AI HW1

R12922029 陳姵安

1. **Show your autograder results and describe each algorithm**
    a. **autograder results**

```
(base) chenpeian@chenpeiandeMacBook-Pro-5 AI2024-hw1 % python autograder.py
Starting on 3-23 at 20:52:31

Question q1
===========
*** PASS: test_cases/q1/pacman_1.test
***     pacman layout:          mediumMaze
***     solution length: 130
***     nodes expanded:         146

### Question q1: 5/5 ###


Question q2
===========
*** PASS: test_cases/q2/pacman_1.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         269

### Question q2: 5/5 ###


Question q3
===========
*** 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: 10/10 ###


Question q4
===========
*** PASS: test_cases/q4/astar_0.test
***     solution:               ['Right', 'Down', 'Down']
***     expanded_states:        ['A', 'B', 'D', 'C', 'G']

### Question q4: 15/15 ###
```

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

### Question q5: 5/5 ###


Question q6
===========
*** PASS: heuristic value less than true cost at start state
path: ['North', 'East', 'East', 'East', 'East', 'North', 'North', 'West', 'West', 'West', 'West', 'No
rth', 'North', 'North', 'North', 'North', 'North', 'North', 'North', 'West', 'West', 'West', 'West',
'South', 'South', 'East', 'East', 'East', 'East', 'South', 'South', 'South', 'South', 'South', 'South
', 'West', 'West', 'South', 'South', 'South', 'West', 'West', 'North', 'East', 'East', 'North', 'Nort
h', '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', 'N
orth', 'North', 'North', 'North', 'West', 'West', 'North', 'North', 'East', 'East', 'North', 'North']
path length: 106
*** PASS: Heuristic resulted in expansion of 459 nodes

### Question q6: 9/9 ###


Finished at 20:52:33

Provisional grades
==================
Question q1: 5/5
Question q2: 5/5
Question q3: 10/10
Question q4: 15/15
Question q5: 5/5
Question q6: 9/9
------------------
Total: 49/49

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

    b. **describe each algorithm**

i. Q1. Depth First Search (1%)

DFS is a recursive, backtracking algorithm that exhaustively searches nodes. It progresses by popping the current node from a stack and pushing its neighbors, backtracking when necessary. Therefore, I created a stack as a frontier for DFS.
Starting from an initial state, it uses a stack (frontier) to keep track of the path and a list (exploredSet) to record visited nodes, avoiding revisits. The algorithm explores deeper into the graph by popping the most recent node from the stack and exploring its successors, which are added to the stack with their associated actions and costs. If a goal state is reached, it returns the list of actions leading to it. Otherwise, it continues exploring until all possible paths are exhausted.

ii. Q2. Breadth First Search (1%)

BFS is an iterative algorithm that explores nodes level by level. It progresses by dequeuing the current node from a queue and enqueuing its immediate neighbors, ensuring a breadth-first search pattern. Therefore, I created a queue as a frontier for BFS.
Starting from an initial state, it uses a queue (frontier) to keep track of the exploration frontier and a list (exploredSet) to remember visited states. For each state, it checks for the goal condition, returns the path (actions) to reach the goal if found, or continues to explore successors by enqueuing them with their respective actions and costs until all possibilities are exhausted.

iii. Q3. Uniform Cost Search (1%)

This code implements the Uniform Cost Search (UCS) algorithm, which explores paths in order of increasing cumulative cost, ensuring the least costly path to the goal is found. It uses a priority queue (frontier) to manage the exploration frontier, prioritizing nodes with lower total costs. An explored set (exploredSet) tracks the lowest cost to reach each state, allowing for efficient pruning of costlier paths. The algorithm iteratively expands the lowest-cost node from the frontier, updates the explored set, and enqueues successors with their cumulative costs. It terminates upon reaching the goal state, returning the sequence of actions leading to it.

iv. Q4. A* Search (null Heuristic) (1%)

The nullHeuristic algorithm represents a trivial heuristic for use in search algorithms, particularly those involving heuristic evaluations like A* search. A heuristic function, in the context of search problems, estimates the cost from a given state to the goal state. The effectiveness of a search algorithm can depend significantly on the accuracy of this estimation. It serves as a baseline or placeholder heuristic. When used in search algorithms like A*, it effectively turns the heuristic component off, reducing the algorithm to a uniform-cost search or Dijkstra's algorithm. This happens because the heuristic contributes no additional information about the distance to the goal beyond the actual costs of paths explored.

v. Q5. Breadth First Search (Finding all the Corners) (1%)

The cornerHeuristic algorithm designed here is used to solve the corner searching problem, where the goal is to find an efficient path that visits all corners of a grid maze. Based on the same logic of BFS, it aims at finding all corners in the maze. Therefore, we have to further maintain a list to mark the corner that has been found.

vi. Q6. A* Search (Corners Problem: Heuristic) (1%)

Based on the structure of A* searching algorithm at Q4, which implement null heuristics, the A* algorithm here uses non-trivial heuristics in order to find an optimal solution. To do so, I implemented three helper functions to help the non-trivial A* algorithm executing, such as findClosestPoint(), findFarthestPoint(), mazeDistance(). Then, I implemented the mazeDistance() function to calculate the real cost(currToClosest) and the estimated cost(closestToFarthest) to get the final heuristic value.

2. Describe the difference between Uniform Cost Search and A* Contours

The main difference between Uniform Cost Search (UCS) and the A* Search Algorithm lies in how they select the next node to explore and how they use heuristic functions to guide the search process. The main differences can be discussed from the following four aspects:

1. Concept:
    a. UCS expands equally in all “directions”.
    b. A* expands mainly toward the goal, but does hedge its bets to ensure optimality.
2. Cost Calculation:
    a. UCS does not use a heuristic function and chooses to use a cost function, g(n), as the path cost, which is the cost from the starting node to the current node.
    b. A* uses a heuristic function to estimate the cost from the current node to the goal node, combining the real cost function g(n) and h(n) to form a combined estimated cost function f(n), which is a mix of real cost and heuristic cost. This helps the A* search algorithms find an optimal solution.
2. Node Selection:
    a. UCS selects the node with the lowest cost from the start to the current node at each step, regardless of how far that node is from the goal node.
    b. A* selects the node with the smallest f(n) = g(n) + h(n) at each step, where g(n) is the actual cost from the starting node to the current node n, and h(n) is the estimated cost from the current node n to the goal node.
3. Objective:
    a. UCS: To find the minimum cost path from the starting node to the goal node.
    b. A*: To find an effective path from the starting node to the goal node while trying to minimize the total cost.
4. Advantages:
    a. UCS guarantees to find the lowest cost path (if one exists), suitable for problems where all actions have associated costs.
    b. A* typically finds the goal node faster than UCS when an appropriate heuristic function is available, as it can avoid exploring nodes that are far from the goal node.

3. Describe the idea of Admissible Heuristic (2%)

Due to the restriction of admissible heuristic that the cost is estimated by heuristics *h(n)* (which is the estimated cost from the current node *n* to the goal node) must be less than or equal to *h∗(n)* (which is the actual cost from the current node *n* to the goal node), admissible heuristics slow down optimality by trapping good plans on the fringe bad plans but never outweigh true costs.That's also why admissible heuristics is most of what's involved in using A* in practice. Compared with inadmissible heuristics, it is more likely for admissible heuristics to find an optimal solution.