RBE 550 - Basic Search Algorithms Implementation

Overview

In this assignment, you are going to implement **BFS**, **DFS**, **Dijkstra** and **A*** algorithms with Python 3. These are the basic algorithms for discrete planning, which also share a lot of similarities. This template is provided to you as a starting point. After you finish coding, you would be able to create your own map to test different algorithms, and visualize the path found by them.

Files included:

- **search.py** is the file where you will implement your algorithms. As we will use a grader code to help us grade your assignments more efficiently. Please do not change the existing functions names.
- **main.py** is the scrip that provides helper functions that load the map from csv files and visualize the map and path. You are not required to modify anything but you are encouraged to read and understand the code.
- **map.csv** is the map file you could modify to create your own map.
- **test_map.csv** restores a test map for doc test purpose only. Do not modify this file.

Please finish reading all the instructions and rubrics below before starting actual coding.

Get Started

Before starting any coding, please run the code first:

```
python search.py
```

When running **search.py** as a main function, it will run a doc test for all the algorithms. It loads **test_map.csv** as the map for testing.

As you haven't written anything yet, you would see you fail all the doc tests. After implementing each algorithm, you should run this file again and make sure to pass the doc tests (you will see nothing if you pass the test).

But please be noted that, passing doc tests does not necessarily mean that your algorithm is done without any problems. It just shows that your algorithms are working in this simple **test_map.csv** with its given start and end position. (It may still fail, for example, if you change the goal to an obstacle.)

For visualization, please run:

```
python main.py
```

There should be 4 maps shown representing the results of 4 algorithms. As said before, there would be no path shown in the graph as you haven't implemented anything yet. The **main.py** loads the map file **map.csv**, which you are free to modify to create your own map.

More details

- Please first read the algorithm description in **seach.py** and make sure you understand the input/arguments and required output/return of the algorithms.
- Keep in mind that, the coordinate system used here is **[row, col]**, which could be different from [x, y] in Cartesian coordinates.
- For Dijkstra and A*, the cost to move each step is set to be 1. The heuristic for each node is its Manhattan distance to the goal.
- When you explore the nearby nodes in the map, please follow this order **"right, down, left, up"**, which means "[0, +1], [+1, 0], [0, -1], [-1, 0]" in coordinates. There is nothing wrong using other exploring orders. It is just that the test result was gotten by algorithms using this order. A different order may result in different results, which could let you fail the test and the grader code.
- Also, for the output of the function, steps and path should include and count the first / start position and the goal / end position. (We assume that the start and goal won't be the same.)

Until now, I hope you have a basic understanding of the template code and the requirements. After you go through the Rubrics below, you may start coding!

Rubrics

- (0.5 pts) A clear structure without duplicating your code.

  Please figure out the similarity of these four algorithms. As they are somewhat similar, if you implement them separately, you would end up writing a lot of duplicated codes or copying codes here and there. First, it's not a good practice to do so. It also does not reflect your deeper understanding of these algorithms. Please find out the similarity and construct your code in a way that you would not duplicate the codes.

- (1 pts) Your algorithms pass the basic doc test.

  After implementation, as mentioned before, you could run the doc test by:

  `python search.py`

  If you see nothing, it means you pass the simple doc test.

- (4 pts) Your algorithms pass all the tests for grading.

  We will use a grader code with other test cases for grading. Please think of different possible test cases yourself, modify **test.csv** to create your own map and check the result. It's a good practice to think of boundary cases, a map that does not have a valid path, for instance.

- (1.5 pts) Documentation

  Besides the code, you should also include a documentation with the following content:

  - Code and algorithm explanation

    You should briefly explain how these four algorithms work. How they are different and similar. A pseudocode could be helpful for explanation.

  - Test example, test result and explanation

    Run your code with `python main.py`. You could use the map **map.csv** I provided, but I encourage you to create your own map. Run the test and see the paths you found and the steps/time it takes for these algorithms to find them. Try to briefly explain the reason why different algorithm gives different or same result.

  - Reference paper and resources if any

Include the documentation as a pdf file, or if you prefer, a md file.