4- Tower Of Hanoi

Problem Description

There are eight disks of different sizes and four pegs. Initially, all the disks are on the first peg in order of size, the largest on the bottom and the smallest on the top. It's required to use divide and conquer algorithm to transfer all the disks to another peg by a sequence of moves.
The constraints are that only one disk can be moved at a time, and it is forbidden to place a larger disk on top of a smaller one.
Also it's required to use dynamic programming algorithm to transfer all the disks to another peg in 33 moves

Detailed Assumptions

- The disks are labeled with consecutive integers starting from 1. Each disk is represented by an object of Disk class, which has a disk_num attribute.
- The moves variable is used to keep track of the total number of moves performed during solving the Tower of Hanoi Problem.
- The Tower of Hanoi problem is being solved optimally, it aims to minimize the number of moves required to transfer the disks from the starting peg to the end peg.

Detailed Solution (pseudocode and steps description)

```
/*
 * Description : Function to print the number of moves required to solve the puzzle
 * Input : number of disks, intial peg, last peg
 * Output : none
 */
function printMove(disk_num, from, to):
    print "Move Disk " + disk_num + " from peg " + from + " to peg " + to

/*
 * Description : Function to save the number of moves to solve the puzzle in an array
 * Input : number of disks
 * Output : none
 */
function getOptimalDiskNum(n):
    // Arrays to store the minimum moves and corresponding k values
    create an array dp of size n + 1
    create an array optimalK of size n + 1


 // Base cases for dp array
    dp[0] = 0   //number of moves in case of zero disks
    dp[1] = 1   //number of moves in case of one disks
    dp[2] = 3   //number of moves in case of two disks

    //the case of more than two disks
    // Filling the dp and optimalK arrays
    for i = 3 to n:
        minMoves = infinity

        for k = 1 to i - 1:
            moves = 2 * dp[k] + (2^(i-k)) - 1

            if moves < minMoves:
                minMoves = moves
                optimalK[i] = i - k       /* Update the optimal k for this i using
                                             the result of k stored previously in the
                                             optimalK array */
            dp[i] = minMoves       /*storing the min number of moves in the corresponding index that
                                    indicates the number of disks */
```

```
/*
 * Description : Function to solve the tower of hanoi using only three pegs
 * Input : disks - disks array
 *         start - starting rod
 *         aux_peg - the auxiliary rod
 *         end - the target rod
 *
 * Output : none
 */
function hanoi3(disks, start, aux_peg, end):
//base case of number of disks equals 0
    if length of disks == 0:
        return
//base case of number of disks equals 1
    if length of disks == 1:
        printMove(disks[0].disk_num, start, end)
        return
//the case of number of disks more than one disk
    n = length of disks
    rem_disks = copy of the first n-1 disks

    hanoi3(rem_disks, start, end, aux_peg)
    printMove(disks[n-1].disk_num, start, end)
    hanoi3(rem_disks, aux_peg, start, end)
```

```

/*
 * Description : Function to solve the tower of hanoi using four pegs
 * Input : n - number of disks
 *         start - starting rod
 *         aux_peg_1 - the first auxiliary rod
 *         aux_peg_2 - the second auxiliary rod
 *         end - the target rod
 * Output : none
 */
function hanoi4(disks, start, aux_peg_1, aux_peg_2, end):
    n = length of disks
//base case of number of disks equals 0
    if n == 0:
        return
//base case of number of disks equals 1
    if n == 1:
        printMove(disks[0].disk_num, start, end)
        return
//base case of number of disks equals 2
    if n == 2:
        printMove(disks[0].disk_num, start, aux_peg_2)
        printMove(disks[1].disk_num, start, end)
        printMove(disks[0].disk_num, aux_peg_2, end)
        return

// one of the two following values of k is used according to the chosen running algorithm that the two cases are separate
    // the value of k in case of divide and conquer algorithm is that the number of disks is divided into two sub-
        problems
    k=n/2

    // the value of k in case of dynamic programming
    if optimalK is null:
        getOptimalDiskNum(n)

    k = optimalK[n]
    rem_disks = copy of the first n-k disks
    fixed_disks = copy of the last k disks

    hanoi4(rem_disks, start, aux_peg_2, end, aux_peg_1)
    hanoi3(fixed_disks, start, aux_peg_2, end)
 hanoi4(rem_disks, aux_peg_1, start, aux_peg_2, end)

```

```
function main():
    n = 8
    create an array disks of size n

    for i = 0 to n-1:
        disks[i] = new Disk(i + 1)

    hanoi4(disks, 'A', 'B', 'C', 'D')
    print "no of moves = " + moves

class Disk:
//attribute of object from class Disk to determine the number of the disk
    int disk_num
    Disk(disk_num):
        this.disk_num = disk_num
```

Complexity Analysis

1. getOptimalDiskNum function:
    - Time Complexity: O(n^2)
    - Space Complexity: O(n)
2. hanoi3 function:
    - Time Complexity: O(2^n)
    - Space Complexity: O(n)
3. hanoi4 function:
    - Time Complexity: O(2^n)
    - Space Complexity: O(n)
4. main function:
    - Time Complexity: O(2^n)
    - Space Complexity: O(n)

The resulted Time Complexity: O(2^n)

The resulted Space Complexity: O(n)

