CS2023 - Inclass Lab

Week 11 - MST

Note: You are required to answer the below questions and submit a PDF to the submission link provided under this week before the deadline (no extensions will be provided). You can either write / type your answers, but either way your answers should be readable.

Add the link to the GitHub repository

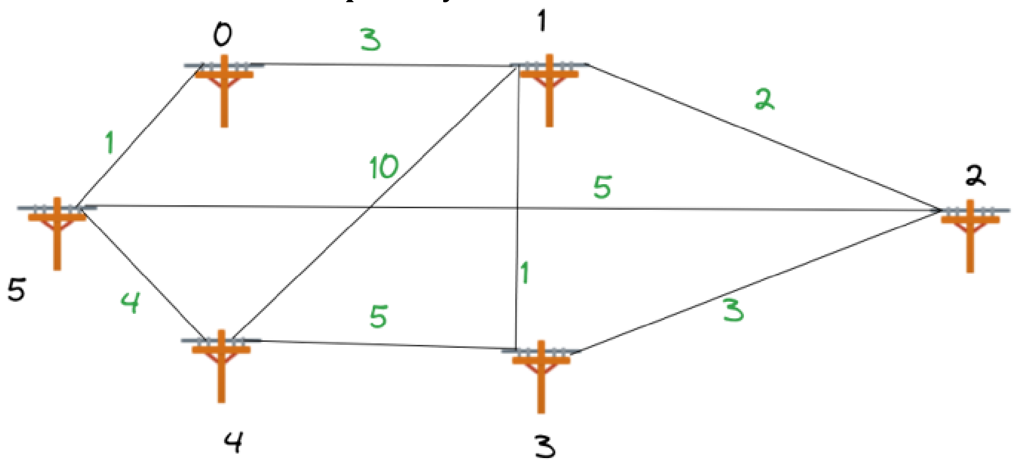


Figure 1: Telephone Pole Graph

Lab instruction

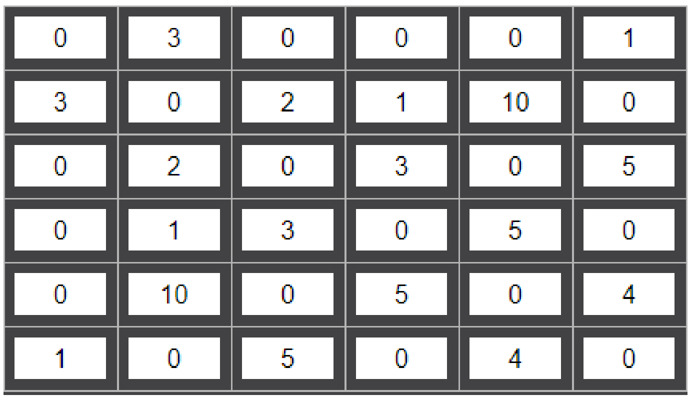
The problem is finding the minimum wire cost required by a telecommunication company to connect 6 streets. The linesmen have given you the possible wiring they can perform and the distance between each pole when the wiring is done. As an engineer, you must give them the best possible wiring connections between poles so that the wiring cost is minimal to your company. Expected submission

1. Write the adajaceny matrix for the graph on Fig 1.

2. Calculate and draw the minimum spanning tree for the graph in Fig1, taking **Node 3** as the start node.
3. Implement Prim's MST algorithm and obtain the minimum spanning tree taking **Node 0** as the start node. Take screen shot of your output.
4. Are the MSTs in Question 2 and Question 3 the same? What is the condition for a graph to have only 1 minimum spanning tree.
5. Discuss the time complexity between Prim's Algorithm and Kruskal's Algorithm.

