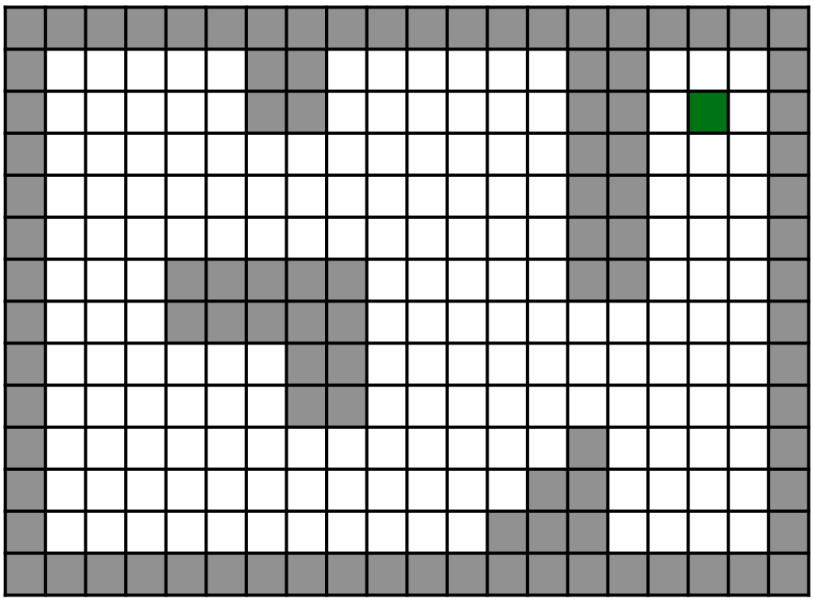
Biomedical Data Analytics: Midterm

Environment:

The problem consists in finding the optimal trajectory towards the goal in a finite 2D environment that is closed and contains some obstacles (in grey) as shown in the figure:



The size of the environment is 20x14. The goal position is marked in green. The environment will be represented as a matrix in the following way:

```
map=[
     1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1;
     1  0  0  0  0  0  1  1  0  0  0  0  0  0  1  1  0  0  0  1;
     1  0  0  0  0  0  1  1  0  0  0  0  0  0  1  1  0  2  0  1;
     1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  1;
     1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  1;
     1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  1;
     1  0  0  0  1  1  1  1  1  0  0  0  0  0  1  1  0  0  0  1;
     1  0  0  0  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  1;
     1  0  0  0  0  0  0  1  1  0  0  0  0  0  0  0  0  0  0  1;
     1  0  0  0  0  0  0  1  1  0  0  0  0  0  0  0  0  0  0  1;
     1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  1;
     1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  1;
     1  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  0  0  0  1;
     1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1;];
```

Note that the obstacle cells are marked with 1, free space with 0, and the goal position with a 2.

Algorithm:

Program in Python the wavefront planner algorithm to compute the optimal path toward the goal. Use 8-point connectivity.

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	21	20	19	18	18	1	1	14	14	14	14	14	14	1	1	3	3	3	1
1	21	20	19	18	17	1	1	14	13	13	13	13	13	1	1	3	2	3	1
1	21	20	19	18	17	16	15	14	13	12	12	12	12	1	1	3	3	3	1
1	21	20	19	18	17	16	15	14	13	12	11	11	11	1	1	4	4	4	1
1	21	20	19	18	17	16	15	14	13	12	11	10	10	1	1	5	5	5	1
1	21	20	19	1	1	1	1	1	13	12	11	10	9	1	1	6	6	6	1
1	21	20	19	1	1	1	1	1	13	12	11	10	9	8	7	7	7	7	1
1	21	20	19	18	17	17	1	1	13	12	11	10	9	8	8	8	8	8	1
1	21	20	19	18	17	16	1	1	13	12	11	10	9	9	9	9	9	9	1
1	21	20	19	18	17	16	15	14	13	12	11	10	10	1	10	10	10	10	1
1	21	20	19	18	17	16	15	14	13	12	11	11	1	1	11	11	11	11	1
1	21	20	19	18	17	16	15	14	13	12	12	1	1	1	12	12	12	12	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

