Computer Networks

Project Report

By:

Hamza Khan (20i-0583)

Huzaifa Ehsan (20i-2651)

Contents

[Introduction 3](#_Toc152666915)

[Work Division 3](#_Toc152666916)

[Tools Utilized 3](#_Toc152666917)

[1. NetworkX 3](#_Toc152666918)

[2. Matplotlib 3](#_Toc152666919)

[3. Heapq 3](#_Toc152666920)

[4. Time 3](#_Toc152666921)

[Methodology 3](#_Toc152666922)

[1. Initialization 3](#_Toc152666923)

[2. Traversal 3](#_Toc152666924)

[3. Termination 4](#_Toc152666925)

[Performance Analysis 4](#_Toc152666926)

[**Case 1** 4](#_Toc152666927)

[6 nodes and 6 edges 4](#_Toc152666928)

[6 nodes and 8 edges (Adding 2 more edges) 5](#_Toc152666929)

[**Case 2** 5](#_Toc152666930)

[**Conclusion** 6](#_Toc152666931)

# Introduction

The primary objective of this project was to create user friendly application by which users can easily create graphs and analyze the performance of Dijkstra's algorithm using them. This report contains the work division, tools used, and methodology used to implement Dijkstra’s algorithm. Finally, we also executed this code for different number of nodes as well as edges and observed how the Dijkstra’s algorithm performs in each scenario.

# Work Division

|  |  |
| --- | --- |
| Hamza Khan | Huzaifa Ehsan |
| Network Simulation | Dijkstra implementation |
| User Interface | Visualization |
| Dynamic Change | Performance Analysis |
| Report |  |

# Tools Utilized

1. NetworkX**:** For the creation and manipulation of graphs. We can use it to add nodes and edges as well as visualize graphs.
2. Matplotlib**:** In this code, Matplotlib is used for network visualization.
3. Heapq**:** Provides an implementation of the heap queue algorithm. The **heapq** module is used here for implementing Dijkstra's algorithm to efficiently compute the shortest path in the graph.
4. Time**:** Used for measuring the execution time for computation of the shortest path using Dijkstra's algorithm.

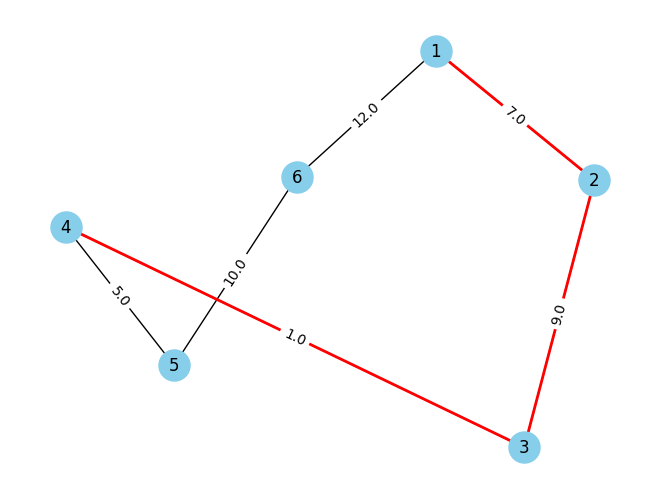
# Methodology

1. Initialization:
   * **distances**: A dictionary is initialized to store the shortest distance from the start node to all other nodes in the graph. Initially, all distances are set to infinity except for the start node, which is set to 0.
   * **priority\_queue**: A priority queue (implemented using **heapq**) is initialized with tuples of **(distance, node)** pairs, where distance represents the cumulative distance from the start node to a particular node.
2. Traversal:
   * The algorithm continues until the priority queue is empty. In each iteration:
     + The node with the minimum distance from the start node is extracted from the priority queue.
     + If the extracted node is the destination node, the shortest path has been found. The algorithm reconstructs and returns the shortest path and its distance.
     + Otherwise, the algorithm explores neighboring nodes of the current node.
       - For each neighboring node, it calculates the distance from the start node through the current node and compares it to the known shortest distance.
       - If this newly calculated distance is shorter than the previously recorded distance, it updates the distance and adds the neighboring node to the priority queue for further exploration.
       - The visited set keeps track of nodes that have been visited to avoid redundant exploration.
3. Termination:
   * If the destination node is not reached after the priority queue is empty, it means there is no path from the start node to the destination node, and an empty path and infinite distance are returned.

# Performance Analysis

## **Case 1**

### 6 nodes and 6 edges

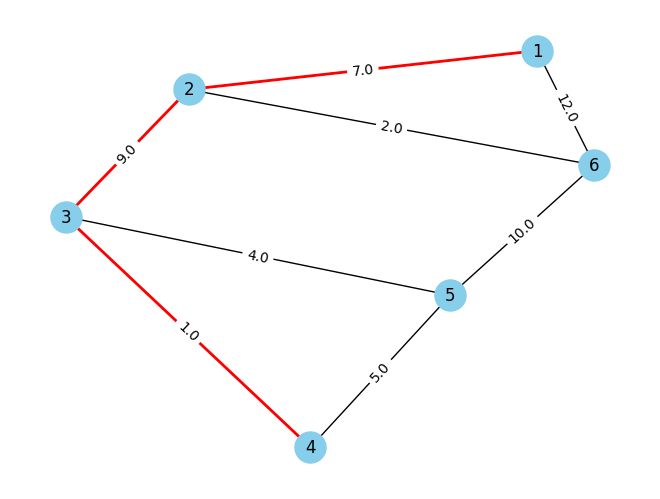


Path: 1 -> 4

Shortest Distance: 17.0

time taken: 4.840000019612489e-05

### 6 nodes and 8 edges (Adding 2 more edges)

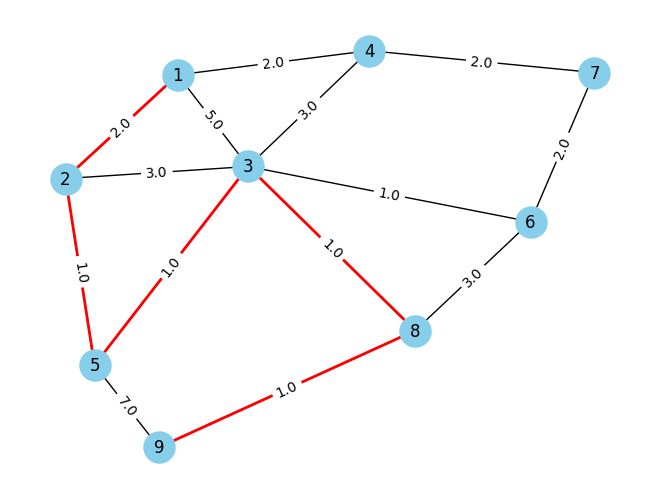


Path: 1 -> 4

Shortest Distance: 17.0

time taken: 4.939999962516595e-05

## **Case 2**



Path: 1 -> 9

Shortest Distance: 6.0

Time taken: 0.00011360000007698545

## **Conclusion**

In case 1, initially both the number of nodes and edges were kept 6. This resulted in very little time to find the shortest path from 1 to 4. Then, 2 extra edges were added to the same graph. The result this time was slightly different as there was a minute increase in time taken to find the same path.

In case 2, there were 9 nodes and 14 edges (compared to case 1, both nodes and edges were considerably increased). In this case, there was a considerable increase in time taken to find the shortest path.