Chapter 4.3: Minimum Spanning Trees

Chapter 4.3: Minimum Spanning Trees

Introduction

A Minimum Spanning Tree (MST) of a weighted graph is a spanning tree whose sum of edge weights is minimized. MSTs are crucial in network design, including computer networks, telecommunications, and transportation networks.

Terminology

- **Spanning Tree:** A subgraph that includes all the vertices of the original graph and is a single connected tree.
- **Weight of a Spanning Tree:** Sum of the weights of the edges in the spanning tree.
- **Minimum Spanning Tree (MST):** A spanning tree with the minimum possible total edge weight.

Properties of MST

1. **Uniqueness:** If all edge weights are distinct, there is a unique MST.
2. **Subgraph of a Graph:** An MST is a subgraph that includes all vertices with minimum edge weight sum.
3. **Cycle Property:** For any cycle in the graph, the edge with the maximum weight in the cycle cannot be in the MST.

Algorithms to Find MST

Kruskal's Algorithm

Kruskal's algorithm is a greedy algorithm that finds an MST by sorting all the edges in increasing order of their weight and adding edges to the MST, skipping those that would form a cycle.

Kruskal's Algorithm Steps

1. Sort all edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If a cycle is not formed, include this edge. Else, discard it.

3. Repeat step 2 until there are (V-1) edges in the spanning tree.

Kruskal's Algorithm Implementation

```java
public class KruskalMST {
    private Queue<Edge> mst = new Queue<Edge>();

    public KruskalMST(EdgeWeightedGraph G) {
        MinPQ<Edge> pq = new MinPQ<Edge>();
        for (Edge e : G.edges()) {
            pq.insert(e);
        }

        UF uf = new UF(G.V());
        while (!pq.isEmpty() && mst.size() < G.V() - 1) {
            Edge e = pq.delMin();
            int v = e.either(), w = e.other(v);
            if (!uf.connected(v, w)) {
                uf.union(v, w);
                mst.enqueue(e);
            }
        }
    }
```

```
    public Iterable<Edge> edges() {
        return mst;
    }
}
```

Prim's Algorithm

Prim's algorithm is a greedy algorithm that finds an MST by building it one vertex at a time, starting from an arbitrary vertex and adding the smallest edge that expands the tree.

Prim's Algorithm Steps

1. Start with an arbitrary vertex and add it to the MST.
2. Find the smallest edge connecting the MST to a vertex not in the MST.
3. Add the edge and the vertex to the MST.
4. Repeat steps 2 and 3 until all vertices are included in the MST.

Prim's Algorithm Implementation

```java
public class PrimMST {
    private Edge[] edgeTo;
    private double[] distTo;
    private boolean[] marked;
    private IndexMinPQ<Double> pq;

    public PrimMST(EdgeWeightedGraph G) {
        edgeTo = new Edge[G.V()];
```

```
    distTo = new double[G.V()];
    marked = new boolean[G.V()];
    pq = new IndexMinPQ<Double>(G.V());
    for (int v = 0; v < G.V(); v++) {
        distTo[v] = Double.POSITIVE_INFINITY;
    }
    distTo[0] = 0.0;
    pq.insert(0, 0.0);
    while (!pq.isEmpty()) {
        visit(G, pq.delMin());
    }
}

private void visit(EdgeWeightedGraph G, int v) {
    marked[v] = true;
    for (Edge e : G.adj(v)) {
        int w = e.other(v);
        if (marked[w]) continue;
        if (e.weight() < distTo[w]) {
            edgeTo[w] = e;
            distTo[w] = e.weight();
            if (pq.contains(w)) pq.changeKey(w, distTo[w]);
            else pq.insert(w, distTo[w]);
        }
    }
}
```

```
    public Iterable<Edge> edges() {
        Queue<Edge> mst = new Queue<Edge>();
        for (int v = 1; v < edgeTo.length; v++) {
            mst.enqueue(edgeTo[v]);
        }
        return mst;
    }
}
```

Performance

- **Kruskal's Algorithm:** O(E log E) due to sorting of edges and union-find operations.
- **Prim's Algorithm:** O(E log V) with a priority queue.

Applications

1. **Network Design:** Designing efficient networks such as electrical grids, computer networks, and transportation systems.
2. **Approximation Algorithms:** Used in algorithms to find approximate solutions to NP-hard problems.
3. **Cluster Analysis:** Grouping objects into clusters based on their distances.

Conclusion

Minimum Spanning Trees are fundamental in optimizing the design and operation of various networks. Algorithms like Kruskal's and Prim's provide efficient means to find MSTs, making them indispensable tools in computer science and operations research.