Dynamic Programming

algrthm

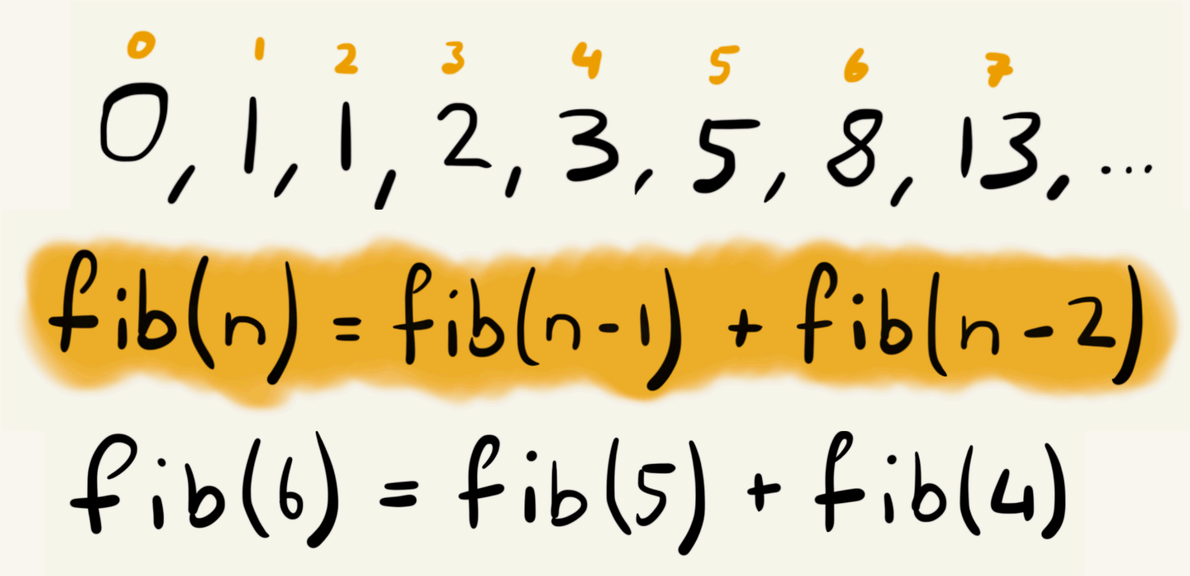
Alex Mackechnie

What is Dynamic Programming?

Dynamic Programming is a method used to solve problems by breaking the main problem up into subproblems, storing and reusing the results, and combining the solutions.

It was developed by Richard E. Bellman in 1953 while working at RAND Corporation.

