PA1: Pathfinding with A* Search
DUE: Tuesday, February 24 at 11:59 PM

Groups: You will work in groups of two for this assignment.

Objective: To understand how A* search works and test some variations of the algorithm/heuristics.

Pathfinding in AI
Pathfinding is a common problem that artificial agents must solve, including mapping services, artificial vehicles, AIs in real-time strategy games, and robots that must navigate in the physical world. For this assignment we will focus on simplified pathfinding problem. Your program will search for the shortest path from a starting location to a goal location on a square grid. However, this grid will have some impassable locations and varying terrain costs that must be accounted for. The agent is allowed to move in any of the four primary directions (up, down, left, right) but not diagonally or outside the bounds of the map.

A* Initial Implementation
For this assignment you will start from a base implementation of A* search available online (so you do not need to implement the algorithm from scratch). The reference implementation is available in Python here:

https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2

Your first task is to read this code carefully to understand how the algorithm is implemented and how the code works. Test the given sample case on your own. As with any code base it is possible that there will be bugs, and you will be making some modifications to this code in the following sections.

Task 1: Varying Costs

The first task is to modify the code so it accepts variable costs and different movement constraints. The map will be specified as digits between 0 and 5 (in the same form as the existing code). However, we will interpret "0" as impassable terrain, and 1-5 as varying costs the agent pays each time it moves INTO a given square (so if it into 2,3 with the number 3, the agent pays a cost of 3 for that move). There is no cost for moving to the starting location. We will start with indices 0,0 in the upper left of the maze. We will also only allow four possible moves: Up, Down, Left, and Right. For the initial implementation/testing you should use the Manhattan distance as your heuristic. Print out the following information at the end of each run of the algorithm:

1) The cost of the path found
2) The path as a sequence of coordinates (row, col), (row col), … , (row, col)

3) The number of nodes created; include in the count duplicate paths found to the same node whether or not they are shorter
4) The runtime of the algorithm in milliseconds

If the algorithm terminates without finding a result, print -1 for the path cost and NULL for the path sequence. You should still print the number of nodes and runtime.

Now test the code on the three examples provide below, as well as two of your own design (with minimum size 10x10). Provide the results as well as the test cases in your report. For this assignment you can add the test cases directly into the code as in the reference implementation; you do not need to read it in from a file.

Start: 1,2
Goal: 4,3
2 4 2 1 4 5 2
0 1 2 3 5 3 1
2 0 4 4 1 2 4
2 5 5 3 2 0 1
4 3 3 2 1 0 1

Start: 3,6
Goal: 5,1
1 3 2 5 1 4 3
2 1 3 1 3 2 5
3 0 5 0 1 2 2
5 3 2 1 5 0 3
2 4 1 0 0 2 0
4 0 2 1 5 3 4
1 5 1 0 2 4 1

Start: 1,2
Start: 8,8
2 0 2 0 2 0 0 2 2 0
1 2 3 5 2 1 2 5 1 2
2 0 2 2 1 2 1 2 4 2
2 0 1 0 1 1 1 0 0 1
1 1 0 0 5 0 3 2 2 2
2 2 2 2 1 0 1 2 1 0
1 0 2 1 3 1 4 3 0 1
2 0 5 1 5 2 1 2 4 1
1 2 2 2 0 2 0 1 1 0
5 1 2 1 1 1 2 0 1 2

Task 2: Varying Heuristics

Now, you will experiment with some different heuristic functions to examine the impact on solution time and the cost of the final path found. Modify your code to use any of the following four heuristics:

H1: All zeros; for every node this just returns the value 0.
H2: The Manhattan distance (already implemented in part 1)
H3: A modified Manhattan distance or other heuristic of your own design; describe it in the report. This does not necessarily need to be admissible, but it should be as accurate as you can make it.
H4: Manhattan distance with error. Use the Manhattan distance but add errors from -3 to +3, excluding 0 with a uniform random distribution (so each error value will occur with probability 1/6). After adding in the error lower bound the value of the heuristic at 0 (so if it is negative the value becomes 0).

Test your 5 test cases with these different heuristics. Create tables to show the results for the cost of the solution found using each heuristic, the total number of nodes created, and the runtimes. Discuss the results in terms of optimality and efficiency.

What to Turn In

Turn in your modified source code in Python; it should be runnable on the command line with the first parameter (1-5) specifying the map to use (4 and 5 are your own), and the second (1-4) specifying the heuristic to use.

In addition, write a short report showing your results for the two different tasks. Show your results/additional test cases for part I, a description of your new heuristic and results for part II with a short explanation of your main findings.