Homework 2
Computer Science
B351 Spring 2017
Prof. M.M. Dalkilic

James Gregory

February 3, 2017

All the work herein is mine.

Introduction

The aim of this homework is to get you acquianted with problem solving and the steps (Real World $\rightarrow$ Concept $\rightarrow$ Logic $\rightarrow$ Implementation). You will turn-in four files

- A *pdf with the written answers called `h2.pdf`
- A Python script called `rv1.py`
- A Python script called `rv2.py`
- A Python script called `rpsg.py` for rock-paper-scissors.

. If you've attempted extra credit, add the comment #ExtraCredit to the programs and ExtraCredit to the homework near the top so it's visible and obvious. I am providing this LaTeX document for you to freely use as well. Please enjoy this homework and ask yourself what interests you and then how can you add that interest to it! Finally, each homework question is worth 100 points.

Homework Questions

1. Problem 3.10 (p. 115) in the text.

   State: a configuration a problem can exist in.
   State Space: all of the possible configurations that can exist for a given problem.
   Search Tree: a collection of Search Nodes which begins with a root node, and ends with leaf nodes (nodes with all null children).
   Search Node: an object containing data and a collection of child nodes. Some, or all, children can be null.
   Goal: a State within a State Space that is closest to a user-defined "ideal."
   Action: choosing which child Node to move to from a parent Node.
   Transition Model: a collection of resulting States from taking each possible Action from a given State.
   Branching Factor: the number of children each State in a State Space has.

2. Problem 3.18 (p. 117) in the text.

   An iterative deepening search will perform much worse than a depth-first search in a state space with a branching factor greater than 1, where all goal nodes have equal, large heights and a goal node exists far to the left.

3. The text (page 95) describes consistency as:

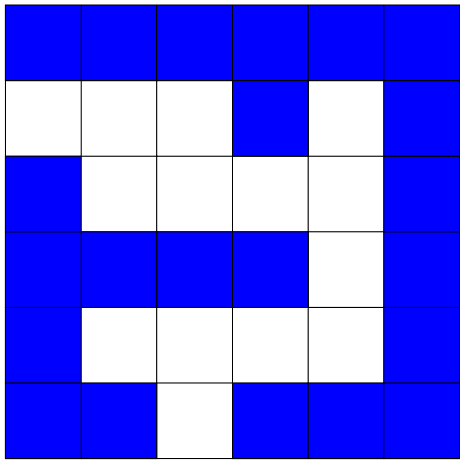
   $$h(n) \quad \leq c(n, a, n') + h(n')$$

   for state n, its successor n' and action a. For $G = (\{A, B, C\}, \{(A, B), (A, C), (B, C)\})$, $Cost = \{((A, B), 2), ((A, C), 5), ((B, C), 1)\}$, and $h(A) = 1, h(B) = 4, h(C) = 3$. Is this consistent?

   If G is a directed graph, where each $u \in E(u, v)$ is the parent, this heuristic is consistent:
   $h(A) \leq c(A, B) + h(B) \equiv 1 \leq 2 + 4$
   $h(A) \leq c(A, C) + h(C) \equiv 1 \leq 5 + 3$
   $h(B) \leq c(B, C) + h(C) \equiv 4 \leq 1 + 3$

   If G is an undirected graph, this heuristic is not consistent because:
   $h(B) > c(A, B) + h(A) \equiv 4 > 2 + 1$

4. Assume you're programming a robot named R to navigate a 2D surface. The robot can only move forward a single step to an adjacent square (not diagonally), but can also rotate ± 90 degrees. R has a single sensor on its front that determines if there is an obstruction, perhaps a wall, is in its path. Your task is to read in a 2D plan and starting at location from the southmost (bottom) side, navigate to another side. The plan below has an opening at (3,1). *One* path is: (3,1), (3,2), . . ., (3,5), (2,5), (1,5). If R is at (4,2) facing north, then its sensor would return 1. If R is at (4,2) and facing east, its sensor would return 0. If R is at (2,2) facing west, to move to (3,2), rotate(90), rotate(90), step. You can *start* R on any available open square on the bottom – you'll have to decide what direction R is facing. The plan is encoded as an array of ones and zeros. The plan below:



would be encoded as:

```
111111
000101
100001
111101
100001
110111
```

(a) Given a floor plan `f.txt` (read in the file), return True and the series of instructions needed to navigate R if there is a path and False otherwise. Name this program `rv1.py`.

see rv1.py

(b) Improve R's programming by returning the *shortest* path if it exists. Name this program `rv2.py`.

see rv2.py

(c) Discuss your search techniques in both solutions. State explicitly your $\hat{h}, \hat{g}, \hat{f}$.

In both rv1.py and rv2.py, my search algorithm starts by finding all possible exit points. Then, for each (exit point, start point) pair, it scores the exit 0, scores all adjacent squares 1, all squares adjacent to those 2, etc. until it finds the start. Then by starting at the starting location (which will have the highest score), the algorithm simply follows the adjacent square with the smallest score until it reaches the exit.
The difference between the two programs is the number of starting positions checked. In rv1.py, only paths connected to the first start located are found. In rv2.py, paths connected to each possible start are found. Both programs return the shortest path they find.
In terms of heuristics, mine simply scored each square by the number of moves it was from the currently focussed exit by scoring each child of each square 1 more than its parent, starting from the focussed exit, and without rescoring squares.

5. Extend Rock/Paper/Scissors from the last assignment that has the computer playing a human. You'll additionally have \$100 dollars worth of \$1 chips. *Before* you show your selection, you must place a wager (at least \$1). Keep the computer's strategy uniform and independent for both how it plays and how it bets. The maximum amount of chips that can be wagered is $\min\{c, h\}$ where c, h are the counts of computer and human chips respectively. Compare this R/P/S with your earlier version and discuss. Name this program `rpsg.py`.

In the old version of R/P/S, we kept track of the number of game played and won by each player, and then calculated the win percentages of each player. We also allowed the user

to choose the number of games to be played. In this newer version, we keep track of the amount of money each player has, ask the human for a bet, select a random, legal bet for the computer player, then force each player to bet an amount equal to the greater of the two bets. This version is also played until one player runs out of money, and the overall winner is the player who doesn't.
The two versions retain the same chance of winning each individual round for each player. Because the true winner newer version is the player who still has money at the end, however, the chances of winning the overall games are very different. By adding betting to the game, the state space of even something as simple as R/P/S is made extremely large. Without knowing exactly what each player will bet at each round, it is all but impossible to determine which player is most likely to win from the beginning of the game (assuming each player starts with the same amount of money).