost
3   // adjacency matrix of an n vertex graph such that cost[i, j] is
4   // either a positive real number or ∞ if no edge (i, j) exists.
5   // A minimum spanning tree is computed and stored as a set of
6   // edges in the array t[1 : n − 1, 1 : 2]. (t[i, 1], t[i, 2]) is an edge in
7   // the minimum-cost spanning tree. The final cost is returned.
8   {
9       Let (k, l) be an edge of minimum cost in E;
10      mincost := cost[k, l];
11      t[1, 1] := k; t[1, 2] := l;
12      for i := 1 to n do  // Initialize near.
13          if (cost[i, l] < cost[i, k]) then near[i] := l;
14          else near[i] := k;
15      near[k] := near[l] := 0;
16      for i := 2 to n − 1 do
17      { // Find n − 2 additional edges for t.
18          Let j be an index such that near[j] ≠ 0 and
19          cost[j, near[j]] is minimum;
20          t[i, 1] := j; t[i, 2] := near[j];
21          mincost := mincost + cost[j, near[j]];
22          near[j] := 0;
23          for k := 1 to n do // Update near[ ].
24              if ((near[k] ≠ 0) and (cost[k, near[k]] > cost[k, j]))
25                  then near[k] := j;
26      }
27      return mincost;
28  }
```

Time Complexity is O(n2), Where, n = number of vertices

Implementation:

```
#include <stdio.h>

#include <limits.h>

int n, i, j;  // Global variables

int graph[20][20], parent[20], key[20], mstSet[20];

void primMST();

int minKey();

void main() {

   printf("Enter the number of vertices: ");
```

```
    scanf("%d", &n);
    printf("Enter the adjacency matrix (0 for no edge):\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    primMST();
}
// Function to find the vertex with the minimum key value
int minKey() {
    int min = INT_MAX, minIndex;
    for (i = 0; i < n; i++) {
        if (mstSet[i] == 0 && key[i] < min) {
            min = key[i];
            minIndex = i;
        }
    }
    return minIndex;
}
// Function to implement Prim's Algorithm
void primMST() {
    for (i = 0; i < n; i++) {
        key[i] = INT_MAX;
```

```
        mstSet[i] = 0;
    }
    key[0] = 0;
    parent[0] = -1;
    for (i = 0; i < n - 1; i++) {
        int u = minKey();
        mstSet[u] = 1;

        for (j = 0; j < n; j++) {
            if (graph[u][j] && mstSet[j] == 0 && graph[u][j] < key[j]) {
                parent[j] = u;
                key[j] = graph[u][j];
            }
        }
    }

    printf("\nMinimum Spanning Tree (MST):\n");
    printf("Edge \tWeight\n");
    for (i = 1; i < n; i++) {
        printf("%d - %d\t%d\n", parent[i], i, graph[i][parent[i]]);
    }
}
```

```
Enter the number of vertices: 5
Enter the adjacency matrix (0 for no edge):
0 1 0 0 1
1 0 1 1 1
0 1 0 1 0
0 1 0 0 1
1 1 0 1 0

Minimum Spanning Tree (MST):
Edge    Weight
0 - 1   1
0 - 2   0
0 - 3   0
0 - 4   1
```

Conclusion:

Prim's algorithm serves as a classic example of a greedy-based technique in algorithm design. Its objective is to determine a minimum spanning tree (MST) in a weighted undirected graph, which ensures that all vertices are connected with the minimum possible total edge weight. Starting from an arbitrary vertex, the algorithm repeatedly selects the least costly edge that connects a vertex in the growing MST to a vertex outside it. This step-by-step greedy approach guarantees optimality, as it always chooses the next best local option, ultimately leading to a globally minimal spanning tree. This makes Prim's algorithm both intuitive and efficient for network design problems.