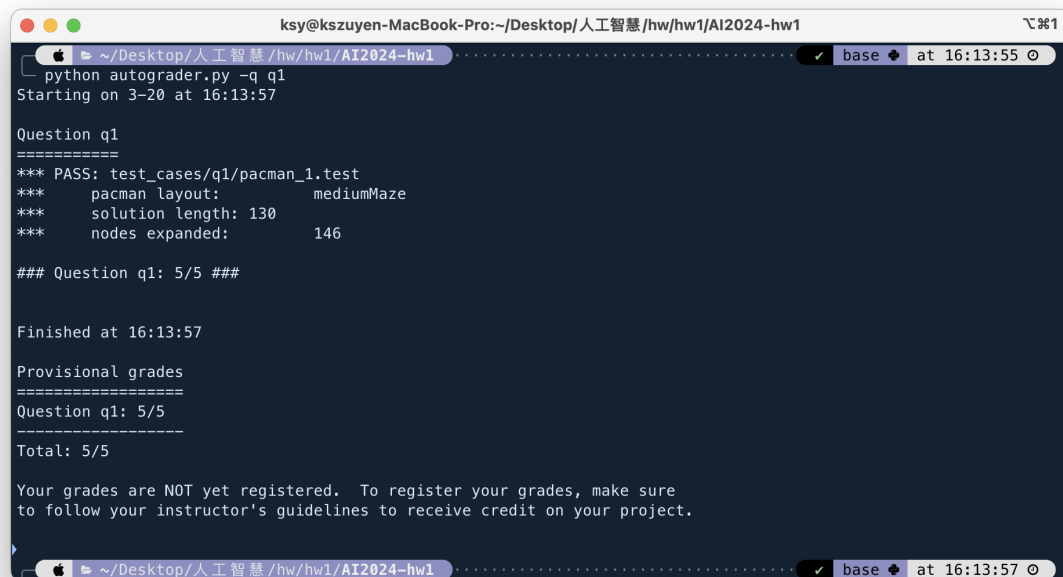


2024 AI - HW#1 Pacman - Search

生醫電資所 碩一 r12945040 郭思言

Show your autograder results and describe each algorithm:

- Q1. Depth First Search (1%)



```
ksy@kszuyen-MacBook-Pro:~/Desktop/人工智能/hw/hw1/AI2024-hw1
python autograder.py -q q1
Starting on 3-20 at 16:13:57

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

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

Finished at 16:13:57

Provisional grades
=====
Question q1: 5/5
-----
Total: 5/5

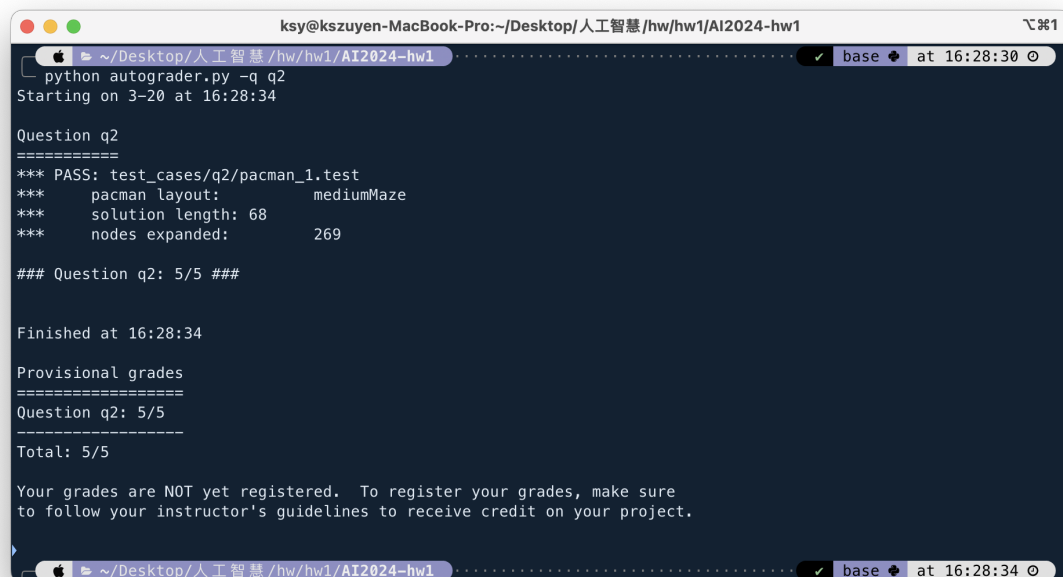
Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

DFS starts at the starting point and explores as far as possible along each branch before backtracking:

1. Start at the initial node and mark it as visited.
2. Explore one of the adjacent nodes that have not been visited yet. If all adjacent nodes are visited, backtrack to the previous node.
3. Repeat step 2 until there are no unvisited nodes left.

Implementation:

- I use a "stack" to keep track of the nodes to visit.
 - "visited": A "set" to keep track of the visited nodes.
 - "visiting": The current node being visited.
 - "path": A dictionary to store the path to each node.
 - In the main loop, while the stack is not empty, pop a node from the stack. If the node has not been visited, check if it is the goal state. If yes, return the path from the start state to the goal state. Mark the node as visited.
 - For each successor of the node, if the successor has not been visited, add it to the stack and update the "path" dictionary with the path to reach that successor.
- **Q2. Breadth First Search (1%)**



