

**FACULTY OF COMPUTING  
& INFORMATION TECHNOLOGY**  
KING ABDULAZIZ UNIVERSITY



**كلية الحاسبات  
وتقنية المعلومات**  
جامعة الملك عبدالعزيز

**King Abdulaziz University**  
**Department of Computer Science**  
**Faculty of Computing and Information Technology**  
**CPCS 324**  
**Algorithms & Data Structure 2**

<b>Student Name</b>	<b>Student ID</b>
Saudh Khaled Almahary	2206655
Nawal Hussein Al-Ghamdi	2206918
Razan Serag Alharthi	2205814

## Contents

Introduction.....	3
Problem Description .....	3
Algorithm Used.....	3
Requirement 1 .....	4
Breakdown of selected paths .....	5
Requirement 2 .....	8
Difficulties Faced During the Phase Design .....	9
Conclusion .....	10
Appendix.....	10

## Introduction

In this project, we implemented Dijkstra's algorithm to find the shortest paths between different locations in a weighted graph that represents an air freight transportation system. The system reads the graph data from a text file, then calculates and displays the shortest routes from each location to all other locations. The main idea is to simulate how logistics systems find the most efficient paths with the least cost or distance. This project was done as part of the CPCS 324 – Algorithms course, and it aims to show how classical graph algorithms can be applied to real-world problems like route optimization, planning cargo transportation, and reducing overall network costs.

## Problem Description

Air freight is one of the most efficient and time-sensitive methods of transporting goods across different locations. However, due to the complexity of air route networks and varying distances between locations, determining the optimal paths for delivery is essential. The main problem addressed in this project is to find the shortest possible routes between all pairs of locations in order to reduce total travel distance, minimize fuel consumption, and improve delivery time. To solve this, the system represents the air freight network as a weighted directed or undirected graph, where nodes represent locations (airports or cargo hubs) and edges represent direct connections with weights representing travel cost or distance. The goal is to apply Dijkstra's algorithm to this graph and compute the shortest paths from every node to all others, enabling the air freight system to operate more efficiently and cost-effectively.

## Algorithm Used

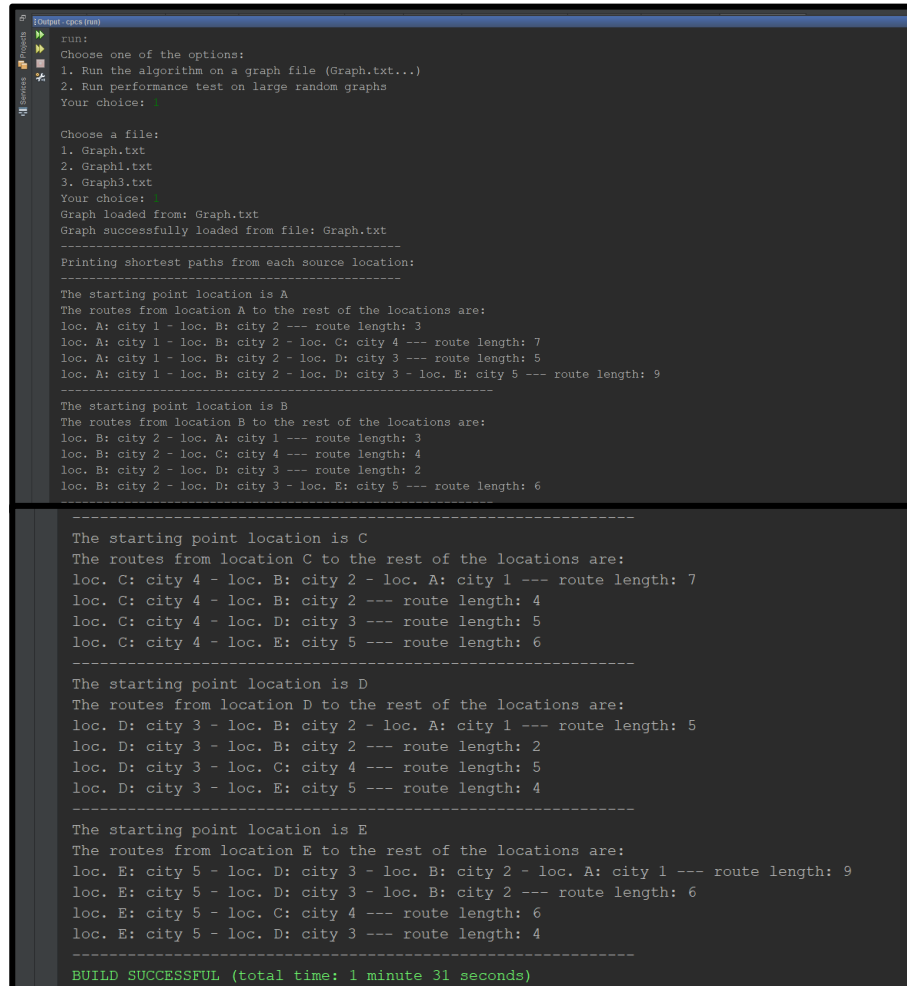
Dijkstra's algorithm is a well-known greedy algorithm used to find the shortest paths from a single source node to all other nodes in a weighted graph with non-negative edge weights. In this project, the algorithm was applied repeatedly from each node in the graph to compute all-pairs shortest paths. This approach ensures that the shortest route between every two locations is identified. The algorithm works by initializing the distance to all nodes as infinity, except for the source node which is set to zero. It then uses a priority queue to always process the node with the smallest known distance, and updates the distances of its neighbors if a shorter path is found. This process continues until all nodes have been visited and the shortest distances from the source have been determined.

