**Implementation Details:**

**GameState Class:**

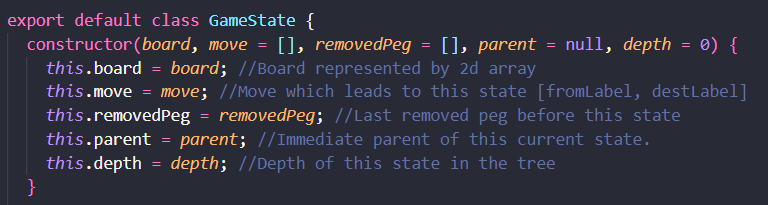
We have a GameState class which holds the current board represented by a 2-D array with 0 for empty slots, 1 for non-empty slots and -1 for invalid slots (slots at the corners).

It has a parent variable which is a reference to another GameState instance. The reason we have parents in our data structure is that when we find a final node, we will be able to iterate all the way up to print all states until the solutions.

Also, we store the depth of each GameState variable to use in Iterative Deepening Search.

Another variable is Move, it is a 2X1 array which holds the labels from and destination. For example, if our move is 14 -> 16 removing the peg 15. Then the array is [14, 16]. We use this variable for visualization only.

We also have a variable for removedPeg. It holds the slot label of the removed peg. It is necessary because we choose the least numbered removed peg in BFS DFS and Iterative Deepening Search.



**Algorithms:**

We have different functions for each algorithm specified in the assignment. And each one of these functions uses an array as frontier and initializes it with the root node which is an instance of GameState given as parameter. And we assign add and remove functions to these arrays to work as queue, stack etc. Also, we have a sort function which defines how should the children states be sorted before added to the frontier. After we define the frontier and the sort function we call the traverseTree function with those parameters.

DFS


Text

Description automatically generatedText

Description automatically generated

**TraverseTree Function**

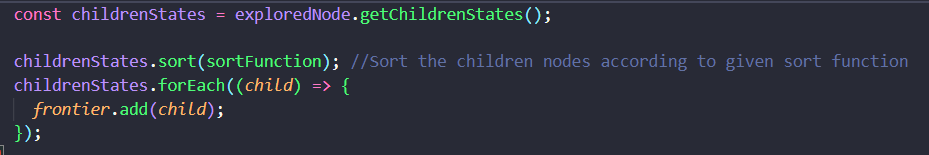
TraverseTree is the common function which all algorithms call. It takes a frontier, a sorting function (to be used for sorting children before adding to frontier). And some options to work with Iterative Deepening Search such as depth limit, isIDS etc.

It first removes the node in the frontier (the order is determined by add and remove function of the frontier as can be seen from above figures). After it removes the top node it checks if it is a solution (it is game over). If it is and its depth is higher than the previous best solution, we update the best solution so far to the newly explored node.

After that we check if the time limit is reached or the solution is the optimal solution, if it is we exit the loop, if it is not we continue to the normal execution.

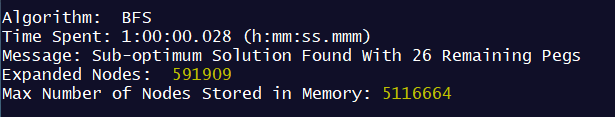
At the end of this checking phase we generate the children states of the newly explored node by calling its method getChildrenStates() which returns all the possible children states of the given node. These generated states are also an instance of the GameState class.

After generating the children, we sort these children with the given sorting function and add all of them to the frontier and continue to the loop.



**Algorithm Outputs**

**a)Breadth-first Search**

****

**NOTE: Below is output to the terminal and since it is a very long output, we copy pasted it instead of taking a screenshot. If you want, you can test our code.**

