23CCE201 DATA STRUCTURES

ASSIGNMENT

NAME: VETHAVIYASH.K **ROLL.NO:** CB.EN.U4CCE23063

Report on Schemes to Obtain Minimum Spanning Trees for Weighted Undirected Graphs

---

1. Aim:

To explore different algorithms for obtaining the Minimum Spanning Tree (MST) of a weighted undirected graph, ensuring the MST connects all vertices with the minimum possible total edge weight without forming cycles. This study focuses on **Prim's Algorithm** and **Kruskal's Algorithm**, each employing distinct approaches for constructing MSTs.

---

2. Logic Followed:

- **Kruskal's Algorithm**: It picks the smallest available edge from the entire graph each time and checks if it connects two separate parts. If it does, we add it to the MST. It keeps doing this until all nodes are connected. A special structure (Union-Find) is used to help keep track of which nodes are already connected.
- **Prim's Algorithm**: It starts from a random node and looks for the smallest edge connecting that node to any new node not yet in the MST. It keeps growing the MST by connecting new nodes one by one through the smallest edges, using a list (priority queue) to keep track of the smallest edges efficiently.

---

3. Algorithm:

Prim's Algorithm:

1. **Initialize each vertex**:

   - Set the "key" value of all vertices to infinity.
   - Set the parent of each vertex to "NIL" (meaning it has no parent yet).
   - Set the starting vertex's "key" value to 0.

2. **Add all vertices** to a queue that will keep track of vertices not yet in the minimum spanning tree.
3. **Repeat until the minimum spanning tree is complete**:

   - Pick the vertex with the smallest "key" value from the queue.
   - For each neighbor of this vertex:
     - If the neighbor is still in the queue and the edge connecting the two vertices has a weight smaller than the neighbor's current "key" value, then:
       - Set the parent of the neighbor to the current vertex.
       - Update the neighbor's "key" value to the weight of the connecting edge.

4. **Continue** this process until all vertices are connected.

Kruskal's Algorithm:

1. **Initialize:**
   Start with a graph G consisting of vertices and edges. Form a new graph T that includes all vertices but no edges.
2. **Sort Edges:**
   Gather all edges from G and organize them by weight. Order them from lowest to highest cost.
3. **Select Edges:**
   Examine the sorted edges:
   - Choose the edge with the lowest weight.
   - If it connects two different components without creating a cycle, include it in T.
   - Continue this process until T has n−1 edges, where n is the total number of vertices.
4. **Complete:**
   The resulting graph T represents the Minimum Spanning Tree (MST) of G. It links all vertices with the minimum total weight

4. Codes:

Kruskal's Algorithm:

```
import sys

class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    def add_edge(self, u, v, w):
        self.graph.append((u, v, w))

    def find(self, parent, i):
        if parent[i] != i:
            parent[i] = self.find(parent, parent[i])
        return parent[i]

    def union(self, parent, rank, x, y):
        if rank[x] < rank[y]:
            parent[x] = y
        elif rank[x] > rank[y]:
            parent[y] = x
        else:
            parent[y] = x
            rank[x] += 1

    def kruskal_mst(self):
        result = []
        self.graph.sort(key=lambda x: x[2])  # Sort edges by weight
        parent = list(range(self.V))
        rank = [0] * self.V

        for u, v, w in self.graph:
            x, y = self.find(parent, u), self.find(parent, v)
            if x != y:  # Check for cycle
                result.append((u, v, w))
                self.union(parent, rank, x, y)

        print("\nEdge \tWeight")
        for u, v, weight in result:
            print(f"{u} - {v}    {weight}")

def input_graph():
    vertices = int(input("\nEnter the number of vertices: "))
    g = Graph(vertices)

    g.graph = []
    print("Enter the adjacency matrix (row by row):")
    for i in range(vertices):
        row = list(map(int, input(f"Row {i + 1}: ").split()))
        for j in range(i + 1, vertices):
```

```
            if row[j] != 0:
                g.add_edge(i, j, row[j])

    g.kruskal_mst()

input_graph()
```

Prim's Algorithm:

```
import sys

class Graph():
    def __init__(self, vertices):
        self.V = vertices  # Number of vertices
        self.graph = [[0] * vertices for _ in range(vertices)]  # Adjacency
matrix

    def printMST(self, parent):
        """Print the edges of the Minimum Spanning Tree."""
        print("\nEdge \tWeight")
        for i in range(1, self.V):
            print(f"{parent[i]} - {i} \t {self.graph[i][parent[i]]}")

    def primMST(self):
        """Implement Prim's algorithm to find the MST."""
        key = [sys.maxsize] * self.V  # Initialize all keys as infinite
        parent = [-1] * self.V  # Array to store constructed MST
        key[0] = 0  # Start from the first vertex
        mstSet = [False] * self.V  # Track vertices included in the MST

        for _ in range(self.V):
            # Select the minimum key vertex not yet included in the MST
            u = min((key[i], i) for i in range(self.V) if not mstSet[i])[1]
            mstSet[u] = True  # Include the selected vertex in the MST

            # Update the key and parent for adjacent vertices
            for v in range(self.V):
                if self.graph[u][v] and not mstSet[v] and key[v] >
self.graph[u][v]:
                    key[v], parent[v] = self.graph[u][v], u

        self.printMST(parent)  # Print the result

def input_graph():
    """Input the graph from the user."""
    vertices = int(input("\nEnter the number of vertices: "))
    g = Graph(vertices)

    print("Enter the adjacency matrix (row by row):")
    for i in range(vertices):
        g.graph[i] = list(map(int, input(f"Row {i + 1}: ").split()))

    g.primMST()  # Compute and display the MST

input_graph()
```

---

5. Results:

Prim's Algorithm:-

Output:

- The output will list the edges included in the Minimum Spanning Tree (MST).
- Each edge will be displayed with its corresponding weight.
- The total weight of all edges in the MST will be summarized.

Kruskal's Algorithm:-

Output:

- Similar to Prim's, the output will show the edges that make up the MST.
- Each edge will also be accompanied by its weight.
- A summary of the total weight for all included edges will be provided.

---

6. Inferences:

The main concept behind creating a Minimum Spanning Tree (MST) is to start with no edges and then add edges one by one. Only edges that won't form a loop (cycle) and have the smallest possible weight are chosen.

- **Prim's Algorithm**: Adds edges based on the smallest weight connected to the growing MST. It starts with one vertex and gradually adds more.
- **Kruskal's Algorithm**: Focuses on selecting the smallest edges from the entire graph. It adds edges without forming cycles until all vertices are connected.
