MARMARA UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING

ARTIFICIAL INTELLIGENCE

CSE4082

HOMEWORK-1

REPORT

Group Members:

ONURCAN İŞLER 150120825

ERKAM KARACA 150118021

Table of Contents

1. INTRODUCTION

1.1. States

We are required to solve Peg Solitaire game with different approaches. In our approach, we represented the states by boards. Boards are 8x8 dimensional Boolean matrices where pegs are represented by ones, and empty slots are represented by zeros. In the document, it was stated that:

> b. For methods a to c, if there are multiple children to be put inside the frontier list, put the children in such an order that the child node with the smallest numbered peg is removed from the board is selected first.

To provide this feature, we must use a table with slot numbers. The table is as follows:

-1	-1	1	2	3	-1	-1
-1	-1	4	5	6	-1	-1
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
-1	-1	28	29	30	-1	-1
-1	-1	31	32	33	-1	-1

With help of this slots table, we will be able to sort the children nodes by checking the removable pegs' slot numbers. That way removable pegs with smallest slots numbers would be picked first.

1.2. Exploring Children

The way we explore the children states is simple. We determine children states with pegs that are edible in either horizontal direction or vertical direction. The way we check if a peg is edible is very tricky and smart. All we do is XOR the adjacent board elements. If the result is one, then adjacent board elements are 1,0 or 0,1. Which means one of them are filled and the other is empty slot. We apply this rule in two directions:

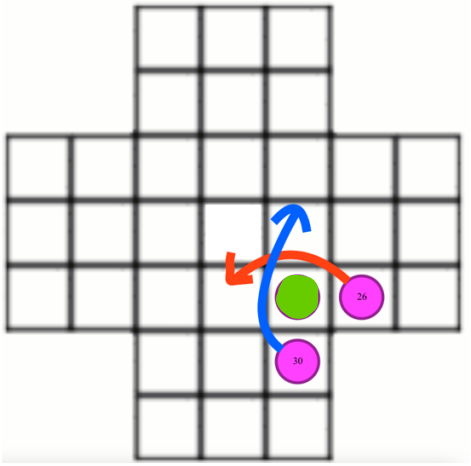
```
isRemovableVertically(i, j, board) || isRemovableHorizontally(i, j, board)
```

Here,

- `(board[i][j+1]^board[i][j-1])==1)` is horizontal check,
- `(board[i+1][j]^board[i-1][j])==1)` is vertical check.

1.3. Deterministic Outputs

First thing to keep in mind that some methods would printout deterministic outputs whereas some may be totally stochastic. In the project document, it was not stated that in which direction pegs will be removed first. See,



Here, the peg with smallest slot number is colored with green. It must be removed but in what direction? So, in our approach, we prefer to eat in vertical direction first to provide deterministic outputs (as indicated in red arrow). Now, let's see which methods in the document are deterministic.

- Standard DFS picking the children with smallest slot number first is deterministic.

- As the name suggests, Random DFS is not deterministic. It is picking the next children purely at random. We shuffle the children and then insert into the stack.

- IDS is deterministic if the direction of eat is defined.

- BFS is also deterministic if the direction of eat is defined.

- DFS with Heuristic is also deterministic. Since comparison results and the heuristic functions will be the same at each run.

2. METHODS

2.1. Breadth-First Search

Here, the frontier is a queue. The removable pegs with smallest slot numbers removed first. Here are the outputs:

```
1. state:
  000
  000
0000000
000.000
0000000
  000
  000
2. state:
  000
  0.0
000.000
0000000
0000000
  000
  000
3. state:
  000
  0.0
0000000
000.000
000.000
  000
  000
4. state:
  000
  0.0
0000000
0..0000
000.000
```

```
  000
  000
5. state:
  000
  0.0
0000000
0.0..00
000.000
  000
  000
6. state:
  000
  0.0
0000000
0.0.0..
000.000
  000
  000
7. state:
  000
  0.0
0000000
0.0.0..
0000000
  0.0
  0.0
Program has terminated early - Memory Limit (500MB)
Sub-optimum Solution Found with 26 remaining pegs.
Search Method: Breadth-First Search.
Time Limit: 60 minutes.
Runtime: 0.3 minutes or 18 seconds.
Number of expanded nodes during the search: 150000
Maximum number of nodes that stored in the memory at any moment: 1247192
Maximum memory size used by the program: 533.492 MB.
```

We got memory error. It is because as we move deeper levels, the more nodes are inserted into the frontier. Eventually, the memory limit will be exceeded. We have kept memory limit at 1GB which is sufficient.