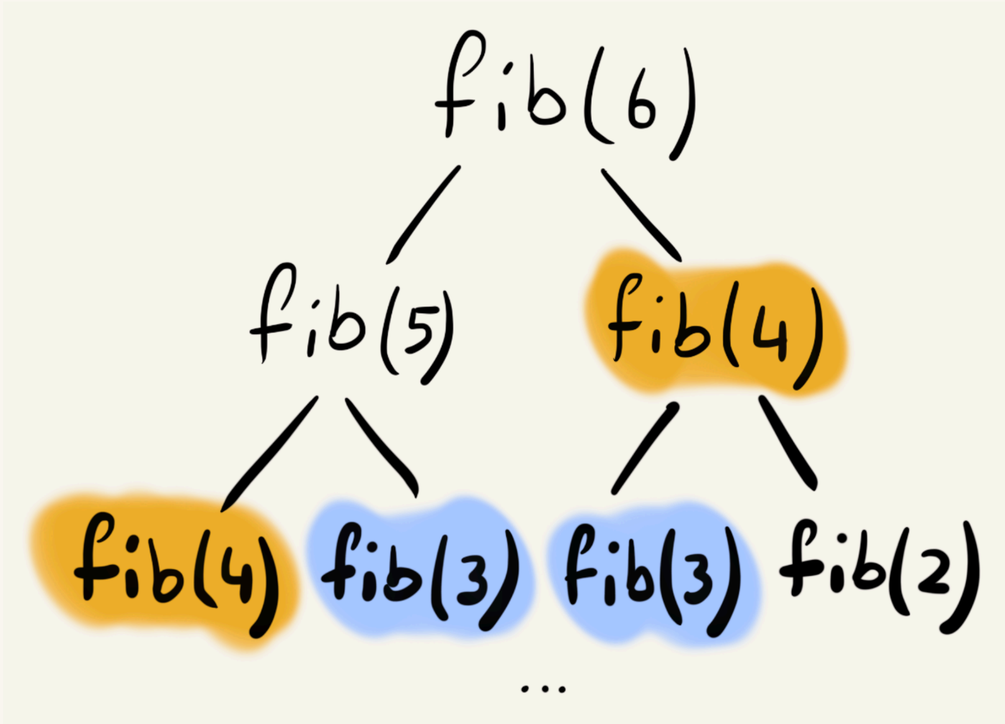
Problem Requirements



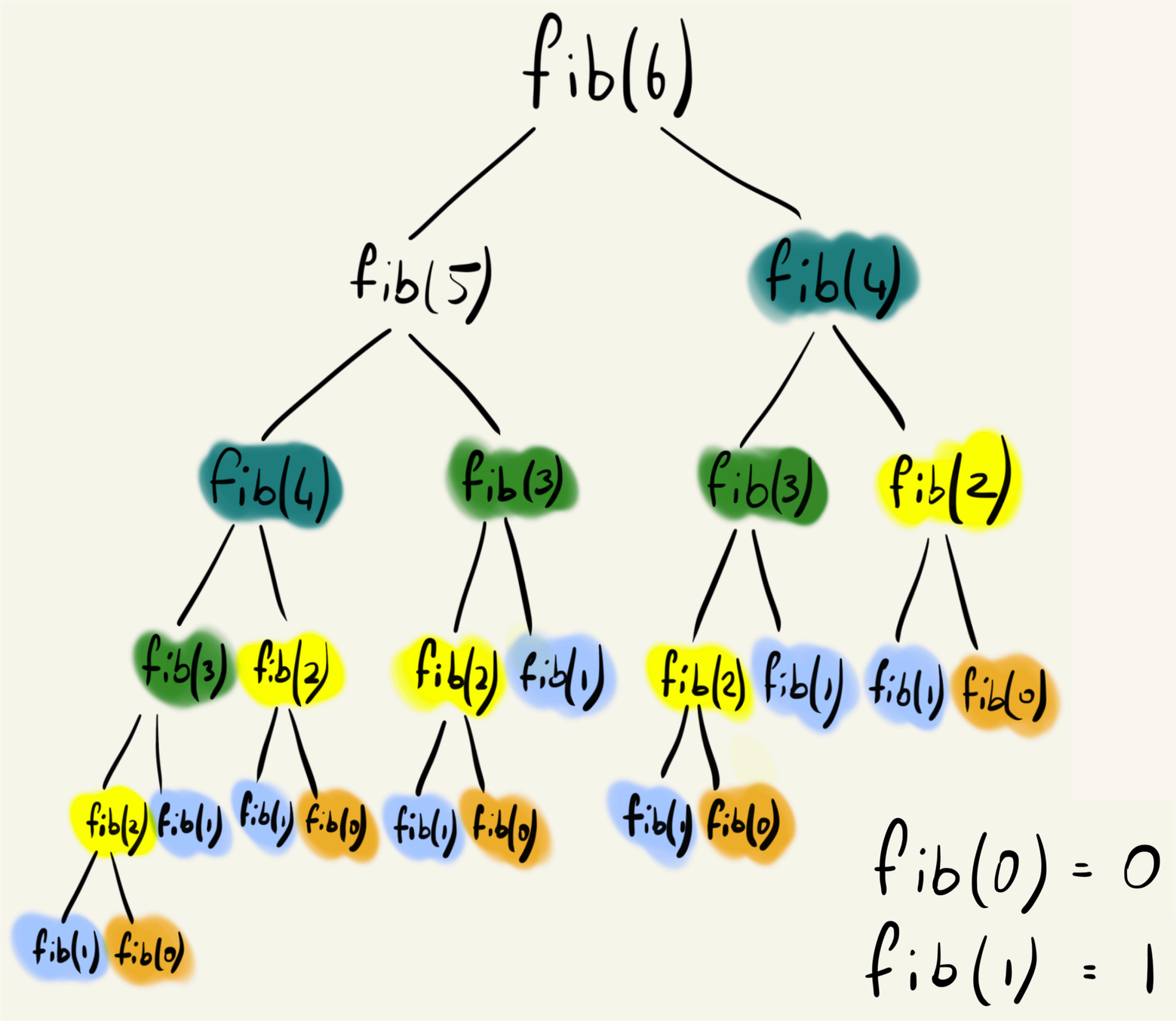
In order for Dynamic Programming to be used to solve a problem, the problem must have the following properties:

1. **Optimal Substructure**
   An optimal solution can be calculated from optimal solutions of subproblems.

2. **Overlapping Subproblems**
   Subproblems are solved multiple times.



Fibonacci Recursion Tree



Recursive Solution

This is not Dynamic Programming.


```
FIB(n)
1    if n == 0 or n == 1 then
2        return n
3    return FIB(n-1) + FIB(n-2)
```

Dynamic Programming Solutions

Recursive Top-Down with Memoization

```
FIB-INIT(n)
1    memo = array of size n + 1 init 0
2    return FIB(n, memo)
```

```
FIB(n, memo)
1    if (n == 0 or n == 1) then
2        return n
3    if memo[n] != 0 then
4        return memo[n]
5    result = FIB(n-1, memo) + FIB(n-2, memo)
6    memo[n] = result
7    return result
```

Iterative Bottom-Up

```
FIB(n)
1    if (n == 0 or n == 1) then
2        return n
3    minusTwo = 0
4    minusOne = 1
5    fib = 0
6    for i = 2 to n-1
7        fib = minusTwo + minusOne
8        minusTwo = minusOne
9        minusOne = fib
10   return fib
```

