جامعة بيرزيت

BIRZEIT UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF COMPUTER ENGINEERING

Artificial Intelligence

ENCS3340

Project 1 Report
Search Algorithms for Route Navigation

Prepared by:

Tariq Odeh -1190699

Yousef Hammad - 1170625

Sec: 1

Instructor: Dr. Adnan Yahya

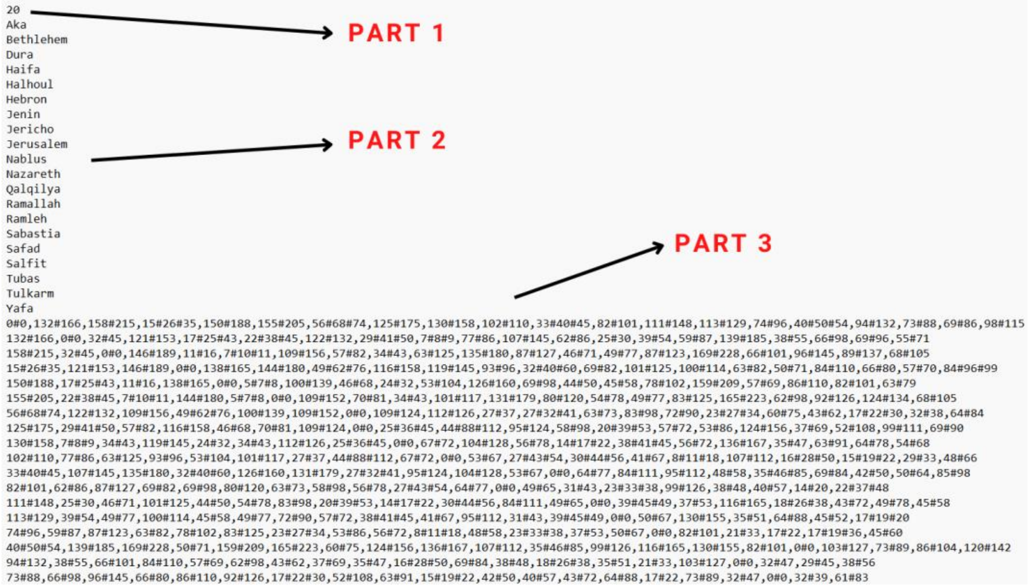
Date: 15th May 2022

Program Implementation

We used Java language to implement our program using Eclipse version (4.23.0). First, we created five classes: Driver, Graph, Record, MyPair and Tuple. A driver class will be used mainly to read input file, display the user interface so that the user can use the algorithms and print all the information like (Real cost, Heuristic, Cities...) smoothly, Graph class used to build the graph using the data that readed in driver class and implement the algorithms (BFS, DFS, UCS, Greedy and A*), Record class has three attributes: visited, path and expanded, all of them defined as ArrayList<Integer> and use to save the record for algorithms, in addition to one function to calculate the total cost for the path, MyPair class has two attributes: cityNum and distance and use in Greedy and UCS algorithms to know the father of each node and from which path it was reached and Tuple class has three attributes: City Num, sum Of Distances and heuristic and use in A* algorithm to know the father of each node and from which path it was reached.

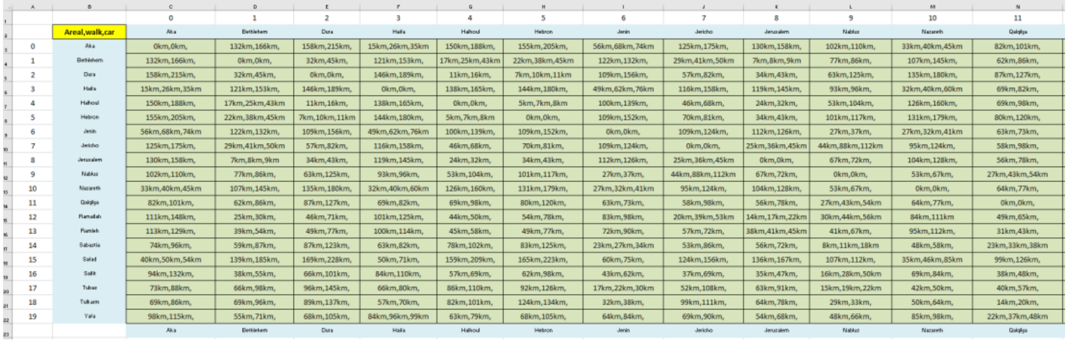
When the program is launched, the user's main interface appears, through which it can print all the information he needs, and it can also find the shortest route using 5 algorithms, which will print the shortest route, the cost, the cities visited in addition to expanded cities, and it can find the shortest route, whether he is driving or walking.

