CSE 4301/5290 Homework 2

Due: Sep 30, Wed, 5pm
Submit Server: class = ai , assignment = hw2

For programming problems (lisp/java/c/c++/python):

- Submit:
  - all files that are needed to compile and run (and executable)
  - README.txt with compilation and run instructions
- Your program should compile and run on `code.fit.edu` (Linux, remote access via ssh)

---

1. Q3.6b, p113, 3Ed (Q3.7b, p90, 2Ed). Consider a 10x10x8 (WxDxH) grid/discrete world. (a) Define states as a set of variables and their possible values. (b) Derive the number of possible states. (c) Successor function: for each action, discuss the input/parent and output/child state. (d) Define goal test with respect to states in (a).

2. Consider Deep Blue can evaluate 200 million positions a second. Assume at each move, a pawn can go to 2 possible positions, a rook 14, a knight 8, a bishop 14, a queen 28, and a king 8. Each side has 8 pawns, 2 rooks, 2 knights, 2 bishops, a queen and a king. Under standard regulations, each side makes 40 moves within the first 2 hours (or 3 minutes a move on the average).

   (a) Using the breadth-first search algorithm, how many levels can Deep Blue evaluate (visit) before each move (in 3 minutes)?

   (b) To examine 20 levels in 3 minutes, how many positions Deep Blue needs to evaluate (visit) in a second?

3. Simplified from Q3.6d, p114, 3Ed (Q3.7d, p90, 2Ed). There are two jugs, one can hold 3 and the other 5 gallons of water. Allowed actions with the jugs: fill one from the faucet, empty one to the drain, and pour from one to the other until either the receiving jug is full or the pouring jug is empty. The objective is to devise a sequence of actions that will produce 4 gallons of water in the larger jug. Only integer values of water are used.

   (a) Explain and derive the number of possible states.

   (b) Devise and explain an *admissible* heuristic function (h) [not the trivial $h(n) = 0$]. The cost of an action is defined as 1 unit for performing the action, an additional 1 unit for "moving" each gallon of water (fill, empty, pour), and an additional 1 unit for wasting each gallon of water (empty). The path cost (g) is the sum of the cost of all the actions.

   (c) For each of these algorithms:

      i. breadth-first search,
      ii. depth-first search,
      iii. uniform-cost search,
      iv. greedy search, and
      v. A*,

      assume both jugs are initially empty, construct a search tree, and provide:

      i. the order of nodes visited with their cost values
      ii. the solution path in sequence of actions.

      Use your heuristic function h in Part b. The "left to right" order of actions is fill-small (FS), fill-large (FL), empty-small (ES), empty-large (EL), pour-small-to-large (PSL), pour-large-to-small (PLS).

   (d) Programming (stated in lisp):

```
;prints the states while they are *visited* (not generated),
;  total number of visited states, and cost of solution path
;returns the solution path in a sequence of actions in a list
(defun general-search (insert-fn initial-state goal-test-fn
              successor-fn step-cost-fn heuristic-fn) ... )

;returns t if the node is a goal state
(defun jug-goal-test (node) ...)

;returns a list of legal successor nodes (generated dynamically)
;heuristic-fn could be nil
;the "left to right" order of actions is: ...
(defun jug-successor (node step-cost-fn heuristic-fn) ...)

;returns the step cost of applying an action at a certain node
(defun jug-step-cost (node action) ...)

;returns the (admissible) heuristic value of node (Part b)
(defun jug-heuristic (node) ...)

;returns the updated frontier with the successors inserted into
;the frontier according to the UCS algorithm
(defun ucs-insert (frontier successors) ...)

;returns the updated frontier with the successors inserted into
;the frontier according to the A* algorithm
(defun astar-insert (frontier successors) ...)
```

   For testing different initial states, provide in Lisp:

```
(defun test-jug (jug-init-state alg)
  (cond
   ((equal alg 'ucs)
    (general-search #'ucs-insert jug-init-state #'jug-goal-test
                    #'jug-successor #'jug-step-cost nil))
   ((equal alg 'astar)
    (general-search #'astar-insert jug-init-state #'jug-goal-test
                    #'jug-successor #'jug-step-cost
                    #'jug-heuristic))
   (t (format t "Error: unknown algorithm ~a~%" alg))
  )
)
```

   or in java/c/c++/python 3 command-line arguments: small-start, large-start, alg; for exmaple: `java TestJug 0 0 astar` and print the solution path returned by `general-search`.

CSE 5290 only

4. Explain and derive the number of *reachable* states in Problem 3 with both jugs empty initially.

5. Q3.9, p115, 3Ed (Q3.9, p90, 2Ed). Missionaries and Cannibals (MC) problem. Name the banks of the river as left and right, and the missionaries and cannibals are initially on the left bank.

   (a) Q3.9a. Cost: one unit to move a missionary and two units to move a cannibal on the boat from one bank to the other.

   (b) Programming: Using the same `general-search` in Problem 3d, add `mc-` functions that can solve the MC problem using UCS and A* search. For A*, describe why your heuristic function is *admissible* in the comments. For testing different initial states, provide the `test-mc` function (similar to `test-jug`). Describe and give an example of `test-mc` parameters (Lisp) or command-line arugments (java,c,c++/python) in the comments.