Roman numeral passwords

Andrew Ford

June 28, 2025

1 Introduction

This week's problem statement can be found in Zach Wissner-Gross's weekly Fiddler on the Proof column at https://thefiddler.substack.com/p/can-you-crack-the-roman-code

2 Original problem

In the original problem statement, you have a password given to you in Roman numerals. It does not contain spaces between the characters, meaning that the password appears as the string "IIIIIIIIII" (ten occurrences of the uppercase letter I, with no spaces to disambiguate it). The keypad has numerical inputs "I", "II", and "III", and we need to figure out how many valid passwords this string represents.

Let $P(n)$ represent the number of valid passwords which a string containing n copies of the letter I could represent. If $n > 3$, the string consists of a first number (either I, II, or III) followed by the remainder of the string. This means the number of valid combinations, for $n > 3$, follows the recursive relationship

$$P(n) = P(n-1) + P(n-2) + P(n-3). \tag{1}$$

To generate $P(n)$ for any value of n, we need three base cases. Computing by hand, we have

n	P(n)	Combinations
1	1	I
2	2	II, I I
3	4	III, II I, I II, I I I

From these base cases, it is straightforward to iteratively compute $P(10) = 274$. This computation is also outlined in the Python file PasswordOnlyI.py.

3 Extra credit problem

In the extra credit problem, the problem is extended to the Roman numerals IV, V, VI, VII, and VIII, and the password string now contains both I's and V's. The specific password string, with spaces removed again, outlined in the extra credit problem is the 15 character string "IIIVIIIVIIIVIII".

For this problem, I wrote more general Python code to take in any string of I's and V's, found in PasswordFull.py. I tracked the number of possible passwords for each progressively longer substring terminating at the end of the password string (for example, the last five characters). I adopted the same general strategy of checking all numerals which could start each substring; since letters in the string vary, there is no simple formula like equation (1). To make the rest of the code function properly, I made the assumption that a string of length zero has one possible combination, the empty combination.

After running this code, I discovered that there are exactly 4000 possible passwords which could be represented by the spaceless string "IIIVIIIVIIIVIII". Additionally, this same code correctly computes the answer to the original problem with the string "IIIIIIIIII."