CSE150 HW1 Report
Boyu Feng A91084226
Dadong Jing A99018211
Rui Deng A91040303

Description of the problem and the algorithms used to solve all the problems.
Describe the data structure used in each algorithm.

3.1
Finding a fixed food dot using depth first search
Algorithm: DFS
Data Structure: Stack

3.2
Finding a fixed food dot using breadth first search
Algorithm: BFS
Data Structure: Queue

3.3
Find the best path which will consider the dangerous steps by adding different cost for steps
Algorithm: UCS
Data Structure: Priority Queue

3.4
Find the best path using astarsearch with empty heuristic function
Algorithm: A* Search with nullheuristic function
Data Structure: Priority Queue

3.5
Find all the corners
Algorithm: BFS
The goal state is found when all 4 corners in the corner set is visited
Add the action if the next position is not wall and visit the corner if it is not visted before.
Data Structure: Queue

3.6
Corner problem with non-trival huristic
Algorithm: A* Search
For the huristic function, input the goal state will return 0 and input other state will return the distance to the goal state. The distance can never be 0 because we have a toReturn value and we take aboslute value to make sure it is positive. Then we call the A* Search with the huristic function.
Data Structure: Priority Queue

3.7
Finding the best path that allows the pacman to eat all the dots
Algorithm: A* Search
We find the distance by calling mazeDistance on the food grid
Data Structure: Priority Queue

3.8
Let the pacman always greedly eat the cloest dot.

Algorithm: A* Search
We set the closet dot as the goal and find the best path.
Data Structure: Priority Queue

Analysis:
3.1
TinySize:
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 15

MediumSize:
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 146

BigSize:
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 390

3.2
TinySize:
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 15

MediumSize:
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269

BigSize:
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 seconds
Search nodes expanded: 620

3.3
TinySize:
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 15

MediumSize:
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem

Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269

BigSize:
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 seconds
Search nodes expanded: 620

3.4
TinySize:
[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

MediumSize:
[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

BigSize:
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 549

3.5
TinySize:
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 269

MediumSize:
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.1 seconds
Search nodes expanded: 1988

BigSize:
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 162 in 0.3 seconds
Search nodes expanded: 7974

3.6
TinySize:
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 215

MediumSize:
Path found with total cost of 106 in 0.0 seconds
Search nodes expanded: 1148

BigSize:
Path found with total cost of 162 in 0.1 seconds
Search nodes expanded: 4395

3.7
TestSearch:
Path found with total cost of 7 in 0.0 seconds
Search nodes expanded: 10

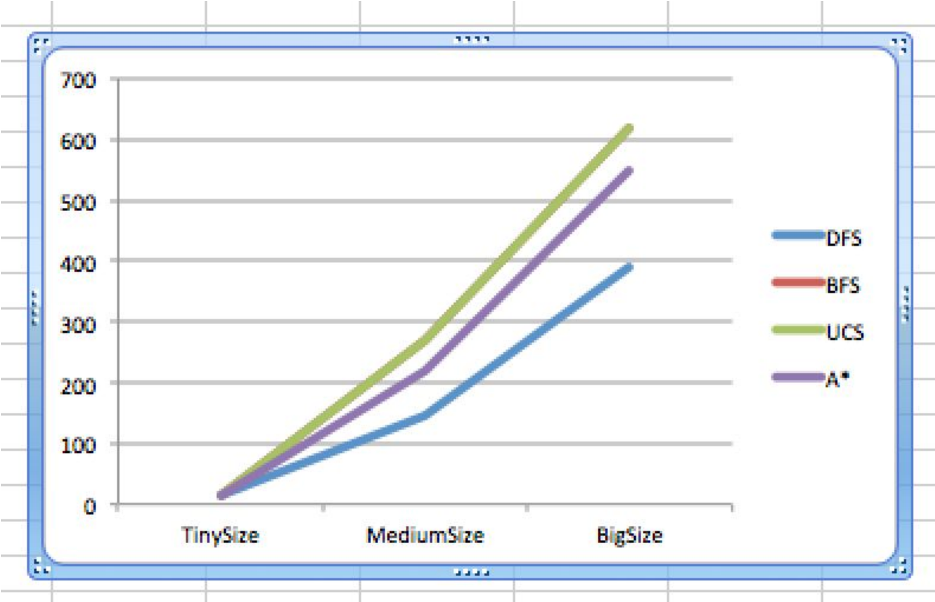
TrickySearch:
Path found with total cost of 60 in 128.7 seconds
Search nodes expanded: 4137

3.8
TinySize:
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with cost 31.

MediumSize:
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with cost 171.

BigSize:
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with cost 350.

Results:
For every problem, the cost and the node expanded increase as the map size increase.
3.1-3.4:
We can clearly know that for the same map with medium or large size, DFS performs better than A* Search and A*Search performs better than BFS for node expanded part. The UCS has the same result with BFS because they are actually the same when the cost is all 1. For a tiny size map, all these algorithm have same results.



3.5-3.6
A* Search perform much better than BFS especailly in large size map.
3.7
Eating all the dots will take much time when running on trickysearch. Although the node expanded is 4000 which is significantly smaller than the requirment, but the running time is 120s.

Contribution:
Boyu Feng: I did some programming and helped debugging. Also did the report part.
Rui Deng: I did the coding part, mostly on 3.5 through 3.8. Try to search online and firgure out each how each algorithm works.
Dadong Jing: I did the coding part, mostly on 3.1 through 3.4. And helped some part of debugging.