HW1 Project Report

1. ***Question 1*** Implement the depth-first search (DFS) algorithm:

   - **Explanation:** To implement Depth First algorithm, I am using a stack data structure, which will be used to store each visited node and its successor nodes before expanding them. Each node is list of triples (successor state, action and stepCost). The algorithm pops the top most element from stack, explore its children, if not already visited then push children nodes in stack and keep on doing this until goal is reached. Once we find the goal state, we backtrack from goal node to the source node and capture the actions(directions) and return the same.
   - **Nodes Expanded:**

Python Command	Nodes Expanded
python pacman.py -l tinyMaze -p SearchAgent --frameTime 0	Search nodes expanded: 15
python pacman.py -l mediumMaze -p SearchAgent --frameTime 0	Search nodes expanded: 146
python pacman.py -l bigMaze -z .5 -p SearchAgent --frameTime 0	Search nodes expanded: 390

2. **Question 2** Implement the breadth-first search (BFS) algorithm:

   - **Explanation:** To implement Breadth First Search, I am using Queue data structure and storing the nodes and all its successor to process in FIFO order. Each node is list of triples (successor state, action and stepCost). The algorithm pops the first element from queue, if not in closed list then explore its children, push children nodes in queue, mark them as visited and keep on doing this until goal is reached. Once we find the goal state, we backtrack from goal node to the source node and capture the actions(directions) and return the same.
   - **Nodes Expanded:**

Python Command	Nodes Expanded
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs --frameTime 0	Search nodes expanded: 269
python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5 --frameTime 0	Search nodes expanded: 620

3. **Question 3** Implement the uniform-cost search (UCS) algorithm

   - **Explanation:** To implement Uniform Cost Search algorithm, I used a priority queue and storing each node and its successor nodes according to priority(path cost). Each node is list of triples (successor state, action and stepCost). The algorithm pops the element with least priority from queue, explore its children, if not already visited calculate the path cost from source to that current node, if new optimal cost found then update the existing element with new path cost or add the node in the queue, simultaneously maintain one backtracking list and keep on doing this

until goal is reached. Once we find the goal state, we backtrack from goal node to the source node and capture the actions(directions) and return the same.

- **Nodes Expanded:**

Python Command	Nodes Expanded
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs --frameTime 0	Search nodes expanded: 275
python pacman.py -l mediumDottedMaze -p StayEastSearchAgent --frameTime 0	Search nodes expanded: 186
python pacman.py -l mediumScaryMaze -p StayWestSearchAgent --frameTime 0	Search nodes expanded: 108

4. **Question 4** Implement A* algorithm

- **Explanation:** To implement A* search algorithm, I am using priority queue which will hold the nodes and their successor along with the path cost (edge cost + heuristic). Each node is list of triples (successor state, action and stepCost). The algorithm pops the element from queue whose path cost if minimum, mark them visited, explore its children and calculate path cost for each node from source, if node cost is already calculated then either we go for an optimal node or update the cost in the queue, if not already visited then push children nodes in queue and keep on doing this until goal is reached. Once we find the goal state, we backtrack from goal node to the source node and capture the actions(directions) and return the same.
- **Nodes Expanded:**

Python Command	Nodes Expanded
python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic --frameTime 0	Search nodes expanded: 549

5. **Question 5** Implement the CornersProblem search problem

- **Explanation:** The program starts by returning the start node which is nothing but the position of Pac-man in the starting state. While finding the successors of the starting node if the next node is not colliding with the wall then verify with the corner value, mark it as visited and append as successor node along with direction and path cost. As we are checking both the condition whether hitting the wall or not and also if the next position is a valid corner state so the resulted successors will be valid state and they will help the pacc-man to solve the problem and reach goal.
- **Nodes Expanded:**

Python Command	Nodes Expanded
python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem --frameTime 0	Search nodes expanded: 435
python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem --frameTime 0	Search nodes expanded: 2448

6. **Question 6** Implement a heuristic for the CornersProblem in cornersHeuristic

- **Explanation:** While implementing heuristic (estimated cost) of a problem we need to check both the condition:
  - Admissible: which means calculated heuristic of current node <= actual cost from current node to goal node
  - Consistent: which means calculated heuristic of current node <= step cost from current node to its successor + heuristic from successor to goal

  Now, I am first identifying which corners are not yet visited. If starting position of Pac-man is not any corner value then finding the manhattan distance from starting position to current node/current corner. If this distance is less than previously calculated distance then assigning this as new heuristic value and by this it's becoming admissible. Also, among all the possible paths, I am considering the path with least heuristic which is not through successors with more heuristic value so my calculated heuristic is also consistent.

