Algorithms and Data Structures for Beginners

9 / 35

9 - Fibonacci Sequence

Mark Lesson Complete View Code Prev Next

Suggested Problems

Status	Star	Problem	Difficulty	Video Solution	Code
☐	☆	Fibonacci Number	Easy		
☐	☆	Climbing Stairs	Easy	▶	C++

Recursion (Two Branch)

A more interesting case of recursion is the two-branch recursion. Again, let's take a mathematical example to try and explain this - the Fibonacci sequence. Generally, the formula to calculate the nth fibonacci number in math is to take the $n - 1$ and $n - 2$ number and add them together. It is given that $F(0) = 1$ and $F(1) = 1$, which, in a recursive function, this would be our base case.

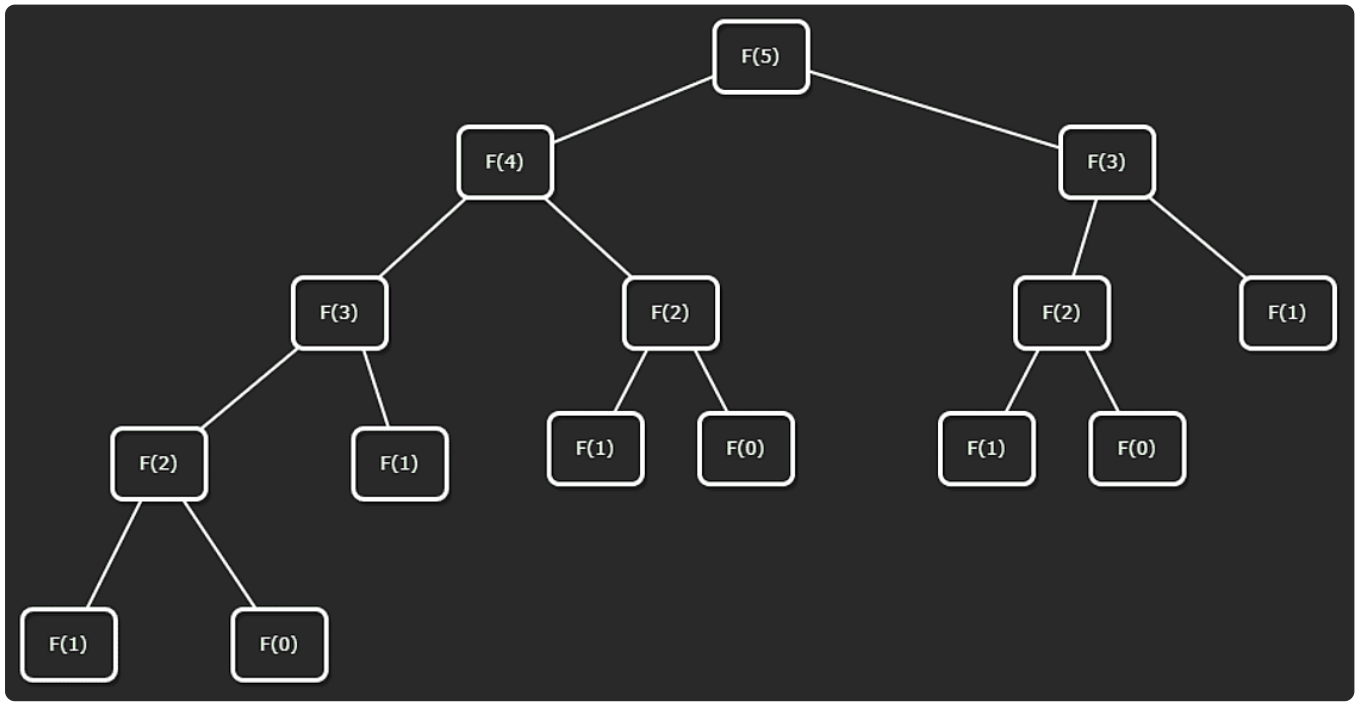
The formula would be written like this:

$$fib(n) = fib(n - 1) + fib(n - 2)$$

> *The above is known as a recurrence relation.*

Conceptualization

We can visualize the mathematical formula with the following tree.




```
fn fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n - 1) + fibonacci(n - 2)
```

The above pseudocode is similar to our previous example with factorial, except this is a branch factor of two. Notice how we are calling the function twice in the last line,

this results in the tree that is shown in the visual.