2.2. Depth-First Search

The two differences between BFS and DFS are that frontier is a stack, and children are now reversed. It is because removable pegs with smallest must be put inside the frontier last. It is because stack is a FIFO container. Here are the outputs:

```
1. state:
  000
  000
0000000
000.000
0000000
  000
  000
2. state:
  000
  0.0
000.000
0000000
0000000
  000
  000
3. state:
  000
  0.0
0..0000
0000000
0000000
  000
  000
4. state:
  .00
  ..0
0.00000
0000000
0000000
  000
  000
5. state:
  0..
  ..0
0.00000
0000000
0000000
  000
```

```
  000
6. state:
  0.0
  ...
0.00.00
0000000
0000000
  000
  000
7. state:
  0.0
  0..
0..0.00
00.0000
0000000
  000
  000
8. state:
  ..0
  ...
0.00.00
00.0000
0000000
  000
  000
9. state:
  ..0
  ...
00...00
00.0000
0000000
  000
  000
10. state:
  ..0
  ...
..0..00
00.0000
0000000
  000
  000
11. state:
  ..0
  ...
..0.0..
00.0000
```

```
0000000
  000
  000
12. state:
  ..0
  ..0
..0....
00.0.00
0000000
  000
  000
13. state:
  ...
  ...
..0.0..
00.0.00
0000000
  000
  000
14. state:
  ...
  ...
0.0.0..
.0.0.00
.000000
  000
  000
15. state:
  ...
  ...
000.0..
...0.00
..00000
  000
  000
16. state:
  ...
  ...
000.00.
...0..0
..000.0
  000
  000
17. state:
  ...
  ...
```

```
000.000
...0...
..000..
  000
  000
18. state:
  ...
  ...
000.000
..00...
...00..
  .00
  000
19. state:
  ...
  0..
00..000
...0...
...00..
  .00
  000
20. state:
  ...
  0..
..0.000
...0...
...00..
  .00
  000
21. state:
  ...
  ...
....000
..00...
...00..
  .00
  000
22. state:
  ...
  ...
....000
....0..
...00..
  .00
  000
23. state:
```

```
  ...
  ..O
.....OO
.......
...OO..
  .OO
  OOO
24. state:
  ...
  ..O
....O..
.......
...OO..
  .OO
  OOO
25. state:
  ...
  ...
.......
....O..
...OO..
  .OO
  OOO
26. state:
  ...
  ...
.......
....O..
.....O.
  .OO
  OOO
27. state:
  ...
  ...
.......
....O..
....OO.
  .O.
  OO.
28. state:
  ...
  ...
.......
.......
.....O.
  .OO
```

```
  OO.
29. state:
  ...
  ...
.......
.......
.....O.
  .OO
  ..O
30. state:
  ...
  ...
.......
.......
....OO.
  .O.
  ...
31. state:
  ...
  ...
.......
.......
...O...
  .O.
  ...
32. state:
  ...
  ...
.......
...O...
.......
  ...
  ...
Optimum solution found.
Search Method: Depth-First Search.
Time Limit: 60 minutes.
Runtime: 51.8167 minutes or 3109 seconds.
Number of expanded nodes during the search: 932074602
Maximum number of nodes that stored in the memory at any moment: 159
Maximum memory size used by the program: 5.73 MB.
```

2.3. Iterative Deepening Search

Here, the frontier is a stack, but a depth limit is used. We will not be looking the nodes deeper than some depth limit. It is much more efficient than BFS if the time limit is larger than 10 seconds. Here are the outputs after 60 minutes limit run:

```
1. state:
  000
  000
0000000
000.000
0000000
  000
  000
2. state:
  000
  000
0000000
0000000
000.000
  0.0
  000
3. state:
  000
  000
0000000
0000000
0000..0
  0.0
  000
4. state:
  000
  000
0000000
000.000
000...0
  000
  000
5. state:
  000
  000
0000000
000.000
0000..0
  0.0
```

```
  0.0
6. state:
  000
  000
0000000
000.000
00..0.0
  0.0
  0.0
7. state:
  000
  000
0000000
000.000
..0.0.0
  0.0
  0.0
8. state:
  000
  0.0
000.000
0000000
..0.0.0
  0.0
  0.0
9. state:
  000
  0.0
0000..0
0000000
..0.0.0
  0.0
  0.0
10. state:
  000
  0.0
00..0.0
0000000
..0.0.0
  0.0
  0.0
11. state:
  000
  0.0
..0.0.0
0000000
```

```
..0.0.0
  0.0
  0.0
Sub-optimum Solution Found with 22 remaining pegs.
Search Method: Iterative Deepening Search.
Time Limit: 60 minutes.
Runtime: 60 minutes or 3600 seconds.
Number of expanded nodes during the search: 131650798
Maximum number of nodes that stored in the memory at any moment: 93
Maximum memory size used by the program: 5.82 MB.
```