- **Nodes Expanded:**

Python Command	Nodes Expanded
python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5 --frameTime 0	Search nodes expanded: 2164

7. **Question 7** foodHeuristic with a consistent heuristic for the FoodSearchProblem

- **Explanation:** While implementing heuristic (estimated cost) of a problem we need to check both the condition:
  - Admissible: which means calculated heuristic of current node <= actual cost from current node to goal node
  - Consistent: which means calculated heuristic of current node <= step cost from current node to its successor + heuristic from successor to goal

  Now, to find the solution of collecting all the foods in the pac-man world, I will start with initializing the pac-man position and food grid. Now if the food items are not on any walls of the problem then we are finding the mazeDistance which returns the maze distance between any two points. As long as this is minimum than the previously calculated distance then we will consider this as the new heuristic value. Hence this is an admissible heuristic. And again, among all the possible paths, I am considering the path with least heuristic which is not through successors with more heuristic value so my calculated heuristic is also consistent.

- **Nodes Expanded:**

Python Command	Nodes Expanded
python pacman.py -l trickySearch -p AStarFoodSearchAgent --frameTime 0	Search nodes expanded: 4328

Autograder Report:

```
Gourabs-MacBook-Pro:search gourabbhattacharyya$ python autograder.py
Starting on 9-20 at 3:01:05

Question q1
===========

*** PASS: test_cases/q1/graph_backtrack.test
***    solution:            ['1:A->C', '0:C->G']
***    expanded_states:     ['A', 'D', 'C']
*** PASS: test_cases/q1/graph_bfs_vs_dfs.test
***    solution:            ['2:A->D', '0:D->G']
***    expanded_states:     ['A', 'D']
*** PASS: test_cases/q1/graph_infinite.test
***    solution:            ['0:A->B', '1:B->C', '1:C->G']
***    expanded_states:     ['A', 'B', 'C']
*** PASS: test_cases/q1/graph_manypaths.test
***    solution:            ['2:A->B2', '0:B2->C', '0:C->D', '2:D->E2', '0:E2->F', '0:F->G']
***    expanded_states:     ['A', 'B2', 'C', 'D', 'E2', 'F']
*** PASS: test_cases/q1/pacman_1.test
***    pacman layout:       mediumMaze
***    solution length: 130
***    nodes expanded:      146

### Question q1: 3/3 ###


Question q2
===========

*** PASS: test_cases/q2/graph_backtrack.test
***    solution:            ['1:A->C', '0:C->G']
***    expanded_states:     ['A', 'B', 'C', 'D']
*** PASS: test_cases/q2/graph_bfs_vs_dfs.test
***    solution:            ['1:A->G']
***    expanded_states:     ['A', 'B']
*** PASS: test_cases/q2/graph_infinite.test
***    solution:            ['0:A->B', '1:B->C', '1:C->G']
***    expanded_states:     ['A', 'B', 'C']
*** PASS: test_cases/q2/graph_manypaths.test
***    solution:            ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***    expanded_states:     ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases/q2/pacman_1.test
***    pacman layout:       mediumMaze
***    solution length: 68
***    nodes expanded:      269

### Question q2: 3/3 ###
```

```
Question q3
===========

*** PASS: test_cases/q3/graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C', 'D']
*** PASS: test_cases/q3/graph_bfs_vs_dfs.test
***     solution:               ['1:A->G']
***     expanded_states:        ['A', 'B']
*** PASS: test_cases/q3/graph_infinite.test
***     solution:               ['0:A->B', '1:B->C', '1:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases/q3/graph_manypaths.test
***     solution:               ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***     expanded_states:        ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases/q3/ucs_0_graph.test
***     solution:               ['Right', 'Down', 'Down']
***     expanded_states:        ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases/q3/ucs_1_problemC.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         275
*** PASS: test_cases/q3/ucs_2_problemE.test
***     pacman layout:          mediumMaze
***     solution length: 74
***     nodes expanded:         260
*** PASS: test_cases/q3/ucs_3_problemW.test
***     pacman layout:          mediumMaze
***     solution length: 152
***     nodes expanded:         173
*** PASS: test_cases/q3/ucs_4_testSearch.test
***     pacman layout:          testSearch
***     solution length: 7
***     nodes expanded:         14
*** PASS: test_cases/q3/ucs_5_goalAtDequeue.test
***     solution:               ['1:A->B', '0:B->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C']

### Question q3: 3/3 ###


Question q4