=== Board States Until the Solution. ===

Move #1: 29 => 17

1 1 1

1 1 1

1 1 1 1 1 1 1

1 1 1 0 1 1 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #2: 10 => 24

1 1 1

1 1 1

1 1 1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 0 1 1 1

1 0 1

1 1 1

Move #3: 19 => 17

1 1 1

1 1 1

1 1 1 0 1 1 1

1 1 1 0 1 1 1

1 1 1 1 1 1 1

1 0 1

1 1 1

Move #4: 16 => 18

1 1 1

1 1 1

1 1 1 0 1 1 1

1 1 1 1 0 0 1

1 1 1 1 1 1 1

1 0 1

1 1 1

Move #5: 14 => 16

1 1 1

1 1 1

1 1 1 0 1 1 1

1 1 0 0 1 0 1

1 1 1 1 1 1 1

1 0 1

1 1 1

Move #6: 2 => 10

1 1 1

1 1 1

1 1 1 0 1 1 1

0 0 1 0 1 0 1

1 1 1 1 1 1 1

1 0 1

1 1 1

1 0 1

1 0 1

1 1 1 1 1 1 1

0 0 1 0 1 0 1

1 1 1 1 1 1 1

1 0 1

1 1 1

**b) Depth-First Search**

**Text

Description automatically generated**

=== Board States Until the Solution. ===

Move #1: 5 => 17

1 1 1

1 1 1

1 1 1 1 1 1 1

1 1 1 0 1 1 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #2: 8 => 10

1 1 1

1 0 1

1 1 1 0 1 1 1

1 1 1 1 1 1 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #3: 1 => 9

1 1 1

1 0 1

1 0 0 1 1 1 1

1 1 1 1 1 1 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #4: 3 => 1

0 1 1

0 0 1

1 0 1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #5: 11 => 3

1 0 0

0 0 1

1 0 1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #6: 16 => 4

1 0 1

0 0 0

1 0 1 1 0 1 1

1 1 1 1 1 1 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #7: 1 => 9

1 0 1

1 0 0

1 0 0 1 0 1 1

1 1 0 1 1 1 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #8: 10 => 8

0 0 1

0 0 0

1 0 1 1 0 1 1

1 1 0 1 1 1 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #9: 7 => 9

0 0 1

0 0 0

1 1 0 0 0 1 1

1 1 0 1 1 1 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #10: 13 => 11

0 0 1

0 0 0

0 0 1 0 0 1 1

1 1 0 1 1 1 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #11: 18 => 6

0 0 1

0 0 0

0 0 1 0 1 0 0

1 1 0 1 1 1 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #12: 3 => 11

0 0 1

0 0 1

0 0 1 0 0 0 0

1 1 0 1 0 1 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #13: 21 => 7

0 0 0

0 0 0

0 0 1 0 1 0 0

1 1 0 1 0 1 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #14: 22 => 8

0 0 0

0 0 0

1 0 1 0 1 0 0

0 1 0 1 0 1 1

0 1 1 1 1 1 1

1 1 1

1 1 1

Move #15: 8 => 10

0 0 0

0 0 0

1 1 1 0 1 0 0

0 0 0 1 0 1 1

0 0 1 1 1 1 1

1 1 1

1 1 1

Move #16: 24 => 22

0 0 0

0 0 0

1 0 0 1 1 0 0

0 0 0 1 0 1 1

0 0 1 1 1 1 1

1 1 1

