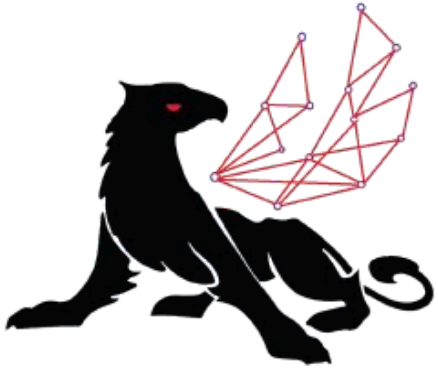
Programación Dinámica



PROGRAMACIÓN COMPETITIVA
URJC -2025

Organizadores:

- Isaac Lozano (isaac.lozano@urjc.es)
- Sergio Salazar (sergio.salazar@urjc.es)
- Adaya Ruiz (am.ruiz.2020@alumnos.urjc.es)
- Eva Gómez (e.gomezf.2020@alumnos.urjc.es)
- Lucas Martín (lucas.martin@urjc.es)
- Iván Penedo (ivan.penedo@urjc.es)
- **Alicia Pina** (alicia.pina@urjc.es)
- **Sara García** (sara.garciar@urjc.es)
- Raúl Fauste (r.fauste.2020@alumnos.urjc.es)

Alejandro Mayoral (a.mayoralg.2020@alumnos.urjc.es)

David Orna (de.orna.2020@alumnos.urjc.es)

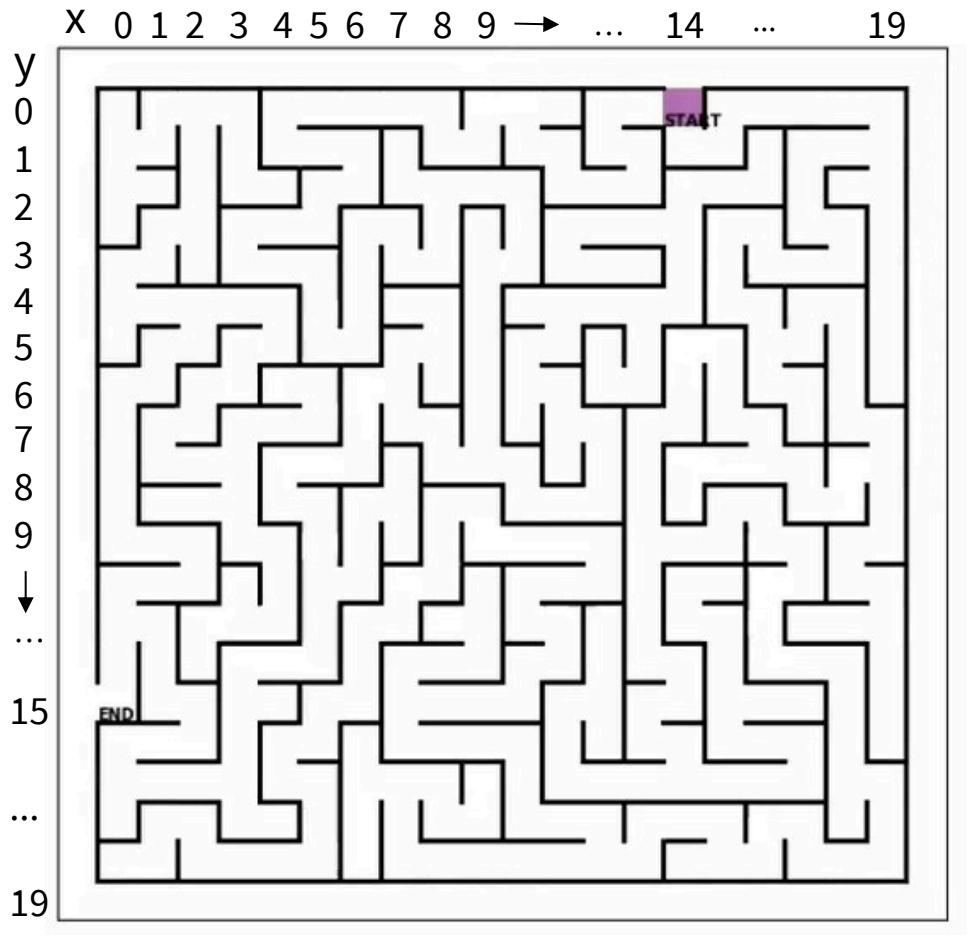
Recordatorio de recursión

- Una función **recursiva** es una función que contiene una llamada a sí misma:

solucion(x_1, x_2,…, x_n):
…
…
solucion(y_1, y_2,…, y_n)

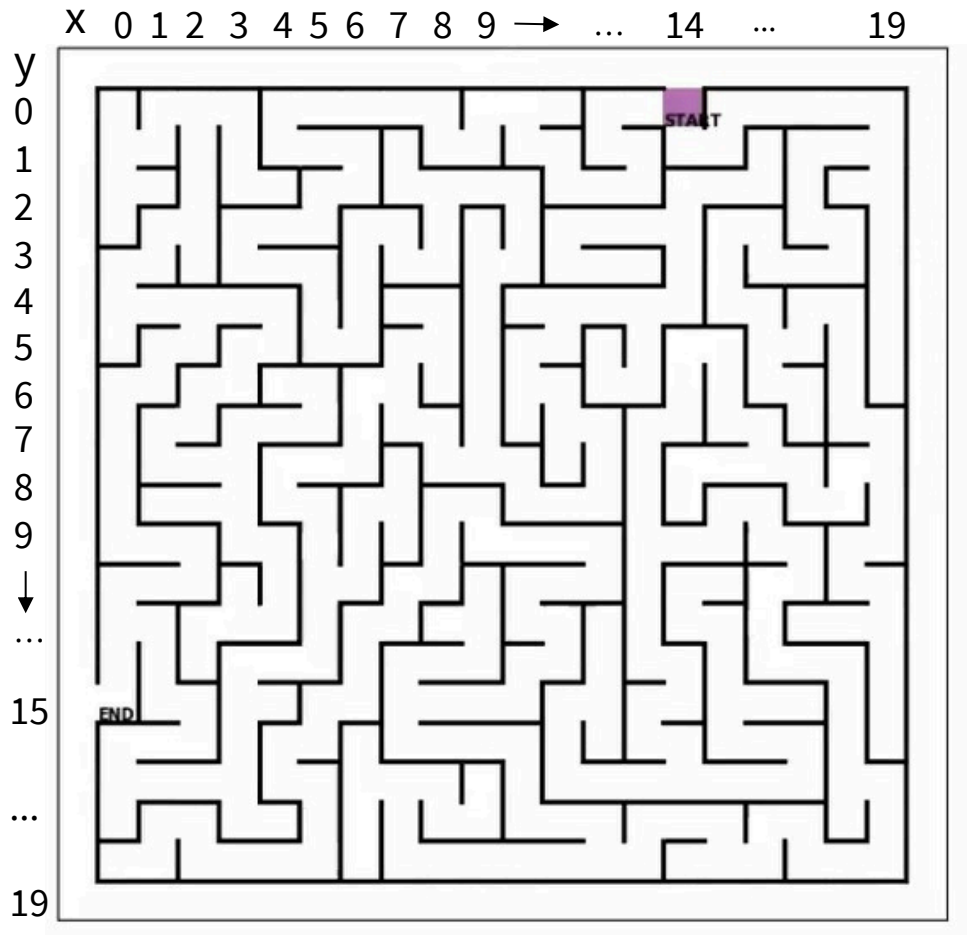
Recordatorio de recursión

- Ejemplo: queremos averiguar si podemos salir de un laberinto



Recordatorio de recursión

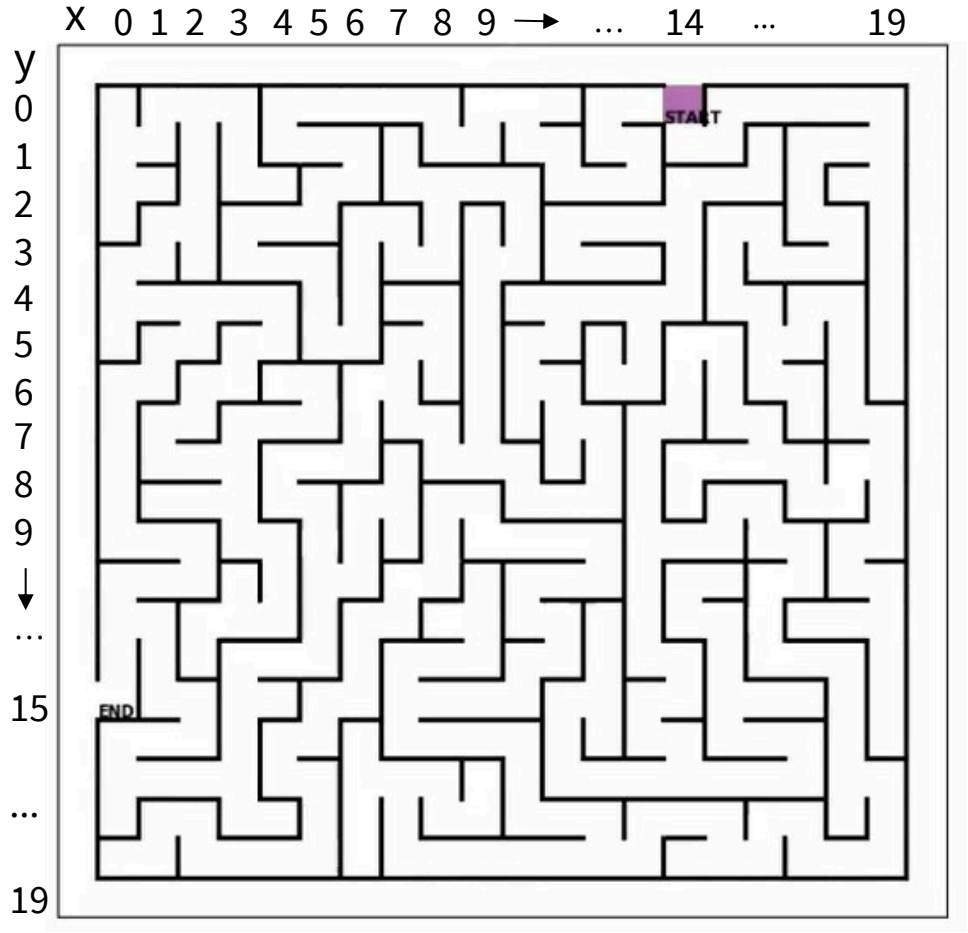
- Ejemplo: queremos averiguar si podemos salir de un laberinto



salir (14,0)?

