CSE537: Artificial Intelligence, Fall 2018

Assignment 1

(Due: 21 September 2018)

Submission Details			
Name	SBU ID	Email Id	% Contribution
Shobhit Khandelwal	112074908	shobhit.khandelwal@stonybrook.edu	50 %
Vivek Bansal	112044493	vivek.bansal@stonybrook.edu	50 %

Answer 1) Depth First Search

Approach:

1) Take a stack and push InitialState into it.
2) Pop a node from stack and check if it is a goal state. If Yes, then return the sequences of moves to reach that goal state.
3) Else, find all the successors of curr_state. If it is not visited push it into fringelist stack and mark that node as visited.
4) Keep doing this until the stack becomes empty.

Statistics:
Test Command Run: **python pacman.py -l tinyMaze -p SearchAgent**

Details of Run			
Total Cost	Nodes Expanded	Score	RunTime
10	15	500	0.0s

Test Command Run: **python pacman.py -l mediumMaze -p SearchAgent**

Details of Run			
Total Cost	Nodes Expanded	Score	RunTime
130	146	380	0.0s

Test Command Run: **python pacman.py -l bigMaze -z .5 -p SearchAgent**

Details of Run			
Total Cost	Nodes Expanded	Score	RunTime
210	390	300	0.0s

Answer 2) Breadth First Search

Approach:

1) Take a Queue and enqueue InitialState into it and mark that state as visited.
2) Deque a node from Queue and check if it is a goal state. If Yes, then return the sequences of moves to reach that goal state.
3) Else, Append the dequeued node from the queue to the visited list. Find all the successors of dequeued node and enqueue them into Queue.
4) Keep doing this until the queue becomes empty.

Statistics:
Test Command Run: **python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs**

Details of Run			
Total Cost	Nodes Expanded	Score	RunTime
68	269	442	0.0s

Test Command Run: **python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5**

Details of Run			
Total Cost	Nodes Expanded	Score	RunTime
210	620	300	0.0s

Answer 3) Uniform Cost Search

Approach:

1) Take a Priority Queue and enqueue InitialState into it.
2) Deque a node from Queue and check if it is a goal state. If Yes, then return the sequences of moves to reach that goal state.
3) Find all the successors of dequeued node and enqueue them into Queue with their costs computed. If not visited push it into priority queue.
4) Keep doing this until the queue becomes empty.

Statistics:
Test Command Run: **python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs**

Details of Run			
Total Cost	Nodes Expanded	Score	RunTime
68	269	442	0.0s

Test Command Run: **python pacman.py -l mediumDottedMaze -p StayEastSearchAgent**

Details of Run			
Total Cost	Nodes Expanded	Score	RunTime
1	186	646	0.0s

Test Command Run: **python pacman.py -l mediumScaryMaze -p StayWestSearchAgent**

Details of Run			
Total Cost	Nodes Expanded	Score	RunTime
68719479864	108	418	0.0s

Answer 4) A Search*

Approach:

1) Take a Priority Queue and enqueue InitialState into it and mark that state as visited.
2) Deque a node from Queue and check if it is a goal state. If Yes, then return the sequences of moves to reach that goal state.
3) Find all the successors of dequeued node and enqueue them into Queue with their total costs computed including the heuristic value as well. Priority is decided by the total cost(f) = cost to reach this state(g)+ heuristic value at this state(h).
4) Keep doing this until the queue becomes empty.

Statistics:
Test Command Run: **python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic**

Details of Run			
Total Cost	Nodes Expanded	Score	RunTime
210	549	300	0.0s

Answer 5) Finding Corner Problem

Approach:
As part of this problem, we have implemented three functions:
1. getStartState() : Returning the complete state of the board including the pacman starting state and the corners un-visited.
2. isGoalState() : if all corners are visited that means we reached the goal state, so return True
3. getSuccessors() : Returns successor states, the actions they require, and a cost of 1.

Steps:
1) We will calculate next_state from the current state. We have 4 directions to move - north, south, east and west, so will calculate the direction vector of each directions to compute next_state by adding those direction vectors to the current state.
2) If next_state is a wall then that can't be the successor as pacman can't move to that position. So we will ignore that.
3) If next_state is a corner then we will skip that state from our corner_left. We will create tuples of other corners and add that corner_state to the successor.

Statistics:
Test Command Run: **python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs, prob=CornersProblem**

Details of Run			
Total Cost	Nodes Expanded	Score	RunTime
28	253	512	0.0s

Test Command Run: **python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem**

Details of Run			
Total Cost	Nodes Expanded	Score	RunTime
106	1967	434	0.2s

Answer 6) Corner Heuristics Problem

Approach:

1) Generate all possible permutations of the corners state.
2) With each generated permutation, find the shortest path possible from start point to goal state (when all corners are finished) using Manhattan distance.
3) return the minimum value among generated values in step (2)

Statistics:
Test Command Run: **python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5**

Details of Run			
Total Cost	Nodes Expanded	Score	RunTime
106	742	434	0.1s

Answer 7) Food Heuristic Problem

Approach:

1) First we tried Manhattan heuristic but that heuristic is not admissible in this case.
2) So we have computed maze distance to every food location.
3) The algorithm is taking time but it is an admissible heuristic and expands just a little above 4000 nodes.

Statistics:
Test Command Run: **python pacman.py -l testSearch -p AStarFoodSearchAgent**

Details of Run			
Total Cost	Nodes Expanded	Score	RunTime
7	10	513	0.0s

Test Command Run: **python pacman.py -l trickySearch -p AStarFoodSearchAgent**

Details of Run			
Total Cost	Nodes Expanded	Score	RunTime
60	4137	570	34.3s