Mini-Project	
A Comparative Study on the Minimum Spanning Tree (MST)	
CSE246 Section: 3	
Group Number: 9	
Md. Shafayat Hossain	**2019-3-60-034**
Rizvee Hassan Prito	**2019-3-60-041**

Introduction-

What is MST? A MST or Minimum Spanning Tree is a special kind of tree which minimizes the cost of the edges of a graph. MST is a subset of edges of a connected graph that connects all the vertices of an undirected graph with weighted edges without forming any cycle and with the minimum possible total edge weight/cost. A spanning tree of a graph is a tree that contains all the original vertices of that graph, reaches out to all the vertices, is acyclic. But for a minimum spanning tree the cost of the edges has to be minimum. There are many use cases of a MST. One popular example is Telecommunications companies trying to lay cables in a neighborhood with minimum cost. Since some paths might be longer than others for which the cost of connecting the houses with a cable will be more costly. In this case for the most minimum cost they use MST to find a specific path where they will lay the cables to connect the neighborhood houses. There are many MST algorithms which are useful as they find many minimum connected paths across various components. Some popular algorithms are Prim's Algorithm & Kruskal's Algorithm.

Applications-

Minimum spanning tree is used in many areas of real life. It is applied in-

- Network Designs: Design of networks, including computer networks, telecommunications networks, transportation networks, water supply networks, and electrical grids.
- Traveling salesman problem: Traveling salesman problem is “Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?”. This problem can be solved easily by using the minimum spanning tree.
- Clustering: In the field of data mining, minimum spanning tree (MST)-based clustering is one of the most important clustering techniques.
- Constructing trees for broadcasting in computer networks.
- Image registration and segmentation.
- Curvilinear feature extraction in computer vision.
- Handwriting recognition of mathematical expressions.
- Circuit designing
- Regionalization of socio-geographic areas, the grouping of areas into homogeneous, contiguous regions.
- Comparing ecotoxicology data.
- Topological observability in power systems.
- Measuring homogeneity of two-dimensional materials.
- Minimax process control.
- Real-time face tracking and verification: Locating human faces in a video stream.

Algorithms-

Here, we will be discussing 4 MST algorithms, how they work and their time complexity. The algorithms are,

1. Prim’s Algorithm
2. Kruskal’s Algorithm
3. Boruvka’s Algorithm
4. Reverse Delete Algorithm

For all the algorithms, input will be a connected undirected graph with weighted edges and the output will be a minimum spanning tree.

Prim’s Algorithm-

Prim’s algorithm is a greedy approach to solving the minimum spanning tree (MST) problems. In this algorithm it starts with an empty spanning tree and slowly adds edges to make a MST. The algorithm maintains two sets of vertices. The first set contains the vertices already included in the MST, and the other set contains the vertices not yet included. In each step, it picks the minimum weight edge from the edges adjacent to the already connected vertices.

In this algorithm 1st it sorts the weighted edges in an ascending order. Initially it will pick a vertice with the highest priority or selected source vertices and then add the minimum weighted edge adjacent to the initial vertice. It will keep adding minimum weighted edges adjacent to the vertices connected in the MST till the edge count is (V-1). Also if the connection creates a cycle then the algorithm will
skip that edge and move to the next edge. That is how Prim’s algorithm creates a MST.

Time complexity may vary for Prim's algorithm depending on the type of data structures used.

Table1: Time Complexity for Prim’s Algorithm

Minimum edge weight Data-structure	Time complexity
Adjacency matrix	O(V^2)
Binary heap & Adjacency list	O(E log V)
Fibonacci heap & Adjacency list	O(E + V log V)

Differences with the other MST algorithms are,

1. Prim’s algorithm is almost similar to Kruskal’s algorithm. In both cases the algorithms try to take the minimum weighted edge and connect it to the MST. In the case of Prim’s, the algorithm takes the minimum edge adjacent to the vertices in the MST but in Kruskal any minimum edge can be connected. But in both cases we have to keep in mind that by connecting them it must not form a cycle. Both Prim’s and Kruskal’s algorithm runs till the edge count is (V-1).
2. In the case of Reverse Delete Algorithm and prim’s algorithm the difference is that Reverse Delete Algorithm 1st takes a graph and then deletes the edges with the highest weight on a descending order till the edge count is (v-1) while keeping in check that deleting the edge does not disconnect the graph. But

Prims' algorithm connects edges with the smallest weight in an ascending order adjacent to the vertices in the MST till edge count is (V-1) while keeping in check that no cycle is formed when connecting.