In the implementation, the graph was built using custom Location and Route classes that extend the Vertex and Edge base classes, respectively. The system uses a combination of a hash map for distance tracking and a priority queue for efficient selection of the next node to process.

## Requirement 1

1.1 Implement the Dijkstra algorithm to compute the length of the shortest paths from all locations to the rest of the locations. Make sure to print the shortest paths with their lengths.

- Store the above graph within a file and use readGraphFromFile() method to store it in memory



```
run:
Choose one of the options:
1. Run the algorithm on a graph file (Graph.txt...)
2. Run performance test on large random graphs
Your choice:
Choose a file:
1. Graph.txt
2. Graph1.txt
3. Graph3.txt
Your choice:
Graph loaded from: Graph.txt
Graph successfully loaded from file: Graph.txt
-----
Printing shortest paths from each source location:
-----
The starting point location is A
The routes from location A to the rest of the locations are:
loc. A: city 1 - loc. B: city 2 --- route length: 3
loc. A: city 1 - loc. B: city 2 - loc. C: city 4 --- route length: 7
loc. A: city 1 - loc. B: city 2 - loc. D: city 3 --- route length: 5
loc. A: city 1 - loc. B: city 2 - loc. D: city 3 - loc. E: city 5 --- route length: 9
-----
The starting point location is B
The routes from location B to the rest of the locations are:
loc. B: city 2 - loc. A: city 1 --- route length: 3
loc. B: city 2 - loc. C: city 4 --- route length: 4
loc. B: city 2 - loc. D: city 3 --- route length: 2
loc. B: city 2 - loc. D: city 3 - loc. E: city 5 --- route length: 6
-----
The starting point location is C
The routes from location C to the rest of the locations are:
loc. C: city 4 - loc. B: city 2 - loc. A: city 1 --- route length: 7
loc. C: city 4 - loc. B: city 2 --- route length: 4
loc. C: city 4 - loc. D: city 3 --- route length: 5
loc. C: city 4 - loc. E: city 5 --- route length: 6
-----
The starting point location is D
The routes from location D to the rest of the locations are:
loc. D: city 3 - loc. B: city 2 - loc. A: city 1 --- route length: 5
loc. D: city 3 - loc. B: city 2 --- route length: 2
loc. D: city 3 - loc. C: city 4 --- route length: 5
loc. D: city 3 - loc. E: city 5 --- route length: 4
-----
The starting point location is E
The routes from location E to the rest of the locations are:
loc. E: city 5 - loc. D: city 3 - loc. B: city 2 - loc. A: city 1 --- route length: 9
loc. E: city 5 - loc. D: city 3 - loc. B: city 2 --- route length: 6
loc. E: city 5 - loc. C: city 4 --- route length: 6
loc. E: city 5 - loc. D: city 3 --- route length: 4
-----
BUILD SUCCESSFUL (total time: 1 minute 31 seconds)
```

This figure displays the output of Dijkstra's algorithm when executed from every node in the graph. The graph was loaded from the file Graph.txt, and the system computes and prints the shortest routes from each location (A, B, C, D, E) to all other locations. Each line in the output represents a full path between two locations, showing the sequence of nodes visited and the total route length.

