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. **You will implement three different search algorithms following the conventions below:**
   a. In DFS you have to add neighbor nodes into the stack in ascending index order so that your code will explore the nodes with higher indexes first.

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

   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 DFS and BFS: 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 A* when selecting the node with the minimum F value select higher indexed node as a tie-breaker. 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.