Efficiently Computing Recurrence Relation Functions

David Brewster

March 23, 2017

1 Abstract

Suppose we have a function $f : \mathbb{F} \to \mathbb{F}$ that is defined in terms of itself (i.e. a recurrence relation). How can we compute $f(n)$ efficiently? We will explore common (previously known) techniques of efficiently answering that question. (Note: I assume the reader is familiar with elementary linear algebra and sequences)

2 Problem Statement

Suppose we have a sequence $a_0, a_1, a_2, ...$ defined as follows:

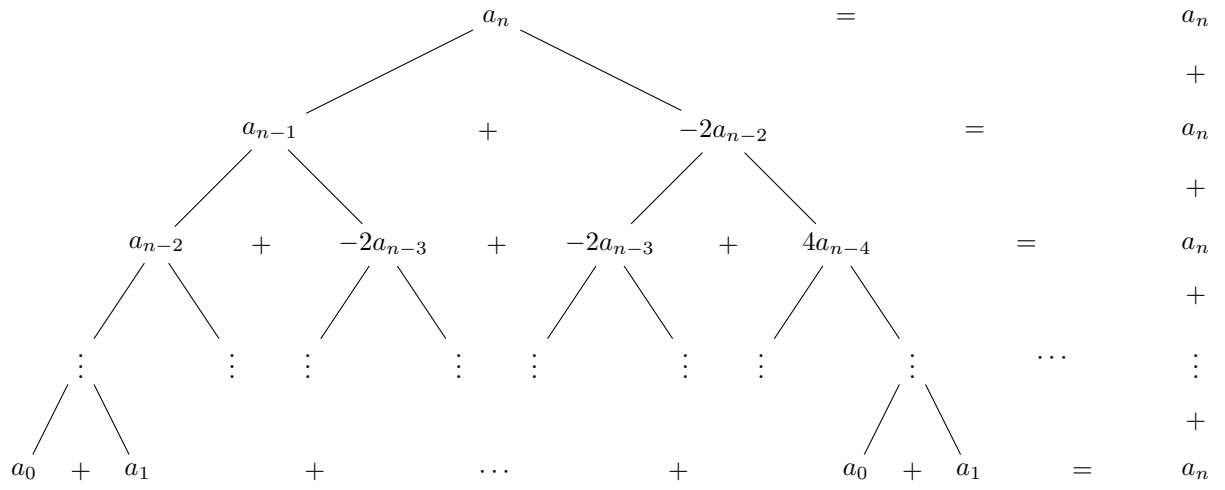
$$a_0 = 0$$
$$a_1 = 1$$
$$a_n = a_{n-1} - 2a_{n-1}$$

So, computing the first couple of values of $a_0, a_1, a_2, ...$, we have:
$0, 1, 1, -1, -3, -1, 5, 7, -3, -17, ...$

Computing this with no optimizations (i.e. naïvely), we have an exponential time complexity of $O(2^n)$. Obviously, this is unacceptable. Let us examine different techniques to minimize the time complexity of solving the above problem.

3 Technique 1

There is one rather simple optimization we can make. Notice that with our naïve method of computation, we use a top-down approach. Let us define a tree such that the parent node is equal to the sum of its immediate children. With that definition, let us draw out the logistics of the top-down approach using the tree definition defined above (this is a rough sketch; in reality, the tree would not be a perfect tree as shown below):



Since this tree is on the order of a perfect tree, we can find how many nodes a perfect k-ary tree has by the following formula:

$$N(k,h) = \frac{k^{h+1}-1}{k-1}$$

So, since $k = 2$ (we have a **bi**nary tree) and $h = n$ (the height of the tree is n), then we have the number of nodes in the tree to be $N(2,n) = \frac{2^{n+1}-1}{2-1} = 2^{n+1} - 1$, which is on the order of 2^n (Also, at each node, we are only doing a constant amount of work).

Instead of doing this top-down approach, what if we utilized a bottom-up approach as follows (python code):

```
def a(n):
    a_n = 0
    a_n_1 = 1

    # loop n times
    for _ in range(n):
        # preserve previous values
        last_a_n = a_n
        last_a_n_1 = a_n_1

        # generate new values
        new_a_n = last_a_n_1
        new_a_n_1 = last_a_n_1 - 2 * last_a_n_2

        # update sequence values
        a_n = new_a_n
        a_n_1 = new_a_n_1
    return a_n
```

The above procedure has a time complexity of $\Theta(n)$. But, can we do any better?

