DSA Assignment

20175535

Kevin Mesquita

2022

Contents

[User Guide: 2](#_Toc104133261)

[Interactive Mode: 2](#_Toc104133262)

[Silent Mode: 3](#_Toc104133263)

[Description of Classes: 3](#_Toc104133264)

[ListMenu: 3](#_Toc104133265)

[DSAGraph 3](#_Toc104133266)

[DSAVertex and DSAEdge 3](#_Toc104133267)

[DSALinkedList, DSAStack, DSAQueue 4](#_Toc104133268)

[Justification of Decisions: 4](#_Toc104133269)

**\*UML is in a png file under UML.png in parent directory**

**\*Tracability matrix file also in parent director under Tracability\_Matrix.xlsx**

# User Guide:

The first thing to not is that there are two ways in which whereNow.py can be run. The first one is interactive mode which only requires one command line argument and the second is silent mode, in which the user must provide map file, journey file, and save file for generated journey.

### Interactive Mode:

Interactive mode has a interface which the user can interact with. A menu with options numbered 1 to 11.

Option 1: Allows user to load graph data, this can either be a serialised file (pickled) or a txt file with the proper formatting

Option 2: Provides node operations, such as find, insert, delete and update. Note that when adding vertices, no edges will be connected to them.

Option 3: Edge Operations provides functionality to find, add, remove and update all edges in the graph. Note that there can be multiple edges in the same direction between two vertices, option 3 provides functionality choosing between the two after the user has selected the vertices to find edges in.

Option 4: Parameter Tweaks adjusts the journey planning algorithm by setting the user security level and whether the user wants to avoid or take stairs in the journey.

Option 5: Display Graph lets the user print out/save an adjacency matrix of the graph.

Option 6: Display world lets the user display a visual graph and an adjacency list. The visual graph doesn’t show all vertices between two nodes, each direction between two nodes will contain only a single directional arrow. Both options allow for the user to save, the visual graph has an option in the pop-up window while the matrix has an option in the menu.

Option 7: Allows the user to enter the start and end vertices. When inputting the start and destination names, the code automatically checks whether there is a node by the name.

Option 8: Generate Routes generates all possible routes between the set start node and end node (note you need to enter a start and end node first) while also satisfying the parameter conditions of security and avoiding/taking stairs. You need to select this before displaying routes.

Option 9: Displays all the generated routes in ranked/unraked order based on the distance required to get to the destination. It also has an option to save the journey to a file. Each journey takes up one line of the file with the best journey on top.

Option 10: Save network allows the user to save the network in later program readable text file or a serialised file.

Option 11: Exit

### Silent Mode:

Where ‘infile’ is the name of the map file to read in (graph data). ‘journey’ is the file which contains data in regards to start and end vertices and other parameters like security level and stairs. The final is savefile, the name of the file to save the generated journey to, this is saved in the directory of whereNow.py although you can save it elsewhere by using ../savefile to go up and down directories.

# Description of Classes:

The main classes in the program are ListMenu, DSAGraph, DSAVertex, DSAEdge, DSALinkedList, DSAQueue and DSAStack.

### ListMenu:

The list menu contains all the functionality of the menu in terms of file reading, writing, sorting menus and displaying and pathfinding. In retrospect, this class should have been broken down since into smaller classes since it covers way too much functionality making not very cohesive.

What I would have kept in the class was the menu and fileIO for reading and writing different data since they were all very specifically made to specific choices in the menu.

The sorting/pathfinding algorithm should have been its own separate thing building of the main algorithm in the graph class. And the network graph code should have been another separate class since it didn’t have anything in common with any other function.

### DSAGraph

DSAGraph also contained a lot of different functionality although most of it was specific to graphs and I would not have split it after the fact. The graph class had extra functions since I had also added a edge class and instead of accessing the class to use its setters and getters, I added the extra functionality to the Graph class.

The graph class allows for getting and setting DSAEdge and DSAVertex variables, some of these function had multiple functions for specific cases of using them, such as deleteEdge and deleteEdges, for deleting a specific edge to deleting all edges connected to a specific Vertex.

There was the main chunk of the pathfinding algorithm, which was going to be put into ListMenu but since the algorithm was an adaption of depth first search, I decided to keep it in the DSAGraph as that was the original structure in the pracs.

### DSAVertex and DSAEdge

These two are grouped together since they share a lot in common, both functions share the same function name getLabel() and setLabel() since the edge label is the name of the vertex it leads to which was vital in pathfinding algorithm since storing only edges meant you couldn’t differentiate some paths which had the same edge direction but slightly different other parameters.

DSAVertex stored its edge values in a linked list from which the edges contained the parameters and also the reference to the destination node.

