

Spring 2021

Dijkstra's Pathfinder

Taylor F. Malamut
tfmalamut@coastal.edu

Follow this and additional works at: <https://digitalcommons.coastal.edu/honors-theses>



Part of the [Theory and Algorithms Commons](#)

Recommended Citation

Malamut, Taylor F., "Dijkstra's Pathfinder" (2021). *Honors Theses*. 423.
<https://digitalcommons.coastal.edu/honors-theses/423>

This Thesis is brought to you for free and open access by the Honors College and Center for Interdisciplinary Studies at CCU Digital Commons. It has been accepted for inclusion in Honors Theses by an authorized administrator of CCU Digital Commons. For more information, please contact commons@coastal.edu.

Dijkstra's Pathfinder

By

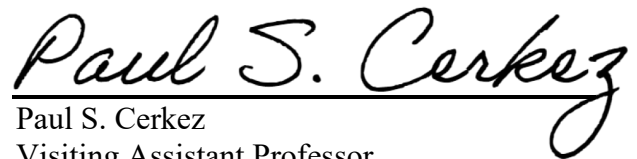
Taylor Malamut

Information Systems

Submitted in Partial Fulfillment of the
Requirements for the Degree of Bachelor of Science
In the HTC Honors College at
Coastal Carolina University

Spring 2021

Louis E. Keiner
Director of Honors
HTC Honors College


Paul S. Cerkez
Visiting Assistant Professor
Department of Computing Sciences
Gupta College of Science

Abstract

Dijkstra's algorithm has been widely studied and applied since it was first published in 1959.

This research shows that Dijkstra's algorithm can be used to find the shortest path between two stations on the Washington D.C. Metro. After exploring different types of research and applying Dijkstra's algorithm, it was found that the algorithm will always yield the shortest path, even if visually a shorter path was initially expected.

Keywords: Dijkstra, graph theory, washington metro, public transportation

Contents

ABSTRACT	2
CHAPTER 1	4
INTRODUCTION.....	4
<i>Problem Statement</i>	4
<i>Relevance and Significance</i>	4
CHAPTER 2	5
REVIEW OF THE LITERATURE.....	5
CHAPTER 3	9
METHODOLOGY.....	9
<i>Content Delivery</i>	9
<i>First Steps</i>	9
<i>GUI Creation</i>	10
<i>Testing</i>	10
CHAPTER 4	12
RESULTS AND DISCUSSION	12
<i>Same Line</i>	12
<i>Different Line</i>	12
<i>GUI Results</i>	12
CHAPTER 5	13
CONCLUSION AND RECOMMENDATIONS	13
<i>Conclusion</i>	13
<i>Future Work</i>	13
REFERENCES.....	15
APPENDIX A.....	16
<i>Brief Introduction to Graph Theory</i>	16
<i>Dijkstra's algorithm</i>	16
APPENDIX B	18
WASHINGTON D.C. METRO MAP	18
APPENDIX C	19
RESULTS FROM TESTING.....	19
APPENDIX D.....	20
GRAPHICAL USER INTERFACE.....	20

Chapter 1

Introduction

Problem Statement

The problem of finding the shortest path between point A and B has been prevalent for centuries. Because of more advanced transportation, its importance has grown exponentially. Transportation systems, notably the subway/metro, can be confusing to navigate. This problem was chosen because of foundational knowledge of graph theory and interest in exploring a pathfinding algorithm. The project used Dijkstra's algorithm to meet the goal of finding the shortest path between two stations on the Washington D.C. Metro.

Relevance and Significance

Locals and tourists alike use the Washington Metro every day. Therefore, it is important that they can navigate the metro efficiently. Dijkstra's algorithm is used for applications like routing protocols and telephone networks (Kumar, 2020). Also, Dijkstra's algorithm is actively being used for new advancements such as delivery drones (Kumar).

Barriers and Issues

As transportation systems add more stations, this raises an issue with the relevance of the project in the future. As this project's lifespan was only one semester, there was no plan to take that into account. Another barrier is that sometimes all trains are not functional, or lines are under maintenance. As the project's main purpose is to implement one algorithm, this was not addressed. Finally, the scope of this project is rail, and did not include any other transportation system.

Chapter 2

Review of the Literature

Introduction

Dijkstra's algorithm is used for a wide array of applications. Therefore, it was important to explore different types of research about Dijkstra's algorithm. The sources in this chapter were obtained by using Coastal Carolina University's student access to the ACM (Association for Computing Machinery) and IEEE (Institute of Electrical and Electronic Engineers) databases. Also, the literature review touches on the importance of the Washington D.C. Metro system.