1 1 1

Move #17: 11 => 9

0 0 0

0 0 0

1 0 0 1 1 0 0

0 0 0 1 0 1 1

0 1 0 0 1 1 1

1 1 1

1 1 1

Move #18: 31 => 23

0 0 0

0 0 0

1 0 1 0 0 0 0

0 0 0 1 0 1 1

0 1 0 0 1 1 1

1 1 1

1 1 1

Move #19: 30 => 18

0 0 0

0 0 0

1 0 1 0 0 0 0

0 0 0 1 0 1 1

0 1 1 0 1 1 1

0 1 1

0 1 1

Move #20: 22 => 24

0 0 0

0 0 0

1 0 1 0 0 0 0

0 0 0 1 1 1 1

0 1 1 0 0 1 1

0 1 0

0 1 1

Move #21: 18 => 16

0 0 0

0 0 0

1 0 1 0 0 0 0

0 0 0 1 1 1 1

0 0 0 1 0 1 1

0 1 0

0 1 1

Move #22: 33 => 31

0 0 0

0 0 0

1 0 1 0 0 0 0

0 0 1 0 0 1 1

0 0 0 1 0 1 1

0 1 0

0 1 1

Move #23: 27 => 25

0 0 0

0 0 0

1 0 1 0 0 0 0

0 0 1 0 0 1 1

0 0 0 1 0 1 1

0 1 0

1 0 0

Move #24: 20 => 18

0 0 0

0 0 0

1 0 1 0 0 0 0

0 0 1 0 0 1 1

0 0 0 1 1 0 0

0 1 0

1 0 0

Move #25: 18 => 30

0 0 0

0 0 0

1 0 1 0 0 0 0

0 0 1 0 1 0 0

0 0 0 1 1 0 0

0 1 0

1 0 0

Move #26: 30 => 28

0 0 0

0 0 0

1 0 1 0 0 0 0

0 0 1 0 0 0 0

0 0 0 1 0 0 0

0 1 1

1 0 0

Move #27: 31 => 23

0 0 0

0 0 0

1 0 1 0 0 0 0

0 0 1 0 0 0 0

0 0 0 1 0 0 0

1 0 0

1 0 0

Move #28: 24 => 22

0 0 0

0 0 0

1 0 1 0 0 0 0

0 0 1 0 0 0 0

0 0 1 1 0 0 0

0 0 0

0 0 0

Move #29: 9 => 23

0 0 0

0 0 0

1 0 1 0 0 0 0

0 0 1 0 0 0 0

0 1 0 0 0 0 0

0 0 0

0 0 0

Move #30: 22 => 24

0 0 0

0 0 0

1 0 0 0 0 0 0

0 0 0 0 0 0 0

0 1 1 0 0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

1 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 1 0 0 0

0 0 0

0 0 0

**d) Depth-First Search with Random Selection**

**Text

Description automatically generated**

=== Board States Until the Solution. ===

Move #1: 19 => 17

1 1 1

1 1 1

1 1 1 1 1 1 1

1 1 1 0 1 1 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #2: 6 => 18

1 1 1

1 1 1

1 1 1 1 1 1 1

1 1 1 1 0 0 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #3: 25 => 11

1 1 1

1 1 0

1 1 1 1 0 1 1

1 1 1 1 1 0 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #4: 33 => 25

1 1 1

1 1 0

1 1 1 1 1 1 1

1 1 1 1 0 0 1

1 1 1 1 0 1 1

1 1 1

1 1 1

Move #5: 31 => 33

1 1 1

1 1 0

1 1 1 1 1 1 1

1 1 1 1 0 0 1

1 1 1 1 1 1 1

1 1 0

1 1 0

Move #6: 4 => 6

1 1 1

1 1 0

1 1 1 1 1 1 1

1 1 1 1 0 0 1

1 1 1 1 1 1 1

1 1 0

0 0 1

Move #7: 24 => 32

1 1 1

0 0 1

1 1 1 1 1 1 1

1 1 1 1 0 0 1

1 1 1 1 1 1 1

1 1 0

0 0 1

Move #8: 23 => 31

1 1 1

0 0 1

1 1 1 1 1 1 1

1 1 1 1 0 0 1

1 1 1 0 1 1 1

1 0 0

0 1 1

Move #9: 21 => 23

1 1 1

0 0 1

1 1 1 1 1 1 1

1 1 1 1 0 0 1

1 1 0 0 1 1 1

0 0 0

1 1 1

Move #10: 17 => 5

