17.10.2018

CmpE 480 - Introduction to A.I. Programming Assignment 1

Deadline: 05.11.2018, 23:59

Overview

In this assignment you are expected to implement three different search algorithms

1. Depth-first search (DFS)
2. Breadth-first search (BFS), and
3. A* search

to two different domains (labyrinth solving and mountain walking). You will read the input from text files for both of the problems. Each text file consists of one line of start coordinate, one line of goal coordinate and N lines of world matrix.

Guideline

- An example input for labyrinth solving problem:

  0 2

  2 2

  1 0 0 0 1

  1 0 1 1 1

  1 0 0 0 0

  Start coordinate is (0,2), end coordinate is (2,2) and the following lines are the 2D labyrinth representation, 0s represent traversable areas and 1s represent walls.

- An example input for mountain walking problem:

  0 1

  2 2

  1.3 1.5 1.7 1.5 1.0

  0.5 1.6 1.7 1.8 2.0

  3.5 3.0 2.0 1.0 0.0

  Start coordinate is (0,1), end coordinate is (2,2) and the following lines are the 2D mountain representation, float numbers represent height. The highest peak is at (2,0) with the height of 3.5, and the lowest point is at (2,4) with the height of 0.

1. You will read the input from a text file as shown above.
2. **<u>You will implement three different search algorithms following the conventions below:</u>**
    a. In <u>DFS</u> you have to add neighbor nodes into the stack in <u>ascending index</u> order so that your code will explore the nodes with higher indexes first.

```
0  0  0  0  0                       0  1  2  3  4

0  0  0  0  0      Indexes ⇒        5  6  7  8  9

0  0  0  0  0                      10 11 12 13 14
```

    Index of a node (i,j) in a 2D map with width W is calculated as follows:

    index(i,j) = i*W + j

    For example starting from coordinates (1,2) you should add neighbor nodes into the stack in the following order considering their indexes: 2, 6, 8, 12. {as coordinates: (0,2), (1,1), (2,2), (1,3).

    b. In <u>BFS</u> you have to add neighbor nodes into the queue in <u>descending index</u> order so that your code will explore the nodes with higher indexes first.

```
0  0  0  0  0                       0  1  2  3  4

0  0  0  0  0      Indexes ⇒        5  6  7  8  9

0  0  0  0  0                      10 11 12 13 14
```

    Index of a node (i,j) in a 2D map with width W is calculated as follows:

    index(i,j) = i*W + j

    For example starting from coordinates (1,2) you should add neighbor nodes into the queue in the following order considering their indexes: 12, 8, 6, 2. {as coordinates: (1,3), (2,2), (1,1), (0,2)}

    c. In <u>DFS</u> and <u>BFS</u>: when using map (or dictionary in Python) for keeping the path, add came_from information only once.
    For example when exploring nodes (2,2) and (1,1) in this order, both have (1,2) as their neighbors. Since (2,2) is explored first, you should add came_from information to (1,2) as (2,2). When exploring (1,1) you should **not** update came_from information of (1,2). At the end when you backtrace the path from (1,2) you should go back to (2,2) since it is explored first.

    d. In <u>A*</u> when selecting the node with the minimum F value select higher indexed node as a <u>tie-breaker</u>. If there is not a tie (only one node with the minimum F value) then you should not compare any indexes.

    Index of a node (i,j) in a 2D map with width W is calculated as follows:
    index(i,j) = i*W + j
    For example in a Nx5 world; if both nodes (8,0) and (7,2) have F value of *x* which is the minimum value among the rest of the nodes, you should select (8,0) because its index (8*5+0 = 40) is greater than the other's (7*5+2 = 37).

e. In A* use:

i. Manhattan distance as a distance measure for labyrinth solving. Manhattan distance between (x1, y1) and (x2, y2) is |x1 - x2|+|y1 - y2|.

ii. Euclidean distance as a distance measure for mountain walking problem. Euclidean distance between (x1, y1) and (x2, y2) with heights h1 and h2 is $\sqrt{(x1-x2)^2 + (y1-y2)^2 + (h1-h2)^2}$.

f. In A* when calculating a new G' value for a neighbor node if the previously calculated G value is the same as the new G' value, do not update anything (skip that neighbor node).

3. Constraints:

a. In both problems you cannot move diagonal. (neighbor nodes are at 4 directions left, right, above and below)

b. In labyrinth solving you cannot go to a node with a value of 1 (which represents a wall).

c. In mountain walking you cannot go from node A to node B if the height difference is greater than 1 (therefore node A and B are not connected).
From A (height=0.5) to B (height=1.6) you cannot move because the height difference is 1.1 and vice versa.

Output

- For each problem (labyrinth solving and mountain walking):
  - For each **example** input (labyrinth_example_<i>.txt, mountain_example_<j>.txt):
    - You have 3 output text files (1 for dfs, 1 for bfs and the other for A*)
- For each problem (labyrinth solving and mountain walking):
  - For each **test** input (labyrinth_test_<i>.txt, mountain_test_<j>.txt):
    - You will output 3 text files (1 for dfs, 1 for bfs and the other for A*)
- For NxM input world, your output text files must follow the conventions below:
  - N lines of NxM **is_discovered** matrix consisting of 1s and 0s. If a node is discovered (meaning that its neighbors are added to the stack, queue or set) it should have a 1 and all the other nodes should have zeros.
  - 1 line of path length T
  - T lines of path (same as in example outputs)
  - 1 line of total distance traveled
    - Euclidean distance for mountain walking
    - manhattan distance for labyrinth solving) (distance travelled can be ∓0.01 from the expected output)

Specifications

- There are no programming language restrictions.
- Your output must exactly match with the expected output, so please make sure that you follow the conventions strictly.
- We include 5 example inputs for each problem and their corresponding outputs for each search algorithm. Please make sure that your algorithms generate exactly the same outputs.
- In total you should submit 30 output text files with the proper names:
  - If the input file name is labyrinth_test_1.txt, your output files must be named:
    - labyrinth_test_1_bfs_out.txt,
    - labyrinth_test_1_dfs_out.txt
    - Labyrinth_test_1_a_star_out.txt

  (for input file <filename>.txt, output files must be named: <filename>_bfs_out.txt, <filename>_dfs_out.txt, <filename>_a_star_out.txt)

- You will combine all the output files into one zip file as <your_student_id>.zip
  - Inside this zip you should add two folders
    - labyrinth
    - mountain
  - Inside each folder you should add your outputs to all of the test inputs
    - labyrinth_test_1_a_star_out.txt
    - labyrinth_test_1_bfs_out.txt
    - labyrinth_test_1_dfs_out.txt
    - ...
- Write your report on the dokuwiki page of yours at: http://robot.cmpe.boun.edu.tr/~cmpe480/doku.php?id=user:students
- Send your output files (<your_student_id>.zip) and your code to metehandoyran@gmail.com with the subject:
  "CMPE 480 Programming Assignment 1"
  (Note that your e-mails will be filtered by the subject)

Report

- Explain your code in detail.
- Compare the three algorithms. Point out in which examples which algorithm finds optimal paths or which algorithm finds a path with discovering less nodes.