Mumbai's BEST Bus System

Although Dijkstra's Pathfinder does not use the bus system, it is important to look at how Dijkstra's algorithm has been applied to similar problems. Kejriwal and Temrikar (2019) explore how the algorithm can be used to find the shortest path between two stations in Mumbai's BEST bus system. Kejriwal and Temrikar pointed out that recent studies have shown that commuters can waste an average of fifteen minutes because of their lack of knowledge on the fastest possible route. The authors also discuss that edges on the graph should be bidirectional and that applies to Dijkstra's Pathfinder as well. Additionally, they made similar assumptions like that all buses are functional and there are no obstacles on the road.

Dijkstra's Algorithm for a Proposed Tramway

Agarana, Omoregbe, and Ogunpeju (2016)'s research focuses on a tramway in a university system. Trams are rail-based and run-on streets - the largest one being in Melbourne, Australia (Agarana, Omoregbe, & Ogunpeju). They point out that Dijkstra's algorithm can be used with

some assumptions, like edges having non-negative weights. Also, there are two common variants of the algorithm. The original algorithm is finding the shortest path between two nodes, while another version is finding the shortest path from a singular source node to every other node in the graph (Agarana, Omoregbe, & Ogunpeju). For Dijkstra's Pathfinder, the original algorithm will be used.

Routing Processes

Dijkstra's algorithm can be used to solve different kinds of routing problems. For instance, it is used for network related protocols and applications like Google Maps (Gupta, Mangla, Jha, & Umar, 2016). Dijkstra's algorithm can also be used for file servers by decreasing the number of "hops" from a server to all of the computers on that network (Gupta, Mangla, Jha, & Umar). The algorithm is useful when planning paths for a robot. The robot is given a source and destination address from a remote server using IEEE's standard communication protocol (Gupta, Mangla, Jha, & Umar). Then, the robot will move in the given direction. Finally, it can be used when creating flight agendas given the departure and arrival time and the distance from the origin airport.

Airline Network Planning

Planning flights is an important effort that constantly needs to be optimized. Yan and Jun (2010)'s research focuses on optimizing the airline network planning process by use of Robust Optimization and Dijkstra's algorithm (Yan & Jun). The combination of these two algorithms is needed because of the rapid growth of air traffic volume. The main part of their paper is about selecting hub airports and having it be properly allocated in a way that some operating costs of the network are minimized (Yan & Jun). This includes variables like load distance and travel

cost. Their paper goes into detail about the Deviation Robust Optimization method, but that is not applicable for Dijkstra's Pathfinder.

Dijkstra's algorithm is used as a way to improve the initial solution and neighborhood structure (Yan & Jun). Yan and Jun also pointed out that their Deviation Robust Optimization Method uses Dijkstra's algorithm to get the best connection status between each pair of cities.

Multi-layered Social Networks

Dijkstra's algorithm, as seen previously in this chapter, is mainly used for routing problems. Another way that the algorithm has been implemented is through analyzing social networks. Social networks can be described as a finite set of actors (nodes) and relationships (edges) that link the actors (Brodka, Stawiak, & Kazienko, 2011). There are different types of networks that can be expressed like corporate partnerships, scientist collaboration networks, and friendship networks (Brodka, Stawiak, & Kazienko). The most notable of these networks are online social networks on social media websites like Facebook. Their research does a variation on the algorithm to fit their needs (involving pre-processing) in order to extract shortest paths.

Music Retrieval

Another case study is the use of Dijkstra's algorithm in non-traditional ways. The Echo Nest corporation is owned by Spotify and focuses on music retrieval and development. They provide a number of retrievals, search, and interactivity tools to places like music stores and record labels (Jehan, Lamere, & Whitman, 2010). Their platform contains tools like audio signal processing, machine learning, natural language processing, and graph manipulation.

For graph manipulation, they have been able to generate playlists using graphs. The playlist is generated by building a directed graph that represents nodes as musicians and edges as similarity. Then, Dijkstra's algorithm is used to find the shortest path between two musicians

(Jehan, Lamere, & Whitman). Because of user constraints like desired musician familiarity and exclude lists, the edges' weights have to be adjusted dynamically. A playlist can be created by having the edges to musicians with a high familiarity yield the lowest cost (Jehan, Lamere, & Whitman).

Why Metro Matters

The main purpose of Dijkstra's Pathfinder is to produce the shortest path. However, it would be a mistake to not include a section in this chapter on the importance of the real-life implementation. The presence of the metro increases access to jobs and businesses. Two million jobs (fifty-four percent of all jobs in the region) are within a half mile radius of all metro rail and bus stations (WMATA, 2012). Also, it makes the region affordable and livable. According to WMATA (Washington Metropolitan Area Transit Authority), the metro saves all households 705 million dollars a year in time savings. If a transit system did not exist, congestion would increase by twenty-five percent and cost more than 1.5 billion dollars annually in wasted time and fuel (WMATA).

