Owen Lindsey

Professor Demland, David

CST-201 Complexity

10/06/2024

Number of Moves for The Largest Disk

Pattern Explanation:

1. The smallest disk (i = 1) moves every other step.

2. The second smallest disk (i = 2) moves every fourth step

3. The third smallest disk (i = 3) moves every eighth step.

4. This will continue to grow exponentially per disk.

The ith Largest Disk Explanation:

Now to find out how many moves are made by the ith largest disk:

1. Total number of moves in the Tower of Hanoi solution:

2^n – 1.

2. Frequency of moves for the ith disk:

Every 2^i steps.

The number of moves for the ith largest disk algorithm is:

2^(n-i) - (1/2^i)

However we cannot count half a move so rounding down this value will give us an approximate answer.

The final answer:

Number of moves for the ith largest disk = floor(2^(n-i))

- n is the total number of disks.
- i is the size of the disk (1<= i <= n), with 1 being the smallest disk
- floor() means we round down to the nearest integer.

This algorithm shows:

- The smallest disk (i = 1) moves 2^(n-1) times.
- The largest disk (i = n) moves only once.
- Each disk moves half as many times as the disk below it.

Pseudo code for Tower of Hanoi

Initialize Towers:

METHOD *InitializeTowers(number_of_disks)*

CREATE *three empty stacks to represent towers A, B, and C*

FOR *each disk from largest to smallest*

PUSH *disk onto tower A*

END FOR

DISPLAY *initial state of towers*

Move Disks Iteratively:

METHOD *MoveDisksIterative(number_of_disks,source,destination,auxilary)*

SET *total_moves to (2* ^number_of_disks)-1

FOR *move_number from 1 to total_moves*

    CALCULATE *source_tower using bitwise operations on move_number*

    CALCULATE *destination_tower using bitwise operations on move_number*

    MAP *calculated indices to actual tower indices (source, auxiliary, destination)*

    CALL *MoveSingleDisk with source_tower, destination_tower, and number_of_disks*

END FOR

Move Single Disk:

METHOD *MoveSingleDisk(source,destination,total_disks)*

IF *source tower is not empty* ***AND*** *(destination_tower is empty OR top disk on source is smaller than top disk on destination)*

THEN

REMOVE *top disk from the source tower*

ADD *removed disk to destination tower*

DISPLAY *current state of towers*

END IF

Pseudo code for Tower of Hanoi

Print Towers:

METHOD *PrintTowers(total_disks)*

CLEAR *console display*

CALCULATE *width for each towers display area*

FOR *each level from top to bottom of towers*

FOR *each tower (A, B, C)*

    IF *tower has a disk at this level* ***THEN***

        PRINT disk with appropriate size and color

    ELSE

PRINT empty space with center pole

    END IF

    MOVE *to next line*

END FOR

PRINT *labels (A, B, C) for towers*

PAUSE *briefly to show towers*

Main:

Method *Main*

PROMPT *user for number of disks*

READ *number_of_disks from user input*

CALL *initializeTowers with number_of_disks*

CALL *MoveDisksIterative with:*

- *number_of_disks*
- *source tower index (0 for A)*
- *destination tower index (2 for C)*
- *auxiliary tower index (1 for B)*

DISPLAY *"Puzzle solved!" message*

Bias and Consequences within the Tower of Hanoi Puzzle

- ***Scale Insensitivity Bias****:*

  Our algorithm scales up to handle any number of disks, but it doesn't consider practical limitations that might arise with extremely large numbers.