Recordatorio de recursión

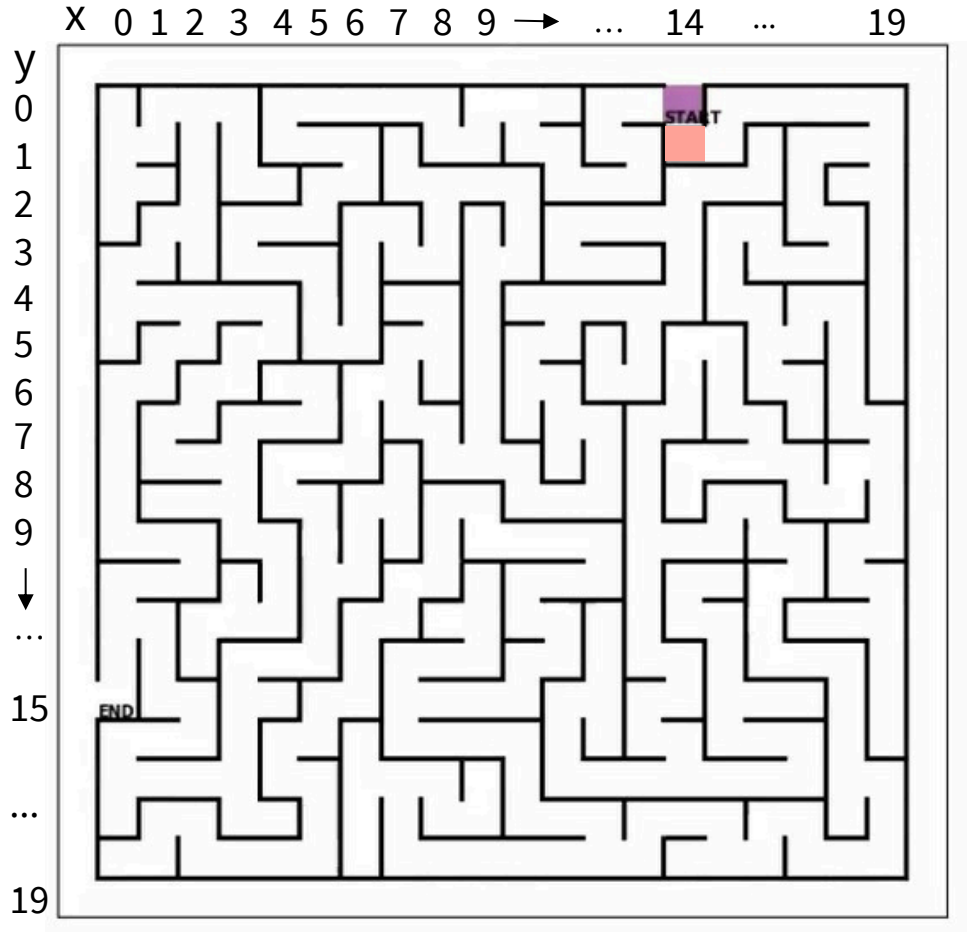
- Ejemplo: queremos averiguar si podemos salir de un laberinto



salir (14,0)=
  salir(14,1) OR
  salir (13 ,0)

Recordatorio de recursión

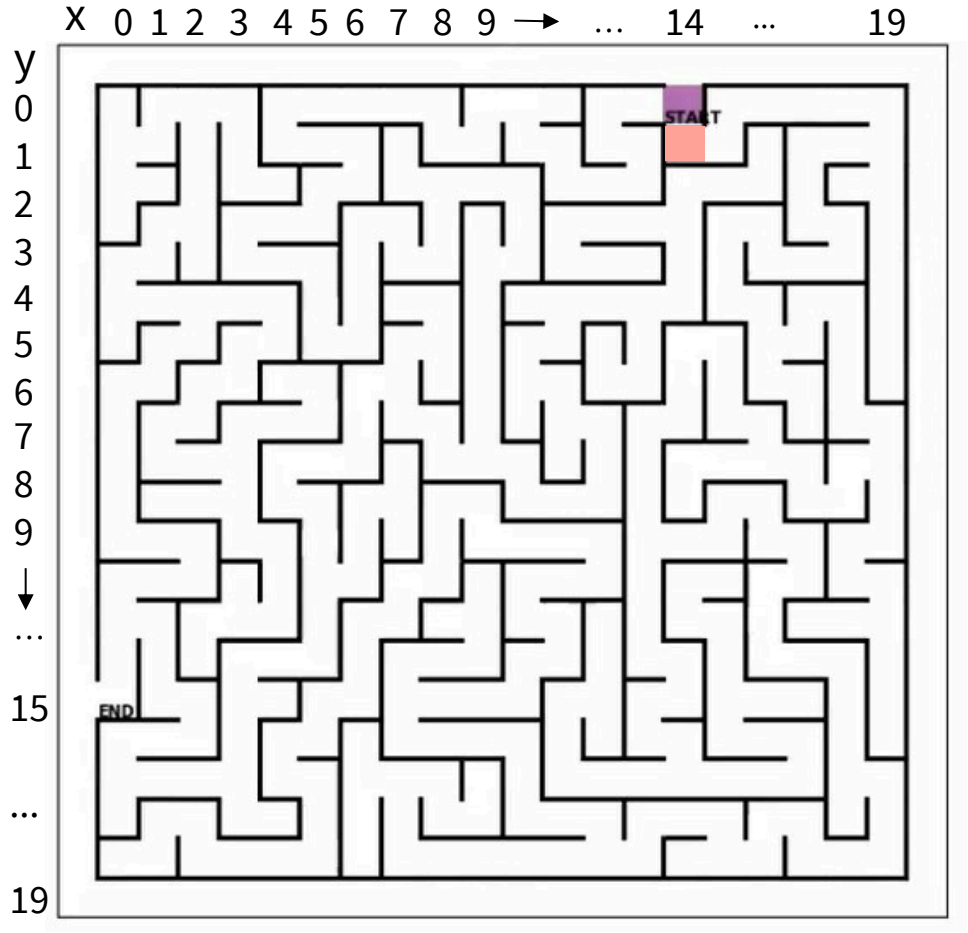
- Ejemplo: queremos averiguar si podemos salir de un laberinto



salir (14,0)=
  salir(14,1) OR
  salir (13 ,0)

Recordatorio de recursión

- Ejemplo: queremos averiguar si podemos salir de un laberinto

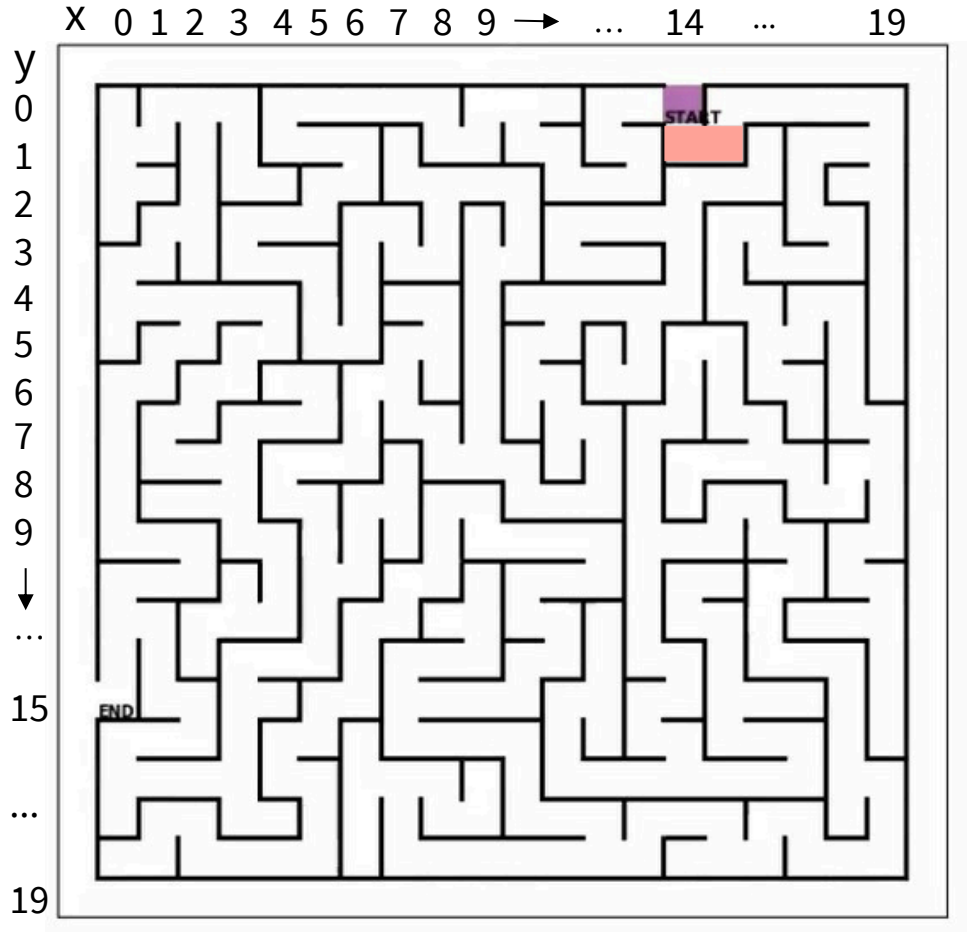


salir (14,0)=
  salir(14,1) OR
  salir (13 ,0)

  salir (14,1)=
    salir (15,1)