```
ksy@kszuyen-MacBook-Pro:~/Desktop/人工智能/hw/hw1/AI2024-hw1
python autograder.py -q q2
Starting on 3-20 at 16:28:34

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

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

Finished at 16:28:34

Provisional grades
=====
Question q2: 5/5
-----
Total: 5/5

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

Unlike DFS, which goes as deep as possible along each branch before backtracking, BFS explores all neighbor nodes at the present depth prior to moving on to nodes at the next depth level:

1. Start at the initial node and mark it as visited.

2. Add the initial node to a queue.
3. While the queue is not empty, do the following:
 - Remove a node from the front of the queue.
 - Explore all unvisited neighbor nodes of the removed node and mark them as visited.
 - Add these unvisited neighbor nodes to the back of the queue.
4. Repeat step 3 until the queue is empty.

Implementation:

- Use a "queue" to keep track of the nodes to visit.
 - Same as the Q1. DFS for the "visited", "visiting", and "path".
 - In the main loop, while the queue is not empty, dequeue a node from the front of the queue. If the node is the goal state, return the path from the start state to the goal state. Mark the node as visited. For each successor of the node, if the successor has not been visited, add it to the queue, mark it as visited, and update the "path" dictionary with the path to reach that successor.
- **Q3. Uniform Cost Search (1%)**

```
ksy@kszyuen-MacBook-Pro:~/Desktop/人工智能/hw/hw1/AI2024-hw1
Starting on 3-20 at 16:48:08

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 ###

Finished at 16:48:08

Provisional grades
=====
Question q3: 10/10
-----
Total: 10/10

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

UCS uses a priority queue to always select the node with the lowest cost:

1. Start at the initial node and mark its cost as 0.
 2. Add the initial node to a priority queue ordered by cost.
 3. While the priority queue is not empty, do the following:
 - Remove the node with the lowest cost from the priority queue.
 - If the removed node is the goal node, return the path to this node.
 - Otherwise, for each unvisited neighbor of the removed node, calculate the total cost to reach the neighbor (the cost of the removed node plus the cost to move to the neighbor) and add the neighbor to the priority queue with this total cost.
 - Mark the removed node as visited.
 4. Repeat step 3 until the priority queue is empty or the goal node is found.
- **Q4. A* Search (null Heuristic) (1%)**

```
ksy@kszuyen-MacBook-Pro:~/Desktop/人工智能/hw/hw1/AI2024-hw1
python autograder.py -q q4
Starting on 3-20 at 16:50:49

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

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

Finished at 16:50:49

Provisional grades
=====
Question q4: 15/15
Total: 15/15

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

A* search algorithm is an informed search algorithm that uses a heuristic to guide the search towards the goal node while considering the cost of reaching the current node:

Implementation:

1. Initialization:

- `pqueue`: A priority queue to keep track of nodes to visit, ordered by the A* score.
- `cost`: A dictionary to store the cost of reaching each node from the start node.
- `astarscore`: A dictionary to store the A* score of each node, which is the sum of the cost to reach the node and the heuristic estimate from the node to the goal.
- `path`: A dictionary to store the path from the start node to each node.
- `visited`: A set to keep track of visited nodes.

2. Initialize the start node:

- Set the start node as the current node `visiting`.

- Set the cost to reach the start node as 0.
- Calculate the A* score for the start node using the heuristic function.
- Add the start node to the `visited` set and initialize its path as an empty list.

3. Expand successors of the start node:

- For each successor of the start node, calculate the cost to reach the successor from the start node and update the `cost` and `astarscore` dictionaries.
- Add the successor to the `visited` set and push it to the priority queue `pqueue` with its A* score.
- Update the path to the successor in the `path` dictionary.

4. Main loop:

- While the priority queue is not empty, do the following:
 - Pop the node with the lowest A* score from the priority queue.
 - If the popped node is the goal node, return the path to this node.
 - Mark the popped node as visited.
 - For each unvisited successor of the popped node, calculate the cost to reach the successor and update the `cost` and `astarscore` dictionaries if the new cost is lower than the existing cost.
 - Add the successor to the priority queue with its updated A* score and update the path to the successor.

5. **Return:** If no path to the goal node is found, return `None`.

