CPSC 390
Artificial Intelligence
Spring 2018

Homework #2
Best-First Robot Navigation
Due: 3/6/18

How to complete this HW: This homework is a programming assignment to be done individually. When you are done, you should submit ONLY your commented source code, Makefile, and a README (with your name, student id, etc) in a zip on blackboard.

Note on Collaboration: You may discuss this program with other students in the class and/or look into other documents (books, web sites), with the exception of published solutions. However, all code and answers must be your own work. If you have discussed the program with other students, indicate their name(s) in the README.

Problem Description

In this assignment, you will implement a solution to a navigation problem, which is as follows:

> A robot represented as a point moves in a regular two-dimensional NxN grid (e.g., N = 100). Each point of the grid is either "free" or "forbidden" (obstacle). From any position (i,j) in the grid the robot can reach each of the 4 adjacent positions (i,j-1), (i,j+1), (i-1,j), (i+1,j), if it is not forbidden. A navigation problem consists of finding a path in the grid (sequence of adjacent free points) that connects a given initial position to a given goal position. Each move has a cost of 1.

General Information

Write a program implementing the Best-First Search algorithm to solve this problem. To do this, first formulate the navigation problem as a search problem: what are the states, the successor function, the initial and goal states? Allowing the search tree to contain nodes labeled by the same state leads to an infinite search tree. So, the program should avoid duplicating states.

You should have sections in your program to test each of the following four evaluation functions:

- f(N) = Euclidean distance from N to the goal (Strategy 1)
(i.e. the length of a straight line between two points, E((i,j),(i',j')) = sqrt[(i-i')^2+(j-j')^2])

- f(N) = Manhattan distance to the goal (Strategy 2)
(i.e. the length of the shortest path obtainable by traversing only in the cardinal directions, ignoring any obstacles, M((i,j),(i',j')) = |i-i'| + |j-j'|)

- f(N) = g(N) + h(N), where:
    - g(N) is the cost of the path found so far from the initial node toN
    - h(N) is: - the Euclidean distance from N to the goal (Strategy 3)
      - the Manhattan distance to the goal (Strategy 4)

Thought question: Is the algorithm A* with the last two functions? If yes, which of the two h should produce the smallest search tree?

For each problem-function pair, your program should output the generated path, its cost, and the number of nodes in the search tree when the solution was found.

Getting Started

All programming is to be done in Java or C++. Code on any platform you like, but make sure your program compiles correctly using Oracle's SDKor g++ under linux.

Your program will take as a command-line argument the name of a text file representing the map of the world the simulation will be carried out in.

The input files will contain information about the map that the robot will traverse. The first line will have a number N that is the width and the height of the map (all maps are NxN, i.e. same height and width). No map will have N greater than 80. The following N lines will detail the map. Each line will have N characters, representing the N spaces in that row. The first line will be row N and the first character in each row will be in column 0. The characters in the rows will be as follows:

. empty space (traversable)
i initial space (traversable, robot starts here)
g goal space (traversable, robot's goal here)
+ obstacle (not traversable)

A sample input file might be as follows:

```
5
...g.
.++..
.i+..
..+..
+...+
```

Every input file is expected to have at least one possible path to the goal. I will provide some sample input files for you to test with. Remember that your robot cannot move diagonally; it can only move to the spaces immediately up, down, left, and right. The robot cannot enter a space occupied by an obstacle.

The maps that you will generate as output should be the same as the maps in the input files, except that each step in the path taken will be represented as an ‘o’ (the lower case letter o). The goal and initial spaces should remain ‘g’ and ‘i’ respectively. As an example, a solution map to the above input map might be:

```
ooog.
o++..
oi+..
..+..
+...+
```

You should format your output as above. (Don’t forget to include path cost, etc. You can print this out below the map for each problem-function pair.)

Deliverables

Your program will be tested for correctness for various maps. The correctness is worth 90 points.

10 points will be awarded for code quality. (Your code will need to be exceptionally badly designed to lose these points, so don’t stress too much about this.)

In addition to the program code, you should supply a README that includes your name, e-mail, ID, and any comments you might have, as well as a basic Makefile.