Recordatorio de recursión

- Ejemplo: queremos averiguar si podemos salir de un laberinto

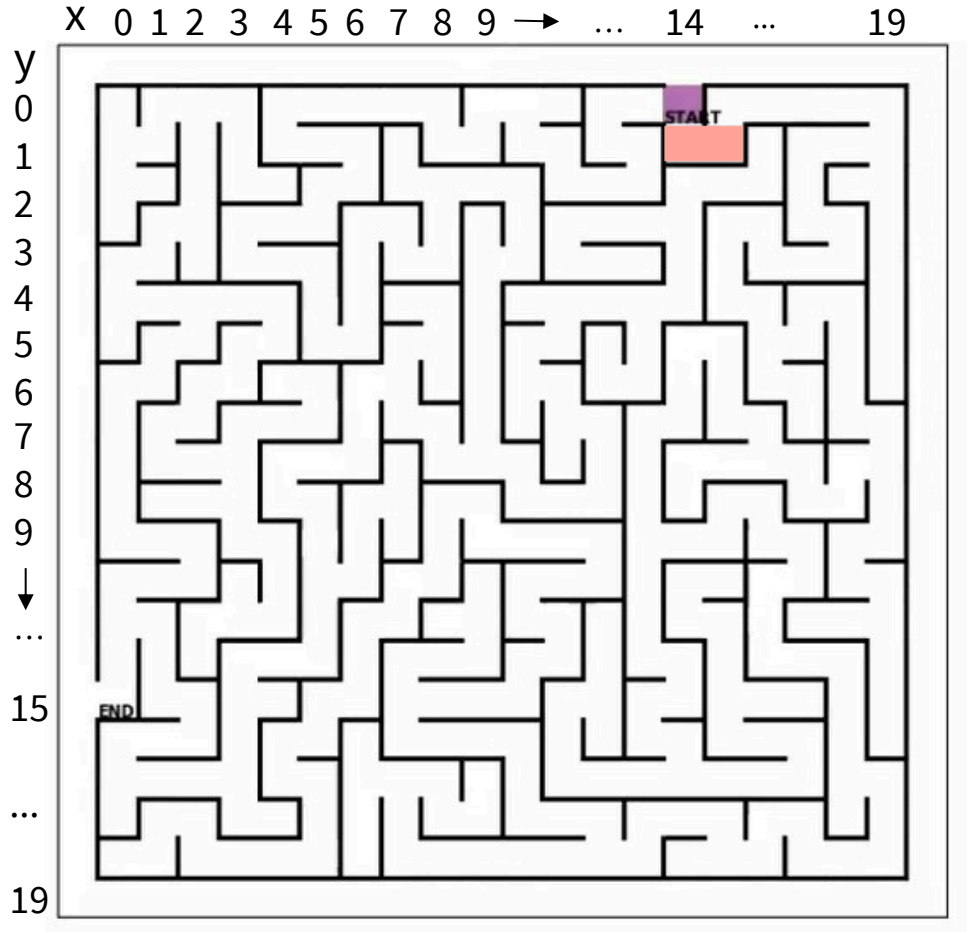


salir (14,0)=
salir(14,1) OR
salir (13 ,0)

salir (14,1)=
salir (15,1)

Recordatorio de recursión

- Ejemplo: queremos averiguar si podemos salir de un laberinto



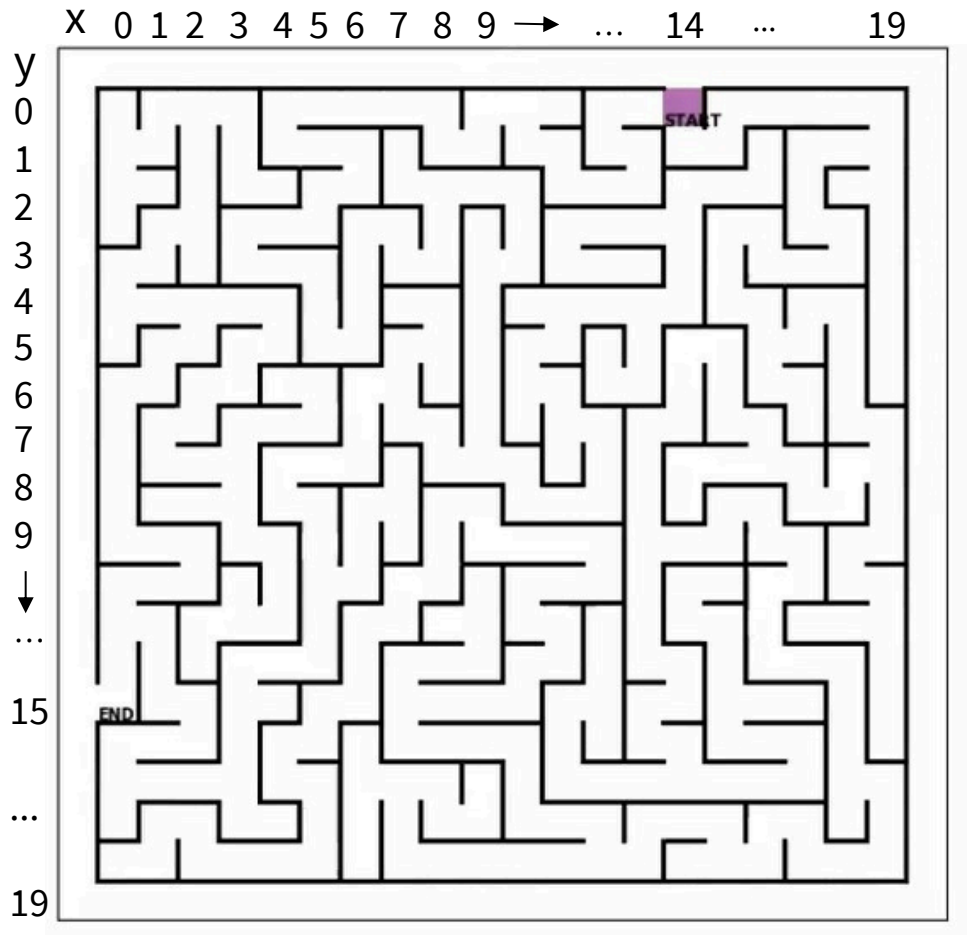
salir (14,0)=
  salir(14,1) OR
  salir (13 ,0)

  salir (14,1)=
    salir (15,1)

    …

Recordatorio de recursión

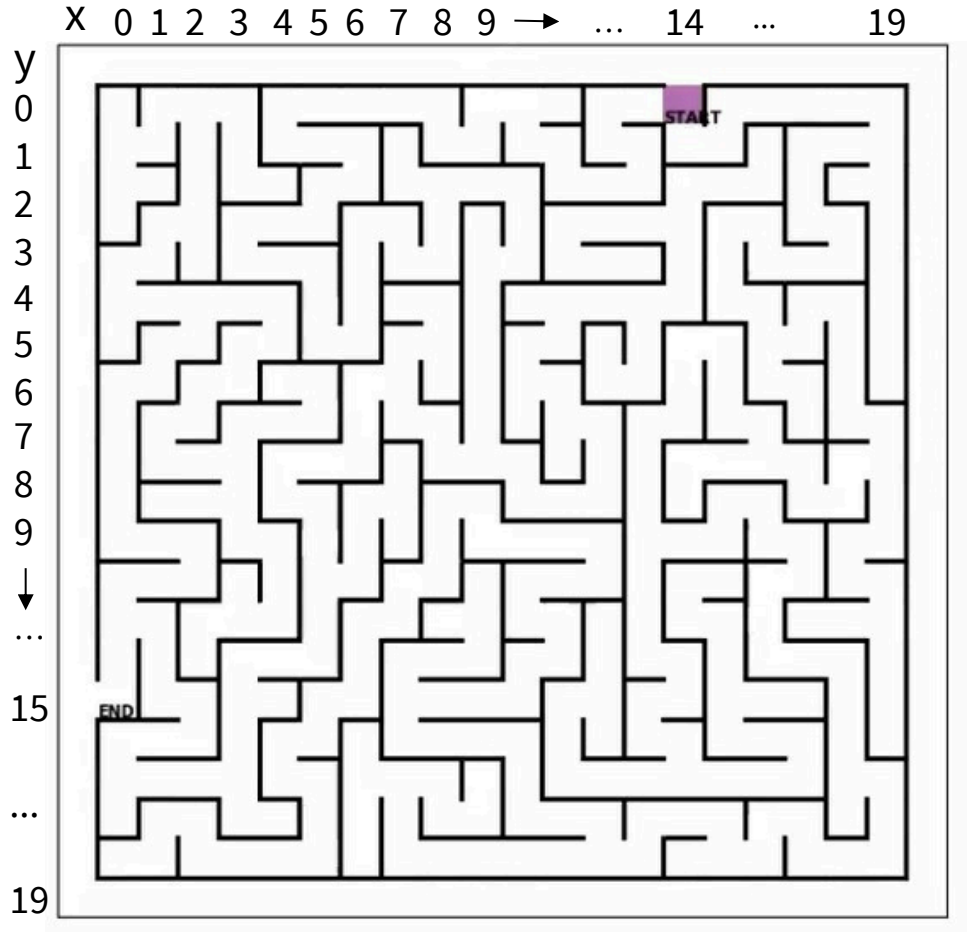
- Ejemplo: queremos averiguar si podemos salir de un laberinto




```
salir(x,y):
    return salir(adyacentes)
```

Recordatorio de recursión

- Ejemplo: queremos averiguar si podemos salir de un laberinto

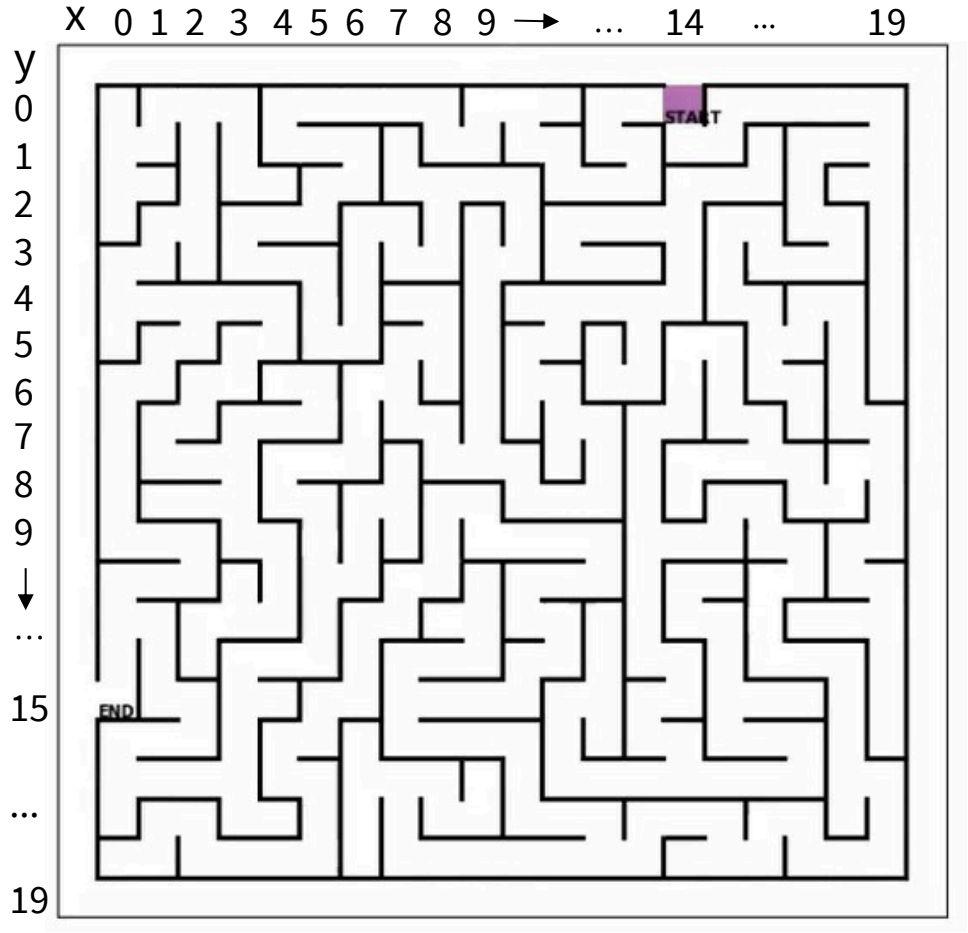


¿cuándo paramos?

