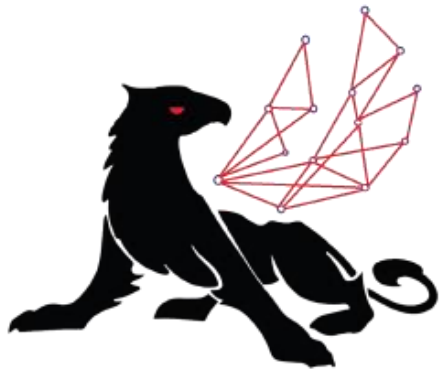


Programación



Dinámica



PROGRAMACIÓN COMPETITIVA

URJC - 2022

Organizadores:

- Isaac Lozano (isaac.lozano@urjc.es)
- Raúl Martín (raul.martin@urjc.es)
- Sergio Salazar (s.salazarc.2018@alumnos.urjc.es)
- Francisco Tórtola (f.tortola.2018@alumnos.urjc.es)
- Cristian Pérez (c.perezc.2018@alumnos.urjc.es)
- Xuqiang Liu (x.liu1.2020@alumnos.urjc.es)
- **Alicia Pina** (a.pinaz.2020@alumnos.urjc.es)
- **Sara García** (s.garciarod.2020@alumnos.urjc.es)
- Raúl Fauste (r.fauste.2020@alumnos.urjc.es)



ÍNDICE

- Motivación
- ¿Qué es?
- Características
- Top Down
- Bottom Up
- Problemas clásicos
- Más allá...



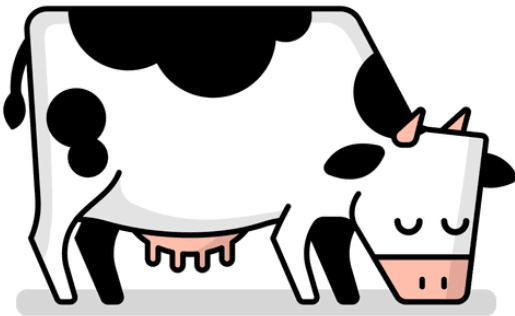
ÍNDICE

- **Motivación**
- ¿Qué es?
- Características
- Top Down
- Bottom Up
- Problemas clásicos
- Más allá...



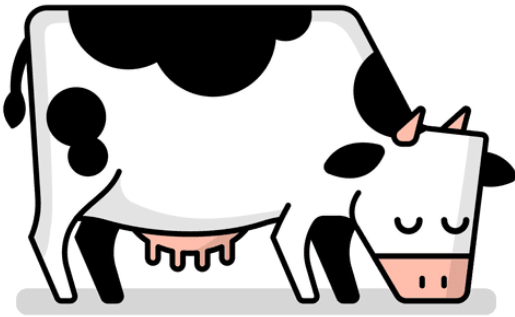
¿Por qué Greedy no es siempre válido?

- ~~Decisión óptima → Solución óptima?~~ **WA**
- Ejemplo: problema de las Vacas Pensantes ([aer 285](#))



Problema de las vacas pensantes

0



0



5

5

10

3

3

1



Problema de las vacas pensantes

0

5



5



5



10



3



3



1

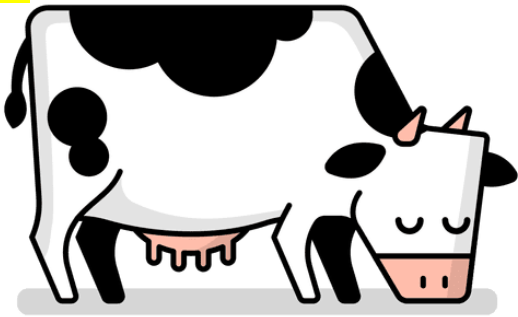


7

Problema de las vacas pensantes

5

5



5



10



3



3



1



8

Problema de las vacas pensantes

5

15



10



3



3



1

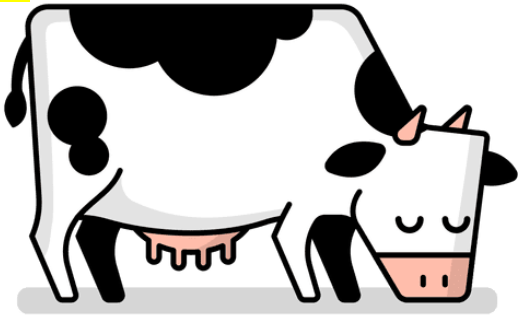


9

Problema de las vacas pensantes

8

15



3



3



1



10

Problema de las vacas pensantes

8

18



3

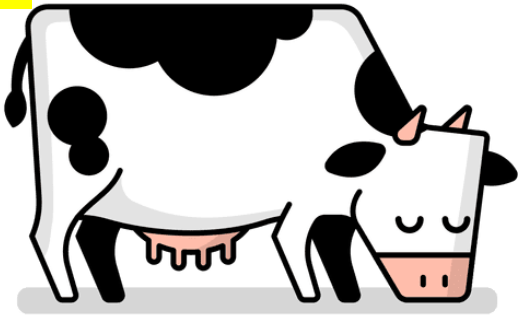
1



11

Problema de las vacas pensantes

9



18



1



12

Problema de las vacas pensantes



9



Problema de las vacas pensantes

0

5



5



5



10



3



3



1

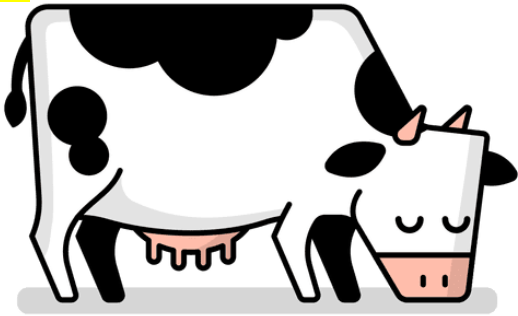


14

Problema de las vacas pensantes

1

5



5



10



3



3



1



15

Problema de las vacas pensantes

1

10



5



10



3



3

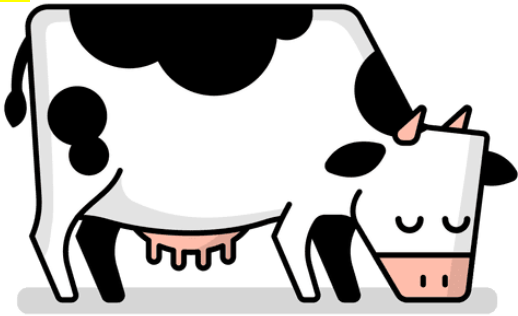


16

Problema de las vacas pensantes

11

10



10



3



3



17

Problema de las vacas pensantes

11

13



3



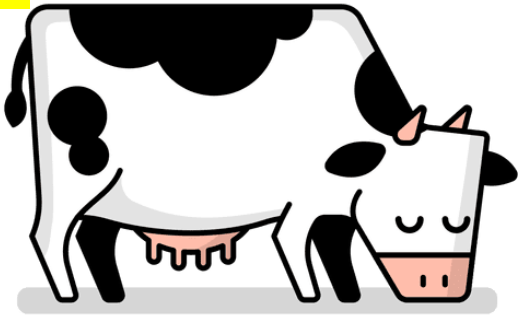
3



18

Problema de las vacas pensantes

14



13



3



19

Problema de las vacas pensantes

14



MANOLIZED TUMBLR



¿Es fuerza bruta la única solución?

- Generar todas las posibles opciones → **TLE**
- Ejemplo: programa recursivo de Fibonacci

Sucesión de **Fibonacci**: 1, 1, 2, 3, 5, 8, ...

Fibonacci(n) =

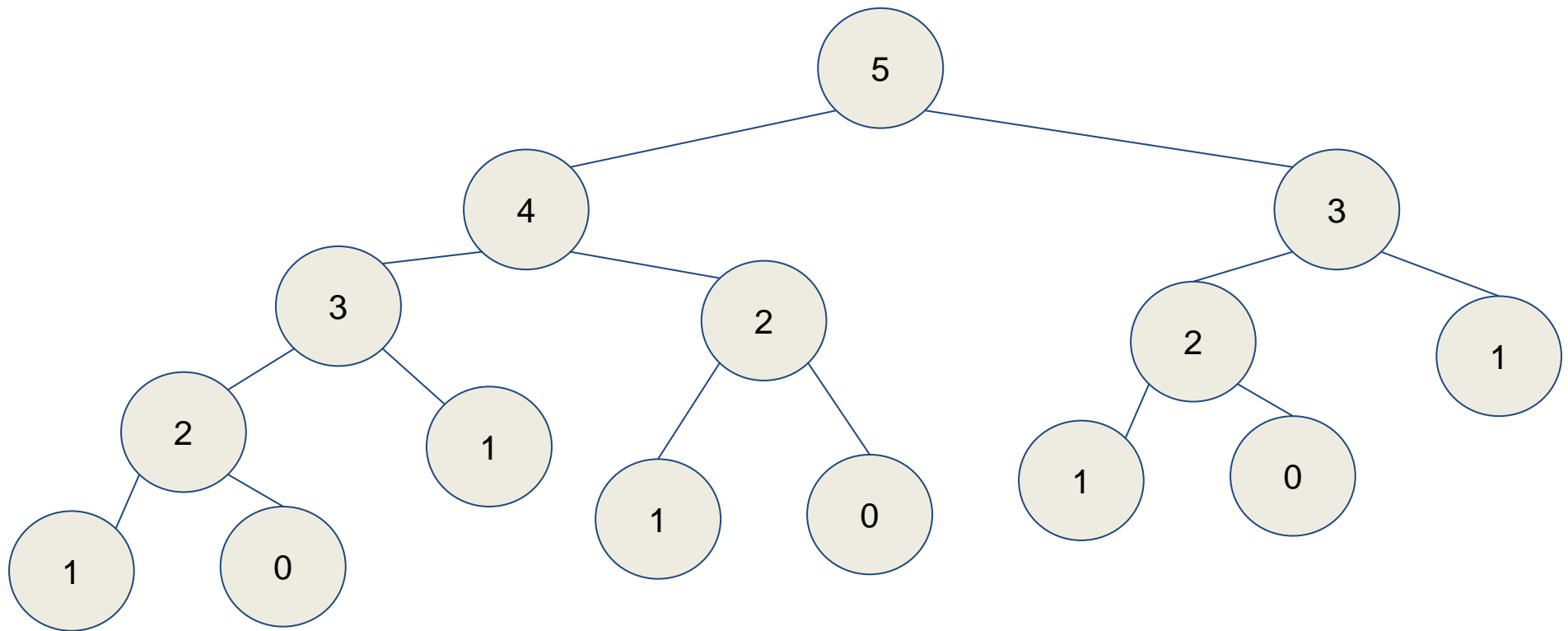
1, si $n==0$ o $n==1$

Fibonacci(n-1) + Fibonacci(n-2), en otro caso



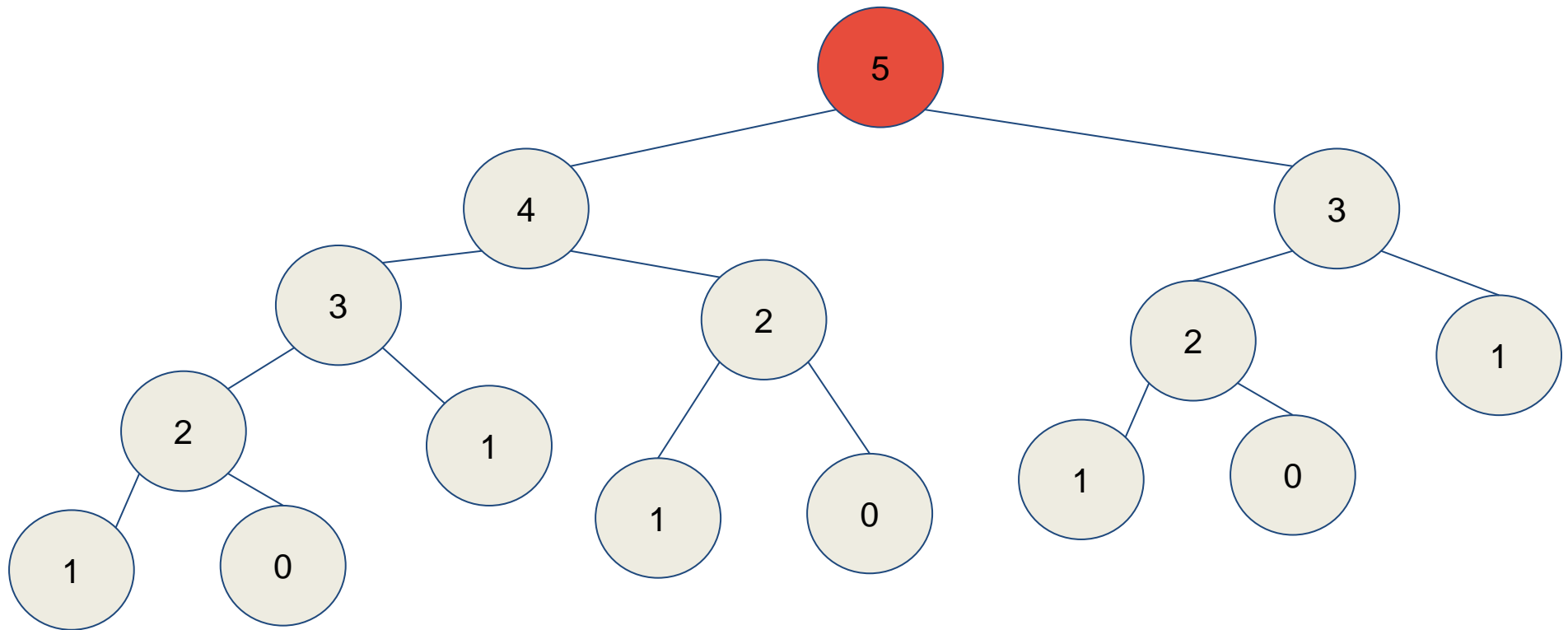
Fibonacci

Ejemplo: Fibonacci(5)



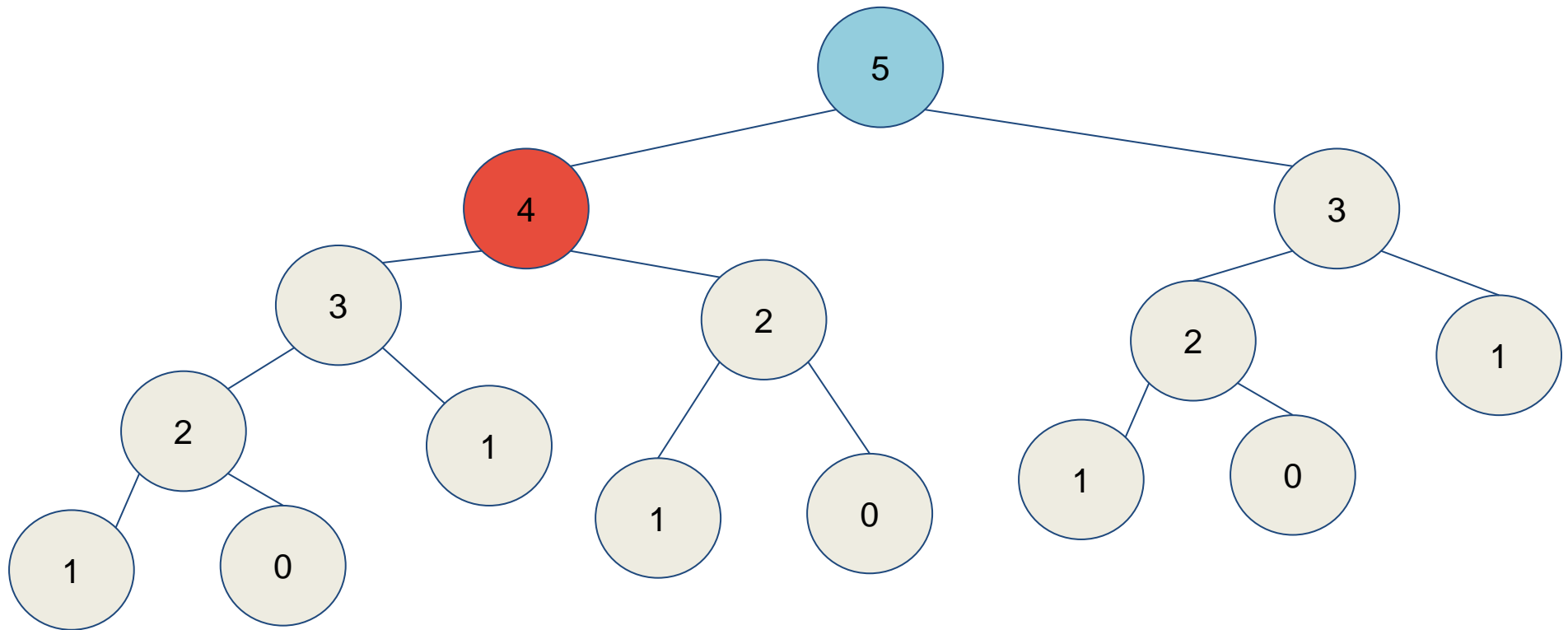
Fibonacci

Ejemplo: Fibonacci(5)



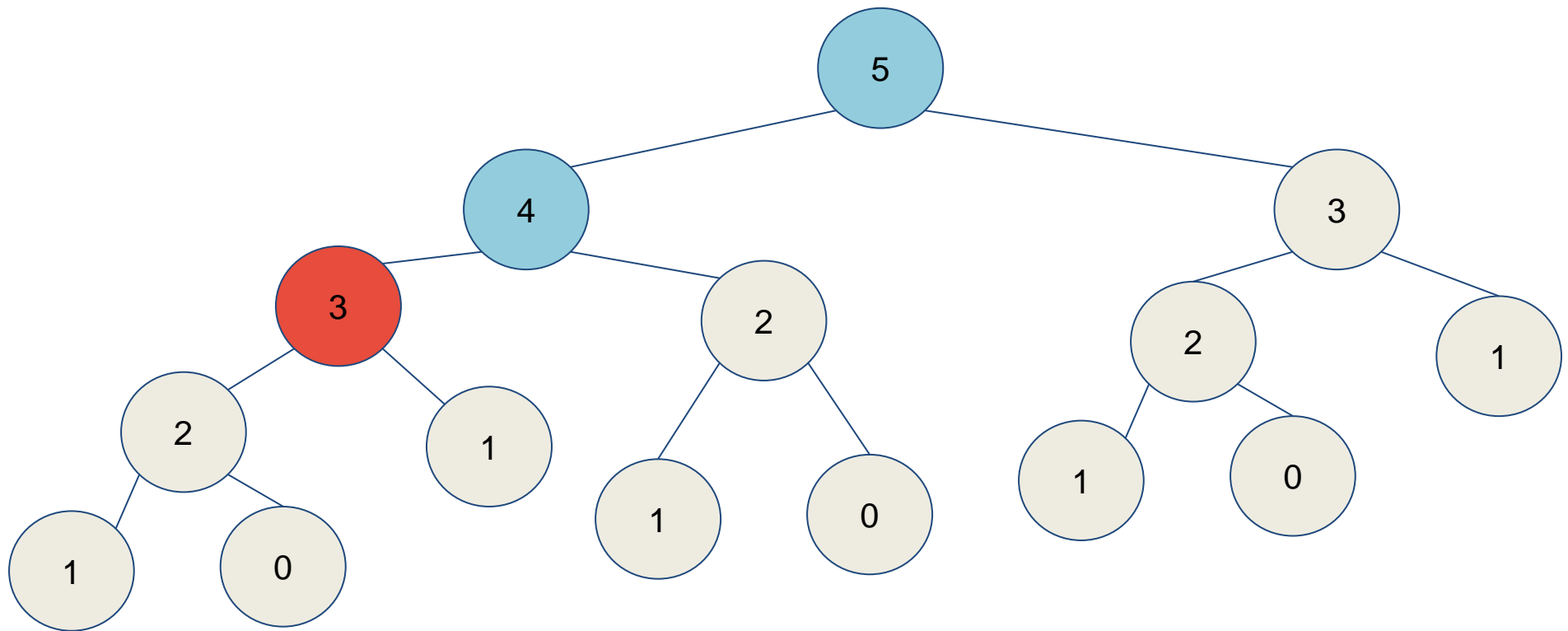
Fibonacci

Ejemplo: Fibonacci(5)



