The Fibonacci Numbers

In this lesson, we will learn about a flagship application of recursion, the Fibonacci numbers.

We'll cover the following

What are the Fibonacci numbers?

Fibonacci numbers are a series of numbers given by the following set of formulas:

$$Fib(0) = 0$$

$$Fib(1) = 1$$

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

This is a classic example of recursion. The first two equations give us the **base cases**, while the third equation is the **recursive step**. Verbally, we can define *n^{th} Fibonacci number as sum of (n-1)th and (n-2)th Fibonacci numbers*.

Are Fibonacci numbers more than just a formula?

Well, yes! Look at the visualization in Figure 1.

In addition to being pretty, Fibonacci numbers find their application in very broad areas. The following are some applications of Fibonacci Numbers:

- Fibonacci numbers are found in a number of natural phenomena like branching in trees and

arrangement of leaves on a stem.

- Fibonacci numbers have been used to compose melodies.
- Fibonacci numbers are used in some pseudorandom number generators.
- Fibonacci numbers are used in creating the Fibonacci heap data structure.

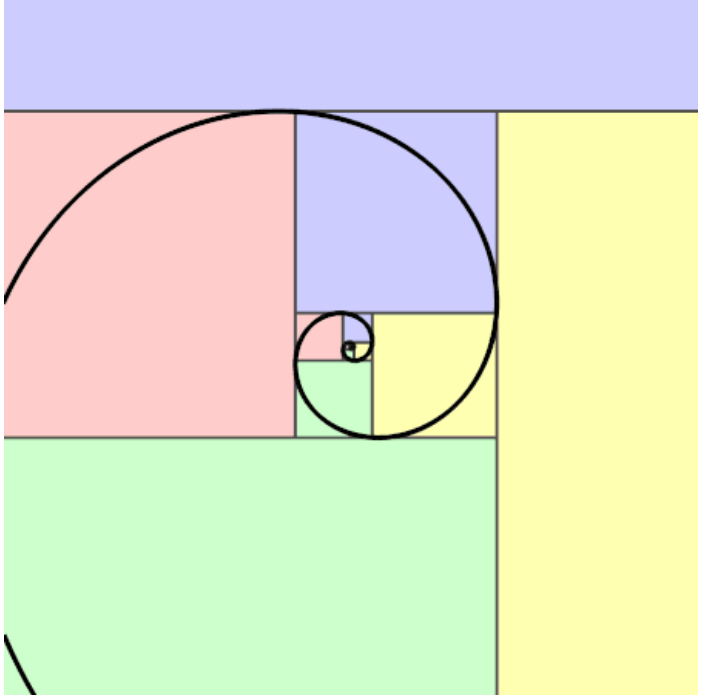


Figure 1: The length of each square is equal to a Fibonacci number.

Implementation of Fibonacci numbers

If you have grasped the formulas well enough, writing code for Fibonacci numbers is not difficult at all. You just have to write the formulas given at the start of this lesson in the form of code. First, we specify our two base cases of when a given number is equal to 0 or 1. Lastly, we will write a recursive formula for the n^{th} Fibonacci number. Let's look at the implementation of Fibonacci numbers.

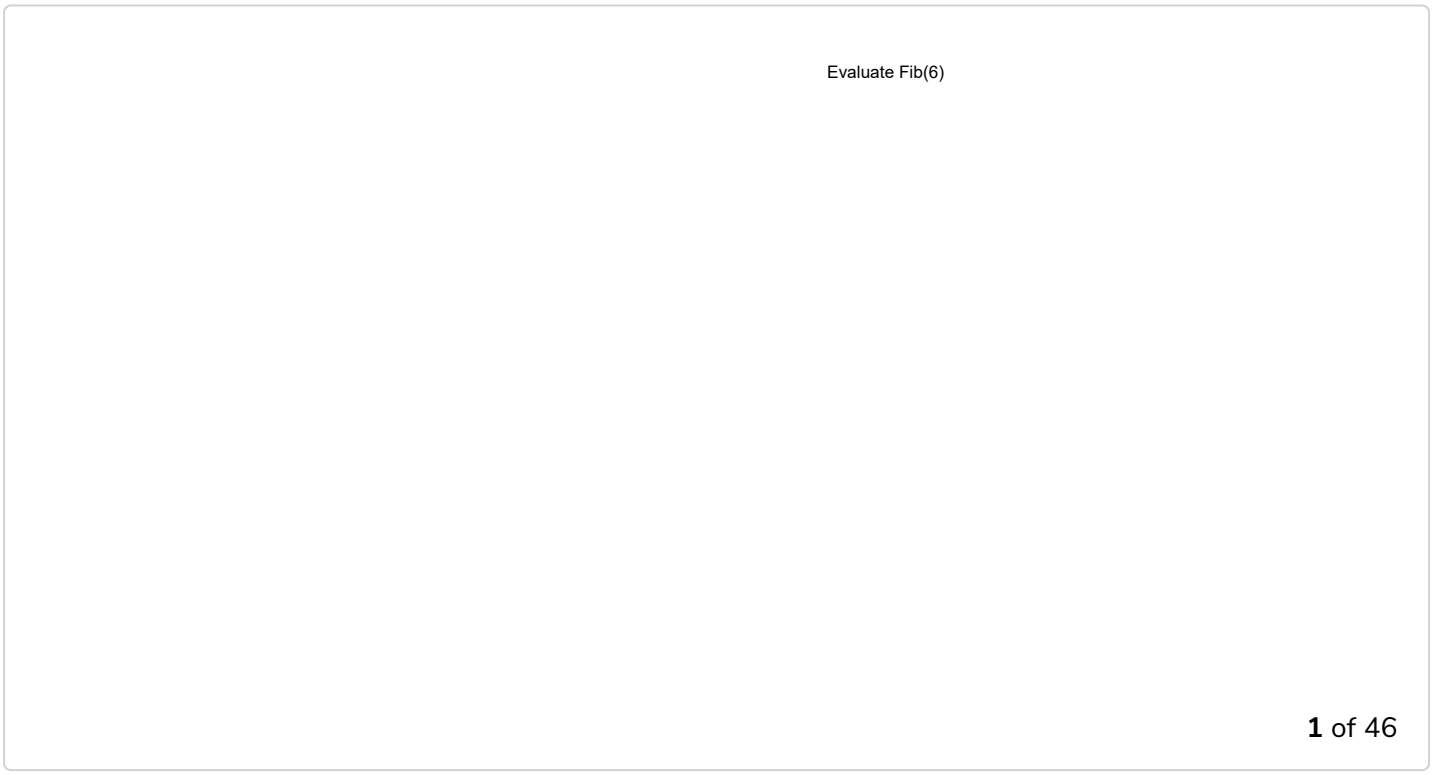
```
def fib(n):
  if n == 0: # base case 1
    return 0
  if n == 1: # base case 2
    return 1
  else: # recursive step
    return fib(n-1) + fib(n-2)

print (fib(10))
```

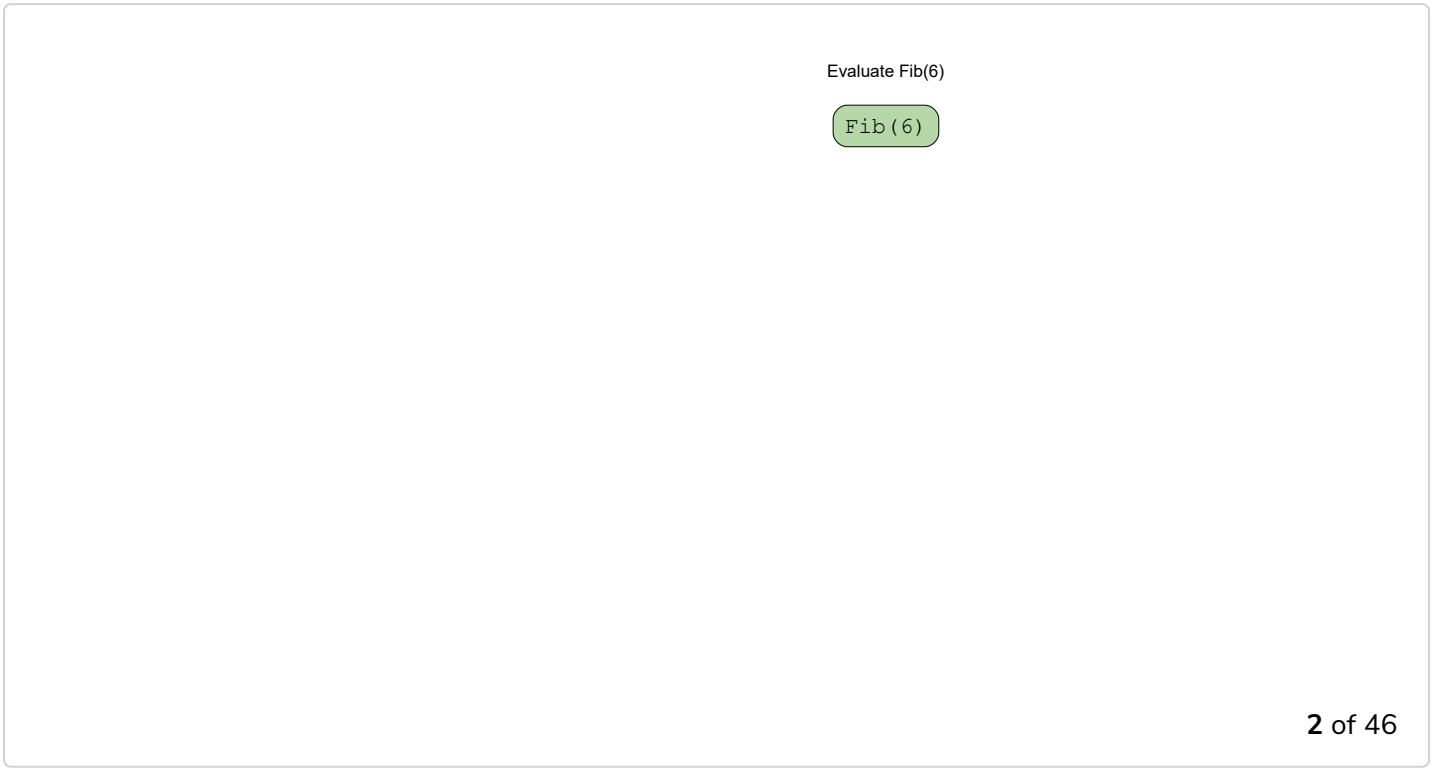
In the above implementation, *line 2* and *line 4* specify the base case, whereas *line 7* gives our recursive step. Note that the order of these instructions is very important. Placing the recursive step before the base cases will result in an endless recursion. This holds true for all kinds of recursive algorithms.