- Cuando no queda nada por recorrer
- Cuando llegamos a la salida

Recordatorio de recursión

- Ejemplo: queremos averiguar si podemos salir de un laberinto




```
salir(x,y):
   if((x,y)==salida)
      return TRUE
   if(no adyacentes)
      return FALSE
   return salir(adyacentes)
```

Recordatorio de recursión

- Esquema básico de recursión:

solucion(x_1, x_2,…,xn):
    if(**caso base**):
        return ------
    llamadas recursivas

Ejemplo clásico: la sucesión de Fibonacci

Ejemplo clásico: la sucesión de Fibonacci

- el primer y segundo números de Fibonacci son el 1
- los siguientes números se calculan sumando los dos anteriores

Ejemplo clásico: la sucesión de Fibonacci

- el primer y segundo números de Fibonacci son el 1
- los siguientes números se calculan sumando los dos anteriores (ej: 21=13+8)

1 1 2 3 5 8 13 21 34 ...

Ejemplo clásico: la sucesión de Fibonacci

- ¿Podemos construir esta sucesión de forma recursiva?
  - cuáles son los casos base?
  - cómo son las llamadas recursivas?

Ejemplo clásico: la sucesión de Fibonacci

Enlace en Telegram

Ejemplo clásico: la sucesión de Fibonacci

- Solución recursiva:

  fibonacci(i):

Ejemplo clásico: la sucesión de Fibonacci

- Solución recursiva:

```
fibonacci(i):
    if(i==1 or i==2):
        return 1
    return fibonacci(i-1) + fibonacci(i-2)
```

Ejemplo clásico: la sucesión de Fibonacci

- Solución recursiva:

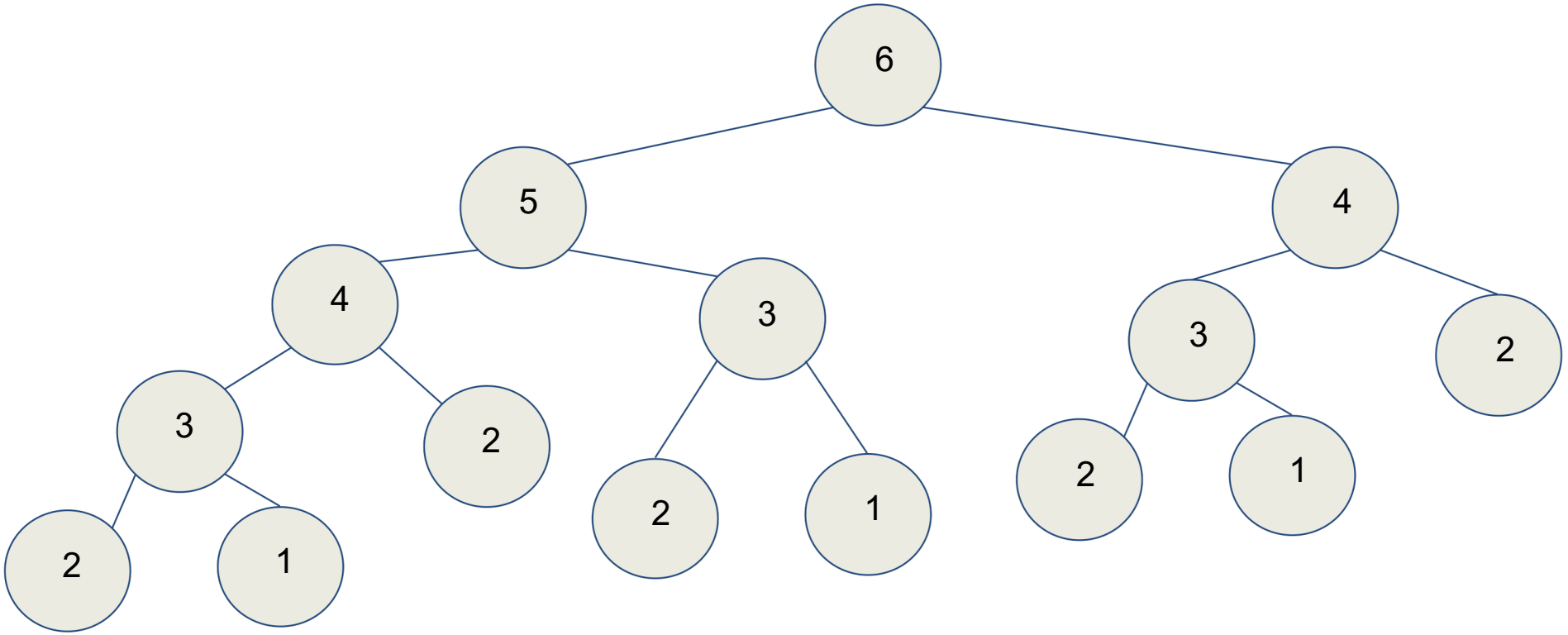
```
fibonacci(i):
    if(i==1 or i==2):        ← casos base
        return 1
    return fibonacci(i-1) + fibonacci(i-2)   ← llamadas recursivas
```

Ejemplo clásico: la sucesión de Fibonacci

- ¿Es esto eficiente?

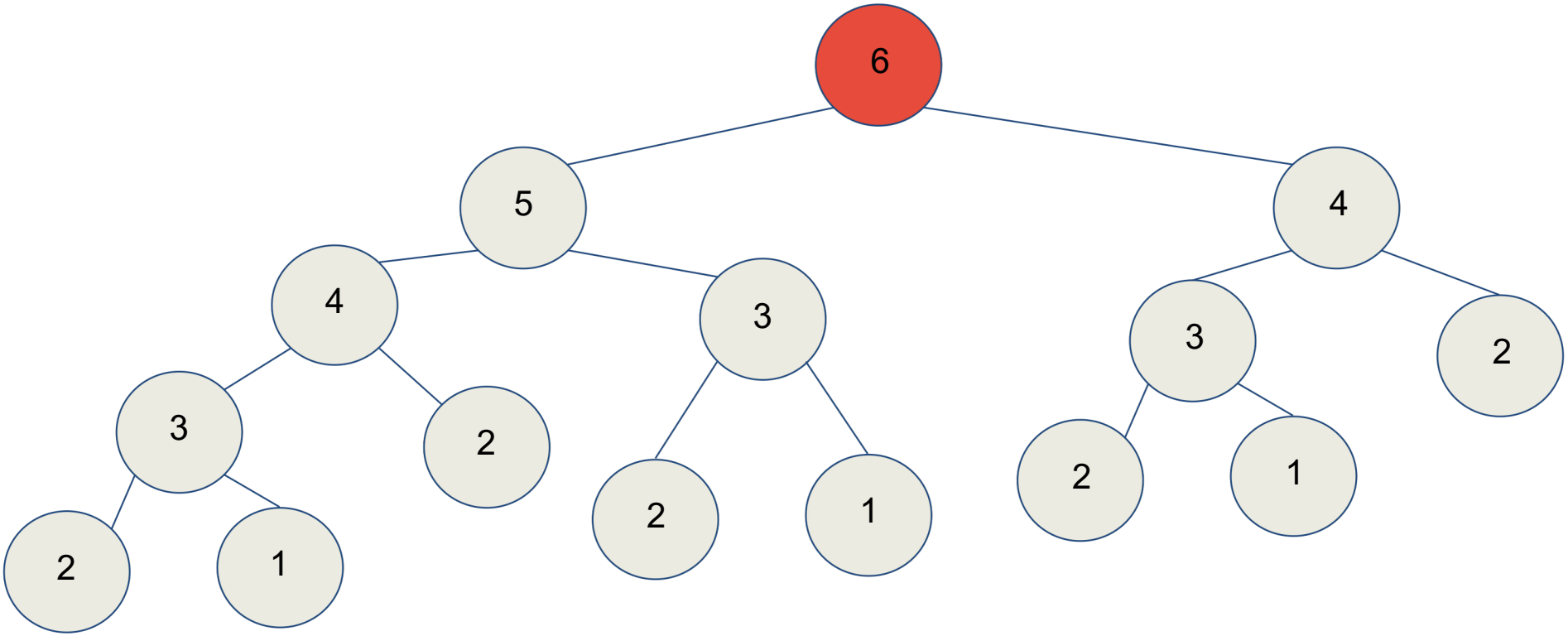
Fibonacci

Ejemplo: Fibonacci(6)



