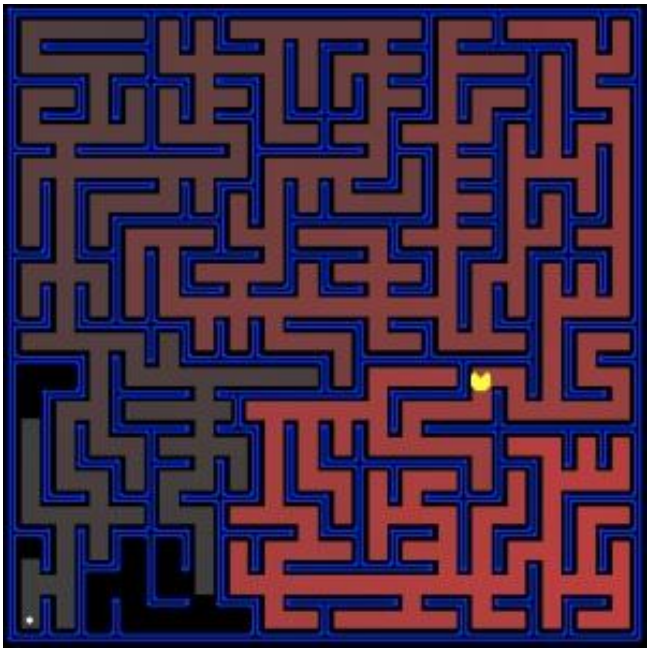
CS591 Programming Project 1

Maze Search



Grade: 100 marks in total (Program: 70%, Team Review: 30%).

Type: Group work (Team size cannot exceed 5)

Programming language: Python (preferred), Java, C/C++, or any other advanced programming languages

Hints: Textbook online repository files (search.py, search.ipynb, search4e.ipynb) in aima-python, or similar files in other folders, such as aima-java.

Tasks:

Part 1: Basic pathfinding

To begin with, you will consider the problem of finding a path through a maze from a given start state to a given goal state. This scenario is illustrated in the figure above, where the start position is indicated by the "Pacman" icon and the goal state is a dot. The maze layout will be given to you in a simple text format, where '%' stands for walls, 'P' for the starting position, and '.' for the goal (see sample maze file). For this part of the assignment, all step costs are equal to one.

Select **ONE** from the following search algorithm Task choice 1 and the A* search algorithm in Task choice 2 for solving different mazes:

Task choice 1:

- Depth-first search;
- Breadth-first search;

Task choice 2:

- A* search.

For A* search, use the Manhattan distance from the current position to the goal as the heuristic function.

Run each of the above algorithms on the small maze, medium maze, big maze, and the open maze. For each problem instance and each search algorithm, report the following:

a. The solution and its path cost;
b. Number of nodes expanded;
c. Maximum tree depth searched;
d. Maximum size of the fringe.

You can display the solution by putting a '.' in every maze square visited on the path (example solution to the big maze).

Part 2: Search with multiple goals

Now we consider a harder problem of finding the shortest path through a maze while hitting *multiple* goals (that is, you want to make the Pacman, initially at P, eat *all* the dots). trickySearch.lay is a sample problem instance. Once again, in this part, we assume unit step costs.

Revise your code from Part 1 to deal with this scenario. This will require changing the goal test (have you eaten all the dots?) and the state representation (besides your current position in the maze, is there anything else you need to know?).

Run the two search algorithms from Part 1 on the tiny search, small search, and tricky search. For each search method and problem instance, report the solution cost and number of nodes expanded.

It may come as a surprise to discover that uninformed searches can be very inefficient even when dealing with small problems. As such, it is recommended to set a reasonable upper limit on the number of nodes to be expanded, and to terminate the search without a solution if this limit is surpassed. To improve the chances of finding a solution in a timely manner, it is crucial to develop a strong heuristic. It is advisable to dedicate some time to thinking about this. When writing the report, it is important to discuss the chosen heuristic and provide an explanation for why it is admissible. It is also acceptable to propose multiple heuristics and present the results for each of them. The goal should be to develop a heuristic that is even more effective than the others.

Tips

- Make sure you get all the bookkeeping right. This includes handling of repeated states (in particular, what happens when you find a better path to a state already on the fringe) and saving the optimal solution path.

- Pay attention to tiebreaking. If you have multiple nodes on the fringe with the same minimum value of the evaluation function, the speed of your search and the quality of the solution may depend on which one you select for expansion.

- You will be graded on the correctness of your solution, not on the efficiency and elegance of your data structures. For example, I don't mind whether your priority queue or repeated state detection uses brute-force search, as long as you end up expanding (roughly) the correct number of nodes and find the optimal solution. So, feel free to use "dumb" data structures as long as it makes your life easier and still enables you to complete the assignment in a reasonable amount of time.

- Make your report easy to understand and interesting to read. Feel free to note any interesting observations you made or insights you gained while doing the assignment. The goal of the assignments is to get you to explore and to learn, and I would like to see what you learned reflected in your report.

- I reserve the right to give **bonus points** for any advanced methods or especially challenging solutions that you implement. For this assignment, I may give you bonus points if you decide to do a nicer graphical visualization of the mazes and the found solutions. (bonus is 5 ~ 10 points)

Submission Instructions

You will need to turn in the following:

1. A **report** in **PDF format**. The report should briefly describe your implemented solution and fully answer the questions for every part of the assignment. Your description should focus on the most "interesting" aspects of your solution, i.e., any non-obvious implementation choices and parameter settings, and what you have found to be especially important for getting good performance. Feel free to include pseudocode or figures if they are needed to clarify your approach. **Your report should (ideally) make it possible for me to understand your solution without having to run your source code.**

2. At the **beginning** of the report, the report must contain all the team members' name, ID, graduate or undergraduate (e.g., Juefei Yuan, S0000000, graduate)

3. Your **source code**. The code should be well commented, and it should be easy to see the correspondence between what's in the code and what's in the report. You don't need to include executables or various supporting files (e.g., utility libraries) whose content is irrelevant to the assignment. If I find it necessary to run your code in order to evaluate your solution, I will get in touch with you.

Please **zip** both the report and source code into one package and name it based on the following rule:

"**Team leader_First_Name_ Team leader _Last_Name_P1.zip**"

(i.e., Juefei_Yuan_P1.zip).

Submission venue: Submit your files through Canvas.

Late policy: You lose 25% of the points for every day the assignment is late. If you have a compelling reason for not being able to submit the assignment on time and would like to make a special arrangement, you must let me know **at least a week before the due date** (any genuine emergency situations will be handled on an individual basis).

Academic integrity: Feel free to discuss the assignment with each other in general terms, and to search the Web for general guidance (not for complete solutions). Coding should be done individually. If you make substantial use of some code snippets or information from outside sources, be sure to acknowledge the sources in your report.