UNIVERSITY OF SASKATCHEWAN

Department of Computer Science
176 Thorvaldson Building
110 Science Place, Saskatoon, SK, S7N 5C9, Canada
Telephine: (306) 966-4886, Facimile: (306) 966-4884

Assignment 1

AIMA Chapter 3: Solving Problems by Searching

Date Due: 1 October 2018, 11:59pm **Total Marks: 50**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- If you intend to use resources not supplied by the instructor, please check first. In any case, you must provide an attribution for any external resource (library, API, etc) you use. **You should not use any resource that substantially solves the problems in this assignment.** If there's any doubt, ask! No harm can come from asking, even if my answer is no.
- Each question indicates what to hand in.
- Do not submit folders, or zip files, even if you think it will help.
- **Assignments must be submitted to Moodle.**

Version History

- **30/09/2018**: Clarifications to evaluation, and what to hand in for all questions.
- **14/09/2018**: released to students

Overview

In this assignment we'll explore the ideas in AIMA Chapter 3 using a problem defined as follows.

Inverse Arithmetic Problem

. You are given a finite list of integers, L, and a target integer, T. The problem is to determine an integer arithmetic expression E, using some or all of the integers in L, so that if E is evaluated according to the rules of integer arithmetic, the answer is T.

Integer expressions will be constructed using only the operations $+$, $-$, $\times$, and integer division / (integer division means we ignore the remainder, and use only the quotient, e.g. $5/2 = 2$).

For example, the given information could be as follows:

- $L = [1, 2, 3, 4, 5, 7]$
- $T = 47$

There are several possible expressions we can construct:

- $((2 \times 4) \times 5) + 7$
- $((3 + 4) \times 7) - 2$
- $(7 \times 7) - 2$
- $(3 \times 4) + (5 \times 7)$

For pragmatic reasons, we will look for linear expressions only, e.g., $((2 \times 4) \times 5) + 7$, and we will not be trying to find non-linear expressions like $(3 \times 4) + (5 \times 7)$. Restricting the shape makes your job easier, and including non-linear expressions does not increase the insight you gain from this exercise. For this assignment, assume that each value from L can be used at most once, but the operators can be used as many times as you need.

We're interested in finding the expression that uses the fewest operations. Most problems will not have a single unique shortest expressions, and almost all problems will have many much longer expressions.

Programming

You'll have to do some programming here, and I'll provide strong guidance about how to get things organized. You can use any programming language you like, as long as it's Python, Java, or C/C++. If you want to use a different language, please let me know before proceeding.

The arithmetic expressions are probably best represented by text strings. In Python, you have access to the function `eval()`, which is quite powerful, but we can use it to calculate the value of any expression represented as a string. If you're using Java, you are permitted to use a third-party library to evaluate expression strings; there are a few I found by Googling.

You'll also be running your programs on a collection of examples. You'll be expected to report on aspects of this exercise.

Execution instructions

The markers may or may not wish to run your program, to verify your results. To help them, you should provide brief instructions on what to do to get your program running. Include:

- Programming language used (including version, e.g., Java 8 or Python 3)
- A simple example of compiling and/or running the code from a UNIX shell would be best.

Keep it brief, and name it with the question number as the following example: `a1q2_EXECUTION.txt`. If your assignment uses third party libraries, they have to be included in your submission.

Question 1 (10 points):

Purpose: To practice specifying problems, and to lay the groundwork for later questions.

Degree of Difficulty: Easy

AIMA Chapter(s): 3.1, 3.2

Specify the problem. Define:

1. The initial state.
2. The goal state.
3. The actions that can be applied to a given state.
4. The result of applying an action to a given state.
5. The path cost.

You should draw at least a portion of the problem space defined by the initial state and the actions in the specification, just to get a sense of what you're dealing with. But you don't need to hand your drawing in.

Implement your specification using your programming language of choice. In particular, implement:

1. Problem State class
2. Problem class

Be sure to test your implementations thoroughly.

What to Hand In

- A file named `a1q1.txt` containing your definitions.
- A file named `a1q1.LANG` containing your implementation. Use the file extension appropriate for your choice of programming language, e.g., `.py` or `.java`. **If you have more than one file to submit, you should use `a1q1_` to prefix your filenames.**

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

- 5 marks: You defined the initial state, the goal state, the actions, and results, and the path cost. Your file `a1q1.txt` describes:
  - The components of the data structure (or class) that stores the problem state.
  - The method `is_goal()`.
  - The method `actions()`, and how you represented your actions.
  - The method `result()`.
  - The path cost of sequences of actions (or sequences of states).