Summary

The first four sources directly describe how Dijkstra's algorithm is used for navigation. The Multi-layered Social Networks and Music Retrieval sections explain how Dijkstra's algorithm can be used for other applications successfully. Finally, the last source explains the importance of the Washington D.C. Metro system. The literature guided this project by providing tangible research on how Dijkstra's algorithm can be used for pathfinding. Moreover, it was important to include research that uses Dijkstra's algorithm for other purposes to bring in additional context on the diverse number of applications of the algorithm.

Chapter 3

Methodology

Content Delivery

The content was delivered as a desktop application. The project used Java SE (Standard Edition) 15.0.2, which was the latest release by Oracle. Java Swing, a GUI (Graphical User Interface) toolkit, was used to develop the frontend of the application. Swing is included in the Oracle JDK (Java Development Kit). In the GUI, users are able to pick a starting and stopping station. Then, the algorithm finds the shortest path between the two stations and displays the list of stations to visit. Because of efforts related to getting the algorithm completed, the dynamic GUI originally proposed did not get implemented due to time constraints by the end of the semester and is now described in the Future Work section of Chapter 5. After receiving permission from the instructor of this course, Eclipse was used as the IDE (Integrated Development Environment).

First Steps

The first step was to collect the names of all of the stations. The purpose of having all of the stations in a text file was to have it read in by the drop-down menus. Afterwards, the rest of the text files were created, including the edges for each line. Next, the base files were created. The Vertex class represents a single station, and the Edge class represents the weight/distance between each pair of stations. Next, the Dijkstra class was created to compute the shortest path. Originally, a Graph class was created to represent the vertices and edges. However, it was found that it created unnecessary obstacles. Instead, it was easier to use the Vertex and Edge classes

directly in the Dijkstra class. An explanation of graphs, vertices, edges, and Dijkstra's algorithm can be found in Appendix A. Finally, the Tester class was created to pass in the two stations, call the algorithm, and print the path.

GUI Creation

The Tester class was renamed to Display to properly communicate that class's purpose. The GUI class was created to be executed from the Display class. The two drop-down menus were populated with all of the stations from the text file described in the First Steps section of this chapter. Next, the go and reset buttons were created. The go button is responsible for calling the algorithm once the user makes a valid selection (two non-identical stations). See Figure 4 in Appendix D for an example of what the user sees after clicking the go button with a valid selection.

If the user tries to click the go button without selecting a starting and/or stopping station, a pop-up message is displayed instructing the user to make a valid selection (see Figure 2 in Appendix D). Also, if the user selects two identical stations, a pop-up message is displayed instructing the user to select non-identical stations and the drop-down menus are automatically reset after the user clicks the ok button in the pop-up message box (see Figure 3 in Appendix D). If the user selects the reset button, both drop-down menus return to their default positions, shown on Figure 1 in Appendix D. The final step was to insert an image of the metro map. The image can be found in Appendix B. Also, see Figure 1 in Appendix D for the initial starting screen with a full view of the map.

Testing

The algorithm, described in Appendix A, was tested on a subset of the metro before including the entire system. First, a portion of the Red Line was used to test when the two stations are on

the same line. Then, the entire Red Line was included to conclude the testing of that functionality. To test when the two stations are on different lines, the Silver Line was used with the Orange Line. Once that was functioning correctly, the rest of the lines were added one-by-one.

The GUI was tested by first ensuring that the drop-down menus contained every station. After checking the menus, it was important to make sure that the user cannot execute the algorithm if a valid selection is not made. If the user clicked the go button without selecting a station for either menu, the pop-up message was displayed properly each time and testing concluded for that feature. The other pop-up message described in the GUI Creation section was tested similarly. Finally, the path display was tested by ensuring that it matched the path computed when testing without the GUI.

Chapter 4

Results and Discussion

Same Line

If two stations are on the same line, the intermediate stations were collected to yield the correct path. The same line functionality was tested without the GUI and worked as expected. Detailed results can be found on Table 1 in Appendix C.

Different Line

Dijkstra's algorithm became more involved when the two stations were not on the same line. A description of the algorithm can be found in Appendix A. On Table 2 in Appendix C, the last two cases worked as initially expected. The last case did use a different transfer station, but it did not impact the total cost of the trip as either transfer station could have been used (see Appendix B to view the map).

On the other hand, the first two cases yielded a shorter path than initially expected. Although visually another path was expected, Dijkstra's algorithm found the actual shortest path. The different line functionality was also tested without the GUI.

