# EXPERIMENT NO. 05

Roll No.:24141001
Batch:I1

**Title:** Minimum cost spanning tree of a given undirected graph using Prim's algorithm and Kruskal's algorithm and compare.

**Introduction:**

A Minimum Cost Spanning Tree (MST) of an undirected graph connects all vertices together with the minimum total edge weight, without forming any cycles. Two popular greedy algorithms to find MST are Prim's and Kruskal's algorithms:

- Prim's Algorithm starts from any vertex and grows the MST by adding the nearest vertex at each step, always choosing the smallest adjacent edge connecting visited and unvisited vertices. It is ideal for dense graphs.
- Kruskal's Algorithm works by sorting all edges by ascending weight and adding edges one at a time to the MST, as long as they don't create cycles. It is efficient for sparse graphs and uses union-find to detect cycles.

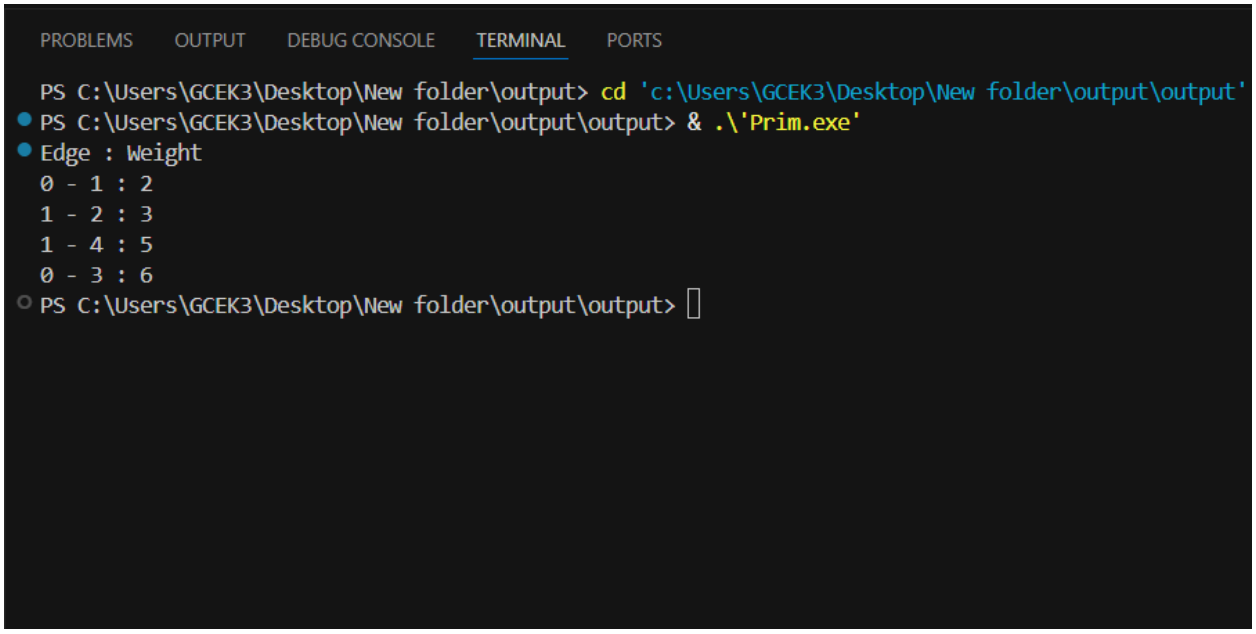
**1)Prim's Algorithm**

Program:

```
#include <stdio.h>
#define INF 9999999

int main() {
   int cost[5][5] = {
      {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}
   };
   int visited[5] = {0};
   int i, j, n = 5, edges = 0;
   visited[0] = 1;

   printf("Edge : Weight\n");
   while (edges < n - 1) {
      int min = INF, a = -1, b = -1;
      for (i = 0; i < n; i++) {
```

```
        if (visited[i]) {
            for (j = 0; j < n; j++) {
                if (!visited[j] && cost[i][j]) {
                    if (cost[i][j] < min) {
                        min = cost[i][j];
                        a = i;
                        b = j;
                    }
                }
            }
        }
        printf("%d - %d : %d\n", a, b, min);
        visited[b] = 1;
        edges++;
    }
    return 0;
}
```

OUTPUT:




**2)Kruskal's Algorithm**

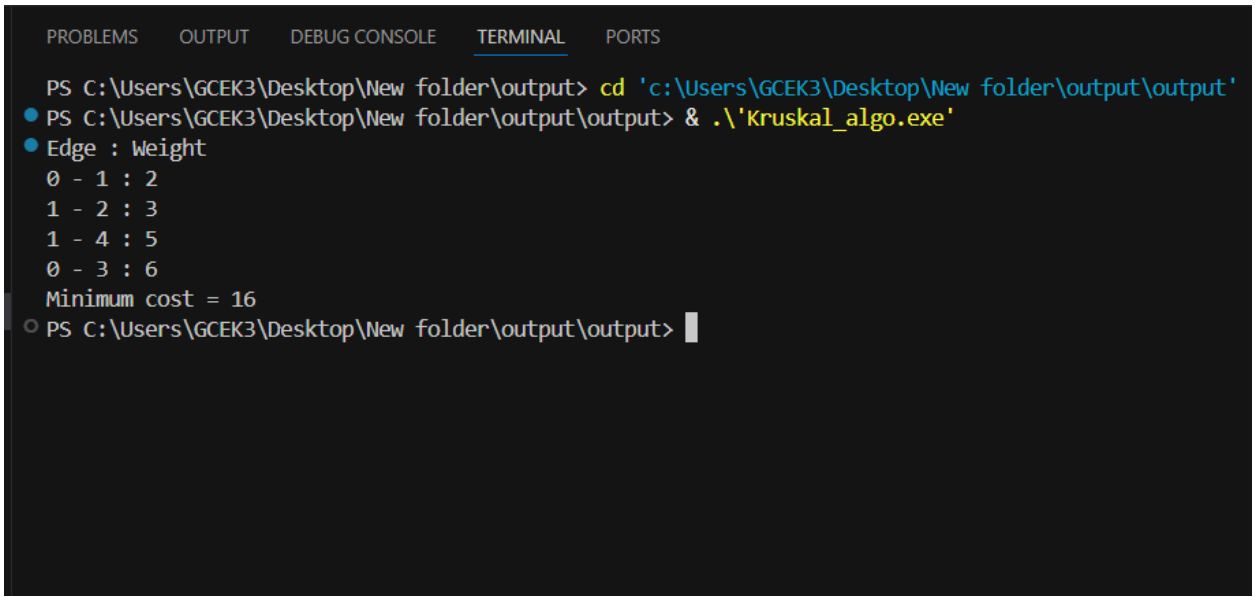
Program:

```
#include <stdio.h>
#define MAX 30
```

```
typedef struct {
   int u, v, w;
} Edge;

int parent[MAX];

int find(int i) {
   while (parent[i]) i = parent[i];
   return i;
}

int uni(int i, int j) {
   if (i != j) {
      parent[j] = i;
      return 1;
   }
   return 0;
}

int main() {
   Edge edge[MAX];
   int n = 5, e = 7;
   Edge input[] = {
      {0, 1, 2}, {1, 2, 3}, {0, 3, 6}, {1, 3, 8},
      {1, 4, 5}, {2, 4, 7}, {3, 4, 9}
   };
   for (int i = 0; i < e; i++)
      edge[i] = input[i];
   int count = 0, mincost = 0;
   printf("Edge : Weight\n");
   while (count < n - 1) {
      int min = 999999, a = -1;
      for (int j = 0; j < e; j++) {
         if (edge[j].w < min) {
            min = edge[j].w; a = j;
         }
      }
      int u = find(edge[a].u), v = find(edge[a].v);
      if (uni(u, v)) {
         printf("%d - %d : %d\n", edge[a].u, edge[a].v, edge[a].w);
```

```
            mincost += edge[a].w;
            count++;
        }
        edge[a].w = 999999;
    }
    printf("Minimum cost = %d\n", mincost);
    return 0;
}
```

OUTPUT:




**Comparison:**

- Prim's algorithm grows the MST one vertex at a time by choosing the smallest adjacent edge but Kruskal's adds the smallest edges globally without forming cycles.
- Prim's is best for dense graphs but Kruskal's is better for sparse graphs.
- Prim's uses priority queues but Kruskal's uses union-find data structures.
- Prim's algorithm implicitly avoids cycles by growing connected components but Kruskal's explicitly manages cycles using union-find.
- Prim's requires a connected graph but Kruskal's can handle disconnected graphs and produce a forest.
- Kruskal's algorithm is generally easier to implement but Prim's can be more complex due to priority queue management.
- Prim's algorithm starts from an arbitrary vertex but Kruskal's starts with the smallest weight edge.

**Conclusion:**

Prim's and Kruskal's algorithms are fundamental greedy approaches to finding the Minimum Cost Spanning Tree. While both guarantee an MST with minimal total edge weight, their approaches, efficiencies, and best-use scenarios differ. Prim's algorithm excels when dealing with dense graphs and connected components by growing from a starting vertex. Kruskal's algorithm is effective for sparse or disconnected graphs as it works globally with edges, ensuring no cycles using union-find structures. Mastery of both algorithms and their underlying principles is essential for solving practical network and optimization problems.