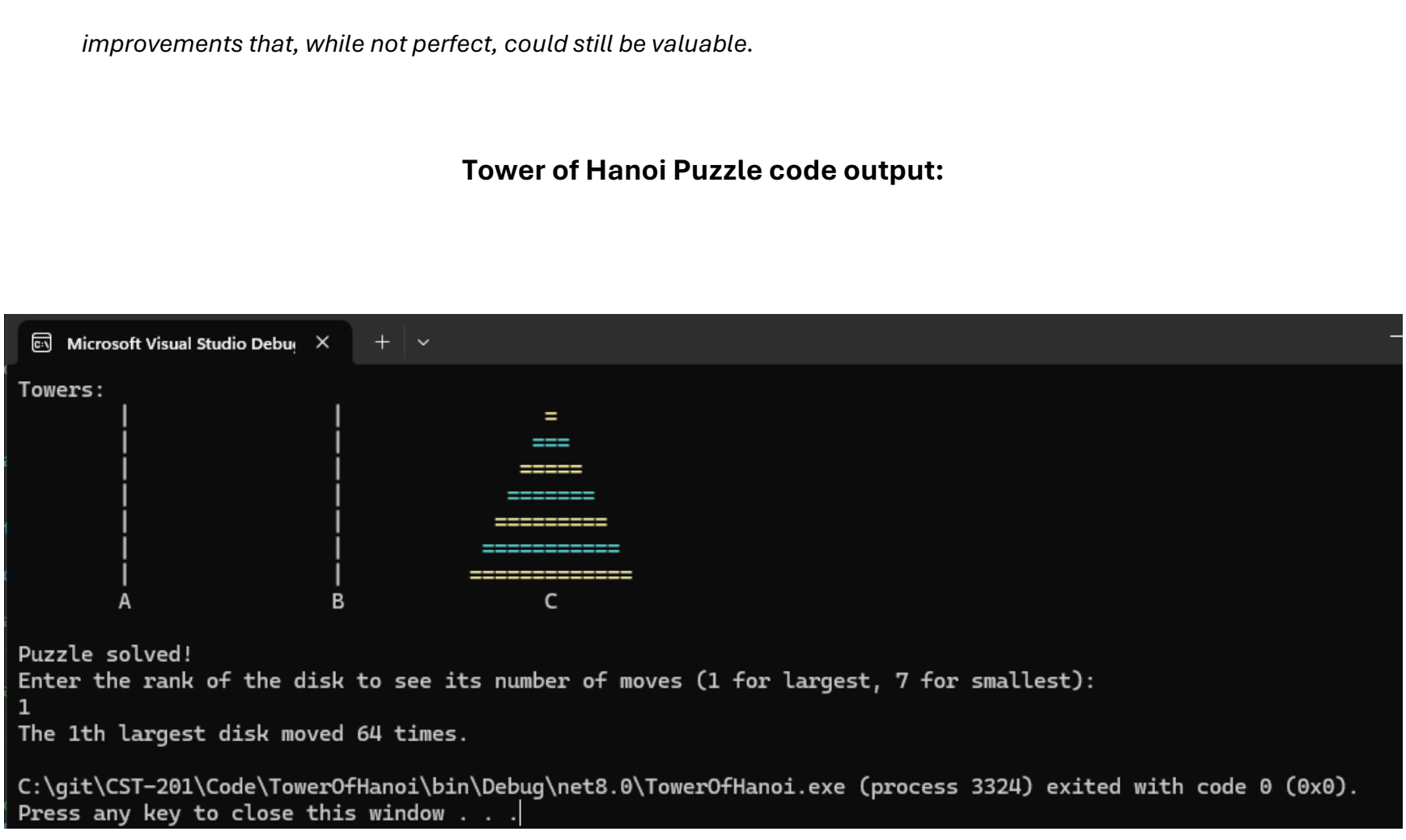
  Consequence: *In real-world applications, this could lead to proposing solutions that are theoretically correct but practically infeasible due to time, resource, or physical constraints.*

- ***Binary Outcome Bias****:*

  The Tower of Hanoi problem has a clear, binary outcome - either the puzzle is solved or is not solved. My algorithm is designed with this binary success/failure mentality.

  Consequence: *In more nuanced real-world problems, this bias could lead to overlooking partial solutions or improvements that, while not perfect, could still be valuable.*

Tower of Hanoi Puzzle code output:



Video Explanation for Tower of Hanoi Puzzle

Loom Video: *Tower Of Hanoi Problem implementation by Owen Lindsey*