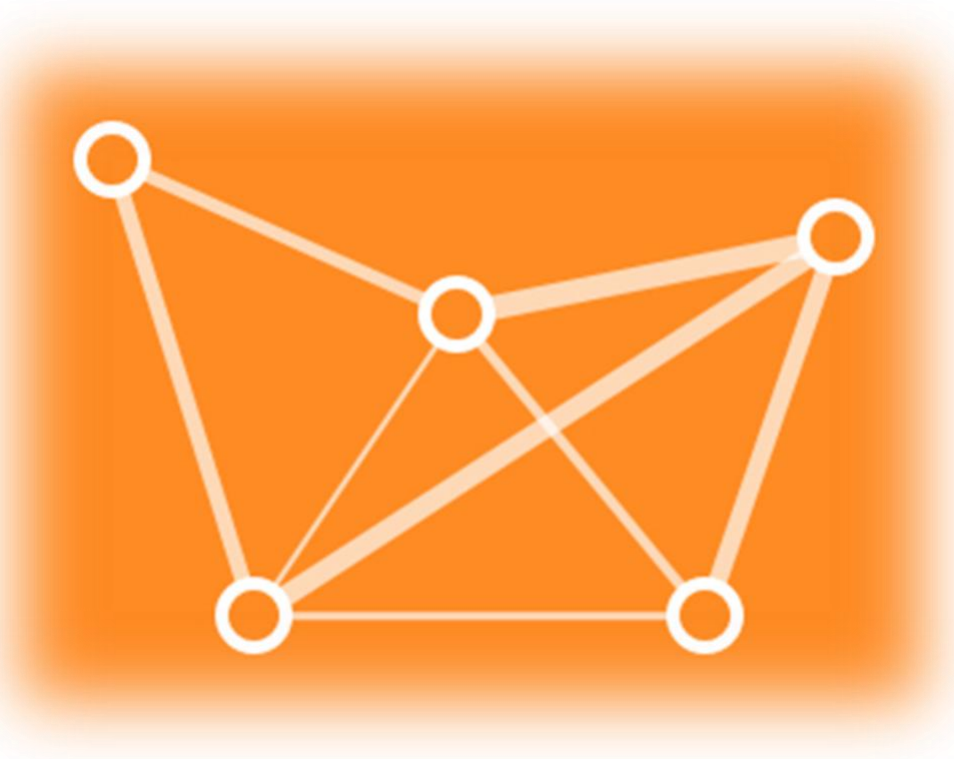
King Abdulaziz University, Jeddah, Saudi Arabia.
Faculty of Computing and Information Technology
Department of Computer Science
CPCS324-Algorithms and Data Structure (II)
Group Project – Part I

Minimum Spanning Tree Using Kruskal's and Priority queue Based Prim Algorithms



Student Name	ID	Section
[redacted]	[redacted]	B0A
Amani Biraik	[redacted]	B8
[redacted]	[redacted]	B9B
[redacted]	[redacted]	B9A

Table of Contents

1. Illustrations

2. Introduction

The minimum spanning tree is the tree whose sum of the weighted edges has the lowest cost among other spanning trees. The algorithms we will use in this report to find the minimum spanning tree are Kruskal's and Prim's algorithms. In Kruskal's algorithm, we add the edges of the graph in ascending order of weight and skip the vertices that may form a cycle, while in Prim's algorithm, we expand the sub-trees in a greedy matter and stops when all the graph's vertices become included. In the first phase of this project, we will compute the minimum spanning tree with two functions ReadGraph and MakeGraph, and print it using both Kruskal's and priority queue based prim algorithms and then perform an extensive set of experiments to test, plot, and analyze the efficiencies of each algorithm and compare them.

3. Tools and Decisions

- NetBeans
- Java language
- Microsoft Excel
- GitHub

We used these tools to complete the project's requirements. Java programming was done using NetBeans. The running times were analyzed in Microsoft Excel, and our code was uploaded to GitHub.

