Q1: PEAS

PEAS stands for (P)erformance Measure, (E)nvironment, (A)ctuators, (S)ensors. PEAS analysis for a domestic service robot is:

Observable: Partially
Determinism: Stochastic
Episodic: Sequential
Static: Dynamic
Discrete: Continuous
Agents: Multi

I would choose the **'Learning Agent Architecture'** for this agent because it should measure the performance and outcomes of its own actions and improve/adjust its behaviours according to feedbacks.

Q2: Heuristic

In this part, I will use the notation from our course slides:

$h(n)$ - heuristic cost of node n.
$c(n, a, n')$ - real cost of action to advance from node n to node n'
$r(n)$ - real cost from node n to a goal node.

For consistency:

$$h(n) \leq c(n, a, n') + h(n')$$

condition should be satisfied. In the goal node, $h(goal)$ and $r(goal)$ is 0. For any other node n, there should be at least one node n' that satisfies:

$$r(n') = r(n) - 1$$

In admissibility $h(n') \leq r(n')$ condition should be true, so we can change the first condition to:

$$h(n) \leq c(n, a, n') + r(n')$$

as we know $c(n, a, n') + r(n') = r(n)$, it takes us to $h(n) \leq r(n)$.

For an example of a heuristic that is admissible but inconsistent, let's examine the graph given below:

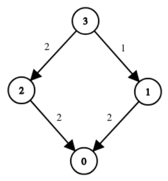


Figure 1: Graph

In this graph, heuristic costs are given in nodes and action costs are given on edges. The node that has 0 heuristic cost is the goal node and the node has 3 heuristic cost is the initial node. $h(n) \leq r(n)$ is satisfied for all nodes. But heuristic cost of 3 is bigger than heuristic cost of 1 + actual cost of 1(edge between them)

Q3: Peg Problem

In this part, we are wanted to implement and analyze BFS, DFS and A* algorithms on given peg problem.

Problem Formulization

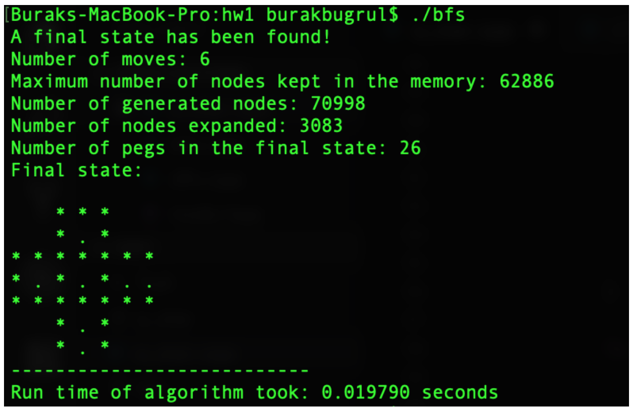
State: A 7x7 matrice of 1s and 0s. 1s represent holes with pegs in them and 0's represent empty holes.
Action: Making a peg jump orthogonally over an adjacent peg into an empty hole and then the jumped peg is removed. This action creates a new state.
Initial State: State that given in the picture in the assignment pdf.
Goal State: Any state that can make no valid action.
Path Cost: Evey peg move counts with the same cost in this problem. So, path cost is the number of actions.

Algorithms

You can compile the codes with the `'sh compile_all.sh'` command.

BFS Algorithm

You can run the algorithm with the `'./bfs'` command.
BFS expands a lot of unnecessary node due to its Breadth first iteration but in this problem it is guaranteed for BFS to find an optimal solution. Because all actions costs are same. Here is the analysis of the BFS run:

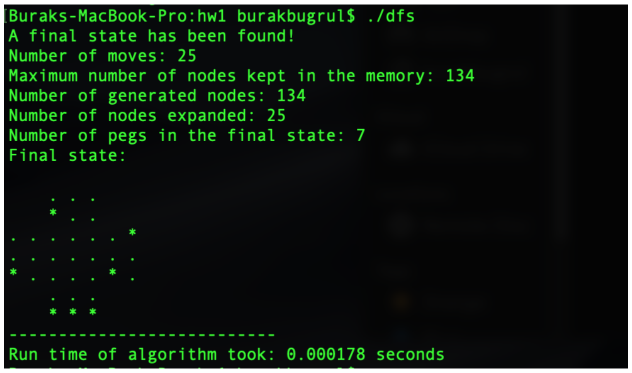


```
[Buraks-MacBook-Pro:hw1 burakbugrul$ ./bfs
A final state has been found!
Number of moves: 6
Maximum number of nodes kept in the memory: 62886
Number of generated nodes: 70998
Number of nodes expanded: 3083
Number of pegs in the final state: 26
Final state:

    * * *
    * . *
* * * * * * *
* . * . * . .
* * * * * * *
    * . *
    * . *
--------------------------
Run time of algorithm took: 0.019790 seconds
```

