Experiment 5: Kruskal/Prims Algorithm

AIM: To study & implement the Kruskal/Prims Algorithm (using Greedy approach)

THEORY:

The Kruskal and Prim's algorithms are two fundamental approaches in graph theory used for finding the minimum spanning tree (MST) of a connected, undirected graph. Both algorithms employ a greedy strategy, making locally optimal choices at each step with the goal of finding the globally optimal solution, i.e., the minimum spanning tree.

Kruskal's Algorithm:

Kruskal's algorithm starts by sorting all the edges of the graph in non-decreasing order of their weights. It then iterates through these sorted edges, adding each edge to the MST if it does not create a cycle. It maintains a forest of trees initially, where each tree is a single vertex. As it adds edges to the MST, it merges smaller trees into larger ones until all vertices are connected.

The key idea behind Kruskal's algorithm is the disjoint-set data structure, which efficiently keeps track of the connected components of the graph. This data structure allows the algorithm to determine whether adding an edge creates a cycle in near-constant time.

Prim's Algorithm:

Prim's algorithm, on the other hand, starts with an arbitrary vertex as the initial MST and then grows the MST one vertex at a time. At each step, it selects the edge with the minimum weight that connects a vertex in the MST to a vertex outside the MST. This process continues until all vertices are included in the MST.

Prim's algorithm relies on the concept of priority queues, where vertices are prioritized based on their minimum edge weight connecting them to the MST. It maintains a set of vertices in the MST and updates the priorities of vertices adjacent to the MST as edges are added.

Comparing the Algorithms:

Both Kruskal's and Prim's algorithms guarantee the construction of a minimum spanning tree, but they differ in their implementations and efficiencies.

Kruskal's algorithm is generally preferred for sparse graphs, where the number of edges is much less than the number of vertices. Its time complexity is O(E log E), where E is the number of edges.

Prim's algorithm, on the other hand, is more efficient for dense graphs, where the number of edges is close to the number of vertices. Its time complexity is O(V^2) with a simple array-based implementation and can be improved to O(E + V log V) using more advanced data structures like Fibonacci heaps.

CODE:

Kruskal algorithm:

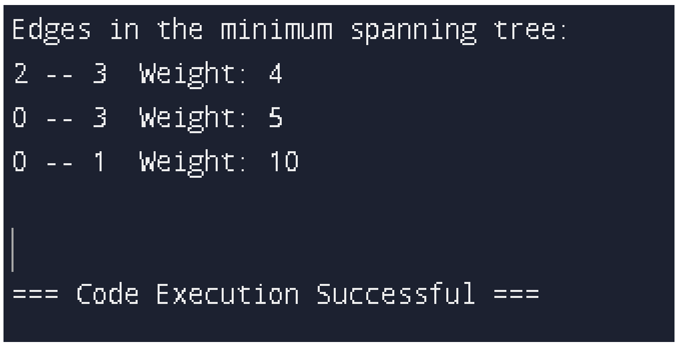
```
#include <stdio.h>
#include <stdlib.h>


#define MAX_VERTICES 100
#define MAX_EDGES 100


typedef struct {
   int u;
   int v;
   int weight;
} Edge;
typedef struct {
   int parent;
   int rank;
```

```
} Subset;


int find(Subset subsets[], int i) {
   if (subsets[i].parent != i)
      subsets[i].parent = find(subsets, subsets[i].parent);
   return subsets[i].parent;
}


void Union(Subset subsets[], int x, int y) {
   int xroot = find(subsets, x);
   int yroot = find(subsets, y);
   if (subsets[xroot].rank < subsets[yroot].rank)
      subsets[xroot].parent = yroot;
   else if (subsets[xroot].rank > subsets[yroot].rank)
      subsets[yroot].parent = xroot;
   else {
      subsets[yroot].parent = xroot;
      subsets[xroot].rank++;
   }
}


int comparator(const void* a, const void* b) {
   return ((Edge*)a)->weight - ((Edge*)b)->weight;
}


void kruskalMST(Edge edges[], int V, int E) {
   Edge result[V];
   Subset subsets[V];
   int e = 0, i = 0;
```

```
    for (i = 0; i < V; ++i) {
        subsets[i].parent = i;
        subsets[i].rank = 0;
    }


    qsort(edges, E, sizeof(Edge), comparator);


    i = 0;
    while (e < V - 1 && i < E) {
        Edge next_edge = edges[i++];
        int x = find(subsets, next_edge.u);
        int y = find(subsets, next_edge.v);
        if (x != y) {
            result[e++] = next_edge;
            Union(subsets, x, y);
        }
    }


   printf("Edges in the minimum spanning tree:\n");
    for (i = 0; i < e; ++i)
        printf("%d -- %d\tWeight: %d\n", result[i].u, result[i].v,
result[i].weight);
}


int main() {
    int V = 4; // Number of vertices
    int E = 5; // Number of edges
    Edge edges[MAX_EDGES] = {{0, 1, 10}, {0, 2, 6}, {0, 3, 5}, {1, 3, 15}, {2,
3, 4}};
    kruskalMST(edges, V, E);
    return 0;
```

```
}
```

