COGS 125 / CSE 175
Introduction to Artificial Intelligence
Specification for Programming Assignment #0

David C. Noelle

Due: 11:59 P.M. on Friday, September 29, 2023

Overview

This initial programming assignment has two primary learning goals. First, the assignment will provide you with an opportunity to exercise your knowledge of Python program generation in the PyCharm integrated development environment (IDE) that will be used throughout this course. In this way, this assignment will provide practice in some of the fundamental skills that will be needed to successfully complete future programming assignments. Second, this assignment will provide you with experience in implementing basic uninformed search algorithms, namely *breadth-first search* and *depth-first search*. A solid understanding of these basic approaches to search will form an important foundation for knowledge of the more advanced techniques to be covered in this class.

In summary, you will implement both *breadth-first search* and *depth-first search* algorithms in Python in the context of a simple shortest-path map search problem. These implementations will also support the optional checking for repeated states during search.

Submission for Evaluation

To complete this assignment, you must generate two Python script files: `bfs.py` and `dfs.py`. The first of these files should implement a breadth-first search algorithm, and the second should implement a depth-first search algorithm, as described below. These are the only two files that you should submit for evaluation.

To submit your completed assignment for evaluation, log onto the class site on CatCourses and navigate to the "Assignments" section. Then, locate "Programming Assignment #0" and select the option to submit your solution for this assignment. Provide your two program files as two separate attachments. Do not upload any other files as part of this submission. Comments to the teaching team should appear as header comments in your Python source code files.

Submissions must arrive by 11:59 P.M. on Friday, September 29th. Please allow time for potential system slowness immediately prior to this deadline. You may submit assignment solutions multiple times, and only the most recently submitted version will be evaluated. As discussed in the course syllabus, late assignments will not be evaluated and will receive *no credit*.

If your last submission for this assignment arrives by 11:59 P.M. on Tuesday, September 26th, you will receive a 10% bonus to the score that you receive for this assignment. This bonus is intended to encourage you to try to complete this assignment early.

Activities

You are to provide a Python function that implements the breadth-first search algorithm and another that implements the depth-first search algorithm. Your provided Python source code *must* be compatible with provided Python utility code which implements simple road maps, allowing your search algorithms to be used to find the shortest routes between locations on such maps. Indeed, your assignment solution will be evaluated by combining your submitted files with copies of the provided utility files and testing the resulting complete program against a variety of test cases. In other words, *your solution absolutely must work with the provided utilities, without any modifications to these provided files*.

More specifically, you are to provide a function called `BFS` in a source code file named "`bfs.py`", and you are to provide a function called `DFS` in a source code file named "`dfs.py`". The first of these functions is to implement a breadth-first search algorithm, and the second is to implement a depth-first search algorithm. Both functions must have the following features ...

- takes two arguments:
    1. `problem` — a `RouteProblem` object
    2. `repeat_check` — a boolean indicating if repeated state checking is to be done
- implements the pseudocode for generic search provided during class lectures
- deviates from this pseudocode only by performing repated state checking if and only if the `repeat_check` argument is `True`
- in particular, the goal test is performed on a search tree node just before it is expanded
- makes use of a `Frontier` object to maintain the search tree fringe
- returns a search tree node corresponding to a solution, or `None` if no solution is found

In general, your functions should allow the provided "`main.py`" script to output correct solutions (including path cost and expansion count statistics) for *any* map search problem provided as input to it. Note that this means that the functions that you implement should write *no output*, as this will clutter the output produced by the "`main.py`" script. If you include any statements that write output in your code (perhaps as tools for debugging) these should be removed prior to submitting your code files for evaluation. You may receive *no credit* for your submitted solution if it produces extraneous output.

The Python utility code that you are required to use is provided in a ZIP archive file called "`PA0.zip`" which is available in the "Files" section of the class CatCourses site, under "Programming Assignment #0". These utilities include:

- `RoadMap` class — This class encodes a simple road map, involving locations connected by road segments. Each location has a name and coordinates, which can be conceived as longitude and latitude. Each road segment represents a one-way connection from one location to another. Each road segment has a name and a cost of traversal.
- `RouteProblem` class — This class encapsulates a formal specification of a search problem. It includes a `RoadMap`, as well as a starting location and goal location. It also provides a goal test function called `is_goal`.
- `Node` class — This class implements a node in a search tree. Each node has a corresponding location, a parent node, and a road segment used to get from the location of the parent to the location of the node. (Note that the root of the search tree, corresponding to the starting location, has no parent or road segment.) Each node also tracks its own depth in the search tree, as well as the partial path cost from the root of the treee to the node. The class provides an `expand` function, which expands the node.
- `Frontier` class — This class implements the frontier, or fringe, of a search tree. The root node of a search tree is provided upon creation, initially populating the frontier with that one node. If the `stack` argument is `True` at the time of creation, the frontier is implemented as a stack. Otherwise, it is implemented as a queue. The class provides functions to `add` a node to the frontier and `pop` a node from the frontier, as well as testing if the frontier `is_empty` or if it `contains` a node with a matching location.

The contents of these utility files will be discussed during a laboratory session, and comments in these files should assist in your understanding of the provided code. Questions are welcome, however, and should be directed to the teaching team.

Your implementations of both breadth-first search and depth-first search should largely mirror the generic search algorithm presented during class lectures. Note that this algorithm deviates from the "`BREADTH-FIRST-SEARCH`" pseudocode provided in the course textbook, Russell & Norvig (2020) (Figure 3.9). Specifically, your code must test for goal attainment just prior to expanding a node (*not* just prior to insertion into the frontier). No repeated state checking should be done unless the `repeat_check` argument is `True`. When repeated state checking is being done, a child node should be discarded if its location matches that of a previously expanded node or of a node currently in the frontier.

The provided "`main.py`" script includes an example map, which you may use to perform initial tests of your code. It is important that your search functions perform correctly when given *any* possible map, however, so your code should be tested on additional maps of your own design that potentially contain unusual features. **It is possible for your code to contain serious errors but still perform well on the provided test example.**

Your submission will be evaluated primarily for accuracy, with efficiency being a secondary consideration. Your source code *will* be examined, however, and the readability and style of your implementation will have a substantial influence on how your assignment is evaluated. As a rough rule of thumb, consider the use of good software writing practices as accounting for approximately 10% to 20% of the value of this exercise. Please use the coding practices exemplified in the provided utility files as a guide to appropriate readability and style. Note also that, as discussed in the course syllabus, submissions that fail to run without crashing on the laboratory PyCharm IDE will not be evaluated and will receive *no credit*.

As for all assignments in this class, submitted solutions should reflect the understanding and effort of the individual student making the submission. **Not a single line of computer code should be shared between course participants. This is *not* a group assignment.** If there is ever any doubt concerning the propriety of a given interaction, it is the student's responsibility to approach the instructor and clarify the situation *prior* to the submission of work results. Also, helpful conversations with fellow students, or any other person (including members of the teaching team), should be explicitly mentioned in submitted assignments (e.g., in comments in the submitted source code files). These comments should also explicitly mention any written resources, including online resources, that were used to complete the exercise. Citations should clearly identify the source of any help received (e.g., "Dr. David Noelle" instead of "a member of the teaching team", "The Python Tutorial at `docs.python.org/3/tutorial/`" instead of "Python documentation"). **Failure to appropriately cite sources is called *plagiarism*, and it will not be tolerated!** Policy specifies that detected acts of academic dishonesty must result minimally with a zero score on the assignment, and it may result in a failing grade in the class. Please see the course syllabus for details.

To be clear, please note that all of the following conditions must hold for a submitted solution to receive *any* credit:

- solution submitted by the due date
- solution runs without crashing on the laboratory PyCharm IDE
- solution produces no extraneous output
- solution works with unmodified utility code, as provided
- solution does *not* include code written by another person (with or without modifications)
- solution explicitly cites all help received, whether from a person or some other source

While partial credit will be given for submissions that meet these conditions, failure to meet one or more of these conditions will result in *no credit* and, in the case of academic dishonesty, may result in a failing grade in the course.

The members of the teaching team stand ready to help you with the learning process embodied by this assignment. Please do not hesitate to request their assistance.

Generic Search

```
function SEARCH(problem) returns a solution node or failure
  node <- a node containing the initial state of problem
  if node contains a goal state of problem then return node
  initialize the frontier to contain node
  initialize the reached set to contain node
  while frontier is not empty
    node <- a leaf node removed from frontier
    if node contains a goal state of problem then return node
    expand node
    for each child of node
      add child to frontier
        and add child to the reached set,
        only if not already in the reached set
  return failure
```

This content is protected and may not be shared, uploaded, or distributed.