Assignment 3
Prim and Kruskal

Student name: Dastan Bekesh

Group: SE-2433

Practice Teacher: Aidana Aidynkyzy

Introduction

I used Kruskal and Prim algorithms in this assignment to find minimum spanning tree to optimize of a city transportation network.

What are the Kruskal and Prim algorithms?

Both of them are being used to find minimum spanning tree.

Prim - grow-from-vertex approach
It starts from any vertices and will choose shortest path and then will create MST

It uses adjacency list and Minimum PriorityQueue;

In my implementation Time Complexity is O(E logV)

Because I used Min PQ

Time Complexity also may be $O(V^2)$, if we use common array;

Or, O(E + V log V) if we use Fibonacci heap.

Kruskal – greedy adge-based algorithm.

It starts from sorting by weight (ascending).

Then, it will choose the smallest edge. If the cycle is not formed, it will include it, if not, it will discard it.

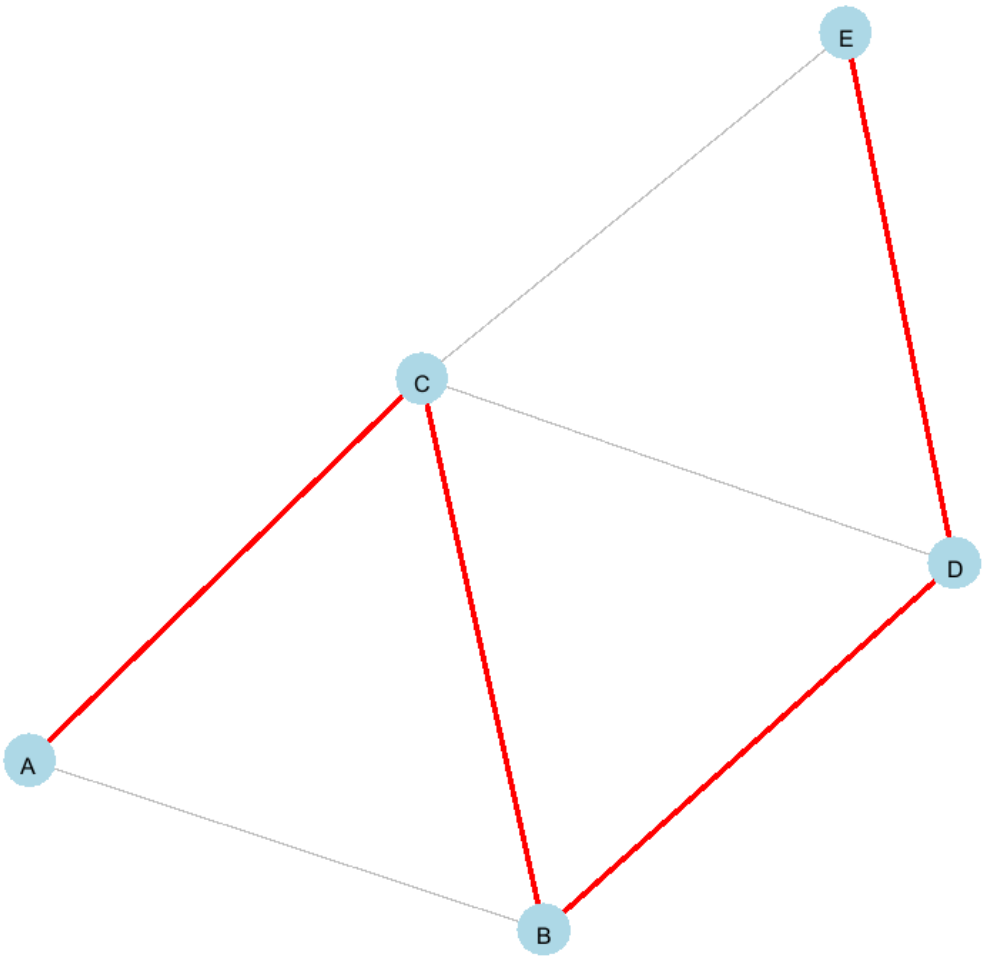
It uses DSU to prevent cycles and to connect graph.

