***Question 1 (10 points)***Implement the depth-first search (DFS) algorithm in the depthFirstSearch function in [search.py](http://vision.cs.stonybrook.edu/lehhou/cse537/project01-2013/search.html). To make your algorithm complete, write the graph search version of DFS, which avoids expanding any already visited states

Stats for python pacman.py -l tinyMaze -p SearchAgent

$ 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

Win Rate: 1/1 (1.00)

Record: Win

Stats for python pacman.py -l mediumMaze -p SearchAgent

$ 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

Win Rate: 1/1 (1.00)

Record: Win

**NOTE**: When we push successors onto the fringe in reverse order, we get below stats.

Stats for python pacman.py -l mediumMaze -p SearchAgent

$ python pacman.py -l mediumMaze -p SearchAgent

[SearchAgent] using function depthFirstSearch

[SearchAgent] using problem type PositionSearchProblem

Path found with total cost of 152 in 0.0 seconds

Search nodes expanded: 245

Pacman emerges victorious! Score: 358

Average Score: 358.0

Scores: 358

Win Rate: 1/1 (1.00)

Record: Win

Stats for python pacman.py -l bigMaze -p SearchAgent

$ python pacman.py -l bigMaze -p SearchAgent

[SearchAgent] using function depthFirstSearch

[SearchAgent] using problem type PositionSearchProblem

Path found with total cost of 210 in 0.1 seconds

Search nodes expanded: 390

Pacman emerges victorious! Score: 300

Average Score: 300.0

Scores: 300

Win Rate: 1/1 (1.00)

Record: Win

**Questions**:-

The Pac-Man 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? – YES

Since we use Stack to implement DFS, the last pushed in successor is first popped out and we have defined the order of actions it should take (N, S, E, W). Looking at the solution, we expected the same exploration order.

Does Pac-Man actually go to all the explored squares on its way to the goal? – NO

It will only go the explored squares which are in the path it picked.

Is this a least cost solution? If not, think about what depth-first search is doing wrong.

NO. DFS expands the deepest unexpanded nodes and finds simply any solution without any guarantees of optimal solution.

**Question 2 (10 points)** Implement the breadth-first search (BFS) algorithm in the breadthFirstSearch function in [search.py](http://vision.cs.stonybrook.edu/lehhou/cse537/project01-2013/search.html). Again, write a graph search algorithm that avoids expanding any already visited states. Test your code the same way you did for depth-first search.

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

$ 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: 268

Pacman emerges victorious! Score: 442

Average Score: 442.0

Scores: 442

Win Rate: 1/1 (1.00)

Record: Win

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

$ 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 0.1 seconds

Search nodes expanded: 618

Pacman emerges victorious! Score: 300

Average Score: 300.0

Scores: 300

Win Rate: 1/1 (1.00)

Record: Win

**Questions**:

Does BFS find a least cost solution? YES

BFS will only find a path with the minimum number of edges. Since the total cost is the total number of edges in the path (if cost = 1 per step), BFS will definitely be a least cost solution as it will only find a path with minimum number of edges.

***Note****:* If you've written your search code generically, your code should work equally well for the eight-puzzle search problem (R&N 3ed Section 3.2, Figure 3.4) without any changes.

python eightpuzzle.py

$ python eightpuzzle.py

A random puzzle:

-------------

| 2 | 4 | 5 |

-------------

| 6 | 3 | 8 |

-------------

| 1 | | 7 |

-------------

BFS found a path of 15 moves: ['up', 'left', 'down', 'right', 'up', 'up', 'left', 'down', 'down', 'right', 'right', 'up', 'up', 'left', 'left']

***Question 3 (10 points)***Implement the uniform-cost search (UCS) algorithm in the uniformCostSearch function in [search.py](http://vision.cs.stonybrook.edu/lehhou/cse537/project01-2013/search.html). We encourage you to look through [util.py](http://vision.cs.stonybrook.edu/lehhou/cse537/project01-2013/util.html) for some data structures that may be useful in your implementation. You should now observe successful behavior in all three of the following layouts, where the agents below are all UCS agents that differ only in the cost function they use (the agents and cost functions are written for you):

