

Hyunjin Lee 2020320023

Professor Hyunwoo Kim

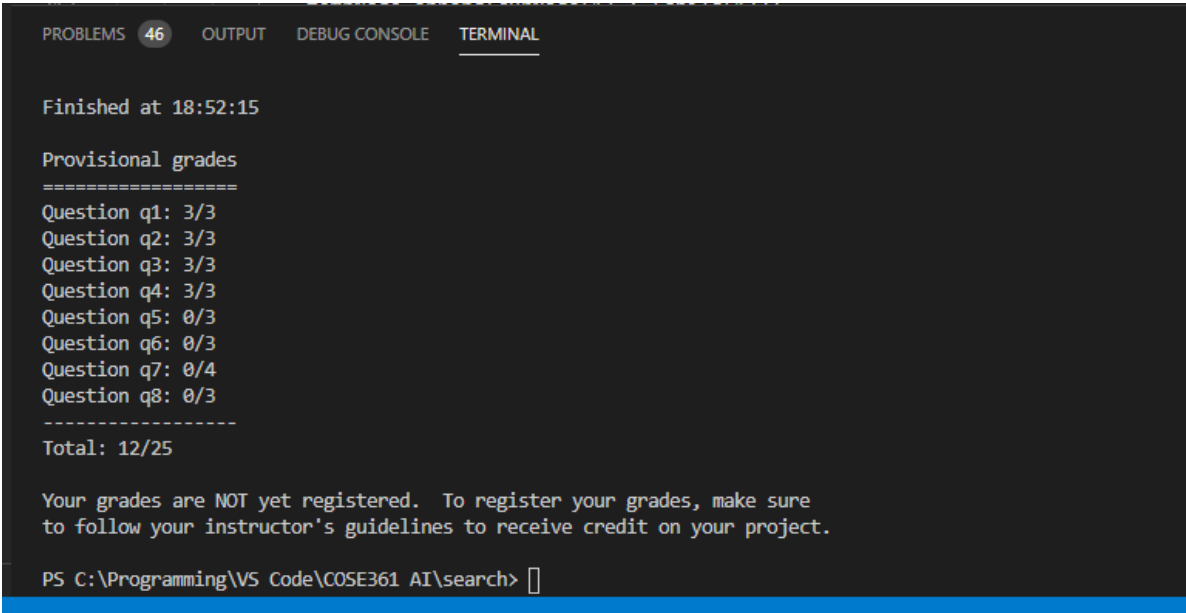
COSE361-03

07 April 2021

Assignment #1

Development Environment: Visual Studio Code

1) Result:



```
PROBLEMS 46 OUTPUT DEBUG CONSOLE TERMINAL

Finished at 18:52:15

Provisional grades
=====
Question q1: 3/3
Question q2: 3/3
Question q3: 3/3
Question q4: 3/3
Question q5: 0/3
Question q6: 0/3
Question q7: 0/4
Question q8: 0/3
-----
Total: 12/25

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

PS C:\Programming\VS Code\COSE361 AI\search> 
```

DFS Code Explanation:

Utilize DFS Maze Algorithm with Stack. Push in currNode's children if the child has not been visited yet. The stack acts as a solution (the paths) to the problem. Only pop out a node when there no longer exists an unvisited child given the currNode. Because backtracking is forbidden in this given problem, push in the child's siblings along with the child when an unvisited node has been found. Thus, instead of going back to the parent when there no longer exists a path, travel to one of the unvisited sibling node. Continue until the goal state has been reached and popped out.

BFS Code Explanation:

BFS using a Queue. Starting from the StartState, enqueue its unvisited child nodes along with the path from the head state to each child. Dequeue from the queue and copy the current saved path to actions array. Repeat the process until the goalState has been reached and dequeued; return actions.

UCS Code Explanation:

UCS using a Priority Queue. Starting from the StartState, enqueue its unvisited child nodes along with the path from the head state to each child. Dequeue the node with least cost and copy the current saved path to actions array. Repeat the process until the goalState has been reached and dequeued; return actions.