The input file is divided into three parts: The first part is the number of cities; second part is the names of cities and the third part is distances (Aerial#Walk#Driving). So that in the third part, the values of cities are separated by commas (,), and distance values for one city are separated by hashtags (#), and the values were represented in it as they are in the excel file shown in the figure 2.



```
20
Aka
Bethlehem
Dura
Haifa
Halhoul
Hebron
Jenin
Jericho
Jerusalem
Nablus
Nazareth
Qalqilya
Ramallah
Ramleh
Sabastia
Safad
Salfit
Tubas
Tulkarm
Yafa
0#0,132#166,158#215,15#26#35,150#188,155#205,56#68#74,125#175,130#158,102#110,33#40#45,82#101,111#148,113#129,74#96,40#50#54,94#132,73#88,69#86,98#115
132#166,0#0,32#45,121#153,17#25#43,22#38#45,122#132,29#41#50,7#8#9,77#86,107#145,62#86,25#30,39#54,59#87,139#185,38#55,66#98,69#96,55#71
158#215,32#45,0#0,146#189,11#16,7#10#11,109#156,57#82,34#43,63#125,135#180,87#127,46#71,49#77,87#123,169#228,66#101,96#145,89#137,68#105
15#26#35,121#153,146#189,0#0,138#165,144#180,49#62#76,116#158,119#145,93#96,32#40#60,69#82,101#125,100#114,63#82,50#71,84#110,66#80,57#70,84#96#99
150#188,17#25#43,11#16,138#165,0#0,5#7#8,100#139,46#68,24#32,53#104,126#160,69#98,44#50,45#58,78#102,159#209,57#69,86#110,82#101,63#79
155#205,22#38#45,7#10#11,144#180,5#7#8,0#0,109#152,70#81,34#43,101#117,131#179,80#120,54#78,49#77,83#125,165#223,62#98,92#126,124#134,68#105
56#68#74,122#132,109#156,49#62#76,100#139,109#152,0#0,109#124,112#126,27#37,27#32#41,63#73,83#98,72#90,23#27#34,60#75,43#62,17#22#30,32#38,64#84
125#175,29#41#50,57#82,116#158,46#68,70#81,109#124,0#0,25#36#45,44#88#112,95#124,58#98,20#39#53,57#72,53#86,124#156,37#69,52#108,99#111,69#90
130#158,7#8#9,34#43,119#145,24#32,34#43,112#126,25#36#45,0#0,67#72,104#128,56#78,14#17#22,38#41#45,56#72,136#167,35#47,63#91,64#78,54#68
102#110,77#86,63#125,93#96,53#104,101#117,27#37,44#88#112,67#72,0#0,53#67,27#43#54,30#44#56,41#67,8#11#18,107#112,16#28#50,15#19#22,29#33,48#66
33#40#45,107#145,135#180,32#40#60,126#160,131#179,27#32#41,95#124,104#128,53#67,0#0,64#77,84#111,95#112,48#58,35#46#85,69#84,42#50,50#64,85#98
82#101,62#86,87#127,69#82,69#98,80#120,63#73,58#98,56#78,27#43#54,64#77,0#0,49#65,31#43,23#33#38,99#126,38#48,40#57,14#20,22#37#48
111#148,25#30,46#71,101#125,44#50,54#78,83#98,20#39#53,14#17#22,30#44#56,84#111,49#65,0#0,39#45#49,37#53,116#165,18#26#38,43#72,49#78,45#58
113#129,39#54,49#77,100#114,45#58,49#77,72#90,57#72,38#41#45,41#67,95#112,31#43,39#45#49,0#0,50#67,130#155,35#51,64#88,45#52,17#19#20
74#96,59#87,87#123,63#82,78#102,83#125,23#27#34,53#86,56#72,8#11#18,48#58,23#33#38,37#53,50#67,0#0,82#101,21#33,17#22,17#19#36,45#60
40#50#54,139#185,169#228,50#71,159#209,165#223,60#75,124#156,136#167,107#112,35#46#85,99#126,116#165,130#155,82#101,0#0,103#127,73#89,86#104,120#142
94#132,38#55,66#101,84#110,57#69,62#98,43#62,37#69,35#47,16#28#50,69#84,38#48,18#26#38,35#51,21#33,103#127,0#0,32#47,29#45,38#56
73#88,66#98,96#145,66#80,86#110,92#126,17#22#30,52#108,63#91,15#19#22,42#50,40#57,43#72,64#88,17#22,73#89,32#47,0#0,32#39,61#83
```

