Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Τεχνητή Νοημοσύνη 1

Pacman Project 1

Ονοματεπώνυμο: Γιώργος Μητρόπουλος Αριθμός Μητρώου: 11150202000128

- Q1: Depth First Search
  Since DFS follows a LIFO policy i used the Stack structure from util.py
  General idea: The starting state is inserted into the fringe.Then, we enter a loop where we remove a node from the fringe, we get its successor states, and we insert them into the fringe. If we pop the state that is already visited, we skip it and pop the next node from the fringe. If we pop the goal State, we end the loop and start building the path that will be returned. If successor state is already visited, its won't be inserted into the fringe. To keep some helpful information regarding each state, I used a Node class which contains a parent state, the action that brought us to the state e.t.c.
  To build the path, after the loop has finished, we get the action of the last state that was popped from the fringe and insert it to a path list. Then we get the parent of each state all the way back to the rout adding each action at the start of the path. If we reach None, they loop finishes, so we return the path .

- Q2: Breadth First Search
  The only difference from the DFS is that here we use a Queue as BFS follows FIFO policy.

- Q3: Uniform Cost Search
  Also similar idea, except we use PriorityQueue in order give prior to actios with smaller costs.

- Q4: A* Search
  PriorityQueue used also here, it gives prior to the cost + the result of the heuristic function (hn).

- Q5: Corners Problem: Representation
  Each state consists of the position of pacman and a list of corners that has not visited yet. For the starting state we place all the corners as pacman has not reached anything yet.
  For the goal state we check if the position of pacman is one of the corners and the list of corners is empty.
  getSuccessors: We copy a list of the remaining corners for the successor state. if the successor's position is a corner and we havent reached it we remove the corner from the list. So the successor state consists of the new cordinates along with the remaining corners

- Q6: Corners Problem: Heuristic
  If we have no remaining corners we have reached the goal. We use the Manhattan Distance form util.py to calculate all desired distances. We use a list that calculates and stores the distance between our current position and all corners. The maximum distance is being returned.

- Q7: Eating All The Dots: Heuristic
  Similar to previous one we use the MazeDistance function to find all the distances between our current position and all the food and return the maximum distance.

- Q8: Suboptimal Search
  For the findPathToClosestDot i use the BFS function problem = AnyFoodSearchProblem(gameState) that returns the path to the closest dot. For the goal state we check if we have food at your current state and return true or false.