GUI Results

The drop-down menus contain a verified list of all stations in the Washington D.C. Metro system. In order to prevent errors, the system does not allow the start and stop stations to be the same or not selected at all. The details of how the menus were constructed and tested can be found in Chapter 3.

Chapter 5

Conclusion and Recommendations

Conclusion

In the results chapter, it can be seen that proper implementation of Dijkstra's algorithm will always yields the shortest path. Therefore, Dijkstra's algorithm is the best tool for finding the shortest path on the Washington D.C Metro. Based on this research, Dijkstra's algorithm could work for similar transportation systems. While the algorithm was done on a single metro system, it would work for other metro systems by retrofitting the software for that particular metro system (e.g., modifying text files). Additionally, it could also be used for navigating between cities.

Future Work

In order for the application to stay relevant, it is important to have a way to add new stations and lines. Future capabilities could include a dynamic map and a way to visually see the path traced on the map. Also, real-time updates about maintenance and weather delays could be added to factor into travel times. Additionally, the application could take into account the size of each train, capacity, and time of day. As rail is not the only transportation system in the Washington D.C. Metro, buses could be added to the pathfinding functionality. Furthermore, taxies and other ridesharing platforms like Uber and Lyft could be included as well. If travelling by bus or taxi, there would need to be a way to determine the traffic level - the software could use Google Map's API (Application Programming Interface).

Currently, the software is delivered as a desktop application. For further ease of use, a mobile implementation would be ideal for a commuter on the go. As the project was written in Java, the most natural transition to a mobile application would be Android. Also, the application could be expanded to include other cities' transportation systems like the New York City Subway. Furthermore, some commuters travel by water to arrive at their destination. Finally, the application could include air for longer trips.

References

- Agarana, M. C., Omoregbe, N. C., & Ogunpeju, M. O. (2016). Application of Dijkstra Algorithm to Proposed Tramway of a Potential World Class University. *Applied Mathematics*, 07(06), 496-503. doi:10.4236/am.2016.76045
- Brodka, P., Stawiak, P., & Kazienko, P. (2011). Shortest Path Discovery in the Multi-layered Social Network. *2011 International Conference on Advances in Social Networks Analysis and Mining*. Doi:10.1109/asonam.2011.67
- Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. In *Numerische Mathematlk* (Vol. 1, pp. 269-271). doi:10.1007/BF01386390
- Gupta, N., Mangla, K., Jha, A., & Umar, M. (2016). Applying Dijkstra's Algorithm in Routing Process. *International Journal of New Technology and Research (IJNTR)*, 2(5), 122-124.
- Jehan, T., Lamere, P., & Whitman, B. (2010). Music retrieval from everything. *Proceedings of the International Conference on Multimedia Information Retrieval - MIR 10*. doi:10.1145/1743384.1743428
- Kejriwal, A., & Temrikar, A. (2019). Graph Theory and Dijkstra's Algorithm: A solution for Mumbai's BEST buses. *The International Journal of Engineering and Science (IJES)*, 8(10), 1st ser., 40-47.
- Kumar, D. (2020, August 21). Applications of Dijkstra's shortest path algorithm. Retrieved from <https://www.geeksforgeeks.org/applications-of-dijkstras-shortest-path-algorithm/>
- Naimzada, A. K., Stefani, S., & Torriero, A. (2009). *Networks, topology and dynamics: Theory and applications to economics and social systems*. Berlin: Springer.
- WMATA. (2012, February). Why Metro Matters. Retrieved from <https://www.wmata.com/initiatives/case-for-transit/>
- Yan, Z., & Jun, Z. (2010). Dijkstra's algorithm based robust optimization to airline network planning. *2010 International Conference on Mechanic Automation and Control Engineering*. doi:10.1109/mace.2010.5536824

Appendix A

Algorithmic Description

Brief Introduction to Graph Theory

Before describing Dijkstra's algorithm, it is important to briefly explain what a graph is. A graph X is an ordered pair of sets where $X = (V, E)$ (Naimzada, Stefani, & Torriero, 2009). V represents the vertices (also known as nodes) and E represents the edges. Each edge connects a pair of vertices. Also, edges can have weights associated with them. For example, weights can represent distance or time. Visually, a graph can be represented by drawing the vertices as dots and edges as lines between each pair of dots (Naimzada, Stefani, & Torriero, 2009).

Dijkstra's algorithm

Dijkstra's algorithm was created by Dutch computer scientist Edsger Dijkstra in 1956 and published in 1959. In his original paper, *A Note on Two Problems in Connexion with Graphs*, Dijkstra proposed solutions to two problems in graph theory. The first problem asks to "construct the tree of minimum total length between the n nodes (a tree is a graph with one and only one path between every two nodes)" (Dijkstra, 1959). The second problem Dijkstra proposed was to "find the path of minimum length between two given nodes P and Q ." The second problem was solved first in 1956 and is the focus of this capstone project.

