Artificial Intelligence Project 01: The Searchin' Pac-Man

Q1: Implement the depth-first search (DFS) algorithm

For DFS, we need LIFO strategy, hence we are using Stack as a data structure to store fringe nodes. On this data structure, for one state we are storing the state coordinates, action needed to be taken to reach to this state and the cost of that action.

Each fringe node stores a tuple containing the path that needs to be travelled to reach to that node. Last element in tuple represents the information about the current node i.e. its coordinates, action and cost.

We ran the DFS for different type of Maze sizes w.r.t. PositionSearchProblem and the results are following:

	Tiny Maze	Medium Maze	Big Maze
Search Nodes Expanded	15	146	390
Time Taken to get the action list (s)	0.001	0.008	0.029

Q2: Implement the breadth-first search (BFS) algorithm

For BFS, we need FIFO strategy, hence we used Queue as a data structure to store fringe nodes. The information saved on fringe nodes is same as DFS. The only difference is that when we are getting node from fringe list we are following FIFO order in BFS while in DFS we follow LIFO order.

The result for different type of Maze sizes w.r.t PositionSearchProblem is following:

	Tiny Maze	Medium Maze	Big Maze
Search Nodes Expanded	15	269	620
Time Taken to get the action list (s)	0.001	0.015	0.042

Q3: Implement the uniform-cost search (UCS) algorithm

In case of UCS, we need to expand the node having minimum cost among nodes present in fringe list. So we used the Priority Queue as a data structure. On this data structure, for one state we are storing the state coordinates, action needed to be taken to reach to this state and the cost of that action.

Each fringe node stores a tuple containing the path that needs to be travelled to reach to that node and the cost of that path which is the equal to the sum of the cost of path till parent node and the cost of edge from parent to current node. Last element in tuple represents the information about the current node.

The result for different type of Maze sizes w.r.t PositionSearchProblem is following:

	Tiny Maze	Medium Maze	Big Maze
Search Nodes Expanded	15	269	620
Time Taken to get the action list (s)	0.001	0.015	0.045

We got very low(1) and very high(68719479864) path cost for the StayEastSearchAgent and StayWestSearchAgent respectively, due to their exponential cost functions.

Q4: Implement the A* algorithm

In case of A* Search, we need to expand the node having minimum cost among nodes present in fringe list. So we used the Priority Queue as a data structure. On this data structure, For one state we are storing the state coordinates, action needed to be taken to reach to this state and the cost of that action.

Each fringe node stores a tuple containing the path that needs to be travelled to reach to that node and the cost of that path.
Total cost of Path = cost of the path from start state to parent node + cost of edge connecting parent node to the current node + heuristic for current node
Last element in tuple represents the information about the current node.
The result for different type of Maze sizes w.r.t PositionSearchProblem is following:

	Tiny Maze	Medium Maze	Big Maze
Search Nodes Expanded	14	221	549
Time Taken to get the action list (s)	0.001	0.015	0.041

From results, we can conclude that A search is the best approach as number of nodes expended are very less compared to other searches.*

Q5: Implement the CornersProblem search problem
In Corners Problem, food is present at corners and Pacman need to visit all corners to get the food. So instead of saving only the coordinates of a state as we did in earlier problems, we also saved a boolean array having the info about the food at corners. False value in array means food is still present at that corner.
So State data structure is like this: (x-coordinate, y-coordinate, (bool array for corners))
We used this data structure so that at every step we are able to find the current state of the corners i.e. which corners are need to be visited now. And the Goal state is the state when all corners have been visited.
The result for Corners Problem w.r.t Mazes is:

	Tiny Corners Maze	Medium Corners Maze
Search Nodes Expanded	252	1966
Time Taken to get the action list (s)	0.005	0.082

Q6 Implement cornerHeuristic
"Results are corresponding to mediumCorners"
In cornersHeuristic, We tried four different heuristics.

1. First, we tried by giving the minimum of Manhattan distance or Euclidean distance of the remaining corners to be visited. So we tried this approach and got the below results:
   - Search nodes expanded: 1475
   - Autograder - q6: 2/3
   - Number of nodes expanded were too much

Thought process - Pacman needs to at least visit this much minimum distance to reach its goal. However this was very tight bounded, so we again relaxed our problem state

2. Now we tried by giving the sum of Manhattan distance/Euclidean distance of the remaining corners to be visited as the heuristic. Results:
   - This heuristic resulted in FAIL: inconsistent heuristic using autograder. However Search nodes expanded were only 910

Reason - It can't be admissible, as while visiting nearest corner, it might be now more nearer to the remaining corner, so we just overestimated the remaining distance left. As it is not admissible, hence it cannot be consistent.