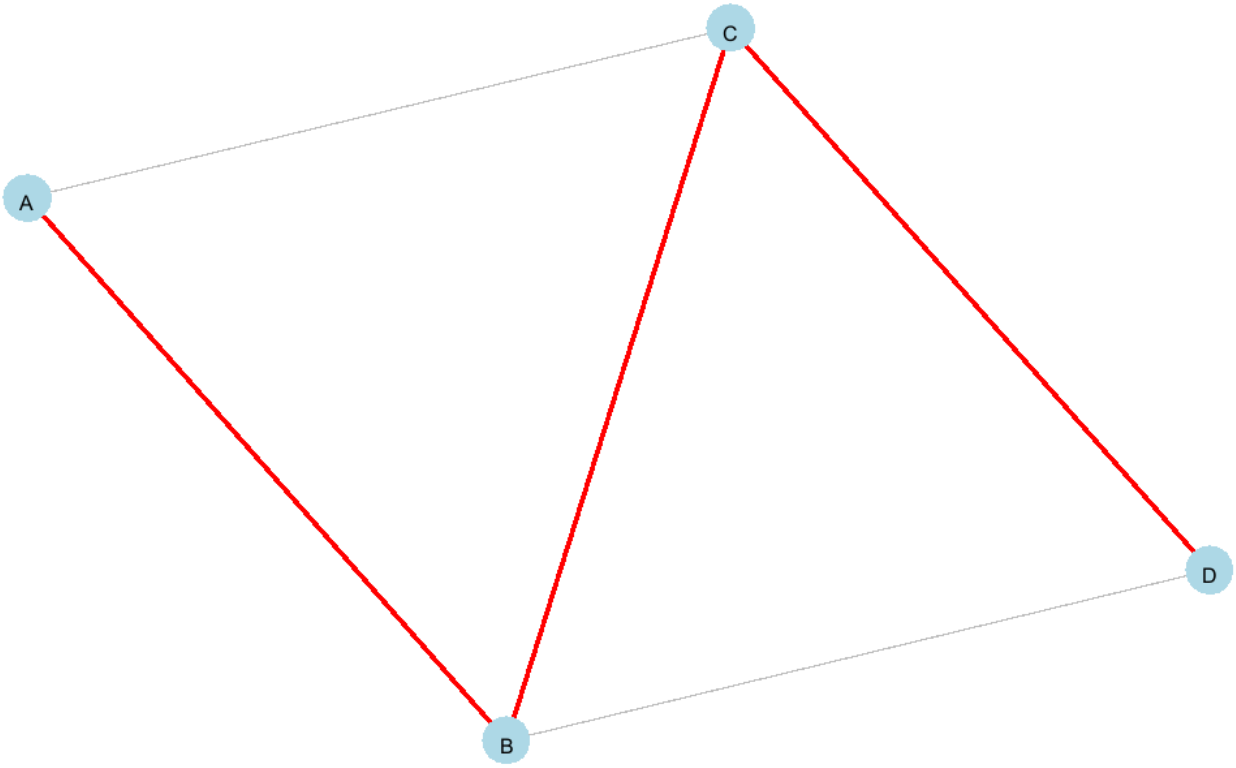
Time complexity:

Sorting: O(E log E)

DSU ops: $O(E\ \alpha(V)) \approx O(E)$

I've made the representations of Minimum Spaning Tree:


E
C
D
A
B


C
A
D
B

Let's study this algorithm more deeply:

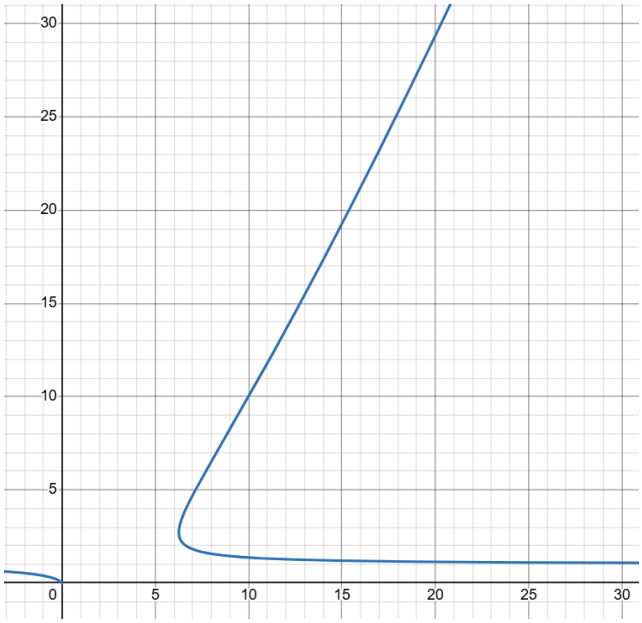
Prim

In theory the time complexity of my algorithm is O(E logV);

