Project Part A

Searching

COMP30024 Artificial Intelligence

16 March 2022

Overview

In this first part of the project, you will solve a simple search-based problem on the *Cachex* game board. Before you read this specification, please make sure you have carefully read the entire 'Rules for the Game of *Cachex*' document[1]. Although you won't be writing an agent to play the game just yet, you should be aiming to get very familiar with the board layout and corresponding hex coordinate system.

The aims for Project Part A are for you and your project partner to (1) refresh and extend your Python programming skills, (2) explore some of the algorithms you have encountered in lectures, and (3) become more familiar with the *Cachex* task environment. This is also a chance for you to develop fundamental Python tools for working with the game: Some of the functions and classes you create now may be helpful later (when you are building your full game-playing program for Part B of the project).

Path-finding on the *Cachex* board

You will be given an already populated *Cachex* board of arbitrary size n, along with a *start* coordinate and a *goal* coordinate. The task is to find the lowest *cost* path from the *start* coordinate to the *goal* coordinate, while avoiding already-occupied cells. To compute this path you are asked to use an **A* search** and design an **admissible heuristic** to optimise its performance. There are a number of assumptions you should make:

1. The *start* and *goal* cells will always be unoccupied (but any other cells may be occupied).
2. All given cell coordinates will be within the bounds of a board of size n. More precisely, for any given cell coordinate (r, q), $0 \leq r < n$ and $0 \leq q < n$. You may also assume $n \geq 1$.
3. The *cost* of a path is defined as the number of cells that form a continuous path from the *start* cell to the *goal* cell (including these cells).[2]
4. If there is a *tie*, that is, multiple minimal paths of the same *cost* exist on the same board configuration, any such path is a valid solution.
5. There may not always be a valid path from the *start* cell to the *goal* cell. It is possible for occupied cells to completely block all potential paths.

[1] Posted under Project Details page

[2] By this definition the 'shortest' possible path has *cost* 1. In this case, the *start* cell and *goal* cell would have the same coordinate. This also means a path of cost 0 is not possible; we will use this value to capture the case where there is no valid path in the first place.

Your tasks

Firstly, you will **design and implement a program** that finds the lowest cost path on a given *Cachex* board. That is, given a starting board configuration, the program must find a minimal, ordered sequence of cells from the *start* cell to the *goal* cell. Secondly, you will **write a brief report** discussing and analysing the strategy your program uses to solve this search problem. These tasks are described in detail in the following sections.

The program

You must create a program to solve this search problem in the form of a Python 3.6 **module**[3] called `search`. When executed, your program must read a JSON-formatted board configuration (along with a *start* coordinate and a *goal* coordinate), calculate a minimal sequence of cells denoting the lowest cost path, and print out this sequence of cells. The input and output format are specified below, along with the coordinate system we will use for both input and output.

Coordinate system

For input and output, we'll index the board's hexes with an *axial coordinate system*[4]. In this system each hex is addressed by a **row** (r) and **column** (q) pair, as shown in Figure 1.

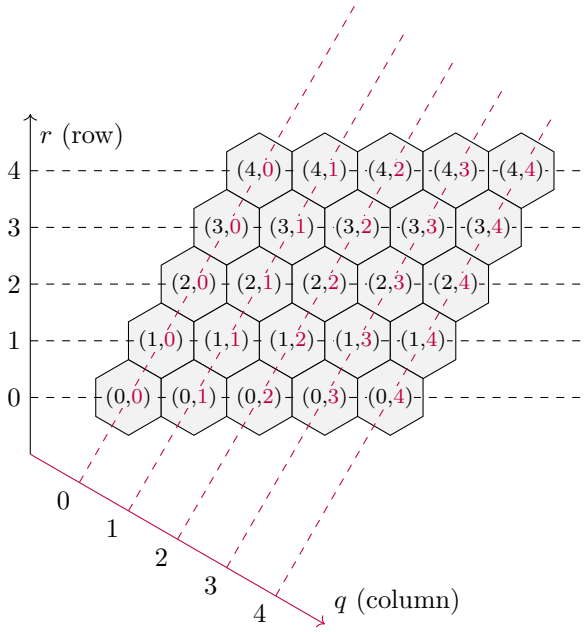


Figure 1: The r and q axes, with (r, q) indices for all hexes ($n = 5$).

