**01.11.2018**

# **ISTANBUL TECHNICAL UNIVERSITY**

# **BLG 435 E**

# **ARTIFICIAL INTELLIGENCE**

# **HW I**

STUDENT NAME: BURAK METE

STUDENT NUMBER: 150140131

**Q3 )**

**a) States :** Each individual different positioning of the pegs over the board.

**State Elements:**

* pegboard- a matrix for preserving the game board and positions of the pegs.
* leftOverPegs- How many pegs are there over the board

**# of States:** 2^33

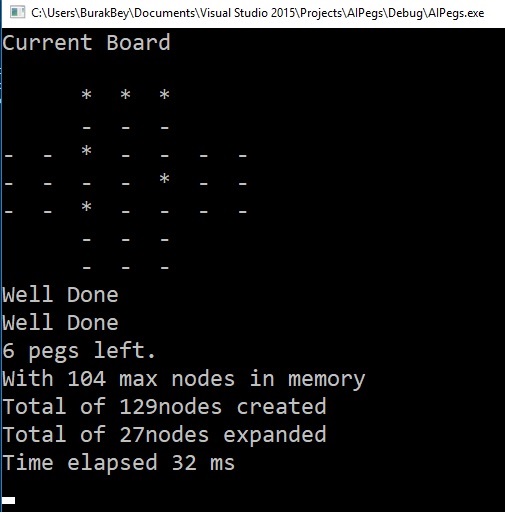
**Actions:** Each Peg move, there is a valid action where there is a piece adjacent to the initial piece, and there exist a empty cell in their extension

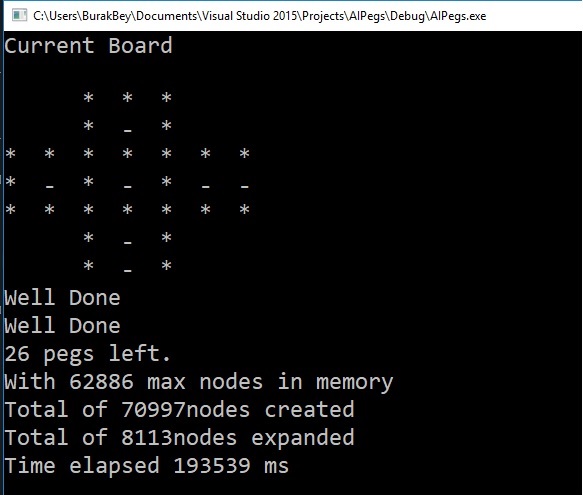
* moveUp
* moveBottom
* moveLeft
* moveRight

**Goal State:** The state where no valid action exists.

|  |  |  |
| --- | --- | --- |
|  | **BFS** | **DFS** |
| Nodes Generated | 70997 | **129** |
| Nodes Expanded | 8113 | **27** |
| Max nodes kept in memory | 62286 | **104** |
| Running Time | 193 seconds | **32 milliseconds** |
| Pegs left in final state | 26 | **6** |

**b)**





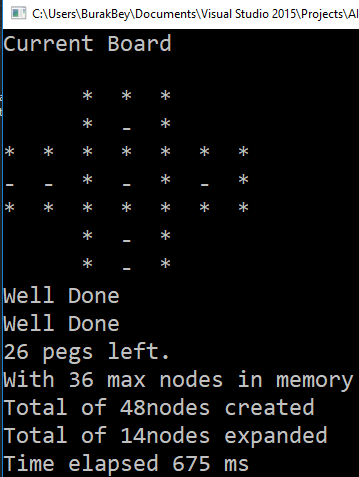
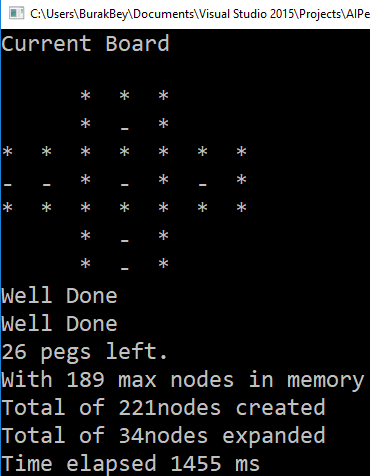
***Exemplary BFS run***  ***Exemplary DFS run***