2.4. Depth-First Search with Random Selection

Here, the frontier is a stack. Before we insert the children of the current node, we shuffle the children array. That way, DFS would pick the next children totally random. Here are the outputs after 60 minutes run:

```
1. state:
  000
  000
0000000
000.000
0000000
  000
  000
2. state:
  000
  0.0
000.000
0000000
0000000
  000
  000
3. state:
  000
  0.0
0000..0
0000000
0000000
  000
  000
4. state:
  000
```

```
  000
000...0
000.000
0000000
  000
  000
5. state:
  000
  000
000..00
000.0.0
00000.0
  000
  000
6. state:
  000
  000
000.000
000...0
0000..0
  000
  000
7. state:
  000
  000
000.000
000...0
00000.0
  00.
  00.
8. state:
  000
  000
000.000
000...0
00000.0
  ..0
  00.
9. state:
  000
  000
0..0000
000...0
00000.0
  ..0
  00.
```

```
10. state:
  000
  000
0..0000
000...0
0000..0
  ...
  000
11. state:
  .00
  .00
0.00000
000...0
0000..0
  ...
  000
12. state:
  .00
  .00
0.00000
00....0
00.0..0
  0..
  000
13. state:
  0..
  .00
0.00000
00....0
00.0..0
  0..
  000
14. state:
  0..
  .0.
0.00.00
00..0.0
00.0..0
  0..
  000
15. state:
  0..
  .0.
0.00.00
00..0.0
..00..0
```

```
  0..
  000
16. state:
  0..
  .0.
00...00
00..0.0
..00..0
  0..
  000
17. state:
  0..
  .0.
00..0..
00..0.0
..00..0
  0..
  000
18. state:
  0..
  .0.
..0.0..
00..0.0
..00..0
  0..
  000
19. state:
  0..
  .0.
..0.0..
00..0.0
.0....0
  0..
  000
20. state:
  0..
  .0.
..0.0.0
00..0..
.0.....
  0..
  000
21. state:
  0..
  .00
..0...0
```

```
00.....
.0.....
  0..
  000
22. state:
  0..
  .00
..0...0
00.....
.00....
  ...
  .00
23. state:
  0..
  .00
..0...0
00.....
.00....
  ...
  0..
24. state:
  0..
  .00
..0...0
..0....
.00....
  ...
  0..
25. state:
  0..
  000
......0
.......
.00....
  ...
  0..
26. state:
  ...
  .00
..0...0
.......
.00....
  ...
  0..
27. state:
  ...
```

```
  O..
..O...O
.......
.OO....
  ...
  O..
28. state:
  ...
  ...
......O
..O....
.OO....
  ...
  O..
29. state:
  ...
  ...
......O
.......
.O.....
  O..
  O..
30. state:
  ...
  ...
......O
.......
.OO....
  ...
  ...
31. state:
  ...
  ...
......O
.......
...O...
  ...
  ...
Sub-optimum Solution Found with 2 remaining pegs.
Search Method: Depth-First Search with Random Selection.
Time Limit: 60 minutes.
Runtime: 60 minutes or 3600 seconds.
Number of expanded nodes during the search: 1059281930
Maximum number of nodes that stored in the memory at any moment: 183
Maximum memory size used by the program: 5.93 MB.
```

Standard DFS were able to find optimal solution under 60 minutes whereas DFS with random selection is not able to achieve the Sub-optimum solutions with single pegs. Maybe it is because DFS with random selection is getting lost some bad branches with no solutions.

2.5. Depth-First Search with a Node Selection Heuristic

Well, maybe the most meaningful part of the homework is this part. We tried many combinations,

- Sum of Manhattan distances of pegs to the center slot,
- Number pegs left on the board,
- Pagoda function matrices,
- The visiting time of the state,
- Manhattan distances between each peg left on the boards etc.

At the end, we decided to use tuples of heuristic functions. For example, in our code, we have tuples as follows:

```
(PagodaMatrixCost, ManhattanDistanceSum, NumberOfPegs, VisitingTime)
```

So, how do we compare the tuples? Consider the tuples below:

- `Tuple-1: (3,6,2,7)`
- `Tuple-2: (3,6,9,7)`

Here, Tuple-1 is smaller than Tuple-2. So, Tuple-1 would be picked first. Since it has less punishment.

Now, we will discuss how do we compute these heuristic values. The number of pegs on the board, and Manhattan distances of pegs to the center slot are very trivial to understand. We mainly discuss Pagoda matrices. What are they?