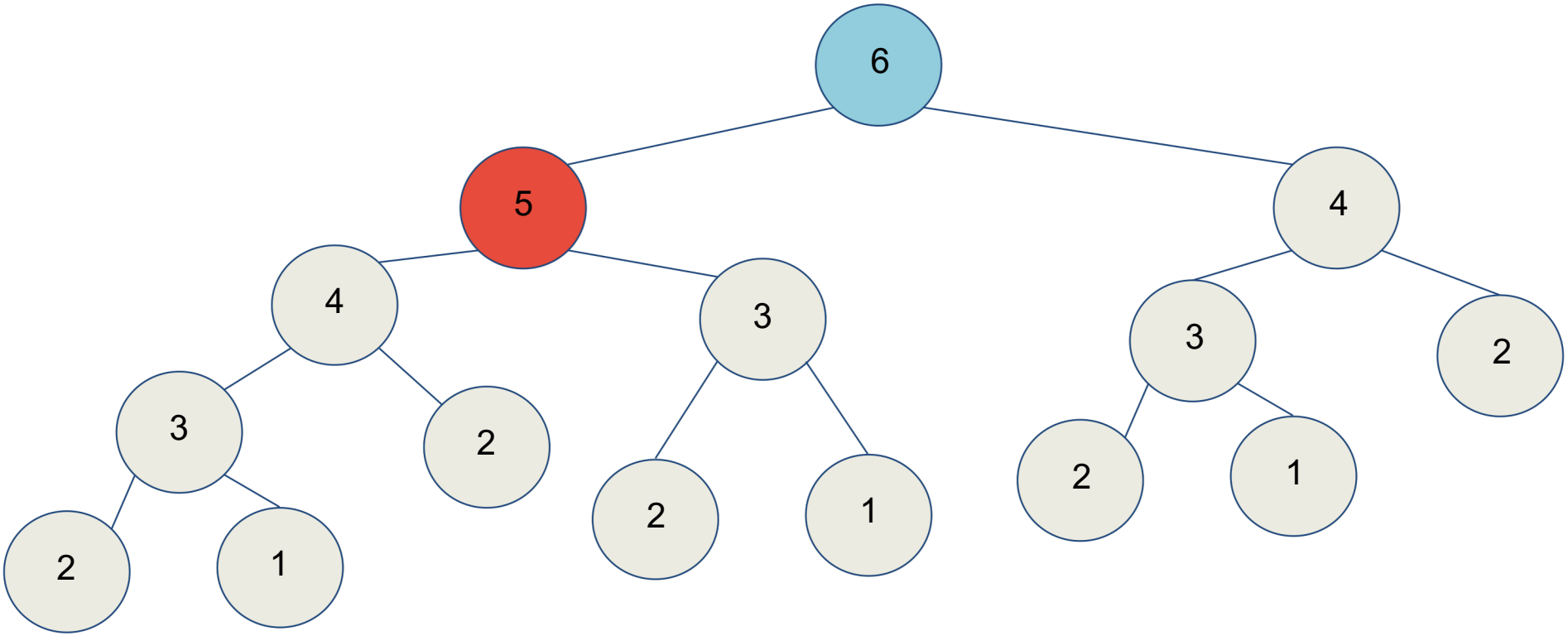
Fibonacci

Ejemplo: Fibonacci(6)



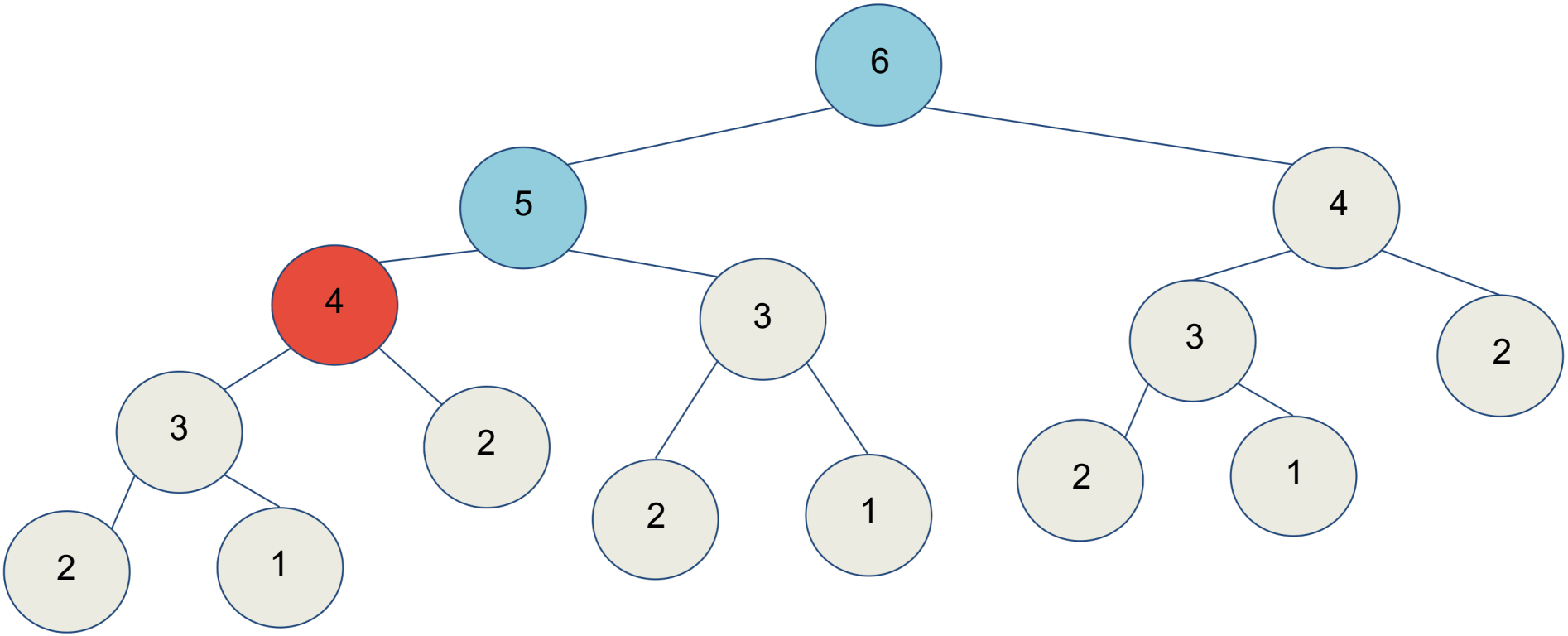
Fibonacci

Ejemplo: Fibonacci(6)



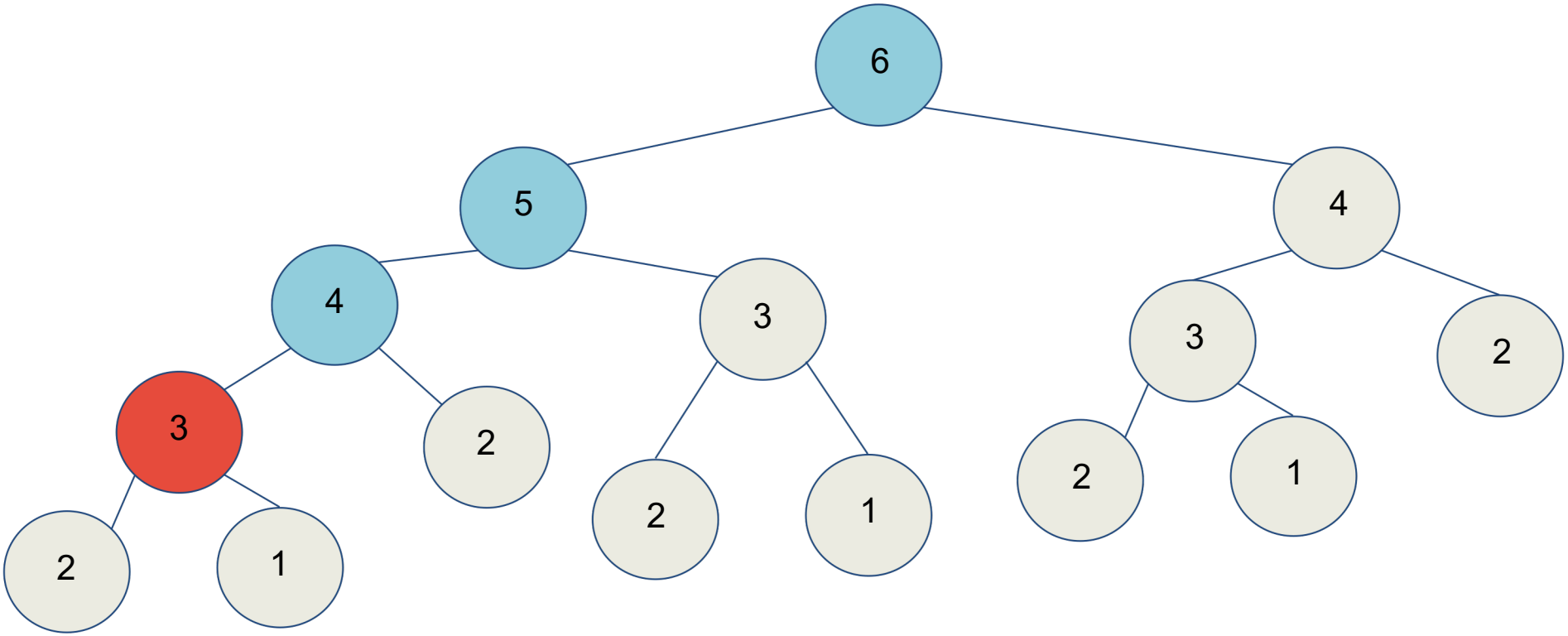
Fibonacci

Ejemplo: Fibonacci(6)