A* Code Explanation:

A* Search using a Priority Queue. Starting from the StartState, enqueue its unvisited child nodes along with the path from the head state to each child. Dequeue the node with least cost (backcost + heuristic) and copy the current saved path to actions array. Repeat the process until the goalState has been reached and dequeued; return actions.

2) Discussions:

1. For the medium maze, BFS algorithm is better than DFS algorithm because BFS finds a solution with less total cost than DFS algorithm. Both algorithms expand 269 search nodes, but BFS algorithm found a path with total cost of 68 while DFS algorithm found a path with total cost of 246. This is due to DFS initially taking a path in the wrong direction, or a farther direction, leading to the path of higher total cost. However, if the DFS algorithm push successors onto the stack in the order provided by getSuccessors function, only 146 search nodes are expanded (compared to 269 search nodes from BFS algorithm) with 130 total cost.
2. There does not seem to be any other heuristic function that is more efficient for this given problem. Both Euclidian and Manhattan heuristic functions result in equal number of nodes searched and total cost. This algorithm (with the Euclidian and Manhattan functions) works effectively for any situation involving numeric attributes. Ex. Finding the shortest path from one city to another. However, the algorithm fails to work if the problem involves other attributes such as categorical; in this case, we will have to use different algorithm along with different heuristic functions, for example a Hamming function.
3. Question) What is more important in a maze search: number of nodes searched or the total cost? Answer) Generally, I believe in a maze algorithm, finding the solution with least total cost is the most important task. However, I believe there exists an association between number of nodes searched and the total cost. Although it is preferable to find the path quickly (least number of nodes searched), to find the least total cost solution, it is in fact better to search through more paths (a greater number of nodes searched). Thus, number of nodes searched and the total cost may be equally important as they are related to each other.

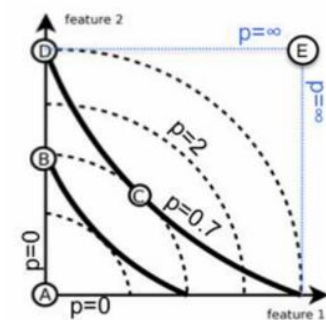
3) MyHeuristic and Result:

```
259 def myHeuristic(position, problem, info={}):
260     "You build your own heuristic function here"
261
262     """
263     Minkowski distance(p-norm) with p = 0.7
264
265     With p = 0.7, we assign higher cost/credit to multiple deviation along
266     the 2 features(x and y) compared to a large deviation along 1 feature(x or y).
267     This differs from the euclidean distance as for the euclidean distance,
268     having a large deviation along 1 feature(x or y) results in greater cost/credit
269     compared to multiple deviations along the 2 features(x and y).
270     """
271     xy1 = position
272     xy2 = problem.goal
273     return ( (xy1[0] - xy2[0]) ** 0.7 + (xy1[1] - xy2[1]) ** 0.7 ) ** (1 / 0.7)
274
```

PROBLEMS 13 OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Programming\VS Code\C0SE361 AI\search> python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=myHeuristic
[SearchAgent] using function astar and heuristic myHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 seconds
Search nodes expanded: 535
PS C:\Programming\VS Code\C0SE361 AI\search>
```

I have implemented Minkowski distance (p-norm) with $p = 0.7$ heuristic function. With $p = 0.7$, we assign higher cost/credit to multiple deviation along the 2 features(x and y) compared to a large deviation along 1 feature(x or y). This differs from the euclidean distance as for the euclidean distance, having a large deviation along 1 feature(x or y) results in greater cost/credit compared to multiple deviations along the 2 features(x and y).



Comparing this heuristic function with manhattan and euclidian function, all 3 functions result in 210 total cost for A* algorithm applied to the big maze. However, this heuristic function resulted in least number of search nodes expanded with 535, compared to 549 from manhattan and 557 for euclidean.

```
PS C:\Programming\VS Code\C0SE361 AI\search> python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 seconds
Search nodes expanded: 549
PS C:\Programming\VS Code\C0SE361 AI\search> python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=euclideanHeuristic
[SearchAgent] using function astar and heuristic euclideanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 seconds
Search nodes expanded: 557
PS C:\Programming\VS Code\C0SE361 AI\search>
```