Comparison

	Divide and Conquer	Decrease and Conquer
Advantages	Efficient for large problem size in case the number of disks increase	Simplicity as it reduces the problem by one until reaches the base case Lower Memory requirements as it often operate on the problem in place
Disadvantages	Overhead of combining sub-problems Increase memory usage	May not be efficient for large problems
Number of moves	Transfers 8 disks in 33 moves	Transfers 8 disks in 41 moves

Sample of Output

- **Case of five disks**

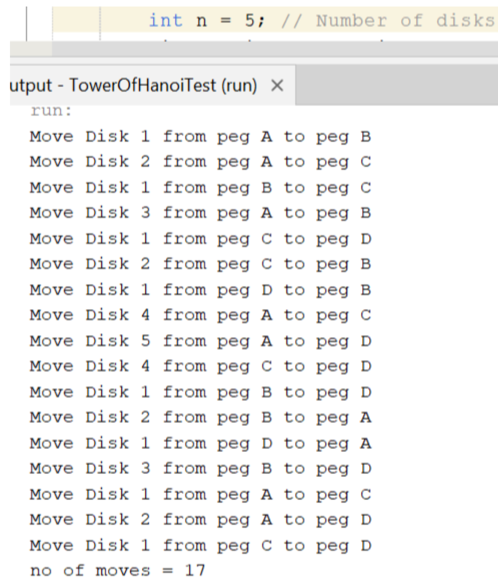
The five disks are divided into two sub-problems of size three and two disks based of the value of optimal k

The upper three disks are ordered using the four rods in the first call of function hanoi4

The lower two disks are ordered using the remaining three rods in the call of function hanoi3

The upper three disks move back again to be ordered above the bottom two disks using the four rods in the second call of function hanoi4

The number of moves is calculated based on dynamic programming that stores the result of the base case which are 0,1, and 2 disks in



Figure(): Output of 5 disks

0, 1, and 3 moves then storing the number of moves in the dp array to be used in the next number of disks

-**Case of six disks**

The six disks are divided into two sub-problems of equal size of three disks each based of the value of optimal k

The upper three disks are ordered using the four rods in the first call of function hanoi4

The lower three disks are ordered using the remaining three rods in the call of function hanoi3

The upper three disks move back again to be ordered above the bottom three disks using the four rods in the second call of function hanoi4

The number of moves is calculated based on

dynamic programming that stores the result of the base cases which are 0,1, and 2 disks in

0, 1, and 3 moves then storing the number of

Figure(): Output of six disks

moves in the dp array to be used in the next number of disks

-**Case of eight disks**

The six disks are divided into two sub-problems of equal size of four disks each based of the value of optimal k

In the first call of hanoi4

The upper four disks are ordered by recursively calling hanoi4 and divide the four disks into two sub-problems of the same size of two disks each based on the value of optimal k

The upper two disks (disk 1 and 2) ordered on peg C using the four rods in the call of function hanoi4 while the lower two disks (disk 3 and 4) are ordered on peg B using the three rods in the call of function hanoi3 then the upper two disks are ordered above the lower two disks on peg B using the four rods in the call of function hanoi4

The lower four disks are ordered by calling hanoi3 and recursively reduce the problem by one into sub-problem until it reaches the base case which is two disks (disk 5 an disk 6) then order disk 7 then order disk 8 on peg D

In the second call of hanoi4

```
Move Disk 1 from peg A to peg B
Move Disk 2 from peg A to peg C
Move Disk 1 from peg B to peg C
Move Disk 3 from peg A to peg D
Move Disk 4 from peg A to peg B
Move Disk 3 from peg D to peg B
Move Disk 1 from peg C to peg D
Move Disk 2 from peg C to peg B
Move Disk 1 from peg D to peg B
Move Disk 5 from peg A to peg C
Move Disk 6 from peg A to peg D
Move Disk 5 from peg C to peg D
Move Disk 7 from peg A to peg C
Move Disk 5 from peg D to peg A
Move Disk 6 from peg D to peg C
Move Disk 5 from peg A to peg C
Move Disk 8 from peg A to peg D
Move Disk 5 from peg C to peg D
Move Disk 6 from peg C to peg A
Move Disk 5 from peg D to peg A
Move Disk 7 from peg C to peg D
Move Disk 5 from peg A to peg C
Move Disk 6 from peg A to peg D
Move Disk 5 from peg C to peg D
Move Disk 1 from peg B to peg D
Move Disk 2 from peg B to peg A
Move Disk 1 from peg D to peg A
Move Disk 3 from peg B to peg C
Move Disk 4 from peg B to peg D
Move Disk 3 from peg C to peg D
Move Disk 1 from peg A to peg C
Move Disk 2 from peg A to peg D
Move Disk 1 from peg C to peg D
no of moves = 33
```

Figure(): Output of eight disks

The upper four disks are ordered by recursively calling hanoi4 and again divide the four disks into two sub-problems of the same size of two disks each based on the value of optimal k

The upper two disks (disk 1 and 2) ordered on peg A using the four rods in the call of function hanoi4 while the lower two disks (disk 3 and 4) are ordered on peg D using the three rods in the call of function hanoi3 then the upper two disks are ordered above the lower two disks on peg D using the four rods in the call of function hanoi4.

Conclusion

In Conclusion, using Divide and Conquer with the Dynamic Programming approach to divide the problem to two sub-problems and calculate the minimum number of moves required for different number of disks and storing them leads to optimizing the solution that offers a modified approach to solving the Tower of Hanoi Problem using four pegs, allowing more efficient solution compared to the traditional approach.

References

The Four- Peg Tower of Hanoi Puzzle by Richard Johnsonbaugh.