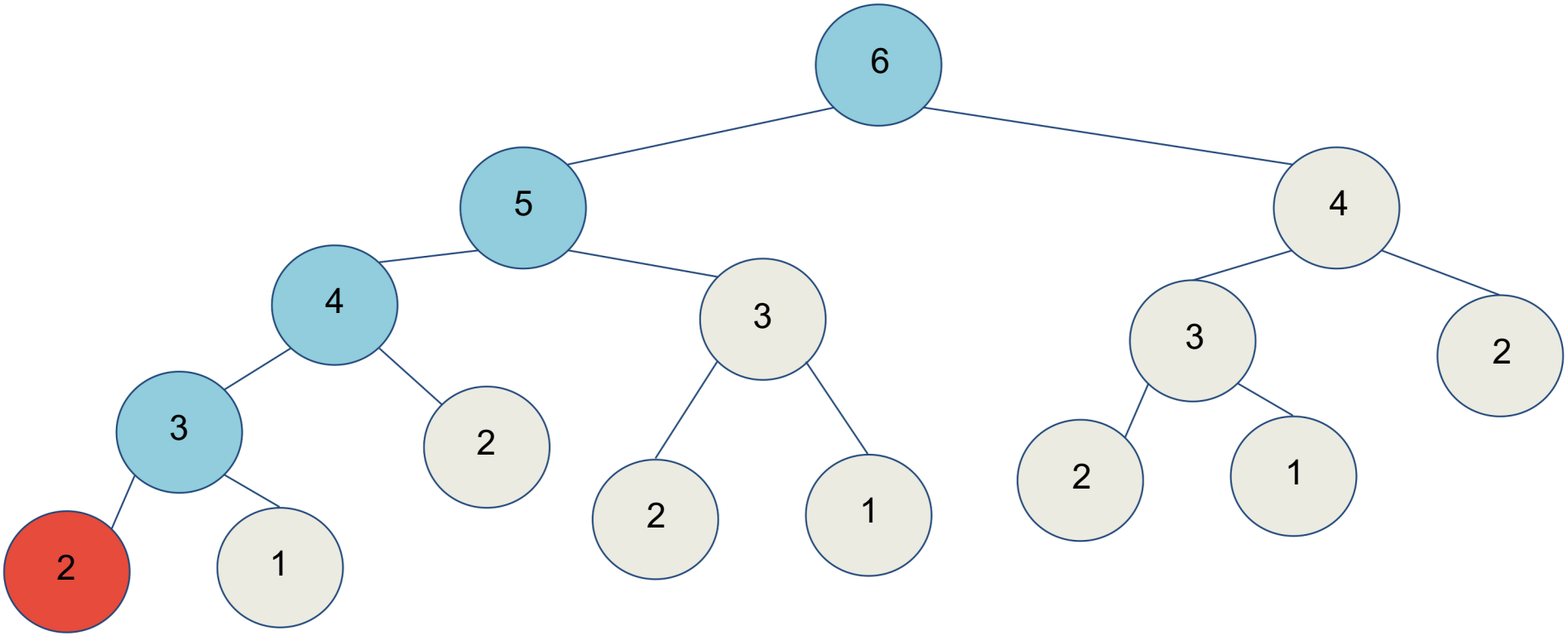
Fibonacci

Ejemplo: Fibonacci(6)



Fibonacci

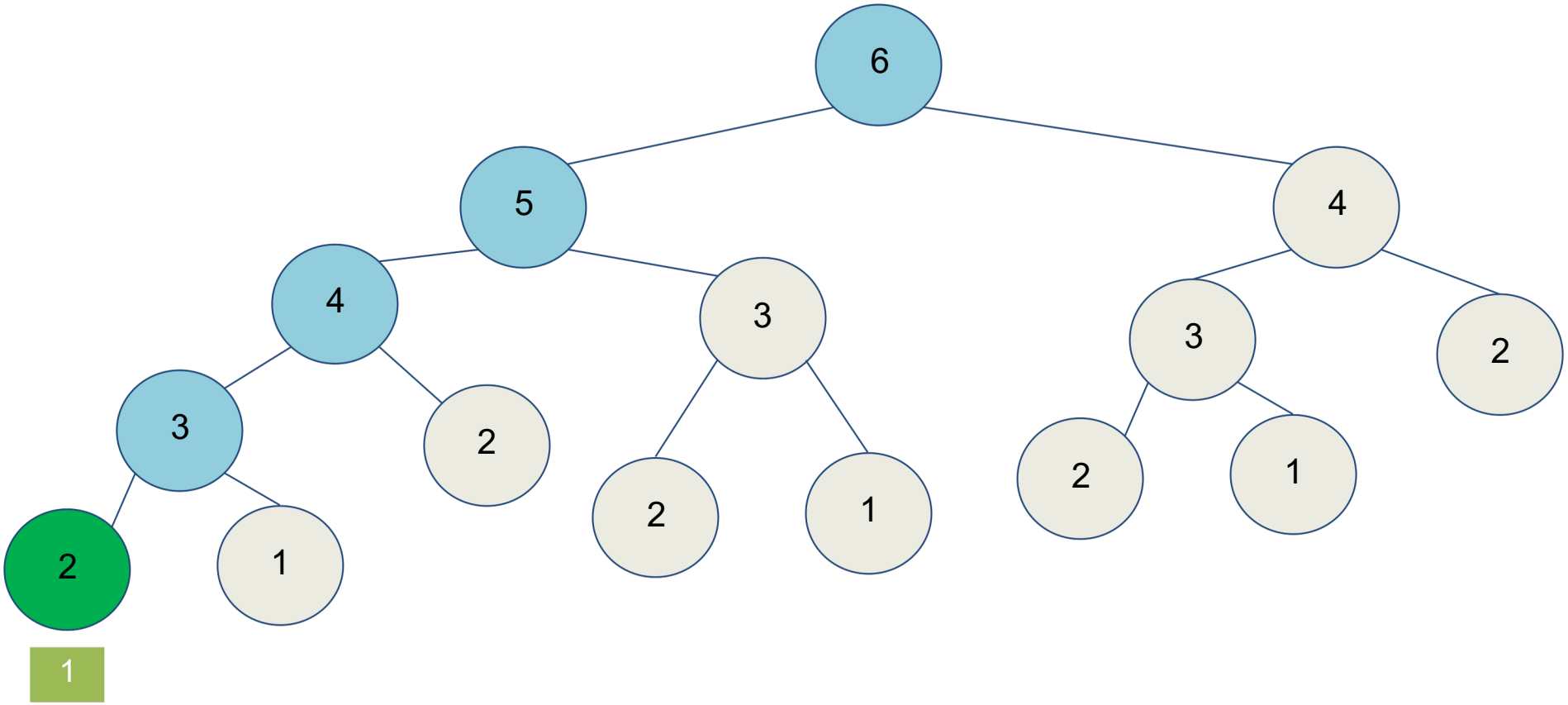
Ejemplo: Fibonacci(6)



Caso base: Fibonacci(2) = 1

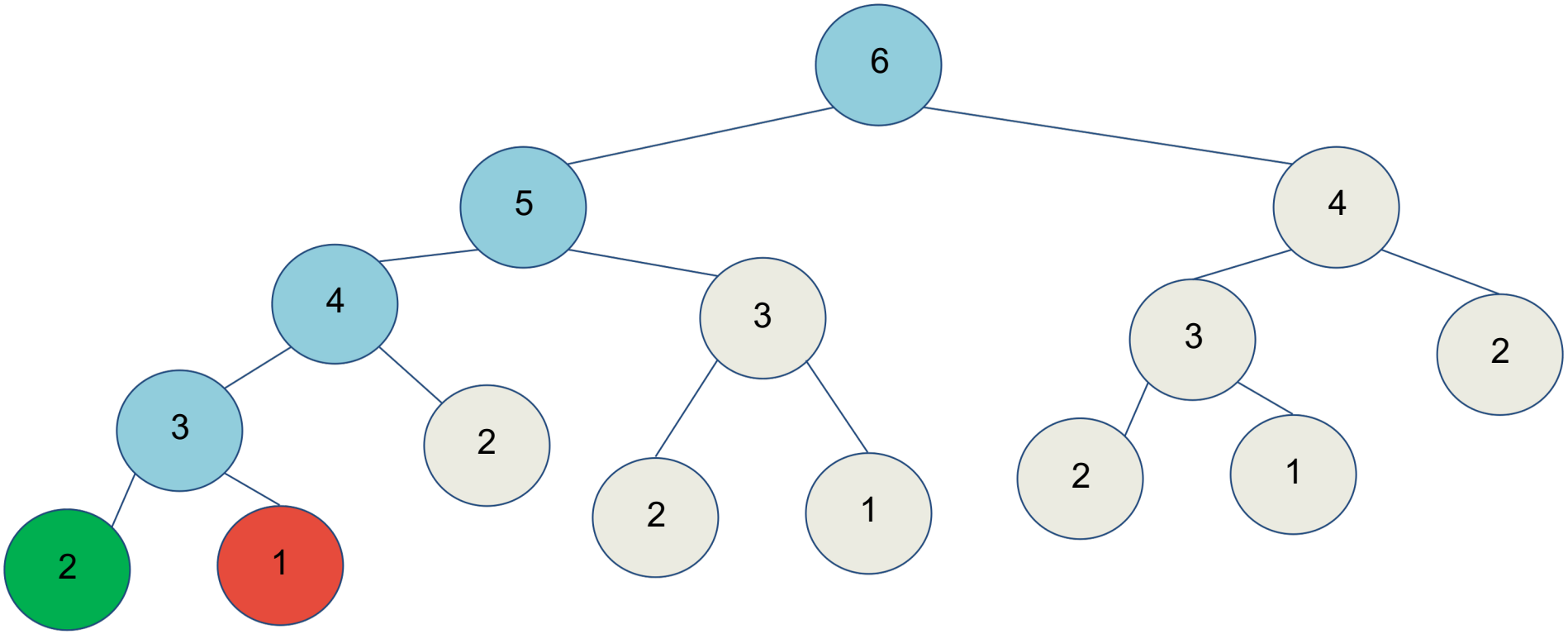
Fibonacci

Ejemplo: Fibonacci(6)