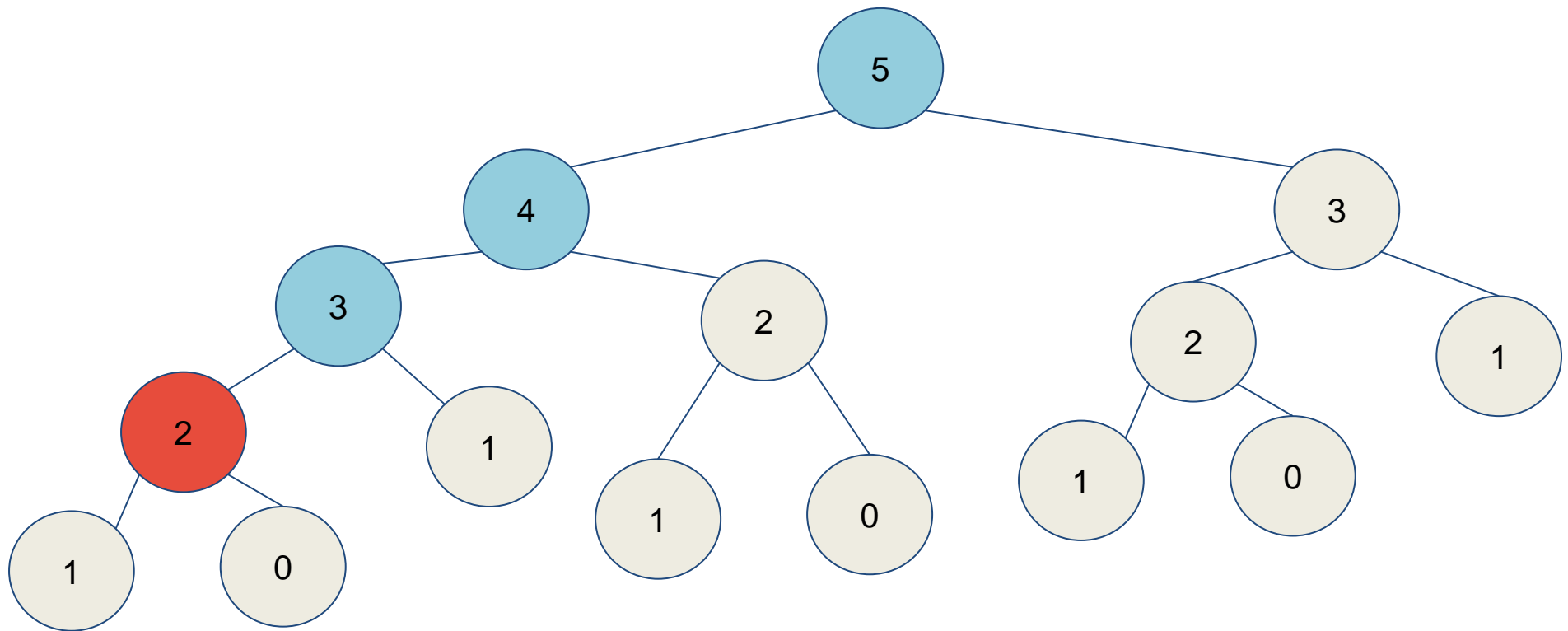
Fibonacci

Ejemplo: Fibonacci(5)



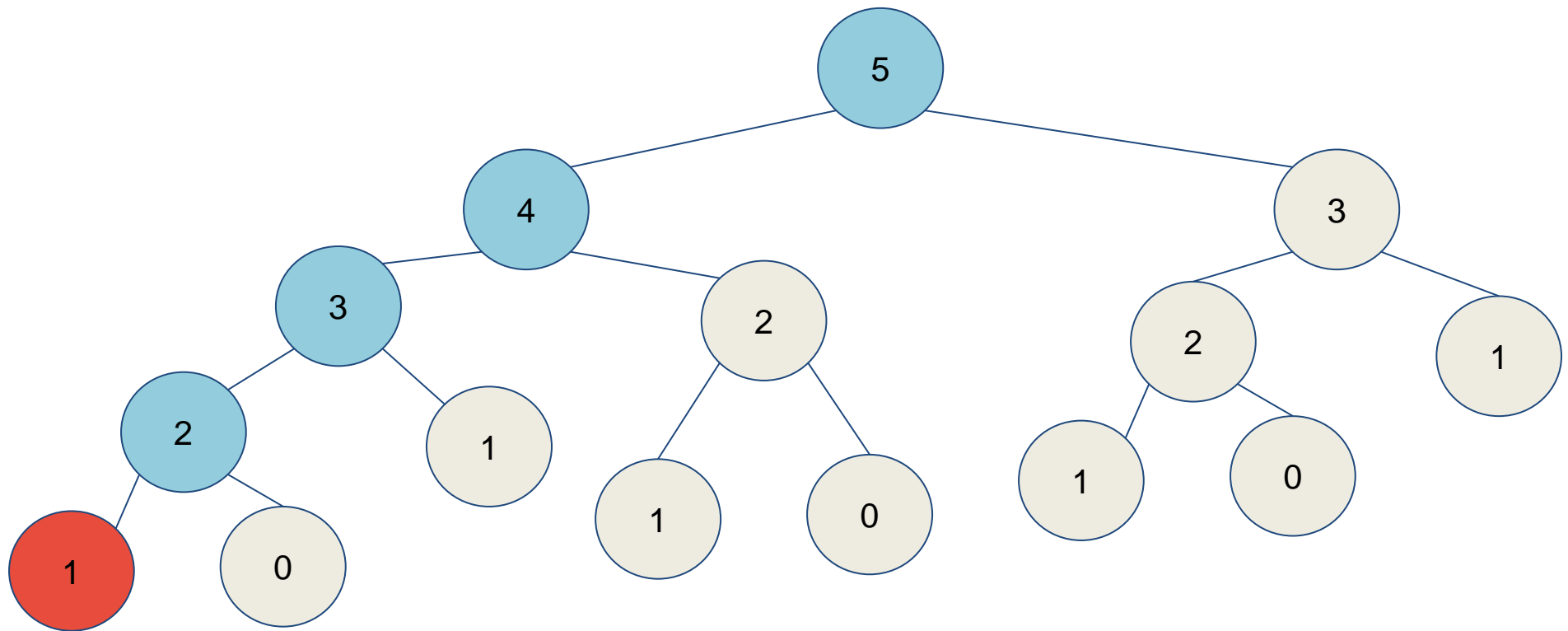
Fibonacci

Ejemplo: Fibonacci(5)



Fibonacci

Ejemplo: Fibonacci(5)

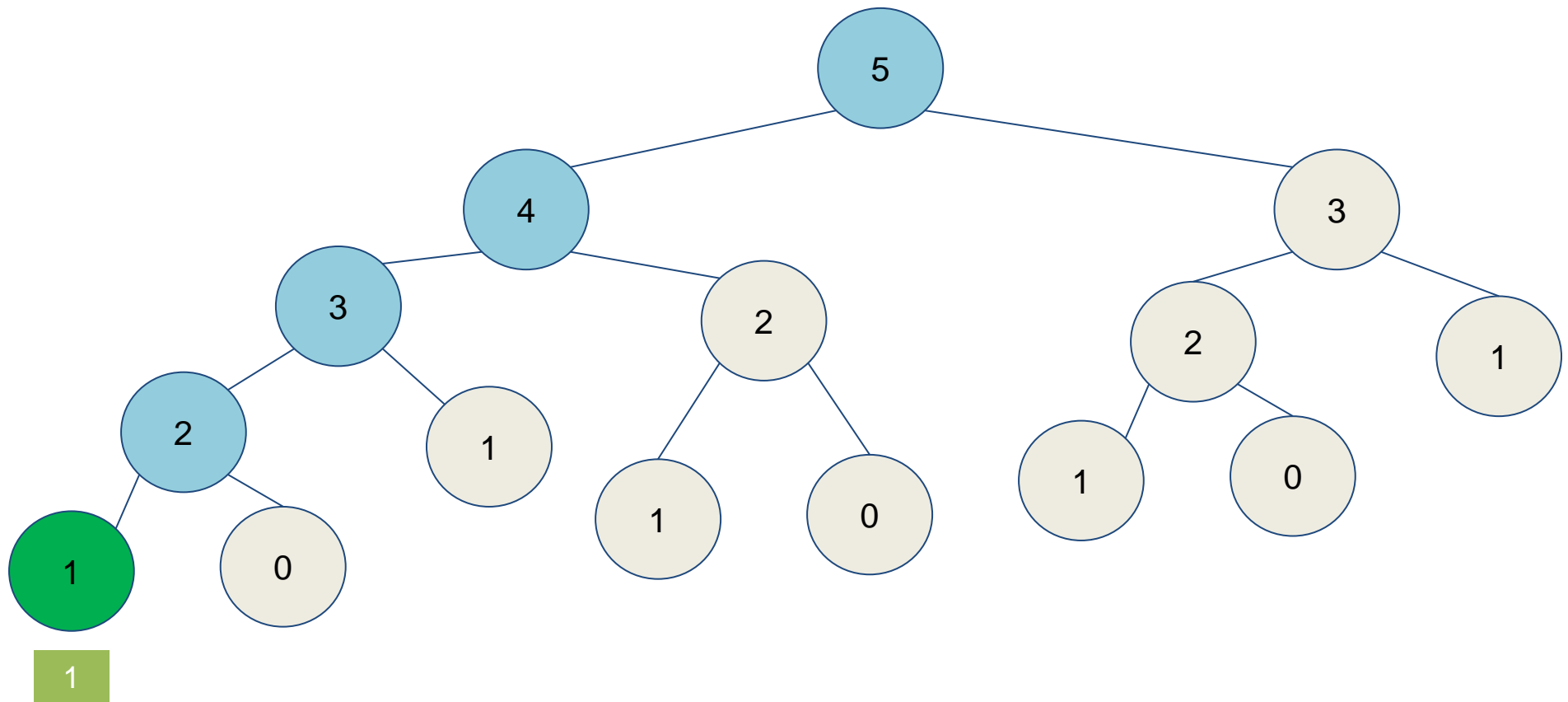


Caso base: $\text{Fibonacci}(1) = 1$



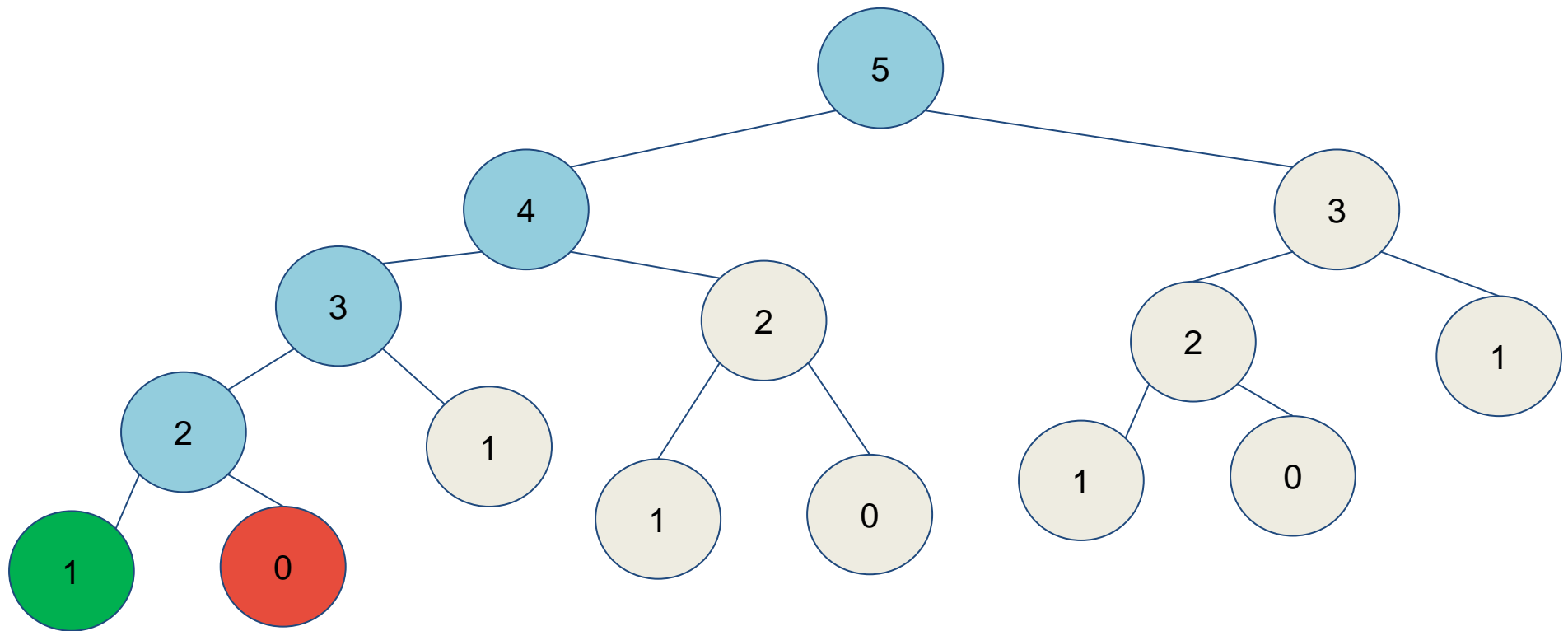
Fibonacci

Ejemplo: Fibonacci(5)



Fibonacci

Ejemplo: Fibonacci(5)



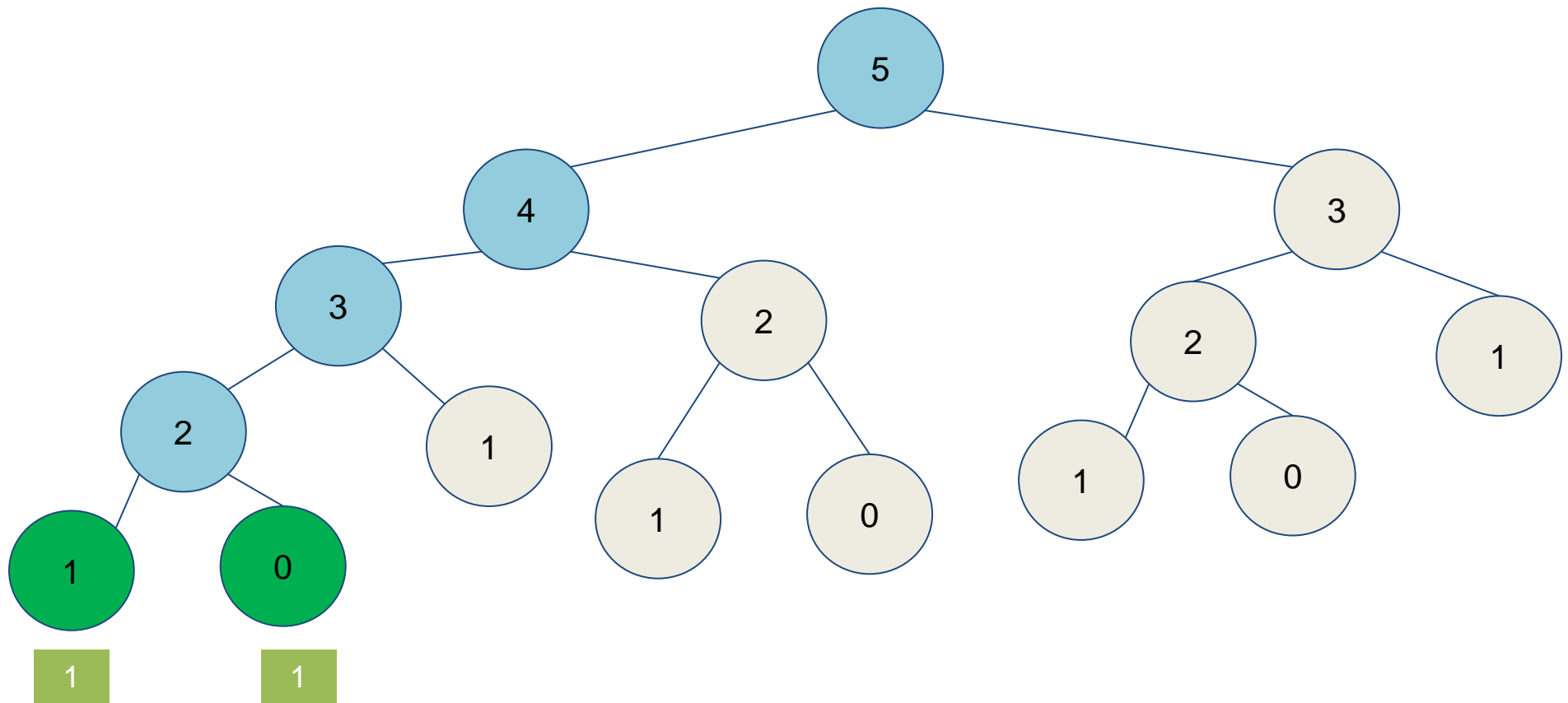
1

Caso base: $\text{Fibonacci}(0) = 1$



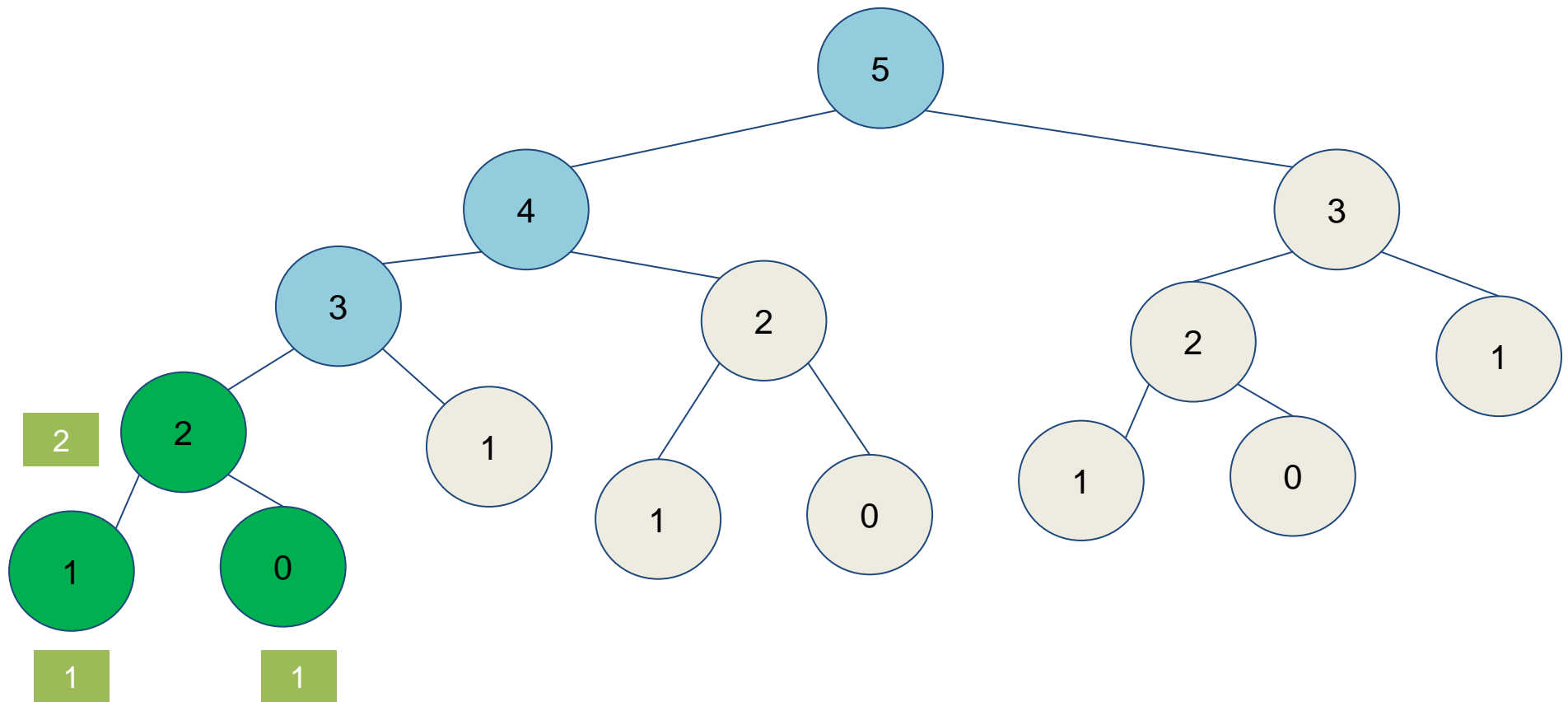
Fibonacci

Ejemplo: Fibonacci(5)