Figure 1: input file.txt

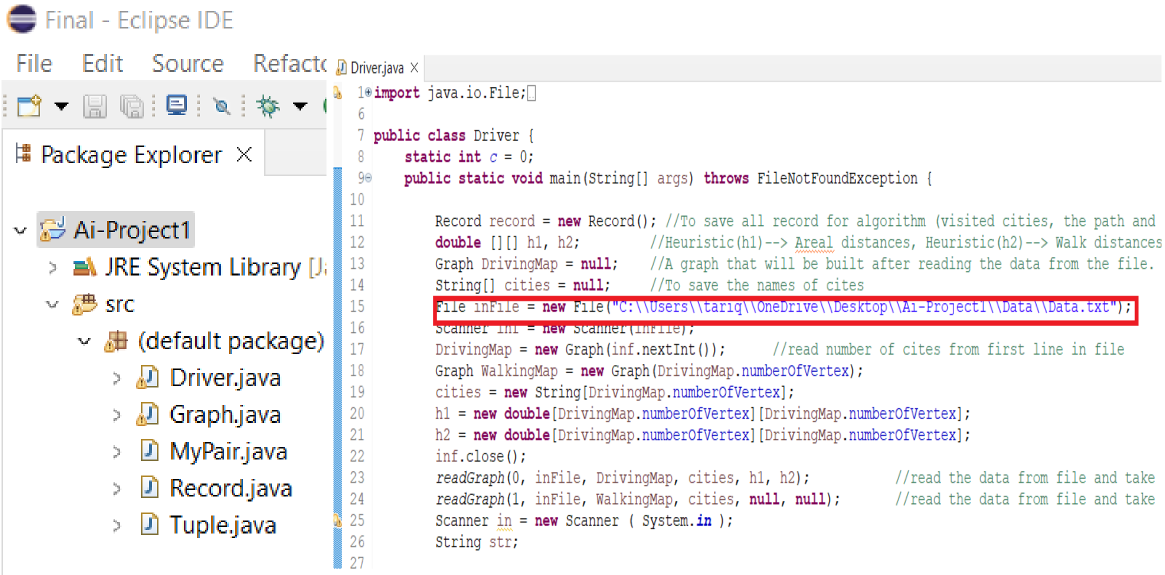


		0	1	2	3	4	5	6	7	8	9	10	11
	Areal,walk,car	Aka	Bethlehem	Dura	Haifa	Halhoul	Hebron	Jenin	Jericho	Jerusalem	Nablus	Nazareth	Qalqilya
0	Aka	0km,0km,	132km,166km,	158km,215km,	15km,26km,35km	150km,188km,	155km,205km,	56km,68km,74km	125km,175km,	130km,158km,	102km,110km,	33km,40km,45km	82km,101km,
1	Bethlehem	132km,166km,	0km,0km,	32km,45km,	121km,153km,	17km,25km,43km	22km,38km,45km	122km,132km,	29km,41km,50km	7km,8km,9km	77km,86km,	107km,145km,	62km,86km,
2	Dura	158km,215km,	32km,45km,	0km,0km,	146km,189km,	11km,16km,	7km,10km,11km	109km,156km,	57km,82km,	34km,43km,	63km,125km,	135km,180km,	87km,127km,
3	Haifa	15km,26km,35km	121km,153km,	146km,189km,	0km,0km,	138km,165km,	144km,180km,	49km,62km,76km	116km,158km,	119km,145km,	93km,96km,	32km,40km,60km	69km,82km,
4	Halhoul	150km,188km,	17km,25km,43km	11km,16km,	138km,165km,	0km,0km,	5km,7km,8km	100km,139km,	46km,68km,	24km,32km,	53km,104km,	126km,160km,	69km,98km,
5	Hebron	155km,205km,	22km,38km,45km	7km,10km,11km	144km,180km,	5km,7km,8km	0km,0km,	109km,152km,	70km,81km,	34km,43km,	101km,117km,	131km,179km,	80km,120km,
6	Jenin	56km,68km,74km	122km,132km,	109km,156km,	49km,62km,76km	100km,139km,	109km,152km,	0km,0km,	109km,124km,	112km,126km,	27km,37km,	27km,32km,41km	63km,73km,
7	Jericho	125km,175km,	29km,41km,50km	57km,82km,	116km,158km,	46km,68km,	70km,81km,	109km,124km,	0km,0km,	25km,36km,45km	44km,88km,112km	95km,124km,	58km,98km,
8	Jerusalem	130km,158km,	7km,8km,9km	34km,43km,	119km,145km,	24km,32km,	34km,43km,	112km,126km,	25km,36km,45km	0km,0km,	67km,72km,	104km,128km,	56km,78km,
9	Nablus	102km,110km,	77km,86km,	63km,125km,	93km,96km,	53km,104km,	101km,117km,	27km,37km,	44km,88km,112km	67km,72km,	0km,0km,	53km,67km,	27km,43km,54km
10	Nazareth	33km,40km,45km	107km,145km,	135km,180km,	32km,40km,60km	126km,160km,	131km,179km,	27km,32km,41km	95km,124km,	104km,128km,	53km,67km,	0km,0km,	64km,77km,
11	Qalqilya	82km,101km,	62km,86km,	87km,127km,	69km,82km,	69km,98km,	80km,120km,	63km,73km,	58km,98km,	56km,78km,	27km,43km,54km	64km,77km,	0km,0km,
12	Ramallah	111km,148km,	25km,30km,	46km,71km,	101km,125km,	44km,50km,	54km,78km,	83km,98km,	20km,39km,53km	14km,17km,22km	30km,44km,56km	84km,111km	49km,65km,
13	Ramleh	113km,129km,	39km,54km,	49km,77km,	100km,114km,	45km,58km,	49km,77km,	72km,90km,	57km,72km,	38km,41km,45km	41km,67km,	95km,112km,	31km,43km,
14	Sabastia	74km,96km,	59km,87km,	87km,123km,	63km,82km,	78km,102km,	83km,125km,	23km,27km,34km	53km,86km,	56km,72km,	8km,11km,18km	48km,58km,	23km,33km,38km