### DSALinkedList, DSAStack, DSAQueue

All three of these were data structures made in the practicals and then used to make the program run smoother. They all contain specific function to how the data structures should work. There isn’t much to explain here.

# Justification of Decisions:

There are lots of a few things that need to be discussed for decisions/assumptions for the assignment

* Security Level, the security level is assumed that the person would need to have a security level of or higher than that of the path to travel on it. Originally, I thought that it was lower but that didn’t make sense.
* Journey Time, there was no journey time data since there was nothing changing in the route by the time of day. If the map changed based on the time of day, then there would have been a requirement for it.
* The stairs parameter was though of as avoid stairs, not should the person take the stairs, they give opposite results. The problem with avoid stairs is that there are no paths for the test cases since the first path is generally stairs and there are no journeys that don’t avoid stairs.
* The networkx graph only has a single directed arrow for all edges going from one node to another even if they have more than one edge. This is because I couldn’t figure out how to implement it.
* Parameter tweaks doesn’t have editing of all edge parameters. I thought that editing all parameters won’t do that much since they all have their relative distances and multiplying them by two is not going to make one path shorter than another.
* Was unsure how to test the interface methods, therefore didn’t test them as would somehow need to get the program to input instead of the user.
* A few of the graph and method functions only have tests for no data and valid data. This is due to lack of time.
* Insertion sort in Menu.py taken from prac 1
* Linked List class taken from prac 6
* DSAQueue and DSAStack classes taken from prac 3
* Used linked lists to store edges in vertices since they automatically resize.
* The sorted journey is stored in a 2d array since the size doesn’t change (the journey itself in index 2 of each row is linked list since that can change size)
* Depth first search pathfinding algorithm taken from this reference:

Elgabry, Omar. "Path Finding Algorithms", OmarElgabry's Blog, October, 11, 2016. <https://medium.com/omarelgabrys-blog/path-finding-algorithms-f65a8902eb40>

# Showcase:

For instructions on how the code works, refer to the User guide at the top of the document. There will be a scenario showcasing inputting a simple graph by hand and finding all shortest paths. The second reading in a file and setting the destination and the last is just a silent example.

## Scenario 1: simple input graph

A picture containing sky, indoor

Description automatically generated

Implementing this graph.

1. First start interactive program, ‘python3 whereNow.py -i'
2. Go to node operations (2) and using insert (2) add all nodes A,B,C,D,E,F
3. Going back to menu (5) and then edge operations then insert (2) add all edges with respective values as shown on graph, S = security 2 and B = stairs.
4. Go back to menu and then to (7) enter journey details, entering A as start and F as end
5. Now you can test which scenario is the best, without taking stairs and no security clearance, only 1 path A-B-E-F. Scenario tweaks (4) in parent menu allows you to change the parameters.
6. You can compare these outputs to the premade saved journey files
7. (if really need be, can load scenario1.txt in scenario 1 file to skip to step 4)

### Results:

#### O Security level, avoid Stairs:

Distance: 26 Journey: AB D:10 Sec:0 Stairs:False; E D:12 Sec:0 Stairs:False; F D:4 Sec:0 Stairs:False;

#### No Security level, take Stairs:

Distance: 21 Journey: A D D:3 Sec:0 Stairs:True; B D:2 Sec:0 Stairs:True; E D:12 Sec:0 Stairs:False; F D:4 Sec:0 Stairs:False;

Distance: 26 Journey: A B D:10 Sec:0 Stairs:False; E D:12 Sec:0 Stairs:False; F D:4 Sec:0 Stairs:False;

## Scenario 2:

For scenario 2, will be showing loading of the first map file and then outputting and saving a map of it.

Steps:

1. Starting the program python3 whereNow.py -i
2. (1) load input file
3. (1) Read File, input Map.txt
4. Back to main menu, go to parameter tweaks, change avoid stairs to False, security level to 2
5. Back to main menu, go to enter journey details, starting journey 314.221lab and ending journey 204.238.lab
6. Generate route and save to file

A quick way to reproduce this is to use silent mode and editing journey file to not avoid stairs:

python3 whereNow.py -s Map.txt Journey.txt scenario2Confirm.txt

## Scenario 3:

For scenario 3, doing same as scenario 3 except doing silent mode where reading in all the files data, except with Map2 and Journey2 files (Map2 is slightly edited to not avoid stairs)

Steps:

1. In Journey2.txt file if there is no hashtag character before “Avoid” add hashtag to it
2. Enter command into bash: python3 whereNow.py -s Map2.txt Journey2.txt scenario3.txt