Dijkstra points out the fact that "if R is a node on the minimal path from P to Q , knowledge of the latter implies the knowledge of the minimal path from P to R ." In Dijkstra's solution, "the minimal paths from P to the other nodes are constructed in order of increasing length until Q is reached."

Before explaining the algorithm further, Dijkstra lists how the nodes and branches (edges) are subdivided into sets. Set A contains "the nodes for which the path of minimum length from P is

known; nodes will be added to this set in order of increasing minimum path length from node P.” Set B contains “the nodes from which the next node to be added to set A will be selected; this set comprises all those nodes that are connected to at least one node of set A but do not yet belong to A themselves.” Set C contains the rest of the nodes. Set I contains “the branches occurring in the minimal paths from node P to the nodes in set A.” Set II contains “the branches from which the next branch to be placed in set I will be selected; one and only one branch of this set will lead to each node in set B.” Set III contains the rest of the branches.

The following steps also originate from Dijkstra (1959). The first step is to consider all branches connecting the node just transferred to set A with nodes R in sets B or C. If node R belongs to set B, then it is investigated whether the use of the current branch yields a shorter path from P to R than the known path that uses the branch in set II. If that is the case, it replaces the branch in set II. Otherwise, the branch is rejected. If node R is in set C, then it is added to set B and the branch is added to set II.

Next, Dijkstra acknowledges that “every node in set B can be connected to node P in only one way if we restrict ourselves to branches from set I and one from set II.” Therefore, each node in set B has a distance from node P and the node with minimum distance from P is transferred from set B to set A, and the corresponding branch is transferred from set II to set I. Next, return to the steps described in the paragraph directly above until node Q is in set A. Then, the solution has been found.

Appendix B

Washington D.C. Metro Map



Appendix C

Results from Testing

Start	Stop	Expected Total Weight (mi)	Actual Total Weight (mi)
Shady Grove	Friendship Heights	13.5	13.5
Branch Ave	Congress Heights	9.0	9.0
Huntington	Pentagon	11.6	11.6

Table 1 - Same Line

Start	Stop	Expected Transfer	Expected Total Weight (mi)	Actual Transfer	Actual Total Weight (mi)
McLean	Waterfront	Rosslyn + Pentagon	17.0	L'Enfant Plaza	16.5
Metro Center	Waterfront	L'Enfant Plaza	2.4	Gallery Place	2.3
Brookland-CUA	Waterfront	Gallery Place	8.5	Gallery Place	8.5
McLean	Minnesota Ave	East Falls Church	21.3	Stadium-Armory	21.3