## Breakdown of selected paths

- **From A to B:**

A direct edge with weight 3 is used.

→ loc. A: city 1 – loc. B: city 2 --- route length: 3

- **From A to C:**

The path goes through B ( $A \rightarrow B \rightarrow C$ ) with a total cost of  $3 + 4 = 7$ .

→ loc. A: city 1 – loc. B: city 2 – loc. C: city 4 --- route length: 7

- **From A to E:**

the shortest path is  $A \rightarrow B \rightarrow D \rightarrow E$  with a total cost of  $3 + 2 + 4 = 9$ .

→ loc. A: city 1 – loc. B: city 2 – loc. D: city 3 – loc. E: city 5 --- route length: 9

- **From C to A:**

The shortest path is  $C \rightarrow B \rightarrow A$  with a total cost of  $4 + 3 = 7$ .

- **From D to C:**

A direct connection exists with weight 5.

→ loc. D: city 3 – loc. C: city 4 --- route length: 5

This detailed output validates that the algorithm is functioning correctly and that it accurately selects the optimal route based on edge weights, whether through direct or intermediate nodes

```
Output - cpcx (run)
Run:
Choose one of the options:
1. Run the algorithm on a graph file (Graph.txt...)
2. Run performance test on large random graphs
Your choice: 1

Choose a file:
1. Graph.txt
2. Graph1.txt
3. Graph3.txt
Your choice: 2
Graph loaded from: Graph1.txt
Graph successfully loaded from file: Graph1.txt
-----
Printing shortest paths from each source location:
-----

The starting point location is A
The routes from location A to the rest of the locations are:
loc. A: city 1 - loc. B: city 2 --- route length: 1
loc. A: city 1 - loc. C: city 3 --- route length: 2
loc. A: city 1 - loc. C: city 3 - loc. D: city 4 --- route length: 6
loc. A: city 1 - loc. C: city 3 - loc. E: city 5 --- route length: 7
loc. A: city 1 - loc. C: city 3 - loc. F: city 6 --- route length: 8
-----

The starting point location is B
The routes from location B to the rest of the locations are:
loc. B: city 2 - loc. A: city 1 --- route length: 1
loc. B: city 2 - loc. C: city 3 --- route length: 3
loc. B: city 2 - loc. C: city 3 - loc. D: city 4 --- route length: 7
loc. B: city 2 - loc. C: city 3 - loc. E: city 5 --- route length: 8
loc. B: city 2 - loc. C: city 3 - loc. F: city 6 --- route length: 9
-----
```

```

>> The starting point location is C
> The routes from location C to the rest of the locations are:
> loc. C: city 3 - loc. A: city 1 --- route length: 2
> loc. C: city 3 - loc. B: city 2 --- route length: 3
> loc. C: city 3 - loc. D: city 4 --- route length: 4
> loc. C: city 3 - loc. E: city 5 --- route length: 5
> loc. C: city 3 - loc. F: city 6 --- route length: 6
> -----
> The starting point location is D
> The routes from location D to the rest of the locations are:
> loc. D: city 4 - loc. C: city 3 - loc. A: city 1 --- route length: 6
> loc. D: city 4 - loc. C: city 3 - loc. B: city 2 --- route length: 7
> loc. D: city 4 - loc. C: city 3 --- route length: 4
> loc. D: city 4 - loc. E: city 5 --- route length: 7
> loc. D: city 4 - loc. C: city 3 - loc. F: city 6 --- route length: 10
> -----
> The starting point location is E
> The routes from location E to the rest of the locations are:
> loc. E: city 5 - loc. C: city 3 - loc. A: city 1 --- route length: 7
> loc. E: city 5 - loc. C: city 3 - loc. B: city 2 --- route length: 8
> loc. E: city 5 - loc. C: city 3 --- route length: 5
> loc. E: city 5 - loc. D: city 4 --- route length: 7
> loc. E: city 5 - loc. F: city 6 --- route length: 8
> -----
> The starting point location is F
> The routes from location F to the rest of the locations are:
> loc. F: city 6 - loc. C: city 3 - loc. A: city 1 --- route length: 8
> loc. F: city 6 - loc. C: city 3 - loc. B: city 2 --- route length: 9
> loc. F: city 6 - loc. C: city 3 --- route length: 6
> loc. F: city 6 - loc. C: city 3 - loc. D: city 4 --- route length: 10
> loc. F: city 6 - loc. E: city 5 --- route length: 8
> -----

```

This figure shows the output of Dijkstra's algorithm applied to a larger graph loaded from the file Graph1.txt. The graph includes 6 locations labeled A through F. The algorithm is executed from each location to compute the shortest paths to all other nodes in the graph.

