IE531: Algorithms for Data Analytics
Spring, 2020
Programming Assignment 1: K-Peg Tower of Hanoi
Due Date: February 7, 2020

In "*Lesson 1: Computing*" you were introduced to the *Frame-Stewart solution* for the 4 Peg version of the Tower Hanoi Problem. For this programming assignment, I want you to write a piece of Python code that solves the *K Peg Version of the Tower of Hanoi Problem* that uses recursion.

That is, you have pegs $\{1, 2, \ldots, K\}$ (cf. figure 1). Peg 1 has n-many disks stacked in such a way that that any smaller diameter disk sits above all disks that are of larger diameter in peg 1. The objective is exactly the same as before – we want to find a way of transferring the stack of disks from peg 1 to peg K such that at no point in the process a larger disk sits on top of a smaller disk on a peg.

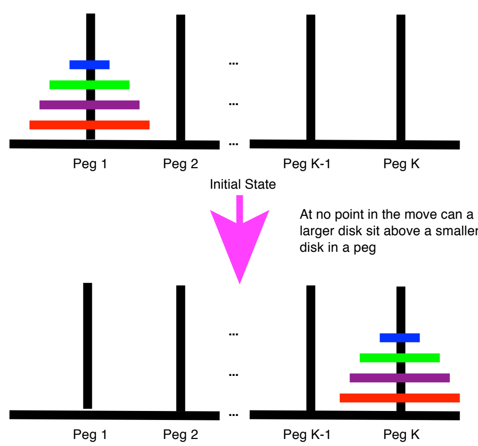

Figure 1: The *K Peg Version of the Tower of Hanoi* Problem.

This can be solved by a generalization of the Frame-Stewart Recursive Solution used in the previous programming assignment. That is, we want a recursive function $move(n, 1, K, \{2, 3, \ldots, K-1\})$, defined as

1. Let p be some number that is less than n. Move p disks from the top of peg 1 to the i-th peg, where $i \in \{2, 3, \ldots, K-1\}$ using $move(p, 1, i, \{2, 3, \ldots, K-1\})$.

2. Move the $n-p$ disks from peg 1 to peg K using $move(n-p, 1, K, \{2, 3, \ldots, K-1\} - \{i\})$. That is, this move does not involve/touch the i-th peg.

3. Move the p disks from peg i to peg K using $move(p, i, K, \{1, 2, \ldots, K-1\})$.

Here is what your program should do

1. Take as input, K, the number of pegs and the number of disks, n, of decreasing size, in peg 1.

2. Write a recursive routine that moves the n disks from peg 1 to peg K such that at no point during the moves, a larger disk sits on top of a smaller one in any peg using the generalized *Frame-Stewart Solution* described above. You have some leeway in deciding what p should be. I have found that

$$p = \begin{cases} \lfloor \frac{n}{2} \rfloor & \text{if } K - 2 > 1 \text{ (i.e. \#free pegs} > 1) \\ n - 1 & \text{otherwise.} \end{cases}$$

   work quite well.

3. Your code should output (1) each individual move, and (2) the total number of moves in your solution.

4. Your implementation should have a function that checks if each move generated by your implementation is legal/valid. That is, for any value of K and n, you should check if no larger disk is placed on top a smaller disk.