Table 2 - Different Line

Appendix D

Graphical User Interface

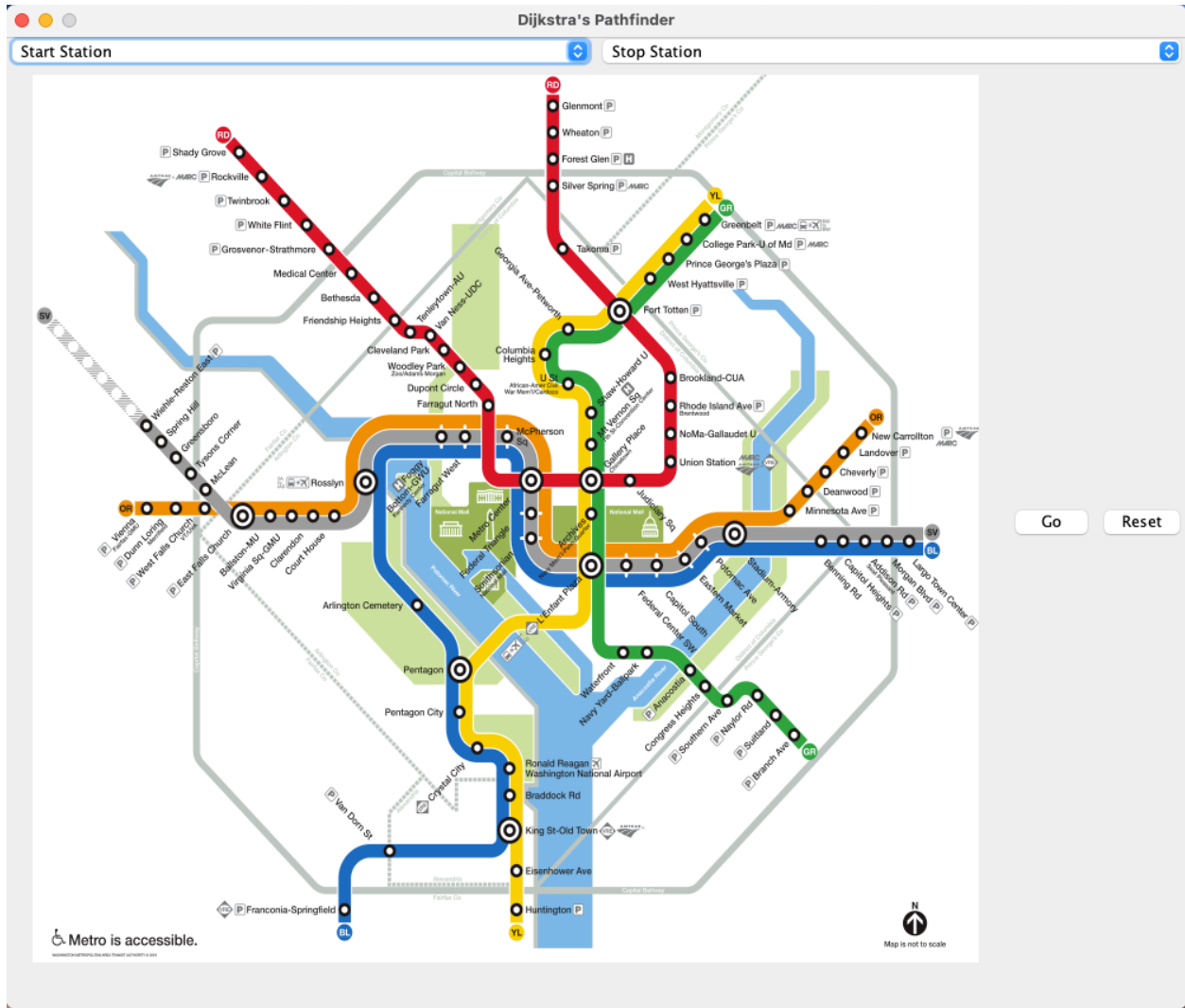


Figure 1 - Start screen

