Coursework 2

**“Deliveries”, version 1.0**

YIT19488399, Tony.

Delivery date: 09 March 2022

Value: 25% of total

**• A brief explanation of how you designed your navigation program. If you add a 4th algorithm, you should explain why you choose this algorithm.**

This program is designed using only functions with a driving block code at the end. The program called “Deliveries” is allowing the user first to choose the initial city and the destination city, and secondly choose an algorithm to travel between cities. There are 5 algorithms in total designed in different block functions. Plus, there is an option to see all the possible routes between the two cities in two different formats. The algorithms are DFS, BFS, A\*, Dijkstra and Bellman-Ford.

Graphical user interface, text, application

Description automatically generated

DFS, BFS and A\* are explained in lectures with Dr Gu and Dr Arturo. Dijkstra’s algorithm was explained by Dr Roberto in the previous semester this academic year. And Bellman-Ford algorithm is like Dijkstra, only allowing negative values of edges that do not exist in the present graph. A\* algorithm uses and heuristic value and this is the only difference in between with Dijkstra’s algorithm. BFS start from top to button and DFS from one side to the other side (left, right). I included some links commented in the source code in Jupyter Notebook.

I discovered while typing and doing my research about an interesting library called “Networkx”

[ <https://networkx.org/>, [https://networkx.org/documentation/stable/index.html#](https://networkx.org/documentation/stable/index.html) ]. I start developing my DFS and BFS algorithms and the next algorithms is a combination of long source code and build-in functions. The final block of code is just the build-in function. In the driving block, I combine both to verify that everything is the same, and this confirms that all is OK.

![Text

Description automatically generated]()

The beginning of my “Deliveries” program is to print a nice map of the UK. It has names and the distance between the cities. After I just developed code to confirm the correctness of my code and print a text with important and valued information.

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

Next blocks of code validate input from the user. Only allowed names for cities. After, I check if cities are the same, neighbours or not neighbours. If cities are not neighbours, function “choose\_searching\_algorithm()” ask for input an integer to choose an algorithm to travel between the cities in the UK map. And next blocks of code are the implementation of the searching algorithms presented at the beginning.

Resources used in this coursework are principally from the Internet:

<https://www.askpython.com/python-modules/networkx-package> to begin my graph design without knowledge about the “Networkx” library.

<https://www.annytab.com/a-star-search-algorithm-in-python/> to understand one of the searching algorithms (A\*).

<https://stackabuse.com/graph-theory-and-graph-related-algorithms-theory-and-implementation/> and in general <https://stackabuse.com/depth-first-search-dfs-in-python-theory-and-implementation/> helped me to understand and develop the rest of the searching algorithm in python.

![Graphical user interface, text, application

Description automatically generated]()

![Timeline

Description automatically generated]()

**• Result of a demo run of your navigation program. You should take screenshots of your results from Jupyter Notebooks and insert them into your report.**

This part of the coursework is included in the explanatory part of my program.

**• A brief reflection. It may include but is not limited to the discussions about the topics below:**

**o Have you met any difficulties/bugs during this coursework, and how did you tackle them?**

This coursework is easier than the previous, and the most difficult part was to design the A\* searching algorithm because the heuristic function was difficult to define. I solved the problem by adding a heuristic value of the final node to the result.

**o What did you think is the most difficult part of this coursework?**

The most difficult part of this coursework is designing the searching algorithms.

**o What have you learnt from investigating this problem?**

I learned a new library that is very useful (“Networkx”). And I discovered another similar library but did not use it, because it is not popular and require installation. This last library is much advanced, complicated, and difficult to use. And I learned some build-in functions for searching algorithms in a weighted bidirected or undirected graph.

Graphical user interface, text, website

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text

Description automatically generated

**o Which part(s) in your code design is your favourite and why?**

My favourite part of the code is the design of the map because it shows a graphical output. I am very familiar with text output on the screen but less familiar with graphics.

**o Which of the searching algorithms that you implemented is the best? Why?**

Bellman-Ford is my preferred algorithm because is a build-in function that requires only 1 line of code.

**o Any references you might have used to complete the task.**

All the references are included in the previous text.

END.