You must program a Python function with the following input and output parameters:

[value_map, trajectory]=planner(map, start_row, start_column)

Your program, when called with the previous map and the following parameters should generate:

[value_map, trajectory]= planner(map, 13, 2)

```
value_map =

     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1
     1    21    20    19    18    18     1     1    14    14    14    14    14    14     1     1     3     3     3     1
     1    21    20    19    18    17     1     1    14    13    13    13    13    13     1     1     3     2     3     1
     1    21    20    19    18    17    16    15    14    13    12    12    12    12     1     1     3     3     3     1
     1    21    20    19    18    17    16    15    14    13    12    11    11    11     1     1     4     4     4     1
     1    21    20    19    18    17    16    15    14    13    12    11    10    10     1     1     5     5     5     1
     1    21    20    19     1     1     1     1     1    13    12    11    10     9     1     1     6     6     6     1
     1    21    20    19     1     1     1     1     1    13    12    11    10     9     8     7     7     7     7     1
     1    21    20    19    18    17    17     1     1    13    12    11    10     9     8     8     8     8     8     1
     1    21    20    19    18    17    16     1     1    13    12    11    10     9     9     9     9     9     9     1
     1    21    20    19    18    17    16    15    14    13    12    11    10    10     1    10    10    10    10     1
     1    21    20    19    18    17    16    15    14    13    12    11    11     1     1    11    11    11    11     1
     1    21    20    19    18    17    16    15    14    13    12    12     1     1     1    12    12    12    12     1
     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1
```

```
trajectory =

    13     2
    13     3
    13     4
    13     5
    13     6
    13     7
    13     8
    13     9
    13    10
    13    11
    12    12
    11    13
    10    14
     9    15
     8    16
     7    17
     6    17
     5    17
     4    17
     3    18
```

It is very important that you generate the planner map (value_map) and the trajectory using EXACTLY the same format you can see in the previous example. Your Python file will be evaluated exclusively based on the results it generates which must follow the previous format. The algorithms will be tested in different environments.

Note that several trajectories can be obtained, all of them being optimal. Your code must generate one of the optimal trajectories considering 8-point connectivity. So, the previous trajectory is only one example of several optimal trajectories you could generate.

Note finally that the wavefront planner does not distinguish between up, down, left, right movements, and diagonal movements. Since we know that diagonal movements will be longer, you can modify your code to prioritize up, down, left, and right movements if possible.

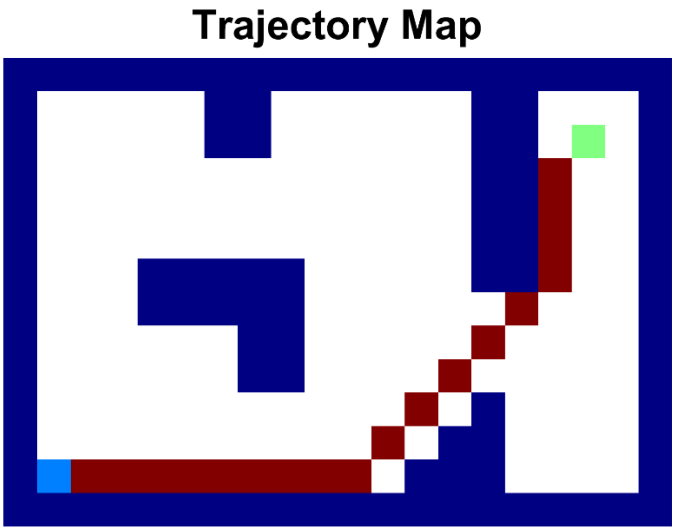
Given the start point and the goal point, the algorithm is implemented as follows:

1. The goal point is initialized with 2
2. The neighboring cells to the goal that are not obstacles are assigned the value of the goal point + 1
3. Update the goal value, that is, goal = goal + 1 then repeat steps 1 & 2 until all the spaces are filled.
4. The trajectory (shortest distance to the goal) is easily found by taking differences with all neighboring cells, that is, the minimum of the neighboring cells.

** For your convenience, consider your priority such as :[upper, Right, Lower, Left, Upper right, lower right, Lower Left, Upper Left].