15	Safad	40km,50km,54km	139km,185km,	169km,228km,	50km,71km,	159km,209km,	165km,223km,	60km,75km,	124km,156km,	136km,167km,	107km,112km,	35km,46km,85km	99km,126km,
16	Salfit	94km,132km,	38km,55km,	66km,101km,	84km,110km,	57km,69km,	62km,98km,	43km,62km,	37km,69km,	35km,47km,	16km,28km,50km	69km,84km,	38km,48km,
17	Tubas	73km,88km,	66km,98km,	96km,145km,	66km,80km,	86km,110km,	92km,126km,	17km,22km,30km	52km,108km,	63km,91km,	15km,19km,22km	42km,50km,	40km,57km,
18	Tulkarm	69km,86km,	69km,96km,	89km,137km,	57km,70km,	82km,101km,	124km,134km,	32km,38km,	99km,111km,	64km,78km,	29km,33km,	50km,64km,	14km,20km,
19	Yafa	98km,115km,	55km,71km,	68km,105km,	84km,96km,99km	63km,79km,	68km,105km,	64km,84km,	69km,90km,	54km,68km,	48km,66km,	85km,98km,	22km,37km,48km
		Aka	Bethlehem	Dura	Haifa	Halhoul	Hebron	Jenin	Jericho	Jerusalem	Nablus	Nazareth	Qalqilya

Figure -2: Data

Program Runs and Example

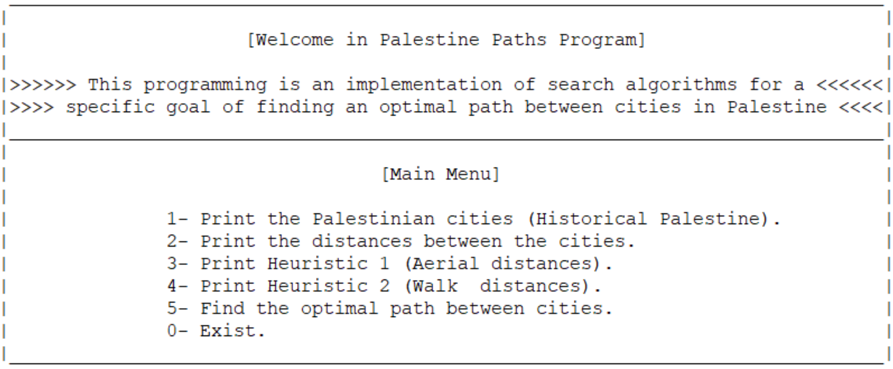
First, all classes must be added to the program, and make sure that the input file is in the order as shown in the figure 1 above, and verify the location of the file as shown in the figure below.



```java
import java.io.File;

public class Driver {
    static int c = 0;
    public static void main(String[] args) throws FileNotFoundException {

        Record record = new Record(); //To save all record for algorithm (visited cities, the path and
        double [][] h1, h2;         //Heuristic(h1)--> Areal distances, Heuristic(h2)--> Walk distances
        Graph DrivingMap = null;    //A graph that will be built after reading the data from the file.
        String[] cities = null;     //To save the names of cites
        File inFile = new File("C:\\Users\\tariq\\OneDrive\\Desktop\\Ai-Project1\\Data\\Data.txt");
        Scanner inf = new Scanner(inFile);
        DrivingMap = new Graph(inf.nextInt());      //read number of cites from first line in file
        Graph WalkingMap = new Graph(DrivingMap.numberOfVertex);
        cities = new String[DrivingMap.numberOfVertex];
        h1 = new double[DrivingMap.numberOfVertex][DrivingMap.numberOfVertex];
        h2 = new double[DrivingMap.numberOfVertex][DrivingMap.numberOfVertex];
        inf.close();
        readGraph(0, inFile, DrivingMap, cities, h1, h2);          //read the data from file and take
        readGraph(1, inFile, WalkingMap, cities, null, null);      //read the data from file and take
        Scanner in = new Scanner ( System.in );
        String str;

