**Project 2 Report**

Joey Parker

University of North Carolina at Charlotte

ITCS 6114-81

Dr. Dewan Ahmed

July 28, 2024

Project GitHub Repository: <https://github.com/Ninjajkl/ITCS-6114-Project-2/>

Table of Contents

[1 Insertion Sort 3](#_Toc171280704)

[2 Merge Sort **Error! Bookmark not defined.**](#_Toc171280705)

[3 Heap Sort **Error! Bookmark not defined.**](#_Toc171280706)

[4 Quick Sort **Error! Bookmark not defined.**](#_Toc171280707)

[4.1 In-Place Quick Sort 5](#_Toc171280708)

[4.2 Modified Quick Sort **Error! Bookmark not defined.**](#_Toc171280709)

[5 Simulation **Error! Bookmark not defined.**](#_Toc171280710)

[5.1 Results Analysis **Error! Bookmark not defined.**](#_Toc171280711)

[5.2 Special Cases **Error! Bookmark not defined.**](#_Toc171280712)

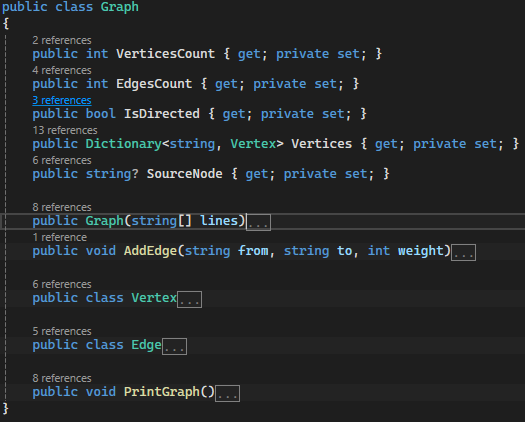
[5.2.1 Sorted Input **Error! Bookmark not defined.**](#_Toc171280713)

[5.2.2 Reverse Sorted Input **Error! Bookmark not defined.**](#_Toc171280714)

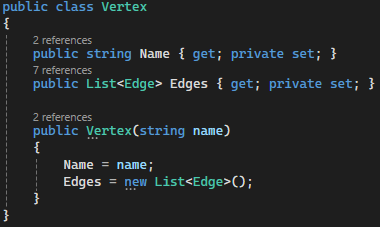
[5.3 Takeaway **Error! Bookmark not defined.**](#_Toc171280715)

# 1 Graph Data Structure

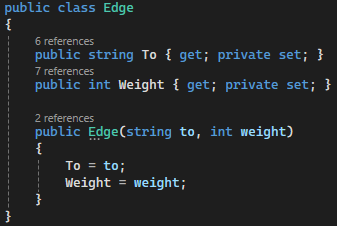
My graph structure has five members. The Vertices Count (int), the Edges Count (int), whether the graph is directed or undirected (bool), the name of the Source node/vertex (string), and a <string, Vertex> Directory called Vertices that holds all the vertices in the graph, given their string name.



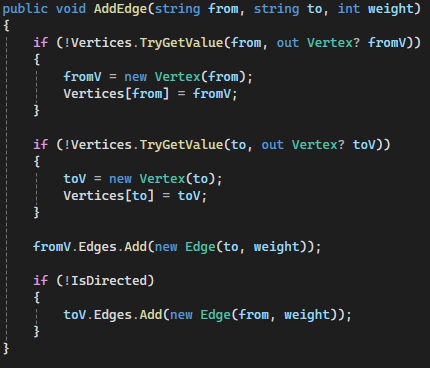
Each Vertex contains its Name (string) and a list of Edges originating at the Vertex.



Each Edge contains the To Vertex Name (string) and the Weight of the Edge (int).



To make the graph possible for undirected edges (as the edges are only stored by one of the connected vertices), each time we add an edge to a vertex, we also add the edge again to the connected vertex. This does not happen for directed graph generation.



# 2 Note on Input Format + Instructions to Run

The input format is the same as the given file format, but the last line depicting the Source Node/Vertex is Not Optional.

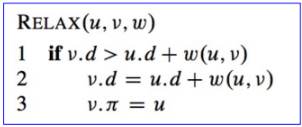
To run the program, follow the instructions in the README.md

# 3 Shortest Path

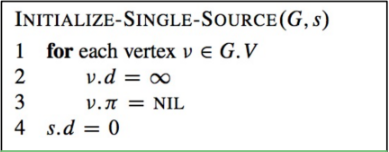
## 3.1 Pseudocode

The Pseudocode for Dijkstra’s algorithm is given in 3 parts: Relax, Initialize-Single-Source, and the main body.

