Putting it all together

Problem ID: a07p13puttingitalltogether

Alright, now for the hard part; solving the actual puzzle. To do that, we are going to create one last function named `move_many`. The function recieves four parameters, how many discs there are, from what pillar we are moving from, to which pillar we are moving to, and a game state. The function then returns the updated state as it is after having moved all the discs, as well as printing the game state after every move.

Despite its name, the function can only move one disc at a time. Moreover, it should print out the state after each move it makes. To achieve this, we are going to use a trick called recursion. What that means is that this function is going to call itself! It is a bit like that Inception movie.

Recursion

The general solution method is as follows. When asked to move m discs from pillar x to pillar y, what we do is

- Let z be the number of the remaining pillar
- Call `move_many` recursively to move $m - 1$ discs from x to z
- Move the $n - th$ disc from x to y
- Call `move_many` recursively to move $m - 1$ discs from z to y

We have to be careful not to recurse too far. When n equals 0, then that is our terminating case and our function just returns the state unchanged.

Getting an intuition for why this solution method actually works is not trivial. As is often the case, WikiPedia has a pretty good explanation.

Our program, when successfully implemented, will be able to solve the Tower of Hanoi puzzle for up to 9 discs. It prompts the user for the number of discs and then shows the state after every move that is made until all the discs are on the right-most pillar.

Note that we are testing your code differently in this task, please only submit your function definitions, without any code outside the functions! The main python file, which handles input and output, is provided again this time, and includes code corresponding to the input as described below, as well as a bit of code you can use for the default configuration of the puzzle, where you only need to supply the number of disks. The latter is also given in the starter code, nested inside an if statement that ensures it is not run when importing functions from your file, so you can also try out your code without the main file if you prefer.

Input

This time, we need to distinguish between the input to the test case and the input to the function, because although the function will be called with the input from the test case as arguments, it will also be called by itself with other arguments.

The function recieves four parameters. an integer m, how many discs to move between two pillars, an integer f, the pillar we are moving from, an integer t, the pillar we are moving to, and a sequence s representing the state of the game.

In the test cases, the input to the main program (see the samples below), consists of four lines: The first line contains an integer n, the total number of disks there are in the puzzle. The second line contains an integer k_b, the pillar where the stack of disks begins. The third line contains an integer k_e, the pillar where the stack of disks should end up. And the fourth line contains a string s_0 representing the initial state of the game.

In the tests, n will be restricted to $1 \leq n \leq 9$, k_b will be restricted to $1 \leq k_b \leq 3$, k_e will be restricted to $1 \leq k_e \leq 3$ with $k_e \neq k_b$, and s_0 will be a string with $4 \leq |s_0| \leq 12$.

The function will then be called with the arguments $m = n$, $f = k_b$, $t = k_e$ and $s = s_0$, but as mentioned, during its execution, it will call itself with other values for the parameters m, f, t and s.

It is good if your function also works for other types of sequences and elements, or for input outside these specifications, but that is not part of the requirements.

Output

Note, this time, in addition to returning a sequence s', the updated game state after moving m discs from pillar f to pillar t, the function should also print the updated game state after each move directly to the output.

As the function repeatedly calls itself, this will result in a sequence of lines in the output, one for each move. The total number of moves required to solve the puzzle is $2^n - 1$. (Can you see why?)

So the output should consist of $2^n - 1$ lines, where the i-th line includes the game state after the i-th move, ending once the puzzle is fully solved.

Sample Input 1

```
3
1
3
321|||
```

Sample Output 1

```
32||1|
3|2|1|
3|21||
|21|3|
1|2|3|
1||32|
||321|
```

Sample Input 2

```
4
1
3
4321|||
```

Sample Output 2

```
432|1||
43|1|2|
43||21|
4|3|21|
41|3|2|
41|32||
4|321||
|321|4|
|32|41|
2|3|41|
21|3|4|
21||43|
2|1|43|
|1|432|
||4321|
```

Sample Input 3	Sample Output 3
5 1 3 54321|||	5432||1| 543|2|1| 543|21|| 54|21|3| 541|2|3| 541||32| 54||321| 5|4|321| 5|41|32| 52|41|3| 521|4|3| 521|43|| 52|43|1| 5|432|1| 5|4321|| |4321|5| 1|432|5| 1|43|52| |43|521| 3|4|521| 3|41|52| 32|41|5| 321|4|5| 321||54| 32||541| 3|2|541| 3|21|54| |21|543| 1|2|543| 1||5432| ||54321|