```

Figure 3: classes and location of the file input

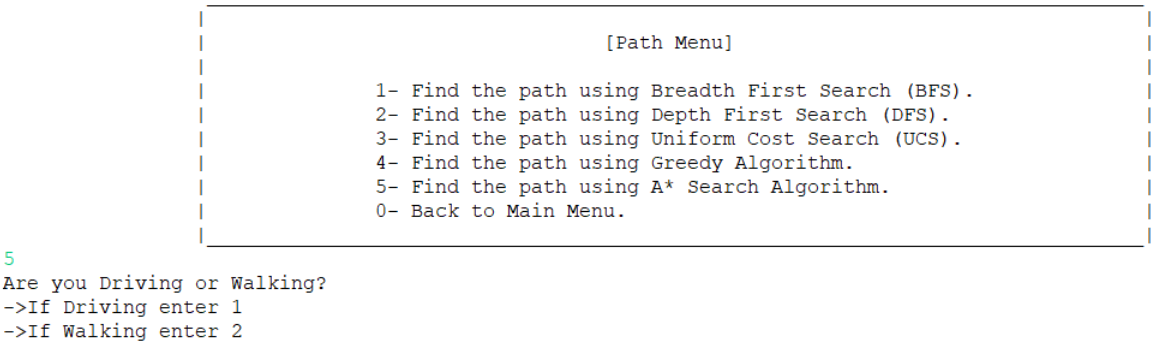
When we press run button, we will see the main menu, and we can print the city names, distances between them, heuristics and find the optimal path between cities.



```
 ______________________________________________________________________
|                                                                      |
|                     [Welcome in Palestine Paths Program]             |
|                                                                      |
|>>>>>> This programming is an implementation of search algorithms for a <<<<<<|
|>>>> specific goal of finding an optimal path between cities in Palestine <<<<|
|______________________________________________________________________|
|                                                                      |
|                              [Main Menu]                             |
|                                                                      |
|             1- Print the Palestinian cities (Historical Palestine).  |
|             2- Print the distances between the cities.               |
|             3- Print Heuristic 1 (Aerial distances).                 |
|             4- Print Heuristic 2 (Walk  distances).                  |
|             5- Find the optimal path between cities.                 |
|             0- Exist.                                                |
|______________________________________________________________________|
```