It can be seen that DFS over performed BFS, for all the preliminary criteria, however, it can be said that this is caused by the nature of the problem itself. Since the goal of the solution is to end up in a state where no valid move exist, **DFS guarantees to find a solution in at most number of moves equal to the number of pegs.** It does not guarantees to find the most optimal, but since it goes deeper at each iteration, it has a better memory usage and better performance in terms of reaching a goal state. I implemented the DFS search algorithm by using a stack, simply pushing each possible child states to state, and at the end of each iteration popping and continuing from the stack.

Whereas, since BFS runs level by level, and iterate over all the branched states, it guarantees to find the solution in the most-upper level. **Therefore it also guarantees to find the solution with most pegs left (26).** But, since all the child states are created and added to the queue, there is too much unnecessary memory usage.

In brief, when the goal is to end up in a random goal state, DFS has a better Time complexity & Spatial performance, however it does not guarantee optimality, whereas BFS performs poorly due to the nature of the solution (solution is not in the initial levels and branching factor increases as the tree goes deeper), but if the goal is to end up in a goal state with few possible moves, BFS guarantees to find that solution.

**c)**

**A\* with minimum possible moves heuristic A\* with minimum pegs left alone heuristic**

To run the A\* algorithm, I implemented several heuristics. Moreover I defined a **priority queue**, and several **comparing operators that overloads the comparing operation.** In order to reach a goal state with minimum moves,I used two different heuristics

**Minimum possible moves heuristic:** Generated states in the priority queue are sorted, by their number of valid moves over the board. Because, in order to reach an optimal solution, we need a state with 0 possible moves left on the board, and we need to reach that state as fast as possible.

**Minimum pegs left alone heuristic:** A similar approach is to calculate the each peg, that is isolated from the other pegs on the board. As this value gets smaller, we get closer to reach to the goal state. Our priority queue sorts the states based on this value.

|  |  |  |
| --- | --- | --- |
|  | **Minimum Moves Left** | **Minimum Alone Pegs** |
| Nodes Generated | **48** | 221 |
| Nodes Expanded | **14** | 34 |
| Max nodes kept in memory | **36** | 189 |
| Running Time | **675 milliseconds** | 1455 milliseconds |
| Pegs left in final state | **26 (optimal)** | **26 (optimal)** |

Although both heuristics reach to the optimal state, it can be seen that *minimum moves left* heuristic over performed *minimum alone pegs* heuristic, since it formulates the goal state better. In the goal state we need a state where there is no possible move left. And since the new states are selected based on this value, implemented by moveCount() function

d)

Finding a solution with one peg left:

**BFS:** Firstly, BFS will affected the worst. Because, the goal state where there is one peg left, is at the bottom of the search tree. Therefore, since BFS searches level by level, it goes through all levels and check goes through all possible states. Time complexity and memory usage will be the worst, when they grow exponentially.

**DFS**: Memory usage will still be less than BFS, however time spent is going to be very similar, since solution is at the deepest level. Memory is still less because DFS searches by going down at each iteration, so it will not create and hold the nodes for each state until it reaches to the final level.

**A\*:** We cannot say a definitive premise, since the runtime and the memory usage depends on how the heuristic explains the solution. We may come up with a heuristic that defines the solution perfectly, and finds the solution with least possible moves, or else, it may never find a solution.

Although in this homework, there is no need to find a solution where 1 peg left, I tried to come up with a different heuristics to solve the problem

**Manhattan Distance:** Total Manhattan distance between each peg pair

**Center Distance**: Total distances between each peg and the center of the board.

**Left over Pegs \* Central Distance:** Product of central distance and number of pegs over the board.

However, none of them led me to the optimal case where there is 1 move left.