Fibonacci

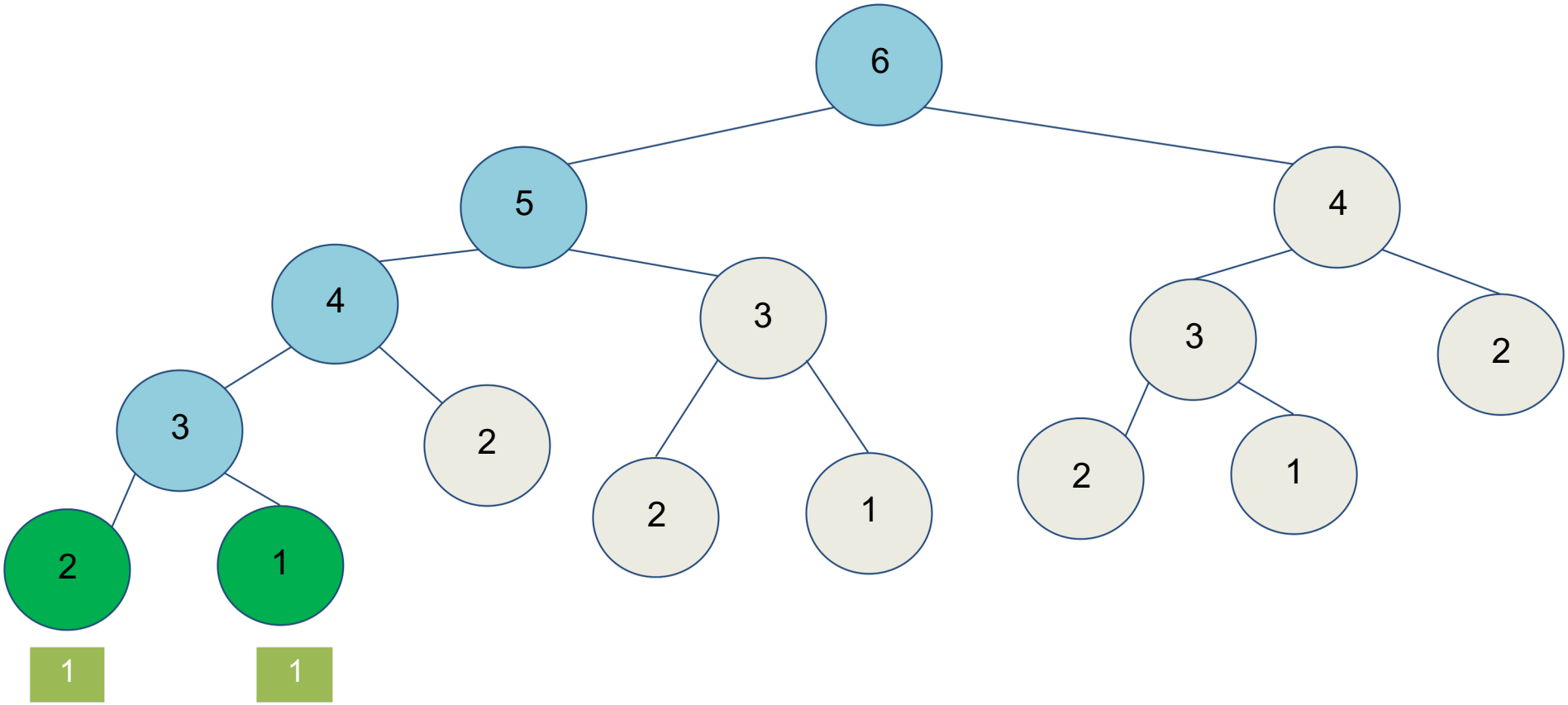
Ejemplo: Fibonacci(6)



1 Caso base: Fibonacci(1) = 1

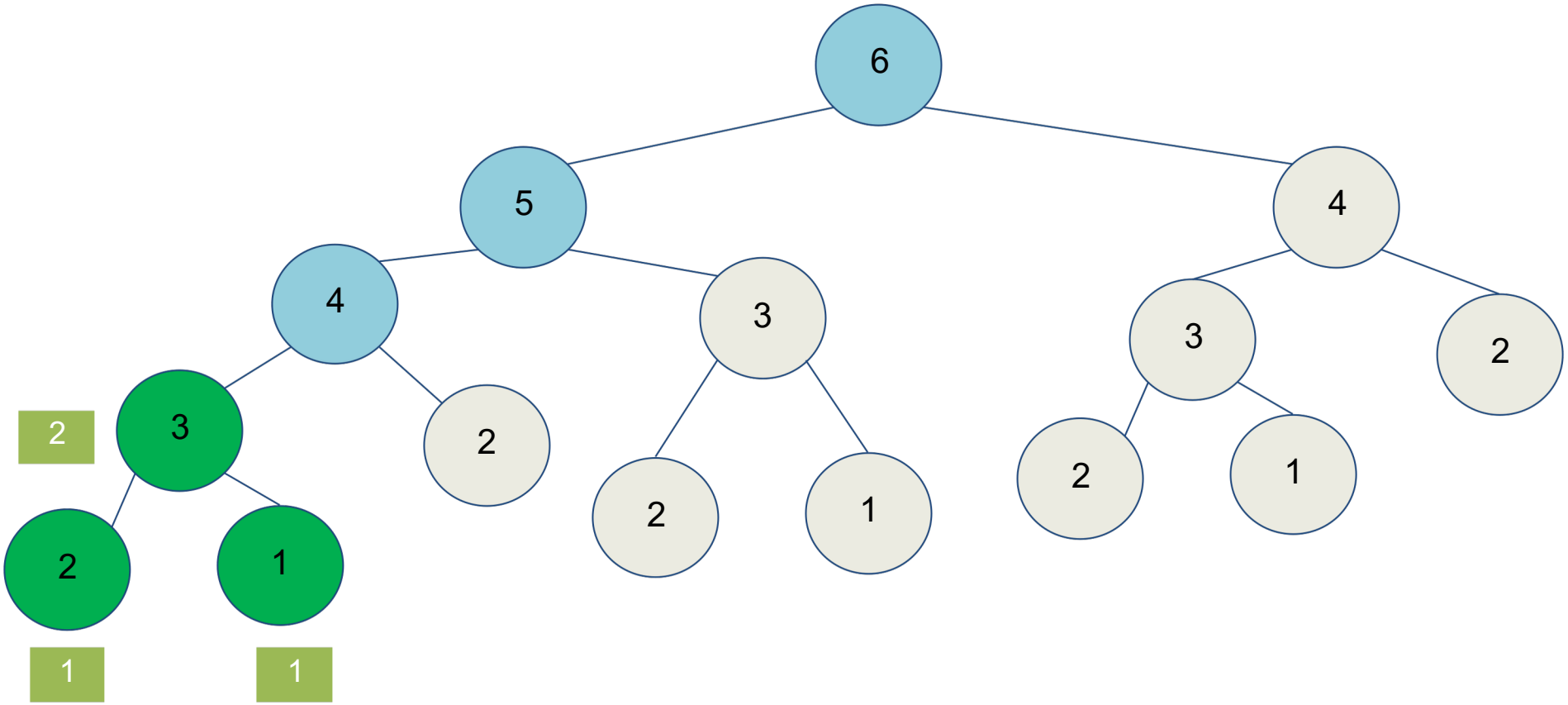
Fibonacci

Ejemplo: Fibonacci(6)



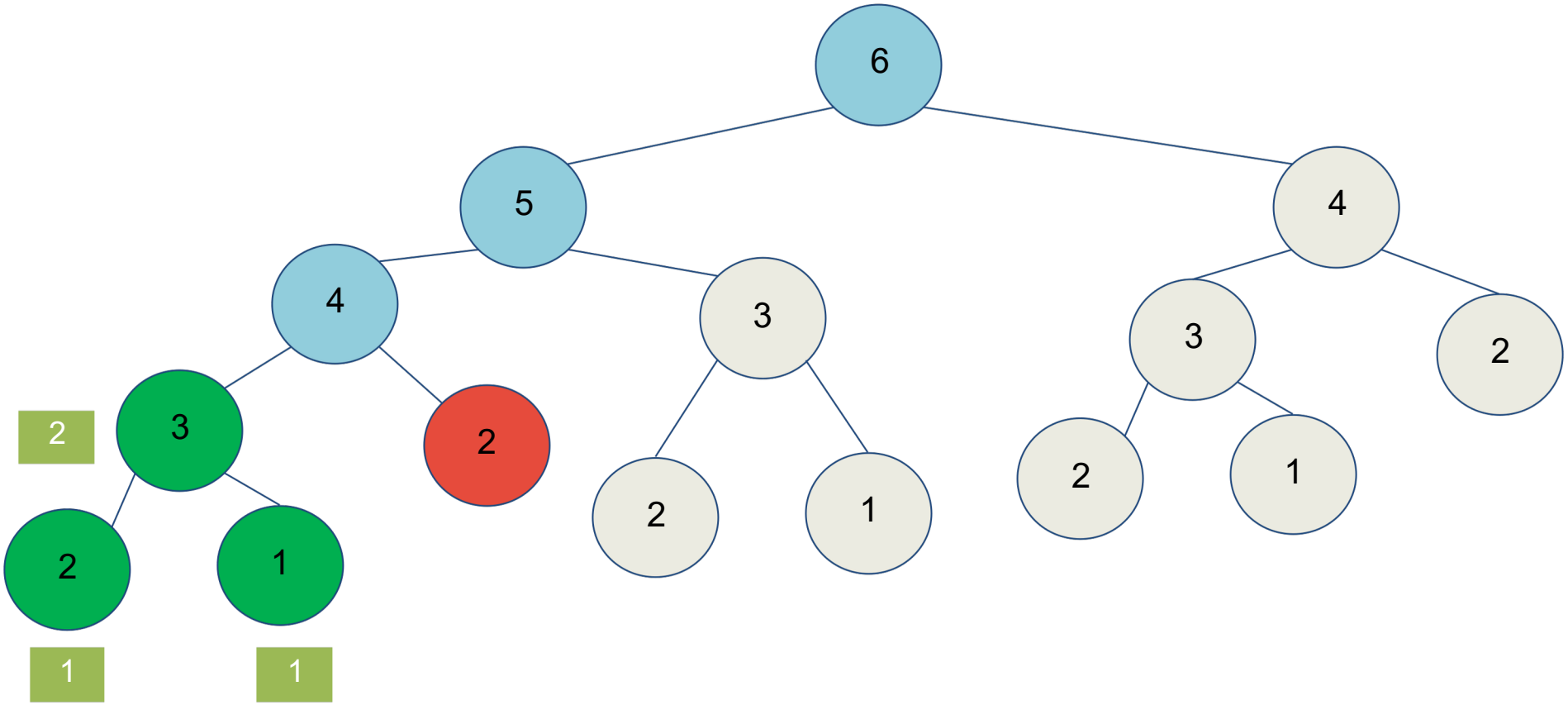
Fibonacci

Ejemplo: Fibonacci(6)