Figure 4: Main menu

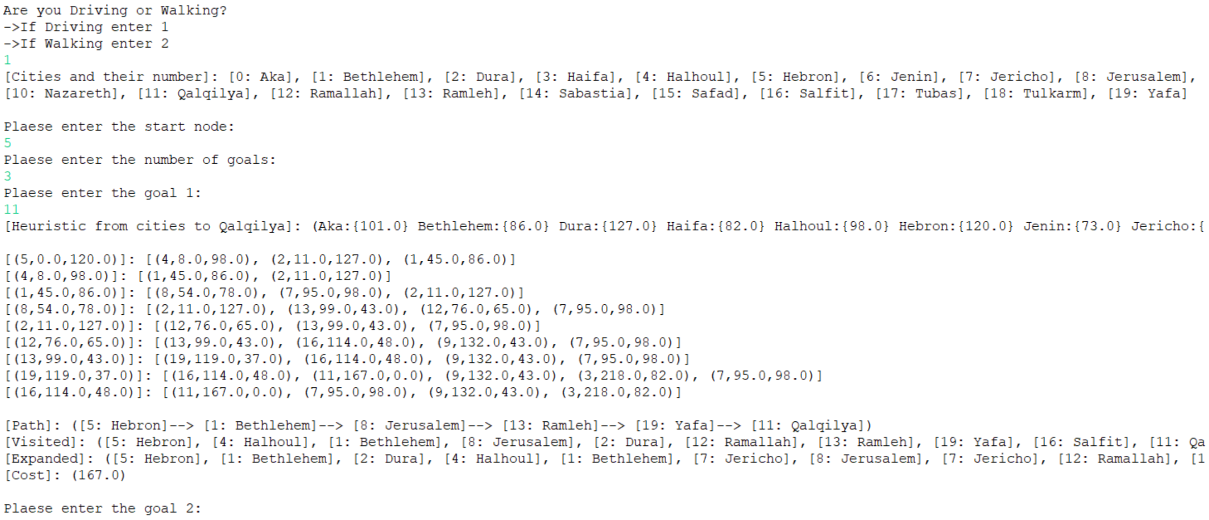
And when we choose the fifth option, we have the menu of the algorithms and we can find the shortest path using these five algorithms, and as an example we will choose A* algorithm to find optimal path between Source: Hebron [5] and Goals: (Qalqilya [11], Haifa [3] and Ramleh [13]).



```
    ______________________________________________________________
   |                                                              |
   |                         [Path Menu]                          |
   |                                                              |
   |       1- Find the path using Breadth First Search (BFS).     |
   |       2- Find the path using Depth First Search (DFS).       |
   |       3- Find the path using Uniform Cost Search (UCS).      |
   |       4- Find the path using Greedy Algorithm.               |
   |       5- Find the path using A* Search Algorithm.            |
   |       0- Back to Main Menu.                                  |
   |______________________________________________________________|
5
Are you Driving or Walking?
->If Driving enter 1
->If Walking enter 2
```

Figure 5: Path menu

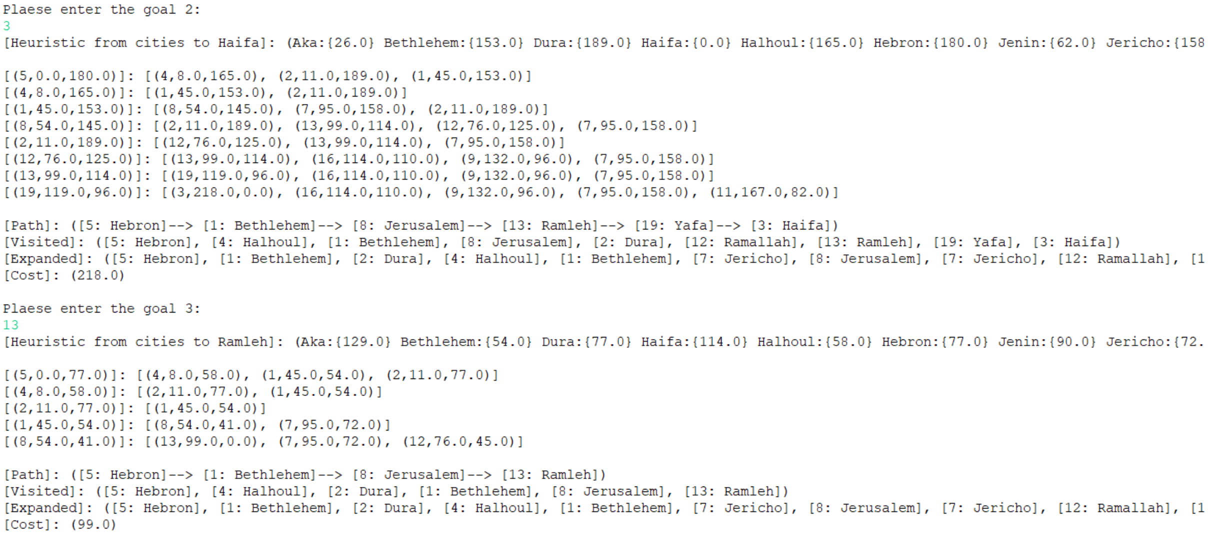
When choosing the algorithm, it asks if we driving or walking, and we will choose driving, it will print the names of the cities with their numbers and then ask for the starting city, and then the number of goals, and whenever we enter the goal, it prints the information as in the figure below.



```
Are you Driving or Walking?
->If Driving enter 1
->If Walking enter 2
1