Pagoda matrices forces pegs to move towards some area in the board. For example, we can force pegs to move towards below by using such a cost matrix below:

		9	9	9		
		8	8	8		
7	7	7	7	7	7	7
6	6	6	6	6	6	6
5	5	5	5	5	5	5
		4	4	4		
		3	3	3		

Above slots have higher costs. So, when we multiply each binary peg value with these costs, the states with many pegs above areas will not be picked. That way, we mainly traverse the states where pegs are in below levels. Here, we built a matrix to push pegs towards to the center slot. Consider the matrix below:

		1	1	1		
		1	0	1		
1	1	0	0	0	1	1
1	0	0	0	0	0	1
1	1	0	0	0	1	1
		1	0	1		
		1	1	1		

It performed, very well at first. However, we were not able arrive optimal solution in 10 minutes of run. We then decided to play these cells and change the weights on these slots. We tried numerous combinations. At the end we found a cost matrix which have found the optimal solution under one minute.

		4	1	4		
		1	0	1		
4	1	3	0	3	1	4
1	0	0	1	0	0	1
4	1	3	0	3	1	4
		1	0	1		
		4	1	4		

We then applied extremely soft touches this matrix to improve even more:

		4	0	4		
		0	0	0		
4	0	3	0	3	0	4
0	0	0	1	0	0	0
4	0	3	0	3	0	4
		0	0	0		
		4	0	4		

The result was incredible. Our DFS Heuristic was able to find the optimal solution by visiting only 35 nodes! The compilation time was literally 0 microseconds! Considering the fact that the optimal solution is hidden at 33rd depth level, this achievement so far, is the best solution ever.

```
1. state:
  000
  000
0000000
000.000
0000000
  000
  000
2. state:
  000
  0.0
000.000
0000000
0000000
  000
  000
3. state:
  000
  0.0
```

```
0..0000
0000000
0000000
  000
  000
4. state:
  .00
  ..0
0.00000
0000000
0000000
  000
  000
5. state:
  .00
  ..0
00..000
0000000
0000000
  000
  000
6. state:
  .00
  ..0
00.0..0
0000000
0000000
  000
  000
7. state:
  .0.
  ...
00.00.0
0000000
0000000
  000
  000
8. state:
  .0.
  ...
00...00
0000000
0000000
  000
  000
9. state:
  .0.
  ...
..0..00
0000000
0000000
  000
  000
10. state:
  .0.
  0..
.....00
00.0000
0000000
  000
  000
11. state:
  .0.
  0..
.....00
0000000
00.0000
  .00
```

```
  000
12. state:
  .0.
  0..
....0..
0000000
00.0000
  .00
  000
13. state:
  .0.
  0.0
.......
0000.00
00.0000
  .00
  000
14. state:
  .0.
  0.0
.......
0000000
00.0.00
  .0.
  000
15. state:
  .0.
  0.0
.......
0000000
..00.00
  .0.
  000
16. state:
  .0.
  0.0
.......
0000000
.0...00
  .0.
  000
17. state:
  .0.
  0.0
.......
0000000
.0..0..
  .0.
  000
18. state:
  .0.
  0.0
.......
0000.00
.0.....
  .00
  000
19. state:
  .0.
  0.0
.......
0000.00
.0..0..
  .0.
  00.
20. state:
  .0.
  0.0
```

```
.......
00..000
.0..0..
  .0.
  00.
21. state:
  .0.
  0.0
.......
00...00
.0.....
  .00
  00.
22. state:
  .0.
  0.0
.......
..0..00
.0.....
  .00
  00.
23. state:
  .0.
  0.0
.......
..0.0..
.0.....
  .00
  00.
24. state:
  .0.
  0.0
.......
..0.0..
.0.....
  .00
  ..0
25. state:
  .0.
  0.0
.......
..0.0..
.0..0..
  .0.
  ...
26. state:
  .0.
  0.0
....0..
..0....
.0.....
  .0.
  ...
27. state:
  .00
  0..
.......
..0....
.0.....
  .0.
  ...
28. state:
  0..
  0..
.......
..0....
.0.....
  .0.
```

```
  ...
29. state:
  ...
  ...
..O....
..O....
.O.....
  .O.
  ...
30. state:
  ...
  ...
.......
.......
.OO....
  .O.
  ...
31. state:
  ...
  ...
.......
.......
...O...
  .O.
  ...
32. state:
  ...
  ...
.......
...O...
.......
  ...
  ...
Optimum solution found.
Search Method: Depth-First Search with a Node Selection Heuristic.
Time Limit: 60 minutes.
Runtime: 0 minutes or 0 seconds.
Number of expanded nodes during the search: 38
Maximum number of nodes that stored in the memory at any moment: 118
Maximum memory size used by the program: 5.42 MB.
```