-

Design And Analysis Of Algorithm

Group Members

- Zeeshan Ali (197)
- Shayan Hassan Abbasi (167)
- Muhammad Sheraz Ajmal (130)

Submitted to: Ma’am Umarrah Qaseem

****

Assignment # 03

Puzzle # 73: **Rooster Chase**

Problem Statement:

This game is played on the grid. The F counter represents a farmer; the R counter represents a rooster. The farmer and the rooster move alternately until the rooster is captured. On each move, each of them can move to a neighboring point on the grid: up, down, left, or right. A capture occurs when the farmer can move on a point occupied by the rooster.

Objective:

The objective of the Rooster Chase problem is to implement the game logic and simulate the movements of the farmer and the rooster on the grid, checking for the capture conditions and determining the final outcome of the game.

Algorithm (Pseudo-Code):

Algorithm RoosterChase(rF, cF, rR, cR)

Input: Starting Positions of the Rooster and Farmer represented through Row and Column

Output: Prints the message if Rooster is captured or not

Gridsize ← 8

For move ← 1 TO 14 Do
For i ← 0 to gridsize-1 DO
For j ← 0 to gridsize-1 DO
if i=rF-1 && j= cF-1 THEN
Grid[i][j] ← ‘F’
Else if i=rR-1 && j= cR-1
Grid[i][j] ← ‘R’
Else
Grid[i][j] ← ‘-’
Print grid[i][j] + “ ”
End if
End for
End for
IF rF-rR = 1 && cF-cR = 0 || cF -rR = 0 && cF-cR = 1|| rR-rF =0 && cR-cF=1 ||rR-rF =1 && cR-cF=0 Then
Grid[rF-1] [rF-1] = ‘F’
Grid [rR-1][cR-1] = ‘R’
For i ← 0 to gridsize-1 do
For j ← 0 to gridsize-1 do
Print grid[i][j] + “ ”

```
                                    End for
                                End for
                            Print “Rooster captured”
                            Return
                            End if
                           if rR>rF Then
                                rR++
                            else if cR >cF
                                cR--
                           End if
                           if rR = 1 && cR =1
                                Grid[rR-1] [cR-1] ← ‘R’
                                For I ← 0 to gridsize -1 do
                                        For j=0 to gridsize -1 do
                                                Print grid[i][j] + “ ”
                                        End for
                                End for
                             Print “Rooster is captured in the upper corner”
                            Return
                            End if
                           rowDistance = rR - rF
                           colDistance = cR - cF
                           d = max(rowDistance, colDistance)
                           if colDistance > rowDistance then
                                        cF++
                           else
                                        rF--
                           End if
                            if rF == rR and cF == cR then
                                        Print “Rooster captured”
                            Return
                            End if
                           if rF <1|| rF > gridsize || cF < 1 || cF> gridsize || rR < 1 ||
rR > gridsize || cR < 1 || cR > gridsize
                                        Print “ Rooster is not captured”
                            Return
                            End if
                End for
```

Efficiency Analysis (Time Complexity):

Step 1: Creating and displaying the grid

- The nested loop iterates over the grid of size 8x8.
- The outer loop runs for gridSize iterations, which is 8 in this case.
- The inner loop also runs for gridSize iterations, which is 8 as well.

- Therefore, the total number of iterations in the nested loop is gridSize x gridSize = 8 x 8 = 64, which is a constant number.
- As a result, the time complexity for this step is O(1).

Step 2: Checking if the farmer and rooster are adjacent
- The code performs a constant number of arithmetic comparisons to check if the farmer and rooster are adjacent.
- This step has a constant time complexity of O(1).

Step 3: Updating the rooster's position
- The code updates the rooster's position based on some conditions.
- The conditions involve simple arithmetic operations and assignment statements, which can be considered constant time operations.
- Thus, the time complexity for this step is O(1).

Step 4: Checking if the rooster is captured in the upper right corner
- The code performs a constant number of comparisons to check if the rooster's position is in the upper right corner.
- This step has a constant time complexity of O(1).

Step 5: Updating the farmer's position.
- Like step 4, this step involves updating the farmer's position based on some conditions, which are constant time operations.
- The time complexity for this step is O (1).

Step 6: Checking if the farmer or rooster is out of bounds.
- The code performs a constant number of comparisons to check if the farmer or rooster has moved outside the grid boundaries.
- This step has a constant time complexity of O(1).

Since each step has a constant time complexity of O(1), we can analyze the overall time complexity for the loop:
The loop runs for a maximum of 14 moves, and each move has a time complexity of O(1). Therefore, the overall time complexity of the loop can be expressed as O(1) * 14, which simplifies to O(14) or simply O(1).

Overall time complexity = O(1) * 14 + O(1) + O(1) = O(14) + O(1) + O(1) = **O(1).**

1. Asymptotic Upper Bound: (WORST CASE)

Big- O: 11383 is O(1).

Here f(n) is 11383 and g(n) is 1 because f(n) = O(g(n)).

f(n) <= c * g(n).

11383 <= c * 1.

Here c = 11383.

11383 <= 11383.

2. Asymptotic Lower Bound: (BEST CASE)

Big- Ω: 16 is Ω(1).

Here f(n) is 16 and g(n) is 1 because f(n) = Ω(g(n)).

$$f(n) >= c * g(n).$$

$$16 >= c * 1.$$

Here c = 16.

$$16 >= 16.$$

3. Asymptotic Tight Bound: (BEST OF WORST CASES)

In a problem, we have given a fixed moves which is 14 for 8x8 so best of worst cases will also remain same as worst case.

Big- θ: 11383 is θ(1).

Here f(n) is 11383 and g(n) is 1 for upper and lower bounds.

$$c_1 * g(n) <= f(n) <= c_2 * g(n).$$

$$c_1 * 1 <= 11383 <= c_2 * 1.$$

Here c_1 for Best case = 11383 and c_2 for Worst case = 11383.

$$11383 <= 11383 <= 11383.$$

Space Complexity:

1. Grid Storage:

- The algorithm uses a 2D array Grid to represent the positions of the Rooster and Farmer on the chessboard.
- The grid has dimensions gridsize x gridsize = 8x8 in this case.
- Therefore, the space complexity for the grid is O(64) in this specific case which results into O(1).

2. Variables:

- The algorithm uses variables **(rF, cF, rR, cR, move, i, j, rowDistance, colDistance, d)** that are not dependent on the size of the input but are rather constant in terms of space complexity which is O(1).
- These variables do not contribute significantly to the space complexity.

Total Space Complexity:

Hence the Total Space Complexity will be constant which can be expressed as O(1), indicating that the space required by the algorithm is constant or does not depend on the input size.

Conclusion:

This problem is solved by using *Greedy* algorithm strategy. The functionality of this algorithm is to show whether the Farmer chased down the Rooster or not within 14 Moves. All the necessary conditions and requirements were kept in mind while designing the algorithm.