[Cities and their number]: [0: Aka], [1: Bethlehem], [2: Dura], [3: Haifa], [4: Halhoul], [5: Hebron], [6: Jenin], [7: Jericho], [8: Jerusalem],
[10: Nazareth], [11: Qalqilya], [12: Ramallah], [13: Ramleh], [14: Sabastia], [15: Safad], [16: Salfit], [17: Tubas], [18: Tulkarm], [19: Yafa]

Plaese enter the start node:
5
Plaese enter the number of goals:
3
Plaese enter the goal 1:
11
[Heuristic from cities to Qalqilya]: (Aka:{101.0} Bethlehem:{86.0} Dura:{127.0} Haifa:{82.0} Halhoul:{98.0} Hebron:{120.0} Jenin:{73.0} Jericho:{

[(5,0.0,120.0)]: [(4,8.0,98.0), (2,11.0,127.0), (1,45.0,86.0)]
[(4,8.0,98.0)]: [(1,45.0,86.0), (2,11.0,127.0)]
[(1,45.0,86.0)]: [(8,54.0,78.0), (7,95.0,98.0), (2,11.0,127.0)]
[(8,54.0,78.0)]: [(2,11.0,127.0), (13,99.0,43.0), (12,76.0,65.0), (7,95.0,98.0)]
[(2,11.0,127.0)]: [(12,76.0,65.0), (13,99.0,43.0), (7,95.0,98.0)]
[(12,76.0,65.0)]: [(13,99.0,43.0), (16,114.0,48.0), (9,132.0,43.0), (7,95.0,98.0)]
[(13,99.0,43.0)]: [(19,119.0,37.0), (16,114.0,48.0), (9,132.0,43.0), (7,95.0,98.0)]
[(19,119.0,37.0)]: [(16,114.0,48.0), (11,167.0,0.0), (9,132.0,43.0), (3,218.0,82.0), (7,95.0,98.0)]
[(16,114.0,48.0)]: [(11,167.0,0.0), (7,95.0,98.0), (9,132.0,43.0), (3,218.0,82.0)]

[Path]: ([5: Hebron]--> [1: Bethlehem]--> [8: Jerusalem]--> [13: Ramleh]--> [19: Yafa]--> [11: Qalqilya])
[Visited]: ([5: Hebron], [4: Halhoul], [1: Bethlehem], [8: Jerusalem], [2: Dura], [12: Ramallah], [13: Ramleh], [19: Yafa], [16: Salfit], [11: Qa
[Expanded]: ([5: Hebron], [1: Bethlehem], [2: Dura], [4: Halhoul], [1: Bethlehem], [7: Jericho], [8: Jerusalem], [7: Jericho], [12: Ramallah], [1
[Cost]: (167.0)

Plaese enter the goal 2:
```

