ARTIFICIAL INTELLIGENCE : ASSIGNMENT 1 REPORT

Part 1 Analysis

Autograder marks

Question	Marks	Reason Marks Lost
Question 1	3/3	N/A
Question 2	3/3	N/A
Question 3	3/3	N/A
Question 4	3/3	N/A
Question 5	3/3	N/A
Question 6	3/3	N/A
Question 7	5/4	N/A
Question 8	3/3	N/A
Total Marks Part 1	26/25	N/A

PROBLEM: (Finding All Corners) Without using Heuristics
Running Command :
```
python pacman.py -l mediumCorners -p SearchAgent -a fn=(dfs,bfs,astar,ucs) ,prob=CornersProblem
```

Algorithm	Criteria 1 (Time taken for execution in seconds)	Criteria 2 (Nodes expanded)	Criteria 3 (Cost)
DFS	0.0237321853638	378	221
BFS	0.72314786911	2448	106
UCS	0.740188837051	2448	106
A Star	0.767651081085	2448	106

Table 1: (Finding All Corners) Without using Heuristics

PROBLEM: Finding All Corners Problem With Heuristic
Running Command :
```
python pacman.py -l mediumCorners -p SearchAgent -a fn=(dfs,bfs,ucs,astar),prob=CornersProblem,
heuristic=cornersHeuristic.
```

Algorithm	Criteria 1 (Time taken for execution in seconds)	Criteria 2 (Nodes expanded)	Criteria 3 (Cost)
DFS	0.0239300727844	378	221
BFS	0.72173500061	2448	106
UCS	0.776878118515	2448	106
A Star	0.134000062943	901	106

Table 2: (Finding All Corners) using Heuristics

PROBLEM: Eating All Dots Problem
Running Command :
```
python pacman.py -l trickySearch -p SearchAgent -a
fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic --frameTime 0
```

Algorithm	Criteria 1 (Time taken for execution in seconds)	Criteria 2 (Nodes expanded)	Criteria 3 (Cost)
DFS	0.053820848465	362	216
BFS	69.5168528557	16688	60
UCS	70.444849968	16688	60
A Star	25.0728518963	376	60

Table 3: Eating All Dots Problem

DFS:

- DFS visit nodes of a graph **depth wise.**
- Stack Data Structure (LIFO Queues) are used for the implementation of DFS.
- DFS is faster and require less memory in comparison to BFS algorithm, as we can see from the above table 1 that DFS is taking 0.023 seconds for execution and only 378 nodes are expanded in comparison to BFS taking 2448 nodes expanded and 0.723 seconds of execution time.
- DFS is taking more cost for travelling in comparison to all other algorithms when executed with or without heuristics. As we can see from the table 1, table 2 that total cost for travelling for DFS is 221 which is more than twice what was taken from BFS, UCS and A star algorithm.
- With or without heuristics the time taken for execution, Nodes expanded and cost nearly remains same.
- For very complex problem like finding all food DFS is taking very less time to execute in comparison to all other algorithms(table 3)

BFS:

- BFS visit nodes **level by level** in a Graph.
- Queue Data Structure (FIFO Queues) are used for implementation of BFS.
- BFS is slower and require more memory than DFS as we can see from the above table . The time taken for execution by BFS is 0.72 seconds in comparison to 0.023 seconds for DFS. The number of nodes expanded by DFS is 2448 in comparison with 378 nodes with DFS which shows higher memory consumption by BFS.
- For very complex problem like finding all food BFS is taking very large time to execute(69.5168528557 seconds) as well as very high memory consumption(16688 nodes).
- With or without heuristics the time taken for execution, Nodes expanded and cost nearly remains same.

UCS

- UCS visit **least count unexpanded node** in a Graph.
- Priority Queue Data Structure are used for implementation of UCS.
- UCS perform similar to BFS when we are running the corner problem with or without heuristics as we can see from table 1 and table 2.
- Performance of UCS remains similar to BFS when the we are testing it in a very complex scenario like finding all food (table3).
- UCS is taking much more time for execution(70.44 seconds) and memory(16688 nodes expanded) in comparison to A star algorithm in very complex problem like finding all food.

A Star

- A Star considers the actual cost of the path along with estimated cost for reaching goal for travelling a graph.
- Priority Queue Data Structure are used for implementation of UCS.
- On comparing table 1 results we can see that when we are using A star algorithm without heuristics the results(execution time , node expanded and cost) remains very much similar to UCS and BFS.
- But when we are using heuristics in the same problem the execution time reduces sharply from 0.76 seconds to 0.13 seconds as well as the node expanded also reduces from 2448 to 901.
- From table number 3 we can see that, in case of very complex problem like finding food, A star performs much better than other 3 algorithm , taking very less execution time(25 seconds) and very less memory consumption(276 nodes expanded) to provide least cost route(60 cost).

Part 2 Analysis

Autograder marks

Question	Marks	Reason Marks Lost
Question 1	4/4	N/A
Question 2	5/5	N/A
Question 3	5/5	N/A
Total Marks part 2	14/14	N/A

Question 4 has not been attempted and Question 5 has been implemented but commented.

Running Command:

```
python pacman.py -p (AlphaBetaAgent,MinimaxAgent) -l minimaxClassic -a depth=2
```

Algorithm	Criteria 1 (Time taken for execution in seconds)	Criteria 2 (Nodes expanded)	Score
Mini Max	6.37990379333	905	510.0
Alpha Beta Pruning	3.08824205399	175	516.0

Table 4: comparison of Alpha Beta and Minimax Agent at similar score at depth =2

Running Command

```
python pacman.py -p (AlphaBetaAgent,MinimaxAgent) -l minimaxClassic -a depth=3
```

Algorithm	Criteria 1 (Time taken for execution in seconds)	Criteria 2 (Nodes expanded)	Score
Mini Max	5.13506317139	3857	513.0
Alpha Beta Pruning	4.37893295288	2445	514.0

Table 5: comparison of Alpha Beta and Minimax Agent at similar score at depth =3

MiniMax Agent:

On implementing adversarial search agent provided Minimax Agent decision rule, which expands the game tree to an arbitrary depth as it has multiple min layers for each for every max layer. When we are running it on minimaxClassic the agent is able to win many time easily but in case of smallClassic or medium Classic the pacman is not good at winning but perform good in case of survival . Many a time it just waits for ghost to come near to it then only it move. It only becomes active when ghost approaches it.

Alpha Beta Agent

Alpha-beta pruning is more efficiently exploring the minimax tree and hence performs much better and faster than Mini Max agent, after running it for multiple time we can see that, Alpha Beta Agent is more efficient in winning the matches not only in minimaxClassic but also in smallClassic and mediumClassic map.

From Table 4 we can see that in winning scenario when they are having very similar score, Alpha Beta pruning is taking much less time(3.08 seconds) as well as very less memory(175 nodes expanded) in comparison to Mini Max Agent taking 6.37 seconds for execution and expanding 905 nodes . On increasing the depth to 3, for a similar result(table 5) we can see that Alpha Beta pruning still performs much better than Mini Max Agent.