Experiment No. 6
Prim's Algorithm
Date of Performance:
Date of Submission:

Experiment No. 6

Title: Prim's Algorithm.

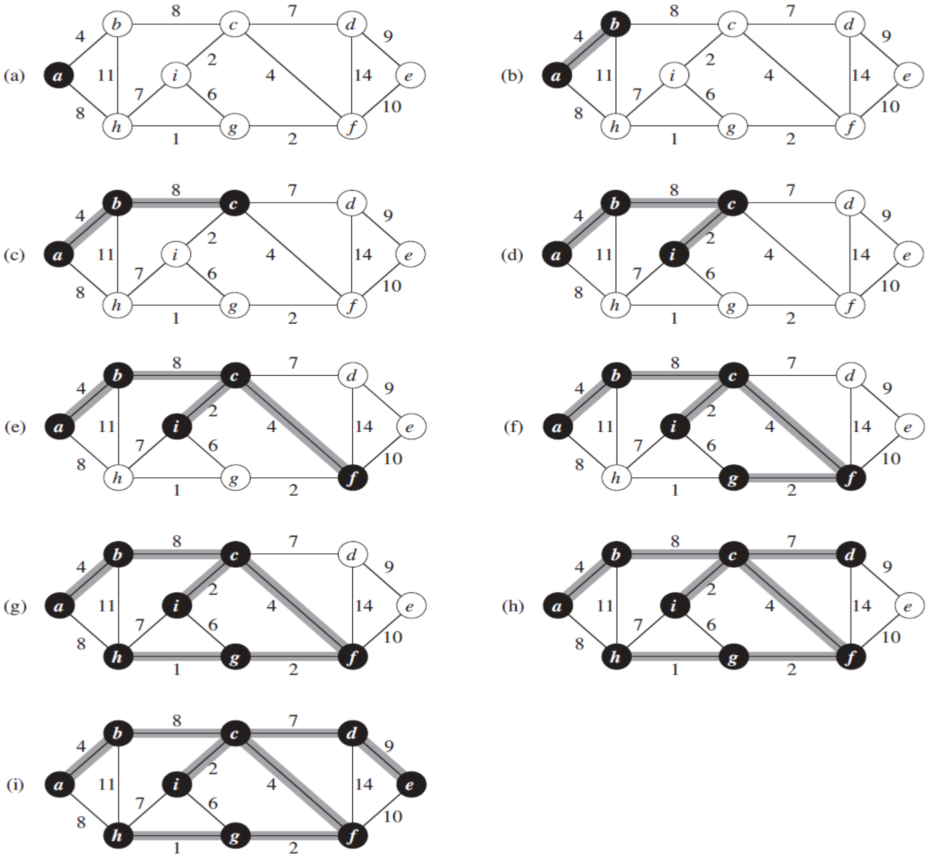
Aim: To study and implement Prim's Minimum Cost Spanning Tree Algorithm.

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:

```
Algorithm Prim(E, cost, n, t)
// E is the set of edges in G. cost[1 : n, 1 : n] is the cost
// adjacency matrix of an n vertex graph such that cost[i, j] is
// either a positive real number or ∞ if no edge (i, j) exists.
// A minimum spanning tree is computed and stored as a set of
// edges in the array t[1 : n − 1, 1 : 2]. (t[i, 1], t[i, 2]) is an edge in
// the minimum-cost spanning tree. The final cost is returned.
{
    Let (k, l) be an edge of minimum cost in E;
    mincost := cost[k, l];
    t[1, 1] := k; t[1, 2] := l;
    for i := 1 to n do  // Initialize near.
        if (cost[i, l] < cost[i, k]) then near[i] := l;
        else near[i] := k;
    near[k] := near[l] := 0;
    for i := 2 to n − 1 do
    { // Find n − 2 additional edges for t.
        Let j be an index such that near[j] ≠ 0 and
        cost[j, near[j]] is minimum;
        t[i, 1] := j; t[i, 2] := near[j];
        mincost := mincost + cost[j, near[j]];
        near[j] := 0;
        for k := 1 to n do // Update near[ ].
            if ((near[k] ≠ 0) and (cost[k, near[k]] > cost[k, j]))
                then near[k] := j;
    }
    return mincost;
}
```

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

Implemenation:

```
#include <stdio.h>

#include <stdlib.h>

#include <limits.h>



#define V 5 // Number of vertices in the graph



// Function to find the vertex with minimum key value,
```

```
// from the set of vertices not yet included in MST
int minKey(int key[], int mstSet[]) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == 0 && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}

// Function to print the constructed MST stored in parent[]
void printMST(int parent[], int graph[V][V]) {
    printf("Edge   Weight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d    %d \n", parent[i], i, graph[i][parent[i]]);
}

// Function to construct and print MST for a graph represented
// using adjacency matrix representation
void primMST(int graph[V][V]) {
    int parent[V]; // Array to store constructed MST
    int key[V];    // Key values used to pick minimum weight edge in cut
```

```
int mstSet[V]; // To represent set of vertices not yet included in MST


// Initialize all keys as INFINITE
for (int i = 0; i < V; i++)
    key[i] = INT_MAX, mstSet[i] = 0;


// Always include first vertex in MST.
// Make key 0 so that this vertex is picked as first vertex.
key[0] = 0;
parent[0] = -1; // First node is always root of MST


// The MST will have V vertices
for (int count = 0; count < V - 1; count++) {
    // Pick the minimum key vertex from the set of vertices
    // not yet included in MST
    int u = minKey(key, mstSet);


    // Add the picked vertex to the MST set
    mstSet[u] = 1;


    // Update key value and parent index of the adjacent vertices
    // of the picked vertex. Consider only those vertices which are
    // not yet included in MST
```

```
    for (int v = 0; v < V; v++)

      // graph[u][v] is non-zero only for adjacent vertices of m
      // mstSet[v] is false for vertices not yet included in MST
      // Update the key only if graph[u][v] is smaller than key[v]
      if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])
        parent[v] = u, key[v] = graph[u][v];
  }

  // Print the constructed MST
  printMST(parent, graph);
}

// Driver code
int main() {
  // Graph representation using adjacency matrix
  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}
  };
```

```
    // Print the MST
    primMST(graph);

    return 0;
}
```

Conclusion: Prim's algorithm has been successfully implemented.