4 Technique 2

From the problem statement, we have the formula $a_n = a_{n-1} - 2a_{n-2}$ where $a_0 = 0$ and $a_1 = 1$. Now consider the system of equations:

$$\begin{aligned} a_n - 2a_{n-1} &= a_{n+1} \\ a_n &= a_n \end{aligned}$$

This is then equivalent to:

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

But, how do we know what a_n and a_{n-1} are? Well, we can set up the same system of equations:

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

But, how do we know what a_{n-1} and a_{n-2} are? Well, we can set up the same system of equations:

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

But, how do we know... I think you get the point. Our question will finally terminate when we ask, "But, how do we know what a_1 and a_0 are?" Well, $a_0 = 0$ and $a_1 = 1$, as stated in the problem statement. Since we have answered the base case question, we can go one level up in our recursion and answer that question, so on and so forth until we answer our original question. This means, with back-substitution:

$$\left(\begin{pmatrix} 1 & -2 \\ 1 & 0 \end{pmatrix} \left(\begin{pmatrix} 1 & -2 \\ 1 & 0 \end{pmatrix} \left(\begin{pmatrix} 1 & -2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} a_{n-2} \\ a_{n-3} \end{pmatrix}\right)\right)\right) = \begin{pmatrix} a_{n+1} \\ a_n \end{pmatrix}$$

$\Longrightarrow$

$$\left(\begin{pmatrix} 1 & -2 \\ 1 & 0 \end{pmatrix} \left(\begin{pmatrix} 1 & -2 \\ 1 & 0 \end{pmatrix} \left(\begin{pmatrix} 1 & -2 \\ 1 & 0 \end{pmatrix} \cdots\right)\right)\right) \begin{pmatrix} a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} a_{n+1} \\ a_n \end{pmatrix}$$

$\Longrightarrow$

$$\begin{pmatrix} 1 & -2 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} a_{n+1} \\ a_n \end{pmatrix}$$

"Okay,", you might ask, "but how is this more efficient?". Well, remember, we can compute the exponentiation of a number a to the power n in $O(log_2(n))$ time as follows (pythonic psuedocode):

```
def qe(a, n):
    if n == 0: return 1
    if n is odd: return a * qe(a, n-1)
    else:
        half = qe(a, n/2)
        return half * half
```

Well, we can do the same with matrices:

```
def qe(A, n):
    if n == 0: return I
    if n is odd: return A * qe(A, n-1)
    else:
        half = qe(A, n/2)
        return half * half
```

More generally, if $A = \begin{pmatrix} 1 & -2 \\ 1 & 0 \end{pmatrix}$, $x_0 = \begin{pmatrix} a_1 \\ a_0 \end{pmatrix}$, and $b = \begin{pmatrix} a_{n+1} \\ a_n \end{pmatrix}$, then we can conclude $A^n x_0 = b$ and solve that equation in $O(log_2(n))$ time using quick exponentiation.

5 Technique 2.1

Again, suppose we want to solve $A^n x_0 = b$. We can then use the property $A^n = PD^nP^{-1}$ for a diagonal matrix D and some matrix P. Hence, we need to diagonalize A.

First, we find the eigenvalues of A. So we solve:

$$\begin{aligned}
& det(\lambda I - A) = 0 \\
\implies & \begin{vmatrix} \lambda - 1 & -2 \\ 1 & \lambda - 0 \end{vmatrix} = 0 \\
\implies & \begin{vmatrix} \lambda - 1 & -2 \\ 1 & \lambda \end{vmatrix} = 0 \\
\implies & \lambda(\lambda - 1) - (-2) = 0 \\
\implies & \lambda(\lambda - 1) + 2 = 0 \\
\implies & \lambda^2 - \lambda + 2 = 0 \\
\implies & \lambda = \frac{1 \pm \sqrt{(-1)^2 - 4 \cdot (1) \cdot (2)}}{2 \cdot (1)} \\
\implies & \lambda = \frac{1 \pm \sqrt{-7}}{2} \\
\implies & \lambda = \frac{1 \pm i\sqrt{7}}{2}
\end{aligned}$$

Next, we find the eigenvectors associated with each eigenvector:

When $\lambda = \frac{1 + i\sqrt{7}}{2}$, the eigenvector associated with λ is $\begin{pmatrix} \lambda - 1 & -2 \\ 1 & \lambda \end{pmatrix} \mathbf{x}_1 = 0 \implies \mathbf{x}_1 = \begin{pmatrix} \frac{1 + i\sqrt{7}}{2} \\ 1 \end{pmatrix}$

When $\lambda = \frac{1 - i\sqrt{7}}{2}$, the eigenvector associated with λ is $\begin{pmatrix} \lambda - 1 & -2 \\ 1 & \lambda \end{pmatrix} \mathbf{x}_2 = 0 \implies \mathbf{x}_2 = \begin{pmatrix} \frac{1 - i\sqrt{7}}{2} \\ 1 \end{pmatrix}$