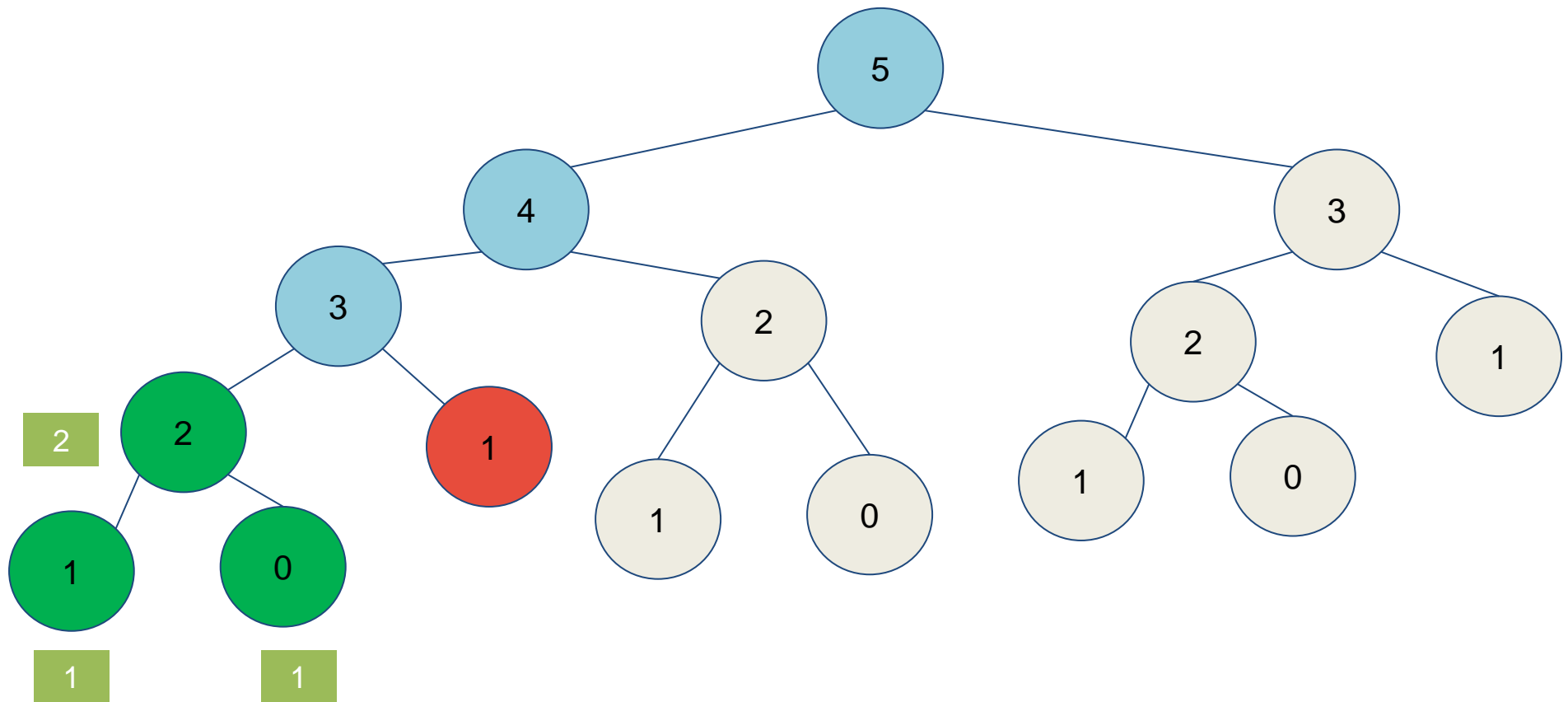
Fibonacci

Ejemplo: Fibonacci(5)



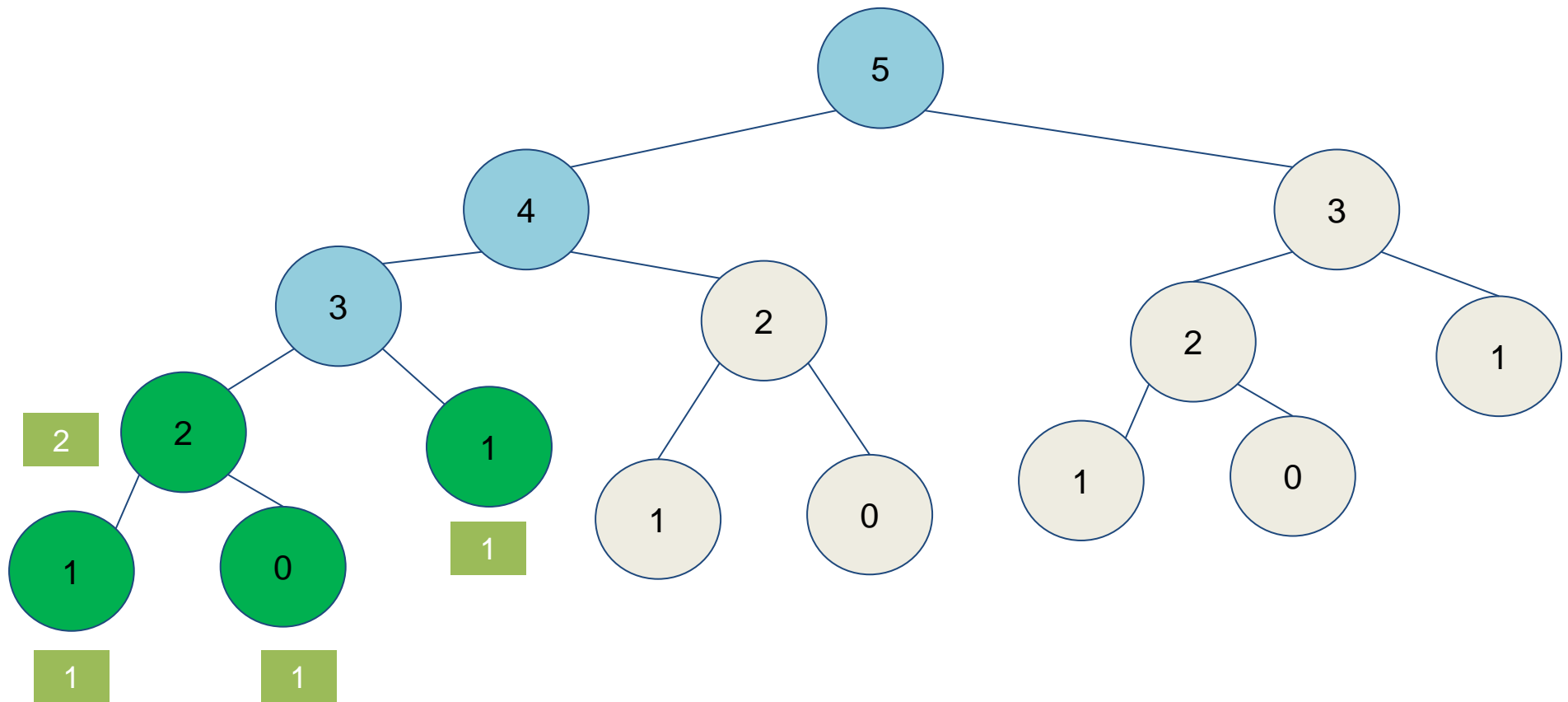
Fibonacci

Ejemplo: Fibonacci(5)



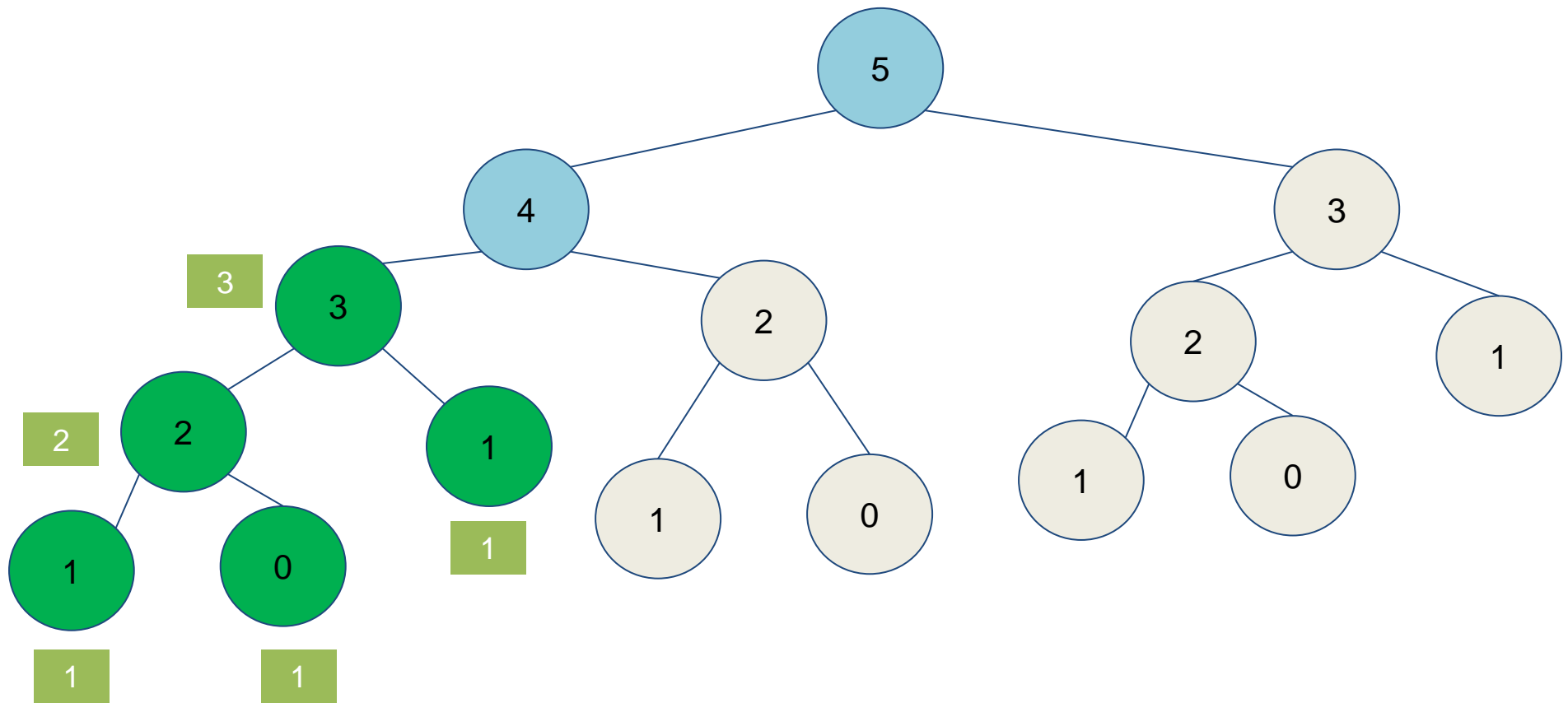
Fibonacci

Ejemplo: Fibonacci(5)



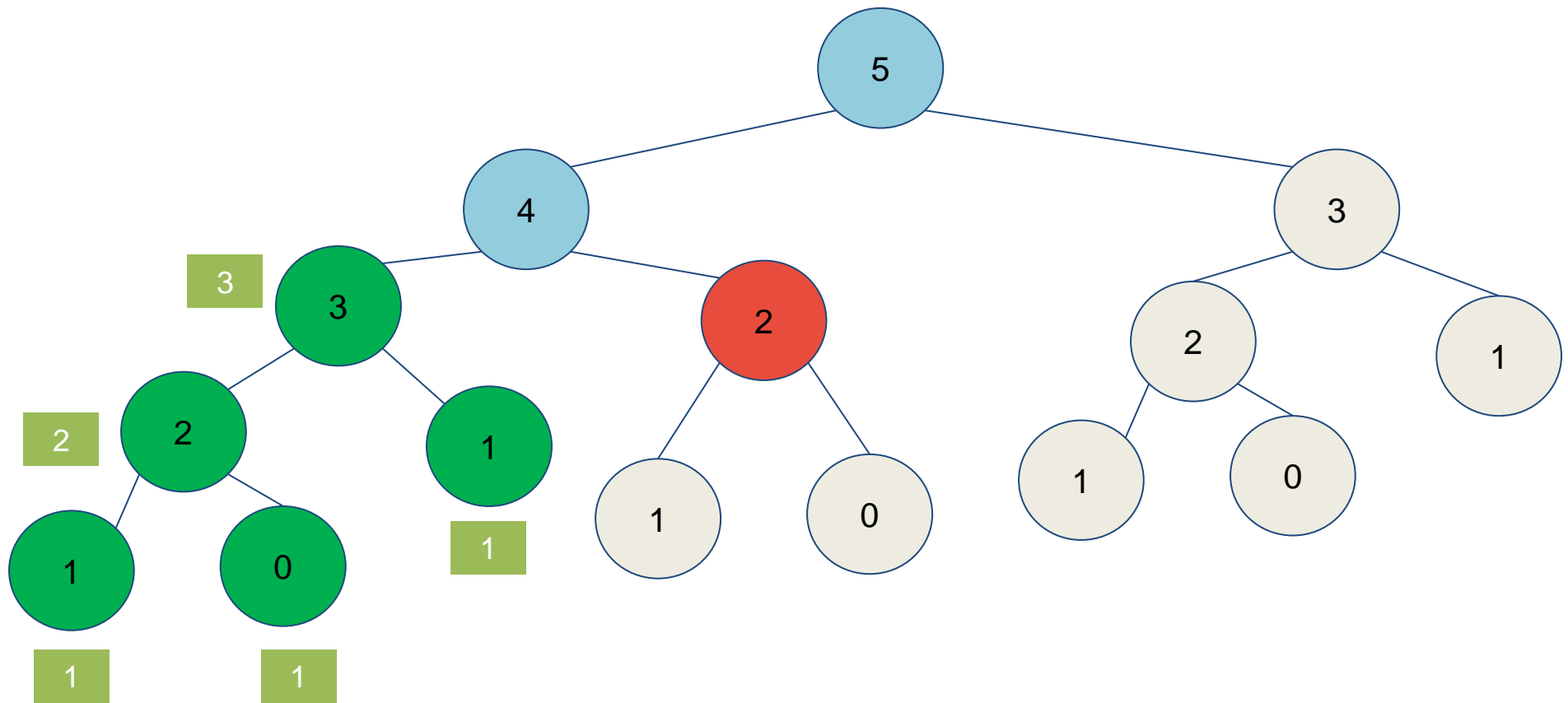
Fibonacci

Ejemplo: Fibonacci(5)



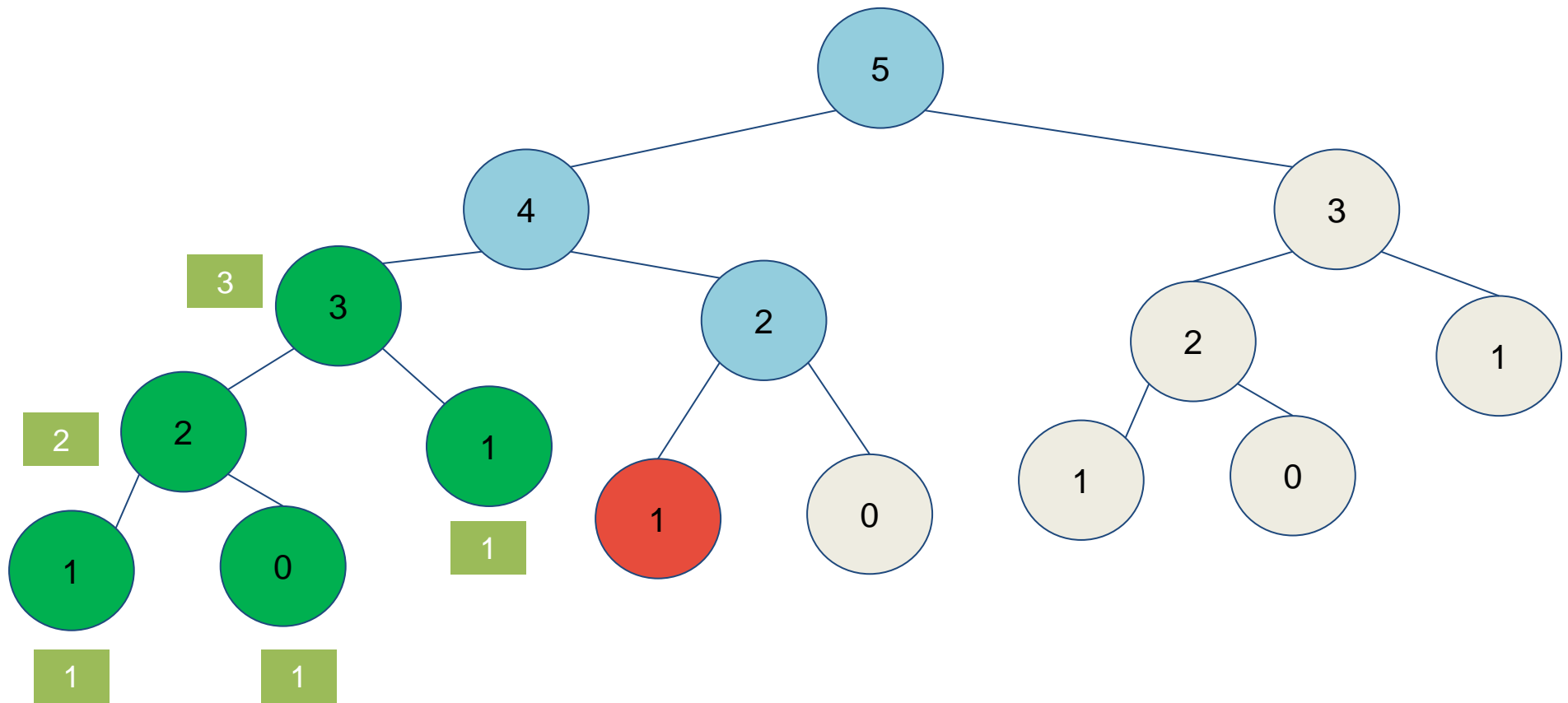
Fibonacci

Ejemplo: Fibonacci(5)