  You may copy/paste block comments from your source code, if you've documented it well. Point-form is fine. Full marks will be given if the description is short, and if it is complete.
- 3 marks: Your implementation follows the suggested interface guidelines. Specifically:
  - You have a function or method `is_goal(state)` that returns a Boolean.
  - You have a function or method `actions(state)` that returns a list of actions.
  - You have a function or method `result(state, action)` that returns a new state.
- 2 marks: Your implementation is well-documented. Specifically:
  - Every function or method has at least a brief description of its purpose.
  - Your name, NSID, and student number are in the file.

Question 2 (10 points):

Purpose: To build and apply uninformed search algorithms to the problem.

Degree of Difficulty: Moderate

AIMA Chapter(s): 3.3, 3.4

Using the guidelines provided in the textbook, and in lectures 03 and 04, implement TreeSearch, with the following search strategies:

- Breadth-first
- Depth-first
- Depth-limited
- Iterative-deepening

Remember that the first two strategies differ only in the Frontier ADT implementation (queue, stack). Depth-limited search discards search nodes that exceed the depth-limit; iterative-deepening calls depth-limited search with increasing depth limits until a solution is found.

Test your algorithms by applying them to the example problems found in the file `simple_examples.txt`. Question 5 gets you to put your algorithms to a more intensive test. Each line in the example files gives a target followed by the list of integers (separated by spaces).

Demonstrate by copy/paste from your output that your program generates solutions to a few interesting problems, showing that the expression returned does evaluate to the target.

What to Hand In

- A file named `a1q2_EXECUTION.txt` containing brief instructions for compiling and/or running your code. See page 1.
- A file named `a1q2.txt` containing a demonstration of the output of your program, which is copy/paste from a console, or output file. You only need to show a few examples for each search strategy. If this file is missing, the marker will assume your implementation is incomplete or not working.
- A file named `a1q2.LANG` containing your implementation of the search algorithms. Use the file extension appropriate for your choice of programming language, e.g., `.py` or `.java`. You may submit multiple files, provided that the filenames begin with `a1q2_`

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

- 2 marks: Your file `a1q2.txt` contains a few examples of output from your program.
- 4 marks: Your implementation follows the suggested interface guidelines. Specifically:
  - You implemented tree search with a FIFO queue, i.e., breadth-first search.
  - You implemented tree search with a LIFO queue, i.e., depth-first search.
  - You implemented a version of depth-first search that limits the depth of search , i.e., depth-limited search.
  - You implemented iterative-deepening search using repeated calls to a depth-limited search.

  The implementations may be separate functions, or they can be implemented by reusing a generalized treesearch algorithm.
- 2 marks: Your treesearch algorithm(s) use the methods `is_goal()`, `actions()`, and `result()` to interface with the Problem class.
- 2 marks: Your implementation is well-documented. Specifically:
  - Every function or method has at least a brief description of its purpose.
  - Your name, NSID, and student number are in the file.

Question 3 (10 points):

Purpose: To practice the design of heuristic evaluation functions for informed search.

Degree of Difficulty: Moderate

AIMA Chapter(s): 3.5, 3.6

Recall that a heuristic evaluation function is a measure attached to a given problem state; this measure estimates the cost of the path from the problem state to the goal state. In the text, it's represented by the function $h(n)$, where confusingly n is a search node containing a problem state. Recall also the lesson in AIMA Chapter 3, which is that one way to derive a heuristic function is to relax one or more constraints in the problem, and use the true future cost in the relaxed problem as an estimate for the future cost in the problem you're trying to solve.

Design a heuristic for the Inverse Arithmetic Problem. Give pseudo-code for calculating it, based on your problem state, and the goal state. In your description, explain how your function comes from a relaxed problem, and give two distinct examples of the heuristic evaluation function applied to a few simple problem states. Address the question of whether or not your heuristic function is admissible.

Notes:

- It's more fun if your heuristic is good, but it doesn't have to be good to get full marks. Make sure you can make a case that it estimates cost to goal, even if not very well.
- You are not required to find an admissible heuristic, but you need to know whether it is or not!

What to Hand In

- A file named `a1q3.txt` containing a brief description of your heuristic evaluation function, mentioning the points described above.

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

- 2 marks: Your description is clear and generally well-written.
- 3 marks: Your pseudocode description gives a clear picture of the heuristic function.
- 2 marks: You gave at least two examples of the heuristic function to real problem states.
- 3 marks: You addressed the question of admissibility clearly demonstrating that you understand the concept. It does not matter if your heuristic function is or is not admissible; just be sure you know which!

Question 4 (10 points):

Purpose: To build and apply informed search algorithms to the problem.