4. Run Algorithms

4.1. Analysis

In the following tables, we display the empirical analysis of each algorithm utilizing the physical run time measurement for a particular set of input sizes (n, m). Each measure resulted in an output of distinct data recordings. For more precision, we calculated the runtime for each input ten times and found its best, average, and worst runtimes.

Input Size												total	Best	Average	worst
n	m	trail 1	trail 2	trail 3	trail 4	trail 5	trail 6	trail 7	trail 8	trail 9	trail 10				
1,000	10,000	55	54	53	50	47	48	53	52	55	52	519	47	51.9	55
	15,000	58	57	59	64	69	56	53	69	76	58	619	53	61.9	76
	25,000	87	85	86	99	86	82	92	86	93	100	896	82	90	100
5,000	15,000	110	105	107	103	109	117	107	114	101	109	1,082	101	108	117
	25,000	154	166	159	165	149	157	150	171	156	147	1,574	147	157	171
10,000	15,000	235	225	211	219	212	224	224	219	211	216	2,196	211	220	235
	25,000	265	263	272	256	265	272	276	267	265	261	2,662	256	266	276
total		964	955	947	956	937	956	955	978	957	943	9,548	897	955	1,030

Table 1 Kruskals Algorithm

Input Size												total	Best	Average	worst
n	m	trail 1	trail 2	trail 3	trail 4	trail 5	trail 6	trail 7	trail 8	trail 9	trail 10				
1,000	10,000	14	12	20	13	12	15	13	13	25	19	156	12	15.6	25
	15,000	17	19	17	14	18	19	19	19	22	19	183	14	18.3	22
	25,000	19	16	36	35	27	18	26	23	19	15	234	15	23	36
5,000	15,000	23	19	21	18	27	24	19	19	28	23	221	18	22	28
	25,000	37	31	34	24	34	25	24	41	25	29	304	24	30	41
10,000	15,000	36	32	25	22	19	30	27	29	20	22	262	19	26	36
	25,000	39	39	30	31	46	32	47	39	40	33	376	30	38	47
total		185	168	183	157	183	163	175	183	179	160	1,736	132	174	235

Table 2 Prims Algorithm

To further compare and evaluate the effectiveness of each algorithm, we used the data from the tables to build a graph representing the average case. The x-axis represents the vertices and edges, and the y-axis represents the running time.