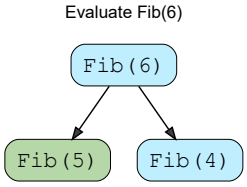
recursion. This holds true for all kinds of recursive algorithms.

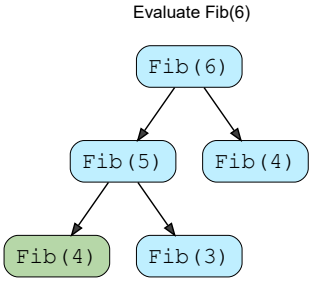
Time complexity

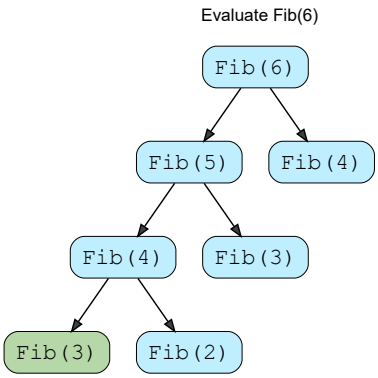
In the code above, try plugging in slightly bigger values instead of `10` in *line 9*. You will notice that as the values get bigger, our algorithm starts to slow down. A value near `50` would result in a timeout on our platform. This is because our algorithm has exponential time complexity, more specifically $\mathbf{O(2^n)}$. To understand why the time complexity is exponential, look at the following visualization.

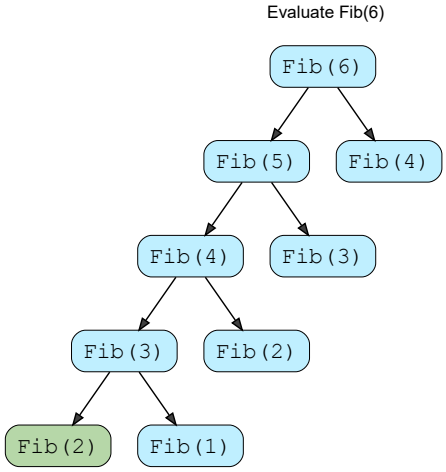


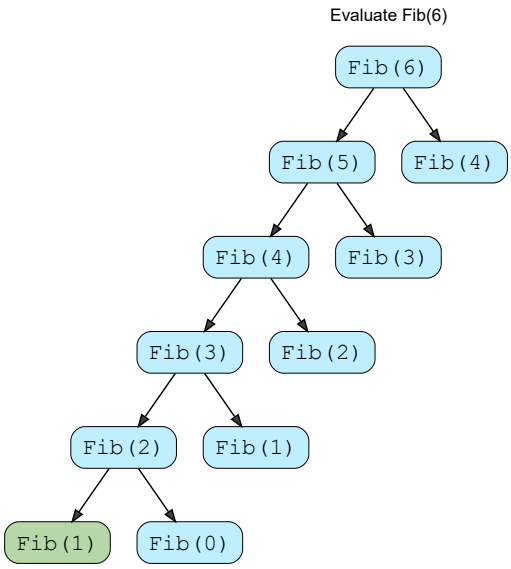


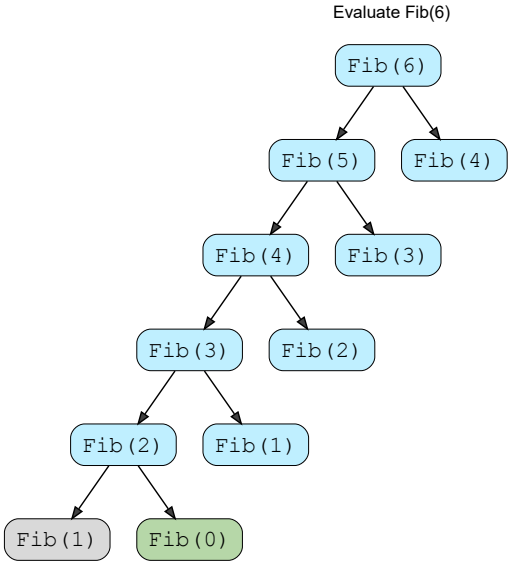

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)

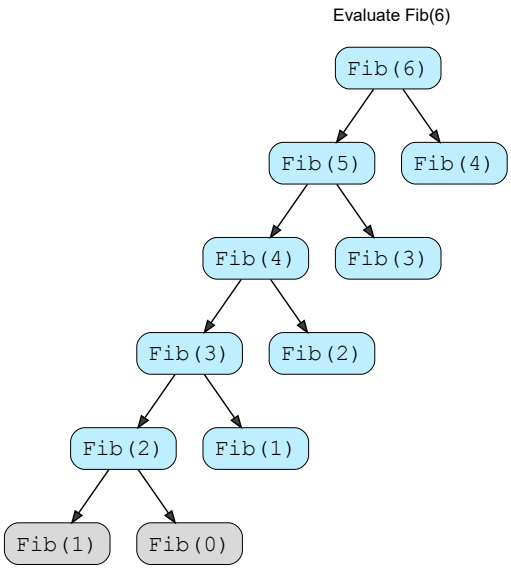

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)

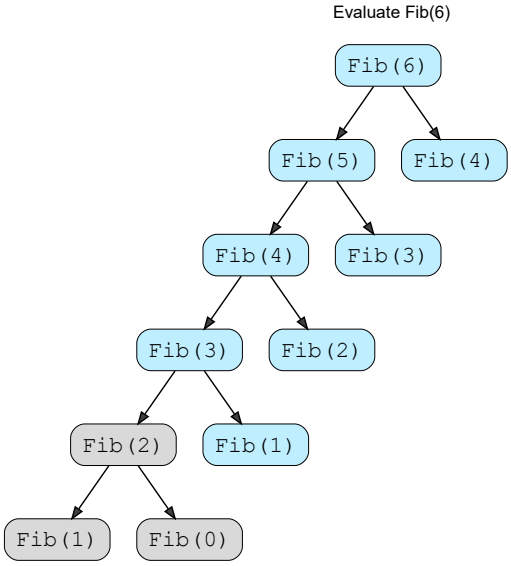

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)

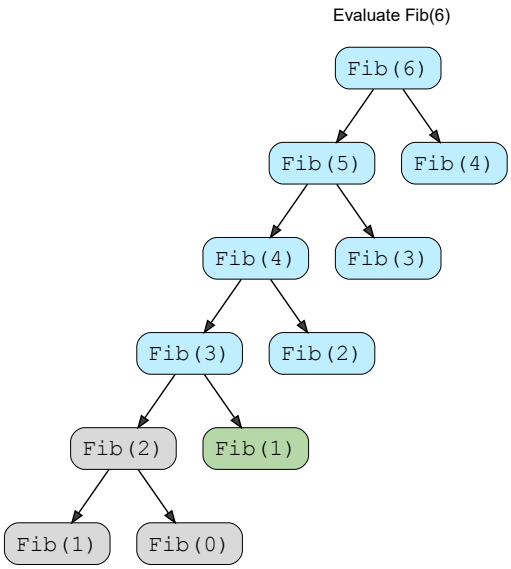

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)

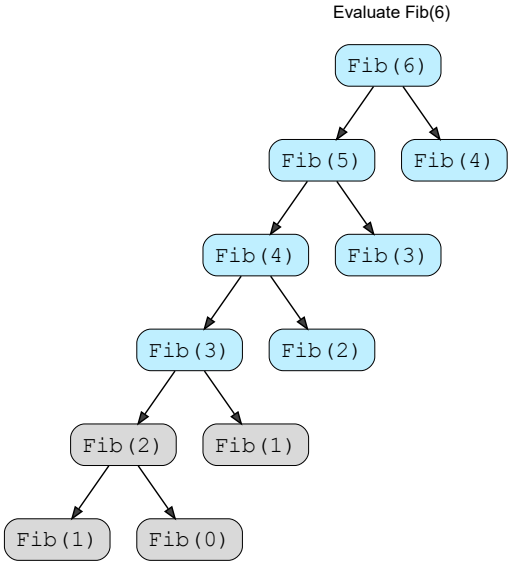

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)

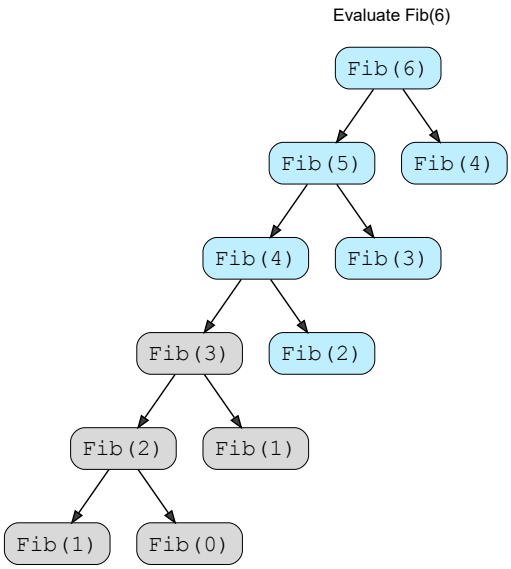

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)