3. Boruvka's algorithm finds all of the minimum edge incidents of each node repeatedly in parallel until there is exactly one vertex remaining. Then add those vertices with the minimum edge incidents to form a tree. Where prims take the minimum weighted edge adjacent to the vertices of the MST. By combining both Prim's and Boruvka's algorithms a new type of Fast parallel algorithms can be formed.

Kruskal's Algorithm-

Kruskal's algorithm is also a greedy algorithm which finds the minimum spanning tree from an input graph like weighted undirected graph. This algorithm finds the minimum spanning tree by finding the cheapest edges in the input graph. This algorithm can also find minimum spanning forest from the graph which is not connected.
Kruskal algorithm first sorts all the edges in ascending order of their weight. Then it takes the smallest edge and checks if this edge forms any cycle in the MST. If not, then it includes that edge. Otherwise, it leaves that edge. Algorithm then repeats the process of including edges and the algorithm terminates itself when it includes V-1 edges. This is how the Kruskal algorithm finds the minimum spanning tree from a weighted undirected graph.
Time complexity of Kruskal's algorithm:
With the Disjoint Set Union data structure, Kruskal algorithm takes $O(E \log_2 E)$ time where V is the vertices and E is the edges. Here is how-

- Sorting the edges in the graph-> E edges-> $O(E \log_2 E)$
- Loop over all the edges in the list: O(E).
  Calling the DSU's merge() function for each edge: $O(\log_2 V)$.
  Here, the DSU merge() function checks if the edge making cycle is in the MST. If not, then it includes the edge in the MST.
- Combined time complexity of step 2: $O(E \log_2 V)$
- Overall time complexity: $O(E \log_2 E + E \log_2 V) = O(E \log_2 E)$

Differences with the other MST algorithms are,

1. Kruskal algorithm just takes the minimum weighted edges from the graphs and forms MST by including them whereas the prims algorithm takes the minimum weighted edges of each vertex.

2. Kruskal algorithm takes the cheapest edges which do not form a cycle. After taking V-1 edges, it forms an MST. In Boruvka's algorithm, the algorithm starts from a vertex and takes the minimum weighted edge for each vertex and forms trees with those edges. Then It repeats the process of taking the minimum weighted edge for each vertex or tree. This algorithm can easily do this process parallely which Kruskal algorithm can not do.
3. Reverse delete algorithm sorts the weighted edges in descending order and deletes the edges of maximum weight from the sorted list but before deleting the edge it checks if that edge is not disconnecting the graph. If it disconnects the graph then this algorithm skips that edge. So this is the full opposite version of Kruskal Algorithm because Kruskal Algorithm sorts the weighted edges in ascending order and adds the edges of minimum weight from the sorted list and before adding the edge in the MST it checks if that edge forms any cycle.

Boruvka's Algorithm-

Boruvka's algorithm is a greedy algorithm like all other MST algorithms for finding a minimum spanning tree in a graph. Boruvka's algorithm finds MST by finding the cheapest edge of each vertex from the input graph like connected, weighted and undirected graphs.
From an input graph, this algorithm first finds the minimum-weighted edge of each vertex in the graph. Then creates trees by adding those edges. It then repeats this process for those trees. As a result, those trees get connected by the minimum-weighted edge. Repeating the process each time reduces the number of trees. After logarithmically many repetitions of the process, there will be no more trees to be connected with each other. Then, the process terminates. This is how Boruvka's algorithm makes MST from an input graph like a connected, weighted and un-directed graph.

Time complexity of Boruvka's algorithm:
Boruvka's algorithm takes O (log V) time for the iterations of the outer loop until it terminates. Therefore, it takes total O (E log V) time, where E is the number of edges, and V is the number of vertices in G (assuming E ≥ V).

