

Jordan Pemberton

CS325 Algorithms

Extra Credit: Nth Step Problem

Nth Step Problem

I approached this problem by first thinking about it as a permutation with repetition problem.

First, I wrote out the possible permutations for $n=0$ through $n=7$, being able to take either 1 or 2 steps at once.

Using $n=7$:

With 7 stairs ($n=7$), and being able to take either 1 or 2 steps at once, there are 21 permutations. I realized that these 21 options can be divided into 4 groups:

- There is 1 option that is 7 single steps.
- There are 6 options that involve taking 5 single steps and 1 double step.
- There are 10 options that involve taking 3 single steps and 2 double steps.
- And there are 4 options that involve taking 1 single step and 3 double steps.

The number of options in each of these groups follows the permutation with repetition formula

$$m! / a!b!$$

where m is the number of spaces being filled, and a and b are how many times items are repeated.

In this problem, m is the total number of steps taken, and a and b are the counts of single and double steps.

The counts for the 4 groups (if $n=7$) are:

$$\begin{aligned} 7! &/ 7!0! \\ 6! &/ 5!1! \\ 5! &/ 3!2! \\ 4! &/ 1!3! \end{aligned}$$

Using this formula, I determined that the count of each of the groups is equal to:

$$(n-k)! / ((n-2k)! * k!)$$

where $k = 0$ through $n // 2$.

The total number of permutations, then, should be equal to:

```
sum((n-k)! / ((n-2k)! * k!)), k=0 to n//2
```

This summation is equal to:

```
(F[n] + L[n]) / 2
```

where $F[n]$ is the n th Fibonacci number, and $L[n]$ is the n th Lucas number.

Knowing this, and thinking about the problem more intuitively, I realized I could code this problem very similarly to how a Fibonacci generator would be written. Like the Fibonacci problem, recursion can be used, but it is faster to use memoization (dynamic programming). Since the result is a sum of all the preceding results, you can iterate through n and save prior results with memoization.

To Run:

These files can be run with Python 3 from the command line.

Running `test_nth_step.py` will test both `nth_step.py` (for the 1-2 step problem and the 1-2-3 step version of this problem) and `nth_step_math.py` (for the 1-2 step problem), and your output will be written to the text file `nth_step_output.txt`.

`nth_step.py` uses the DP method discussed, while `nth_step_math.py` uses the factorial permutation formula discussed.