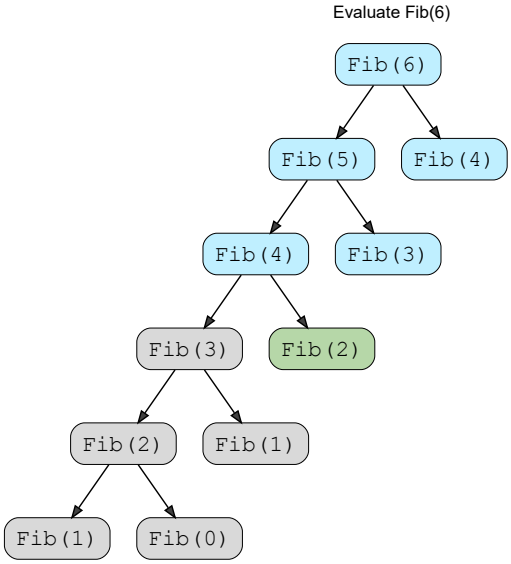

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)

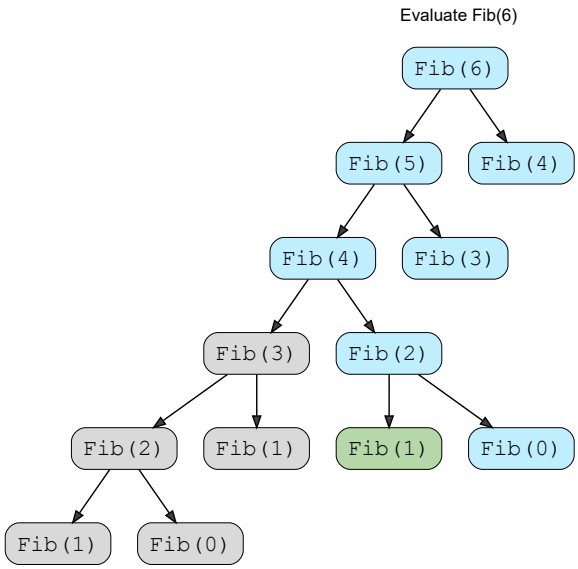

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)

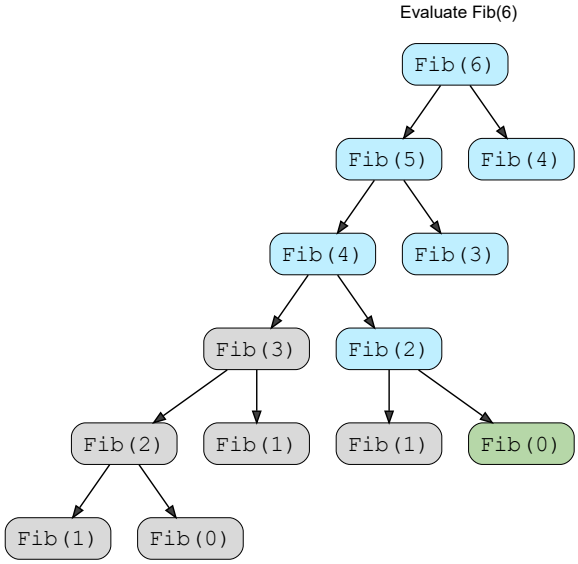

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)

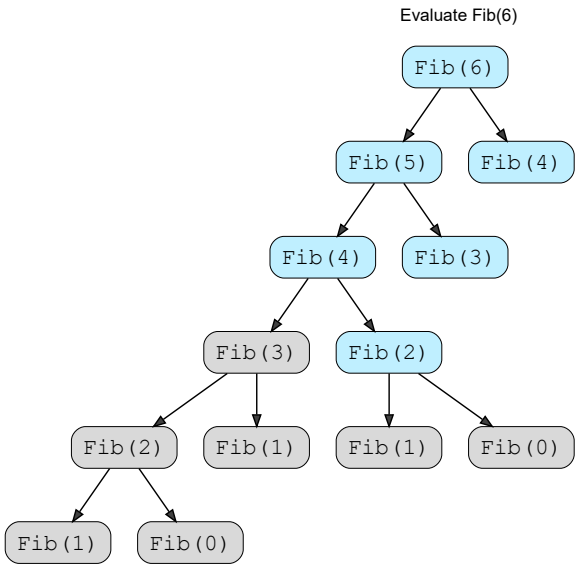

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)

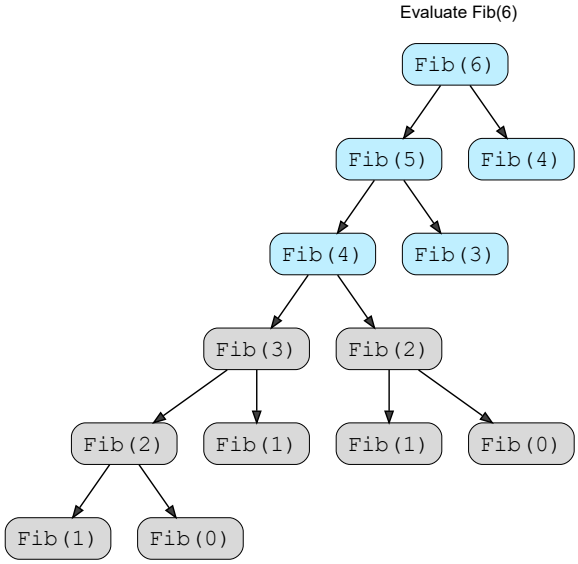

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)

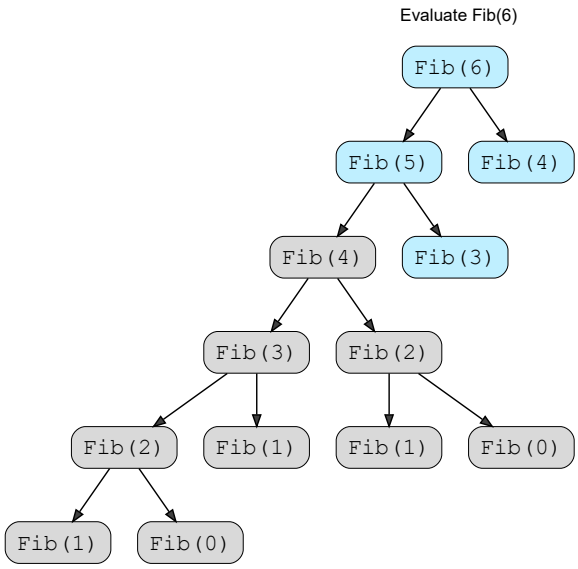

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)