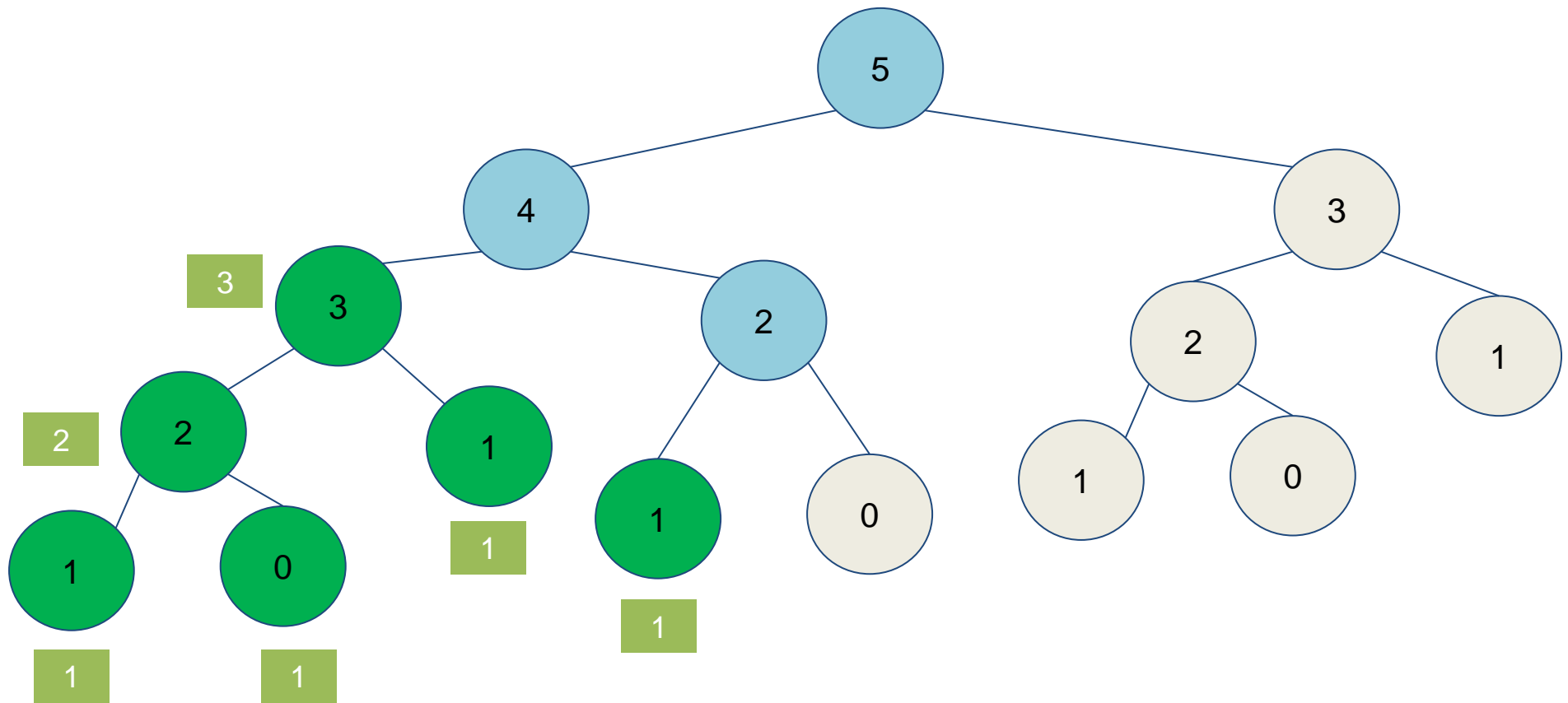
Fibonacci

Ejemplo: Fibonacci(5)



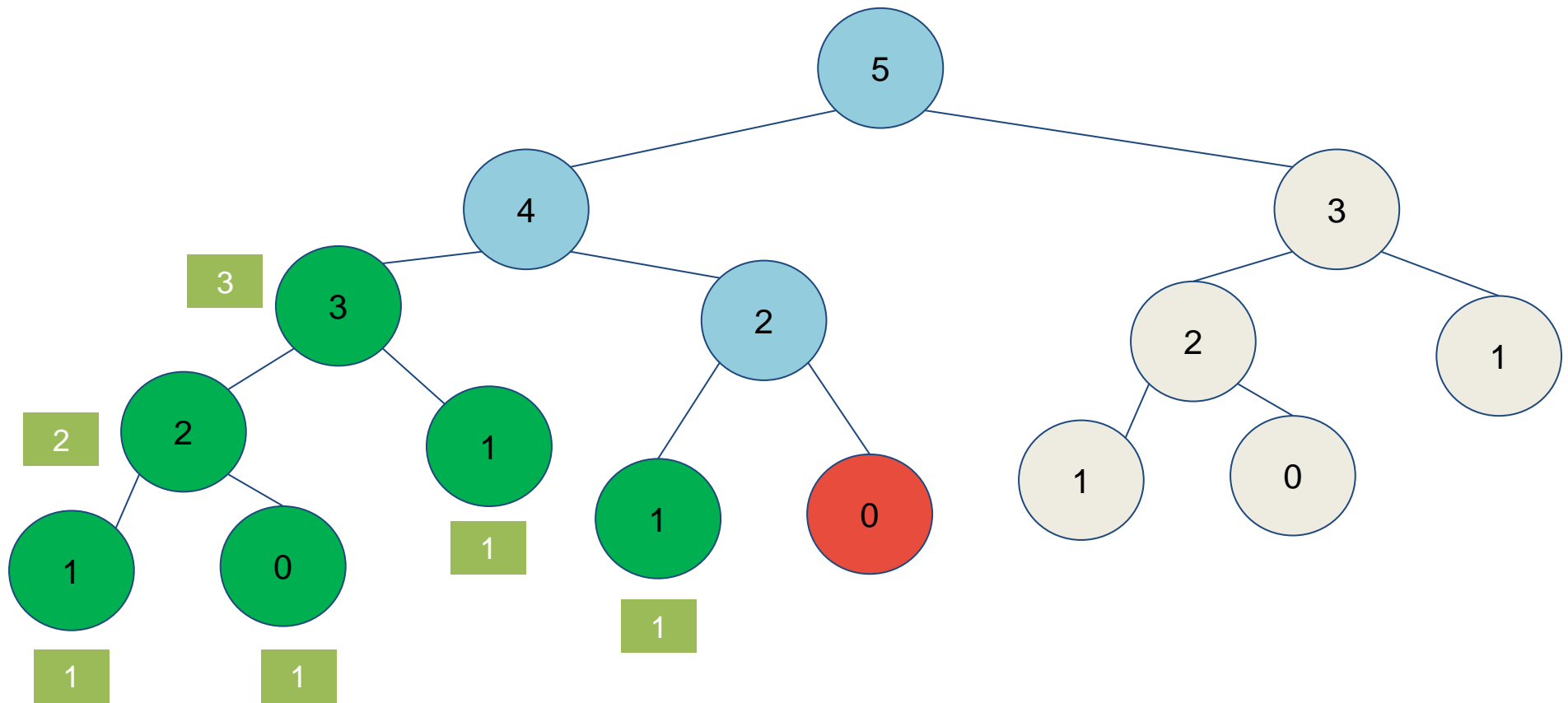
Fibonacci

Ejemplo: Fibonacci(5)



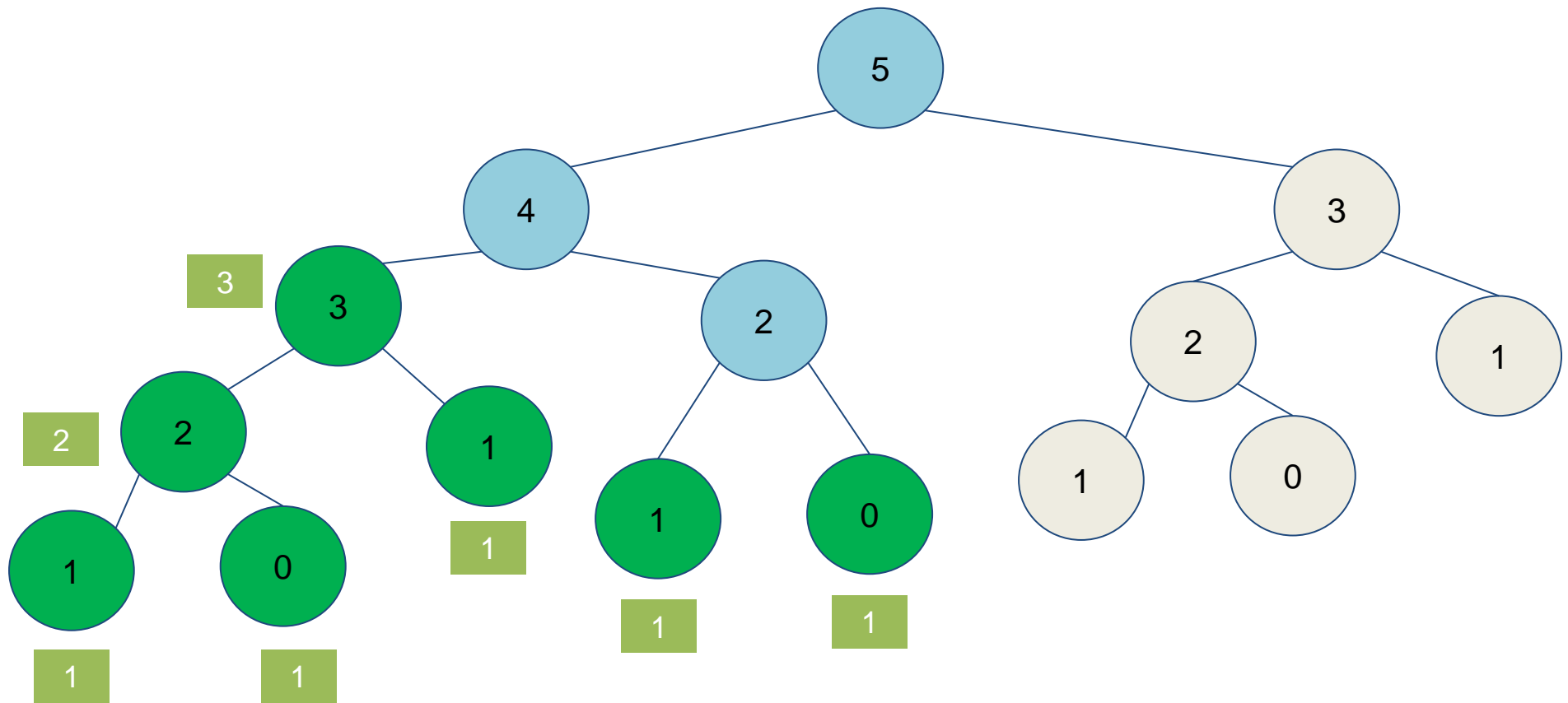
Fibonacci

Ejemplo: Fibonacci(5)



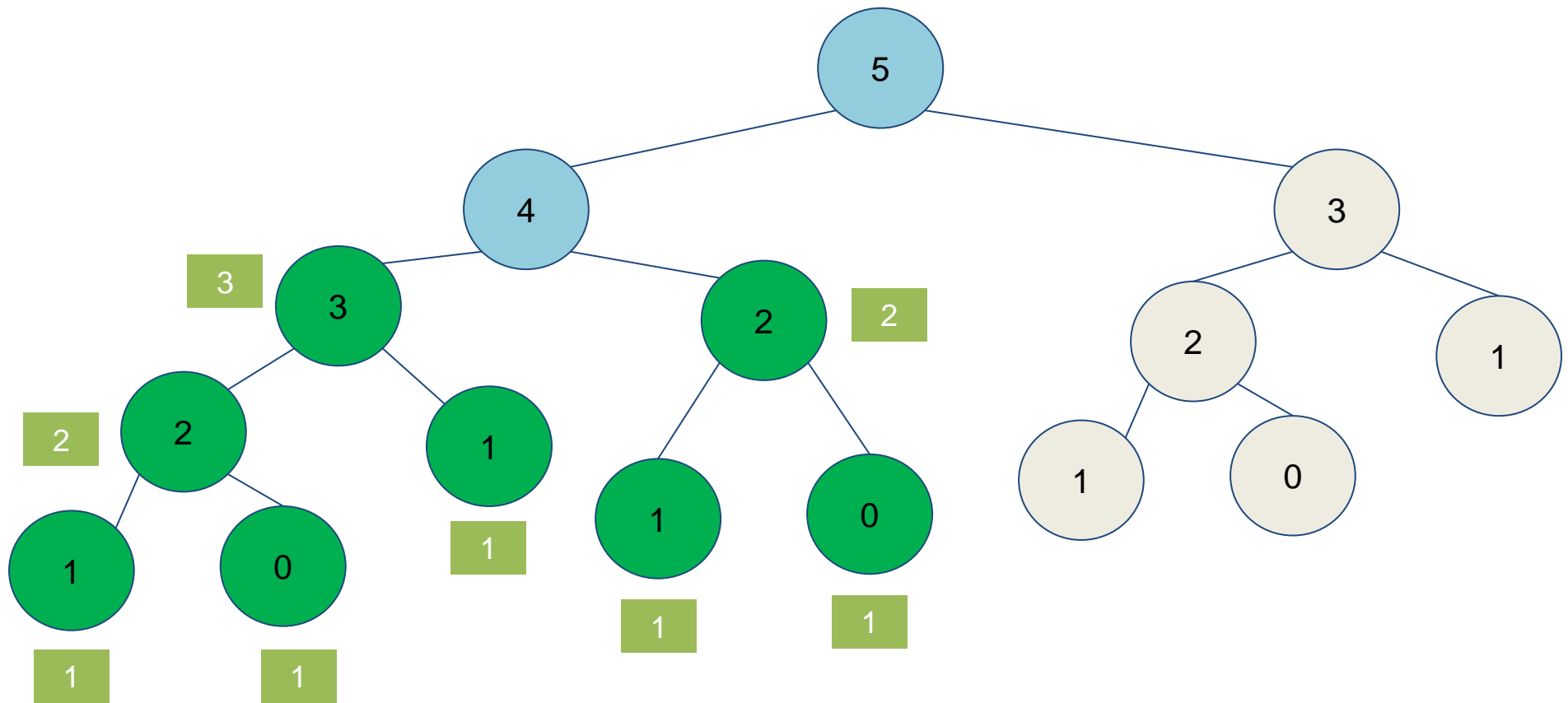
Fibonacci

Ejemplo: Fibonacci(5)



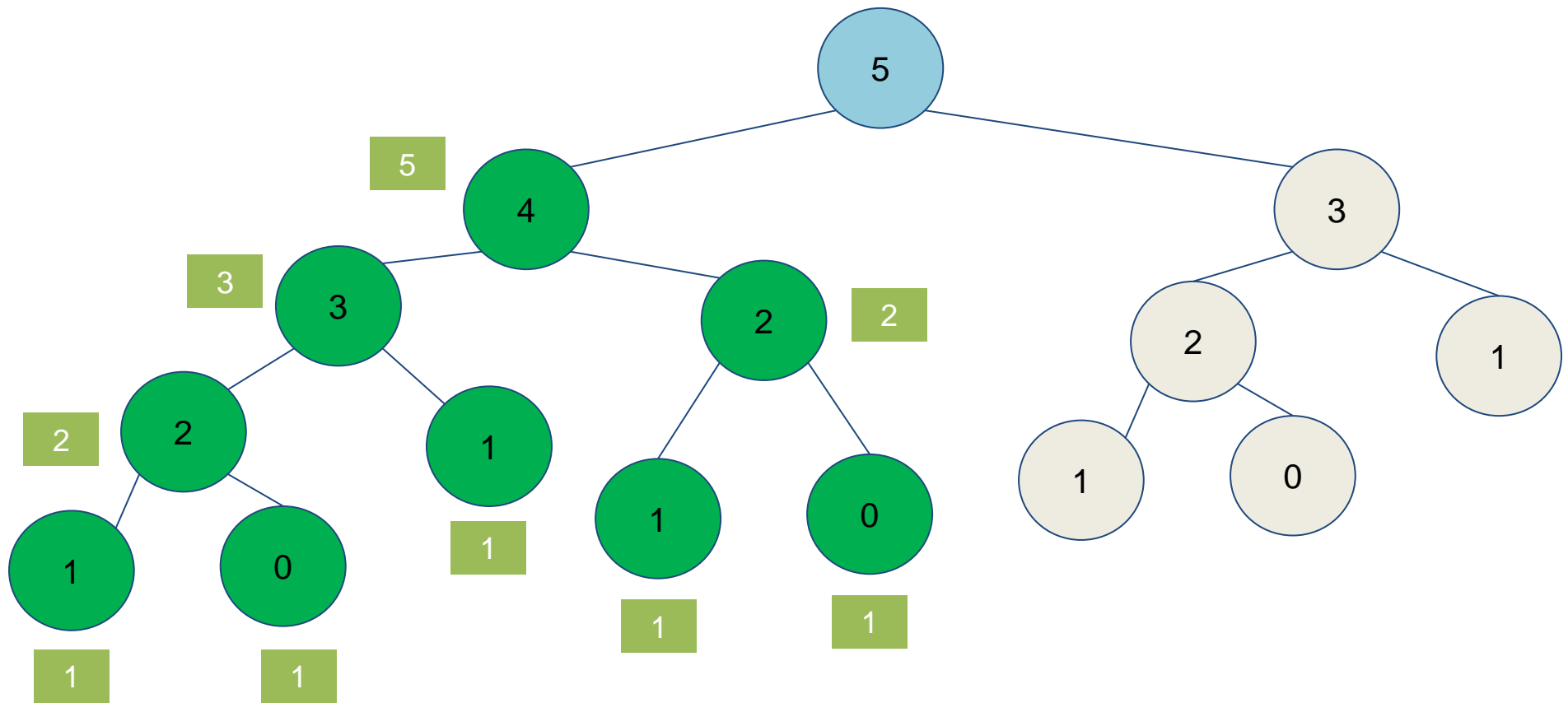
Fibonacci

Ejemplo: Fibonacci(5)