Each line in the output displays a complete route, including intermediate locations (if any) and the total route length. Despite the increased number of nodes, the algorithm continues to perform efficiently while maintaining accurate and well-formatted results.

For example:

- From A to F: the shortest path is  $A \rightarrow C \rightarrow F$  with a total route length of 8.
- From C to D: a direct connection exists with a weight of 4.
- From E to A: the optimal path is  $E \rightarrow C \rightarrow A$  with a total route length of 7.

This figure demonstrates the system's ability to scale with larger graphs while preserving correctness and clarity in output presentation.

```

[Output - c:\ps\run]
>> run:
Choose one of the options:
1. Run the algorithm on a graph file (Graph.txt...)
2. Run performance test on large random graphs
Your choice: 1

Choose a file:
1. Graph.txt
2. Graph1.txt
3. Graph3.txt
Your choice: 3
Graph loaded from: Graph3.txt
Graph successfully loaded from file: Graph3.txt

-----
Printing shortest paths from each source location:
-----

The starting point location is A
The routes from location A to the rest of the locations are:
loc. A: city 1 - loc. B: city 2 --- route length: 3
loc. A: city 1 - loc. B: city 2 - loc. C: city 5 --- route length: 4
loc. A: city 1 - loc. B: city 2 - loc. C: city 5 - loc. D: city 6 --- route length: 10
loc. A: city 1 - loc. E: city 4 --- route length: 6
loc. A: city 1 - loc. F: city 3 --- route length: 5
-----

The starting point location is B
The routes from location B to the rest of the locations are:
loc. B: city 2 - loc. A: city 1 --- route length: 3
loc. B: city 2 - loc. C: city 5 --- route length: 1
loc. B: city 2 - loc. C: city 5 - loc. D: city 6 --- route length: 7
loc. B: city 2 - loc. F: city 3 - loc. E: city 4 --- route length: 6
loc. B: city 2 - loc. F: city 3 --- route length: 4
-----

The starting point location is C
The routes from location C to the rest of the locations are:
loc. C: city 5 - loc. B: city 2 - loc. A: city 1 --- route length: 4
loc. C: city 5 - loc. B: city 2 --- route length: 1
loc. C: city 5 - loc. D: city 6 --- route length: 6
loc. C: city 5 - loc. F: city 3 - loc. E: city 4 --- route length: 6
loc. C: city 5 - loc. F: city 3 --- route length: 4
-----

The starting point location is D
The routes from location D to the rest of the locations are:
loc. D: city 6 - loc. F: city 3 - loc. A: city 1 --- route length: 10
loc. D: city 6 - loc. C: city 5 - loc. B: city 2 --- route length: 7
loc. D: city 6 - loc. C: city 5 --- route length: 6
loc. D: city 6 - loc. F: city 3 - loc. E: city 4 --- route length: 7
loc. D: city 6 - loc. F: city 3 --- route length: 5
-----

The starting point location is E
The routes from location E to the rest of the locations are:
loc. E: city 4 - loc. A: city 1 --- route length: 6
loc. E: city 4 - loc. F: city 3 - loc. B: city 2 --- route length: 6
loc. E: city 4 - loc. F: city 3 - loc. C: city 5 --- route length: 6
loc. E: city 4 - loc. F: city 3 - loc. D: city 6 --- route length: 7
loc. E: city 4 - loc. F: city 3 --- route length: 2
-----

The starting point location is F
The routes from location F to the rest of the locations are:
loc. F: city 3 - loc. A: city 1 --- route length: 5
loc. F: city 3 - loc. B: city 2 --- route length: 4
loc. F: city 3 - loc. C: city 5 --- route length: 4
loc. F: city 3 - loc. D: city 6 --- route length: 5
loc. F: city 3 - loc. E: city 4 --- route length: 2
-----

BUILD SUCCESSFUL (total time: 6 seconds)

```

This figure presents the results of running Dijkstra's algorithm on the graph loaded from Graph3.txt, which contains 6 nodes (A to F) with a more complex set of connections compared to previous tests. The system computes and prints the shortest paths from each location to all others, considering direct and indirect routes.