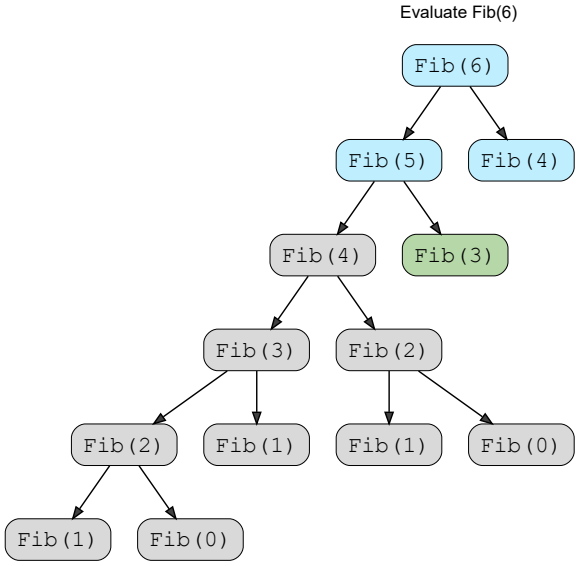

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

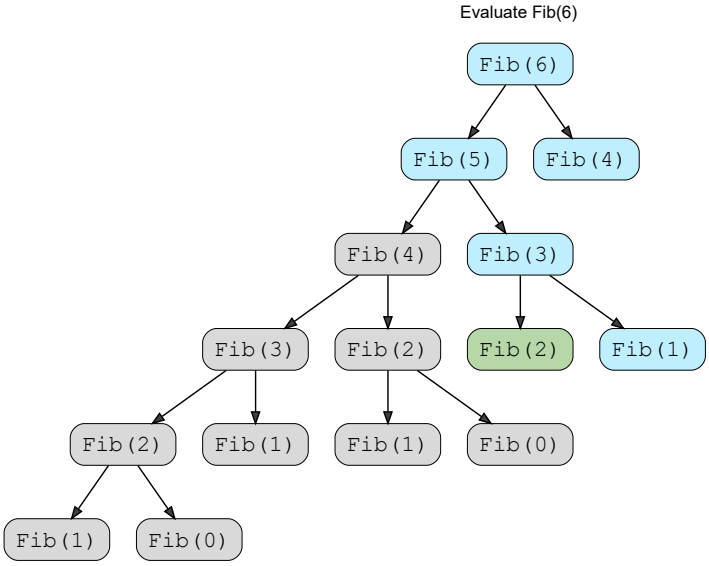

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

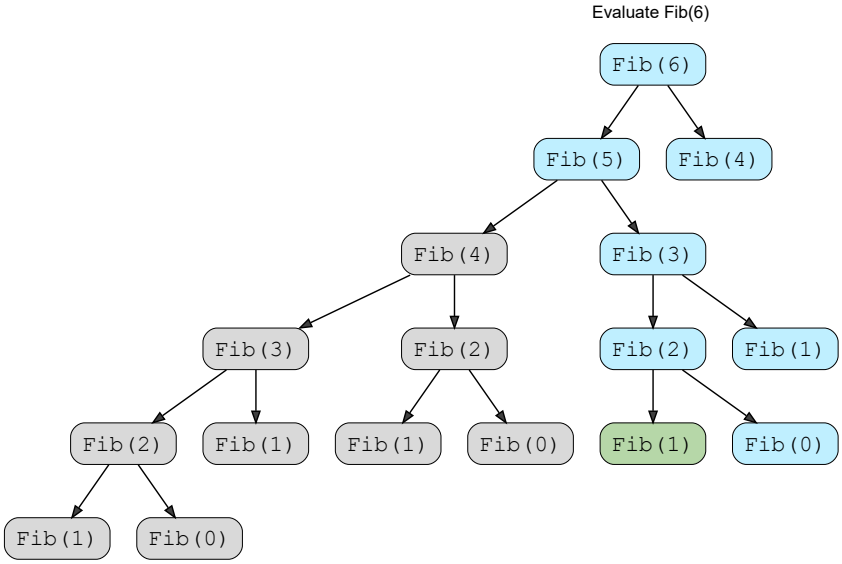

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

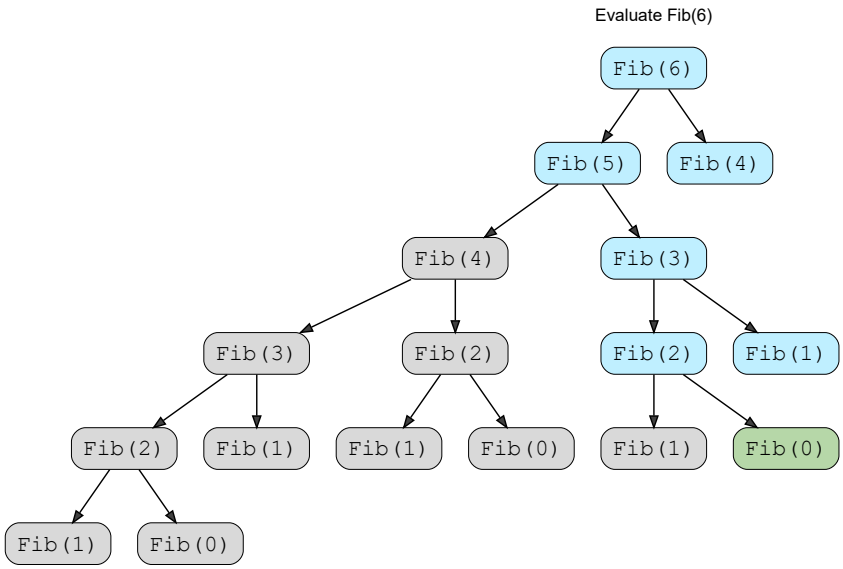

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

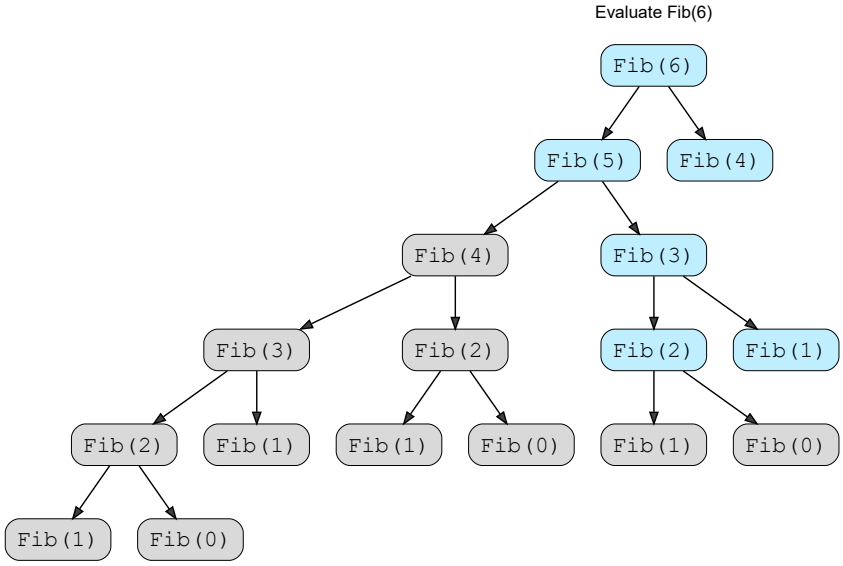

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

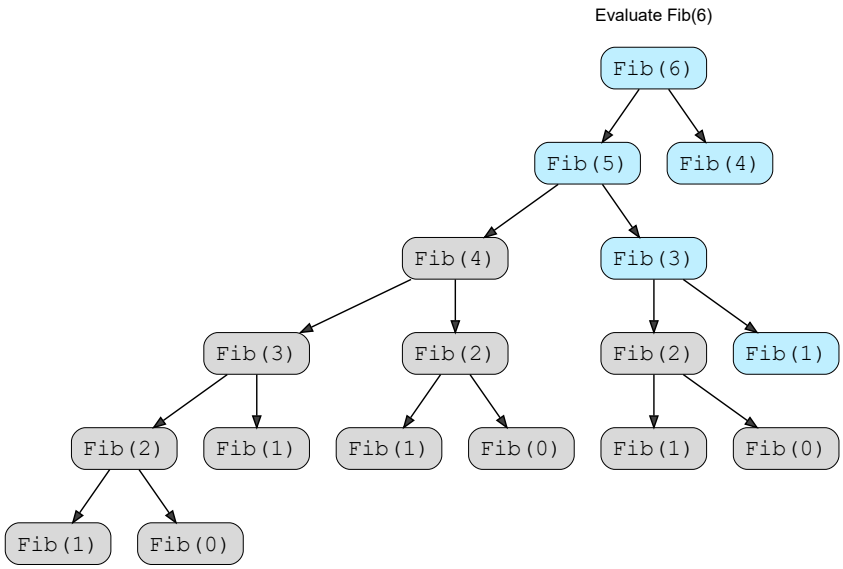

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

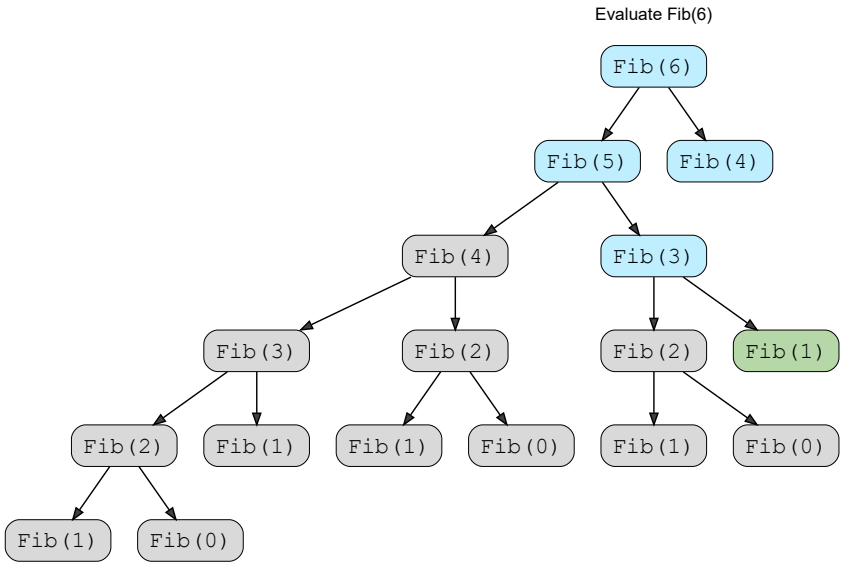

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

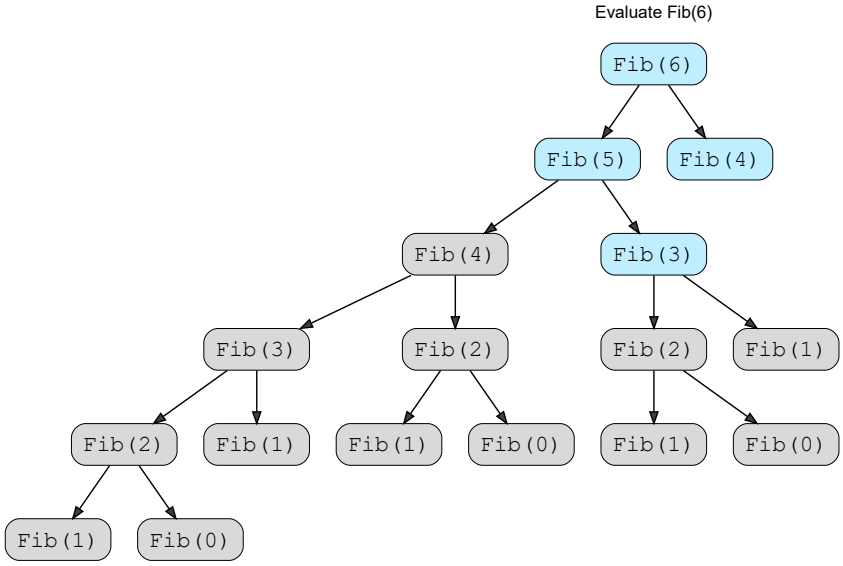