Practical metrics

NODES	EDGES	Kruskal (Timems)	Kruskal(Comparisons)	Kruskal(Finds)	Kruskal(Unions)	Prim(Timems)	Prim(Comparisons)	Prim(Finds)	Prim(Unions)	Weight
4	3	0.0175	20	12	3	0.0171	5	0	0	6
11	10	0.11	60	42	10	0.49	12	0	0	28
24	23	0.12	191	100	23	0.09	28	0	0	103

O(ElogV)



In this we can see that it’s almost always increasing

However, in our practical results, we can see that increasing the number of nodes does not always lead to a proportional increase in execution time. This happens because some graphs remain sparse even with more vertices, which means the algorithms still process a relatively small number of edges.

The execution time may also vary due to CPU caching effects, memory access patterns, and background processes managed by the operating system. These low-level factors can introduce noticeable timing fluctuations, especially when the algorithms run very quickly.

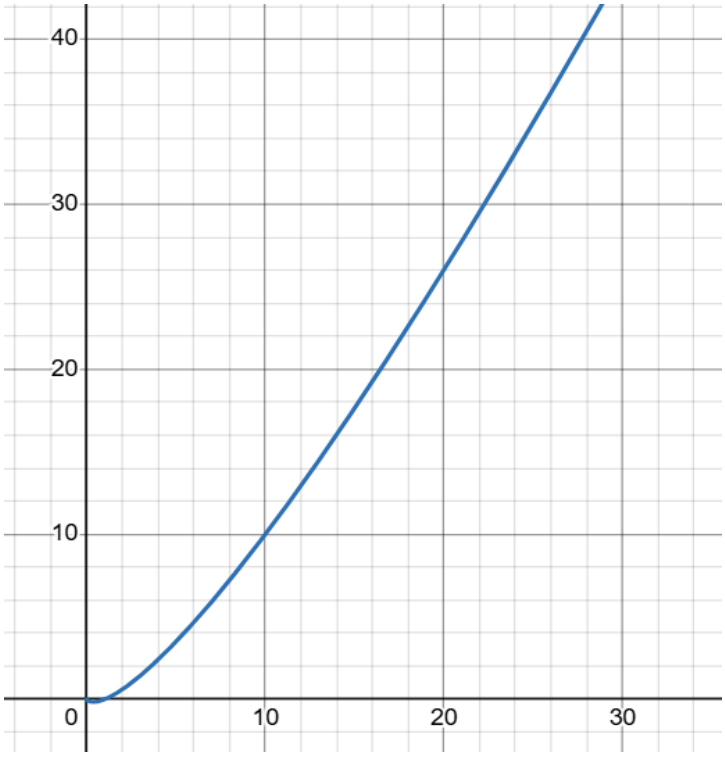
There are no Finds or Unions, because this algorithm does not use Disjoint Set Union in main algorithm. It uses it only in checking.

Kruskal

In theory the time complexity of my algorithm is O(E log E);

NODES	EDGES	Kruskal (Timems)	Kruskal(Comparisons)	Kruskal(Finds)	Kruskal(Unions)	Prim(Timems)	Prim(Comparisons)	Prim(Finds)	Prim(Unions)	Weight
4	3	0.0175	20	12	3	0.0171	5	0	0	6
11	10	0.11	60	42	10	0.49	12	0	0	28
24	23	0.12	191	100	23	0.09	28	0	0	103

O(E log E):



In this we can see that it's always increasing

However, it's not always in practice. Because there are lot things than influence to speed.

This algorithm uses DSU, so there are unions and finds.

We can see tendency that number of unions is equal to number of edges.

All comparisons are being done in sorting.

