Coding Task: 1

Rules:

1. Use **Python 3.7** or higher in **Google Colab**. It comes with pre-installed packages like numpy, time, etc.
2. Submission should be a single **.zip** file (no other extensions like .rar, .gz) Keep this in mind.
3. Submission should contain
    a. Code files in **.ipynb** format. Use separate **.ipynb** files for separate questions. **DO NOT** use .py files. We will test your code in Google Colab.
    b. README file describing how to run the code. TAs will run your code to see if they are really working.
    c. A write-up in **.pdf** format. It should contain answers to the questions with appropriate screenshots or test results from the output of your code. Treat this as a mini-report that TAs will evaluate after your code successfully runs.
4. **DO NOT** use built-in functions for cases where you have been asked to implement that exact same functionality. You can use built-in functions if you want to test whether your designed function produces the correct output.

I. Naive Robot Navigation Problem:

Design a program that uses the **BFS (Breadth-First Search)** algorithm to find any of the shortest navigation routes for a line-following robot tasked with navigating a grid maze by following only white tiles. Traversal to adjacent tiles takes the same time regardless of the adjacent tile's direction. The program takes the following inputs **(Do not hardcode any of the following parameters)**:

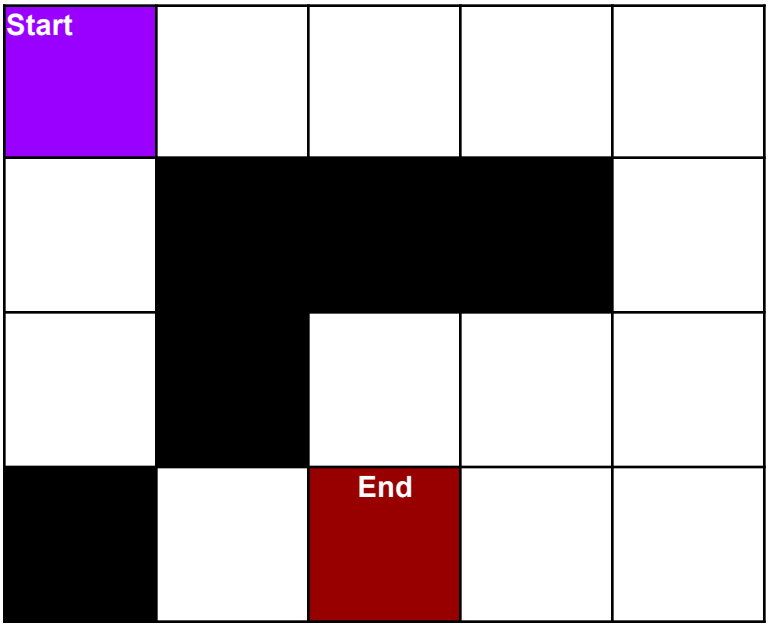
1. A maze configuration, ***maze***, represented as an MxN 2D list, where M is the number of rows and N is the number of columns in the maze. Each element of the list represents a tile in the maze, with '1' indicating a white tile and '0' indicating an obstacle or black tile.
2. The starting coordinate, ***start***, a two-element tuple (row_start_index, col_start_index) where the robot begins its journey within the maze.
3. The end coordinate, ***end***, a two-element tuple (row_end_index, col_end_index) representing the desired destination point.
4. Legal moves boolean, ***all_direction***. A value **False** indicates that the robot can move only in four directions (up, down, left, right), whereas **True** indicates that the robot can move in all directions (up, down, left, right, up-left, up-right, down-left, down-right).

The program should provide the following output as a tuple with two elements:

1. A boolean value indicates whether the endpoint is reachable from the starting point (**True** if reachable, **False** if unreachable).
2. A list representing the navigation plan that the robot should follow successively to traverse the maze. This list should contain a sequence of coordinates representing the path from the starting point to the endpoint, considering the shortest possible route.

Note: Your program should work for a maze of any size and any start and end coordinates within maze. We'll test your program's output on a random maze configuration

Following is an illustrative example of maze configuration for clarification of problem statement




```
maze = [[1, 1, 1, 1, 1],
        [1, 0, 0, 0, 1],
        [1, 0, 1, 1, 1],
        [0, 1, 1, 1, 1]]


start = (0,0)
```

End = (3,2)

all_direction=True
Output:
True,
[(0,0),(1,0),(2,0),(3,1),(3,2)]

all_direction=False
Output:
True,
[(0,0),(0,1),(0,2),(0,3),(0,4),(1,4),(2,4),(3,4),(3,3),(3,2)]

Or

True,
[(0,0),(0,1),(0,2),(0,3),(0,4),(1,4),(2,4),(2,3),(2,2),(3,2)]

II. Smarter Robot Navigation Problem:

In the above problem statement, The navigation route does not consider loss of speed due to changing directions. Consider the following two routes, Route2 and Route3 in the example above

Route2:[(0,0),(0,1),(0,2),(0,3),(0,4),(1,4),(2,4),(3,4),(3,3),(3,2)]

Route3:[(0,0),(0,1),(0,2),(0,3),(0,4),(1,4),(2,4),(2,3),(2,2),(3,2)]

Even though the two routes, Route2 and Route3, have the same distance, Route2 has fewer turns (Two turns vs. Three turns), and therefore, the robot can navigate Route2 faster. Assuming that the robot has two wheels, symmetric front and rear sensor setup(i.e., Front side and rear side are functionally symmetric), and thus needs to take either 0, 45, or 90 degrees turn in any given situation. Further assume following “change of directions” overhead costs

1. A change of 0 degrees adds no extra time.
2. A change of 45 degrees adds an overhead of 0.5 tile traversal time.
3. A change of 90 degrees adds an overhead of 1 tile traversal time.

With these costs (Red **1** indicates turn overheads)
Route2 will cost [1+1+1+1+**1**+1+1+1+**1**+1+1]=11
Route3 will cost [1+1+1+1+**1**+1+1+**1**+1+1+**1**+1]=12
Hence, with **all_direction=False,** Route2 is the most optimal route.

However with **all_directions=True**, then Route1,

`Route1: [(0,0),(1,0),(2,0),(3,1),(3,2)]`

Cost of Route1: [1+1+**0.5**+1+**0.5**+1] =5
Making it the most optimal route.

Write a program considering the above “change of direction” overheads to give the shortest time route using **Dijkstras’s shortest path algorithm**.