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

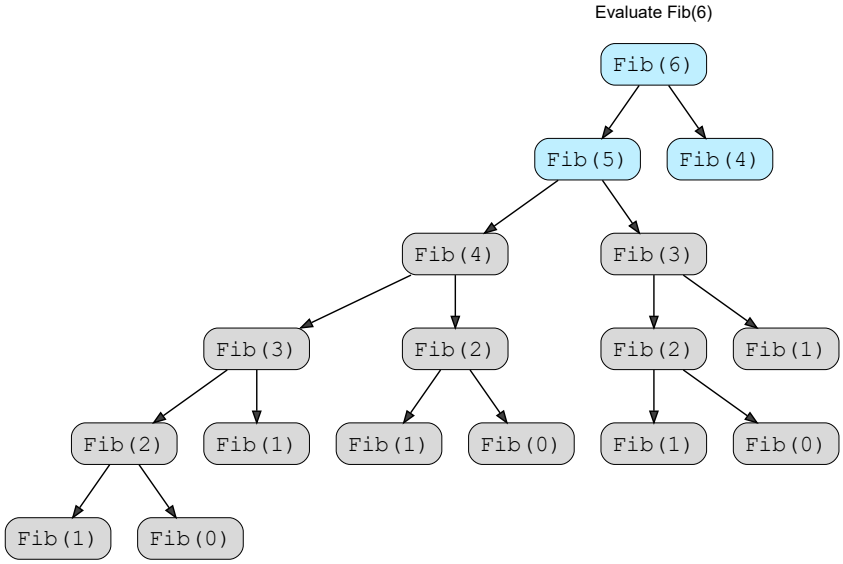

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

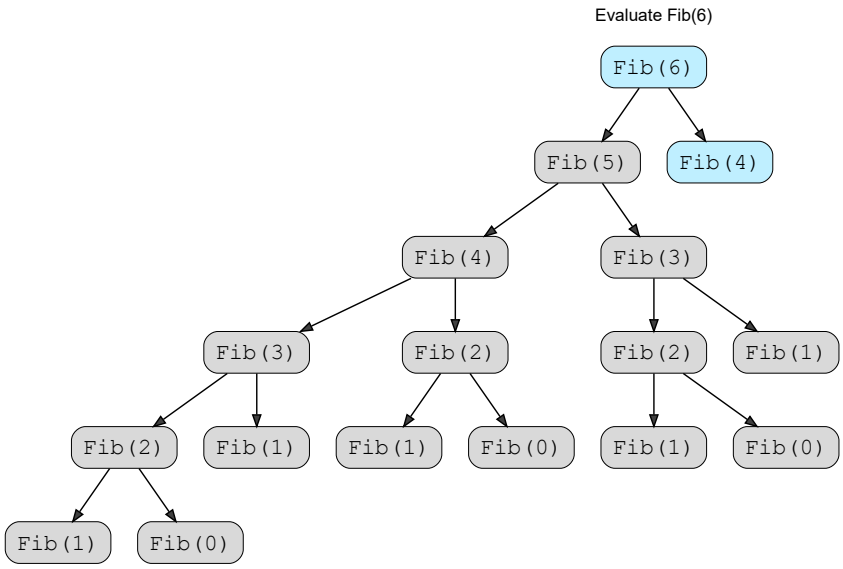

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

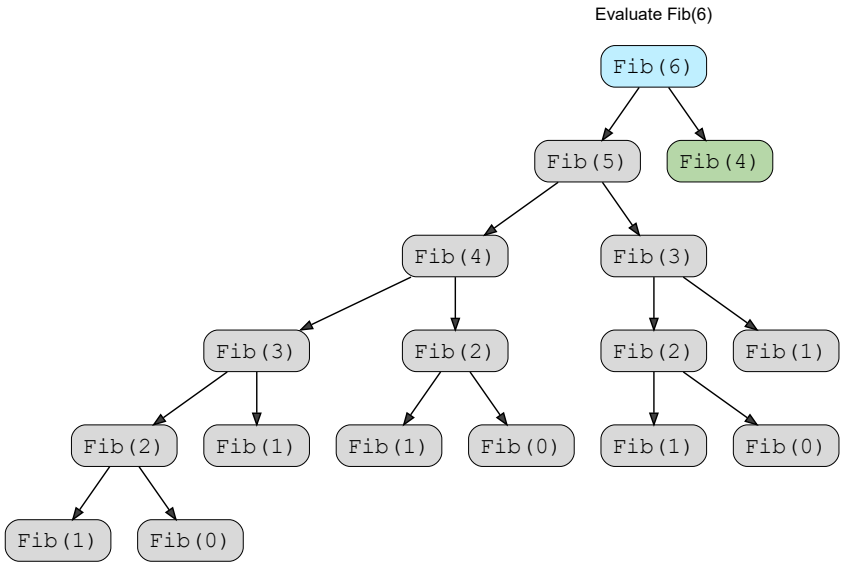

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

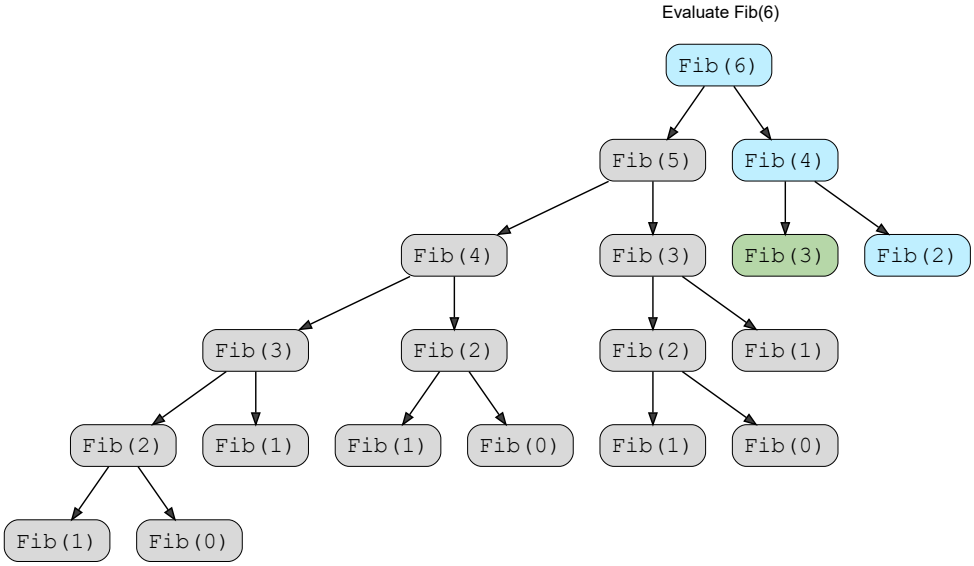

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

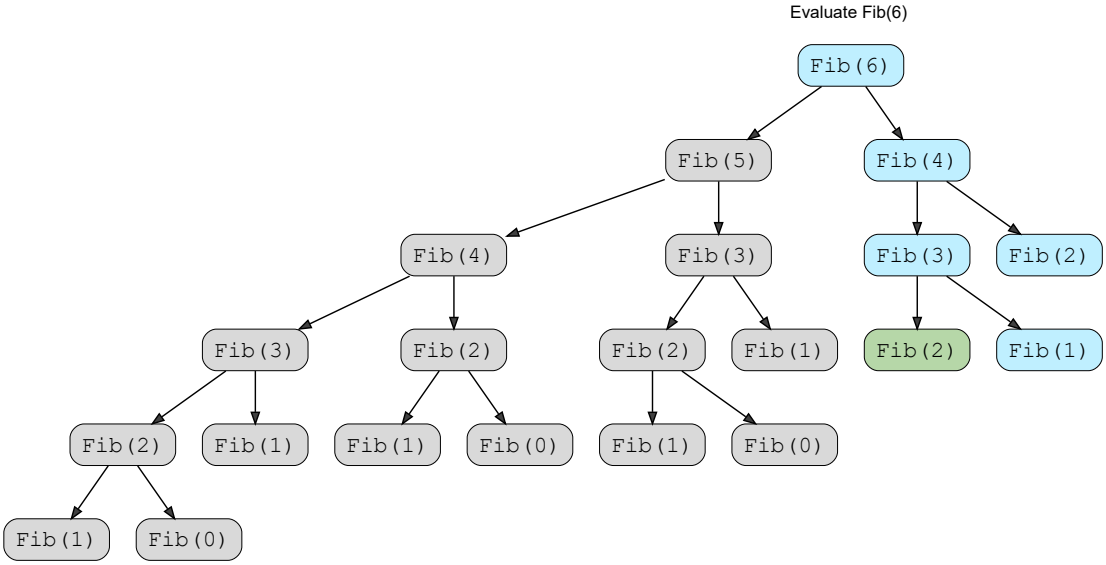

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

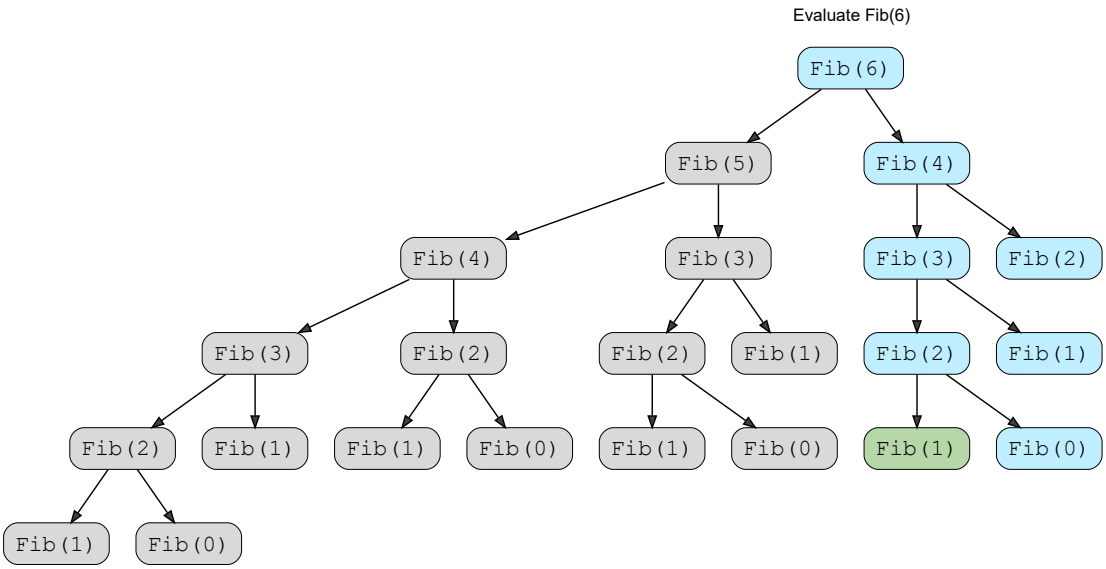

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

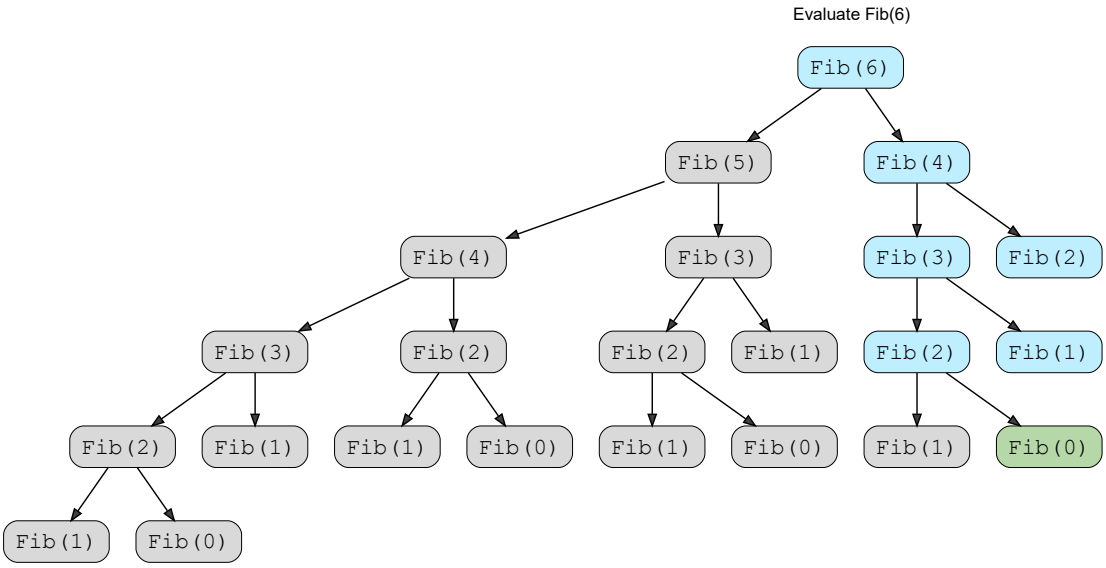

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