To analyze, let's follow the same technique we introduced in the previous chapter. We have our base case, we know the function calls itself in the last return statement, and we know that at some point when the base case is reached, we will have to travel back "up" to calculate the ultimate answer. To calculate `fibonacci(5)`, we get `fibonacci(4) + fibonacci(3)`. Now, both of these will execute the function from line 1. Looking at our tree, `fibonacci(4)` will call `fibonacci(3) + fibonacci(2)` and so on, until `n` hits `1` or `0` after which it will return the result, and keep going back up all the way until `fibonacci(4)` which will give us an answer of 3. Now, we have the answer to `fibonacci(4)` and do the same for `fibonacci(3)` which results in 2. Add the two together, and the $5th$ fibonacci number is 5.

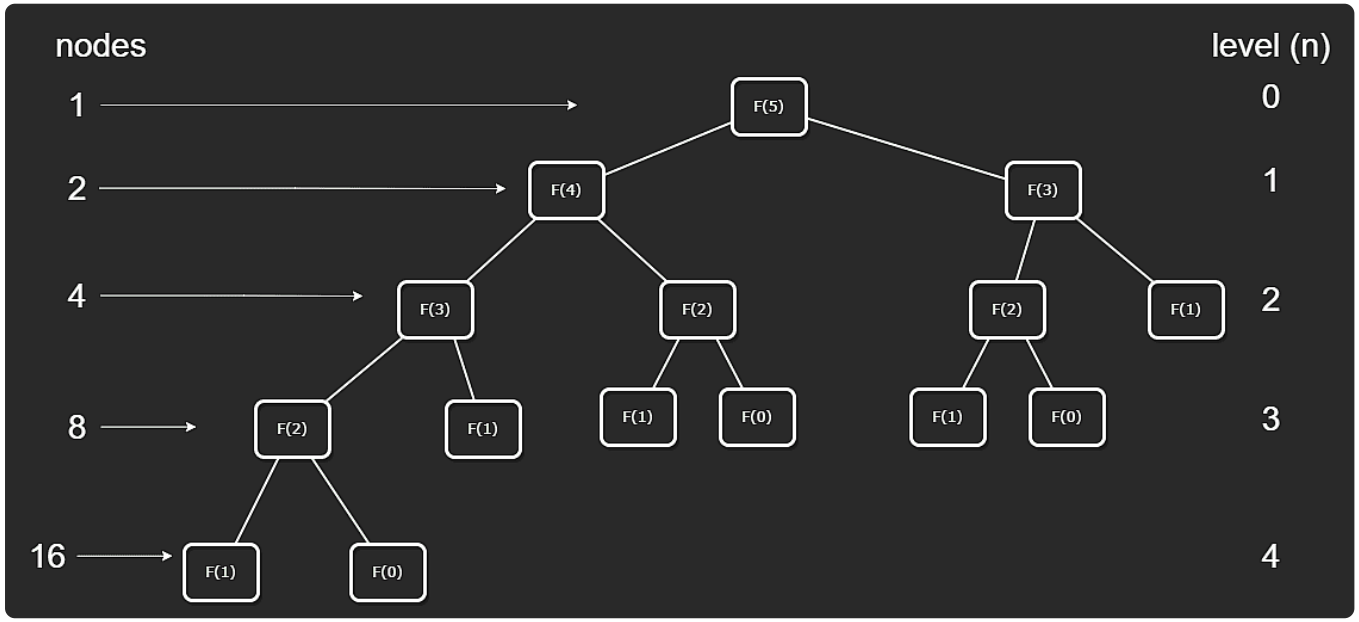
Time Complexity Evaluation

Evaulating the time complexity for this is a little bit more tricky. Let's analyze the tree, and the number of nodes on each one of the levels. On the 1st level (0 indexed), there is 1, on the 2nd level, there are 2, then 4, after which we can see a pattern. Each level has the potential to hold 2x the nodes of the previous level.

That only gives us half the answer. If n is the level we are currently on, this means that to get the number of nodes at any level n, the formula is 2^n. Since we have to potentially traverse all the way to the nth level, and each level has twice as many nodes, we can say the function is upper bounded by 2^n. Recall the power series

concept discussed in the dynamic array chapter where the last term is the dominating term. Notice how on the last level (4) there can be at most 16 nodes. Since the last level is in $O(2^n)$, it must be the case that the entire tree is in $O(2^n)$.

Algorithmically speaking, even if we did have $2 * 2^n$ or $2^n - 1$ operations, it would still belong to $O(2^n)$ because constants do not affect the bound.



Closing Notes

We will keep coming back to recursion again and again in the course, and it really shines when it comes to traversing trees and graphs.

For now, to solidify your understanding, try the problems linked below the video.