National University of Singapore
School of Computing
CS3243 Introduction to AI

Project 1.1: Introduction to Search

Issued: 28 Aug 2023 Due: 10 Sep 2023, 2359hrs

1 Overview

In this project, you will **implement 2 search algorithms** to find valid paths in a maze.

1. **Depth-first search (DFS)** algorithm
2. **Uniform-cost search (UCS)** algorithm

This project is worth 2% of your course grade.

1.1 General Project Requirements

The general project requirements are as follows.

- **Individual** project, but you are *allowed to consult the P2ST (ChatGPT) App*. More details can be found in Section 4.2
- Python Version: $\geq$ **3.8**
- Deadline: **10 Sep 2023**, **2359 hours**
- Submission folder: **Canvas** > **CS3243** > **Assignments** > **Project 1.1**. More details can be found in Section 4.

1.2 Academic Integrity and Late Submissions

Note that any material that does not originate from you (e.g., is taken from another source), with the exception of the P2ST (ChatGPT) App, should not be used directly. You should do up the solutions on your own. Failure to do so constitutes plagiarism. Sharing of materials between individuals is also strictly not allowed. Students found plagiarising or sharing their code will be dealt with seriously.

For late submissions, there will be a 20% penalty for submissions received within 24 hours after the deadline, 50% penalty for submissions received between 24-48 hours after the deadline,

and 100% penalty for submissions received after 48 hours after the deadline. For example, if you submit the project 30 hours after the deadline and obtain a score of 92%, a 50% penalty applies and you will only be awarded 46%.

---

2 Project 1.1: Escape the Maze!

2.1 Functionality

You will be given a static maze that has a variable initial size. Given a starting position, the **objective** is to find a path from the starting position to a designated goal square **without** passing through any obstacles.

As such, the following are some constraints on what you can or cannot do.

- You can only move in 4 directions: Up, Down, Left, or Right.
- You cannot move onto a cell blocked by an obstacle.
- You cannot move outside the bounds of the maze.

2.2 Board

In this project, the initial maze size is a **parameter**, with the maximum number of columns being 1100 and the maximum number of rows being 1100.

2.2.1 Coordinate System - Matrix Coordinates

The coordinate system used on a maze is the **matrix coordinate system** i.e. `(row, col)`. For example, in a 5×10 grid, there are 5 rows and 10 columns. The index of the rows are 0, 1, 2, 3, 4 from **top to bottom**, and the index of the columns are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 from **left to right**. These indices are used to represent the squares on the maze, e.g., the square on row 3 and column 0 is represented as $(3, 0)$.
All inputs and outputs relating to positions will be given in matrix coordinates! Note that the matrix coordinate system *is not* the Cartesian coordinate system.

2.3 Input Constraints

In the following section, a `Position` type is a `List[int]` of length exactly 2.
You will be given a `Dict` with the following keys:

- `columns`: The number of columns the maze has. Type is `int`.
- `rows`: The number of rows the maze has. Type is `int`.
- `obstacles`: The list of positions on the maze occupied by obstacles. Type is `List[Position]`.
- `start`: The starting position. Type is `Position`
- `goals`: The goal positions. Type is `List[Position]`

The action cost to get to any position is 1.

2.4 Requirements

You are to implement a function called `search` that **takes in** a dictionary as described in Section 2.3 and **returns** a valid path to a given goal. In particular, you should return a `List[Tuple[int, int]]` representing the path that your agent will take. For example, if you start at $(0, 0)$, then move to $(0, 1)$, then move to $(1, 1)$, you should output `[(0, 0), (0, 1), (1, 1)]`.

The following are some **general requirements** on your output.

1. All positions along the produced path must be free of obstacles
2. All positions along the produced path must be reachable from the previous position by a single movement in the 4 directions: UP, DOWN, LEFT, RIGHT, subject to maze boundaries and obstacles.
3. Paths of non-zero length must terminate at a given goal position
4. Paths of non-zero length must begin from the given start position
5. If there are no legal paths, return an empty list i.e. a path of zero length.