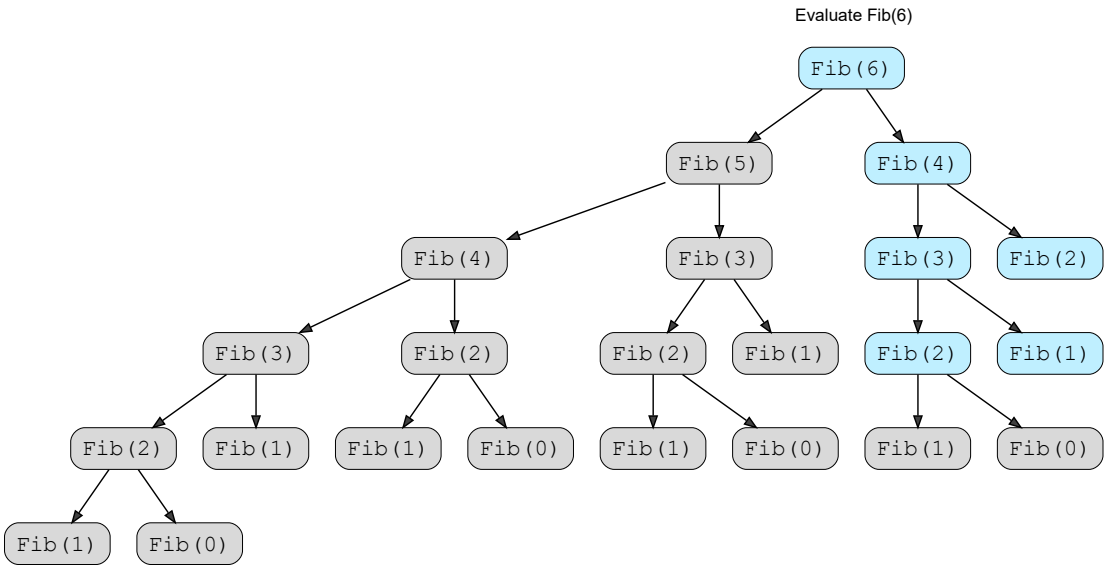

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

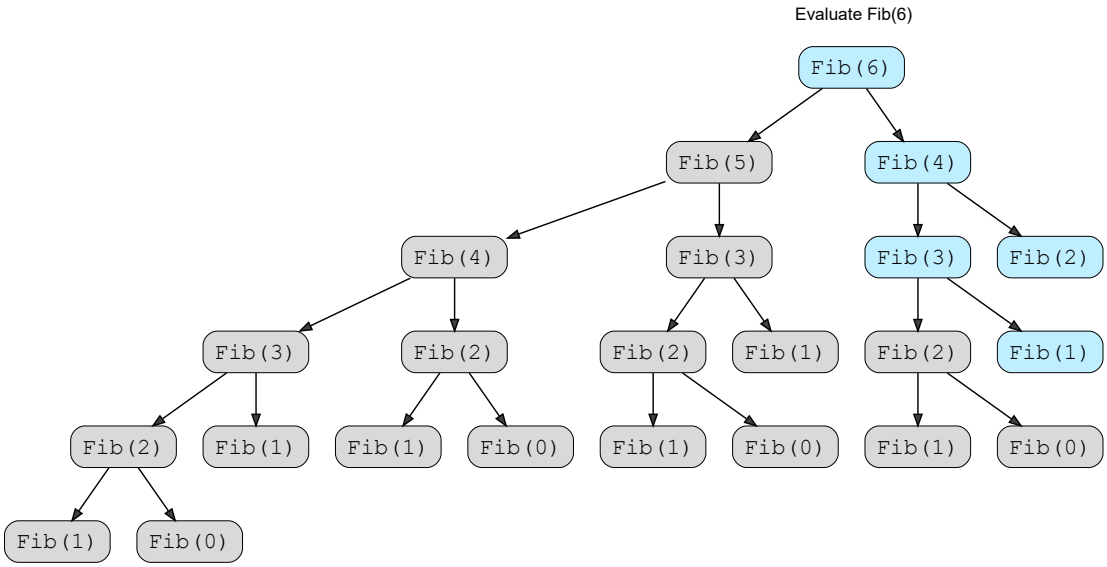

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

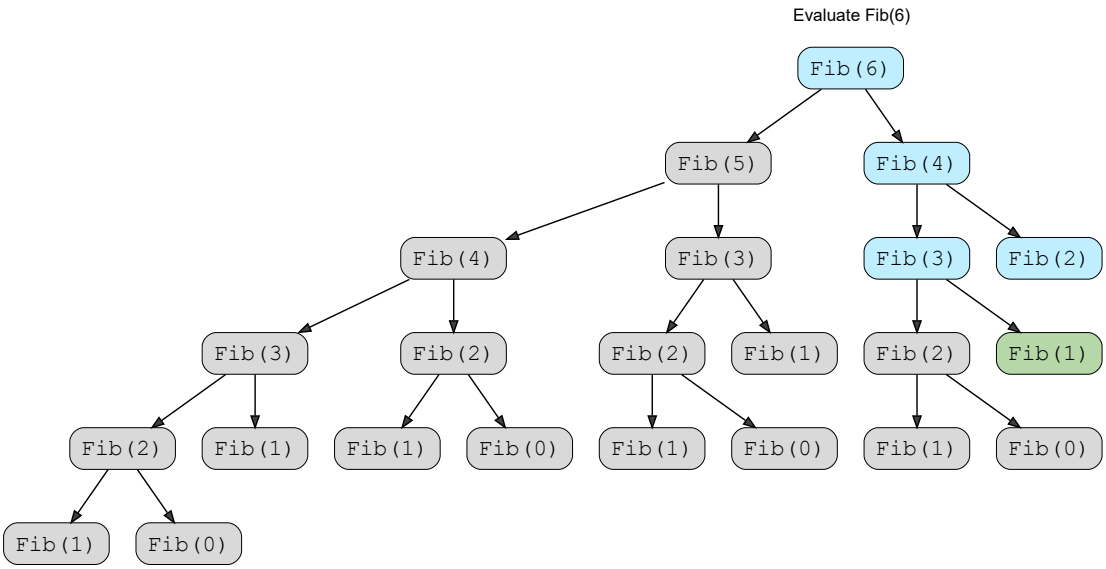

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

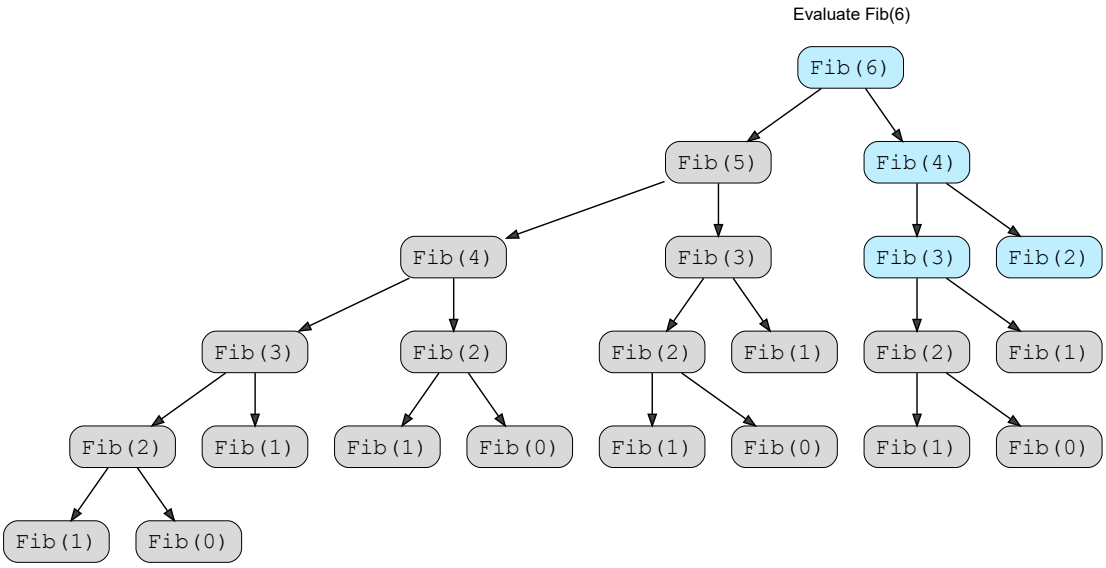

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

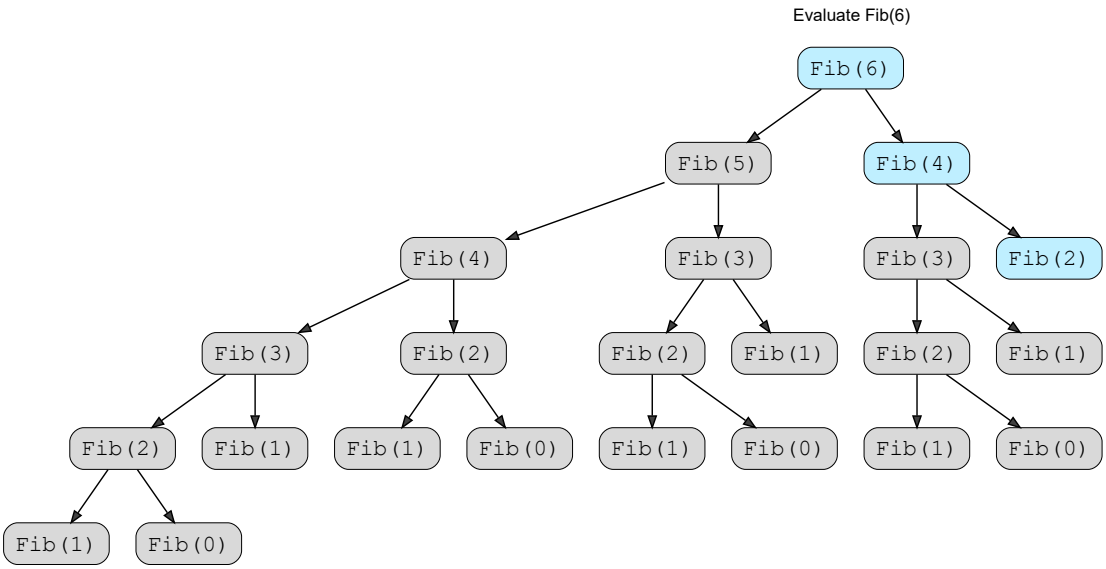

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

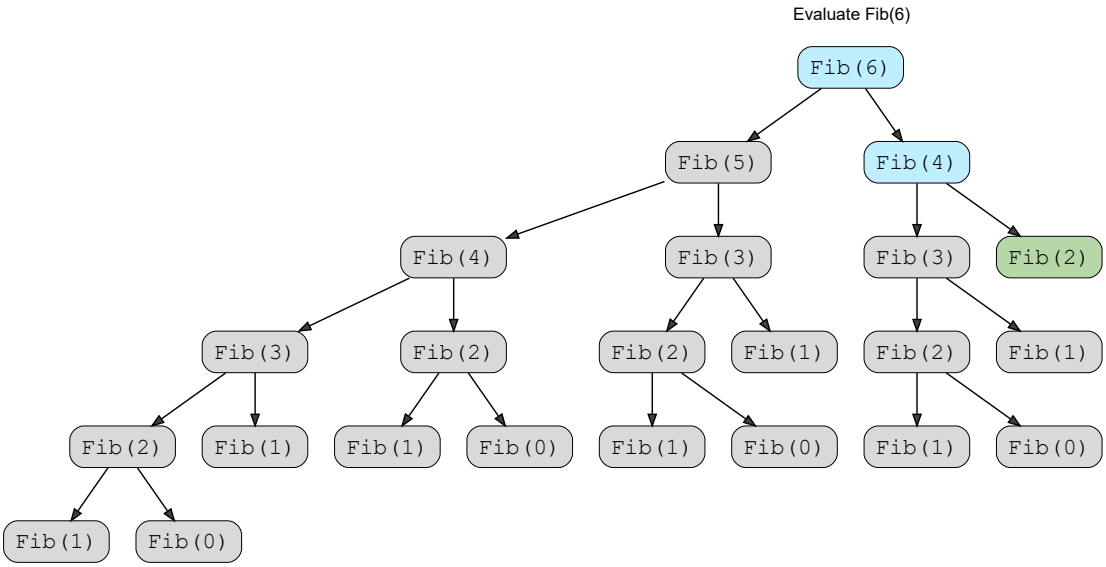

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

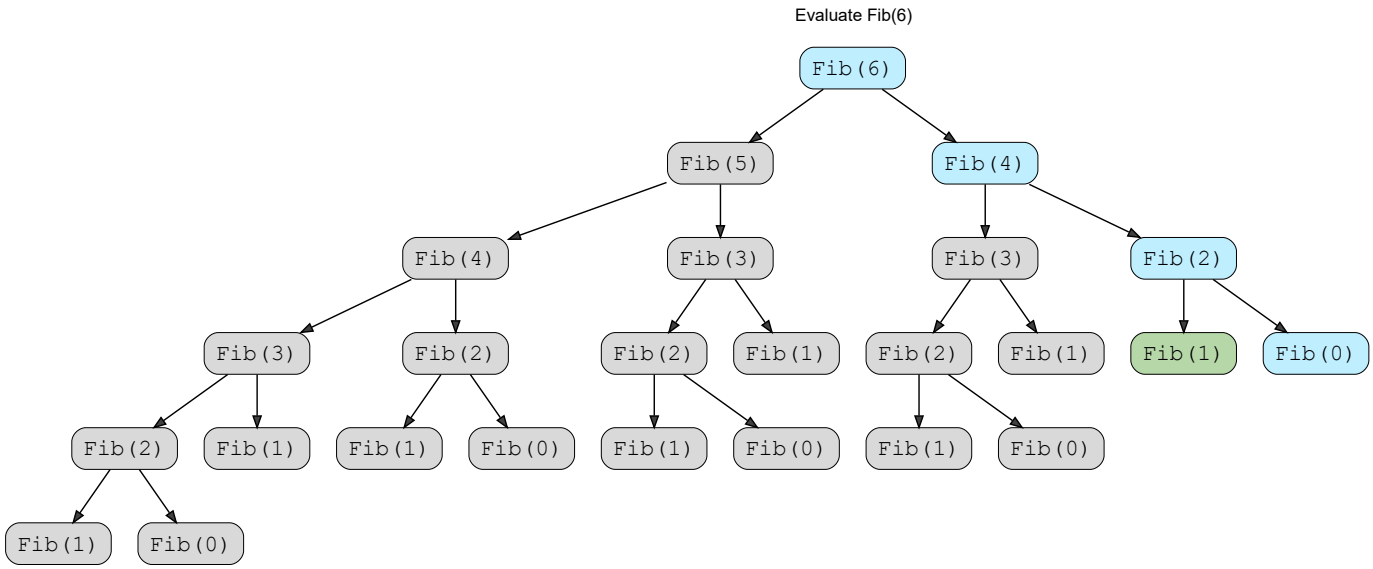

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

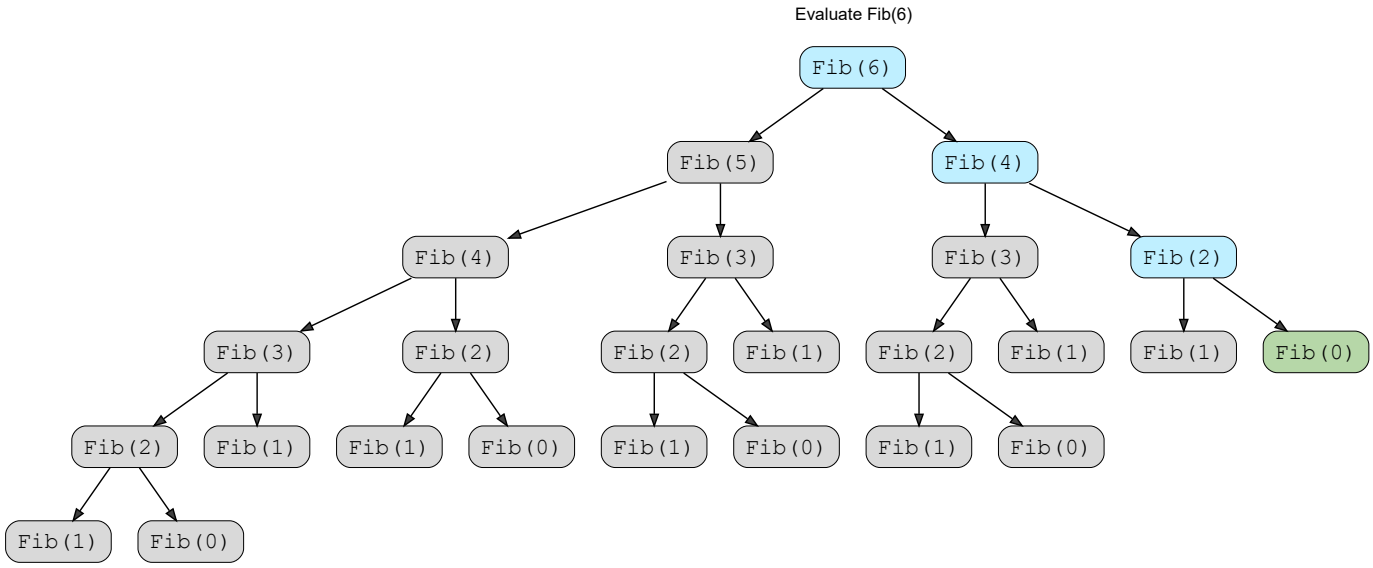