KRUSKAL VS PRIM

1.Graph Focus

Kruskal uses Edge-based greedy selection, While

Prim uses Vertex-based expansion

2. Data Structure

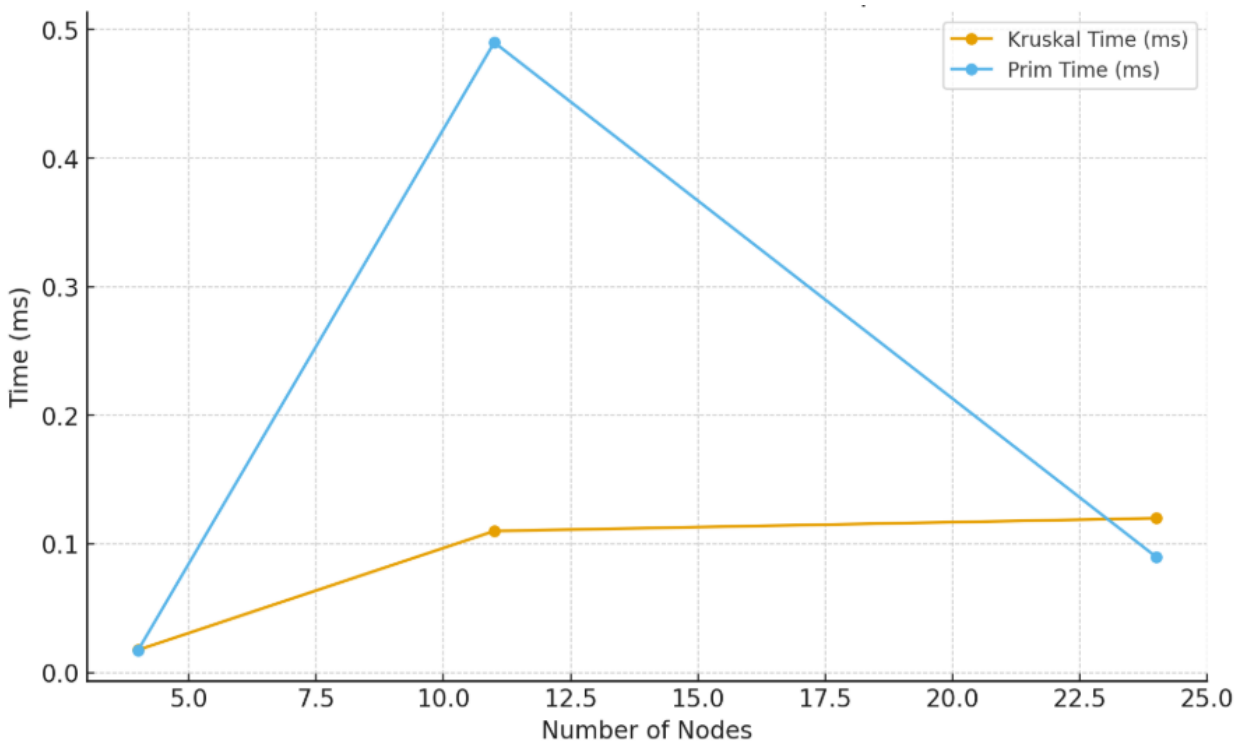
Kruskal uses Sorting + DST

Prim uses Priority Queue + Adjacency Lists

3. Efficiency

Time complexity is almost the same.

Prim is generally more efficient on large and dense graphs, while Kruskal tends to perform better on small or sparse graphs.



In this this graph we can comparison of Prim and Kruskal in practice.

Even though Prim's execution time spikes at N = 11, this deviation is not critical. It may be influenced by factors such as caching, memory allocation, or other system-level effects rather than the algorithm itself.

At the beginning, Kruskal performed slightly faster, but as the graph size increased, it became slower compared to Prim.

Conclusion

Both Prim and Kruskal algorithms successfully construct a Minimum Spanning Tree, but their efficiency depends heavily on graph properties and implementation details.

Kruskal algorithm generally performs better on sparse graphs, where the number of edges is close to the minimum required. The sorting stage dominates the complexity, and with fewer edges it remains efficient. It is also easier to implement when edges are already provided as a list.

Prim's algorithm becomes preferable as graphs grow larger and denser. With a good priority queue implementation, Prim achieves a complexity close to O(E log V), which scales better when the number of edges increases significantly. Prim is also memory-friendly when used with adjacency lists in large graphs.

In practice, both algorithms may show small performance variations caused by hardware effects (CPU caching, memory access, allocation patterns), but the general performance tendencies remain consistent.

REFERENCIES:

GeeksforGeeks. (n.d.). *Kruskal's Minimum Spanning Tree (MST) Algorithm*. Retrieved from https://www.geeksforgeeks.org/dsa/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/

GeeksforGeeks. (n.d.). *Prim's Minimum Spanning Tree (MST) | Greedy Algorithm*. Retrieved from https://www.geeksforgeeks.org/dsa/prims-minimum-spanning-tree-mst-greedy-algo-5/

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.