Figure 2: BFS

DFS Algorithm

You can run the algorithm with the `'./dfs'` command.
DFS expands always the first node it explores due to its depth first iteration. In this problem it is not guaranteed for DFS to find an optimal solution. Because it can explore a longer path to a close child. Here is the analysis of the DFS run:



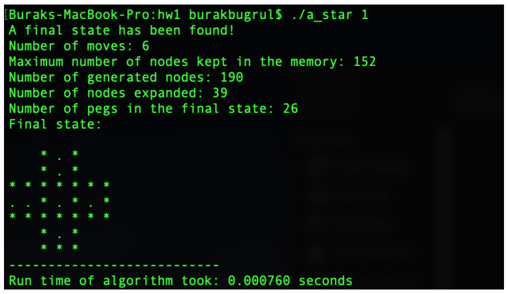
```
Buraks-MacBook-Pro:hw1 burakbugrul$ ./dfs
A final state has been found!
Number of moves: 25
Maximum number of nodes kept in the memory: 134
Number of generated nodes: 134
Number of nodes expanded: 25
Number of pegs in the final state: 7
Final state:

    . . .
    * . .
. . . . . . *
. . . . . . .
* . . . . * .
    . . .
    * * *
---------------------------
Run time of algorithm took: 0.000178 seconds
```

Figure 3: A boat.

A* Algorithm

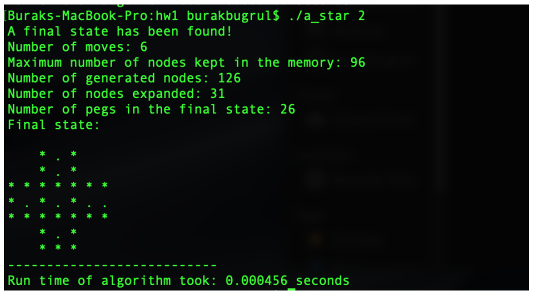
You can run the algorithm with the `'./a_star 1'` or `'./a_star 2'` command.
Independent from the problem, it is guaranteed for A* to find an optimal solution with an admissible and consistent heuristic function. In this problem, I analyzed two different heuristic functions:

Hole(1): It counts number of holes that can be occupied with next action.
Pegs(2): It counts number of pegs] that can be moved with next action.



```
Buraks-MacBook-Pro:hw1 burakbugrul$ ./a_star 1
A final state has been found!
Number of moves: 6
Maximum number of nodes kept in the memory: 152
Number of generated nodes: 190
Number of nodes expanded: 39
Number of pegs in the final state: 26
Final state:

    * . *
    * . *
* * * * * * *
. . * . * . *
* * * * * * *
    * . *
    * * *
--------------------------
Run time of algorithm took: 0.000760 seconds
```

(a) A* with Hole Heuristic.



```
Buraks-MacBook-Pro:hw1 burakbugrul$ ./a_star 2
A final state has been found!
Number of moves: 6
Maximum number of nodes kept in the memory: 96
Number of generated nodes: 126
Number of nodes expanded: 31
Number of pegs in the final state: 26
Final state:

    * . *
    * . *
* * * * * * *
* . * . * . .
* * * * * * *
    * . *
    * * *
--------------------------
Run time of algorithm took: 0.000456 seconds
```

(b) A* with Peg Heuristic

Figure 4: A*

One Peg State

If objective was trying to reach a state that has one peg there will be no optimal solution because every action removes exactly one peg. Every valid solution will have 31 actions because there are 32 pegs in the beginning. In this problem BFS will expand all search space whic is not optimal in terms of memory. DFS will use memory more efficiently but it can try very wrong(not necessary) actions in the deeper states. A* with a good heuristic can find a valid final state in a short time.