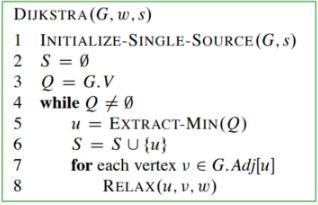
The pseudocode for Relax is as follows (taken from the lecture slides):



The pseudocode for Initialize-Single-Source is as follows (taken from the lecture slides):



The pseudocode for the main body is as follows (taken from the lecture slides):



## 3.2 Pseudocode Runtime

V will be the number of vertices. E will be the number of edges. The Initialize-Single-Source operation takes time O(V). The time to build the binary min-heap Q is O(V log V). The Extract-Min operation for a min-heap priority queue is O(log V). The Relax operation takes time O(log V), and there can be at most O(E) such operations.

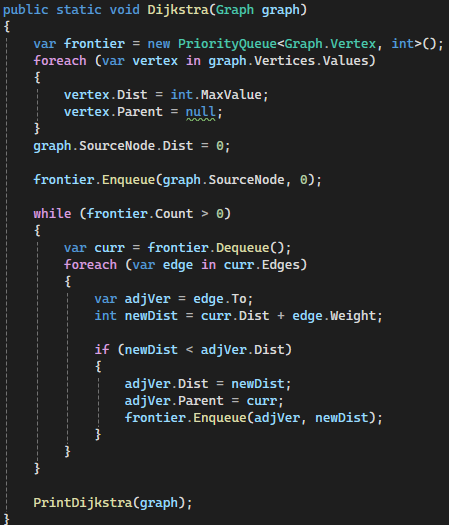
The time complexity of Dijkstra’s when using a min-heap priority queue is therefore O(V log V + E log V) or O((V + E) log V). If all vertices are reachable from the source it is O(E log V).

## 3.3 Runtime of Actual Implementation

My implementation of Dijkstra’s algorithm used the min-heap priority queue data structure for the frontier. As mentioned above, this gives my implementation of Dijkstra’s Algorithm an O((V + E) log V) runtime.

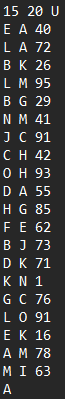
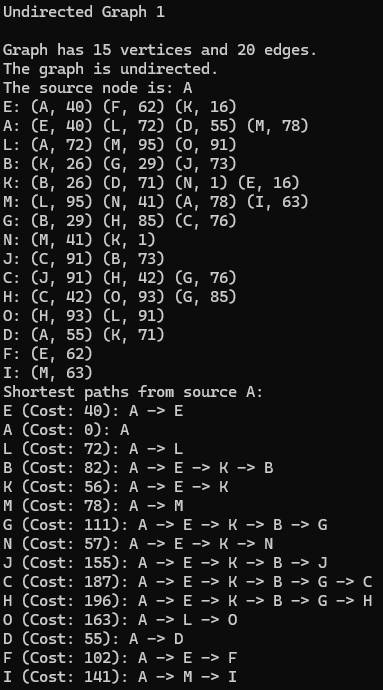
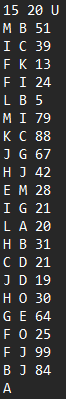
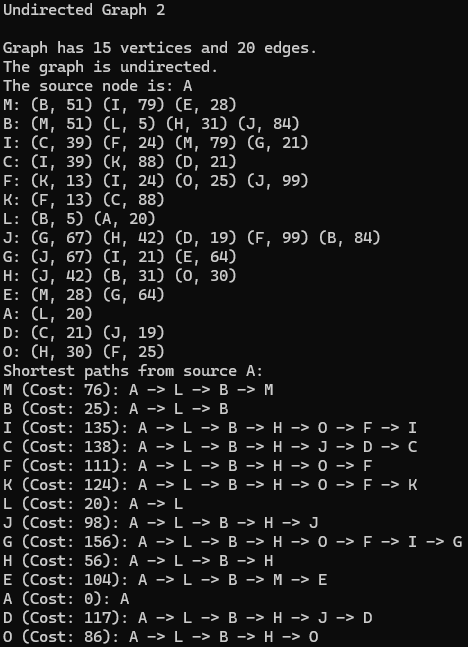
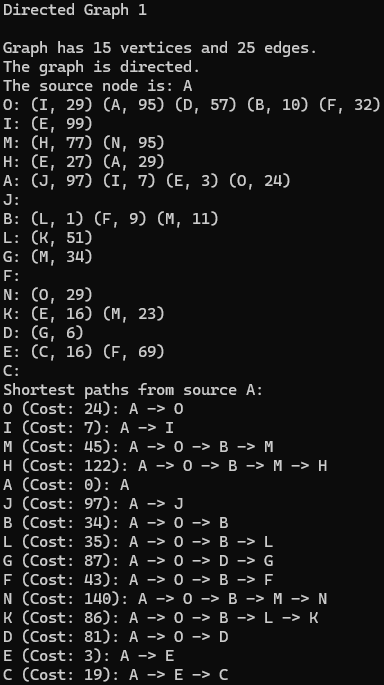
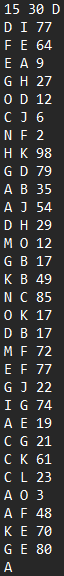
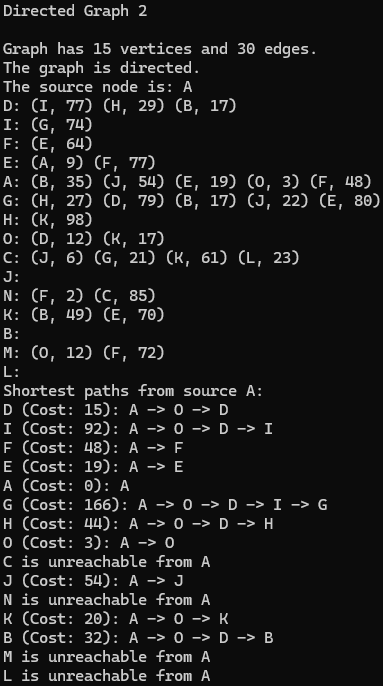
## 3.4 Implementation Code

