National University of Singapore
School of Computing
CS3243 Introduction to AI

Project 1.2: Introduction to Search

Issued: 28 Aug 2023 Due: 24 Sep 2023, 2359hrs

1 Overview

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

1. **Breadth-first search (BFS)** algorithm
2. **A*** algorithm

This project is worth 8% 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: **24 Sep 2023**, **2359 hours**
- Submission folder: **Canvas** > **CS3243** > **Assignments** > **Project 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 1.2: Escape the Dungeon!

2.1 Description

Your character is looking for runes in a dungeon filled with many enemy creeps. Each move you make and each creep encountered will cause you to lose HP. You are looking to get to (exactly) one rune without losing too much HP.
At each timestep, your character can do exactly one of the following:

- Take an **action**; or,
- Use a **skill**

An **action** is a movement (Up, Down, Left, Right) in a 2D dungeon (you have seen this in project 1.1). The exact rules and costs of an action are described in Section 2.2. A **skill** *modifies the next action that is taken or the state of the dungeon*. The rules for skills are described in Section 2.3.
Your tasks are as follows.

- Implement a BFS algorithm to find a sequence of actions for your character to reach a rune; and,
- Implement an A* algorithm to find the best sequence of actions for your character to reach a rune

Note: all coordinates below will be given in **matrix coordinates**. That means coordinates (x, y) refers to row x and column y. Refer to Project 1.1 file for more details.

2.2 Actions

There are 4 actions that your character can take: UP, DOWN, LEFT, and RIGHT. Such an action will bring your character one cell in that direction e.g. if your character starts from (1, 0), then UP will bring your character to (0, 0). You *cannot move onto a cell blocked by an obstacle*. Each action costs 4 HP.
Apart from the cost due to taking an action, there are also *costs due to encountering a creep*. Each creep that is encountered will cost 1 HP.
Example: suppose you start at position (1, 0) with 10 creeps, and you use UP to move to position (0, 0) with 20 creeps. Then, the total HP lost from taking an UP action is 4 + 20 = 24 HP.

2.3 Skills

You have two skills in your arsenal: a standard skill FLASH and an ultimate skill INVERSION.

2.3.1 FLASH

When cast, FLASH changes the next move's rule from ”move by 1 cell” to ”move until an obstacle or dungeon boundary is hit”. Movement cost is modified from 4 HP to 2 HP. Furthermore, only creeps in the final cell reached are counted in the HP calculation. An invocation of FLASH costs 10 HP and **does not change the position of the character**. This skill can only be cast at most j times, to be given in the input. It affects only the immediate action taken, and no other actions can be taken at the same time as its casting.
Examples:

1. Suppose you are at cell $(0, 0)$ in a dungeon with 1 row and 4 columns.
   - Without FLASH, you need 3 actions to reach $(0, 3)$, namely RIGHT $\rightarrow$ RIGHT $\rightarrow$ RIGHT. Total HP cost is $(4 + B) + (4 + C) + (4 + D)$, where B, C, D corresponds to the number of creeps at $(0, 1)$, $(0, 2)$, and $(0, 3)$ respectively.
   - With FLASH, you only need 1 FLASH invocation and 1 action, namely FLASH $\rightarrow$ RIGHT. Total HP cost is $10 + (2 + 2 + 2) + D$, where D is the number of creeps at $(3, 0)$.
   - Note the difference in cost computations when FLASH is used:
     - There is an additional cost of 10 HP. This corresponds to the invocation cost
     - All the 4s become 2s. This is because FLASH modifies action cost from 4 to 2
     - Only the final creep cost D is considered. This is because FLASH allows you to ignore the intermediate creeps B and C.
2. Suppose you are at cell $(0, 0)$ in a dungeon of width 8. Suppose cell $(0, 4)$ is an obstacle. Then:
   - FLASH $\rightarrow$ RIGHT will bring you to $(0, 3)$, since FLASH does not allow you to pass through obstacles.
   - FLASH $\rightarrow$ RIGHT cannot bring you to $(0, 2)$, since this violates the rule that FLASH causes movements to ”move until an obstacle ... is hit”.
   - FLASH $\rightarrow$ FLASH $\rightarrow$ RIGHT is strictly worse than FLASH $\rightarrow$ RIGHT, since the first FLASH incurs an extra 10 HP lost, decrements the remaining number of FLASH that can be cast left, and does not modify any action.
   - FLASH $\rightarrow$ INVERSION $\rightarrow$ RIGHT is strictly worse than INVERSION $\rightarrow$ RIGHT, since the FLASH cast does not actually modify any action while incurring 10 HP cost and decrementing the remaining number of FLASH left.

