Lab 2: Informed Search Methods

> *CS410: Artificial Intelligence*
> *Shanghai Jiao Tong University, Fall 2021*
> *Due Time: 2021.10.17 23:59*

Environment

All problems in this lab are based on the maze environment introduced in Lab 1. Recall in this environment, a maze consists of a character `S`, a character `G`, and characters `%`.

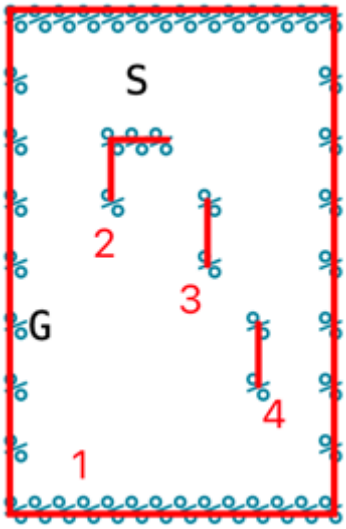
Assignment

Exercise 1

In the maze environment, treat a connected area of characters `%` in the maze as a `lake`. Two characters `%` are adjacent if one could be reached from another in any direction of North, South, West, or East.

A. Number of 'Lakes'

The first question is to use depth-first graph search (DFGS) to find out the number of 'lakes'. The input is an instance of maze, and the output is an integer indicating the number of lakes. It is required to finish `Exercise2_1_1` in `algorithm.py` using the `Stack` data structure implemented in Lab 1. For example, the following maze in `Maze_lab2_1_1.lay` has 4 lakes.



B. DFGS with Iterative Deepening

Consider again the problem of searching the path from the start state S to the goal state G (Lab 1, Exercise 2). If this problem is presented in a large maze, the DFGS would be inefficient. Now improve the DFGS by the trick of iterative deepening (Lecture 2, Slide 39) with TWO different increasing functions on depths. Implement DFGS with the iterative deepening trick in `Exercise2_1_2` in `algorithm.py`. Plot the time and memory consumption against the search depth for these two functions respectively. How are the performances of these two functions? What is the advantage of each one? Then give your discussions on how to define the increasing function on depths in real applications.

Exercise 2: Least-Cost Path (Uniform-Cost Graph Search)

It is required to use uniform-cost graph search (UCGS) to find the least-cost path from the start state S to the goal state G in the maze. The input is an instance of maze and the output is the least-cost path from S to G, together with the cost. Implement `Exercise2_2_1` in `algorithm.py` and run your implementations.

How about the performance of the greedy graph search (GGS) with the Euclidean distance as the heuristic function on the maze `Maze_lab2_2_1.lay`? Implement GGS in `Exercise2_2_2` of `algorithm.py`. Can you design two environments such that each of the UCGS and GGS has better performance? How general environments can you conclude from the design of these two example environments? Based on this, discuss on the drawback of UCGS.

The usage of the `PriorityQueue` data structure in the Python standard library is allowed.

Exercise 3: Least-Cost Path with Heuristics

It is required to use A^* graph search (A^*GS) with the Euclidean distance as the heuristic function to find the least-cost path from the start state S to the goal state G in the maze. The input is an instance of maze and the output is the least-cost path from S to G, together with the cost. Implement `Exercise2_3_1` in `algorithm.py` and run your implementations. Compare the experimental results of UCGS, GGS and A^* GS on the mazes `Maze_lab2_2_1.lay` and `Maze_lab2_3_1.lay` respectively. Are the solutions found by these three methods optimal? How about the number of expanded nodes in the search procedures of these three search methods? For A^* GS, can you design a heuristic function which is better than the Euclidean distance? Explain why the new heuristic function is better both theoretically and experimentally.

Denote the coordinate of the goal state G as (x_G, y_G). Consider this heuristic function. For each state P, define its distance to the goal state G as $\text{dis}(P, G) = |x_P - x_G| + |y_P - y_G| - \mathbb{I}\{|x_P - x_G| \neq |y_P - y_G|\}$, where (x_P, y_P) is the coordinate of the state P and $\mathbb{I}\{\cdot\}$ is the indicator function. Does A^* GS still work under this heuristic function? Give your analysis if it works or design a maze to make it fail otherwise.

The usage of the `PriorityQueue` data structure in the Python standard library is allowed.

Submission

There are **only** two files you need to submit:

- `algorithm.py` where your search algorithms are implemented;
- `report.pdf` for your **brief** report in English.

Name your **zip** file containing the above as `xxxxxxxxxxxx.zip` with your student ID, and submit it on Canvas.