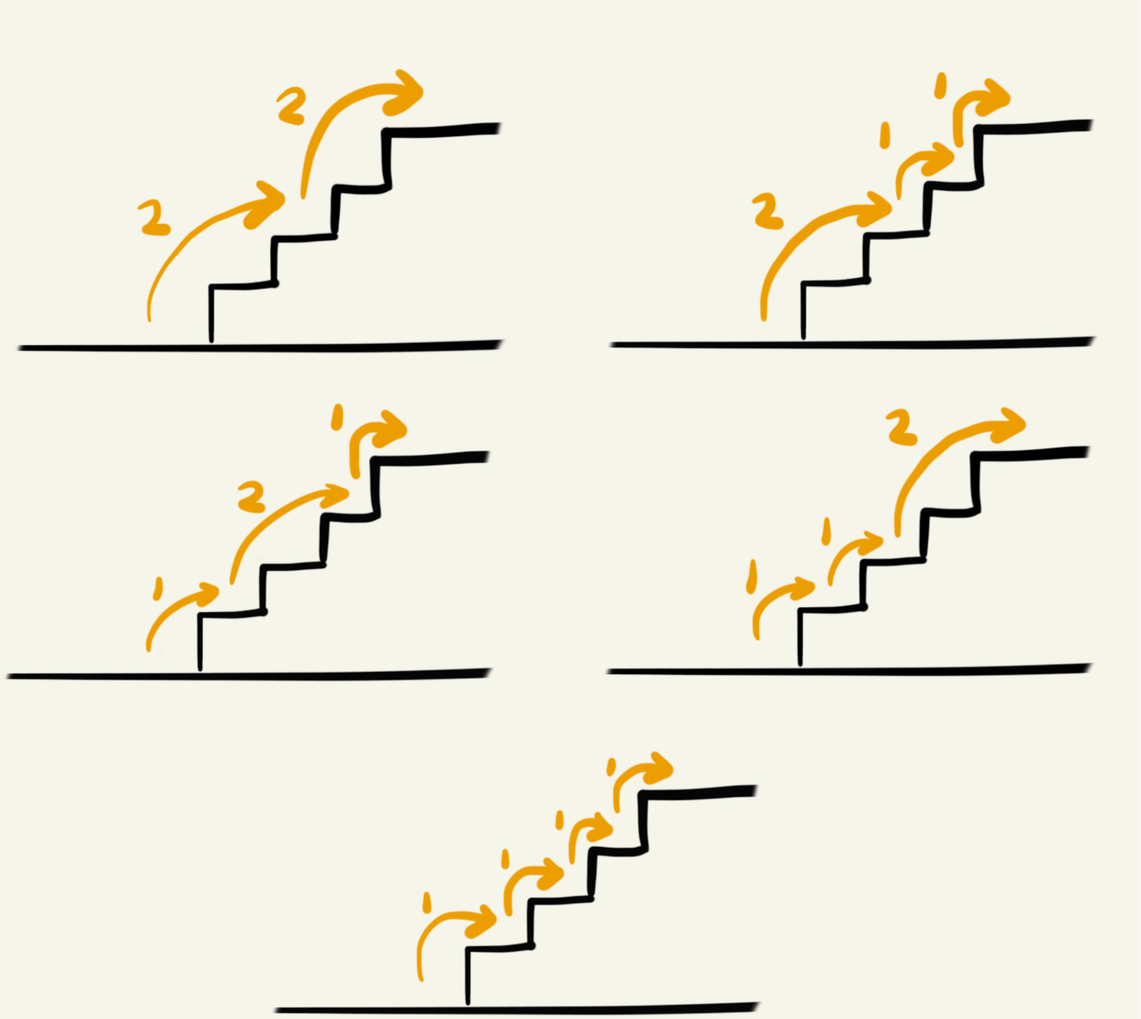
Climbing Stairs Problem

You are climbing a staircase that consists of n steps.

You can only climb 1 or 2 steps at a time. How many distinct ways can you climb to the top?

Example

If n = 4, there are 5 distinct ways to climb the stairs.