3. Now we tried by giving the average of combined Manhattan distance/Euclidean distances of the remaining corners to be visited.Results:

- This performed better than the first Heuristic.
- Number of search nodes expanded: 1318
- Heuristic was also consistent.
- However the number of nodes expanded were still more than 1200(max nodes that can be expanded to get full credit).

4. Finally, we tried another heuristic in which we find the nearest(manhattan distance) corner co-ordinate and from this corner we find the next nearest corner and so on. We sum up all the Manhattan distance and returns it. We also tried using Euclidean distance instead of Manhattan distance, but Manhattan performed better. Results:

- This heuristic was also admissible and consistent and number of expanded nodes were also lesser than all the previous approaches.
- Search nodes expanded: 692
- This heuristic passed both admissibility and consistency test and also reduced the number of expanded nodes.
- So this one is the best approach among the approaches we tried and hence we used it.

Why it is admissible?
It won't overestimate ever, as A* will first visit the nearest corner, then from that corner it will visit the nearest corner from the remaining corner and so on. As manhattan distance is the lower bound on the distance that pacman needs to go, hence this heuristic will be lower bound on the remaining distance that needs to be covered, hence it is admissible. This is similar to what pacman will do, but instead of real distance we are assuming that real distance will be at least equal to manhattan distance.

Why it is consistent?
According to the definition of consistency, a heuristic is consistent, if for every node n, the estimated cost of reaching the goal from n is not greater than cost of going to n'(n' is successor of n) plus estimated cost of reaching the goal from n'.
In our case, manhattan distance is the least distance that needs to be covered for reaching the goal. So, if h(n) is the heuristic of reaching goal G from n and pacman visit some node from n (Assuming n' is in same path of manhattan distance for n to goal) before going to goal, it will atleast travel the manhattan distance to reach n' and if we assume that again it takes the shortest path to reach goal from n', then total distance will be at least equal to h(n). In any case, pacman cannot travel less than manhattan distance to reach its goal.
Can be done: After solving q7, I think, if we can use maze distance instead of the current approach, it may fare better and the number of expanded nodes can be reduced

Q7: Implement foodHeuristic

"Results are corresponding to Tricky Search"
In Food Heuristic, We need to find a heuristic that is admissible and consistent. We tried 4 different heuristics.

1. Firstly we tried by using the same approach that we applied in corners Problem i.e. Find the nearest food co-ordinate and then go to next nearest food and so on. Sum all these food distances and return it.
   But this heuristic is not admissible.
2. Then we tried by taking average of Manhattan distances of all food points from current position and the result is following:
   Time Taken to get the action list: 16.302 s
   Search nodes expanded: 11254
   This heuristic was admissible and consistent but number of nodes expanded were too much.

3. Then we thought that why we need to take the average of all distances. To find a heuristic we need to relax the state and then find the cost. So we can directly use the distance of the food which is farthest. To reach to Goal state, pacman need to reach to farthest food at least. So we tried this approach and return the Manhattan distance of the farthest food point and result was following:
   Time Taken to get the action list: 10.95 s
   Search nodes expanded: 9551

   This heuristic was also admissible and consistent and number of expanded nodes were also lesser than the previous approach.

4. But still this need some improvements. We found a function in searchAgents.py with name "mazeDistance" which calculates the shortest distance between two points. We thought that we should use Maze distance instead of Manhattan distance because this is the actual cost of the farthest food point. So we used Maze distance instead of Manhattan distance and returned the Maze distance of the farthest point and the result was amazing:
   Time Taken to get the action list: 19.829 s
   Search nodes expanded: 4137
   This heuristic passed both admissibility and consistency test and also reduced the number of expanded nodes to 4137.

So using "mazeDistance" of the farthest point is the best approach among the approaches we tried and hence we used it.

Why it is admissible?
It won't overestimate ever,as the Maze distance of the farthest food point is the lower bound on the distance that pacman needs to go, hence this heuristic will be lower bound on the remaining distance that needs to be covered, hence it is admissible.

Why it is consistent?
As in the case of corner heuristic, maze distance will give distance that pacman needs to reach any particular goal state. Also, there are no negative paths in the maze, hence, h(n) will be always less than or equal to the distance that pacman will travel if it first goes to n'(successor of n) and then it goes to the goal state.
h(n) will be equal to g(n') + h(n') when n' is on the path the shortest path that pacman follows while going to the goal, in other cases g(n') + h(n') will be larger than h(n).

h(n) - heuristic for reaching goal from n.
n' - successor of n
g(n') - actual cost of reaching n'