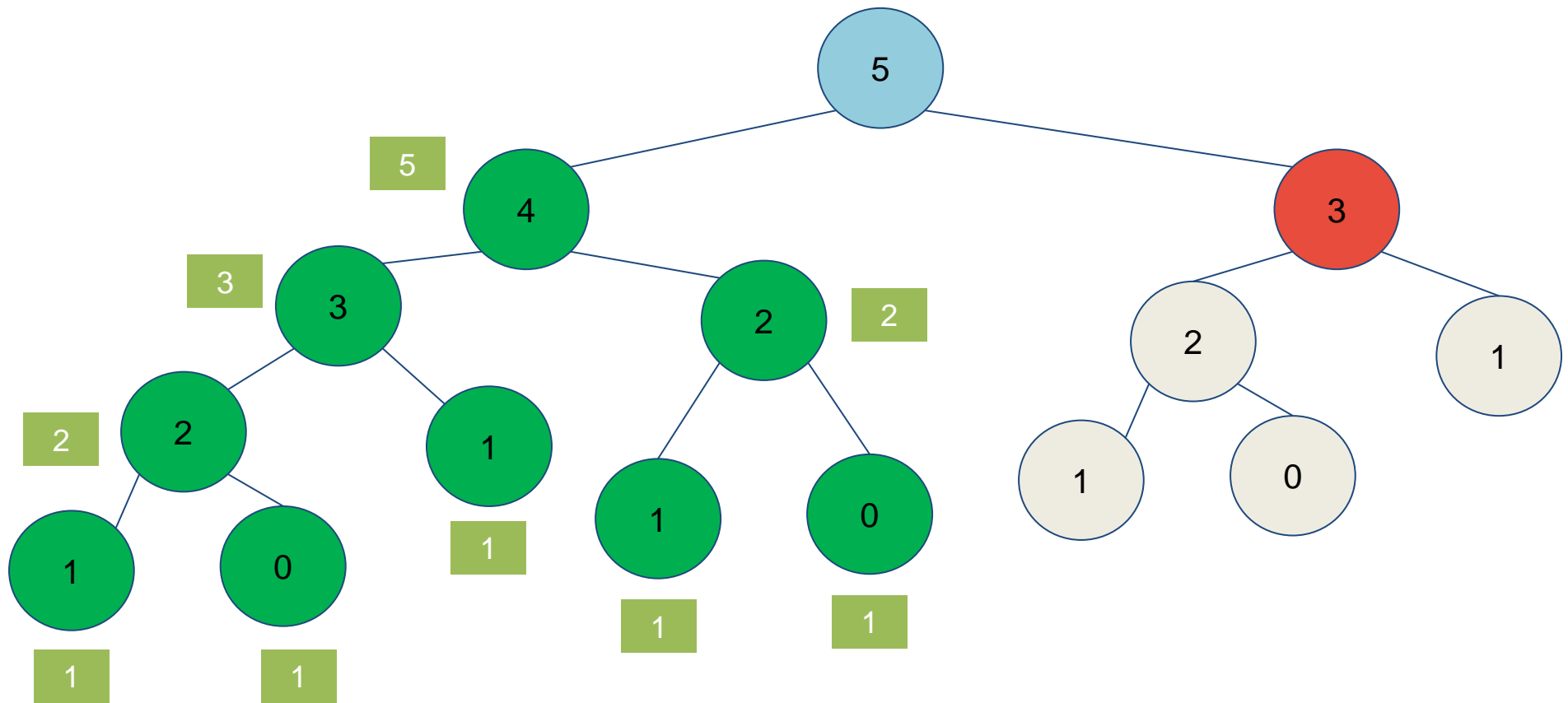
Fibonacci

Ejemplo: Fibonacci(5)



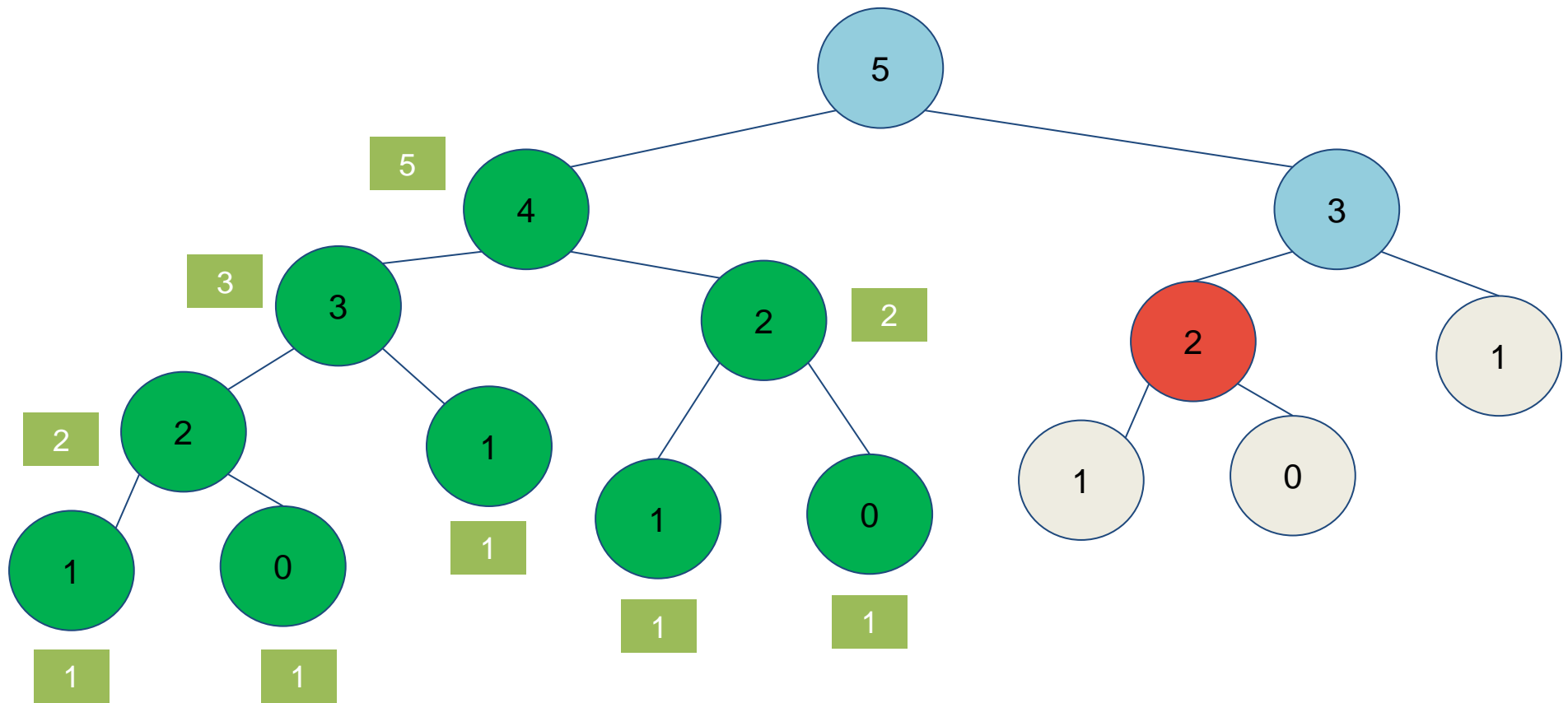
Fibonacci

Ejemplo: Fibonacci(5)



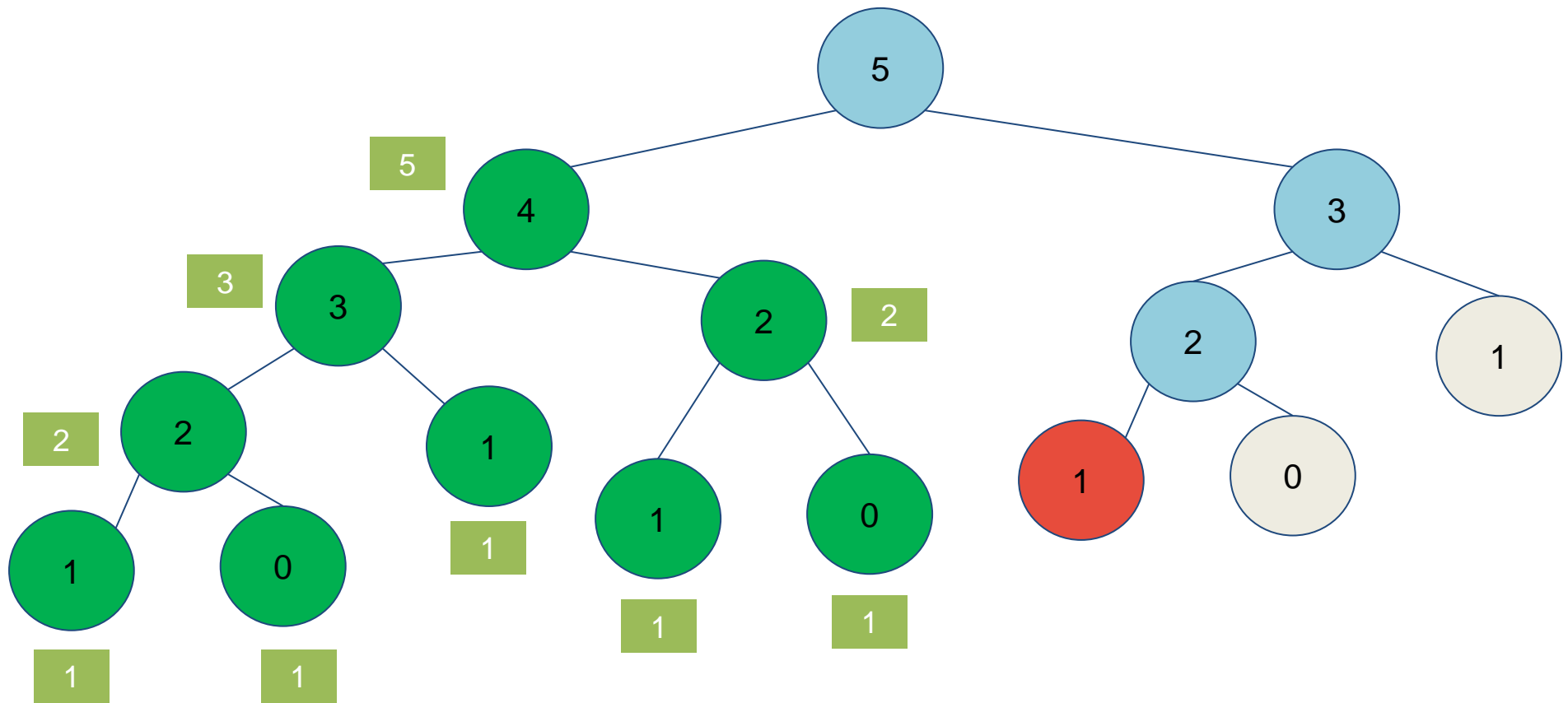
Fibonacci

Ejemplo: Fibonacci(5)



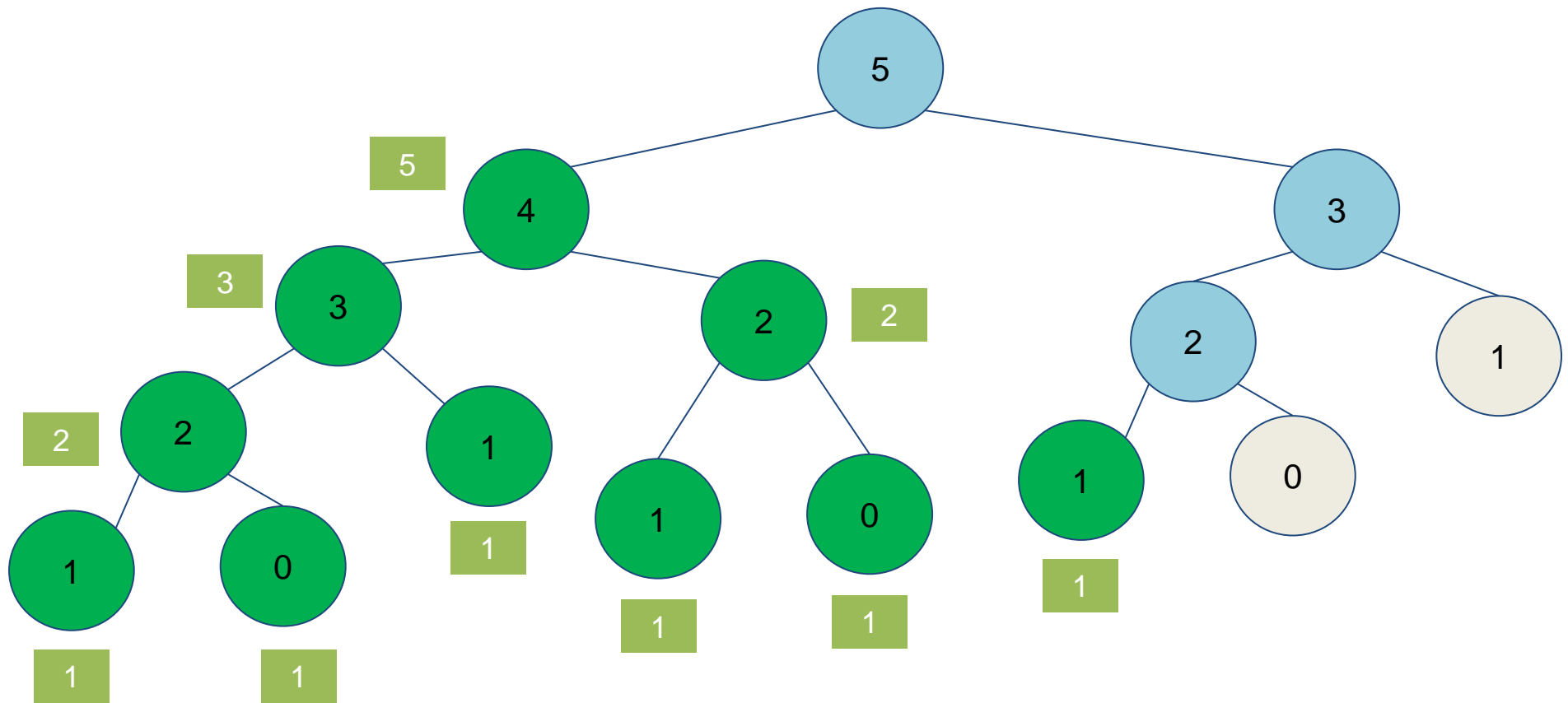
Fibonacci

Ejemplo: Fibonacci(5)



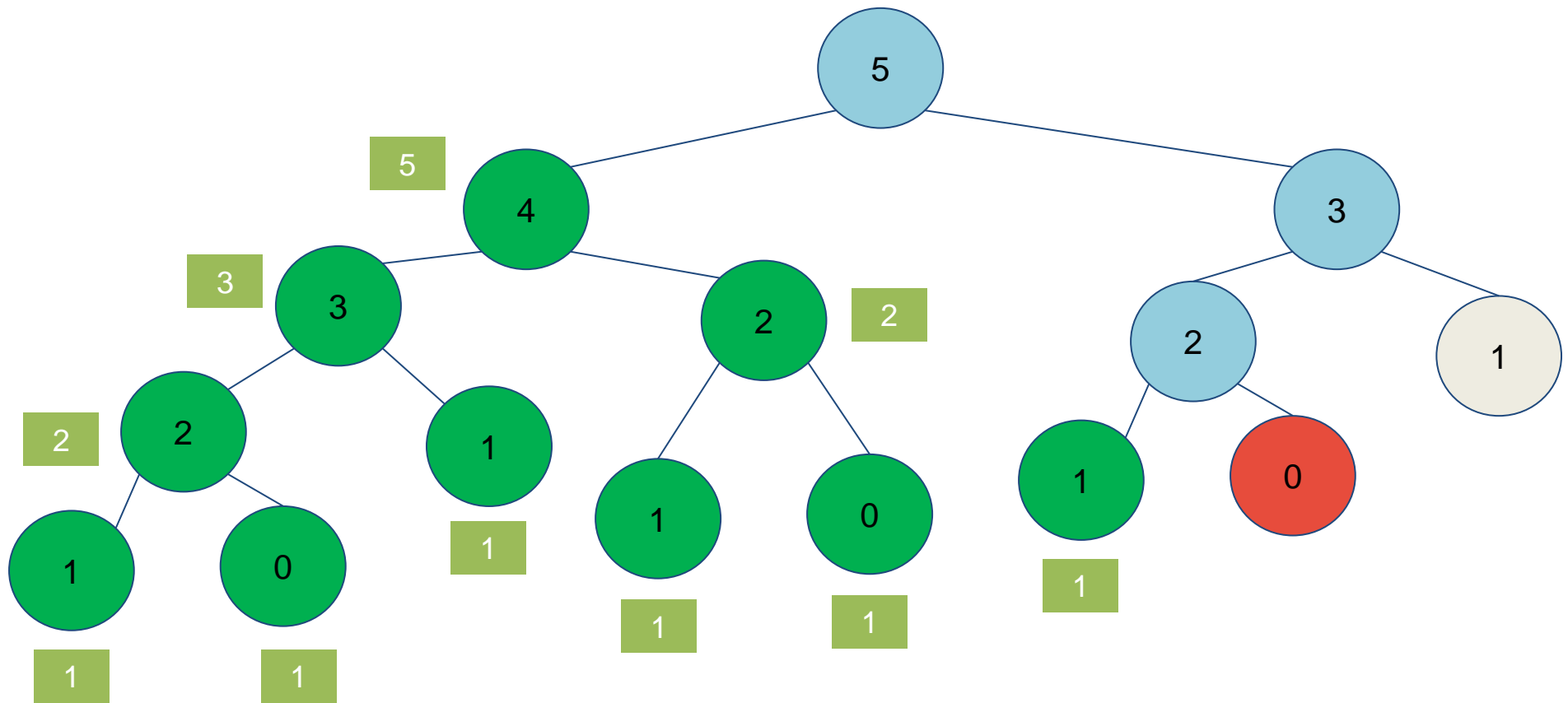
Fibonacci

Ejemplo: Fibonacci(5)