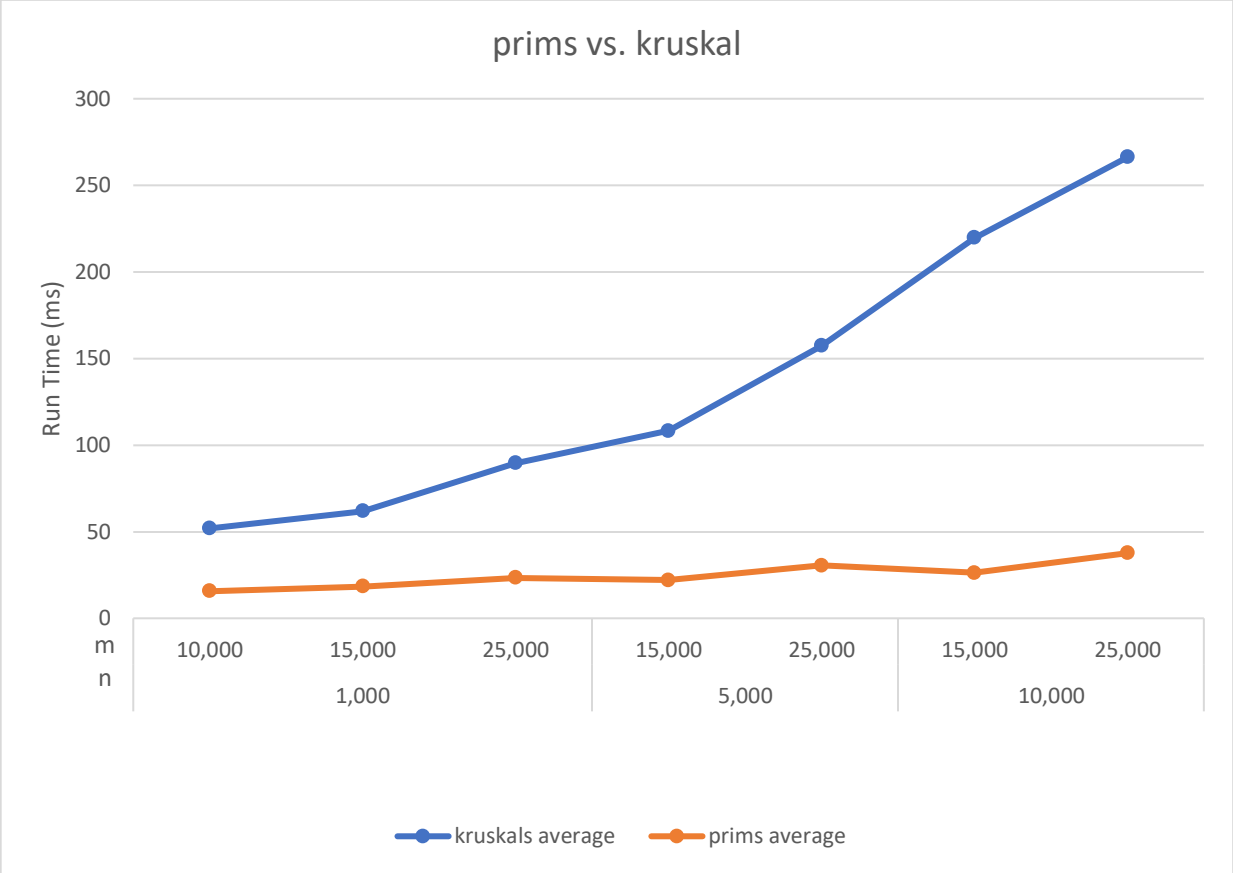


Figure 1 Prim vs. Kruskal

According to the displayed graph, when given the same input, priority queue-based Prim's algorithm computes the minimum spanning tree more quickly and effectively than Kruskal's algorithm. Additionally, theoretical efficiencies show that Prims has an O (E log V) efficiency while Kruskal's has an O (E log E) efficiency.