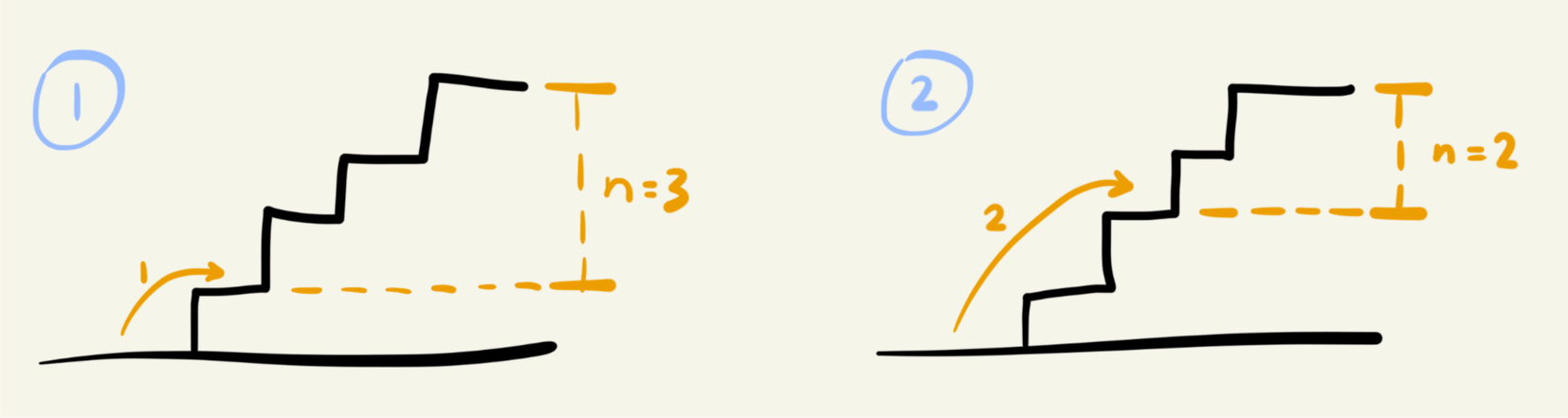


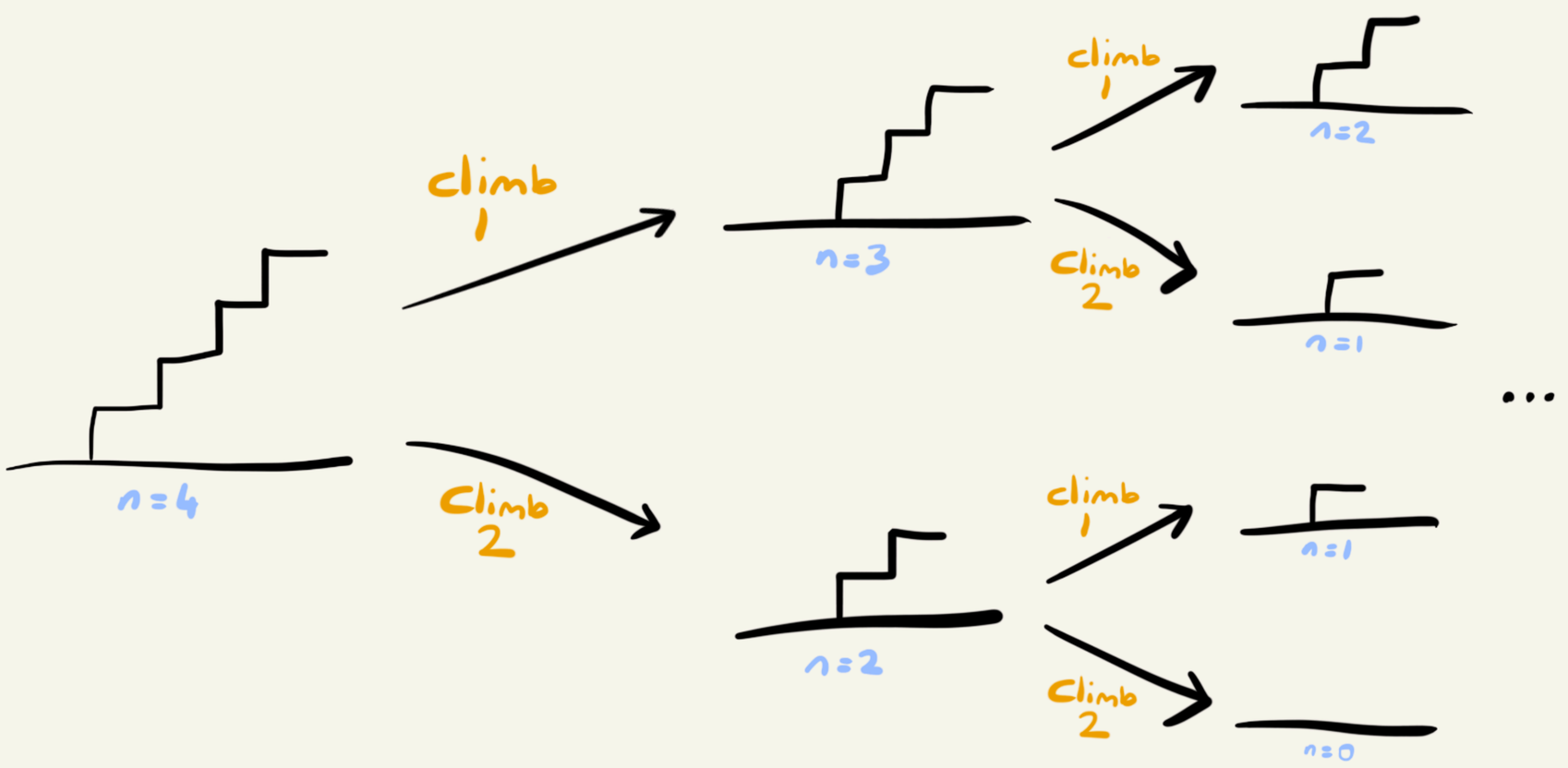
Analysis

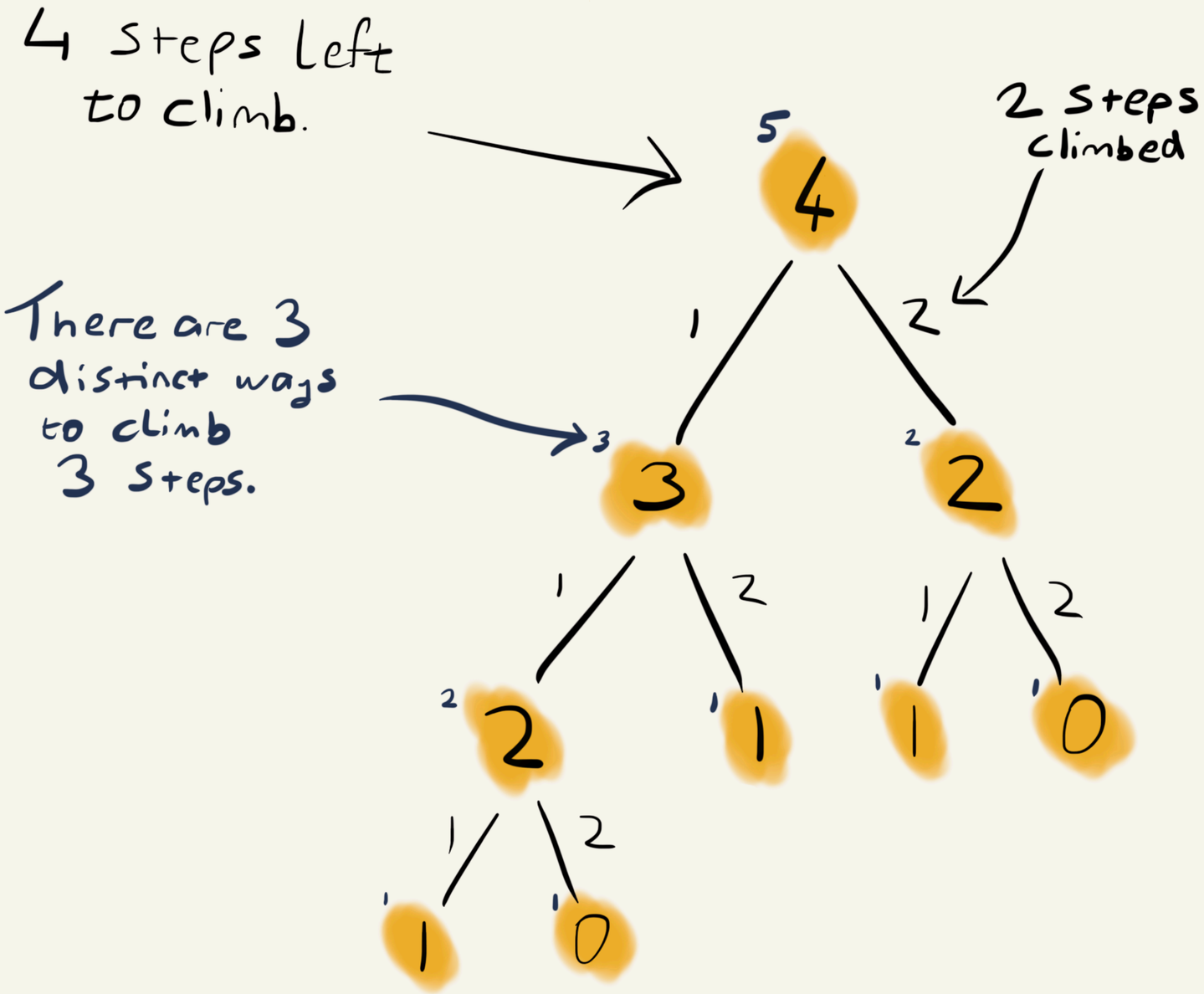
If we are at the bottom of the stairs, we have 2 options:

1. Climb 1 stair
2. Climb 2 stairs

Whichever option we choose, we will be left with a smaller version of the original problem.







Recursion Tree

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

$$f(0) = 1$$

$$f(1) = 1$$

Top-Down with Memoization Solution

```
WAYS-TO-CLIMB-INIT(n)
1   memo = array of size n + 1 init 0
2   return WAYS-TO-CLIMB(n, memo)
```

```
WAYS-TO-CLIMB(n, memo)
1   if (n == 0 or n == 1) then
2      return 1
3   if memo[n] != 0 then
4      return memo[n]
5   memo[n] = WAYS-TO-CLIMB(n-1, memo) + WAYS-TO-CLIMB(n-2, memo)
6   return memo[n]
```

Bottom-Up Solution

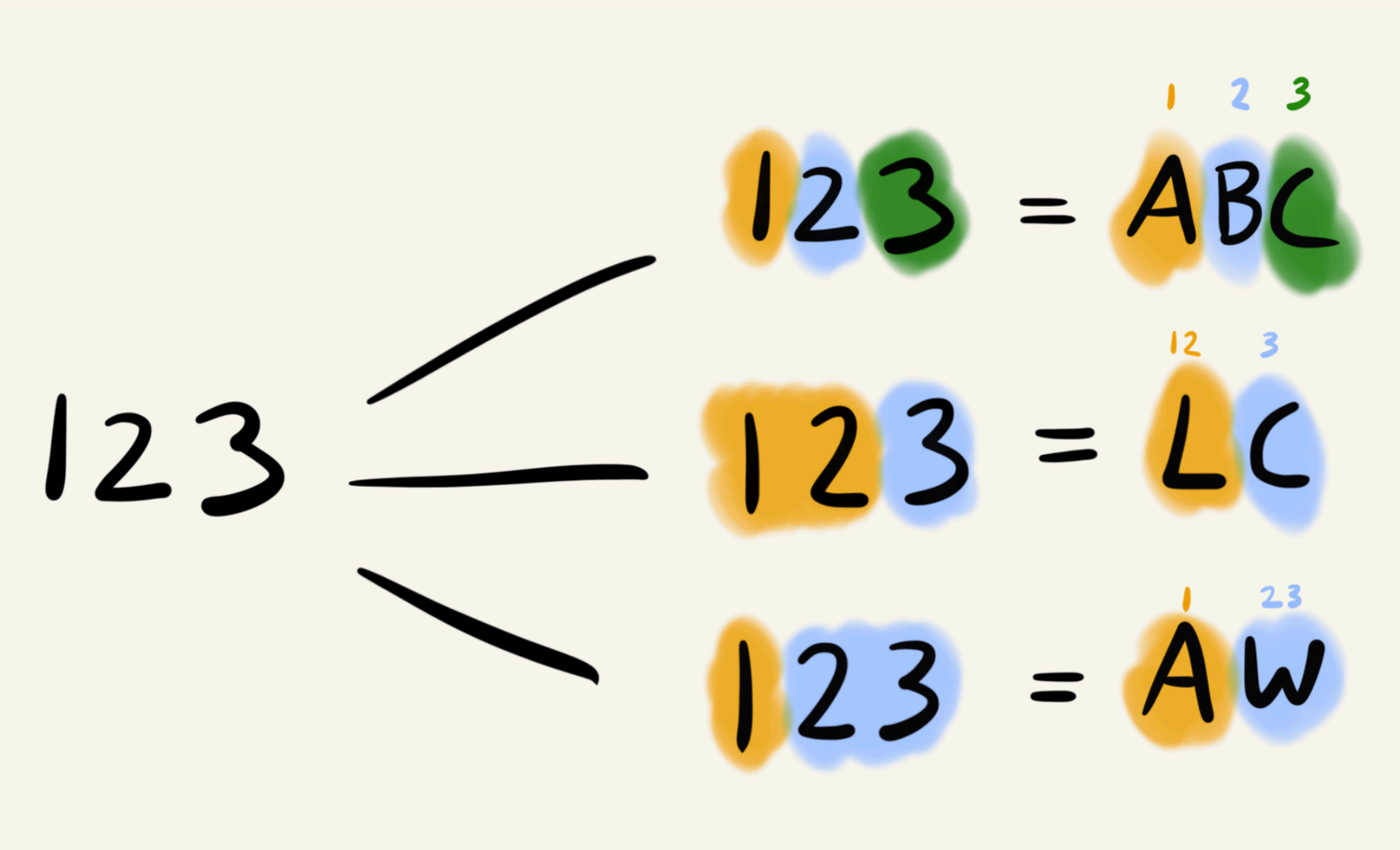
```
WAYS-TO-CLIMB(n, memo)
1    numWays = array of size n + 1 init 0
2    numWays[0] = 1
3    numWays[1] = 1
4    for i = 2 to n
5        numWays[i] = numWays[i-1] + numWays[i-2]
6    return numWays[n]
```

Decode Ways Problem

We are given a message which has been encoded into a string of digits, using the following mapping:

```
A  =  1
B  =  2
   ...
Z  =  26
```

Our goal is to find the number of possible ways we can decode the string of digits.