From the previous steps, we can conclude $D = \begin{pmatrix} \frac{1+i\sqrt{7}}{2} & 0 \\ 0 & \frac{1-i\sqrt{7}}{2} \end{pmatrix}$, $P = \begin{pmatrix} \mathbf{x}_1 \mid \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} \frac{1+i\sqrt{7}}{2} & \frac{1-i\sqrt{7}}{2} \\ 1 & 1 \end{pmatrix}$,

which implies $P^{-1} = \begin{pmatrix} \frac{-i}{\sqrt{7}} & \frac{i+\sqrt{7}}{2\sqrt{7}} \\ \frac{i}{\sqrt{7}} & \frac{i-\sqrt{7}}{2\sqrt{7}} \end{pmatrix}$

So, if $A = PDP^{-1} = \begin{pmatrix} \frac{1+i\sqrt{7}}{2} & \frac{1-i\sqrt{7}}{2} \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \frac{1+i\sqrt{7}}{2} & 0 \\ 0 & \frac{1-i\sqrt{7}}{2} \end{pmatrix} \begin{pmatrix} \frac{-i}{\sqrt{7}} & \frac{i+\sqrt{7}}{2\sqrt{7}} \\ \frac{i}{\sqrt{7}} & \frac{i-\sqrt{7}}{2\sqrt{7}} \end{pmatrix}$,

then

$$A^n = PD^nP^{-1} = \begin{pmatrix} \frac{1+i\sqrt{7}}{2} & \frac{1-i\sqrt{7}}{2} \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \frac{1+i\sqrt{7}}{2} & 0 \\ 0 & \frac{1-i\sqrt{7}}{2} \end{pmatrix}^n \begin{pmatrix} \frac{-i}{\sqrt{7}} & \frac{i+\sqrt{7}}{2\sqrt{7}} \\ \frac{i}{\sqrt{7}} & \frac{i-\sqrt{7}}{2\sqrt{7}} \end{pmatrix} = \begin{pmatrix} \frac{1+i\sqrt{7}}{2} & \frac{1-i\sqrt{7}}{2} \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \left(\frac{1+i\sqrt{7}}{2}\right)^n & 0 \\ 0 & \left(\frac{1-i\sqrt{7}}{2}\right)^n \end{pmatrix} \begin{pmatrix} \frac{-i}{\sqrt{7}} & \frac{i+\sqrt{7}}{2\sqrt{7}} \\ \frac{i}{\sqrt{7}} & \frac{i-\sqrt{7}}{2\sqrt{7}} \end{pmatrix}$$

So, if we want to solve $A^n x_0 = b$, we just solve: $PD^nP^{-1}x_0 = b$

$$\implies \begin{pmatrix} \frac{1+i\sqrt{7}}{2} & \frac{1-i\sqrt{7}}{2} \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \left(\frac{1+i\sqrt{7}}{2}\right)^n & 0 \\ 0 & \left(\frac{1-i\sqrt{7}}{2}\right)^n \end{pmatrix} \begin{pmatrix} \frac{-i}{\sqrt{7}} & \frac{i+\sqrt{7}}{2\sqrt{7}} \\ \frac{i}{\sqrt{7}} & \frac{i-\sqrt{7}}{2\sqrt{7}} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} a_{n+1} \\ a_n \end{pmatrix}$$

$$\implies \begin{pmatrix} \frac{\left(\frac{1+i\sqrt{7}}{2}\right)^{n+1} - \left(\frac{1-i\sqrt{7}}{2}\right)^{n+1}}{i\sqrt{7}} \\ \frac{\left(\frac{1+i\sqrt{7}}{2}\right)^{n} - \left(\frac{1-i\sqrt{7}}{2}\right)^{n}}{i\sqrt{7}} \end{pmatrix} = \begin{pmatrix} a_{n+1} \\ a_n \end{pmatrix}$$

$$\implies a_n = \frac{\left(\frac{1+i\sqrt{7}}{2}\right)^{n} - \left(\frac{1-i\sqrt{7}}{2}\right)^{n}}{i\sqrt{7}}$$

Is this a constant time formula? Well, sadly no because we still have to exponentiate. So, the time complexity is still $O(log_2(n))$.

6 Exercise

A binary string is a string consisting of entirely 0s and 1s. For instance, 1010 is a binary string but 1020 is not.
Given an integer n, find the number of binary strings of length n such that there are no adjacent 1s in the string.

For example, 1000101 is one such binary string that satisfies the property defined above at length 7, but 1001101 is not.

(Note: in this problem, binary strings can begin with a zero, so 001 is a valid binary string of length 3)
(Hint #1: What are the base cases, i.e. what is a_0 and a_1? What is the recurrence relation?)
(Hint #2: At any given place in the binary string, what are the possible options?/how many possible options are there?)