Differences with the other MST algorithms are,

1. Kruskal algorithm first sorts all edges from the cheapest to the most expensive ones. Then it starts to take the cheapest edges which do not form

a cycle. After taking V-1 edges, it forms an MST. But Boruvka’s algorithm starts from a vertex and takes the cheapest edge for that vertex. In this way, it takes the cheapest edge for each vertex or trees to another vertex or trees. Finding the cheapest outgoing edge from each vertex or tree can be done easily in parallel whereas in Kruskal algorithm, we check edges in strict sequential order which cannot keep the parallelism. So, the Kruskal algorithm can’t work in parallel like Boruvka’s algorithm to form an MST.

2. Prim's algorithm chooses a root vertex and repeatedly takes the cheapest edge incident to root until only root remains. Boruvka’s algorithm repeatedly takes all of the cheapest incident edges in parallel until there is exactly one vertex remaining. Fast parallel algorithms can be obtained by combining Prim's algorithm with Boruvka's algorithm.

3. The Reverse-Delete algorithm starts with the original graph and deletes edges from it and Boruvka’s algorithm starts with the empty graph and forms an MST by adding the edges. The Reverse-Delete algorithm deletes the most expensive edges from the input graph by checking whether the edge is not disconnecting the graph. For the input graph, Boruvka’s algorithm adds the cheapest edges of each vertex or tree to another vertex or tree in parallel until all the vertex or trees are connected.

Reverse Delete Algorithm-

Reverse Delete Algorithm is a greedy approach to solving the minimum spanning tree (MST) problems. This is similar to Kruskal’s Algorithm but it is in the complete opposite form, meaning that it starts with a graph and ends with a MST.

How does it work? Just like Kruskal’s algorithm, this algorithm also sorts the weighted edges but in descending order. Then it deletes edges from the top of the descending order of weighted edges and each time checks if deleting an edge disconnects the graph. If the graph gets disconnected then it re-connects the edge and moves to the next edge to do the same. It will keep on doing this process till there are (V-1) edges in the graph for which a MST is formed. So that's how the Reverse Delete Algorithm works.

Time complexity of this algorithm is O(E log V (log log V)^3). How?

- Sorting the edges by weight using merge-sort takes O(E log E) time. Which can be simplified to O(E log V) because the largest E can be V^2.
- The loop will iterate E times where E = edge
- Deleting an edge then checking the connectivity of the resulting graph and if the graph gets disconnected then reinstalling that edge which disconnects the graph can be done in O(log V (log log V)^3)
- To total of O(E log V (log log V)^3) time.

Differences with the other MST algorithms are,

1. As we can see if we compare it to Kruskal’s Algorithm, then we can see that this algorithm is the complete opposite of Kruskal’s. In Kruskal’s we sort the weighted edges in an ascending order but here we sort the edges in a descending order. In Kruskal’s we take the smallest weighted edges from the sorted edge list and connect them with each other while checking that it doesn’t form a cycle till the edge count is (V-1) and form a MST. But for this algorithm we delete the edges from the sorted edge list and keep deleting till we have (V-1) edges while checking if deleting the edge doesn't disconnect the graph. In simple works Kruskal’s algorithm starts with an empty graph and adds edges to get the MST while the Reverse Delete algorithm starts with the original graph and deletes edges from it to get the MST.
2. If we compare it to Prim's algorithm we can see that, since Prims is similar to Kruskal’s Algorithm, Reverse Delete algorithm has almost the same differences with Prim’s Algorithm as Kruskal’s Algorithm. The slight difference is that Prim’s algorithm connects edges with the smallest weights which are adjacent to the currently connected nodes which are in the MST. While the Reverse Delete algorithm deletes edges with the highest weight till it forms MST.
3. In the case of Boruvka’s and Reverse Delete algorithm, we can see that the Reverse Delete algorithm takes a whole graph and deletes the max weighted edges and forms a tree. But Boruvka’s algorithm starts with an empty graph and finds all the minimum weighted edges of each vertex of the graph and adds those vertices with its minimum edge to other vertices in parallel and forms a tree.

Time Complexity Comparison-

Table2: Time Complexity Comparison

Algorithms	Time Complexity
Prim’s Algorithm (Varies depending on Minimum edge weight Data-structure)	1. O(V^2) 2. O(E log V) 3. O(E + V log V)
Kruskal’s Algorithm	O(E log V)
Boruka’s Algorithm	O (E log V)
Reverse Delete Algorithm	O(E log V (log log V)^3)