[3]This module might be a *single file* called `search.py`, or it might include multiple files in a directory called `search/`, with an entry-point in `search/__main__.py`. Either way, you should be able to run the program using the command `python -m search` (followed by command-line arguments). See the skeleton code that will be provided soon on the Project Details page for an example of the correct structure.

[4]The following guide to hexagonal grids may prove useful: `redblobgames.com/grids/hexagons/`. In particular, see the 'coordinates' section for notes on a similar axial coordinates system. **Don't forget to acknowledge** any algorithms or code you use in your program.

Input format

Your program must read a board configuration from a JSON-formatted file. The name of the file will be given to your program as its first and only command-line argument.[5] The JSON file will contain a single dictionary (JSON object) with the following four entries:

- An entry with key `"n"` specifying the size of the board as an integer.
- An entry with key `"board"` specifying the positions of occupied cells as a list of token descriptions (described below).
- An entry with key `"start"` specifying the coordinate of the *start* cell as a list of the form `[`r`,`q`]`.
- An entry with key `"goal"` specifying the coordinate of the *goal* cell as a list of the form `[`r`,`q`]`.

Each token description is a list of the form `[`s`,`r`,`q`]` where s is a string representing the token's symbol (`"r"` for a Red player piece, `"b"` for a Blue player piece), and (r, q) are the axial coordinates of the token's hex (see Figure 1). Note that the actual colour of a piece does not really matter for this part of the project, but of course, it eventually will for Part B when you are actually playing the game.

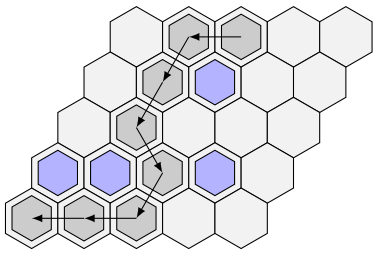
You may assume that (1) the input matches this format description exactly (we will not give you invalid JSON), (2) all coordinates are valid, as stated earlier (we will not give you coordinates outside of the board), and (3) there are no overlapping tokens (we will not give you a configuration with two tokens having the same coordinate).

Output format

Your program must print out the sequence of hex cell coordinates that form the lowest cost path from the *start* cell to the *goal* cell. More specifically, the *cost* of the solution sequence, and then the sequence itself, must be printed to *standard output*. The first line should be an **integer** denoting the cost, and subsequent lines should each be of the form `(r,q)`, denoting the ordered solution sequence of coordinates. If there is no valid solution, you should simply output **0** with no subsequent lines. You **must not** print anything to standard output other than what is described here!

```
{
    "n": 5,
    "board": [
        ["b", 1, 0],
        ["b", 1, 1],
        ["b", 1, 3],
        ["b", 3, 2]
    ],
    "start": [4, 2],
    "goal": [0, 0]
}
```

(a) An example input file.



(b) Starting board and a solution.

```
8
(4,2)
(4,1)
(3,1)
(2,1)
(1,2)
(0,2)
(0,1)
(0,0)
```

(c) Output for this solution.

[5] We recommended using Python's Standard Library module '`json`' for loading this input. An appropriate call would be '`with open(sys.argv[1]) as file:  data = json.load(file)`' (creating `data`, a dictionary version of the input).