Analysis

If we have ‘123’, what are all of the possible options for a first letter?

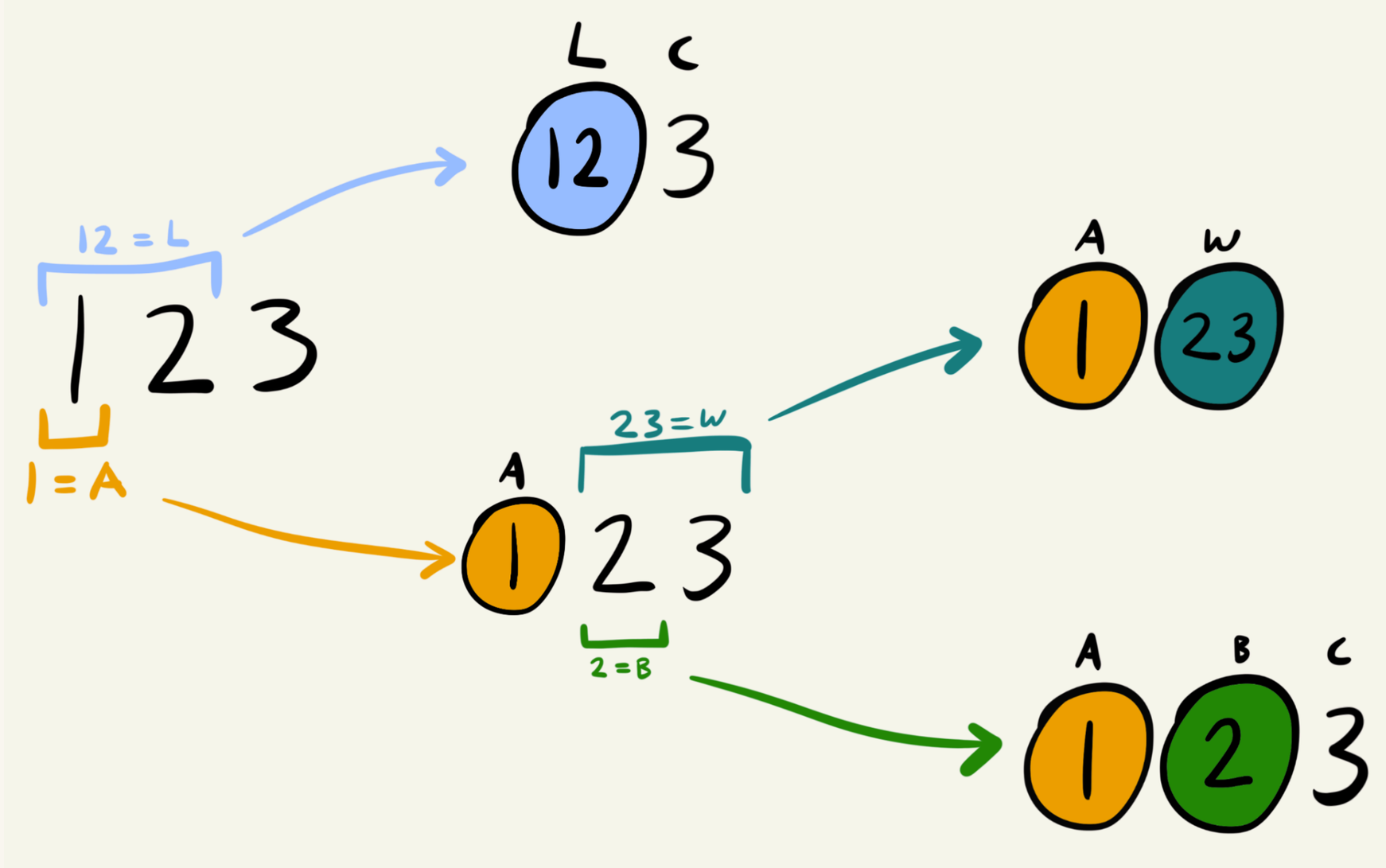
- **1 = A**
- **12 = L**

If we choose 1 (A), then we have the further two options:

- **2 = B**
- **23 = W**

If we choose 12 (L), then we only have one option:

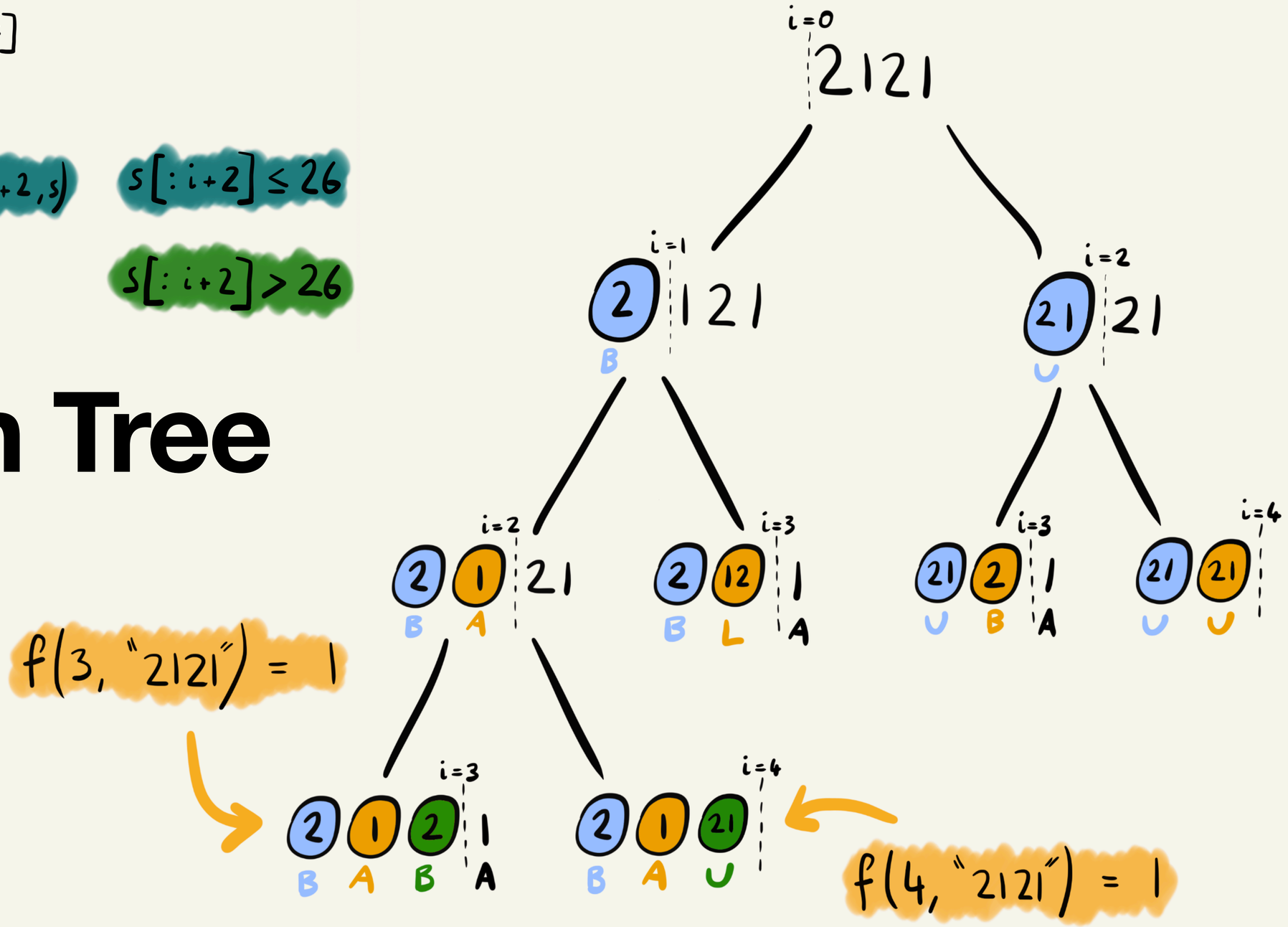
- **3 = C**



number of ways of decoding the string s[i:]

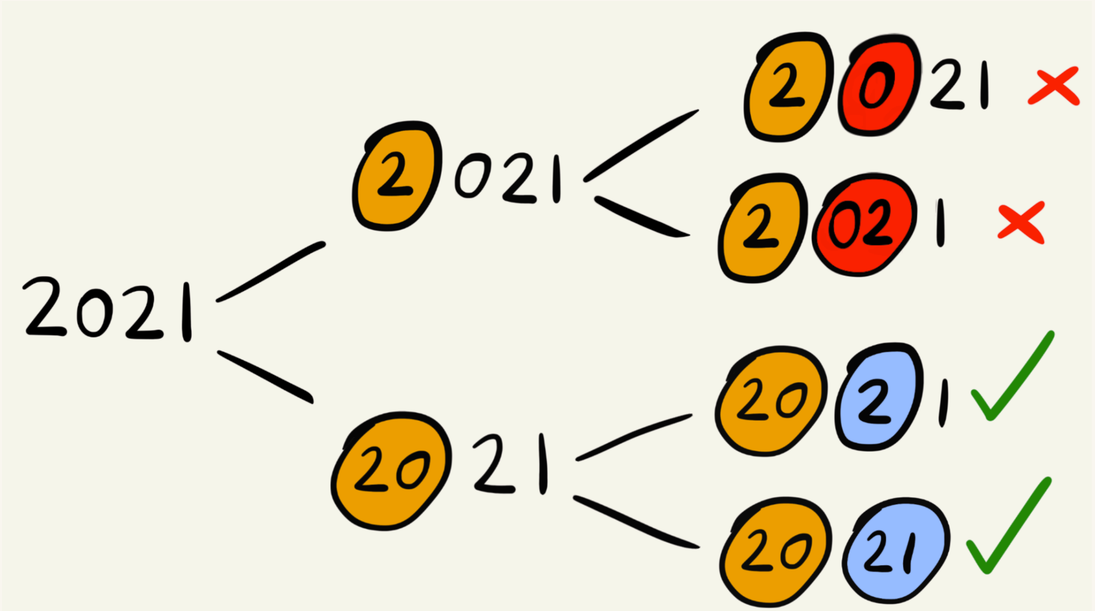
$$f(i,s) = \begin{cases} f(i+1,s) + f(i+2,s) & s[:i+2] \le 26 \\ f(i+1,s) & s[:i+2] > 26 \end{cases}$$