4.2. Sample Screenshots

4.2.1. Read Graph from File

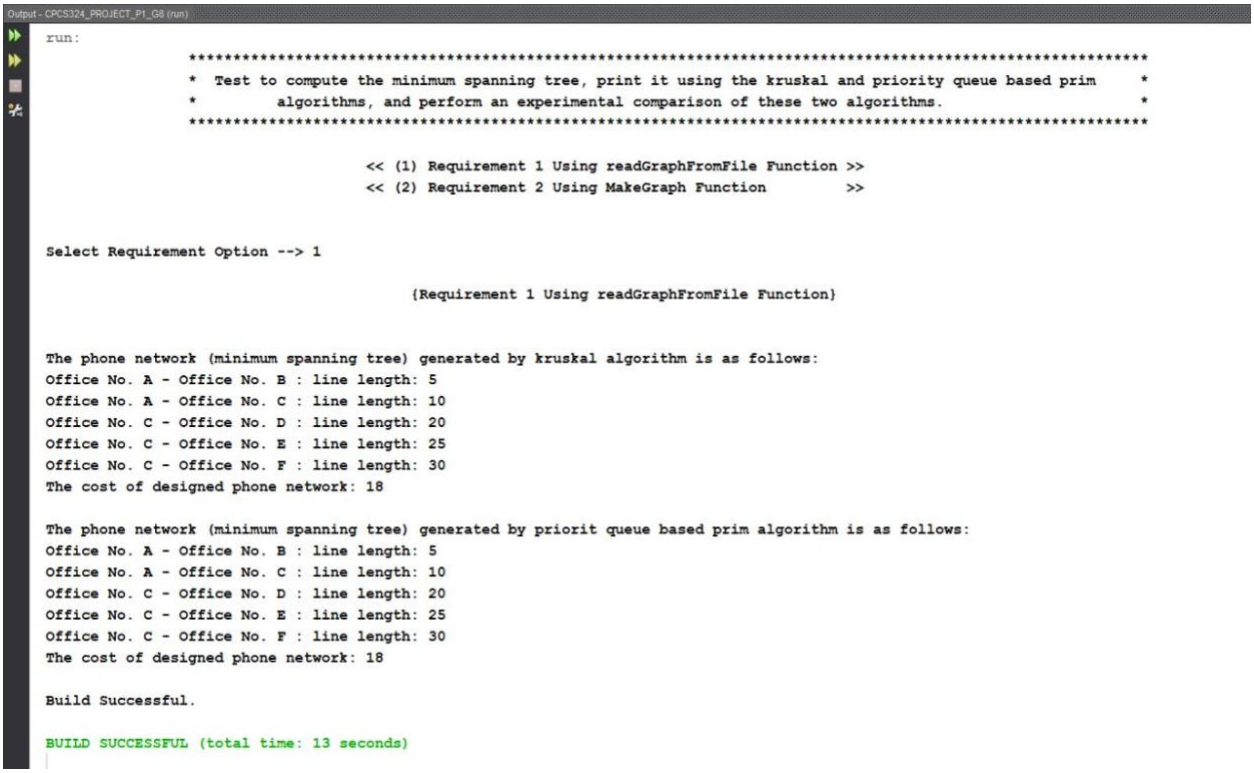


Figure 2 Read Graph from file screenshot.

4.2.2. Make Graph

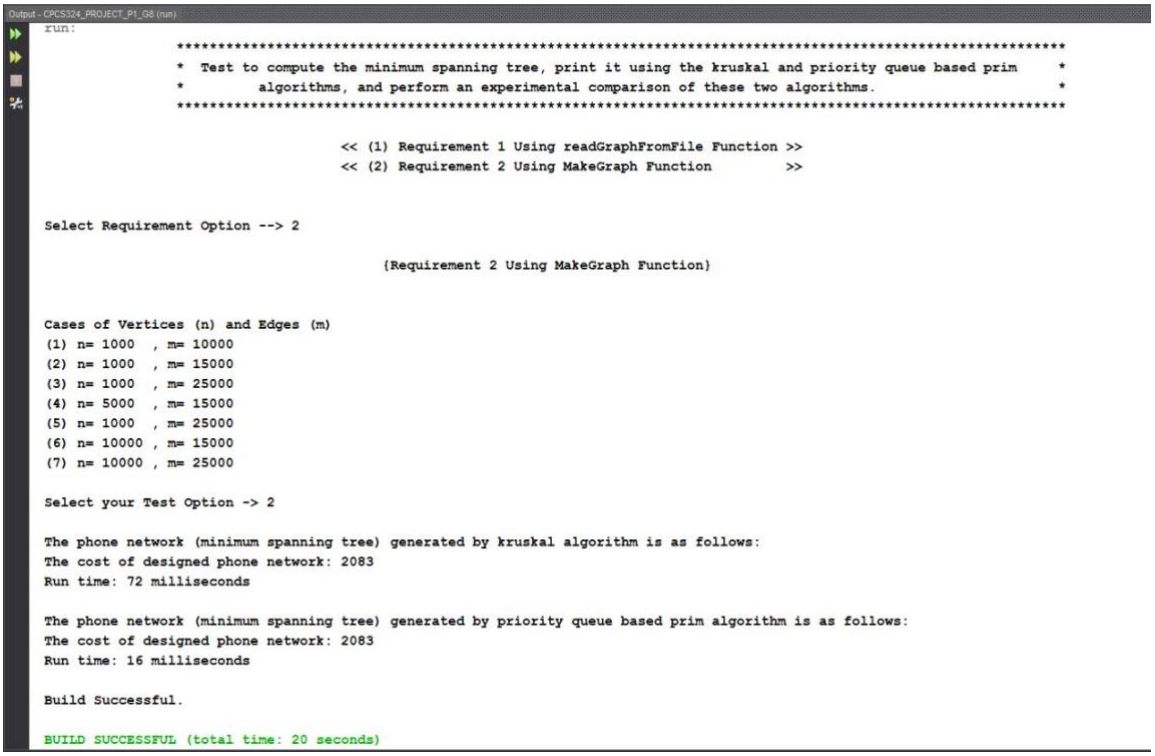


Figure 3 Make Graph case2 screenshot.

