Layup Sequence - Report

Cody Roth — 11rothcody@gmail.com

1 Introduction

The *Layup sequence* we are given is defined as:

$$S(n) = \begin{cases} 1 & n = 1, \\ 2 & n = 2, \\ S(n-1) + S(n-2) & n \text{ even}, \\ 2S(n-1) - S(n-2) & n \text{ odd}. \end{cases} \tag{1}$$

Right away, it is apparent that code which computes (1) using pure recursion (see `attempt.py`) will take far too long to compute $S(10000)$. The code in `attempt.py` already takes nearly 30 seconds to compute $S(40)$.

2 Main Solution

The problem with `attempt.py` is that every iteration in computing $S(n)$ invokes two more function calls, so the time complexity is $O(2^n)$.

We can get rid of all these unnecessary function calls by caching our intermediate results. See `main.py`. In particular, by caching the values $S(k)$ for each $1 \leqslant k \leqslant n$, we ensure that we only compute $S(k)$ exactly once for each such k. The key is that the code first tries looking up the value for $S(k)$ in the cache before trying to do any computations. If $S(k)$ is not in the cache, it then computes it according to (1) and saves the result to the cache. Note that the base cases are explicitly cached to ensure the cache is non-empty.

Because looking up a value in the cache (which in my case is a Python dictionary with keys k and values $S(k)$) is $O(1)$ time, having the function first try checking the cache for values means we do at most n computations when computing $S(n)$. In other words, `main.py` has time complexity $O(n)$, which is very fast.

The code explicitly computes $S(10000)$: see `S_10000.txt` (the number was too large to fit neatly into this report). The runtime I recorded was about **0.13 seconds**.

3 Optimizations

One clear area for optimization is to remove the parity dependence in (1). In other words, we hope to find a new recursive sequence $L(n)$ which produces the terms of $S(n)$ without the piecewise definition based on if n is even or odd.

Suppose $n = 2k$. Then, by definition:

$$\begin{aligned} S(2k) &= S(2k-1) + S(2k-2), \\ &= 2S(2k-2) - S(2k-3) + S(2k-3) + S(2k-4), \\ &= 2S(2k-2) + S(2k-4). \end{aligned}$$

Instead, if $n = 2k + 1$, we have:

$$\begin{aligned} S(2k+1) &= 2S(2k) - S(2k-1), \\ &= 2[S(2k-1) + S(2k-2)] - [2S(2k-2) - S(2k-3)], \\ &= 2S(2k-1) + S(2k-3). \end{aligned}$$

In both cases, we have established that:

$$S(n) = 2S(n-2) + S(n-4). \tag{2}$$

This actually proves the following:

Theorem: Define a recursive sequence $L(n)$ by:

$$L(n) = \begin{cases} 0 & n = 0, \\ 1 & n = 1, 2, 3, \\ 2L(n-2) + L(n-4) & n \geqslant 4. \end{cases} \tag{3}$$

Then, $S(n) = L(n+2)$ for all $n \geqslant 1$, where $S(n)$ is the Layup sequence given in (1). □

The point of defining and relating $S(n)$ and $L(n)$ as done in the theorem above is because we need four base cases to correctly utilize (2). This will also make the code easier to look at (in my opinion).

The code which implements this optimized recursion is featured in `optimized.py`. We use a caching mechanism as before in the main solution, and I defined the function $L(n)$ in the code exactly as in (3). Note the code also explicitly defines $S(n) = L(n+2)$, which is valid based on the theorem. The code again computes $S(10000)$ (which I verified to be the same answer I got before), but this time the runtime is just over **0.03 seconds**. Based on my testing, this new method is **roughly 4 times faster than the main solution**.

3.1 Analytic Solution

(This section probably goes beyond what the assignment asked for but I thought the math ended up being interesting. Feel free to skip it.)

If we can come up with a closed form equation for the Layup sequence, we can effectively compute $S(10000)$ in $O(1)$ time. Indeed, the recurrence in (3) is homogeneous and linear, with characteristic polynomial:

$$p(x) = x^4 - 2x^2 - 1. \tag{4}$$

The roots of p are:

$$\Big\{ \pm\sqrt{\sqrt{2}+1},\ \ \pm i\sqrt{\sqrt{2}-1} \Big\}, \tag{5}$$

which allows us to write:

$$L(n) = c_1\Big(\sqrt{\sqrt{2}+1}\Big)^n + c_2(-1)^n\Big(\sqrt{\sqrt{2}+1}\Big)^n + c_3 i^n\Big(\sqrt{\sqrt{2}-1}\Big)^n + c_4(-i)^n\Big(\sqrt{\sqrt{2}-1}\Big)^n, \tag{6}$$

for constants c_1, c_2, c_3, c_4. Using the base cases $L(0) = 0$, $L(1) = L(2) = L(3) = 1$ allows us to solve for these constants:

$$c_1 = \frac{\sqrt{2} + 2\sqrt{\sqrt{2}-1}}{8}, \qquad c_2 = \frac{\sqrt{2} - 2\sqrt{\sqrt{2}-1}}{8},$$
$$c_3 = \frac{-\sqrt{2} - 2i\sqrt{\sqrt{2}+1}}{8}, \qquad c_4 = \frac{-\sqrt{2} + 2i\sqrt{\sqrt{2}+1}}{8}.$$

Substituting these values back into (6) yields the closed form equation for $L(n)$, and hence $S(n)$ by our theorem. In conclusion:

$$S(n) = \frac{1}{8}\Bigg[\Big(\sqrt{2}+2\sqrt{\sqrt{2}-1}\Big)\Big(\sqrt{\sqrt{2}+1}\Big)^{n+2} + \Big(\sqrt{2}-2\sqrt{\sqrt{2}-1}\Big)(-1)^{n+2}\Big(\sqrt{\sqrt{2}+1}\Big)^{n+2}$$
$$+\Big(-\sqrt{2}-2i\sqrt{\sqrt{2}+1}\Big)i^{n+2}\Big(\sqrt{\sqrt{2}-1}\Big)^{n+2} + \Big(-\sqrt{2}+2i\sqrt{\sqrt{2}+1}\Big)(-i)^{n+2}\Big(\sqrt{\sqrt{2}-1}\Big)^{n+2}\Bigg].$$

This allows us to compute all terms of the Layup sequence without any recurrence.