===========

*** PASS: test_cases/q4/astar_0.test
***     solution:               ['Right', 'Down', 'Down']
***     expanded_states:        ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases/q4/astar_1_graph_heuristic.test
***     solution:               ['0', '0', '2']
***     expanded_states:        ['S', 'A', 'D', 'C']
*** PASS: test_cases/q4/astar_2_manhattan.test
***     pacman layout:          mediumMaze
```

```
***     solution length: 68
***     nodes expanded:         224
*** PASS: test_cases/q4/astar_3_goalAtDequeue.test
***     solution:               ['1:A->B', '0:B->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases/q4/graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C', 'D']
*** PASS: test_cases/q4/graph_manypaths.test
***     solution:               ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***     expanded_states:        ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']

### Question q4: 3/3 ###


Question q5
===========

*** PASS: test_cases/q5/corner_tiny_corner.test
***     pacman layout:          tinyCorner
***     solution length:                28

### Question q5: 3/3 ###


Question q6
===========

*** PASS: heuristic value less than true cost at start state
*** PASS: heuristic value less than true cost at start state
*** PASS: heuristic value less than true cost at start state
path: ['North', 'East', 'East', 'East', 'East', 'North', 'North', 'West', 'West', 'West', 'West', 'North', 'North', 'North',
'North', 'North', 'North', 'West', 'West', 'West', 'West', 'North', 'North', 'East', 'East', 'East', 'East', 'South', 'South',
'South', 'South', 'South', 'South', 'West', 'West', 'South', 'South', 'South', 'South', 'West', 'West', 'South', 'East', 'East',
'North', 'North', 'North', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'South', 'South', 'East', 'East', 'East',
'East', 'East', 'North', 'North', 'East', 'East', 'North', 'North', 'East', 'East', 'North', 'North', 'East', 'East', 'East', 'East',
'South', 'South', 'South', 'South', 'East', 'East', 'North', 'North', 'East', 'East', 'South', 'South', 'South', 'South', 'South',
'North', 'North', 'North', 'North', 'North', 'North', 'North', 'West', 'West', 'North', 'North', 'East', 'East', 'North',
'North']
path length: 106
*** FAIL: Heuristic resulted in expansion of 2164 nodes

### Question q6: 0/3 ###


Question q7
===========

*** PASS: test_cases/q7/food_heuristic_1.test
*** PASS: test_cases/q7/food_heuristic_10.test
*** PASS: test_cases/q7/food_heuristic_11.test
*** PASS: test_cases/q7/food_heuristic_12.test
*** PASS: test_cases/q7/food_heuristic_13.test
```

```
*** PASS: test_cases/q7/food_heuristic_14.test
*** PASS: test_cases/q7/food_heuristic_15.test
*** PASS: test_cases/q7/food_heuristic_16.test
*** PASS: test_cases/q7/food_heuristic_17.test
*** PASS: test_cases/q7/food_heuristic_2.test
*** PASS: test_cases/q7/food_heuristic_3.test
*** PASS: test_cases/q7/food_heuristic_4.test
*** PASS: test_cases/q7/food_heuristic_5.test
*** PASS: test_cases/q7/food_heuristic_6.test
*** PASS: test_cases/q7/food_heuristic_7.test
*** PASS: test_cases/q7/food_heuristic_8.test
*** PASS: test_cases/q7/food_heuristic_9.test
*** PASS: test_cases/q7/food_heuristic_grade_tricky.test
***     expanded nodes: 4328
***     thresholds: [15000, 12000, 9000, 7000]

### Question q7: 5/4 ###


Question q8
===========

[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** Method not implemented: findPathToClosestDot at line 553 of searchAgents.py
*** FAIL: Terminated with a string exception.

### Question q8: 0/3 ###


Finished at 3:01:28

Provisional grades
==================
Question q1: 3/3
Question q2: 3/3
Question q3: 3/3
Question q4: 3/3
Question q5: 3/3
Question q6: 0/3
Question q7: 5/4
Question q8: 0/3
------------------
Total: 20/25

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

Gourabs-MacBook-Pro:search gourabbhattacharyya$
```