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

$ 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: 274

Pacman emerges victorious! Score: 442

Average Score: 442.0

Scores: 442

Win Rate: 1/1 (1.00)

Record: Win

Stats for python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

$ python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

Warning: this does not look like a regular search maze

Path found with total cost of 1 in 0.0 seconds

Search nodes expanded: 190

Pacman emerges victorious! Score: 646

Average Score: 646.0

Scores: 646

Win Rate: 1/1 (1.00)

Record: Win

Stats for python pacman.py -l mediumScaryMaze -p 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

Win Rate: 1/1 (1.00)

Record: Win

**NOTE:**

We got very low path cost for StayEastSearchAgent : 1

And we got very high path costs for StayWestSearchAgent : 68719479864

***Question 4*** *(15 points)*Implement A\* graph search in the empty function aStarSearch in [search.py](http://vision.cs.stonybrook.edu/lehhou/cse537/project01-2013/search.html). A\* takes a heuristic function as an argument. Heuristics take two arguments: a state in the search problem (the main argument), and the problem itself (for reference information). The nullHeuristic heuristic function in [search.py](http://vision.cs.stonybrook.edu/lehhou/cse537/project01-2013/search.html) is a trivial example.

Stats for python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic

$ python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhat

tanHeuristic

[SearchAgent] using function astar and heuristic manhattanHeuristic

[SearchAgent] using problem type PositionSearchProblem

Path found with total cost of 210 in 0.2 seconds

Search nodes expanded: 538

Pacman emerges victorious! Score: 300

Average Score: 300.0

Scores: 300

Win Rate: 1/1 (1.00)

Record: Win

**Questions:**

You should see that A\* finds the optimal solution slightly faster than uniform cost search (about 549 vs. 620 search nodes expanded as reported in the Berkeley implementation, but ties in priority may make your numbers differ slightly).

Using A\*, total search nodes expanded: 538

Using UCS, total search nodes expanded: 619

$ python pacman.py -l bigMaze -p SearchAgent -a fn=ucs

[SearchAgent] using function ucs

[SearchAgent] using problem type PositionSearchProblem

Path found with total cost of 210 in 0.2 seconds

Search nodes expanded: 619

Pacman emerges victorious! Score: 300

Average Score: 300.0

Scores: 300

Win Rate: 1/1 (1.00)

Record: Win

What happens on openMaze for the various search strategies?

For DFS:

$ python pacman.py -l openMaze -p SearchAgent

[SearchAgent] using function depthFirstSearch

[SearchAgent] using problem type PositionSearchProblem

Path found with total cost of 298 in 0.1 seconds

Search nodes expanded: 808

Pacman emerges victorious! Score: 212

Average Score: 212.0

Scores: 212

Win Rate: 1/1 (1.00)

Record: Win

For BFS:

$ python pacman.py -l openMaze -p SearchAgent -a fn=bfs

[SearchAgent] using function bfs

[SearchAgent] using problem type PositionSearchProblem

Path found with total cost of 54 in 0.1 seconds

Search nodes expanded: 680

Pacman emerges victorious! Score: 456

Average Score: 456.0

Scores: 456

Win Rate: 1/1 (1.00)

Record: Win

For A\*:

$ python pacman.py -l openMaze -p SearchAgent -a fn=aStarSearch,heuristic=manha

ttanHeuristic

[SearchAgent] using function aStarSearch and heuristic manhattanHeuristic

[SearchAgent] using problem type PositionSearchProblem

Path found with total cost of 54 in 0.0 seconds

Search nodes expanded: 211

Pacman emerges victorious! Score: 456

Average Score: 456.0

Scores: 456

Win Rate: 1/1 (1.00)

Record: Win

ANALYSIS

BFS and AStarSearch algorithms find an optimal path of total cost 54. However DFS seems to have taken a longer route and its total cost came up to 298.