Representation:

In order to correctly show the trajectory that your algorithm is generating, you can use the Python plotting capabilities to automatically plot the map and the trajectory, such that:



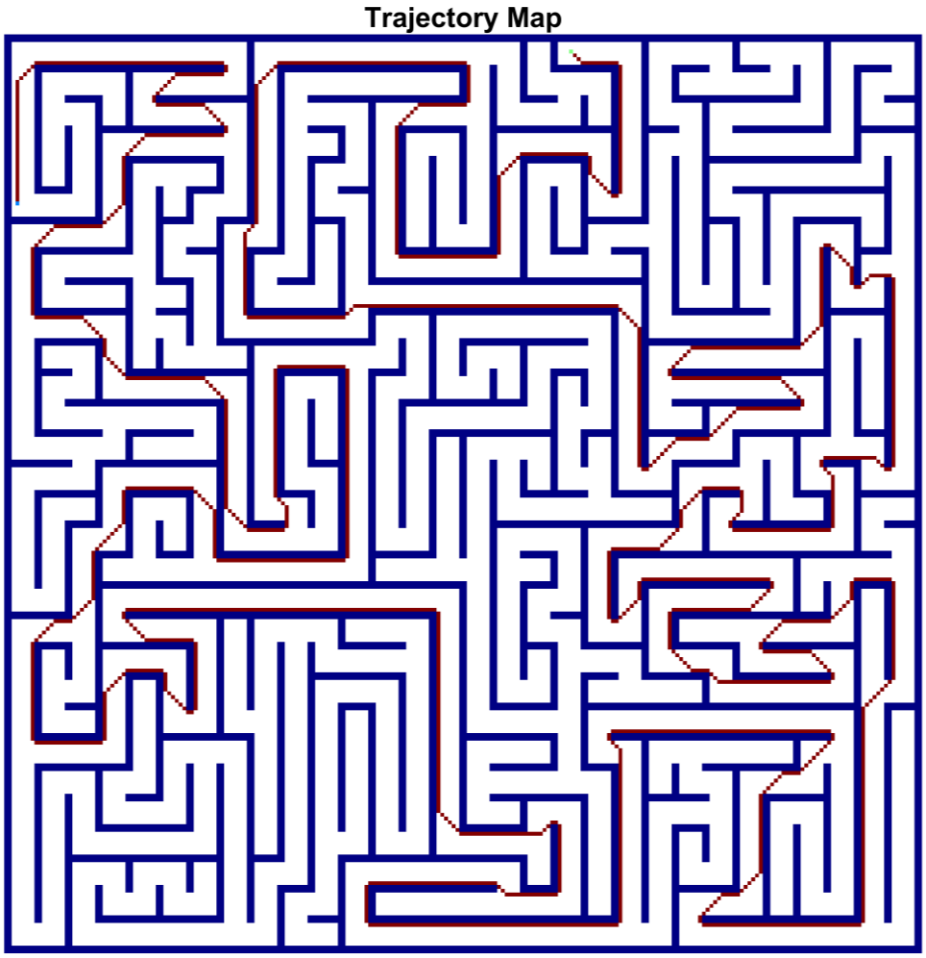
Use your own map and trajectory representation, and include these figures in your report.

Big Maze Environment:

Load the map from the maze.mat file and run your algorithm and representation utility. Use as the initial point the one indicated in the next call:

```
[value_map, trajectory] = planner(map, 45, 4)
```

The map and one of the optimal trajectories are:



Submission:

Submit a report in pdf and the Python file. Explain in detail, in the report, the work done in all the sections. Also, explain the problems you found. You might want to test your algorithm in other environments. NOTE that only these 2 files are required and will be evaluated. Do not send more files.

- **Only two students are allowed per group.**
- **Any form of plagiarism or using someone else's work is not permitted under any circumstances.**
- **If plagiarism is detected, the group will be considered to have committed academic misconduct and will receive a zero.**

Relevant Material

https://web.ist.utl.pt/~margaridaacferreira/RoboticsFun/Wavefront/index.html

https://ashesirobotics.wordpress.com/2012/03/29/task4-path-planning/