Figure 6: Result-1

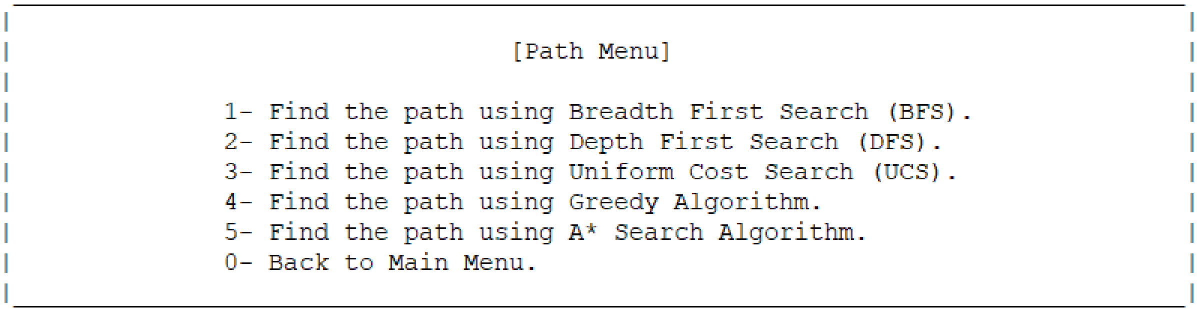
Every time he chooses the goal, the heuristic is printed for the goal, and the process for reaching the goal is shown, in addition to the path, visited cities, expanded cities and the cost.



```
Plaese enter the goal 2:
3
[Heuristic from cities to Haifa]: (Aka:{26.0} Bethlehem:{153.0} Dura:{189.0} Haifa:{0.0} Halhoul:{165.0} Hebron:{180.0} Jenin:{62.0} Jericho:{158

[(5,0.0,180.0)]: [(4,8.0,165.0), (2,11.0,189.0), (1,45.0,153.0)]
[(4,8.0,165.0)]: [(1,45.0,153.0), (2,11.0,189.0)]
[(1,45.0,153.0)]: [(8,54.0,145.0), (7,95.0,158.0), (2,11.0,189.0)]
[(8,54.0,145.0)]: [(2,11.0,189.0), (13,99.0,114.0), (12,76.0,125.0), (7,95.0,158.0)]
[(2,11.0,189.0)]: [(12,76.0,125.0), (13,99.0,114.0), (7,95.0,158.0)]
[(12,76.0,125.0)]: [(13,99.0,114.0), (16,114.0,110.0), (9,132.0,96.0), (7,95.0,158.0)]
[(13,99.0,114.0)]: [(19,119.0,96.0), (16,114.0,110.0), (9,132.0,96.0), (7,95.0,158.0)]
[(19,119.0,96.0)]: [(3,218.0,0.0), (16,114.0,110.0), (9,132.0,96.0), (7,95.0,158.0), (11,167.0,82.0)]

[Path]: ([5: Hebron]--> [1: Bethlehem]--> [8: Jerusalem]--> [13: Ramleh]--> [19: Yafa]--> [3: Haifa])
[Visited]: ([5: Hebron], [4: Halhoul], [1: Bethlehem], [8: Jerusalem], [2: Dura], [12: Ramallah], [13: Ramleh], [19: Yafa], [3: Haifa])
[Expanded]: ([5: Hebron], [1: Bethlehem], [2: Dura], [4: Halhoul], [1: Bethlehem], [7: Jericho], [8: Jerusalem], [7: Jericho], [12: Ramallah], [1
[Cost]: (218.0)

Plaese enter the goal 3:
13
[Heuristic from cities to Ramleh]: (Aka:{129.0} Bethlehem:{54.0} Dura:{77.0} Haifa:{114.0} Halhoul:{58.0} Hebron:{77.0} Jenin:{90.0} Jericho:{72.

[(5,0.0,77.0)]: [(4,8.0,58.0), (1,45.0,54.0), (2,11.0,77.0)]
[(4,8.0,58.0)]: [(2,11.0,77.0), (1,45.0,54.0)]
[(2,11.0,77.0)]: [(1,45.0,54.0)]
[(1,45.0,54.0)]: [(8,54.0,41.0), (7,95.0,72.0)]
[(8,54.0,41.0)]: [(13,99.0,0.0), (7,95.0,72.0), (12,76.0,45.0)]

[Path]: ([5: Hebron]--> [1: Bethlehem]--> [8: Jerusalem]--> [13: Ramleh])
[Visited]: ([5: Hebron], [4: Halhoul], [2: Dura], [1: Bethlehem], [8: Jerusalem], [13: Ramleh])
[Expanded]: ([5: Hebron], [1: Bethlehem], [2: Dura], [4: Halhoul], [1: Bethlehem], [7: Jericho], [8: Jerusalem], [7: Jericho], [12: Ramallah], [1
[Cost]: (99.0)
```

Figure 7: Result-2

When the algorithm is finished, it returns to path menu and we can try any existing algorithm in the same way, and we can go back to the main menu and exit the program.



```
 _______________________________________________________________
|                                                               |
|                          [Path Menu]                          |
|                                                               |
|          1- Find the path using Breadth First Search (BFS).   |
|          2- Find the path using Depth First Search (DFS).     |
|          3- Find the path using Uniform Cost Search (UCS).    |
|          4- Find the path using Greedy Algorithm.             |
|          5- Find the path using A* Search Algorithm.          |
|          0- Back to Main Menu.                                |
|_______________________________________________________________|
```

Figure 8: path menu after finished appear the result

Bonus Elements

• Interface.

We have worked on creating a user interface that makes it easier for the user to use the programs, keep using menu and in addition to printing the results in a manner that is easy to read and track.

• More factors for optimization.

BFS: to optimize the BFS algorithm time and space once the node is expanded it will note be expanded again since any further findings will be either on the same level as the previously expanded or even further down the tree.

Greedy: to optimize the greedy search for the minimum heuristic node which will cost O(n), we used a priority queue which is a heap in java to make the minimum heuristic node always on the top and that costs O(logn).

Uniform cost: to optimize it we used a priority queue for faster search for the minimum cost node O(logn).

A*: same as the greedy and the uniform cost we also used a priority queue to find the least sum of cost + heuristic of expanded nodes and it also costs O(logn) to find the minimum or even update it.

• Extra algorithms you feel of interest.

We implemented five algorithms, two of them is extra algorithms (Uniform Cost and DFS).

• Selectable goals and more options.

We have implemented all the algorithm to be Selectable goals and in addition to all algorithms can be found the path in driving and walking option.