**Question** **5** (10 points) Implement the CornersProblem search problem in [searchAgents.py](http://vision.cs.stonybrook.edu/lehhou/cse537/project01-2013/searchAgents.html). You will need to choose a state representation that encodes all the information necessary to detect whether all four corners have been reached. Now, your search agent should solve:

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

$ 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: 410

Pacman emerges victorious! Score: 512

Average Score: 512.0

Scores: 512

Win Rate: 1/1 (1.00)

Record: Win

NOTE: the shortest path through tinyCorners took the neighborhood of 28 steps.

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

$ 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 0.4 seconds

Search nodes expanded: 2381

Pacman emerges victorious! Score: 434

Average Score: 434.0

Scores: 434

Win Rate: 1/1 (1.00)

Record: Win

**Question 6** (15 points) Implement a heuristic for the CornersProblem in cornersHeuristic. Grading: inadmissible heuristics will get no credit. 5 points for any admissible heuristic. 5 points for expanding fewer than 1600 nodes. 5 points for expanding fewer than 1200 nodes. Expand fewer than 800, and you're doing great!

$ python pacman.py -l mediumCorners -p SearchAgent -a fn=aStarSearch,prob=CornersProblem,heuristic=cornersHeuristic

[SearchAgent] using function aStarSearch and heuristic cornersHeuristic

[SearchAgent] using problem type CornersProblem

Path found with total cost of 106 in 0.2 seconds

Search nodes expanded: 943

Pacman emerges victorious! Score: 434

Average Score: 434.0

Scores: 434

Win Rate: 1/1 (1.00)

Record: Win

**NOTE:**

Expanded fewer than 1200 nodes: 943 precisely

Number of search nodes expanded have reduced significantly from 2381 to 943 when A\* search is used with heuristics when compared to BFS for the CornersProblem.

**Eating All The Dots**

Now we'll solve a hard search problem: eating all the Pac-Man food in as few steps as possible. For this, we'll need a new search problem definition which formalizes the food-clearing problem: FoodSearchProblem in [searchAgents.py](http://vision.cs.stonybrook.edu/lehhou/cse537/project01-2013/searchAgents.html) (implemented for you). A solution is defined to be a path that collects all of the food in the Pac-Man world. For the present project, solutions do not take into account any ghosts or power pellets; solutions only depend on the placement of walls, regular food and Pac-Man. (Of course ghosts can ruin the execution of a solution! We'll get to that in the next project.) If you have written your general search methods correctly, A\* with a null heuristic (equivalent to uniform-cost search) should quickly find an optimal solution to [testSearch](http://vision.cs.stonybrook.edu/lehhou/cse537/project01-2013/layouts/testSearch.lay) with no code change on your part (total cost of 7).

python pacman.py -l testSearch -p AStarFoodSearchAgent

Note: AStarFoodSearchAgent is a shortcut for -p SearchAgent -a fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic.

$ python pacman.py -l testSearch -p AStarFoodSearchAgent

Path found with total cost of 7 in 0.0 seconds

Search nodes expanded: 10

Pacman emerges victorious! Score: 513

Average Score: 513.0

Scores: 513

Win Rate: 1/1 (1.00)

Record: Win

**NOTE**: A\*search found an optimal solution to testSearch with a total cost of 7.

***Question 7 (20 points)***Fill in foodHeuristic in [searchAgents.py](http://vision.cs.stonybrook.edu/lehhou/cse537/project01-2013/searchAgents.html) with a consistent heuristic for the FoodSearchProblem. Try your agent on the trickySearch board:

python pacman.py -l trickySearch -p AStarFoodSearchAgent

Berkeley UCS agent finds the optimal solution in about 13 seconds, exploring over 16,000 nodes. If your heuristic is admissible, you will receive the following score, depending on how many nodes your heuristic expands.

Stats for python pacman.py -l trickySearch -p AStarFoodSearchAgent

$ python pacman.py -l trickySearch -p AStarFoodSearchAgent

Path found with total cost of 60 in 179.4 seconds

Search nodes expanded: 6408

Pacman emerges victorious! Score: 570

Average Score: 570.0

Scores: 570

Win Rate: 1/1 (1.00)

Record: Win

**NOTE:** Number of nodes heuristic expands: 6408