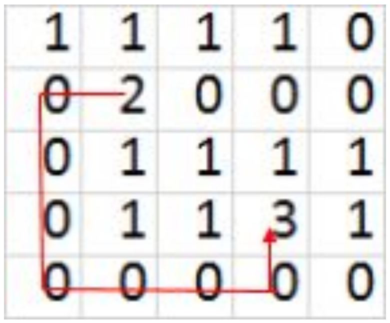
2018 Winter Lab1: Search

Part1:

The task in this part is to write two search functions to help a robot find its way to a destination. The robot exists in a grid world named “map” of size MAP_WIDTH by MAP_HEIGHT, and it can move in all 4 directions (diagonals not allowed) through empty space cells only by steps of exactly 1 cell distance.
The starting location of the robot is marked in map with a number “2” and the goal with a number “3”. The other two values you can find in the map are “1” for walls and “0” for empty space.

Sample map of size MAP_WIDTH 5 and MAP_HEIGHT 5



1	1	1	1	0
0	2	0	0	0
0	1	1	1	1
0	1	1	3	1
0	0	0	0	0

Robot starting location in map[1][1]

Goal in map[3][3]

You are required to complete the code for 2 functions: dfs (testmap) and bfs (testmap). The testmap is a two-dimension list. The first one uses depth first search and the second one uses breadth first search to find a path from the robot starting location to the goal. The functions should return the same testmap and mark the map with a number “4” in all explored cells or with a number “5” in the cells that are part of path found.

To make sure everybody arrives to the same results (very important for the automated grader) you must use the following search order for map[y][x]: First [y][x+1], then [y+1][x], then [y][x-1], and finally [y-1][x]

Considerations:

- We provided several maps to let you test your solution, but the grading will use a different set.
- The starting location of the robot and the destination are part of the path and should

be   marked with a “5” in the map.

● The running time of your algorithm cannot be longer than 5 seconds for a 15x15 maps or smaller, otherwise it will fail the grading tests.

● All maps will have a maximum of 1 possible path between the starting location and the destination (to make it easier).

● There will be no loops in the maps (to make it easier).

Part2:

The task in this part is to write an A-star search algorithm to find the path with lowest cost. You will be provided two data structures (Python dictionaries)– distance and time between locations. One provides the distances for all pairs of locations. The other data structure serves two purposes: 1) the connections (road) between locations, 2) time to traverse the road. Both structures have the same format and the following is a pair of distance and time structure of Evanston (A dictionary in Python):

Time_dic =
{'Campus': {'Campus': None, 'Whole_Food': 4, 'Beach': 3, 'Cinema': None, 'Lighthouse': 1, 'Ryan Field': None, 'YWCA': None},
'Whole_Food': {'Campus': 4, 'Whole_Food': None, 'Beach': 4, 'Cinema': 3, 'Lighthouse': None, 'Ryan Field': None, 'YWCA': None},
'Beach': {'Campus': 4, 'Whole_Food': 4, 'Beach': None, 'Cinema': None, 'Lighthouse': None, 'Ryan Field': None, 'YWCA': None},
'Cinema': {'Campus': None, 'Whole_Food': 4, 'Beach': None, 'Cinema': None, 'Lighthouse': None, 'Ryan Field': None, 'YWCA': 2},
'Lighthouse': {'Campus': 1, 'Whole_Food': None, 'Beach': None, 'Cinema': None, 'Lighthouse': None, 'Ryan Field': 1, 'YWCA': None},
'Ryan Field': {'Campus': None, 'Whole_Food': None, 'Beach': None, 'Cinema': None, 'Lighthouse': 2, 'Ryan Field': None, 'YWCA': 5},
'YWCA': {'Campus': None, 'Whole_Food': None, 'Beach': None, 'Cinema': 3, 'Lighthouse': None, 'Ryan Field': 5, 'YWCA': None}}

Distance_dic =
{'Campus': {'Campus': 0, 'Whole_Food': 3, 'Beach': 5, 'Cinema': 5, 'Lighthouse': 1, 'Ryan Field': 2, 'YWCA':12},
'Whole_Food': {'Campus': 3, 'Whole_Food': 0, 'Beach': 3, 'Cinema': 3, 'Lighthouse': 4, 'Ryan Field': 5, 'YWCA':8},

```
'Beach': {'Campus': 5,  'Whole_Food': 3, 'Beach': 0, 'Cinema': 8, 'Lighthouse': 5, 'Ryan Field': 7,
'YWCA':12,},
'Cinema': {'Campus': 5,  'Whole_Food': 3, 'Beach': 8, 'Cinema': 0, 'Lighthouse': 7, 'Ryan Field': 7,
'YWCA':2},
'Lighthouse': {'Campus': 1, 'Whole_Food': 4, 'Beach': 5, 'Cinema': 7, 'Lighthouse': 0, 'Ryan Field':
1, 'YWCA':15},
'Ryan Field': {'Campus': 2, 'Whole_Food': 5, 'Beach': 7, 'Cinema': 7, 'Lighthouse': 1, 'Ryan Field':
0, 'YWCA':12},
'YWCA': {'Campus': 12, 'Whole_Food': 8, 'Beach': 12, 'Cinema': 2, 'Lighthouse': 15, 'Ryan Field':
12, 'YWCA':0}}
```

For example, the traffic time between Campus and Beach is 3. In the data structure, if a value is None, there is no road between that pair of locations. Otherwise, the value is a number, and that number indicates the time to traverse that road.

The basic function used in the A star search, as always, is f(x) = g(x) + h(x). In this part, g(x) is the time to get to that node from the start node, and h(x) (the heuristic function) is the distance from that node to the end node.

Your goal is to complete the function: a_star_search(dist_map, time_map, start, end), and to output a score dictionary such that all locations in your lowest cost path are keys and the value for each key is another dictionary such that contains the f(x) for each of that locations adjacent nodes. For example, give the time structure above, if the Campus is the first location considered and the goal location is the Cinema, then the entry for “Campus” should b=be the following: ‘Campus’: {‘Lighthouse’: 8, ‘Beach’: 11, ‘Whole_Food’: 7}. Please **include** the start location and **do not include** the end location in the dictionary. You must follow the output format as above to make sure everyone get the same results.