For UCS, there is an additional requirement of having to output a path with the lowest cost.

---

3 Grading

3.1 Grading Rubrics (Total: 2 marks)

Requirements (Marks Allocated)	Total Marks
• Correct implementation of Depth First Search Algorithm (0.5m). • Efficient implementation of Depth First Search Algorithm (0.5m). • Correct implementation of Uniform Cost Search Algorithm (0.5m). • Efficient implementation of Uniform Cost Search Algorithm (0.5m).	2

3.2 Grading Details

3.2.1 Correctness

We will run your code on a set of public and private test cases. There are 4 possible outputs when grading an implementation on a given test case:

- **Accepted (AC):** You have returned the correct path within the given time limit. For UCS, the returned path has an optimal cost. You have **passed** the test case.
- **Wrong Answer (WA):** You have either returned an invalid path or a non-optimal path (for UCS only). You have **not passed** the test case.
- **Time Limit Exceeded (TLE):** Your code runs beyond the given time limit. You have **not passed** the test case.
- **Runtime Error (RTE):** Your code has thrown an exception or caused an exception to be thrown. You have **not passed** the test case.

Different test cases may have different weights. You will get full credit for the implementation only if you obtain AC for all test cases using that implementation. For the specific points distribution, refer to Section 5.2.

3.3 CodePost (Platform for Code Testing)

We will be using CodePost as our standardised platform for you to run and test your code on public and private test cases. You should have received an email from CodePost in your NUS email stating that you have been added to the course on CodePost. If you have not received this email, you can join using the link below. Do note that you may only join using your NUS email.

3.3.1 Using CodePost

Logging in for the first time: invitation link.

Remember to check your spam/junk mail for the activation email. Contact the course staff if you do not receive it within 30 minutes.

Subsequent access: link

1. For each task (DFS/UCS), click on "Upload assignment", and upload your Python file `dfs.py/ucs.py` (DO NOT rename the files).
2. After the submission has been processed, refresh the page and select "**View feedback**".
3. Ideally, if your implementation is correct and is within the runtime threshold, the output will look like this:

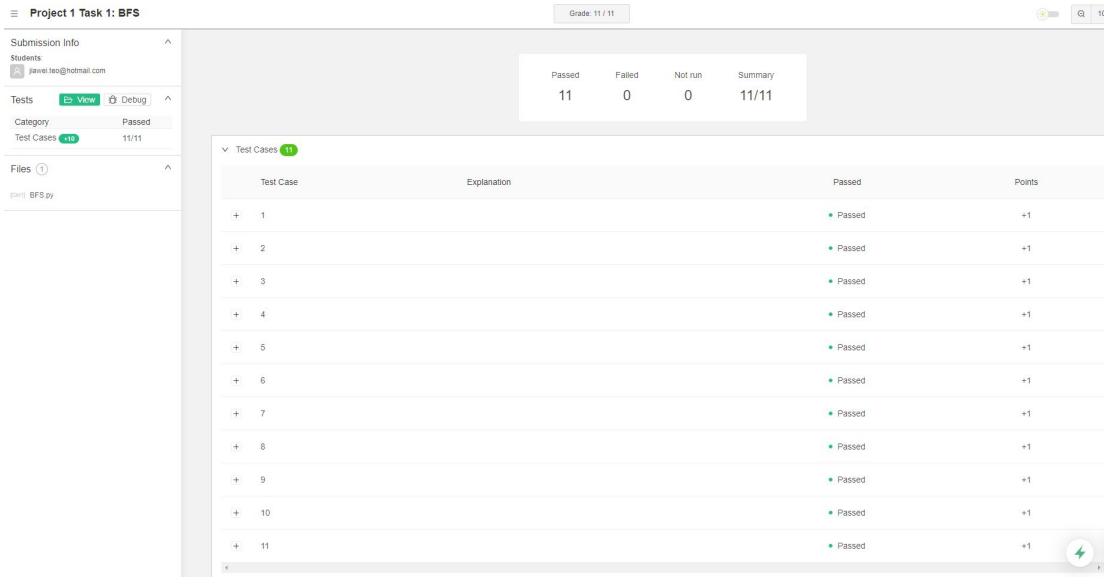

Figure 1: Codepost output

3.3.2 CodePost for Testing

CodePost hosts both the public test cases (which have been released via Canvas together with the skeleton implementation files) and the private test cases. The output on CodePost has been purposefully sanitised to prevent attempts at finding out the private test cases. As such, you are **expected to check your implementations thoroughly** via **custom-made test cases run locally**,

as you will not find CodePost useful for debugging purposes.

Note that CodePost is for you to run and test your code on the test cases. Your solution will **not be graded** using CodePost; instead, your Canvas submission will be used and run on the same environment as CodePost. Therefore, passing all test cases in CodePost **does not guarantee** full credit. Some possible reasons for not obtaining full credit despite passing all test cases in CodePost include: being lucky in CodePost due to randomness in the algorithm, being caught plagiarising, etc. Note that we **compare code with students from previous semesters**. **We will check for any plagiarism and students found plagiarising will be dealt with seriously.**

Note that CodePost is run by an external organisation – we are not responsible for any downtime.

---

4 Submission

4.1 Submission Details via Canvas

- For this project, you will need to submit 1 zip file containing 2 Python files: `dfs.py` and `ucs.py`.
- Place both files in a folder, name the folder as `studentNo` and then zip it. Name the zipped file as `studentNo.zip`. For example: `A0123456Z.zip`.
- When unzipped, your submission file must contain the 2 files in a folder: **A0123456Z.zip** $\rightarrow$ **unzip** $\rightarrow$ **A0123456Z folder** $\rightarrow$ **dfs.py, ucs.py**. There should not be any subfolders.
- Do not modify the file names.
- Please follow the instructions closely. **If your files cannot be opened or if the grader cannot execute your code, this will be considered failing the test cases as your code cannot be tested.**
- Submission folder: **Canvas** $>$ **CS3243** $>$ **Assignments** $>$ **Project 1.1**

You may submit your files as many times as you like, but only the latest one will be graded, **even if it means incurring a late penalty**.

4.2 P2ST (ChatGPT) App Usage

The P2ST (ChatGPT) app has been developed for CS3243 (this course). It is a tool that can be used to generate code from natural language descriptions. The objective is to help you better understand the contents of the course while skipping some of the more tedious parts of coding.

You are **permitted** to use the P2ST (ChatGPT) app to generate code to help you with this project. No other app or AI-generated code may be used.

The plagiarism-checking phase will be conducted using a tool that can identify code that has been copied from other sources. If your code is flagged as duplicated, you may appeal to the teaching team only if the code was generated from or with the help of the P2ST (ChatGPT) app. If the teaching team finds that the code was not generated using the P2ST (ChatGPT) app and was instead generated via other means, e.g. by using other AI assistance tools, plagiarising other people's work, etc., the appeal will not be successful.

Note that there is no guarantee that the code generated by the app will be correct or efficient. You are **responsible for any submissions made**.

5 Appendix

5.1 Allowed Libraries

The following libraries are allowed:

- Data Structures: queue, collections, heapq, array, copy, enum, string
- Math: numbers, math, decimal, fractions, random, numpy
- Functional: itertools, functools, operators
- Types: types, typing

For other libraries, **please seek permission before use!**

5.2 Points Distribution

5.2.1 DFS

1. Correctness: each test case is worth 1 point. Total is 10 points.
2. Efficiency: each test case is worth 1 point. Total is 10 points.

5.2.2 UCS

1. Correctness: each test case is worth 1 point. Total is 10 points.
2. Efficiency: each test case is worth 1 point. Total is 10 points.