You could start with generalizing the 3-peg and 4-Peg solution that I wrote (and can be found on Compass). I am looking for an output that looks like what is shown in figures 2 and 3. The number of disks and the the number of pegs will be provided by the user on the command-line. The first command-line argument is the number of disks, and the second command-line argument is the number of pegs. You will have to use argparse for this.

```
Recursion — -zsh ▸ python — 78×46
(base) sreenivas@MacBook-Air-2 Recursion % ./K-Peg\ Tower\ of\ Hanoi.py 10 6
*************************************************************
6-Tower of Hanoi Problem, with 10-many disks on leftmost peg
*************************************************************
-----------------------------
State of Peg 1 (Top to Bottom): deque([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
Number of Steps = 0
-----------------------------
Move disk 1 from Peg 1 to Peg 5 (Legal)
Move disk 2 from Peg 1 to Peg 6 (Legal)
Move disk 1 from Peg 5 to Peg 6 (Legal)
Move disk 3 from Peg 1 to Peg 4 (Legal)
Move disk 4 from Peg 1 to Peg 3 (Legal)
Move disk 5 from Peg 1 to Peg 5 (Legal)
Move disk 4 from Peg 3 to Peg 5 (Legal)
Move disk 3 from Peg 4 to Peg 5 (Legal)
Move disk 1 from Peg 6 to Peg 1 (Legal)
Move disk 2 from Peg 6 to Peg 5 (Legal)
Move disk 1 from Peg 1 to Peg 5 (Legal)
Move disk 6 from Peg 1 to Peg 6 (Legal)
Move disk 7 from Peg 1 to Peg 4 (Legal)
Move disk 6 from Peg 6 to Peg 4 (Legal)
Move disk 8 from Peg 1 to Peg 3 (Legal)
Move disk 9 from Peg 1 to Peg 2 (Legal)
Move disk 10 from Peg 1 to Peg 6 (Legal)
Move disk 9 from Peg 2 to Peg 6 (Legal)
Move disk 8 from Peg 3 to Peg 6 (Legal)
Move disk 6 from Peg 4 to Peg 1 (Legal)
Move disk 7 from Peg 4 to Peg 6 (Legal)
Move disk 6 from Peg 1 to Peg 6 (Legal)
Move disk 1 from Peg 5 to Peg 6 (Legal)
Move disk 2 from Peg 5 to Peg 1 (Legal)
Move disk 1 from Peg 6 to Peg 1 (Legal)
Move disk 3 from Peg 5 to Peg 4 (Legal)
Move disk 4 from Peg 5 to Peg 3 (Legal)
Move disk 5 from Peg 5 to Peg 6 (Legal)
Move disk 4 from Peg 3 to Peg 6 (Legal)
Move disk 3 from Peg 4 to Peg 6 (Legal)
Move disk 1 from Peg 1 to Peg 5 (Legal)
Move disk 2 from Peg 1 to Peg 6 (Legal)
Move disk 1 from Peg 5 to Peg 6 (Legal)
-----------------------------
State of Peg 6 (Top to Bottom): deque([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
Number of Steps = 33
-----------------------------
(base) sreenivas@MacBook-Air-2 Recursion %
```

Figure 2: Sample Output – where the number of disks is 10, and number of pegs is 6 (provided as command-line arguments).

```
Recursion — -zsh ▸ python — 78×40
(base) sreenivas@MacBook-Air-2 Recursion % ./K-Peg\ Tower\ of\ Hanoi.py 8 5
**************************************************************
5-Tower of Hanoi Problem, with 8-many disks on leftmost peg
**************************************************************
-----------------------------
State of Peg 1 (Top to Bottom): deque([1, 2, 3, 4, 5, 6, 7, 8])
Number of Steps = 0
-----------------------------
Move disk 1 from Peg 1 to Peg 4 (Legal)
Move disk 2 from Peg 1 to Peg 5 (Legal)
Move disk 1 from Peg 4 to Peg 5 (Legal)
Move disk 3 from Peg 1 to Peg 3 (Legal)
Move disk 4 from Peg 1 to Peg 4 (Legal)
Move disk 3 from Peg 3 to Peg 4 (Legal)
Move disk 1 from Peg 5 to Peg 1 (Legal)
Move disk 2 from Peg 5 to Peg 4 (Legal)
Move disk 1 from Peg 1 to Peg 4 (Legal)
Move disk 5 from Peg 1 to Peg 5 (Legal)
Move disk 6 from Peg 1 to Peg 3 (Legal)
Move disk 5 from Peg 5 to Peg 3 (Legal)
Move disk 7 from Peg 1 to Peg 2 (Legal)
Move disk 8 from Peg 1 to Peg 5 (Legal)
Move disk 7 from Peg 2 to Peg 5 (Legal)
Move disk 5 from Peg 3 to Peg 1 (Legal)
Move disk 6 from Peg 3 to Peg 5 (Legal)
Move disk 5 from Peg 1 to Peg 5 (Legal)
Move disk 1 from Peg 4 to Peg 5 (Legal)
Move disk 2 from Peg 4 to Peg 1 (Legal)
Move disk 1 from Peg 5 to Peg 1 (Legal)
Move disk 3 from Peg 4 to Peg 3 (Legal)
Move disk 4 from Peg 4 to Peg 5 (Legal)
Move disk 3 from Peg 3 to Peg 5 (Legal)
Move disk 1 from Peg 1 to Peg 4 (Legal)
Move disk 2 from Peg 1 to Peg 5 (Legal)
Move disk 1 from Peg 4 to Peg 5 (Legal)
-----------------------------
State of Peg 5 (Top to Bottom): deque([1, 2, 3, 4, 5, 6, 7, 8])
Number of Steps = 27
-----------------------------
(base) sreenivas@MacBook-Air-2 Recursion %
```

Figure 3: Sample Output – where the number of disks is 8, and number of pegs is 5 (provided as command-line arguments).