Faculty of Engineering & Technology
Electrical & Computer Engineering Department

ENCS3340

Project 1 Report

Search Algorithms for Route Navigation

Prepared by:
Roa Hanoun - 1190886
Donia Ziad – 1192454

Instructor: Dr. Adnan Yahya.

Section: 1

Date: 12/5/2022.

Abstract:

Searching is the most important task in programming. The search algorithm is an algorithm that searches for the desired element in huge data sets. Efficient searching is important to get the maximum throughput of the system, and ultimately, that system throughput provides maximum efficiency to our end user. In this project, we implemented various searching algorithms such as Breadth-first Search, Iterative Deepening Search, and A* Search on the Java compiler to find the optimal path between Palestinian cities. This is the shortest and most appropriate distance between cities for cars and walking.

Algorithm:

For our project, we used two text files: the first text file (towns), which contains 20 Palestinian cities and the x-axis and y-axis for each city, to obtain city locations on a map; and the second text file (road), which contains two cities in each row and the distance between them, to obtain the distances between the cities.

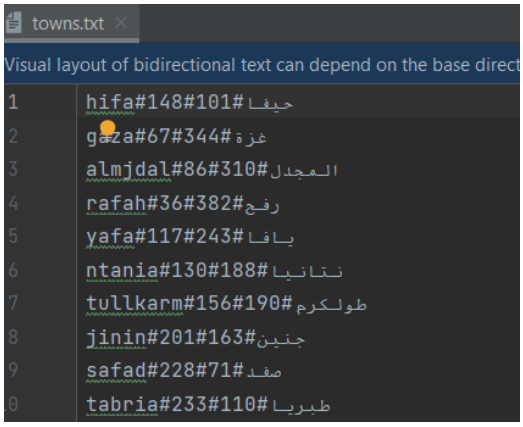

Figure 1: part of road file

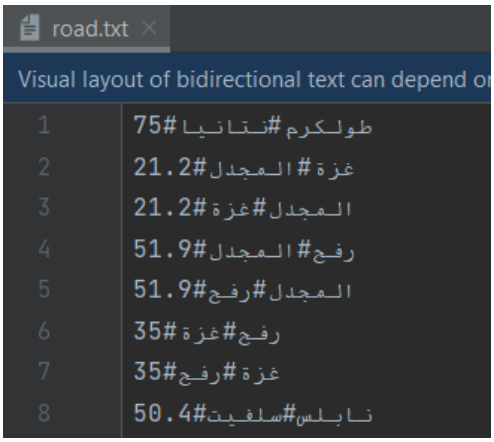

Figure 2: part of towns file

The text file (road) was used as an input for the program where the data was saved as an array of linked lists. I.e. a list for each city that contains the cities connected to it by routes, and the distance between them. Another text file (towns) was made that contains the locations of cities, Cities were saved as an object that has the following elements: ID, Arabic name, name, x-axis, and y-axis. where the data was saved as an array of type "city".

In order to get the distances between the cities, we used the website indicated below.

https://info.wafa.ps/ar_page.aspx?id=2794

We have implemented three algorithms to get the solution.

i. Breadth-First Algorithm:

In this algorithm, we use the priority queues data structure in this algorithm since it is based on the FIFO idea. We have a "Q" priority queue and a "V" array list of cities that have been visited. Begin by comparing the user's chosen start city to the desired outcome. If it is not the goal city, it is added to the "V" list, and all of its associated cities are added to the queue, except those that in the "V" or already in the queue.

Then take the first city in Q's cities and compare it to the goal. The procedure continues until the goal city is reached or the queue is empty, indicating that no path exists in our database connecting the start and goal cities.

We get the path by a predecessor, after each Failed comparison we add the city in the variable 'predecessor' and in the end, we can get the path by Back track.

ii. Iterative-Depth algorithm:

In this algorithm, the user chooses the city and the goal city. We create a loop to clear the list of visited cities, it does not find the target in the current depth and then moves to the next depth. However, this algorithm depends on the depth, we had to call the depth-first search algorithm whose principle of work is to track every depth to reach the target, so start with the first Depth and check every city if it is the target. If not, then puts it in the list of visited cities.

We get the path by a predecessor, after each Failed comparison we add the city in the variable 'predecessor' and in the end, we can get the path by Back track.

iii. A algorithm:*

This algorithm depends on the value of the heuristic and the real cost from the text file (road). We have two heuristics (h1, h2); h1: walking heuristic which is based on calculating The Straight Distance between two points (source city, goal city) and calculating the average between it and the real cost (g_n). h2: Arial heuristic which is based on calculating The Straight Distance between two points (source city, goal city).