1 1 1

0 0 1

1 1 1 1 1 1 1

1 1 1 1 0 0 1

0 0 1 0 1 1 1

0 0 0

1 1 1

Move #11: 2 => 10

1 1 1

0 1 1

1 1 1 0 1 1 1

1 1 1 0 0 0 1

0 0 1 0 1 1 1

0 0 0

1 1 1

Move #12: 16 => 4

1 0 1

0 0 1

1 1 1 1 1 1 1

1 1 1 0 0 0 1

0 0 1 0 1 1 1

0 0 0

1 1 1

Move #13: 7 => 9

1 0 1

1 0 1

1 1 0 1 1 1 1

1 1 0 0 0 0 1

0 0 1 0 1 1 1

0 0 0

1 1 1

Move #14: 6 => 18

1 0 1

1 0 1

0 0 1 1 1 1 1

1 1 0 0 0 0 1

0 0 1 0 1 1 1

0 0 0

1 1 1

Move #15: 13 => 11

1 0 1

1 0 0

0 0 1 1 0 1 1

1 1 0 0 1 0 1

0 0 1 0 1 1 1

0 0 0

1 1 1

Move #16: 10 => 8

1 0 1

1 0 0

0 0 1 1 1 0 0

1 1 0 0 1 0 1

0 0 1 0 1 1 1

0 0 0

1 1 1

Move #17: 14 => 16

1 0 1

1 0 0

0 1 0 0 1 0 0

1 1 0 0 1 0 1

0 0 1 0 1 1 1

0 0 0

1 1 1

Move #18: 26 => 24

1 0 1

1 0 0

0 1 0 0 1 0 0

0 0 1 0 1 0 1

0 0 1 0 1 1 1

0 0 0

1 1 1

Move #19: 18 => 6

1 0 1

1 0 0

0 1 0 0 1 0 0

0 0 1 0 1 0 1

0 0 1 1 0 0 1

0 0 0

1 1 1

Move #20: 3 => 11

1 0 1

1 0 1

0 1 0 0 0 0 0

0 0 1 0 0 0 1

0 0 1 1 0 0 1

0 0 0

1 1 1

Move #21: 1 => 9

1 0 0

1 0 0

0 1 0 0 1 0 0

0 0 1 0 0 0 1

0 0 1 1 0 0 1

0 0 0

1 1 1

Move #22: 8 => 10

0 0 0

0 0 0

0 1 1 0 1 0 0

0 0 1 0 0 0 1

0 0 1 1 0 0 1

0 0 0

1 1 1

Move #23: 16 => 28

0 0 0

0 0 0

0 0 0 1 1 0 0

0 0 1 0 0 0 1

0 0 1 1 0 0 1

0 0 0

1 1 1

Move #24: 10 => 12

0 0 0

0 0 0

0 0 0 1 1 0 0

0 0 0 0 0 0 1

0 0 0 1 0 0 1

1 0 0

1 1 1

Move #25: 27 => 13

0 0 0

0 0 0

0 0 0 0 0 1 0

0 0 0 0 0 0 1

0 0 0 1 0 0 1

1 0 0

1 1 1

Move #26: 13 => 11

0 0 0

0 0 0

0 0 0 0 0 1 1

0 0 0 0 0 0 0

0 0 0 1 0 0 0

1 0 0

1 1 1

Move #27: 31 => 23

0 0 0

0 0 0

0 0 0 0 1 0 0

0 0 0 0 0 0 0

0 0 0 1 0 0 0

1 0 0

1 1 1

Move #28: 23 => 25

0 0 0

0 0 0

0 0 0 0 1 0 0

0 0 0 0 0 0 0

0 0 1 1 0 0 0

0 0 0

0 1 1

Move #29: 33 => 31

0 0 0

0 0 0

0 0 0 0 1 0 0

0 0 0 0 0 0 0

0 0 0 0 1 0 0

0 0 0

0 1 1

0 0 0

0 0 0

0 0 0 0 1 0 0

0 0 0 0 0 0 0

0 0 0 0 1 0 0

0 0 0

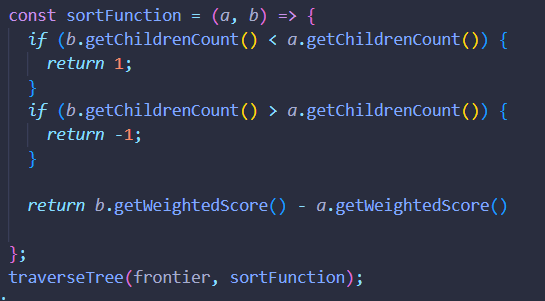
1 0 0

**e) Depth-First Search With a Node Selection Heuristic**

**Heuristic Function:**

In our heuristic function, we first choose the node which yields more children states. And if they are equal, we choose the node which has the least weighted score.

Weighted score is the sum of all the Euclidean distances from each peg and center for each non-empty slot. In this way, we get an optimal Solution in less than 1 second by expanding 19057 nodes and storing maximum of 298 nodes in the memory.