Degree of Difficulty: Easy

AIMA Chapter(s): 3.5

Using the guidelines provided in the textbook, and in lecture, implement TreeSearch, with the following search strategies:

- Uniform-cost search (UCS)
- Greedy best-first search (GBFS)
- A* Search

Remember that these three strategies differ only the measure that the frontier ADT uses to organize the search nodes. UCS uses $g(n)$ only; GBFS uses $h(n)$ only; A* uses $g(n) + h(n)$; here, $g(n)$ is the path cost from the initial state to the state contained in node n, and $h(n)$ is your heuristic evaluation function from the previous question.

Test your algorithms by applying them to the example problems found in the file `simple_examples.txt`. Each line in the example files gives a target followed by the list of integers (separated by spaces).

Demonstrate by copy/paste from your output that your program generates solutions to a few interesting problems, showing that the expression returned does evaluate to the target.

What to Hand In

- A file named `a1q4_EXECUTION.txt` containing brief instructions for compiling and/or running your code. See page 1.
- A file named `a1q4.txt` containing a demonstration of the output of your program, which is copy/paste from a console, or output file. You only need to show a few examples for each search strategy. If this file is missing, the marker will assume your implementation is incomplete or not working.
- A file named `a1q4.LANG` containing your implementation of the search algorithms. Use the file extension appropriate for your choice of programming language, e.g., `.py` or `.java`. You may submit multiple files, provided that they filenames begin with `a1q4_`

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

- 2 marks: Your file `a1q4.txt` contains a few examples of output from your program.
- 4 marks: Your implementation follows the suggested interface guidelines. Specifically:
  - You implemented tree search with a priority queue.
  - You implemented UCS using treesearch with a priority queue, taking path-cost $g(n)$ to order the priority queue.
  - You implemented UCS using treesearch with a priority queue, taking estimated cost to goal $h(n)$ to order the priority queue.
  - You implemented UCS using treesearch with a priority queue, taking estimated solution cost $f(n) = g(n) + h(n)$ to order the priority queue.

  The implementations may be separate functions, or they can be implemented by reusing a generalized treesearch algorithm.
- 2 marks: Your treesearch algorithm(s) use the methods `is_goal()`, `actions()`, and `result()` to interface with the Problem class.
- 2 marks: Your implementation is well-documented. Specifically:
  - Every function or method has at least a brief description of its purpose.
  - Your name, NSID, and student number are in the file.

Question 5 (10 points):

Purpose: To apply your algorithms to more difficult problems, and draw conclusions from the results.

Degree of Difficulty: Moderate

AIMA Chapter(s): 3 (all)

In the files `moderate_examples.txt` and `harder_examples.txt` you'll find problems that should challenge some of your implementations, and will give you an idea about how well your heuristic evaluation function (Question 3) works.

Run your algorithms (uninformed and informed) on these problems to gauge the quality of your implementations. Give a report on what you found. You might discuss all or any of the following criteria for empirical evaluation:

- Which algorithms used the most time to solve the problems?
- Which algorithms used the most memory to solve the problems?
- How often did your algorithms find an optimal solution?
- Did any algorithms run so long that you terminated them before they returned an answer? If so, which algorithms did you terminate most frequently?

To answer this question, you could consider any of the following modifications to your search algorithms:

- Add code to time the search. Use a clock twice: just before starting the search loop, and just before you return a solution.
- Add a time threshold to your search algorithms, to terminate long-running searches. Set the threshold to short values until you are sure everything is working, and then set it longer to collect your data.
- Count the number of nodes that your search generates, and the maximum size of your frontier over the course of solving a single problem. Both of these statistics can be insightful!
- Don't calculate average times (or memory) across the set of problems. An average isn't really meaningful here, because the variation in problem difficulty is too high. Instead, count the number of times one algorithm is faster than another, and by how much.
- How well does your heuristic function from Question 3 work, compared to uninformed algorithms?

Comment on your findings. Include tables, plots, etc to support your findings.

What to Hand In

- A file named `a1q5.txt` (other formats, DOC, DOCX, PDF, are acceptable) containing a discussion of your findings in this question.

Be sure to include your name, NSID, student number, and course number at the top of all documents.

Evaluation

- 5 marks: Your description is clear and generally well-written.
- 5 marks: Your description is based on data (tables, plots), and not simply vague impressions you obtained while coding.
  - You summarized the behaviour of the different uninformed and informed search strategies on the collections of example problems provided.
  - Your summary mentioned issues related to completeness, optimality, as well as performance in terms of time (clock, or nodes expanded), and memory (maximum size of the frontier).
  - Your description included a discussion of the quality of your heuristic function from Question 3, by comparing your informed results to the uninformed results.

Extra work for the ambitious

1. Implement the GraphSearch algorithm, and compare your results to the TreeSearch algorithm results.
2. Drop the assumption that a number can be used at most once. Ask Mike for example problems that are created by allowing multiple uses of some numbers. Compare your results to the other variation.