A* algorithm in our project depends on FN that's equal to the sum of g_n and the heuristic h_n, which is equal to the sum of h1, and h2. We have the priority queue and visited list. First, we add the source city to Q and check if it is the goal or not. The process is based on adding the city and its children to Q, Then the algorithm takes the city that has the minimum FN and compares it with goal. The procedure continues until the goal city is reached or the queue is empty, indicating that no path exists in our database connecting the start and goal cities.

We get the path by a predecessor, after each Failed comparison we add the city in the variable 'predecessor' and in the end, we can get the path by Back track.

iv. User interface:

The main page for users from which to choose the source city, the goal, and the algorithm to find the optimal solution. As you can see below, the result is the optimal path, space (size of fringe in A* and level of depth in IDS), and the time (#visited cities).

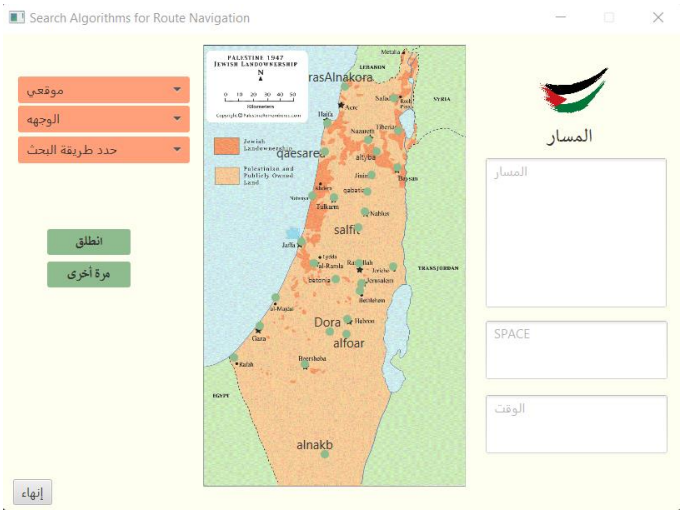

Figure 3: interface

here is the test for each algorithm. source --> haifa, goal --> alnakb.

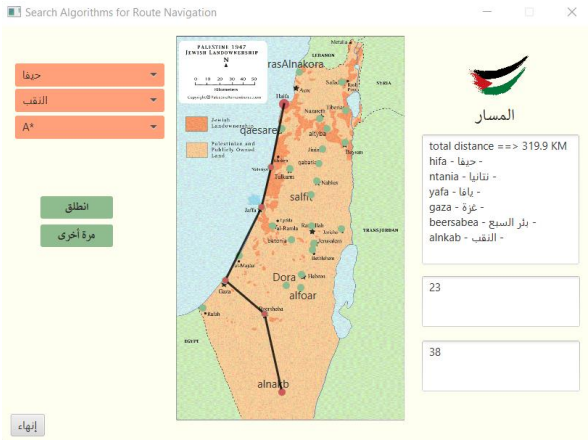

Figure 5:A algorithm*

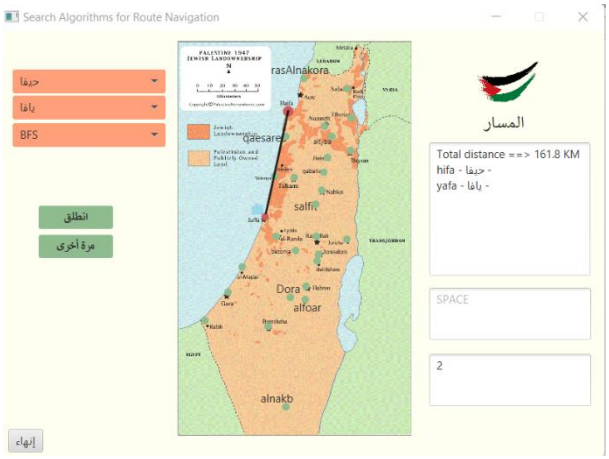

Figure 4:BFS algorthm

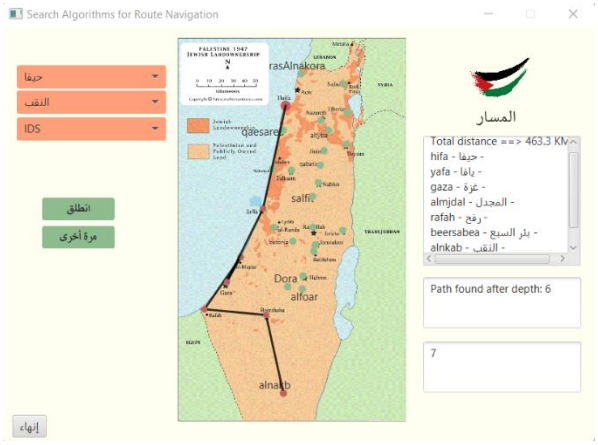

Figure 6:IDS algorithm