****

Text

Description automatically generated

=== Board States Until the Solution. ===

Move #1: 29 => 17

1 1 1

1 1 1

1 1 1 1 1 1 1

1 1 1 0 1 1 1

1 1 1 1 1 1 1

1 1 1

1 1 1

Move #2: 26 => 24

1 1 1

1 1 1

1 1 1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 0 1 1 1

1 0 1

1 1 1

Move #3: 11 => 25

1 1 1

1 1 1

1 1 1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 1 0 0 1

1 0 1

1 1 1

Move #4: 9 => 11

1 1 1

1 1 1

1 1 1 1 0 1 1

1 1 1 1 0 1 1

1 1 1 1 1 0 1

1 0 1

1 1 1

Move #5: 23 => 9

1 1 1

1 1 1

1 1 0 0 1 1 1

1 1 1 1 0 1 1

1 1 1 1 1 0 1

1 0 1

1 1 1

Move #6: 14 => 16

1 1 1

1 1 1

1 1 1 0 1 1 1

1 1 0 1 0 1 1

1 1 0 1 1 0 1

1 0 1

1 1 1

Move #7: 31 => 23

1 1 1

1 1 1

1 1 1 0 1 1 1

0 0 1 1 0 1 1

1 1 0 1 1 0 1

1 0 1

1 1 1

Move #8: 33 => 31

1 1 1

1 1 1

1 1 1 0 1 1 1

0 0 1 1 0 1 1

1 1 1 1 1 0 1

0 0 1

0 1 1

Move #9: 16 => 28

1 1 1

1 1 1

1 1 1 0 1 1 1

0 0 1 1 0 1 1

1 1 1 1 1 0 1

0 0 1

1 0 0

Move #10: 4 => 16

1 1 1

1 1 1

1 1 1 0 1 1 1

0 0 0 1 0 1 1

1 1 0 1 1 0 1

1 0 1

1 0 0

Move #11: 7 => 9

1 1 1

0 1 1

1 1 0 0 1 1 1

0 0 1 1 0 1 1

1 1 0 1 1 0 1

1 0 1

1 0 0

Move #12: 2 => 10

1 1 1

0 1 1

0 0 1 0 1 1 1

0 0 1 1 0 1 1

1 1 0 1 1 0 1

1 0 1

1 0 0

Move #13: 6 => 18

1 0 1

0 0 1

0 0 1 1 1 1 1

0 0 1 1 0 1 1

1 1 0 1 1 0 1

1 0 1

1 0 0

Move #14: 13 => 11

1 0 1

0 0 0

0 0 1 1 0 1 1

0 0 1 1 1 1 1

1 1 0 1 1 0 1

1 0 1

1 0 0

Move #15: 18 => 6

1 0 1

0 0 0

0 0 1 1 1 0 0

0 0 1 1 1 1 1

1 1 0 1 1 0 1

1 0 1

1 0 0

Move #16: 3 => 11

1 0 1

0 0 1

0 0 1 1 0 0 0

0 0 1 1 0 1 1

1 1 0 1 1 0 1

1 0 1

1 0 0

Move #17: 10 => 12

1 0 0

0 0 0

0 0 1 1 1 0 0

0 0 1 1 0 1 1

1 1 0 1 1 0 1

1 0 1

1 0 0

Move #18: 27 => 13

1 0 0

0 0 0

0 0 1 0 0 1 0

0 0 1 1 0 1 1

1 1 0 1 1 0 1

1 0 1

1 0 0

Move #19: 13 => 11

1 0 0

0 0 0

0 0 1 0 0 1 1

0 0 1 1 0 1 0

1 1 0 1 1 0 0

1 0 1

1 0 0

Move #20: 30 => 18

1 0 0

0 0 0

0 0 1 0 1 0 0

0 0 1 1 0 1 0

1 1 0 1 1 0 0

1 0 1

1 0 0

Move #21: 24 => 10

1 0 0

0 0 0

0 0 1 0 1 0 0

0 0 1 1 1 1 0

1 1 0 1 0 0 0

1 0 0

1 0 0

Move #22: 31 => 23

1 0 0

0 0 0

0 0 1 1 1 0 0

0 0 1 0 1 1 0

1 1 0 0 0 0 0

1 0 0

1 0 0

Move #23: 16 => 28

1 0 0

0 0 0

0 0 1 1 1 0 0

0 0 1 0 1 1 0

1 1 1 0 0 0 0

0 0 0

0 0 0

Move #24: 21 => 23

1 0 0

0 0 0

0 0 1 1 1 0 0

0 0 0 0 1 1 0

1 1 0 0 0 0 0

1 0 0

0 0 0

Move #25: 28 => 16

1 0 0

0 0 0

0 0 1 1 1 0 0

0 0 0 0 1 1 0

0 0 1 0 0 0 0

1 0 0

0 0 0

Move #26: 16 => 4

1 0 0

0 0 0

0 0 1 1 1 0 0

0 0 1 0 1 1 0

0 0 0 0 0 0 0

0 0 0

0 0 0

Move #27: 1 => 9

1 0 0

1 0 0

0 0 0 1 1 0 0

0 0 0 0 1 1 0

0 0 0 0 0 0 0

0 0 0

0 0 0

Move #28: 18 => 6

0 0 0

0 0 0

0 0 1 1 1 0 0

0 0 0 0 1 1 0

0 0 0 0 0 0 0

0 0 0

0 0 0

Move #29: 9 => 11

0 0 0

0 0 1

0 0 1 1 0 0 0

0 0 0 0 0 1 0

0 0 0 0 0 0 0

0 0 0

0 0 0

Move #30: 6 => 18

0 0 0

0 0 1

0 0 0 0 1 0 0

0 0 0 0 0 1 0

0 0 0 0 0 0 0

0 0 0

0 0 0

Move #31: 19 => 17

0 0 0

0 0 0

0 0 0 0 0 0 0

0 0 0 0 1 1 0

0 0 0 0 0 0 0

0 0 0

0 0 0

0 0 0

0 0 0

0 0 0 0 0 0 0

0 0 0 1 0 0 0

0 0 0 0 0 0 0

0 0 0

0 0 0

**Note:** If you want to run this program, you need to have Node.js installed in your computer. To run the program after you install node.js go to the project directory and run first **“npm install”** and after you install the dependencies run **“node src/index.js”.**