9/21/2015 2:44 PM

CSCE 625
Programing Assignment #2
due: Tues, Oct 6, 2015 (by start of class)

Objective

The overall goal of this assignment is to implement A* search and use it to solve a classic AI search problem: the "Blocksworld". This problem involves stacking blocks into a target configuration. You are to write a program to solve random instances of this problem using your own implementation of A* search. The focus of the project, however, is to use your intuitions to develop a *heuristic* that will make solving various instances of this problem efficient.

This project will build on code you wrote for programming assignment 1. Specifically, you will use a queue-based search function, where the *priority_queue* is kept sorted based on f(n)=g(n)+h(n). g(n) is just the path length from the root to the current node. **The main focus will be on developing a heuristic function**, h(n), for this domain that estimates the distance to the goal (number of moves to solve the problem) and testing how it affects the efficiency of the search.

Description of the problem

This task involves stacking blocks. The blocks are labeled by letters. Here is an example random state and the goal state for a problem with 5 blocks on 3 stacks. (**written sideways, so the bottom of each stack is on the left**)

```
examples random state:
1 |  D
2 |  C  A
3 |  B  E

goal state:
1 |  A  B  C  D  E
2 |
3 |
```

The state can be represented as list of lists, or an array. However, you should make your code flexible so that you **can test it on different numbers of blocks and stacks** (as input parameters). The initial state can be generated by just randomly assigning blocks to stacks. The goal state is always defined to have all the blocks in order on stack 1. *The operator in this problem can pick up only the top block on each stack and move it to the top of any other stack.*

This problem becomes harder as the number of blocks scales up, and cannot be solved effectively with BFS (though you can try it for small numbers of blocks to compare). You will need A* search coupled with a heuristic to find solutions efficiently. The default heuristic (call it h_0) can be taken to be the "number of blocks out of place". **Your goal is to define a better heuristic** that will enable your algorithm to find solutions faster (with fewer goal tests), and thus solve larger problems (with more blocks).

As with Assignment 1, you should turn in a table with some measurements showing how your heuristic performs on different problems. You should evaluate your heuristic with several different problems spanning a range of difficulty. For example, start with 5 blocks and 3 stacks, then try 6 to 10 blocks to see how **scalable** your performance is. Then try expanding the number of stacks from 3 to 7, to see if this makes the problem easier or harder. See if you can solve "hard" problems, like with 10 blocks on 5 stacks, as shown below (the solution has 18 steps):

```
1 |  D
2 |  E   F   I   J
3 |  B   G
4 |  C   H
5 |  A
```

Your heuristic does not have to be provably admissible, though it will be more efficient the closer to admissible it is.

Implementation

You can use any programming language you like, but you will have to make sure the TA can compile and run it for grading.

You will need to write a new Node class for representing states in the Blockworld, and a successor() function for generating moves (swapping blocks from the tops of stacks to other stacks).

Make your implementation flexible so it can solve problems with an **arbitrary number of blocks and an arbitrary number of stacks** (e.g. provided as command-line arguments).

You will have to write a "problem generator" that generates random initial states for testing. This can be done by starting from the goal state and performing a sequence of random moves to "scramble" it. You can assume the goal will always be the same - to get all the blocks in consecutive order on stack 1.

As before, you will have to have a method for checking for *visited states* and handling them appropriately. In this case, if you find a shorter path to a node you have visited before, you should expand it rather than discarding it.

You might want to set an *upper-limit* on the number of goal tests for convenience. (If you do so, be sure to report the number of 'failures' with your statistics.)

What to Turn in

- You will submit your code for testing using the web-based CSCE *turnin* facility, which is described here: https://wiki.cse.tamu.edu/index.php/Turning_in_Assignments_on_CSNet
- You should include a Word document with instructions on how to compile and run your program, along with a transcript that shows **example program traces** (transcripts) for your A* search.
- You should also include a **table of performance metric**s for things such as number of iterations (goal tests), maximum queue size, mean solution path length, etc., either for 5 selected block-stacking problems, or averaged over at least 5 randomly chosen start/goal combinations. Also include metrics for the simple heuristic (h_0, number blocks out of place), for comparison.
- Include a **written description of how your heuristic works**, or what it is based on, or the logic behind it. Briefly address whether it is **admissible** or not (an informal argument will suffice).

```
> blocksworld 5 3

initial state:
1 |  B
2 |  C   E
3 |  A   D
iter=0, queue=0, f=g+h=5, depth=0
iter=1, queue=15, f=g+h=6, depth=1
iter=2, queue=27, f=g+h=6, depth=1
iter=3, queue=34, f=g+h=6, depth=1
...
success! depth=10, total_goal_tests=2339, max_queue_size=644
solution path:
1 |  B
2 |  C   E
3 |  A   D

1 |
2 |  C   E   B
3 |  A   D

1 |
2 |  C   E   B   D
3 |  A

1 |  A
2 |  C   E   B   D
3 |

1 |  A
2 |  C   E   B
3 |  D

1 |  A   B
2 |  C   E
3 |  D

1 |  A   B
2 |  C
3 |  D   E

1 |  A   B   C
2 |
3 |  D   E

1 |  A   B   C
2 |  E
3 |  D

1 |  A   B   C   D
2 |  E
3 |

1 |  A   B   C   D   E
2 |
3 |
```