2.3.2 INVERSION

When cast, INVERSION allows you to reshuffle creeps in the dungeon. The number of creeps in any particular cell is modified using the rule:

$$num\_creeps := MAX\_NUM\_CREEP - num\_creeps \tag{1}$$

to each valid cell, where MAX_NUM_CREEP is the largest number of creeps in any of the cells in the dungeon *that is not occupied by obstacles*. INVERSION does not change character position and does not cost any HP when cast, however, it can only be cast at most once. The effect of an INVERSION is permanent.

Examples:

1. In a dungeon with 1 row and 3 columns, suppose cell $(0, 0)$ has 2 creeps, cell $(0, 1)$ has 8 creeps, and cell $(0, 2)$ has 10 creeps. Then, casting INVERSION will transform this into 8, 2, and 0 creeps respectively, since $MAX\_NUM\_CREEP = 10$. The best sequence of actions is thus INVERSION $\rightarrow$ RIGHT $\rightarrow$ RIGHT, which costs 0 + (4 + 2) + (4 + 0) = 10 HP lost.

   Other (worse) alternatives include:

   - RIGHT $\rightarrow$ RIGHT, which costs (4 + 8) + (4 + 10) = 26 HP lost
   - FLASH $\rightarrow$ RIGHT, which costs 10 + (2 + 2) + 10 = 20 HP lost
   - RIGHT $\rightarrow$ INVERSION $\rightarrow$ RIGHT, which costs (4 + 8) + 0 + (4 + 0) = 16 HP lost
   - INVERSION $\rightarrow$ FLASH $\rightarrow$ RIGHT, which costs 0 + 10 + (2 + 2) + 0 = 14 HP lost.

2.4 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:

- `cols`: The number of columns the dungeon has. Type is `int`.
- `rows`: The number of rows the dungeon has. Type is `int`.
- `obstacles`: The list of positions on the dungeon occupied by obstacles. Type is `List[Position]`.
- `creeps`: The list of positions on the dungeon with creeps. Type is `List[List[int]]`, where the inner list is of length 3. The 3 numbers correspond to `[x, y, num_creeps]` i.e. `[3, 0, 10]` means there are 10 creeps at $(3, 0)$.
- `start`: The starting position. Type is `Position`
- `goals`: The positions of the runes in the dungeons. Type is `List[Position]`
- `num_flash_left`: The maximum number of times FLASH can be cast. Type is `int`

Note: All positions that are not obstacles and are not listed in `creeps` have 0 creeps in them.

2.5 Requirements

You are to implement a function called `search` that **takes in** a dictionary as described in Section 2.4 and **returns a valid sequence of actions and skills** that, when used, will bring you to a given rune. In particular, you should return a `List[int]` representing the sequence of actions taken. Use the following encoding to represent your actions and skills as integers:

```
class Action(Enum):
    UP = 0
    DOWN = 1
    LEFT = 2
    RIGHT = 3
    FLASH = 4
    INVERSION = 5
```

For example, if you start at $(0, 0)$, then RIGHT to $(0, 1)$, then DOWN to $(1, 1)$, you should output `[3, 1]`.

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

1. No action can bring the character into any obstacles or out of the dungeon bounds
2. If there are no legal actions to be taken, return an empty list.

For A*, there is an additional requirement of having to output a sequence of actions that **causes the least HP lost**. Furthermore, the A* heuristic function **may not be** the zero heuristic.

