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

# Experiment No. 6

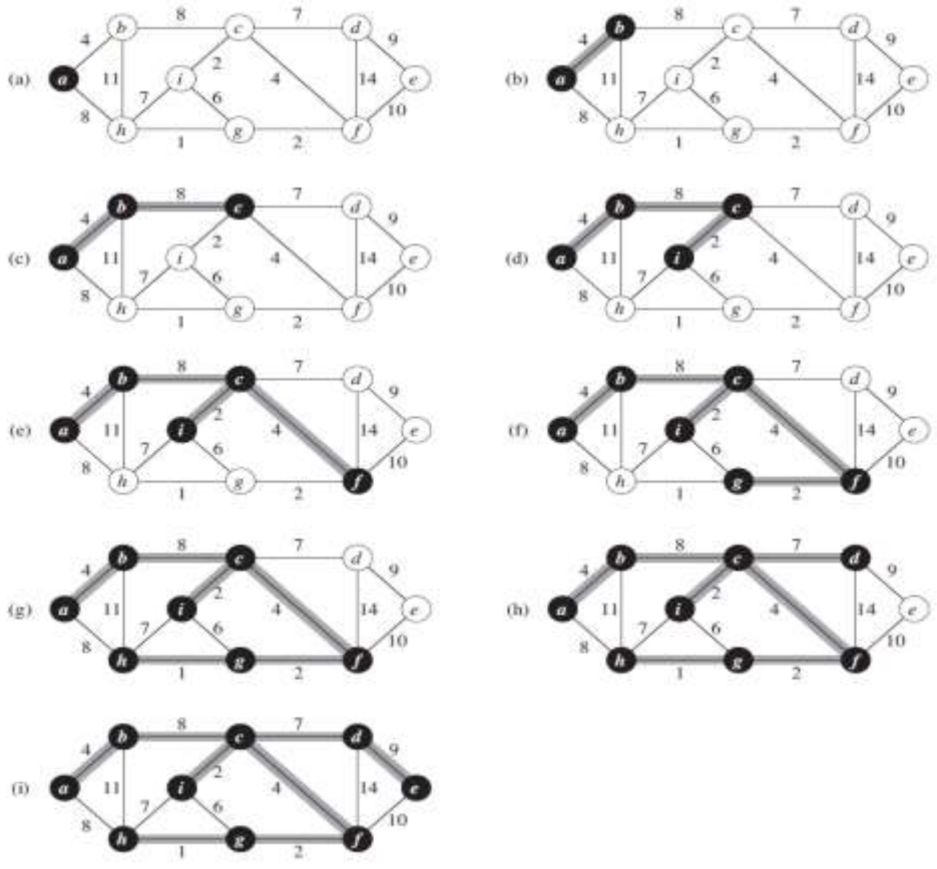
**Title:** Prim's Algorithm.

**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 $\infty$ 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 $near[j] \neq 0$ 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 $((near[k] \neq 0)$ 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:**

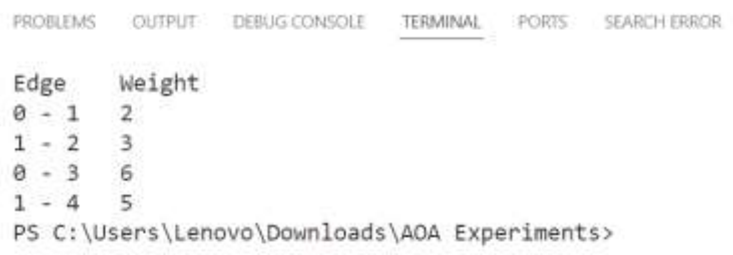
**Implementation:**

**Code:**

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>

#define V 5

int minKey(int key[], bool mstSet[])
```

```
{
    int min = INT_MAX, min_index;

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

    return min_index;
}

int printMST(int parent[], int graph[V][V])
{
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d \t%d \n", parent[i], i,
            graph[i][parent[i]]);
}

void primMST(int graph[V][V])
{
    int parent[V];
    int key[V];
    bool mstSet[V];

    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;
    key[0] = 0;
    parent[0] = -1;

    for (int count = 0; count < V - 1; count++) {

        int u = minKey(key, mstSet);

        mstSet[u] = true;

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

            if (graph[u][v] && mstSet[v] == false
                && 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
PS C:\Users\Lenovo\Downloads\AOA Experiments>
```

**Conclusion:**

In conclusion, the Prim's algorithm efficiently finds the Minimum Spanning Tree (MST) of a connected, undirected graph. The MST is a subgraph that includes all the vertices of the original graph with the minimum possible total edge weight, ensuring that there are no cycles and the graph remains connected.

Prim's algorithm begins with an arbitrary starting vertex and iteratively grows the MST by adding the nearest vertex not yet included in the MST and the edge that connects it to the existing MST. This process continues until all vertices are included in the MST.

This implementation of Prim's algorithm maintains arrays to keep track of the parent vertices and key values (minimum edge weights) for each vertex. It iteratively selects the vertex with the minimum key value from the set of vertices not yet included in the MST, updates the MST, and adjusts the key values accordingly.