sourcemind

Data Structures and Algorithms

Lionel KITIHOUN

Session 2

Recursion

Objectives

- Understand recursion.
- Solve some classic problems using recursion.
- Understand recursion drawbacks.
- Learn how to transform recursive code into its iterative version.

Definition

A recursive function is a function that calls itself.

sourcemind

Common concept

It may seem weird at first, but it is something very common and used a lot in mathematics.

It is built on the idea of a function which uses itself in its definition.

Classic example: the factorial function

We all know the factorial function.

$$n! = \begin{cases} 1, & if\ n \leq 1 \\ n * (n-1)! & otherwise \end{cases}$$

Factorial function pseudocode

```
function factorial(n):
    if n == 1 or n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

Complexity of recursive factorial

- We can identify 4 operations + the call to `factorial(n - 1)`.
- Let T(N) be the cost function.
- So T(N) = 4 + T(N - 1).
- We need to solve this.

Find the complexity formula

- How?
- By trial and test.
- We find the value of the function for a some numbers and we derive a formula.

N	0	1	2	3	4	5	6	7	8	9	10	11	12
T(N)	**3**	**3**	**7**	**11**	**15**	**19**	**23**	**27**	**31**	**35**	**39**	**43**	**47**

- There is a pattern.
- $T(N) = 4N - 1$
- So the complexity is $O(N)$.

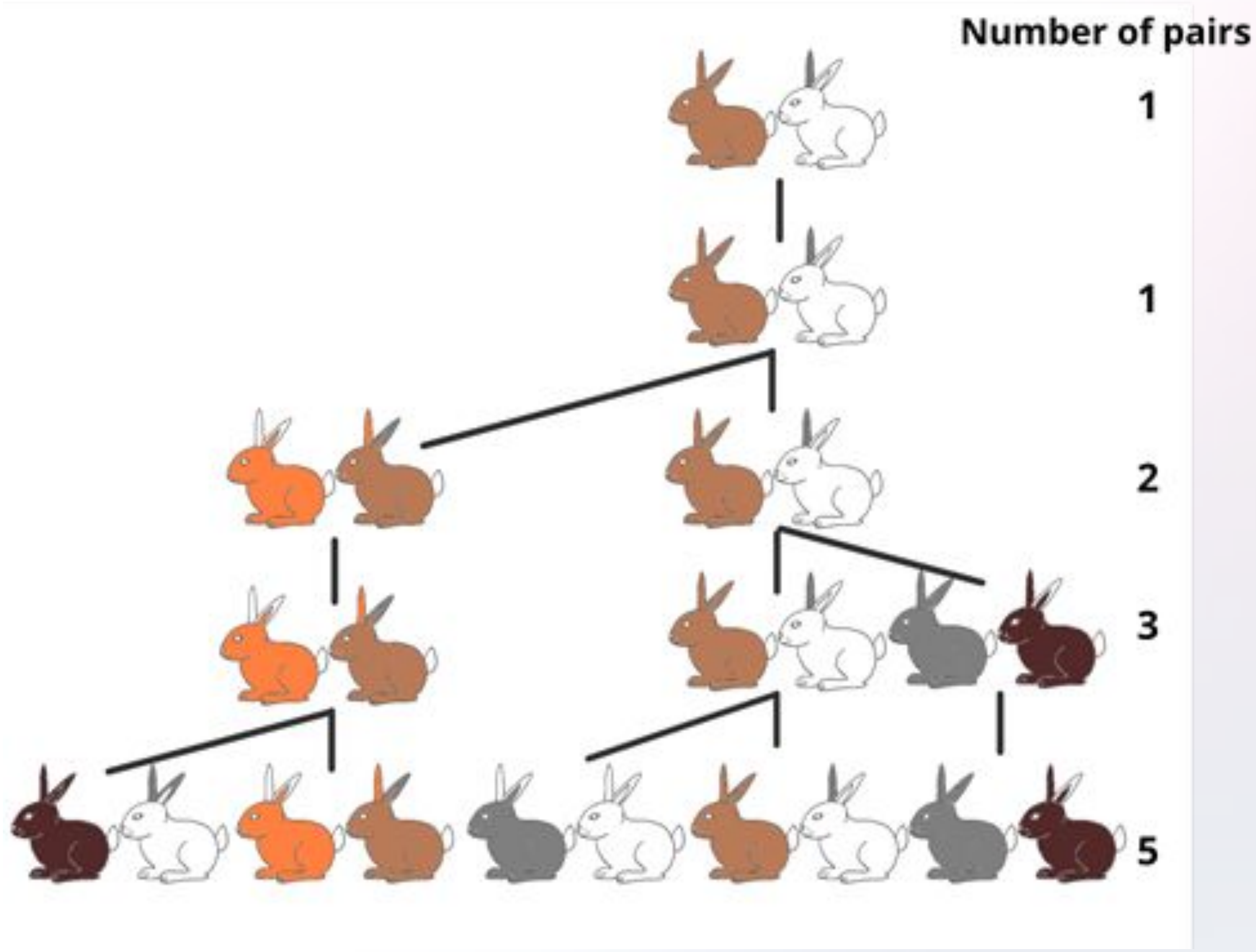
Format

Recursive function often follow a simple pattern.

- A stop condition, which allows to stop the recursion process and return from the function.
- An intermediate state, where we may need to do some computations before calling the function again.

Second example: Fibonacci sequence

A man put a male-female pair of newly born rabbits in a field. Rabbits take a month to mature before mating. One month after mating, females give birth to one male-female pair and then mate again. No rabbits die. How many rabbit pairs are there after one year?



Source: https://www.imaginationstationtoledo.org

Simulation of the process

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Adult	0	1	1	2	3	5	8	13	21	34	55	89
Young	1	0	1	1	2	3	5	8	13	21	34	55
Total	1	1	2	3	5	8	13	21	34	55	89	144

Fibonacci formula

$$F_N = F_{N-1} + F_{N-2}$$

Pseudocode

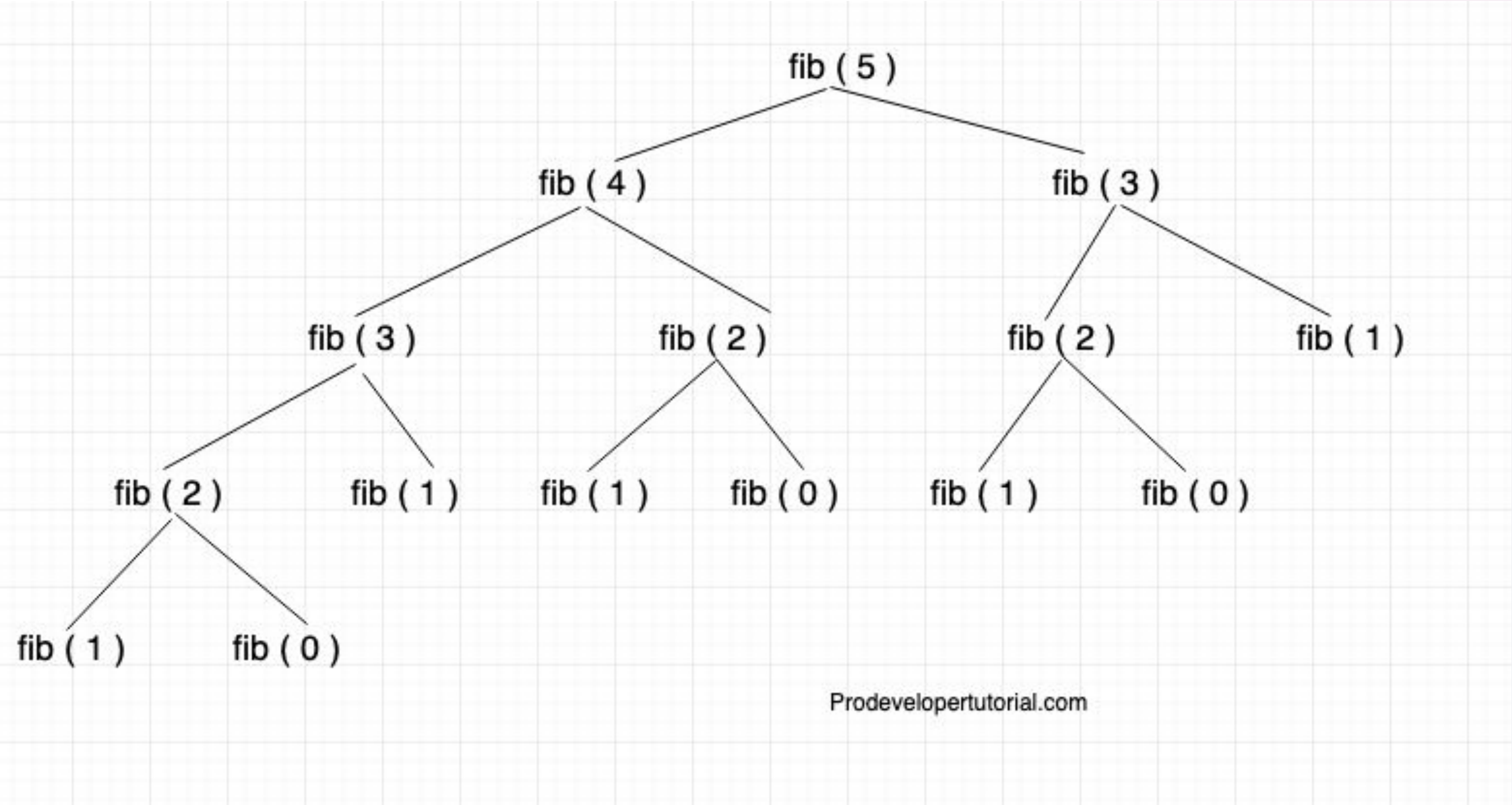
```
function fibonacci(n):
   if n == 1 or n == 0:
         return n
