SI100B, Spring 2018
Homework 4: Dijkstra's for path planning

Due date: 2018/04/25 23:59:59

1 Goals

The goals of this homework are:

- Exercise understanding of graph search algorithms.
- Exercise your python programming skills.
- Exercise good programming practice by adhering to good coding rules.

2 Policy on plagiarism

This is an individual homework. While you may discuss the ideas and algorithms of this homework with others, at no time should you read, possess, or submit the solution source code of any other person (including people outside this course), or allow anyone else to read or possess your source code. We will detect plagiarism using automated tools and will prosecute all violations to the fullest extent of the university regulations, including failing this course, academic probation, and expulsion from the university.

3 Dijkstra's for path planning

3.1 Hand out

- Example input for Dijkstra's: map.txt.
- Example output Dijkstra's: dijkstra.txt.
- Register on our online judge system (http://10.19.125.79/, different from HW1 and HW2!), submit your source code before the due date. Due to the computing resource limits , please run and test your code on your local machine before submit it to the online judge system.

3.2 Description

Input:

The input to the program is a text file. It contains comments, a start position, a goal position and a 2D map. Comment lines begin with a dollar character ($) and should be ignored by your program.

The first non-comment line of that text file encodes the start position of the robot and the second non-comment line is the goal position. Those are two integer numbers separated by spaces. They encode x, y, in that order.

The second part of the input is the map. It consists of either '.' (a free cell) or 'X' (occupied cell = wall). The bottom left cell in the text file is located in the origin of the coordinate system (0, 0). The x-axis follows this line to the east (right) while the y-axis follows this column to the north (up).

The width of the map is defined by the number of characters in the line (if the lines of the map have different length the program behavior is undefined - all test cases have the same line length for the map part.) The height of the map is defined by the number of lines in the map part.

Output:

The program has to output the map with the found path marked with the letter P.

The format follows exactly the input map format, except that the found path, including the start point and the goal point, are marked with the letter 'P' instead of a '.' .

Algorithm:

We save the calculated distance values in the map - each calculated (minimum) distance per map cell. At the beginning all values are initialized as infinite. Distance values for map cells which are occupied ('X') will never get updated.

Remember in Dijkstra's we keep a set of active nodes, which initially is the whole graph. We optimize this by instead using a wavefront algorithm: We start with the active set just consisting of the start node. During the algorithm only the neighboring nodes whose distance value we updated will be added to the active set.

The pseudocode for the Algorithm is now as follows:

```
Initialize all map cells with infinite distance
Initialize the start cell with 0 distance
step counter = 0
Add start cell to wavefront (active set)
While wavefront not empty:
  select the first node with the minimum dist value from the wavefront
    remove node from the wavefront
    if node == goal:
      break
    for all 8 neighbors of the node:
      if the neighbor is valid and not a wall:
        increase step counter
        neighDist = distance saved in node + distanceToNeighbor (1 or sqrt(2))
        if neighDist < distance saved in neighbor:
          update distance saved in neighbor with nieghDist
          append neighbor to the wavefront (active set)
Create the path:
  starting from the goalCell select the neighbor with the lowest distance saved
    add the selection to the path and continue with the selection to save it's lowest neighbor
      till you reached the start point
Print the map with the path
```

Additional Info

- Boundary cases are not handeled with special code. So if the goal node is outside of the map, for example, the the algorithm will still run as specified in the pseudocode till the wavefront is empty.
- Anyways, the only one special case is tested, and this is mainly about a special case when reading the map. Normally both start and goal point are within the map in unoccupied cells.

Length of Functions

The maximum number of lines of code (not counting blank, comment-only lines and sub-functions) per function is 40. If even a single function of your program exceeds this number you will be deducted 50% further points. (Point deductions are multiplied together - if you violate all three rules you will only get 25% of your points.)

Reason for this rule: You should practice to use a good coding style and split your problems/algorithms into sub-tasks using properly named functions.

Comments in the Code

You are required to write meaningful comments for your program. TA will check that you have a comment in at least 50% of the non-blank lines (so please do keep blank lines in your code to make it easier on the eyes). Your comments have to be in English and meaningful. The TAs will check your code by hand for comments. You will loose up to 50% of your score if you don't adhere to this rule.

Reason for this rule: Writing comments is essential once you start programming in a group/company. This will help other programmers (TAs) to understand your code. Having lots of comments is a very good coding practice which is very often neglected.

Exit status

Your program must not exit with a non-zero status. This means that if your program calls exit(), the argument must be zero.