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

Jonathon Cooke-Akaiwa

February 3, 2017

All the work herein is mine.

Homework Questions

1. A **state** is a situation we can find ourselves in.
   A **state space** is a graph made up of all states. These states are stored as nodes in the graph. The connections between nodes are actions that lead to each state.
   A **search tree** is a graph with a root node and no undirected loops. Any children in the tree are reachable by their parent through an action.
   A **search node** is a node in a search tree.
   A **goal** is a state we are trying to reach. An **action** is something that can be performed. These actions link nodes in a graph to each other making them reachable.
   A **successor function** is a function that returns the potential actions and states that those actions link to for a given state. The returned states are immediately reachable from the provided state.
   The **branching factor** is used with search trees. The branching factor is the number of possible actions available from the current state.

2. If a search space contains only nodes with a single successor, a DFS seach will find the goal in $O(n)$ steps. In this situation however, performing iterative deepening will seach one step at a time, increasing its search depth iteratively for a total of $O(n^2)$ steps.

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 $h(A) = 1, h(B) = 4, h(C) = 3$. This is consistent for the graph provided. Looking at each of the transitions individually:

$A \to B$	$A \to C$	$B \to C$
$h(A) \leq c(A,B) + h(B)$	$h(A) \leq c(A,C) + h(C)$	$h(B) \leq c(B,C) + h(C)$
$1 \leq 2 + 4$	$1 \leq 5 + 3$	$4 \leq 1 + 3$
$1 \leq 6$✓	$1 \leq 8$✓	$4 \leq 4$✓

   We can see that our consistency formula holds for the provided dictionary of transitions. Thus our graph is consistent.

4. (a) See file `rv1.py`. This program performs a DFS search until an exit is found, upon finding an exit, the program returns True and the path it has explored to reach that point. If multiple exits exist, the program returns when it encounters the first exit. This is not guaranteed to be the closest exit. If no exit exists the program returns False. The Maze is constructed using a seperate file `Maze.py`. This file contains a class Maze that contains funtions to facilitate exploring the maze. This was also created to prevent duplication of code for the next problem.

   (b) I have used my Maze class as explained above as a base for finding the shortest. This class has a function that will perform a dfs search on the provided maze and return any exits it finds. Using the a* algorithm, for each provided exit cell a search will be performed. Finally only the shortest of each of these paths will be returned as the optimal path. When performing a* we are using a function that returns the distance between two points as our heuristic or $\hat{h}$. This $\hat{h}$ is an expected distance to the end. In this case it would be the minimum distance between two points if no obstacles exist, in this case our current node and the exit. This is calculated by pythagorim's theorem. $\hat{f}$ for each point is based upon the cost to reach the exit node, or $\hat{h}$ plus $\hat{g}$, the total cost accured so far from moving between states. $\hat{g}$ is calcuated as the previous $\hat{g}$ plus the distance between that point and the current position in the maze. The $\hat{g}$ for the entry point to the maze is 0.

5. See `rpsg.py`. When extending Rock/Paper/Scissors a few new constraints must be observed. Namely now that money is involved a player cannot bet more than they have or continue to play without any money. No bets made can exceed the minimum balance between both players. The largest bet will be used and the winning player will take the bet amount from the other player. In the case of a tie no money is lost. If a player makes an illegal bet the prompt is once again presented for their bet, this is implmented with a while loop. The wait for an input prevents an infinite loop of checking from occurring. This will continue until a valid bet is placed. The computer will pick a random amount of at least $1 and up to their maximum amount of money on hand.