Output



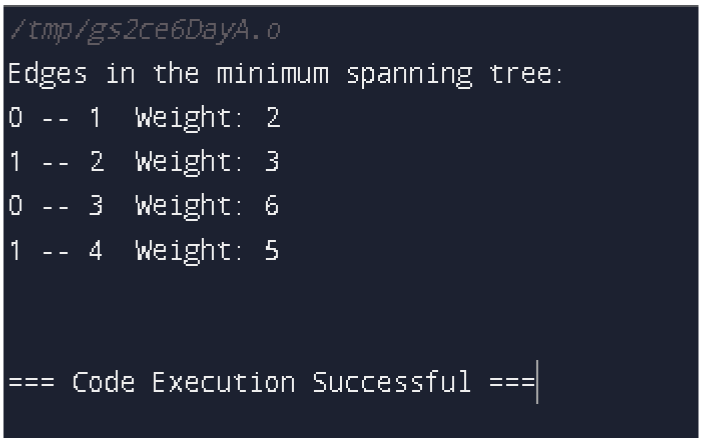
```
Edges in the minimum spanning tree:
2 -- 3  Weight: 4
0 -- 3  Weight: 5
0 -- 1  Weight: 10

|
=== Code Execution Successful ===
```

Prim’s Algorithm:

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


#define MAX_VERTICES 100
int minKey(int key[], int mstSet[], int V) {
   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;
}


void printMST(int parent[], int
graph[MAX_VERTICES][MAX_VERTICES], int V) {
   printf("Edges in the minimum spanning tree:\n");
   for (int i = 1; i < V; i++)
      printf("%d -- %d\tWeight: %d\n", parent[i], i,
graph[i][parent[i]]);
}
```

```
void primMST(int
graph[MAX_VERTICES][MAX_VERTICES], int V) {
   int parent[V];
   int key[V];
   int mstSet[V];


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


   key[0] = 0;
   parent[0] = -1;


   for (int count = 0; count < V - 1; count++) {
      int u = minKey(key, mstSet, V);
      mstSet[u] = 1;


      for (int v = 0; v < V; v++)
         if (graph[u][v] && mstSet[v] == 0 && graph[u][v] <
key[v])
            parent[v] = u, key[v] = graph[u][v];
   }


   printMST(parent, graph, V);
}


int main() {
   int V = 5; // Number of vertices
   int graph[MAX_VERTICES][MAX_VERTICES] = {
      {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, V);
    return 0;
}
```

OUTPUT:



```
/tmp/gs2ce6DayA.o
Edges in the minimum spanning tree:
0 -- 1  Weight: 2
1 -- 2  Weight: 3
0 -- 3  Weight: 6
1 -- 4  Weight: 5


=== Code Execution Successful ===
```

CONCLUSION:

In summary, both Kruskal's and Prim's algorithms offer efficient solutions to the minimum spanning tree problem. Kruskal's algorithm focuses on sorting edges by weight and then greedily adding them to the MST, while Prim's algorithm starts with a single vertex and grows the MST incrementally. The choice between the two algorithms often depends on the characteristics of the graph being analyzed, with Kruskal's algorithm preferred for sparse graphs and Prim's algorithm for dense graphs.