Samuel Pegg 2020280261

$Open\ Project$

For this open project I decided to explore question 4 of the 2018 exam paper: how many ways to compose a binary strings (made up of 0s and 1s) of length n that do not contain "010" or "101"?

To answer this question, its pretty simple to code a naïve answer in python. I simply created an algorithm that created every possible string and counted it. This ran well up to around $n = 30$, where it became so slow it was no longer useable. This is my "count" function in python.

Next, I tried working with recurrence relations. I wanted to program something that would return the number of strings without using the direct formula.

My first attempt was a simple recursive algorithm: $a_1 = 2$, $a_2 = 4$, $a_n = a_{n-1} + a_{n-2}$. This performed well up to about $n = 38$ – barely an improvement. This was my "rec" function in python.

At this point, taking hints from previous OJ tasks, I decided to return the number modulo $10^9 + 7$.

Because a lot of work was being repeated, I decided to store the previous values in a list. This is my "recwithstorage" function in python. This performed well up to $n \approx 900$, where the maximum recursion depth of python stopped it from running. I tried increasing the maximum recursion depth but after $n = 2996$ it just returned a null value. This was a little confusing as it was returning the value for $n = 2996$ in around 0.004 seconds, but I assume there was something breaking deep in python's code that I don't have the knowledge to fix. I tweaked the algorithm so that it only stored the a_{k-2} values, since the a_{k-1} value will then iterate to the a_{k-2} value in the next step, and this returned results up to $n \approx 5980$ – around double what we got before. I could keep reducing the number of values stored, but this didn't seem like the way forward. I moved on to the next algorithm.

Since the recurrence relation is essentially a number of additions, I then tried simply adding up all the values recursively using the code $a_1 = 2$, $a_2 = 4$,

$$a_k = a_{k-1} + a_{k-2}$$

$$a_{k-2} = a_{k-1}$$

$$a_{k-1} = a_k$$

This code can be seen in my "recop" function in python. This algorithm was a huge improvement. This took me up to around $n = 100000000$ in 12 seconds. However, I wanted to try reaching arbitrarily large values of n modulo $10^9 + 7$.

Notice that the two equations

$$a_n = a_{n-1} + a_{n-2}$$

$$a_{n-1} = a_{n-1}$$

can be written using matrices as

$$\begin{pmatrix} a_{n+1} \\ a_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a_n \\ a_{n-1} \end{pmatrix}$$

This means that we can calculate a_n by

$$\begin{pmatrix} a_n \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-2} \begin{pmatrix} a_2 \\ a_1 \end{pmatrix}$$

However, diagonalisation and NumPy were both dead ends for large n. NumPy was even slower than the previous attempt and diagonalisation didn't work due to the surds in the golden ratio. I decided to write my own matrix exponentiation and matrix multiplication methods, using lists. This meant I could take the modulus after each addition and subtraction, to reduce storage. In addition, I used the fact that

$$x^n = x^{n//2} x^{n\%2}$$

to recursively define my power function. This got me the result I wanted. For example, the number of binary strings containing that do not contain "010" or "101" of length n =23948234872390847293857092387287364528357923875982376589263847562736423875283765987658726354872634587623485762834756283475628374568234765928347687627635422200000324237492386876876187187 18 is

$$458166835 \; mod \; 10^9 + 7$$

This result returned in 0.0015025138854980469 seconds.

Of course, this application of matrices can be extended to any recurrence relation. Suppose we have the recurrence relation $a_1 = 1$, $a_2 = 3$ and $a_3 = 7$, and

$$a_n = 3a_{n-1} + 6a_{n-2} + 7a_{n-3}$$

Then clearly

$$\begin{pmatrix} a_n \\ a_{n-1} \\ a_{n-2} \end{pmatrix} = \begin{pmatrix} 3 & 6 & 7 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_{n-1} \\ a_{n-2} \\ a_{n-3} \end{pmatrix}$$

And so

$$\begin{pmatrix} a_n \\ a_{n-1} \\ a_{n-2} \end{pmatrix} = \begin{pmatrix} 3 & 6 & 7 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}^{n-3} \begin{pmatrix} a_3 \\ a_2 \\ a_1 \end{pmatrix}$$

For this sequence, the n =23485720948571986418723938145092384690283457092834795827349857234875682374523948592834672938475293875098237459872349584587230498572304895723948523948572934857239845876124763487123874 term is

$$266476152 \; mod \; 10^9 + 7$$

And this result returned in 0.021646976470947266 seconds.