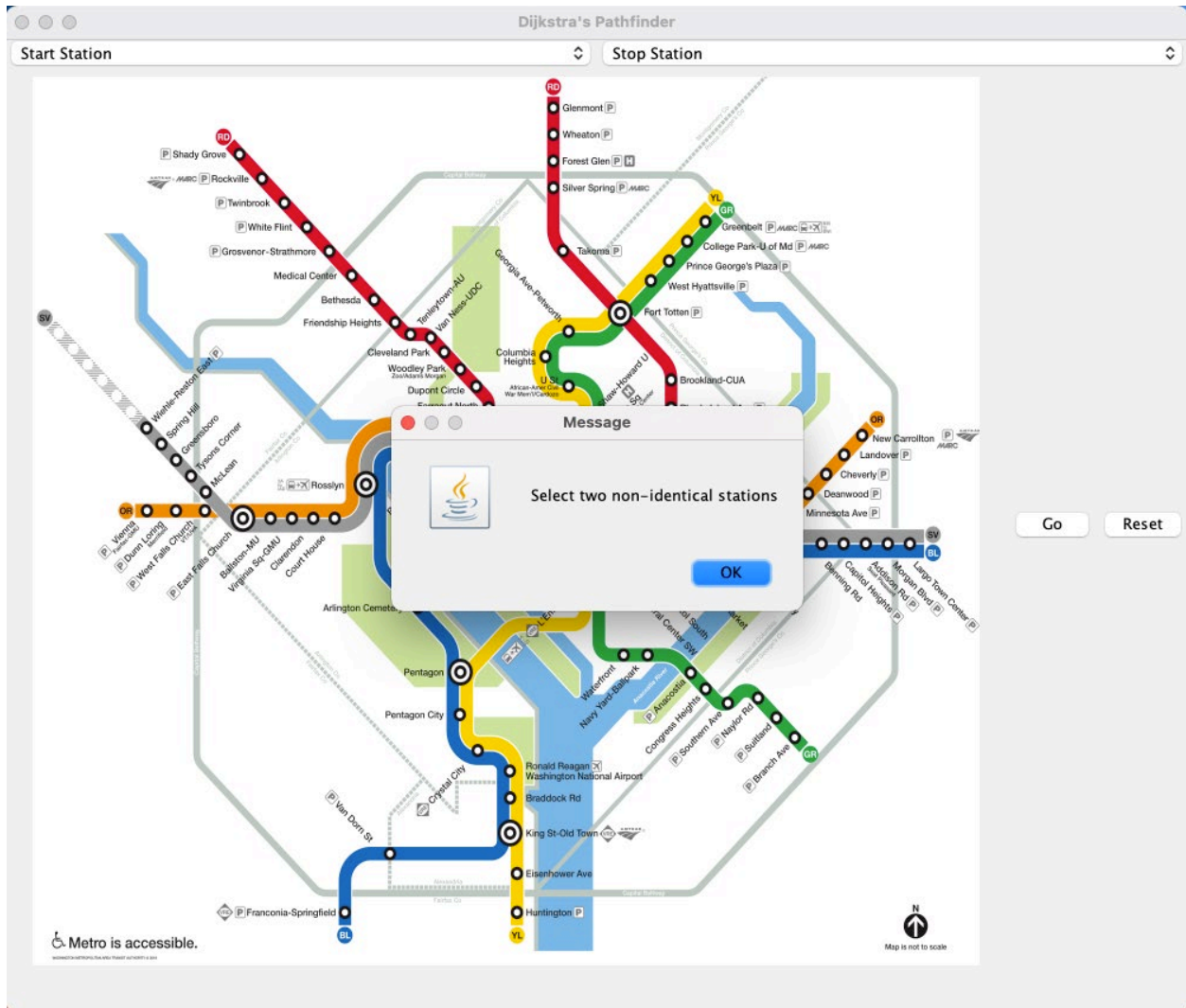


Figure 2 - Pop-up if Go clicked without selecting stations

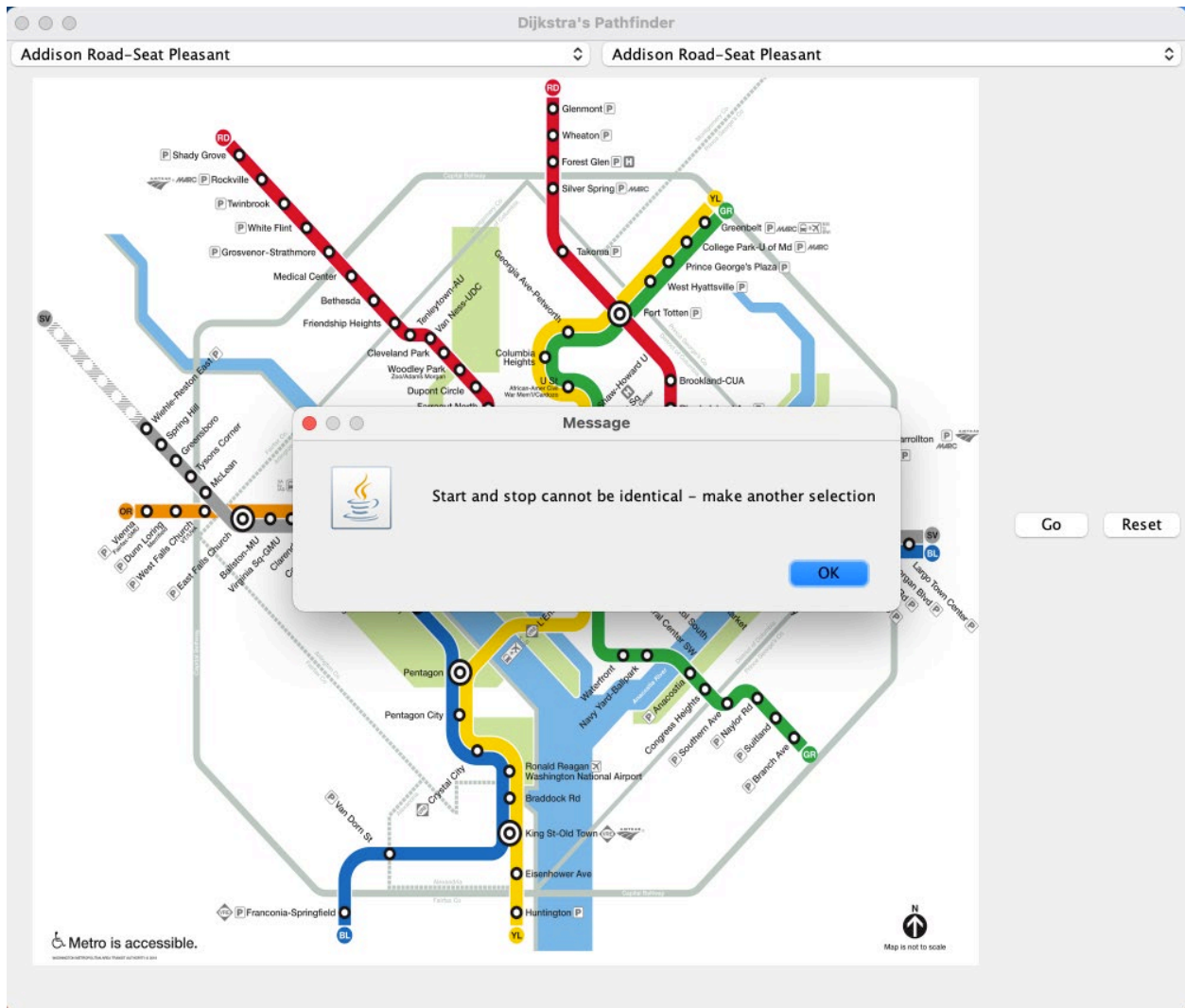