Each printed path includes the complete sequence of visited nodes and the total route length. For example:

- From A to D: the shortest path is A → C → D with a total cost of 10.
- From C to A: the path is C → B → A with a cost of 4.
- From F to E: the shortest path is F → C → E with a cost of 2.

This test validates the system's ability to process graphs with multiple routing options and confirms the correctness of path calculations across all node pairs.

## Requirement 2

2.1 Use the Dijkstra algorithm implemented in *requirement 2* to compute the length of the shortest paths for each pair of vertices and apply it on the above **five** randomly generated graphs. Compute the run duration and compare the algorithm's time efficiency computed from the experiment's results with the one determined by the formula stated in the textbook.

```

Output - cpcs (run)
Run:
Choose one of the options:
1. Run the algorithm on a graph file (Graph.txt...)
2. Run performance test on large random graphs
Your choice: 2
-----
n      |      m      |      Execution Time (ms)
-----
Graph created with 2000 vertices and 10000 edges (connected).
2000   |    10000    |    5101 ms
Graph created with 3000 vertices and 15000 edges (connected).
3000   |    15000    |   13064 ms
Graph created with 4000 vertices and 20000 edges (connected).
4000   |    20000    |   25485 ms
Graph created with 5000 vertices and 25000 edges (connected).
5000   |    25000    |   44639 ms
Graph created with 6000 vertices and 30000 edges (connected).
6000   |    30000    |  102166 ms
-----
BUILD SUCCESSFUL (total time: 3 minutes 16 seconds)

```

n	Running time of Algorithm	Expected theoretical time efficiency
2000	5101	131589
3000	13064	207913
4000	25485	287179
5000	44639	368631
6000	102166	451827

To calculate the expected theoretical time efficiency, we used the standard time complexity formula for Dijkstra's algorithm with a min-heap:  $\log_2(V) \cdot (E + V) = T(V, E)$

n	Ratio of running time	Ratio of expected time efficiency
2000	N/A	N/A
3000	2.561	1.580
4000	1.951	1.381
5000	1.752	1.284
6000	N/A	N/A



The table shows the growth in actual running time versus theoretical efficiency for each experiment. Ratios are calculated by comparing each value with the one directly before it. The first and last rows are marked as “N/A” because no previous value exists before 2000 and no following value exists after 6000 for comparison. Despite slight variations, the trend in empirical time aligns well with the expected theoretical growth.

Although the empirical and theoretical time efficiency ratios differ slightly, both follow a consistent growth pattern. The empirical results are slightly higher due to system-level factors, such as background processes and implementation details. Despite this, the results confirm the expected  $O((V + E) * \log V)$  behavior of Dijkstra's algorithm and are considered accurate for evaluating performance.

## Difficulties Faced During the Phase Design

We encountered several challenges throughout the phase design of this project, including:

1. **Implementing Dijkstra's Algorithm:** Translating the theoretical algorithm into a working implementation required careful consideration of data structures and performance.
2. **Understanding algorithm logic:** Fully understanding how Dijkstra works in an all-pairs context and ensuring the correctness of the algorithm across different graph inputs was not easy.
3. **Team coordination:** Finding suitable time slots where all team members were available to discuss and test the implementation posed a challenge due to differing schedules.

## Conclusion

In conclusion, this project provided a practical and insightful analysis of Dijkstra's algorithm for solving the all-pairs shortest path problem. We conducted multiple experiments using randomly generated graphs of increasing size to evaluate both the empirical and theoretical performance of the algorithm.

The results showed that the growth of actual running time closely followed the theoretical expectation, with slight variations due to external system-level factors. Overall, the results confirmed that Dijkstra's algorithm is effective for finding shortest paths in sparse graphs and can scale well for large datasets. This makes it a suitable and reliable approach in real-world applications where performance and efficiency are key.

## Appendix

- **GitHub Link**

<https://github.com/Saudh0003/AirFreightApp.git>