The report

Finally, you must briefly discuss your approach to solving this problem in a separate file called `report.pdf` (to be submitted alongside your program). Your discussion should answer these three questions:

1. How did you implement the **A* search**? Discuss implementation details, including choice of relevant data structures, and analyse the time/space complexity of your solution.
2. What **heuristic** did you use and why? Show that it is admissible, and discuss the cost of computing the heuristic function, particularly in relation to the overall cost of the search.
3. *(Challenge.)* Suppose the search problem is extended such that you are allowed to use existing board pieces of a specific colour as part of your solution path at **no cost**. An optimal solution is now defined as a minimal subset of unoccupied cells that need to be 'captured' by this colour in order to form a continuous path from the *start* coordinate to the *goal* coordinate. How would you extend your current solution to handle this? Discuss whether the heuristic you used originally would still be admissible.

Your report can be written using any means but must be submitted as **a PDF document**. Your report should be between 0.5 and 2 pages in length, and must not be longer than 2 pages (excluding references, if any).

Assessment

Your team's Project Part A submission will be assessed out of 8 marks, and contribute 8% to your final score for the subject. Of these 8 marks:

- **5 marks** will be for the correctness of your program, based on running your program through a collection of automated test cases. The first mark is earned by correctly following the input and output format requirements. The remaining four marks are available for passing the tests themselves.

  The tests will run with **Python 3.6** on the **student Unix machines** (for example, `dimefox`[6]). There will be a **30 seconds time limit** per test case. Programs that do not run in this environment or do not complete within this time will be considered incorrect.

- **3 marks** will be for the clarity and accuracy of the discussion in your `report.pdf` file, with 1 mark allocated to each of the three questions listed above. A mark will be deducted if the report is longer than 2 pages or not a PDF document.

Your program should use **only standard Python libraries**, plus the optional third-party libraries **NumPy and SciPy** (these are the only libraries installed on `dimefox`). With acknowledgement, you may also include code from the AIMA textbook's Python library, where it is compatible with Python 3.6 and the above limited dependencies.

Academic integrity

Unfortunately, we regularly detect and investigate potential academic misconduct and sometimes this leads to formal disciplinary action from the university. Below are some guidelines on academic integrity for this project. Please refer to the university's academic integrity website (`academicintegrity.unimelb.edu.au`) or ask the teaching team, if you need further clarification.

[6] We strongly recommended that you test your program on `dimefox` before submission. Seek our help early if you cannot access `dimefox`. Note that Python 3.6 is not available on `dimefox` by default, but it can be used after running the command '`enable-python3`' (once per login).

1. You are encouraged to discuss ideas with your fellow students, but **it is not acceptable to share code between teams, nor to use code written by anyone else**. Do not show your code to another team or ask to see another team's code.

2. You are encouraged to use code-sharing/collaboration services, such as GitHub, *within* your team. However, **you must ensure that your code is never visible to students outside your team**. Set your online repository to 'private' mode, so that only your team members can access it.

3. You are encouraged to study additional resources to improve your Python skills. However, **any code adapted or included from an external source must be clearly acknowledged**. If you use code from a website, you should include a link to the source alongside the code. When you submit your assignment, you are claiming that the work is your own, except where explicitly acknowledged.

Submission

One submission is required from each team. That is, one team member is responsible for submitting all of the necessary files that make up your team's solution.

You must submit a single compressed archive file (e.g. a `.zip` or `.tar.gz` file) containing all files making up your solution via the 'Project Part A Submission' item in the 'Assessments' section of the LMS. This compressed file should contain all Python files required to run your program, along with your report.

The submission deadline is **11:00PM on Tuesday the 5$^{\text{th}}$ April, Melbourne time (AEST).**

Note that we have changed the submission that was previously advertised in the lectures as 30$^{\text{th}}$ March.

You may submit multiple times. We will mark the latest submission made by either member of your team unless we are advised otherwise. You may submit late. Late submissions will incur a penalty of **one mark per working day** (or part thereof) late.

Extensions

If you require an extension, please email the lecturers using the subject 'COMP30024 Extension Request' at the earliest possible opportunity. If you have a medical reason for your request, you will be asked to provide a medical certificate. Requests for extensions received after the deadline may be declined.