This algorithm performs a broad-first exploration in order to find the shortest path between 2 configurations of the Tower of London test. It is presented in a pseudo-programming language. Comments are enclosed in boxes

The input of the problem are the initial and final configurations

initialConfiguration : node
finalConfiguration : node

This function determines the shortest path between the two configurations

randomBroadFirstExploration ( )

local variables

Sequence of nodes composing the path currently examined
sequenceOfNodes : array of nodes

Sequence of branching factors, i.e., number of successors of the nodes composing the path currently examined (values = 2, 3 or 4).
sequenceOfBranchingFactors : array of integer

Sequence of moves composing the path currently examined
(identified by a number between 0 and sequenceOfBranchingFactors (i) -1
sequenceOfMoves : array of integers

Sequence of randomized successors corresponding to the path currently examined.
sequenceOfSuccessors (i, j) contains the j-th successor of sequenceOfNodes (i)
sequenceOfSuccessors : array of array of nodes

Number of steps, i.e., comparisons between configurations
numberOfSteps : integer

Initialization

lastIndexOfSequence = 0
sequenceOfNodes (0) = initialConfiguration ;
sequenceOfMoves (0) = 0
sequenceOfSuccessors (0, 0) = initialConfiguration
sequenceOfBranchingFactors (0) = 1

Loop until the targetConfiguration is reached

while true

```
gets the current node, current move and current branching factor
currentMove = sequenceOfMoves ( lastIndexOfSequence )
currentBranchingFactor = sequenceOfBranchingFactors ( lastIndexOfSequence )
currentNode = sequenceOfNodes ( lastIndexOfSequence )


one more step (comparison between configurations)
numberOfSteps ++ ;

target is found
if  currentConfiguration = targetConfiguration
        exit loop

increments the sequence of move numbers (this may increase lastIndexOfSequence)
nextSequenceOfMoves ( )

end of loop While

determines the length of the solution sequence
lengthOfSolution = lastIndexOfSequence + 1 ;


end of function randomBroadFirstExploration


This function generates the next sequence to be explored. It increments the length of the
current sequence only when all the sequences of the same length have been explored.
It returns the length of the

nextSequenceOfMoves ( ) : integer


/* finds the first index of the sequence that still has possible explorations (goes backwards) */
index = lastIndexOfSequence ;
while true

    if anything was explored : increases the length of the paths
    if index = 0
            increment lastIndexOfSequence

        resets the rest of the sequence (see next function)
        resetEndOfSequence ( )
        exit loop while
        end of if
```

```
        there are possible moves at the current index, there are moves that are not already explored
        if sequenceOfMoves ( index )  + 1 < sequenceOfBranchingFactors ( index )

                increment sequenceOfMoves ( index )
                currentMove =  sequenceOfMoves ( index )

                updates current node
                sequenceOfNodes ( index ) = sequenceOfSuccessors ( index, currentMove )


                resets the rest of the sequence (see next function)
                resetEndOfSequence ( )

                end of if


        goes to the previous node  in the sequence
        decrement index

        end of loop while

end of function nextSequenceOfMoves


This function resets the end of the current sequence, from the node index. It resets the sequence of nodes, of branching factors, of moves (begins with move number 0) and of successors. It randomizes the successors.

resetEndOfSequence ( )


while index < lastIndexOfSequence

        previous configuration
        previousConfiguration = sequenceOfNodes ( index )

        goes to next move of the sequence
        increment index

        determines the number of moves , branching factor and successors
        fill sequenceOfSuccessors ( index , * )  with the successors of previousConfiguration in random order

        sequenceOfBranchingFactors ( index ) = number of successors of previousConfiguration

        the current move is set to 0, i.e., the first possible move, the current node is determined from the successors
```

```
        sequenceOfMoves ( index ) = 0
        sequenceOfNodes ( index ) = sequenceOfSuccessors ( index, 0 )


        the current number of successors is set to the move is set to 0, i.e., the first possible
        move, the current node is determined from the successors
        sequenceOfMoves ( index ) = 0

        end of loop while

end of function resetEndOfSequence
```