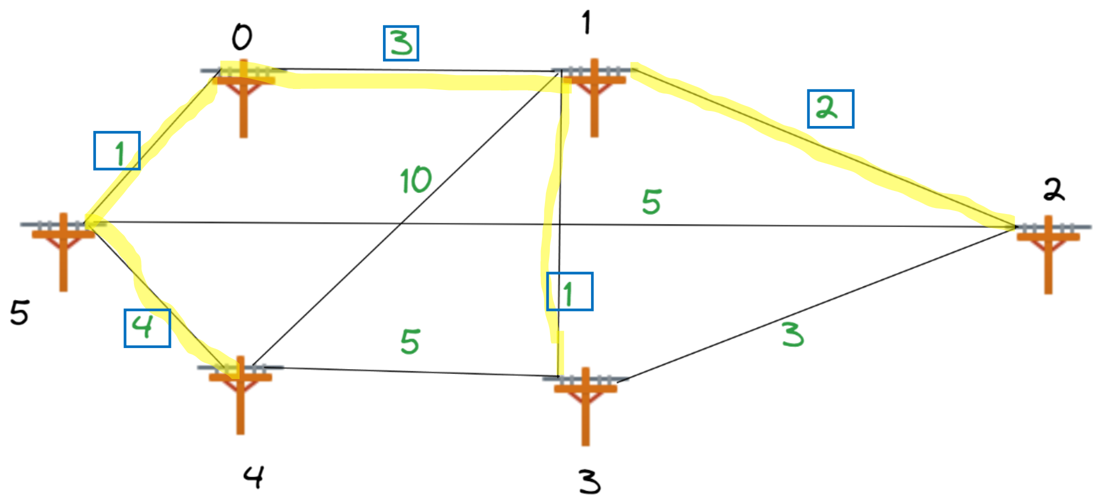
Answer

1.



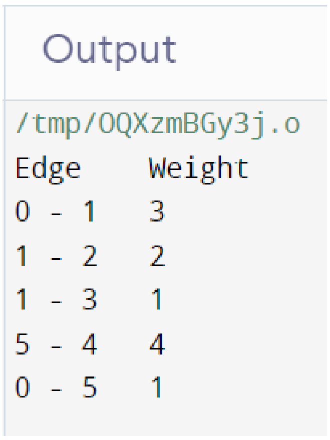
0	3	0	0	0	1
3	0	2	1	10	0
0	2	0	3	0	5
0	1	3	0	5	0
0	10	0	5	0	4
1	0	5	0	4	0

2.



Sum of the minimum spanning tree = 1 + 1 + 2 + 3 + 4 = 11

3.



```
Output
/tmp/OQXzmBGy3j.o
Edge     Weight
0 - 1    3
1 - 2    2
1 - 3    1
5 - 4    4
0 - 5    1
```

4. Yes. If all the weights are distinct then graph has only one MST.

5.

Prim's Algorithm: O((V + E) log V)
Kruskal's Algorithm: O(E log E)

Prim's Algorithm grows the minimum spanning tree one vertex at a time, starting with an arbitrary vertex and repeatedly adding the cheapest edge that connects a vertex in the tree to a vertex outside the tree. Time Complexity: The time complexity of Prim's Algorithm depends on the data structure used to implement the priority queue. When implemented with a binary heap, the time complexity is O((V + E) log V), where V is the number of vertices and E is the number of edges in the graph. The algorithm's time complexity is dominated by the operations of extracting the minimum cost edge and updating the key values in the priority queue.

Kruskal's Algorithm, on the other hand, grows the minimum spanning tree by adding edges in ascending order of their weights and ensuring that adding an edge does not create a cycle in the tree.Time Complexity: The time complexity of Kruskal's Algorithm is dominated by the sorting of the edges. Sorting the edges takes O(E log E) time, where E is the number of edges in the graph. After sorting, the algorithm iterates over the sorted edges and performs disjoint set operations to determine if adding an edge creates a cycle, which takes O(E α(V)) time, where α is the inverse Ackermann function.In practice, the inverse Ackermann function is very slow-growing and can be considered as a constant factor. Therefore, the overall time complexity of Kruskal's Algorithm is O(E log E).