   else:
         return fibonacci(n - 2) + fibonacci(n - 1)
```

Complexity of recursive Fibonacci

Can be found using two methods:

- Intuition
- Recurrence

Fibonacci recursion tree



sourcemind

- There are approximately N levels.
- The number of nodes double on each level

First estimation of Fibonacci complexity

- The number of nodes is bound by $K = 1 + 2 + 4 + \dots + 2^{(N+1)}$
- Looks like complexity is $O(2^N)$
- But it is not a tight bound

Second approach

$T(N) = T(N - 1) + T(N - 2) + 1$

$T(N) = 2\ T(N - 1) + 1$

$T(N) = 2\ (2\ T(N - 2) + 1) + 1 = \ \ 4\ T(N - 2) + 3$

$T(N) = 4\ (2\ T(N - 3) + 1) + 3 = \ \ 8\ T(N - 3) + 7$

$T(N) = 8\ (2\ T(N - 4) + 1) + 7 = 16\ T(N - 4) + 15$

Do you see a pattern?

Formula

- $T(N) = 2^K T(N - K) + (2^K - 1)$
- This formula holds for any value of K, even when N = K
- $T(N) = 2^K T(0) + (2^K - 1)$
- $T(N) = 2^N + 2^N - 1$
- Complexity $O(2^N)$

Better bound for the complexity

It has been proved that the complexity of the function is actually

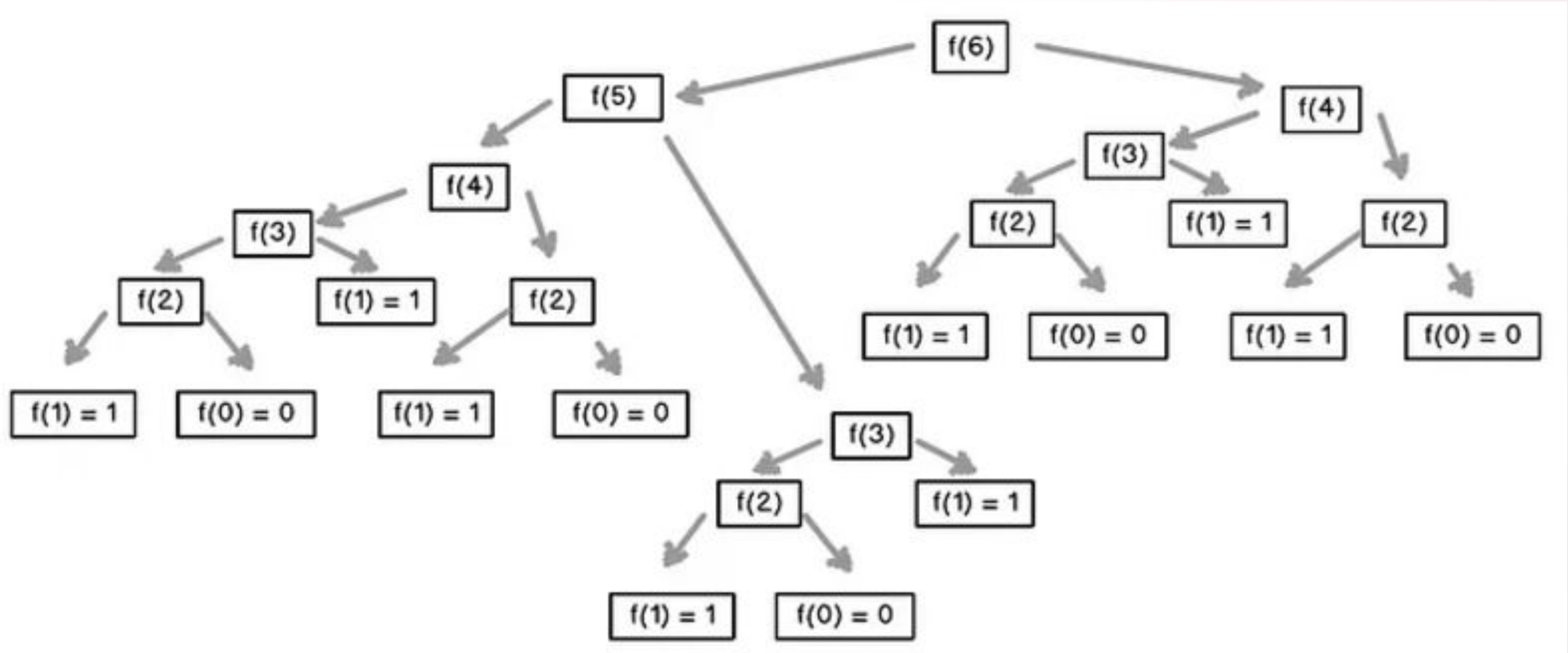
$$O(\Phi^N)$$

where $\Phi = (1 + \sqrt{5}) / 2$

Pros and cons of recursion

Pro	Cons
Cool and elegant 😎	Can be slow
Simplify code	Memory usage (not stack-friendly)
Save time and thinking 😂	**Can lead to useless code repetition if not implemented efficiently**

Fibonacci recursion tree



Source: https://medium.com/launch-school

sourcemind

Avoid repetitive tasks with memoization

A programming technique use to store the result of previous calculations in a convenient data structure (an array, a dictionary, or a map) to avoid repetition.

- Very efficient.
- Help reduce recursion overhead.
- Save time.

Fibonacci with memoization

```
dp = [-1, -1, -1, …, -1]
function fibonacci(n):
    if n == 1 or n == 0:
        return n
    else:
        if dp[n] == -1:
            dp[n] = fibonacci(n - 2) + factorial(n - 1)
        return dp[n]
```

From recursive code to iterative code

Often, a recursive function can be transformed into a loop, with sometimes a little head scratching.

Factorial iterative version

```
function factorial(n):
    f = 1
    for i = 1 to n:
        f = f * i
    return f
```

Fibonacci iterative version

```
function fibonacci(n):
   a, b, c = 0, 1, 1
   for i = 3 to n:
         tmp = c
         c = a + b
         a = b
         b = tmp
   return c
```