The resulting journey file should be a list of 162 different generated journeys

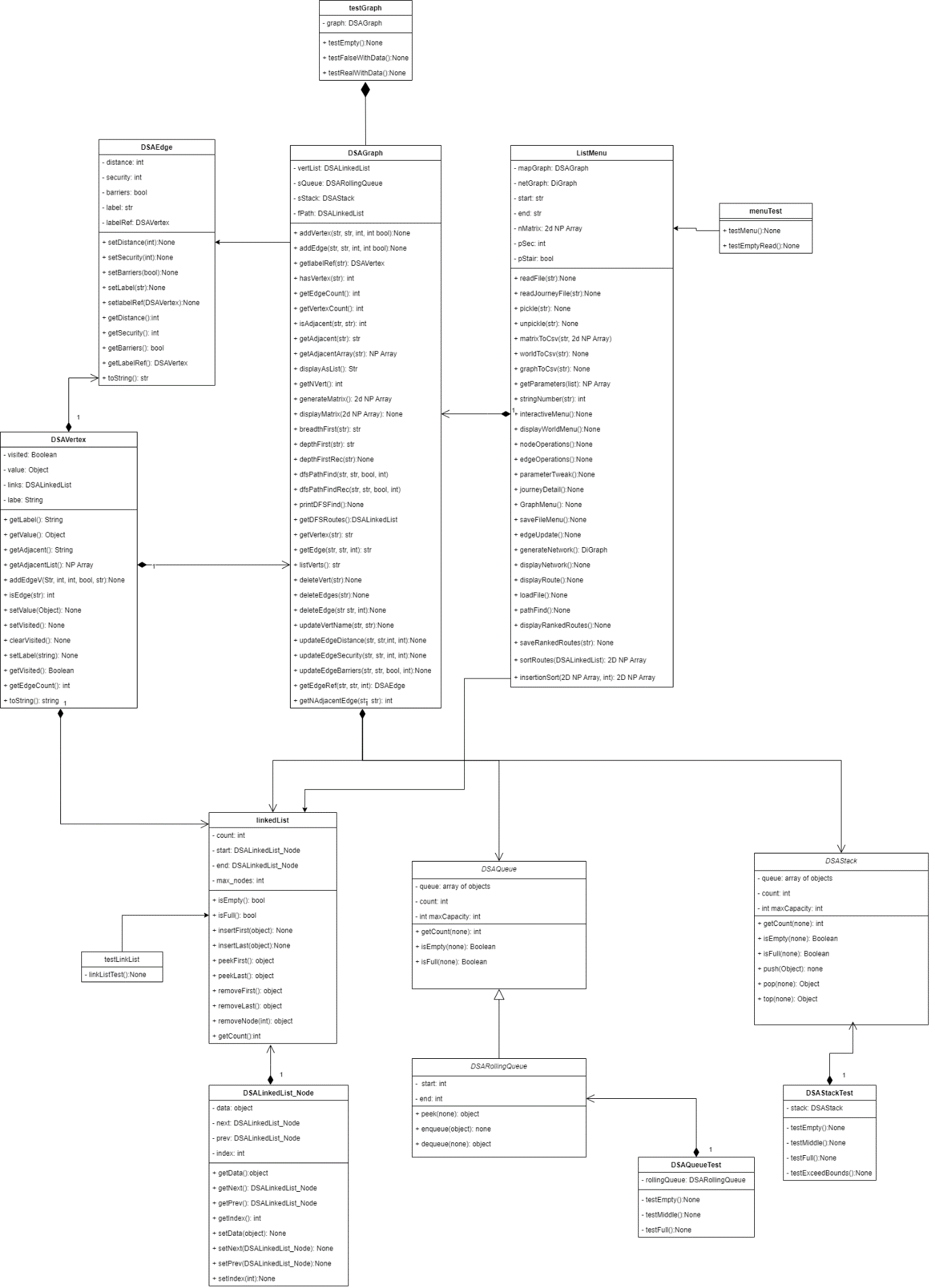
## NOTE: all scenario outputs are in folders and can be accessed if needed to compare outputs.

# Conclusion and Future Work

There are a few minor glitches with the pathfinding algorithm but can’t seem to be able to replicate it. If I were to keep working on this, I would fix this. The network graph made is a cut down version of what I wanted it to be, with it showing all edges. The problem is that network is doesn’t handle it by itself, I would need to change the edge radii for each edge to make them visible and that would have taken too much time to figure out.

I should have added read in journey file to the interactive menu but forgot and retrospect would have been very easy to add.

If I were to have done this again, I would have put proper testing for all my functions as I made them. Not just making crude tests, deleting those tests and moving on. The tests are what I feel like I struggled to get right and for the menu.py and graph.py code, were not very good at testing the functionality of the methods.



Table

Description automatically generated