Figure 3 - Pop-up if Go clicked with two identical stations

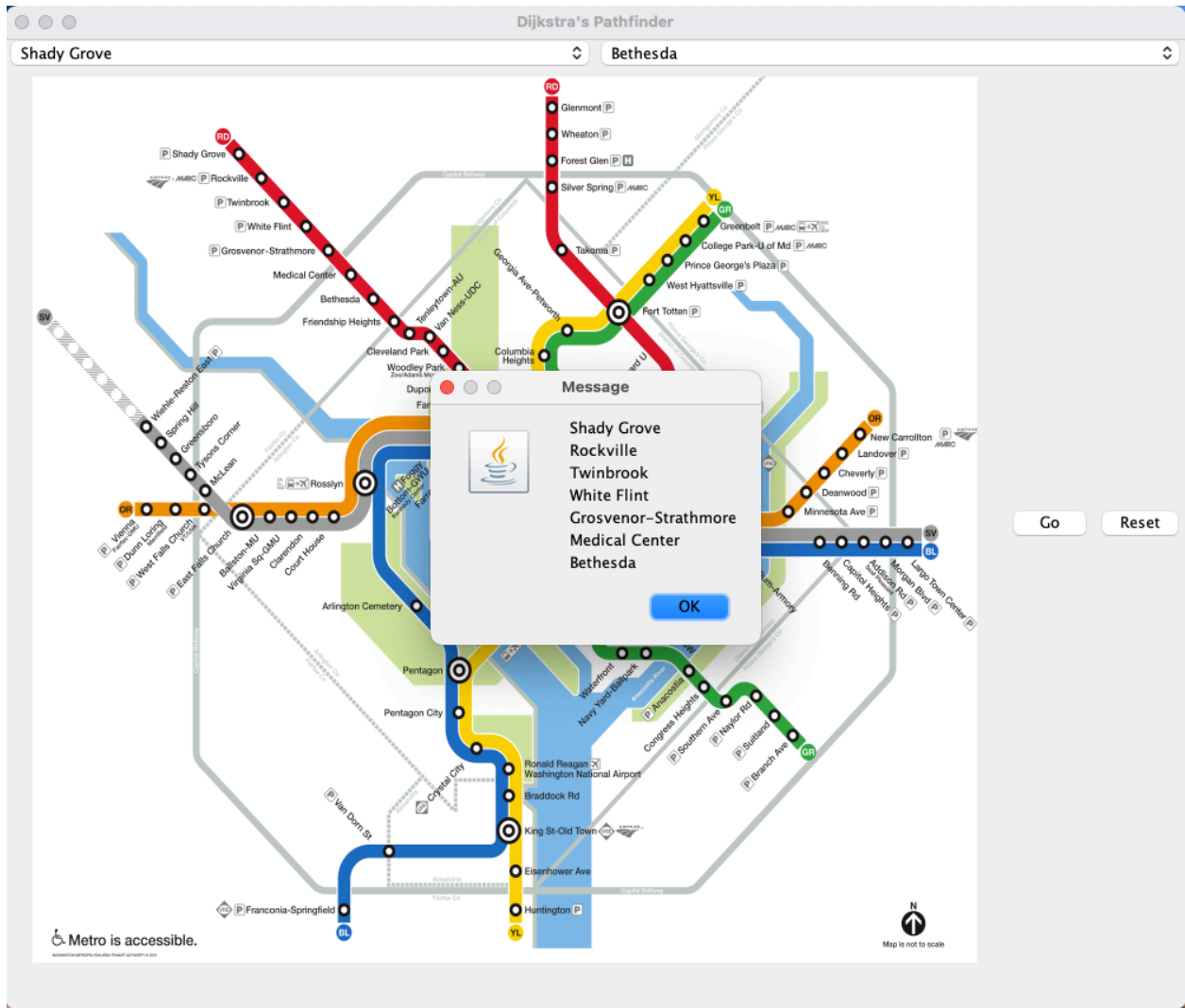


Figure 4 - Displaying list of stations