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

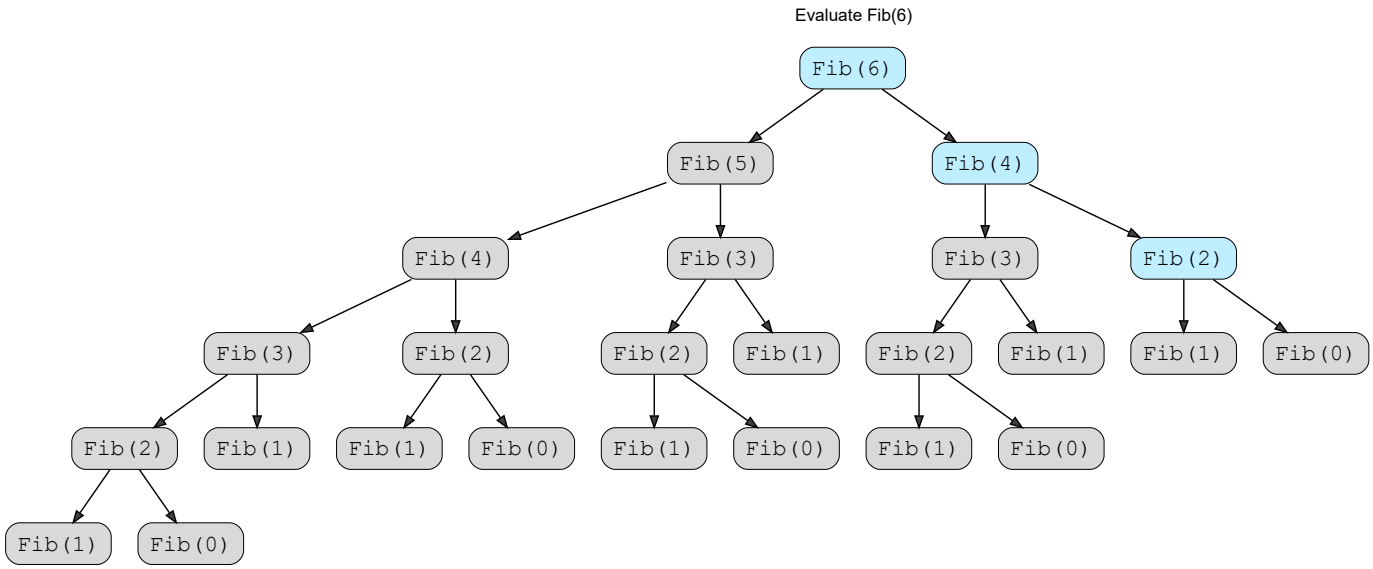

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

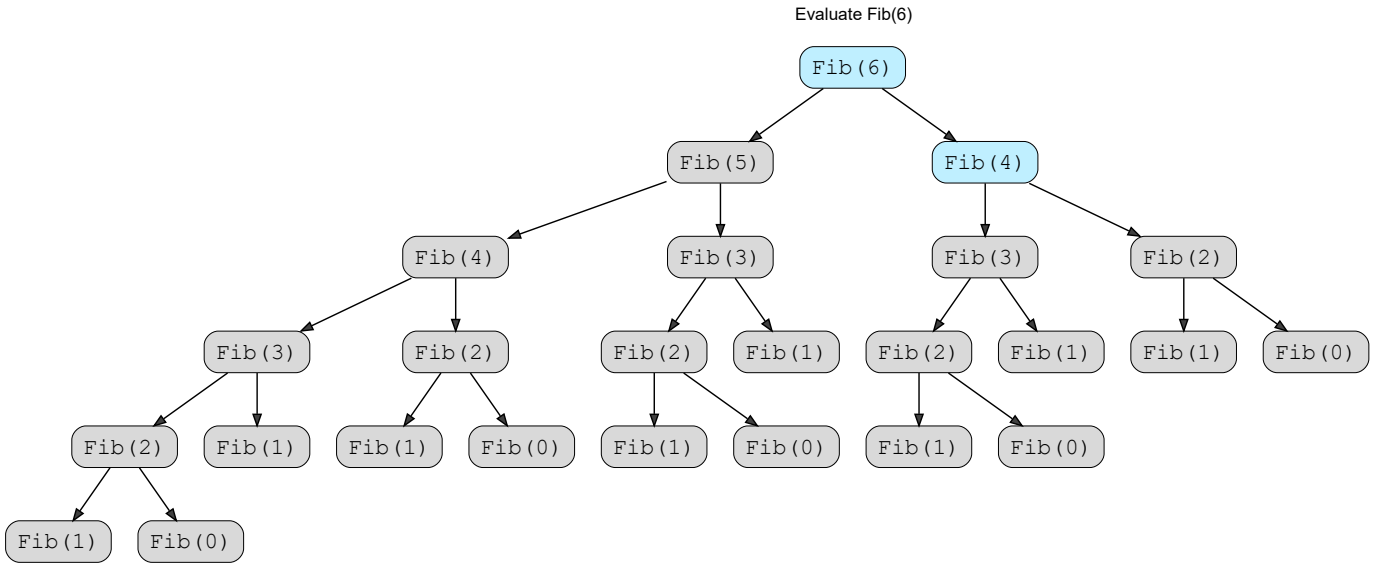

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

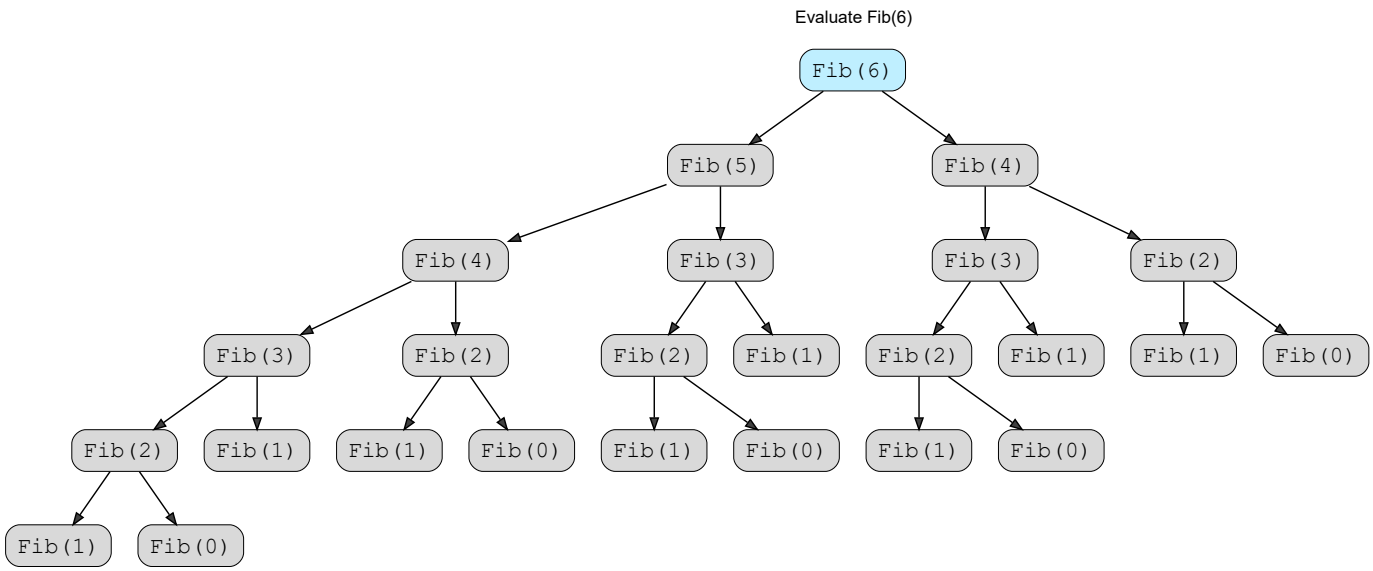

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)


Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

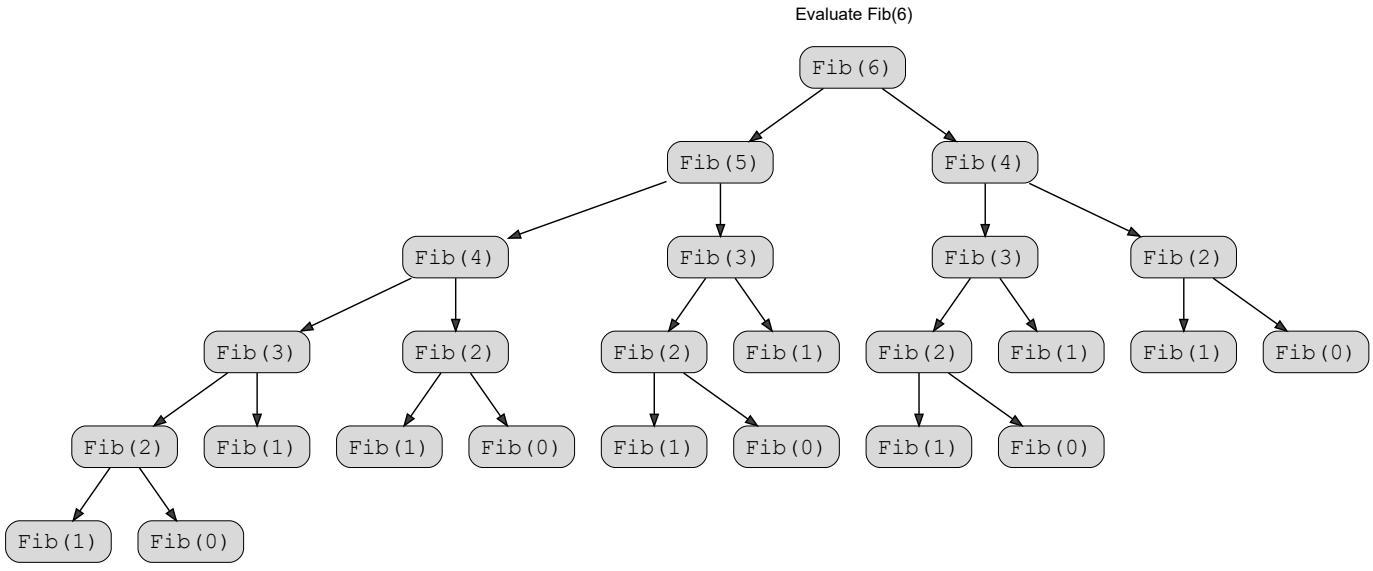

Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)


Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)


Evaluate Fib(6)
Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)
Fib(1)
Fib(0)



45 of 46



46 of 46

In the above visualization, we can see that many Fibonacci numbers are being re-evaluated, e.g., `Fib(4)` is evaluated twice. We will see how we can optimize this in the following chapters. For now, the important thing to notice is how recursion takes place. `Fib(6)`, for example, waits for `Fib(5)` and `Fib(4)` to be evaluated, which in turn waits for their previous two numbers to be evaluated, thus taking the shape of a tree. In Fibonacci numbers, we only make two recursive calls, forming a binary tree. In many other cases, we have to make more than just two

recursive calls, which makes simple recursion slow and inefficient. There are ways around it, and we will learn about them in later chapters of this course.

In the next lesson, we will hone our recursion skills by solving a coding challenge.