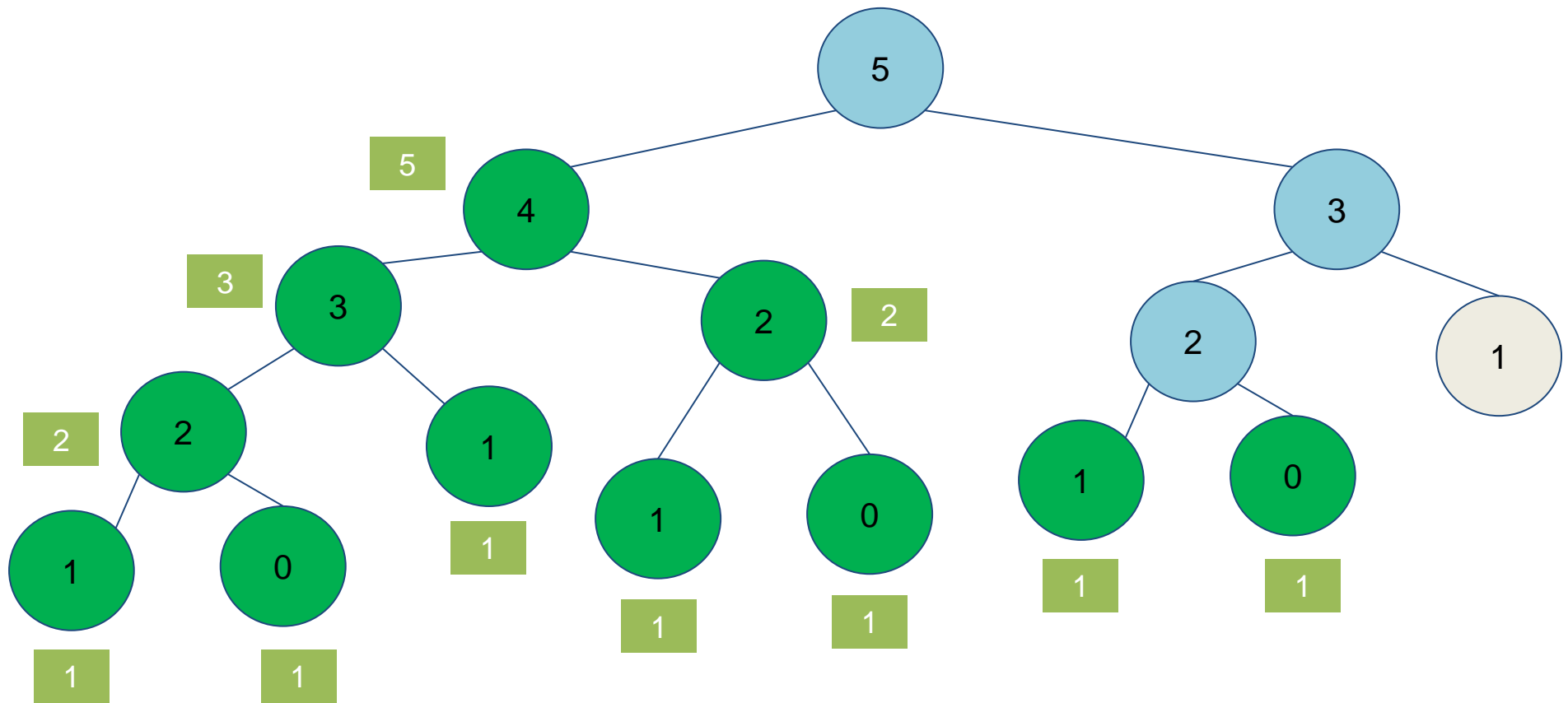
Fibonacci

Ejemplo: Fibonacci(5)



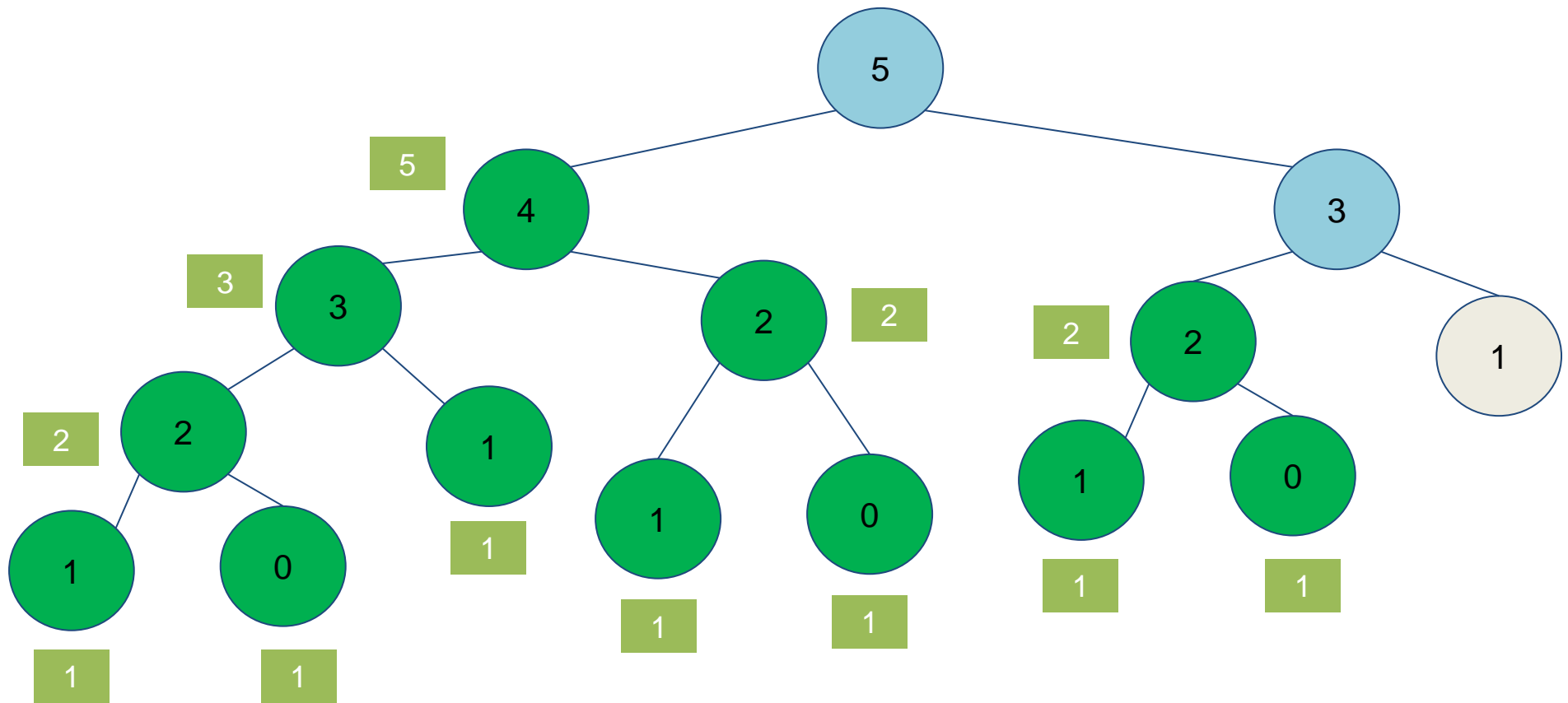
Fibonacci

Ejemplo: Fibonacci(5)



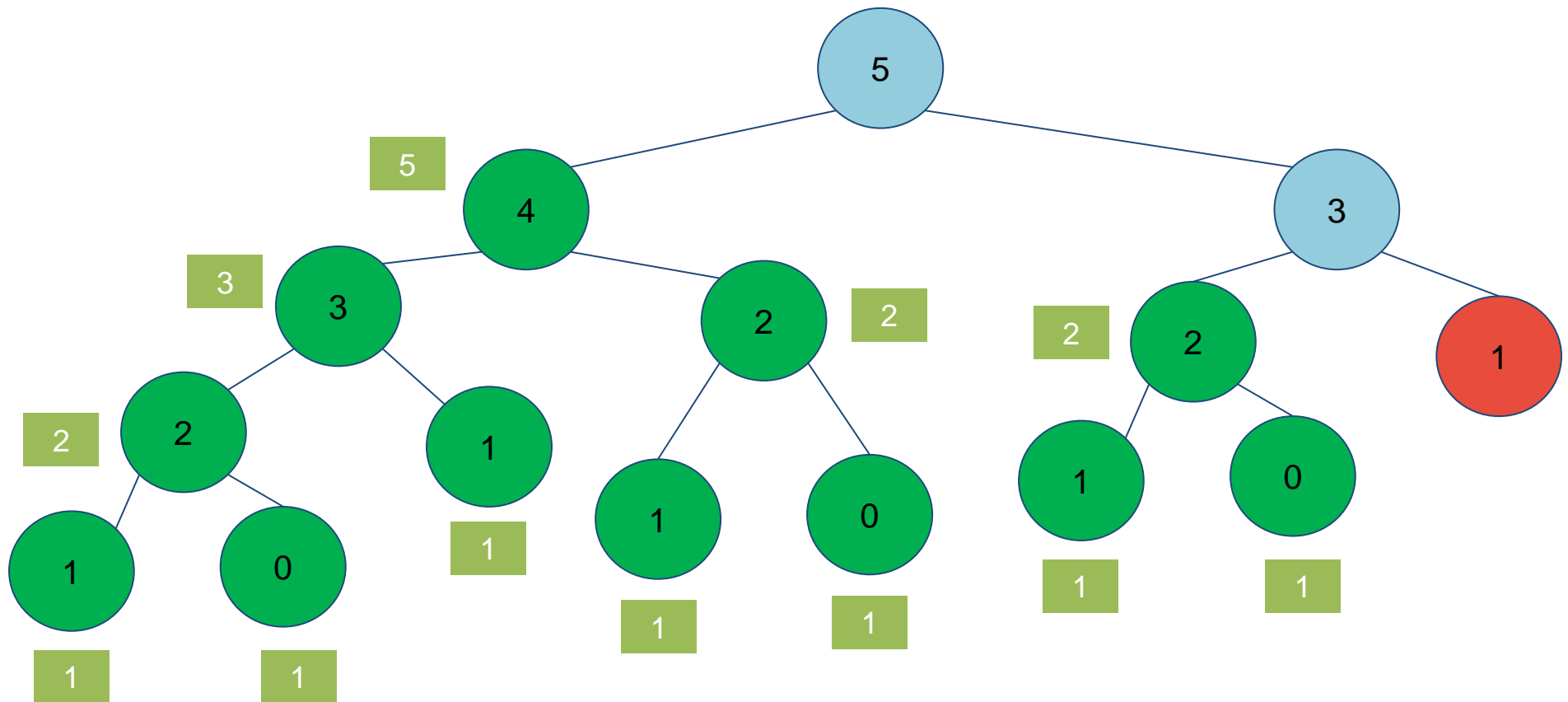
Fibonacci

Ejemplo: Fibonacci(5)



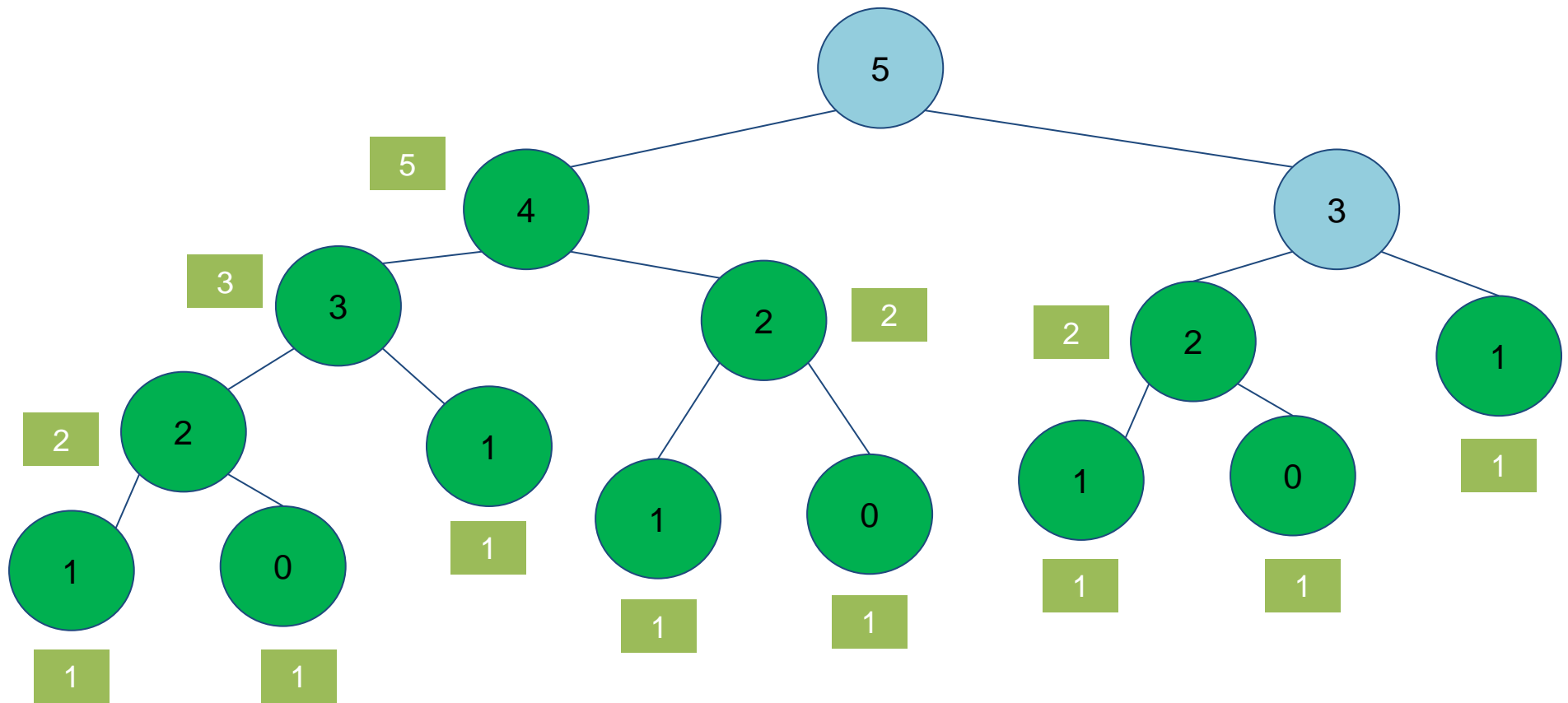
Fibonacci

Ejemplo: Fibonacci(5)



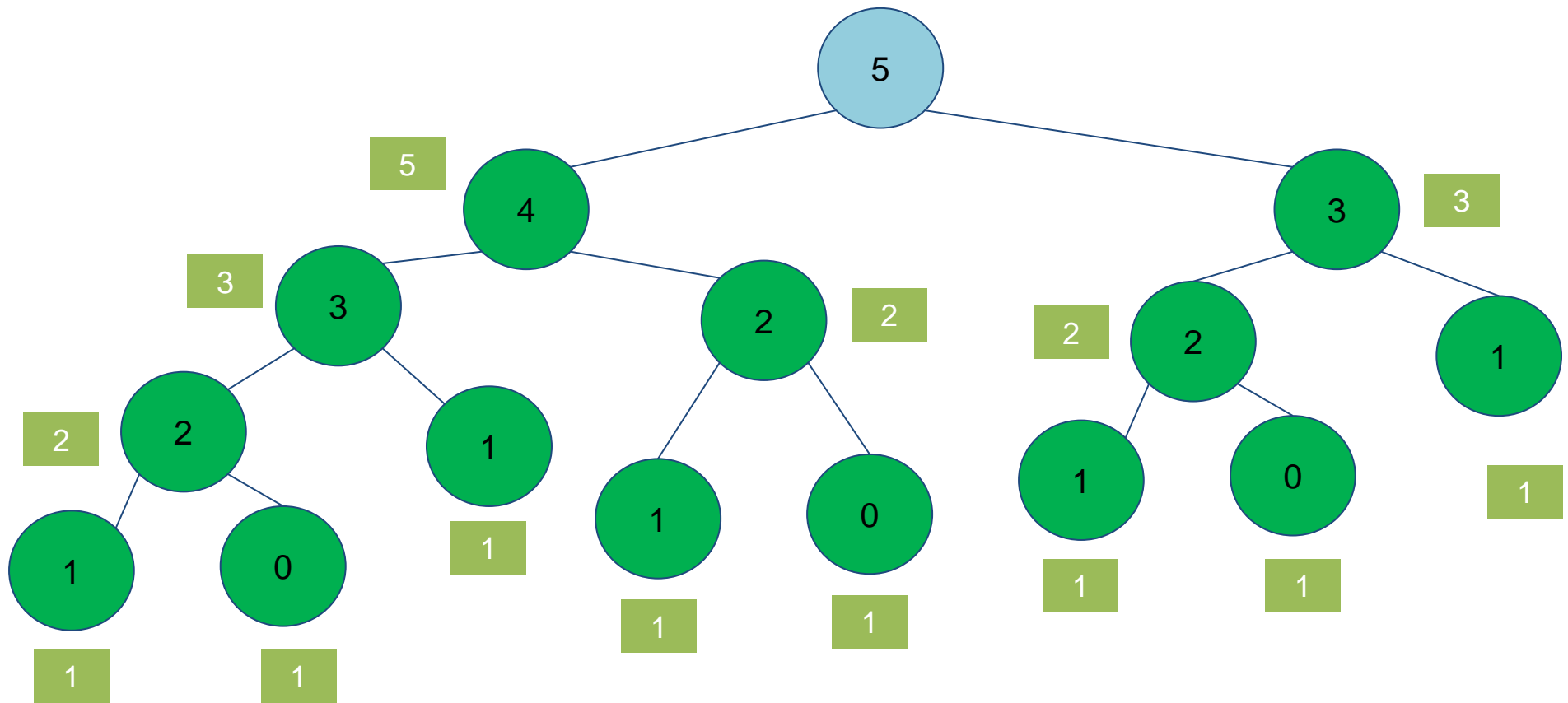
Fibonacci

Ejemplo: Fibonacci(5)



Fibonacci

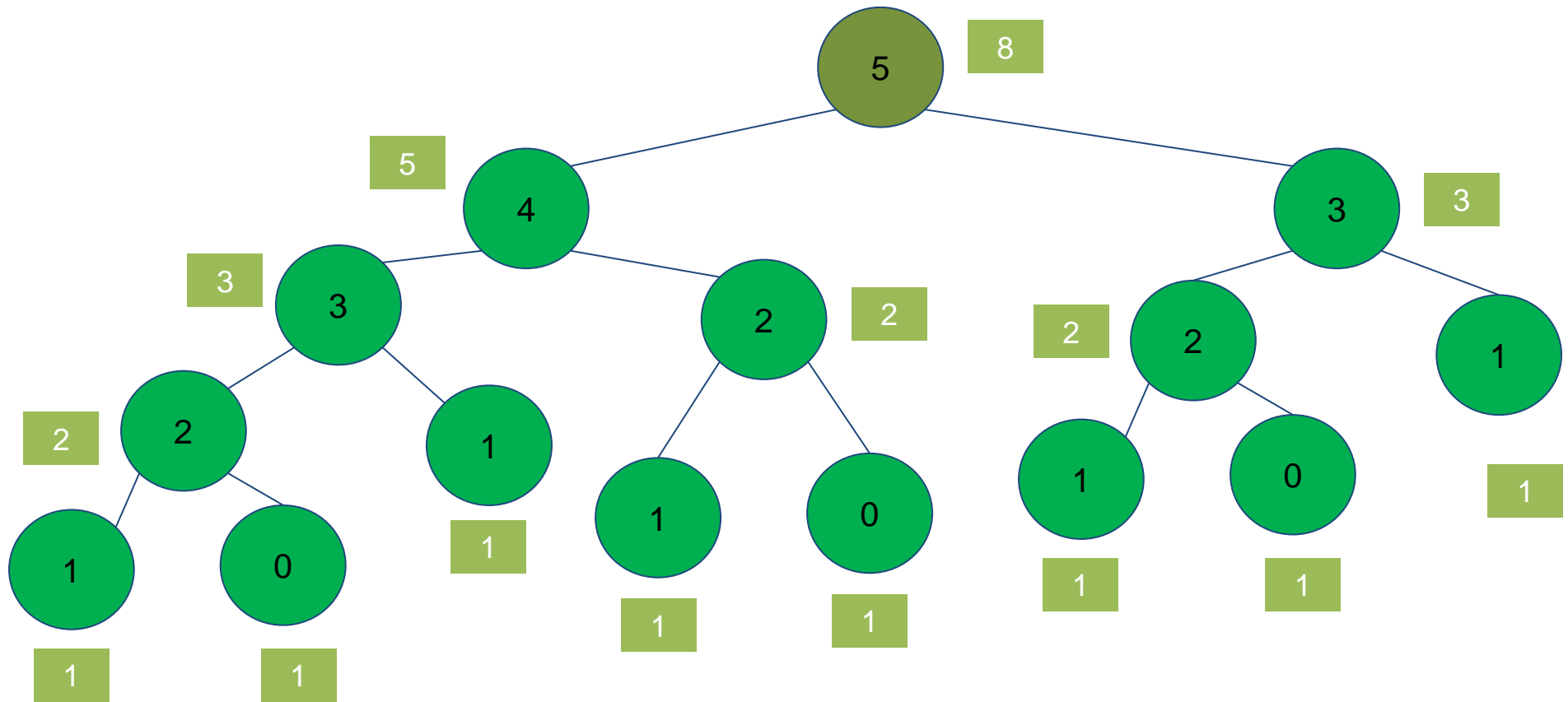
Ejemplo: Fibonacci(5)



