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

David C. Noelle

Due: 11:59 P.M. on Wednesday, September 26, 2018

Overview

This initial programming assignment has two primary learning goals. First, the assignment will provide you with an opportunity to refresh your knowledge of Java™ program generation in the Eclipse 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 Java™ 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 Java™ class files: `BFSearch.java` and `DFSearch.java`. 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 Java™ source code files.

Submissions must arrive by 11:59 P.M. on Wednesday, September 26th. Please allow time for potential system slowness immediately prior to this deadline. You may submit assignment solu-

tions multiple times, and only the most recently submitted version will be evaluated. As discussed in the course syllabus, late assignments will not be evaluated.

If your last submission for this assignment arrives by 11:59 P.M. on Monday, September 24th, 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 Java™ class that implements the breadth-first search algorithm and another that implements the depth-first search algorithm. Your provided Java™ source code must be compatible with a collection of provided Java™ classes that implement 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 class files with copies of the provided map utility class files and testing the resulting complete program against a variety of test cases. In other words, *your solution must work with the provided utilities, without any modifications to these provided files*.

More specifically, you are to provide a class called `BFSearch` in a source code file named "`BFSearch.java`", and you are to provide a class called `DFSearch` in a source code file named "`DFSearch.java`". The first of these classes is to implement a breadth-first search algorithm, and the second is to implement a depth-first search algorithm. Both classes must have the following features ...

- a constructor that takes four arguments:
  1. a complete Map object, encoding the map to be searched
  2. a String providing the name of the starting location
  3. a String providing the name of the destination location
  4. an integer depth limit for the search — if this depth is ever reached during a search (i.e., a node at this depth or deeper is considered for expansion), the search should be completely abandoned and `null` (indicating search failure) should be returned
- a method that actually performs the search, called `search`, with the following properties:
  - it takes a single boolean argument — if this argument is "true", then repeated state checking should be performed, otherwise no such checking should be done during search
  - it returns a Node object from the search tree that corresponds to the target destination, or `null` if no solution was found
- a public integer instance variable called `expansionCount` that contains the number of node expansions performed during the most recent call to the `search` method

In general, your classes should allow the `main` method in the provided `Pzero` class 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 member functions of classes that you implement should write *no output* to the standard output stream, as this will clutter the output produced by the `Pzero` class `main` method. 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. Your submitted solution may be marked down if it produces extraneous output.

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

- `Location` — This object encodes a location on the map. It is a "state" in the "state space" to be searched.
- `Road` — This object encodes a road segment from one location to another. Each `Location` records all of the road segments leading away from that location. This list of `Road` objects provides the "successor function" result for each "state".
- `Map` — This object gathers together all of the `Location` objects on the map. It provides utilities for reading map descriptions from files.
- `Node` — This object encodes a node in the search tree. Each `Node` object represents a `Location` in the context of a particular path from the starting point.
- `Frontier` — This object provides a simple implementation of queues and stacks containing `Node` objects. This can be used to keep track of the nodes in a search tree's "frontier" or "fringe".
- `Pzero` — This object provides a top-level driver that tests both breadth-first search and depth-first search, with repeated state checking both turned on and turned off. Your code must allow `Pzero` to produce correct output.

The contents of these Java™ utility files will be discussed during a course laboratory session, and inline 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 make use of the general search algorithm presented as "`GRAPH-SEARCH`" in the course textbook, Russell & Norvig (2010). Specifically, your code must test for goal attainment just prior to expanding a node (*not* just prior to insertion into the frontier), and repeated state checking must be performed as nodes are inserted into the frontier (*not* upon removal from the frontier). When repeated state checking is being done, a child node should be discarded if its state matches that of a previously expanded node or of a node currently in the frontier. Finally, note that the "`GRAPH-SEARCH`" algorithm will require a slight modification to allow for the *disabling* of repeated state checking, based on the boolean argument provided to the `search` function.

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 successfully compile under the laboratory Eclipse Java™ IDE (i.e., they produce "build errors") 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. 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 Oracle Java Technology Reference API Specifications at `www.oracle.com/technetwork/java/index-jsp-142903.html`" instead of "a Java web page"). Failure to appropriately cite sources is called *plagiarism*, and it will not be tolerated!

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.