Here is the code for my Implementation of Dijkstra’s



## 3.5 Sample Graphs

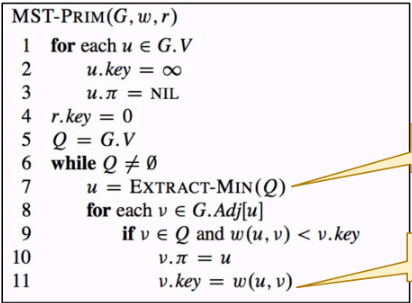
For this section I will have the Input on the left, and the Output for that Graph on the right. The first two graphs are undirected, and the last two are directed.

1.  
2.  
3.  
4.  

# 4 Minimum Spanning Tree

## 4.1 Pseudocode

The Pseudocode for Prim’s algorithm is given in just one part (taken from the lecture slides):



## 4.2 Pseudocode Runtime

V will be the number of vertices. E will be the number of edges. The Initialization operation takes time O(V). The time to build the binary min-heap Q is O(V log V). The Extract-Min operation for a min-heap priority queue is O(log V).

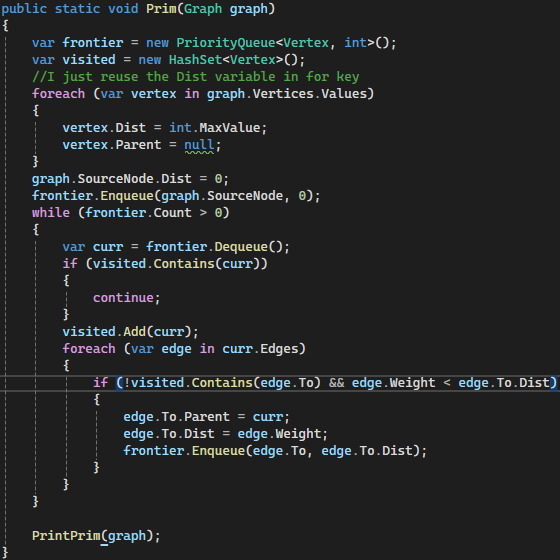
Updating the priority of a vertex’s neighbors has a few sub-complexities within it. It first involves checking each edge connected to the vertex, which is worst case E edges. Considering each edge and changing its values takes O(log V) time. Since this happens at most E times, the total time complexity for this section is O(E log V)

The time complexity of Prim’s when using a min-heap priority queue is therefore O(V log V + E log V) or O((V + E) log V).