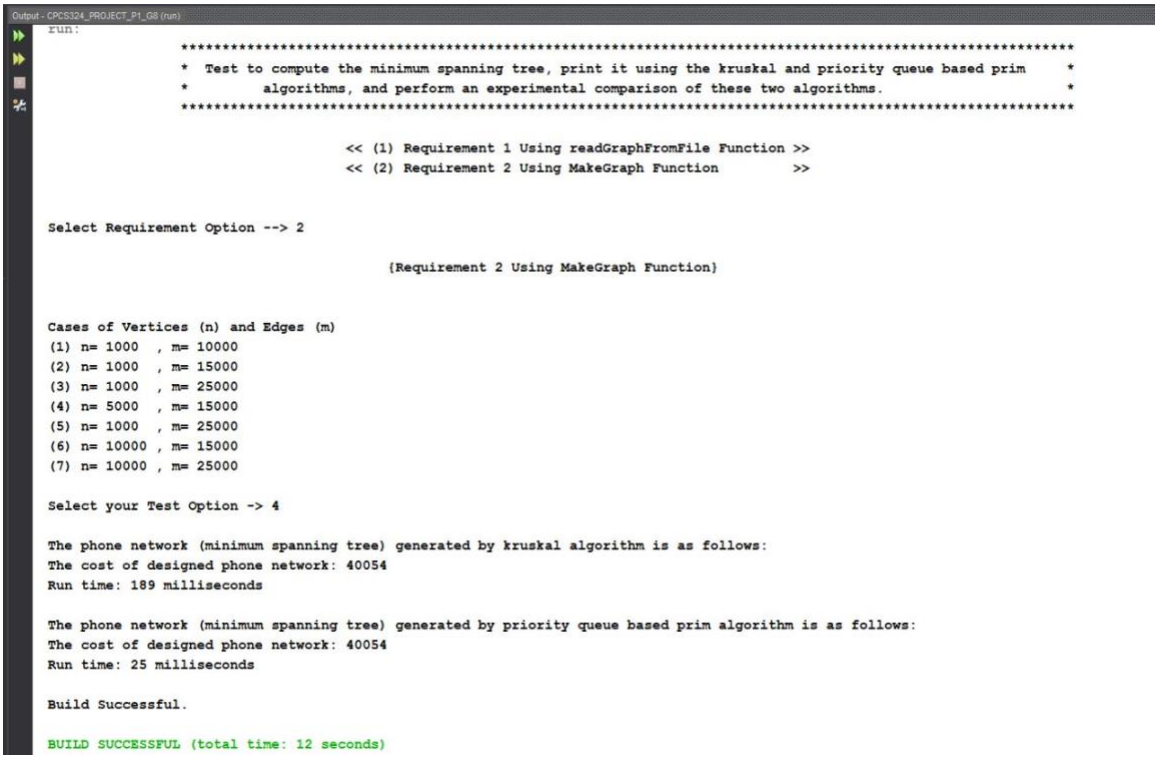


Figure 4 Make Graph case4 screenshot.

