Assignment 2 Results

Authors

Ryan Demo (rdemo1@jhu.edu)

Justin Chung (jchung51@jhu.edu)

600.435 | Artificial Intelligence | Spring 2017

Roles

Justin wrote the algorithms for questions 1-4. Ryan got questions 1-4 all passing autograder and implemented the CornersProblem search problem. Also collected submission materials and recorded results. Wrote the optional corners heuristic successfully and began writing the food heuristic but it currently isn't admissable.

Question 1: Depth-First Search

Tiny Maze

```
python pacman.py -l tinyMaze -p SearchAgent
```

```
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 500
Average Score: 500.0
Scores:        500.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Medium Maze

```
python pacman.py -l mediumMaze -p SearchAgent
```

```
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 146
Pacman emerges victorious! Score: 380
Average Score: 380.0
Scores:        380.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Big Maze

```
python pacman.py -l bigMaze -z .5 -p SearchAgent
```

```
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 390
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Open Maze

```
python pacman.py -l openMaze -z .5 -p SearchAgent
```

```
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 298 in 0.0 seconds
Search nodes expanded: 806
Pacman emerges victorious! Score: 212
Average Score: 212.0
Scores:        212.0
Win Rate:      1/1 (1.00)
Record:        Win
```

The Pacman board will show an overlay of the states explored, and the order in which they were explored (brighter red means earlier exploration). Is the exploration order what you would have expected?

For tinyMaze, the order is that which I would expect because Pacman faces his initial route and goes depth first along it attempting to discover the pellet. MediumMaze follows the same principle, with Pacman starting by heading out west along his initial path, then finding a way through the maze to the pellet after taking a long route. I would expect the upper loop to be explored before the pellet branch is explored, because the fork in the maze that branches back to near the starting position is shorter than the path to the pellet. Therefore, this would have been traversed first and discovered an already expanded node, thus discounting the branch as leading to a goal state without repetition. Observing this behavior in openMaze, it is consistent with the other mazes, and the fact that the upper left boxed area is smaller by area than the rectangle in which he starts out shows how all the red in the upper left box did not lead to a goal state, even though it was deep. It was also interesting to observe how some branches are unexplored completely if the DFS finds a goal state before expanding those branches.

Does Pacman actually go to all the explored squares on his way to the goal?

Pacman does not go to all of the explored squares. The algorithm explores depth-first and only begins Pacman on his journey to the pellet once a goal state is found through some branch. Branches explored first that lead to dead ends are ultimately not traversed by Pacman, since no goal state was found. It highlights the slowness of DFS for applications such as these particular

mazes, where several paths may lead to the pellet, but DFS certainly will not choose the least cost path every time. Pacman actually goes only to the squares along the path that was first discovered to lead to the pellet.

If you use a Stack as your data structure, the solution found by your DFS algorithm for mediumMaze should have a length of 130 (provided you push successors onto the fringe in the order provided by getSuccessors; you might get 246 if you push them in the reverse order). Is this a least cost solution? If not, think about what depth-first search is doing wrong.

We did use a Stack as the data structure and mediumMaze does find a path with total cost (length) of 130. By observation, this is not a least cost solution because Pacman goes out of his way to take a longer path when a shortcut exists close to his start state. Empirically, this is not a least cost solution, since BFS and A* search with a null heuristic both reach the pellet with a cost of 68. Depth first search traverses full paths to the leaves in order to find a goal state, and these full paths may take a much longer route than what BFS or A* would return. The reason for this is that BFS searches from the start position outwards according to breadth, where it looks at multiple paths from the start state and descends along them at the same time. This corrects the DFS issue of taking a long path when a shortcut to another node exists already, since the shortcut is descended at the same rate as the other branches. Thus, when the long way meets back up with the shortcut, the algorithm notices that the path it's about to expand has already been expanded, and ends that graph traversal.

Question 2: Breadth-First Search

Tiny Maze

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=bfs
```

```
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 16
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Medium Maze

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
```

```
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 270
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
```

```
Win Rate:      1/1 (1.00)
Record:        Win
```

Big Maze

```
python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
```

```
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 621
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Open Maze

```
python pacman.py -l openMaze -p SearchAgent -a fn=bfs -z .5
```

```
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 683
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores:        456.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Does BFS find a least cost solution?