Recursion Tree

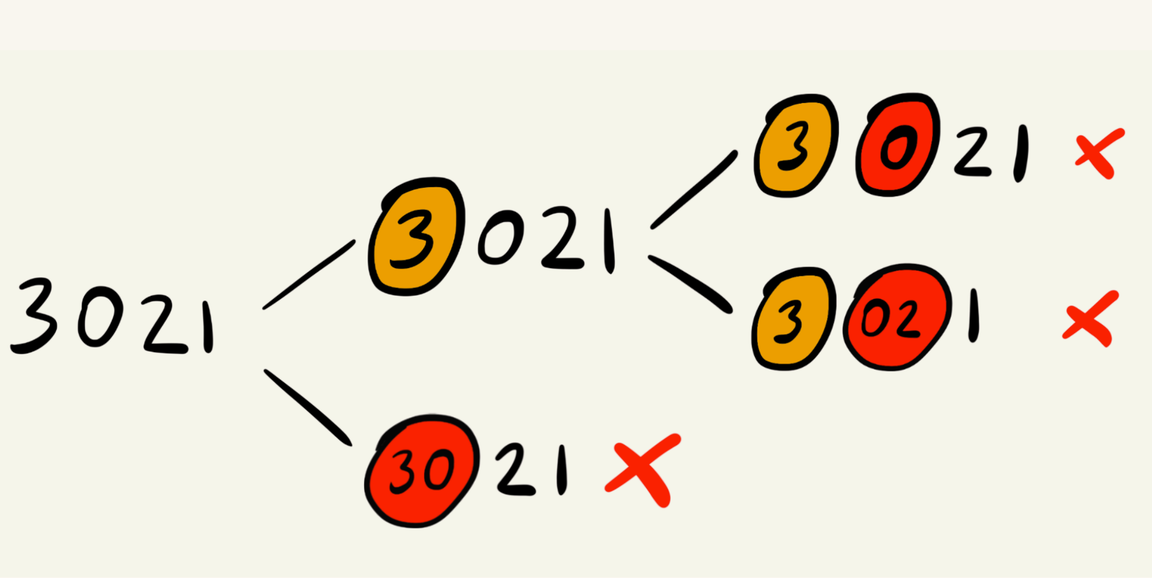


How to Deal with 0s

If a string starts with a 0, there are no ways to decode it, as there are no mappings that start with 0. Mappings start at 1 (A) and finish at 26 (Z).



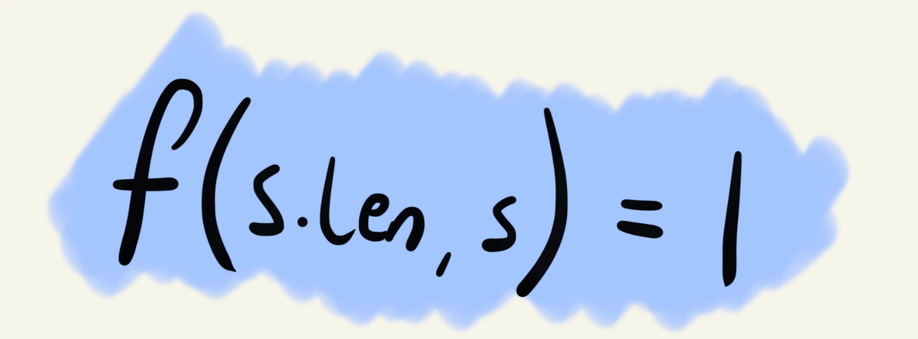
2 solutions



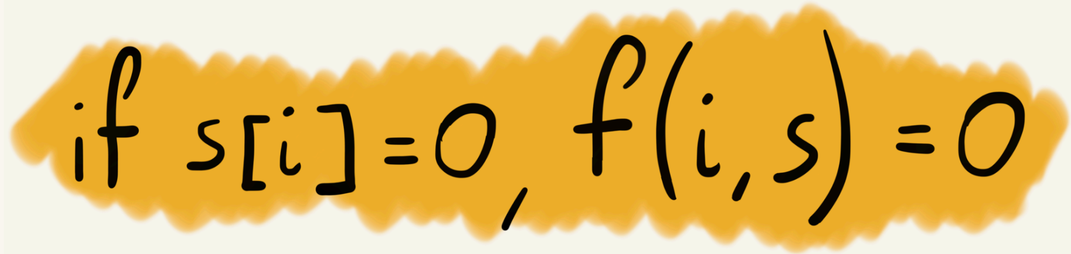
0 solutions

Base Cases

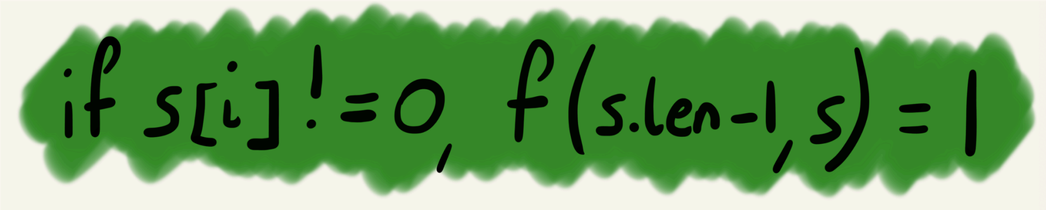
Base Case 1: If we are at the end of the string, we are essentially asking: how many ways are there to decode an empty string? There is one way of decoding “” — it is simply “”.



Base Case 2: If the value of the string at the current index is “0”, then there are going to be no ways of decoding this, as shown previously.



Base Case 3: If we are at the last index of the string, and the value is not “0”, then we know there will be one way of decoding the string as the value is definitely between 1 and 9.



Top-Down with Memoization Solution

```
WAYS-TO-DECODE-INIT(s)
1   if s == null or s.len() == 0 then
2      return 0
3   memo = new hash table
4   return WAYS-TO-CLIMB(n, memo)


WAYS-TO-DECODE(i, s, memo)
1   if memo.containsKey(i) then
2      return memo.get(i)
3   if i == s.len() then
4      return 1
5   if s[i] == 0 then
6      return 0
7   if i == s.len() - 1 then
8      return 1
9   ans = WAYS-TO-DECODE(i + 1, s, memo)
10  if s[i:i+2] <= 26 then
11     ans += WAYS-TO-DECODE(i + 2, s, memo)
12  memo.put(i, ans)
13  return ans
```

Thanks!

Twitter: **@algrthmio** & **@mackechniealex**

Website: **https://algrthm.io** (not live yet!)

Meetup: **https://www.meetup.com/algrthm**

Next Session: **25th February, 2021**