Fibonacci

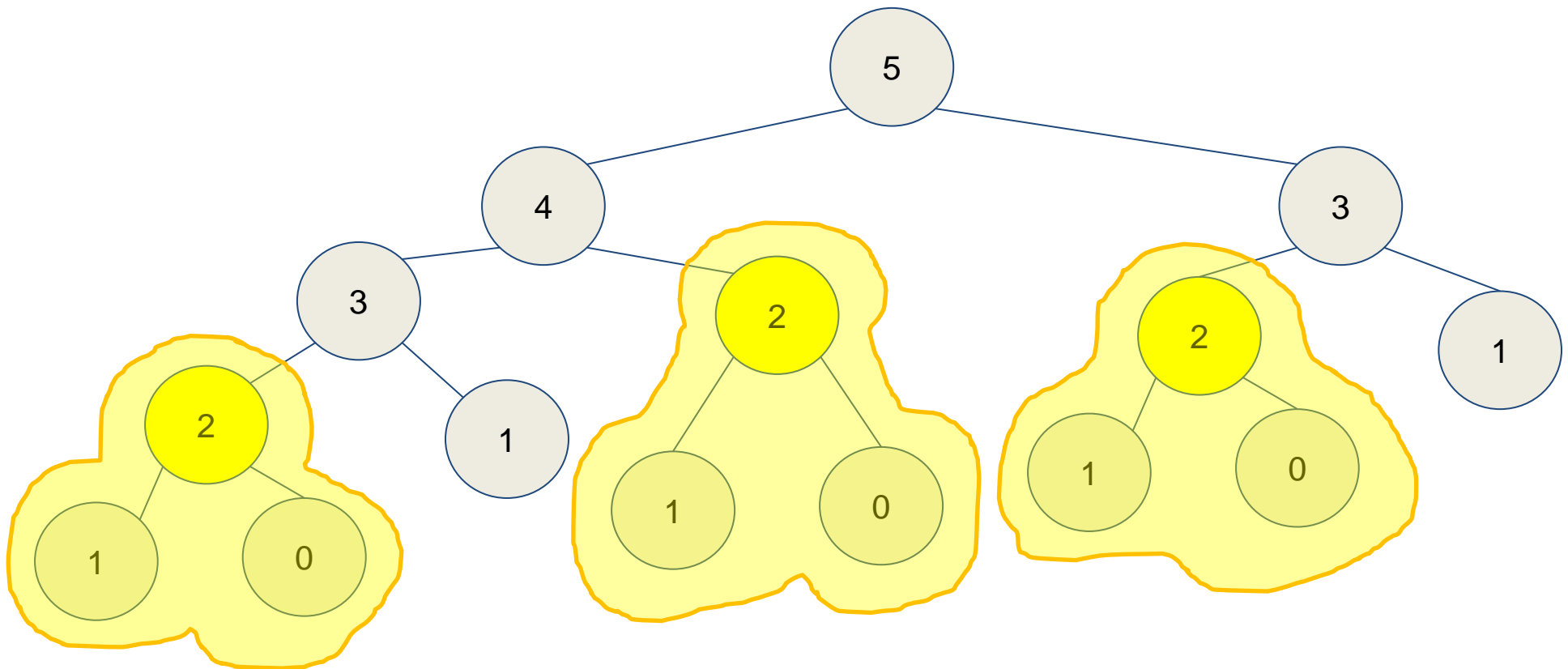
Ejemplo: Fibonacci(5)

Complejidad **exponencial**



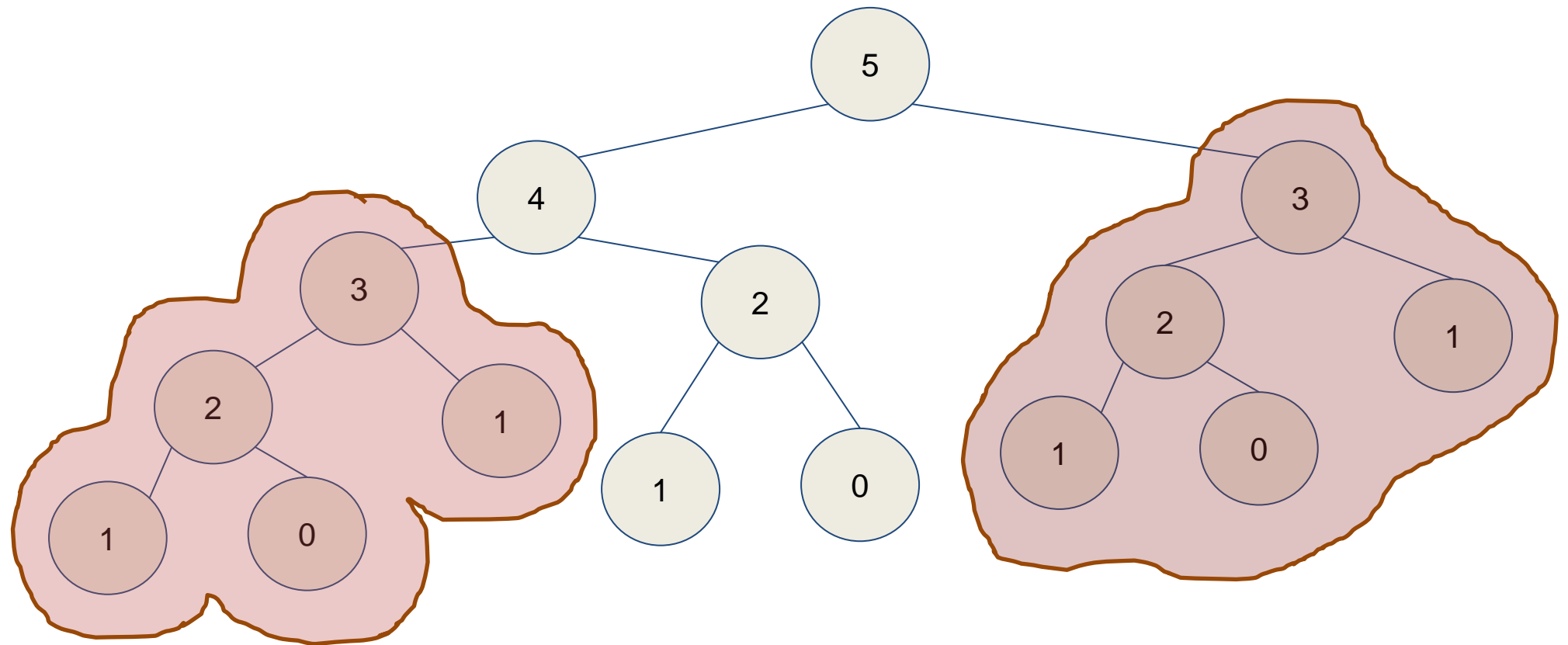
Fibonacci

Ejemplo: Fibonacci(5)




Fibonacci

Ejemplo: Fibonacci(5)



Estados repetidos!

- Calculamos **varias veces** el mismo número de Fibonacci

¿Qué podemos hacer? 

- **Memorizar** lo que ya hemos calculado → **Programación Dinámica**



ÍNDICE

- Motivación
- **¿Qué es?**
- Características
- Top Down
- Bottom Up
- Problemas clásicos
- Más allá...



¿Qué es la Programación Dinámica?

- Paradigmas de la Programación:
 - >> Fuerza Bruta
 - >> Greedy
 - >> Divide y Vencerás
 - >> **Programación Dinámica**
- **Memorizar** resultados calculados para reutilizarlos
- Problemas clásicos: maximizar, minimizar, contar caminos, ...
- 20% de los problemas de un concurso son **DP**



ÍNDICE

- Motivación
- ¿Qué es?
- **Características**
- Top Down
- Bottom Up
- Problemas clásicos
- Más allá...



Características

1. Estado
2. Transición
3. Memoria
4. Casos base



Características

1. Estado

- Situación del problema en la que te encuentras (y que se puede repetir)
- Está determinado por un conjunto de parámetros
- Ejemplo (Fibonacci): número de Fibonacci a calcular

2. Transición

3. Memoria

4. Casos base



Características

1. Estado
2. Transición
 - Paso de un estado a otro
 - Ejemplo (Fibonacci):
$$\text{Fibo}(n) = \text{Fibo}(n-1) + \text{Fibo}(n-2)$$
3. Memoria
4. Casos base



Características

1. Estado

2. Transición

3. Memoria

- Estructura para almacenar los resultados previamente calculados

- Se accede a ella a través de los parámetros que caracterizan el estado

- Ejemplo (Fibonacci):

1	1	2	3	5
0	1	2	3	4

4. Casos base

*en ocasiones podrá ser un mapa u otra estructura



Características

1. Estado
2. Transición
3. Memoria
4. Casos base
 - Estados de los cuales conocemos la solución de antemano
 - Ejemplo (Fibonacci): $\text{Fibonacci}(0) = 1$
 $\text{Fibonacci}(1) = 1$



Primeros pasos para construir un DP

Determinar las características del problema:

- Las características (parámetros) que determinan totalmente un **estado**
- Los **casos bases** para los cuales conocemos la solución
- Las **transiciones** entre estados
- La estructura de **memorización** necesaria



¿Complejidad de un algoritmo de DP?

$O(n^{\circ} \text{ estados} * n^{\circ} \text{ transiciones})$



Ejemplo Fibonacci: $O(n * 2) = \mathbf{O(n)}$



ÍNDICE

- Motivación
- ¿Qué es?
- Características
- **Top Down**
- Bottom Up
- Problemas clásicos
- Más allá...



Top – Down (memoización)

- Llegar desde el problema general hasta los **casos base**
- Enfoque **RECURSIVO**
- DP recursivo = Recursión con Fuerza Bruta + Memo



Recursión -> DP

- Esquema general de la recursión



- DP

```
resolver ( parámetros ) {  
  - caso base? -> solución ya conocida  
  - en otro caso:  
    - transiciones hacia otros  
      estados  
}
```

```
resolver ( parámetros ) {  
  - caso base? -> solución ya conocida  
  - resultado ya calculado? -> Memo  
  - en otro caso:  
    - transiciones hacia otros estados  
    - almacenar en la Memo y devolver el  
      resultado  
}
```



Ejemplo: Fibonacci

- Esquema general de la recursión



- DP

```
Fibonacci ( n ) {  
    si n==0 o n==1: devolver 1;  
    si no: devolver Fibonacci(n-1) + Fibonacci(n-2)  
}
```

```
Fibonacci ( n ) {  
    si n==0 o n==1: devolver 1;  
    si calculado Fibonacci(n): devolver Memo(n)  
    si no: calcular Fibonacci(n-1) + Fibonacci(n-2)  
           almacenar Memo(n)  
           devolver Memo(n)  
}
```



Primer problema

- Escaleranos [\(aer 554\)](#)



Primer problema: características

- Estado: escalón en el que te encuentras
- Transiciones: escalones que puedes saltar
- Casos base: -has llegado al último escalón ($i==n$)
-te has pasado de escalones ($i>n$)
- Memo: array 1-dimensional



ÍNDICE

- Motivación
- ¿Qué es?
- Características
- Top Down
- **Bottom Up**
- Problemas clásicos
- Más allá...



Bottom – Up (tabulación)

- Menos intuitivo



- Enfoque **ITERATIVO**
- Llegar desde los casos base al problema general
- ¿Cómo?
 - 1 – rellenar la memo de los casos base
 - 2 – rellenar los estados que se pueden calcular a partir de los resueltos
 - 3 – repetir 2 hasta llegar al caso deseado



Ejemplo 1: Factorial

- Calcular todos los números factoriales de 1 a 25:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25



Ejemplo 1: Factorial

Rellenar con el caso base: $1! = 1$

1 1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25



Ejemplo 1: Factorial

Rellenar los casos posibles a partir de 1!

$$2! = 1! \cdot 2$$

1 1	2 2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25



Ejemplo 1: Factorial

Rellenar los casos posibles a partir de 2!

$$3! = 2! \cdot 3$$

1 1	2 2	3 6	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25



Ejemplo 1: Factorial

Continuar rellenando con la información disponible...

1 1	2 2	3 6	4 24	5 120
6 720	7 5040	8 40320	9 ...	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25



Ejemplo 2: Fibonacci

Calcular el 5º número de Fibonacci

0	1	2	3	4	5
---	---	---	---	---	---



Ejemplo 2: Fibonacci

Rellenar con los casos base: $\text{Fibonacci}(0) = 1$
 $\text{Fibonacci}(1) = 1$

0	1	1	1	1	1	1
---	---	---	---	---	---	---



Ejemplo 2: Fibonacci

Rellenar los casos posibles a partir de Fibo(0) y Fibo(1)

$$\text{Fibo}(2) = \text{Fibo}(1) + \text{Fibo}(0)$$

0	1	1	1	2				
---	---	---	---	---	--	--	--	--



Ejemplo 2: Fibonacci

Rellenar con la información disponible...

0	1	1	2	3	5	8
---	---	---	---	---	---	---



Top-Down vs Bottom-Up

- Cualquier problema se **debería** poder resolver de las 2 formas

	Top-Down	Bottom-Up
PROS	<ul style="list-style-type: none">- Transformación natural de recursión por Fuerza Bruta- Solo se calculan los resultados necesarios	<ul style="list-style-type: none">- Más rápido si hay muchas llamadas recursivas
CONTRAS	<ul style="list-style-type: none">- Más lento si hay muchas llamadas recursivas- Posible RTE por Stack Overflow	<ul style="list-style-type: none">- Menos intuitivo- Se calculan todos los resultados



ÍNDICE

- Motivación
- ¿Qué es?
- Características
- Top Down
- Bottom Up
- **Problemas clásicos**
- Más allá...



El problema de la Mochila

Dados:

- una mochila



con **capacidad máxima** w

- n objetos con un **valor** y **peso** determinados

Determinar el valor máximo que se podrá acumular en la mochila



El problema de la Mochila

Ejemplo:

Objetos disponibles:

peso

valor



2

3



3

2



2

1



4

4

Peso máximo: 5



El problema de la Mochila

¿Enfoque Greedy?

- Coger siempre el objeto de mayor valor disponible



El problema de la Mochila



p	2	3	2	4
v	3	2	1	4

Peso : 0

Valor : 0



El problema de la Mochila



p	2	3	2	4
v	3	2	1	4

Peso : 0

Valor : 0



El problema de la Mochila

¿Es la solución óptima?



p	2	3	2	4
v	3	2	1	4

Peso: 4

Valor : 4



El problema de la Mochila



p	2	3	2	4
v	3	2	1	4

Peso : 0

Valor : 0



El problema de la Mochila



p	2	3	2	4
v	3	2	1	4

Peso : 2

Valor : 3



El problema de la Mochila

¡NO! No podemos llegar a la solución óptima de forma voraz



p	2	3	2	4
v	3	2	1	4

Peso : 5

Valor : 5



El problema de la Mochila



¿Fuerza bruta?