BFS finds a least cost solution in all cases. In tinyMaze, it is easy to see by inspection that for a uniform cost per step, Pacman takes the least cost solution. In mediumMaze, Pacman chooses the most direct path that takes less cost than the zig-zagging on the right, and does not go in any dead ends. In bigMaze, Pacman again does not take any dead ends, and finds the correct way out of the maze to the food with least cost. Interesting to observe, Pacman also chooses the least cost solution when traversing openMaze, staying to the east while moving downward until reaching the horizonal line that contains the food. Then, Pacman goes directly to the food. This is the least cost because it is the minimum amount of distance needed to travel both south and west to reach the pellet with no extraneous or corrective steps.

Question 3: Varying the Cost Function

UCS Agent

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
```

```
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Medium Dotted Maze with StayEastSearchAgent

```
python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
```

```
Path found with total cost of 1 in 0.0 seconds
Search nodes expanded: 186
Pacman emerges victorious! Score: 646
Average Score: 646.0
Scores:        646.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Medium Scary Maze with StayWestSearchAgent

```
python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
```

```
Path found with total cost of 68719479864 in 0.0 seconds
Search nodes expanded: 108
Pacman emerges victorious! Score: 418
Average Score: 418.0
Scores:        418.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Question 4: A* Search

Tiny Maze

```
python pacman.py -l tinyMaze -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 8 in 0.0 seconds
Search nodes expanded: 14
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Medium Maze

```
python pacman.py -l mediumMaze -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 68 in 0.0 seconds
Search nodes expanded: 221
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Big Maze

```
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
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Open Maze

```
python pacman.py -l openMaze -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 54 in 0.1 seconds
Search nodes expanded: 535
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores:        456.0
Win Rate:      1/1 (1.00)
Record:        Win
```

What happens on openMaze for the various search strategies?

In depth-first search, Pacman zig zags back and forth west and east within wall bounds, stepping vertically closer to the food position only once the horizontal path has been traversed. Dead ends

are not visited, but DFS is very slow and expands 806 nodes. In DFS, parts of the grid are blacked out, as in not considered as paths because the algorithm already found a path to the food. The game board is black on paths that surely will not lead to the pellet, but the algorithm still considers the dead-end space in the top left because it searches depth first. Using BFS, Pacman chooses the least cost solution, staying to the east while moving downward until reaching the horizonal line that contains the food. Then, Pacman goes directly to the food. This is the least cost because it is the minimum amount of distance needed to travel both south and west to reach the pellet with no extraneous or corrective steps. BFS considers all paths according to the coloring of the game board, except for a dead end corner square at the same direction branch depth as the actual pellet. Thus, the pellet was discovered as the goal state before expanding the non-pellet square. A* with the manhattan heuristic has large black areas where direction nodes along those paths are not expanded because the heuristic would indicate that Pacman is moving away from the goal. This heuristic essentially keeps Pacman on the right track. Otherwise, the path behaves exactly like BFS.

Question 5: Finding All the Corners

Tiny Corners

```
python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
```

```
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 436
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores:        512.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Medium Corners

```
python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
```

```
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.3 seconds
Search nodes expanded: 2449
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Discussion

All we needed to do was define the start state as a tuple with the starting position and the visited corners, so that we'd always be able to access which corners remained to be visited. The goal state

returns true when all four corners have been visited.

Question 7: Eating All the Dots

Test Search

```
python pacman.py -l testSearch -p SearchAgent -a
fn=astar,prob=FoodSearchProblem,heuristic=nullHeuristic
```

```
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type FoodSearchProblem
Path found with total cost of 7 in 0.0 seconds
Search nodes expanded: 14
Pacman emerges victorious! Score: 513
Average Score: 513.0
Scores:        513.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Tricky Search

```
python pacman.py -l trickySearch -p SearchAgent -a
fn=astar,prob=FoodSearchProblem,heuristic=nullHeuristic
```

```
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type FoodSearchProblem
Path found with total cost of 60 in 43.3 seconds
Search nodes expanded: 16688
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:        570.0
Win Rate:      1/1 (1.00)
Record:        Win
```