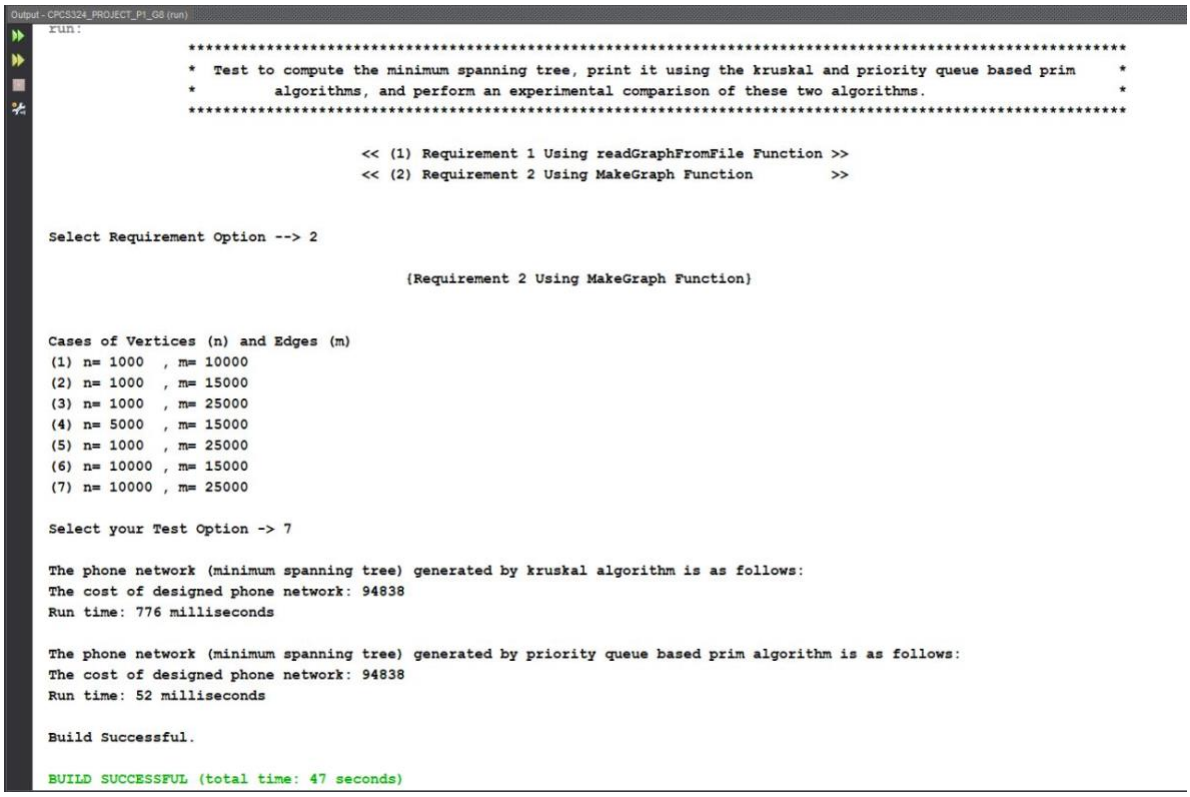


Figure 5 Make Graph case7 screenshot.

5. Difficulties Faced During the Phase Design

The first problem we faced was when we used the Make graph function. When the input was small, everything worked great, but when a more significant input was used, the algorithm did not function correctly and printed a different cost for each algorithm given the same input. Secondly, we had some difficulties in MakeGraph that we needed to randomly connect each vertex to an edge and ensure that the resulting graph was connected and that the edge connecting two specific vertices was not repeated twice. Finally, we initially tried to implement the algorithm's codes without using any source and only depending on our understanding of how the algorithm functions. It worked okay with smaller inputs, but when more significant inputs were used, running the algorithm took some time and sometimes gave an error "outOfMemoryError," indicating that the efficiencies of our algorithms were not the best. Therefore, we used resources from the internet to help improve the efficiency and running time.

6. Conclusion

In conclusion, after implementing both priority queue Prims and Kruskal's algorithms to find the minimum spanning tree, we were able to calculate the efficiencies of each algorithm using their runtime for each case and compare them. We concluded that Prim's algorithm is more efficient than Kruskal's.

7. References

- GeeksforGeeks. (2023). Kruskal s Minimum Spanning Tree MST Algorithm. *GeeksforGeeks*. https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/
- *Prim's algorithm Java - Javatpoint*. (n.d.). www.javatpoint.com. https://www.javatpoint.com/prims-algorithm-java
- Srivastava, S., & Srivastava, S. (2023). Checking if a Java Graph Has a Cycle | Baeldung. *Baeldung*. https://www.baeldung.com/java-graph-has-a-cycle
- Levitin, A. (2023). *Introduction to the Design and Analysis of Algorithms 3th (third) edition*.

8. Appendix

GitHub Link:

https://github.com/Amani-Biraik/CPCS324PROJECT#cpcs324project