- Colocando los objetos en fila, decidir si **escoger** o no cada uno de ellos



El problema de la Mochila



	0	1	2	3
				
p	2	2	4	4
v	3	2	5	4

Peso : 0

Valor : 0



El problema de la mochila

0

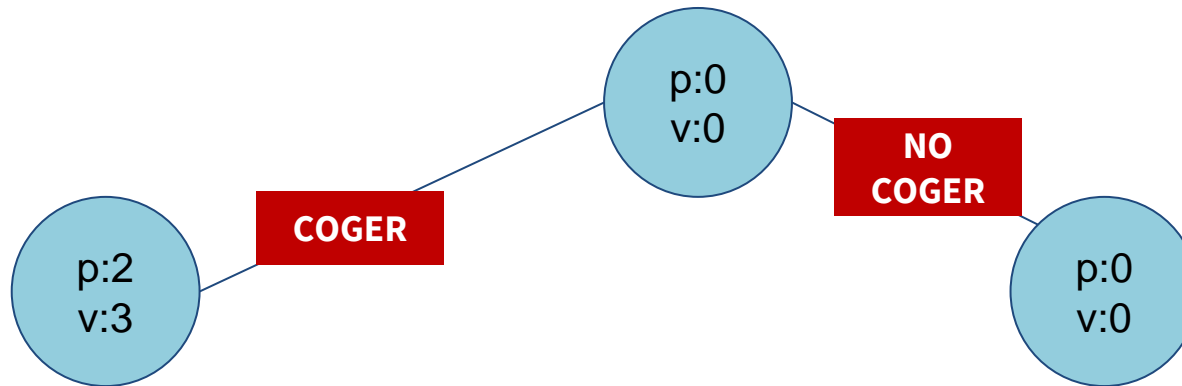
p:0
v:0



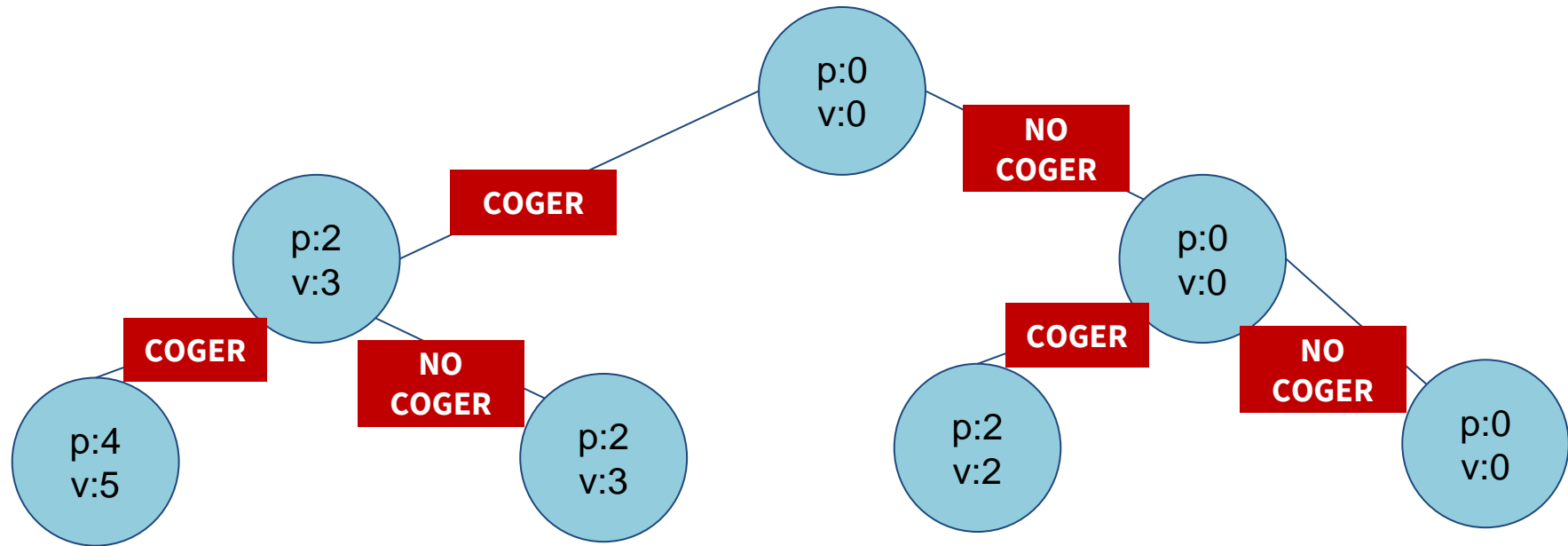
El problema de la mochila

0

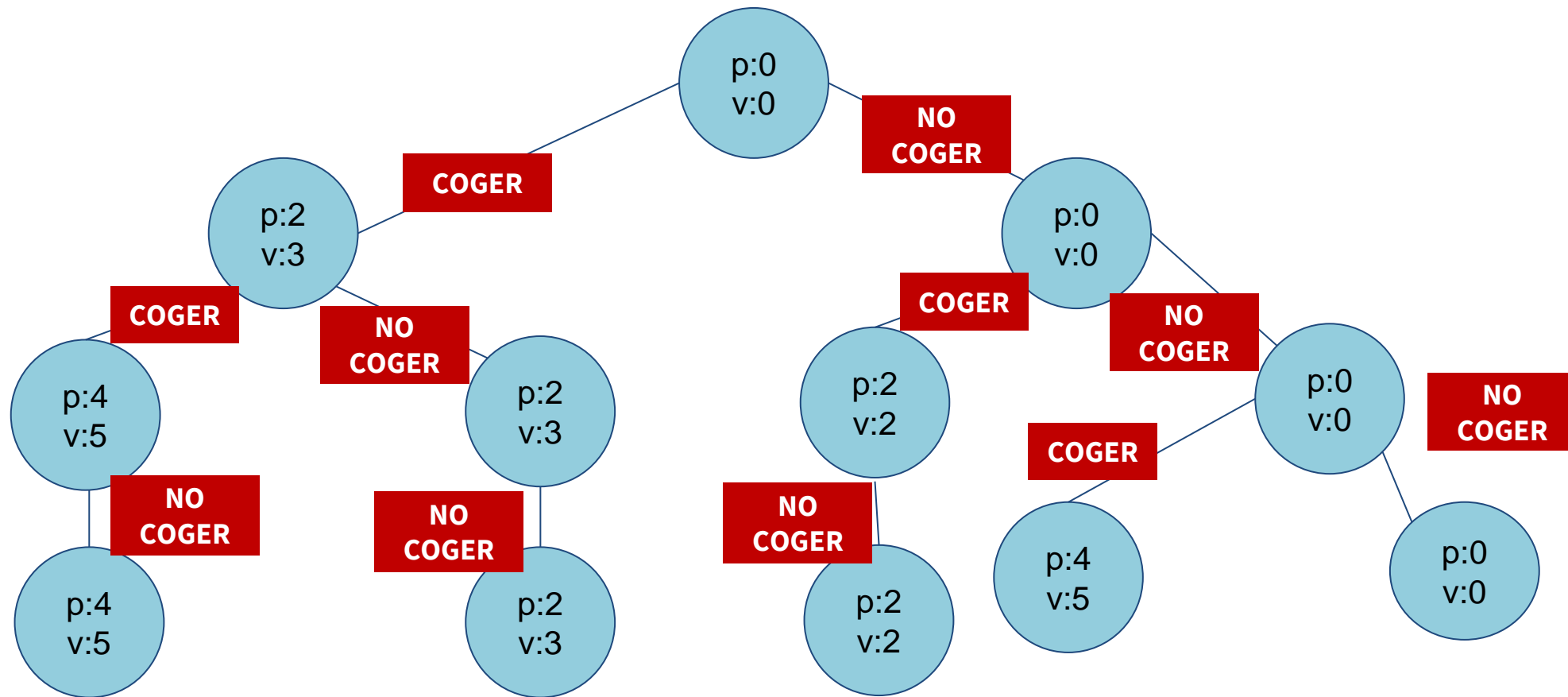
1



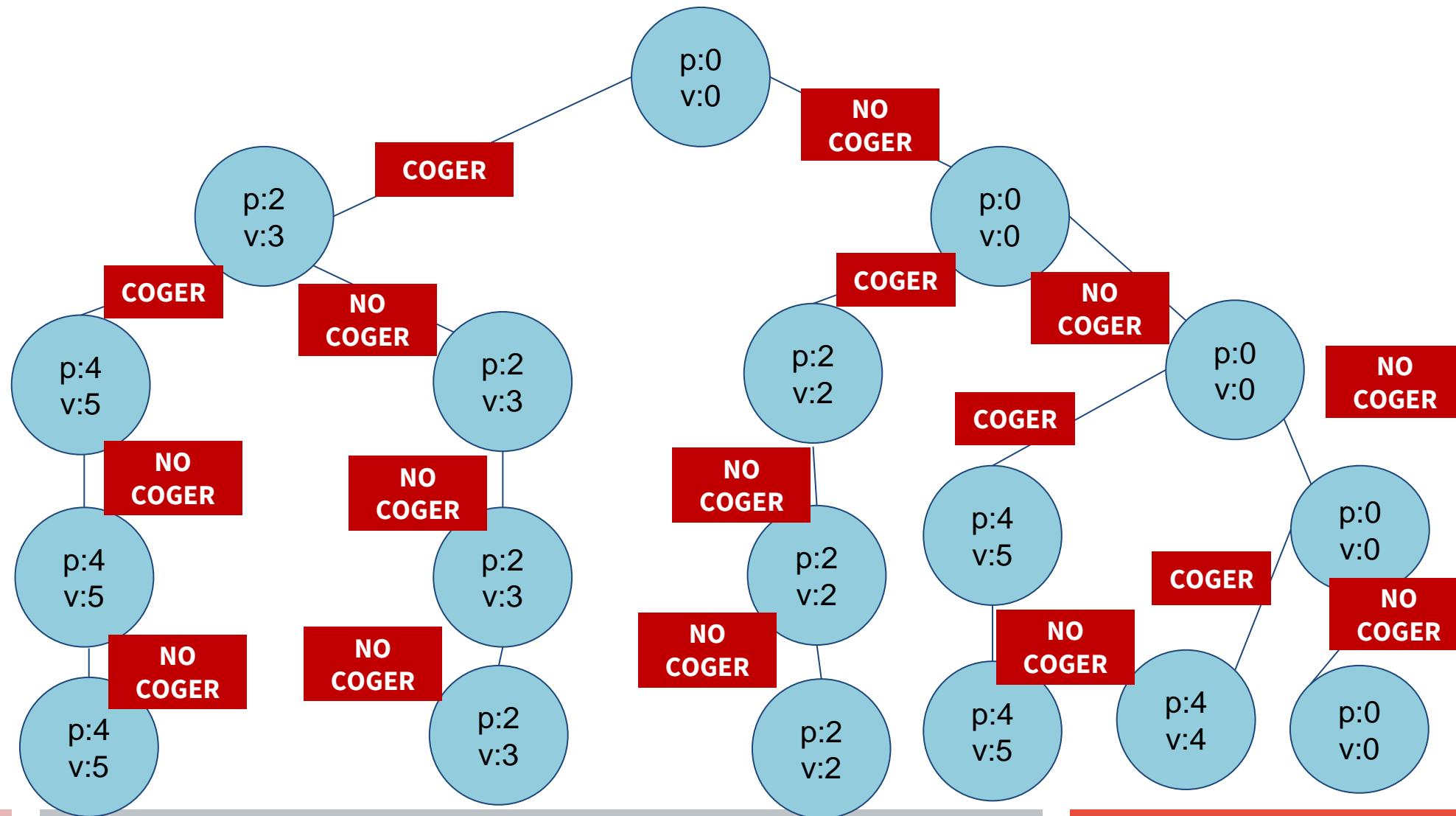
El problema de la mochila



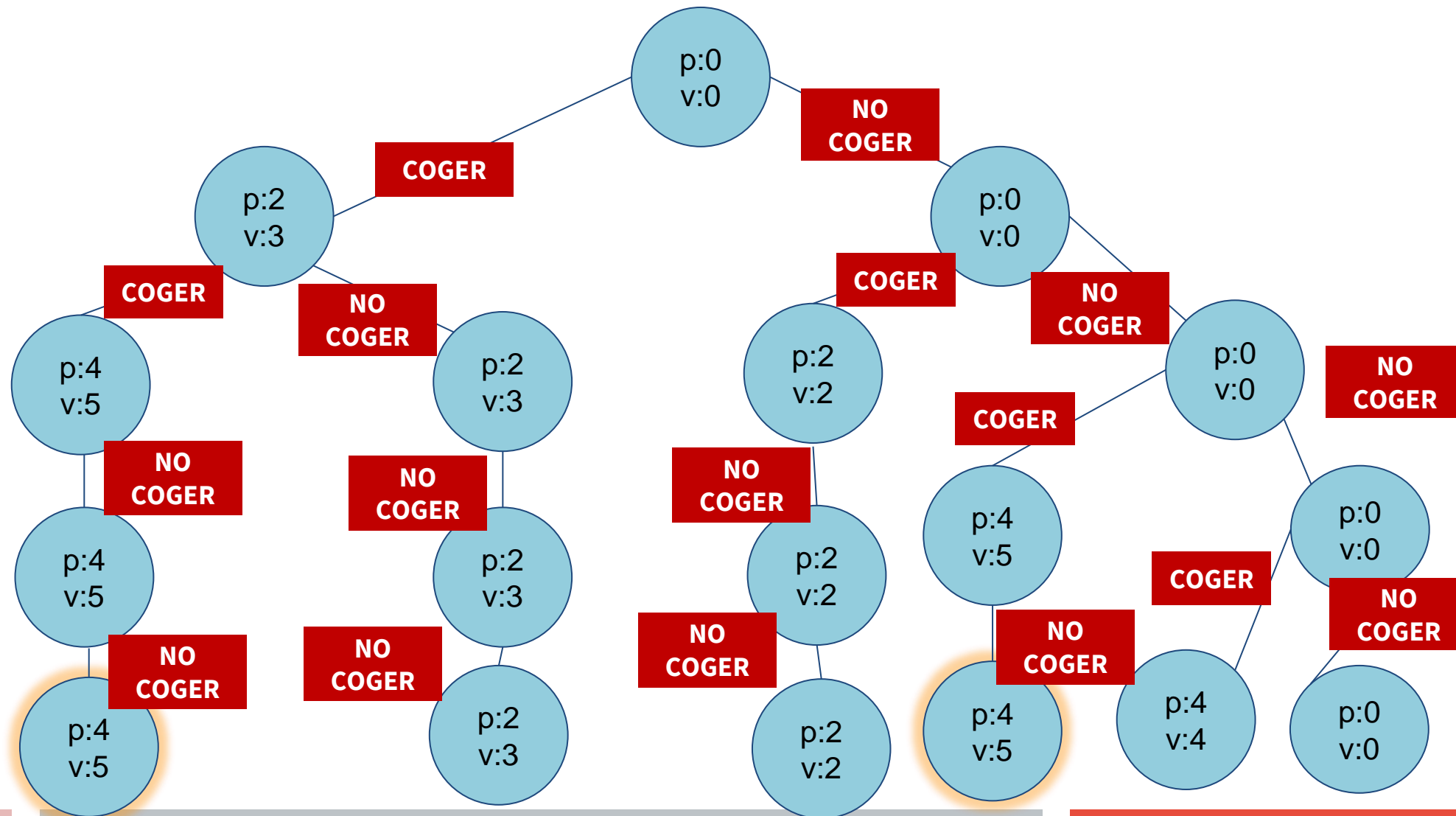
El problema de la mochila



El problema de la mochila



El problema de la mochila



El problema de la mochila

¿Cómo construimos la recursión?

Estados: (índice, peso acumulado)

Transición: coger o no coger el objeto y pasar al siguiente

Casos base: cuando hemos pasado por todos los objetos
(índice= $n+1$)



El problema de la mochila

-Pseudocódigo Fuerza Bruta:

```
Mochila (int i, int peso){  
    //caso base (hemos recorrido toda la fila)  
    si i==n+1: devolver 0;  
    //transiciones  
    si no: devolver Max(Mochila(i+1, peso), //no coger  
                        Mochila(i+1, peso+peso[i])) //coger  
}
```



El problema de la mochila

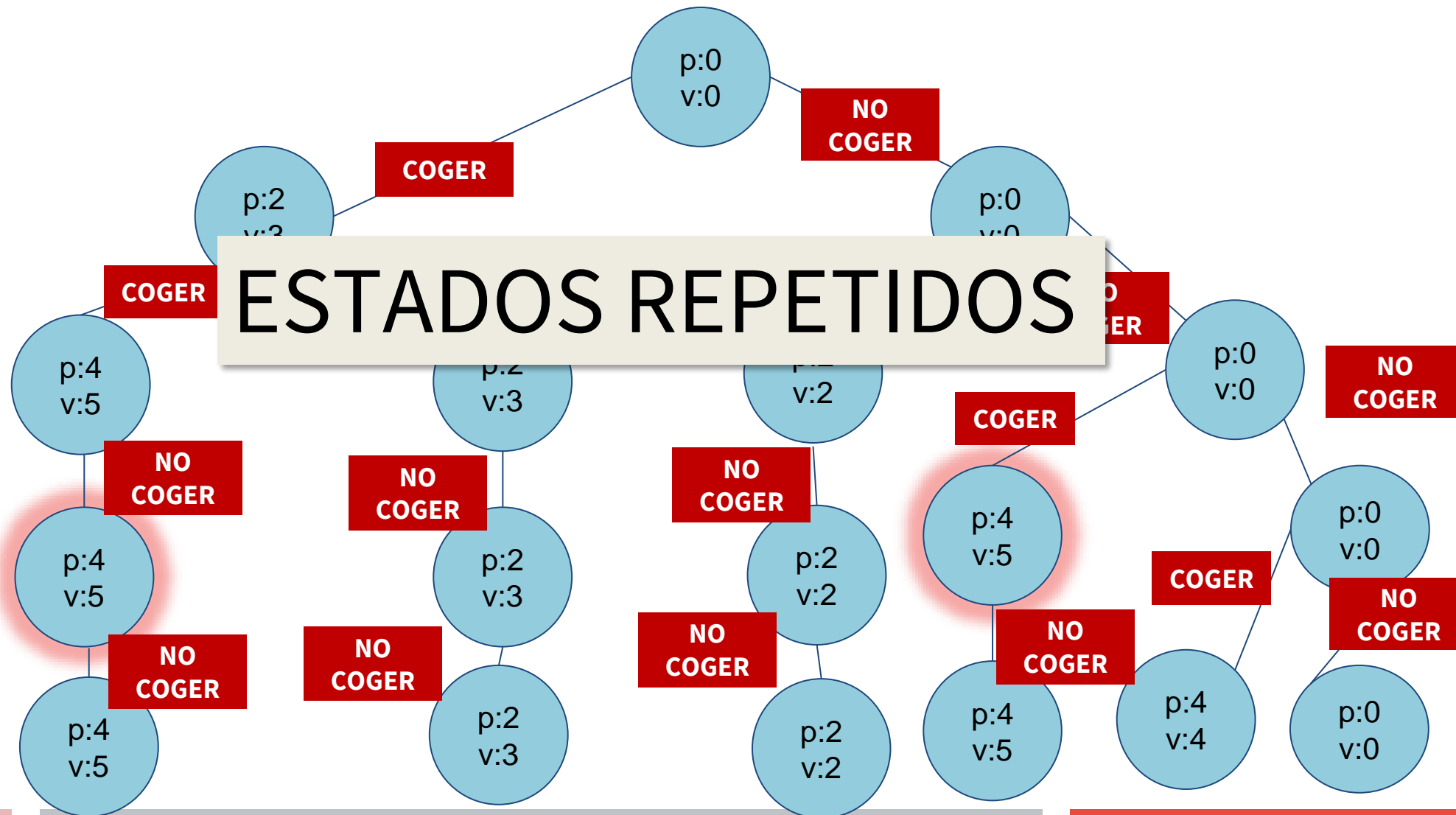
-¿Es suficiente?

- **NO**

Generamos todos los subconjuntos posibles: $O(2^n)$
EXPONENCIAL



El problema de la mochila



El problema de la mochila

-Estados repetidos -> DP

Estado ✓

Transición ✓

Memoria ?

Casos base ✓



El problema de la mochila

-Estados repetidos -> DP

Estado ✓

Transición ✓

Memoria: Memo(índice, peso) -> **bidimensional**

Casos base ✓



El problema de la mochila

-Estados repetidos -> DP

Estado ✓

Transición ✓

Memoria ✓

Casos base ✓



El problema de la mochila

-Pseudocódigo DP:

```
Mochila (int i, int peso){  
    //caso base (hemos recorrido toda la fila)  
    si i==n+1: devolver 0;  
    si Memo(i,peso) conocida: devolver Memo(i,peso)  
    //transiciones  
    si no: Memo(i,peso)= Max(Mochila(i+1, peso), //no coger  
                             Mochila(i+1, peso+peso[i])) //coger  
    devolver Memo(i,peso)  
}
```

-**Complejidad?** $O(n \cdot p_{\max} \cdot 2) = O(n \cdot p_{\max})$



Otros problemas clásicos...

- Longest Increasing Subsequence (LIS)
- Longest Common Subsequence (LCS)
- Coin Problem



ÍNDICE

- Motivación
- ¿Qué es?
- Características
- Top Down
- Bottom Up
- Problemas clásicos
- **Más allá...**



Más allá

- Return choice
- DP con máscara de bits
- Teoría de juegos



Resolviendo el problema de las vacas

- Estado: Cubos de los extremos
- Transiciones: Cubos que te puedes comer
- Casos base: Solo hay dos cubos
- Memo: matriz 2-dimensional



Problemas propuestos

- Parkímetros ([AER 251](#))
- El precio de la gasolina ([AER 606](#))
- Cazatesoros ([AER 353](#))
- El carpintero Ebanisto ([AER 603](#))



Concurso FINAL

- Link para apuntarse por equipos: [forms](#) 