3 Grading

3.1 Grading Rubrics (Total: 8 marks)

Requirements (Marks Allocated)	Total Marks
• Correct implementation of Breadth First Search Algorithm (2m). • Efficient implementation of Breadth First Search Algorithm (2m). • Correct implementation of A* Search Algorithm (2m). • Efficient implementation of A* Search Algorithm (2m).	8

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 sequence of actions within the given time limit. For A*, the returned path has an optimal cost. You have **passed** the test case.
- **Wrong Answer (WA):** You have either returned an invalid sequence of actions or a non-optimal sequence of actions (for A* 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.

You are **required** to implement the algorithms asked for. For example, when submitting for Breadth-First Search (BFS), you may not pass in Depth-First Search (DFS) or any other search algorithms in its place. This requirement will be **enforced** on the final submissions made in Canvas.

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.

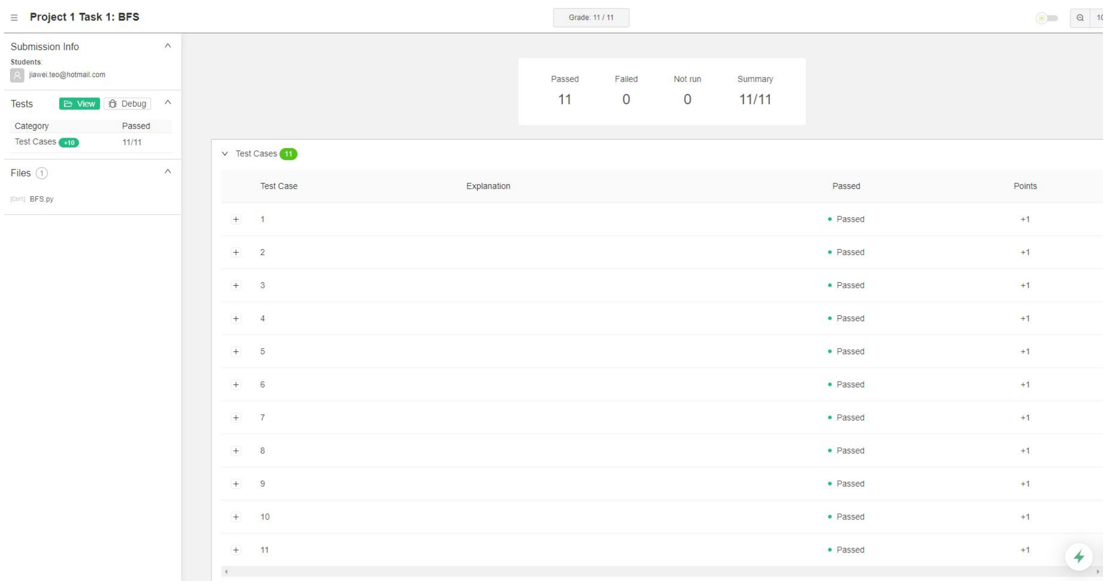
3.3.1 Using CodePost

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 (BFS/A*), click on "Upload assignment", and upload your Python file `bfs.py/astar.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:



Category	Passed
Test Cases +10	11/11

Passed	Failed	Not run	Summary
11	0	0	11/11

Test Case	Explanation	Passed	Points
1		● Passed	+1
2		● Passed	+1
3		● Passed	+1
4		● Passed	+1
5		● Passed	+1
6		● Passed	+1
7		● Passed	+1
8		● Passed	+1
9		● Passed	+1
10		● Passed	+1
11		● Passed	+1

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 in 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.

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: `bfs.py` and `astar.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** $\rightarrow$ **bfs.py, astar.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 1.2**

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 BFS

1. Correctness: each test case is worth 1 point. Total is 10 points.
2. Efficiency: Test cases 2, 5, 6, and 10 are worth 2 points each. Other test cases are each worth 1 point. Total is 14 points.

5.2.2 A*

1. Correctness: each test case is worth 1 point. Total is 10 points.
2. Efficiency: Test cases 2, 5, 6, and 10 are worth 2 points each. Other test cases are each worth 1 point. Total is 14 points.