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

```
ksy@kszuyen-MacBook-Pro:~/Desktop/人工智能/hw/hw1/AI2024-hw1
python autograder.py -q q5
Note: due to dependencies, the following tests will be run: q2 q5
Starting on 3-20 at 16:51:04

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

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

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

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

Finished at 16:51:04

Provisional grades
=====
Question q2: 5/5
Question q5: 5/5
-----
Total: 10/10

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

The `CornersProblem` class defines a search problem where the goal is to find a path that visits all four corners of a layout.

- The `getStartState` method returns the start state, which consists of Pacman's starting position and an empty tuple to represent that no corners have been visited yet.
 - **Goal State:** It checks if Pacman's current position is one of the corners and if it has not been visited before. If so, it adds the corner to the list of visited corners and checks if all four corners have been visited.
 - **Successor States:** It considers all four cardinal directions and checks if moving in that direction would hit a wall. If not, it creates a new successor state with the updated position and updates the list of visited corners if a corner is reached for the first time.
- **Q6. A* Search (Corners Problem: Heuristic) (1%)**

```
ksy@kszuyen-MacBook-Pro:~/Desktop/人工智能/hw/hw1/AI2024-hw1
```

```
python autograder.py -q q6  
Note: due to dependencies, the following tests will be run: q4 q6  
Starting on 3-20 at 16:51:34
```

```
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 q6  
*****  
*** PASS: heuristic value less than true cost at start state  
path: ['North', 'East', 'East', 'East', 'East', 'North', 'North', 'West', 'West', 'West', 'West', 'North', 'North', 'N  
orth', 'North', 'North', 'North', 'North', 'North', 'West', 'West', 'West', 'South', 'South', 'East', 'East',  
'East', 'East', 'South', 'South', 'South', 'South', 'South', 'South', 'West', 'West', 'South', 'South', 'West'  
, 'West', 'East', 'East', 'North', 'North', 'North', 'East', 'East', 'East', 'East', 'East', 'East', 'East',  
'South', 'South', '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', 'N  
orth', 'North', 'West', 'West', 'North', 'North', 'East', 'East', 'North', 'North']  
path length: 106  
*** PASS: Heuristic resulted in expansion of 901 nodes  
  
## Question q6: 9/9 ##
```

```
Finished at 16:51:34
```

```
Provisional grades  
*****  
Question q4: 15/15  
Question q6: 9/9  
*****  
Total: 24/24
```

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

The `cornersHeuristic` function calculates a heuristic for the `CornersProblem` that is both admissible and consistent. It estimates the remaining shortest path from the current state to a goal state by considering the Manhattan distance to the nearest unvisited corner. The heuristic is consistent because the distance from the current state to the nearest unvisited corner is always an underestimate of the actual shortest path to any unvisited corner. This is because it does not account for the distance between unvisited corners or between the last visited corner and the current position.

Describe the difference between Uniform Cost Search and A* Contours (2%)

UCS and A* search are both informed search algorithms, but they differ in their approach to selecting nodes for expansion.

1. Uniform Cost Search (UCS):

- UCS expands nodes in order of their path cost from the start node. It always expands the node with the lowest path cost.
- UCS does not consider any additional information about the goal or the remaining path. It is solely focused on finding the path with the lowest cost.
- UCS is optimal and complete if the cost of each step is non-negative.

2. A* Search:

- A* search expands nodes based on a combination of the cost to reach a node from the start node ($g(n)$) and an estimate of the cost to reach the goal from that node ($h(n)$, h is a heuristic function). Based on their total estimated cost, $f(n) = g(n) + h(n)$, A* search prioritizes nodes that are estimated to be closer to the goal.
- A* is also optimal and complete, provided that the heuristic function $h(n)$ is admissible (never overestimates the actual cost to reach the goal) and consistent (satisfies the triangle inequality).

In summary, UCS expands nodes based solely on the path cost from the start node, while A* considers both the path cost and an estimate of the remaining cost to the goal. This allows A* to be more efficient in many cases, as it can focus its search on more promising paths.

Describe the idea of Admissibility Heuristic (2%)

An admissible heuristic is a function used in heuristic search algorithms (such as A*) that **never overestimates the cost to reach the goal from any given node**. In other words, an admissible heuristic is optimistic - it always provides a lower bound on the cost to reach the goal.

The idea behind an admissible heuristic is to guide the search algorithm towards the goal efficiently without exploring unnecessary paths. If the heuristic is admissible, the search algorithm is guaranteed to find the optimal solution, i.e., the shortest path to the goal.

Ex: Manhattan distance in pathfinding problem

- In a pathfinding problem, the Manhattan distance between the current state and the goal state is an admissible heuristic if there are no obstacles or the cost of moving between states is uniform. This is because the Manhattan distance provides a lower bound on the actual distance to the goal, as it only considers movements along the grid lines and ignores any obstacles or detours that might be present.