## 4.3 Runtime of Actual Implementation

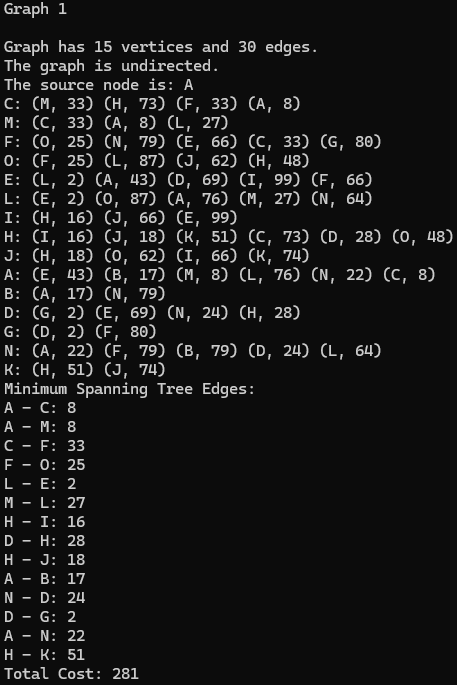
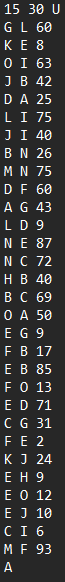
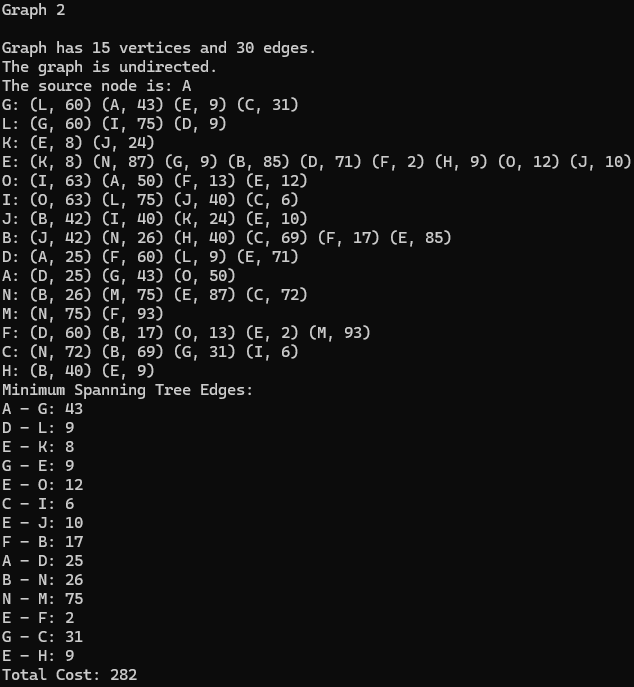
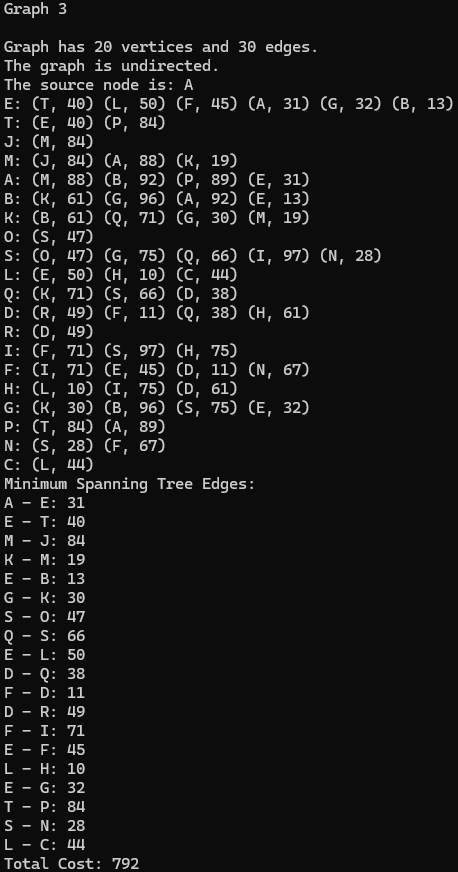
My implementation of Prim’s algorithm used the min-heap priority queue data structure for the frontier. As mentioned above, this gives my implementation of Prim’s Algorithm an O((V + E) log V) runtime.

## 4.4 Implementation Code



## 4.5 Sample Graphs

For this section I will have the Input on the left, and the Output for that Graph on the right.

1.  
2.  
3.  
4.  