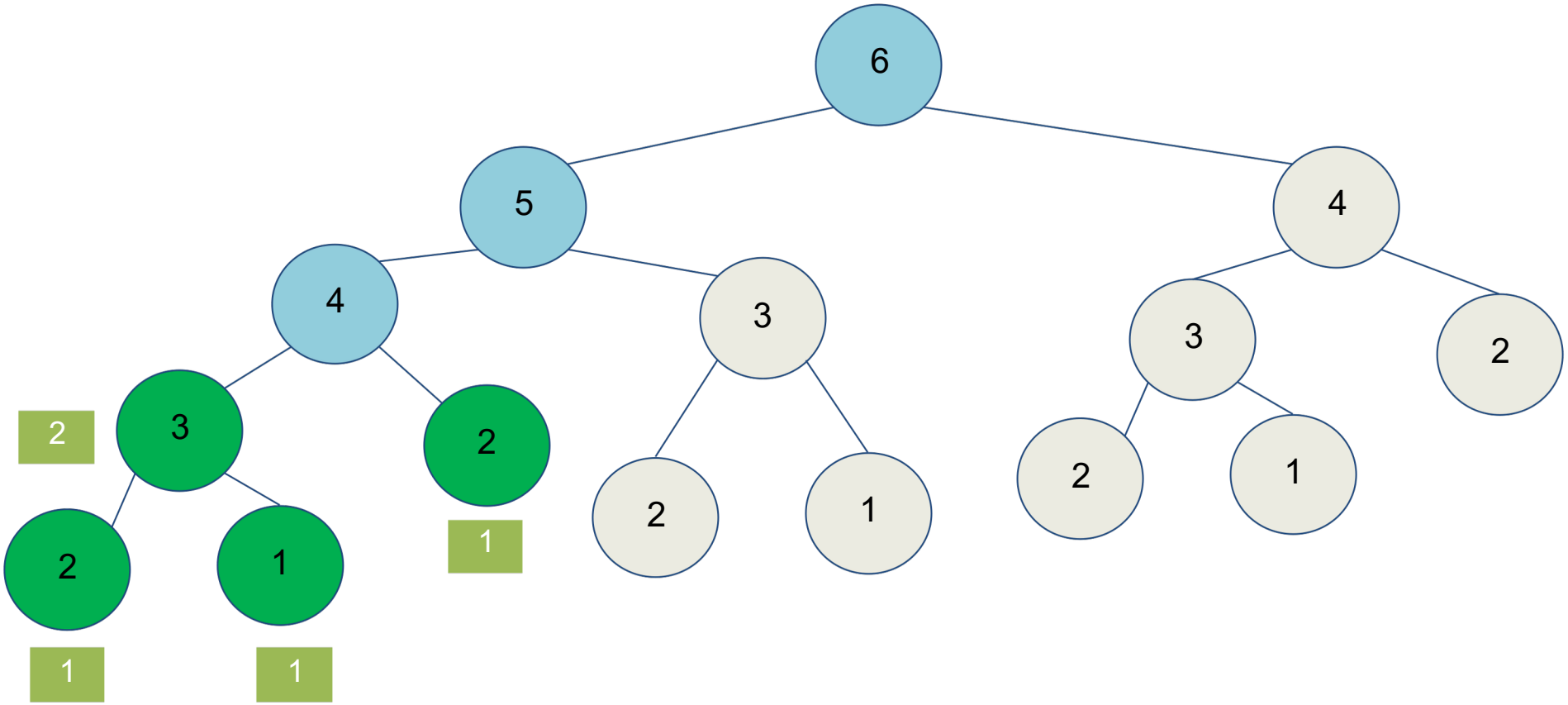
Fibonacci

Ejemplo: Fibonacci(6)



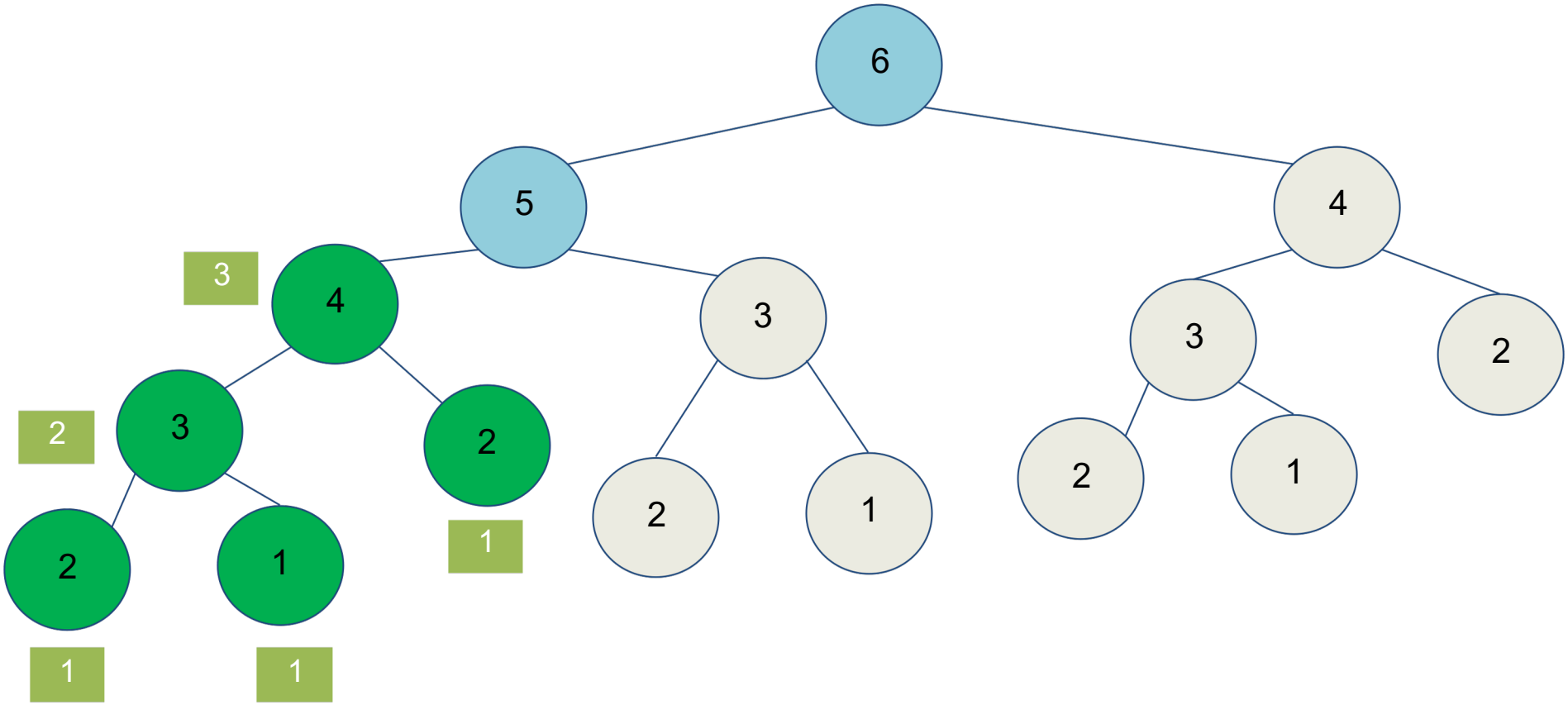
Fibonacci

Ejemplo: Fibonacci(6)



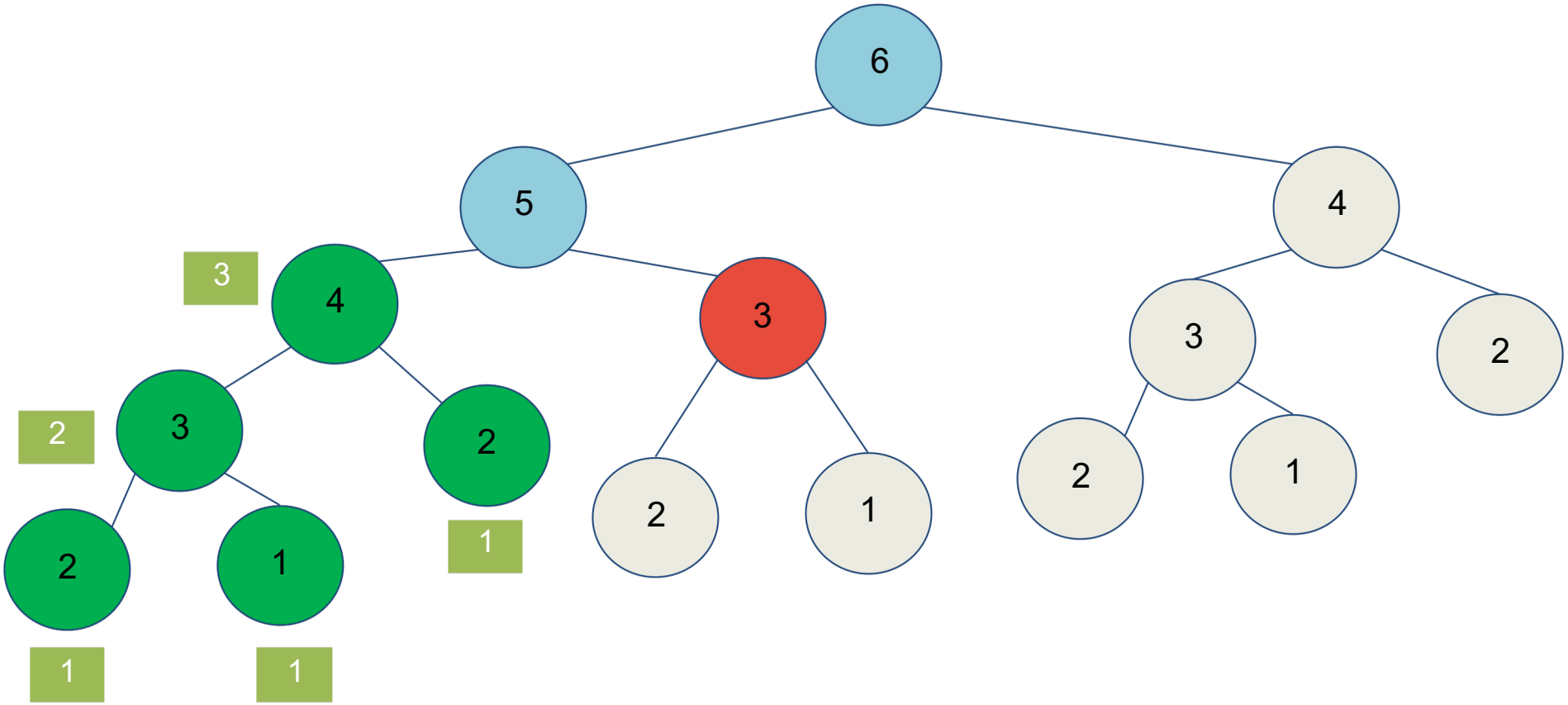
Fibonacci

Ejemplo: Fibonacci(6)



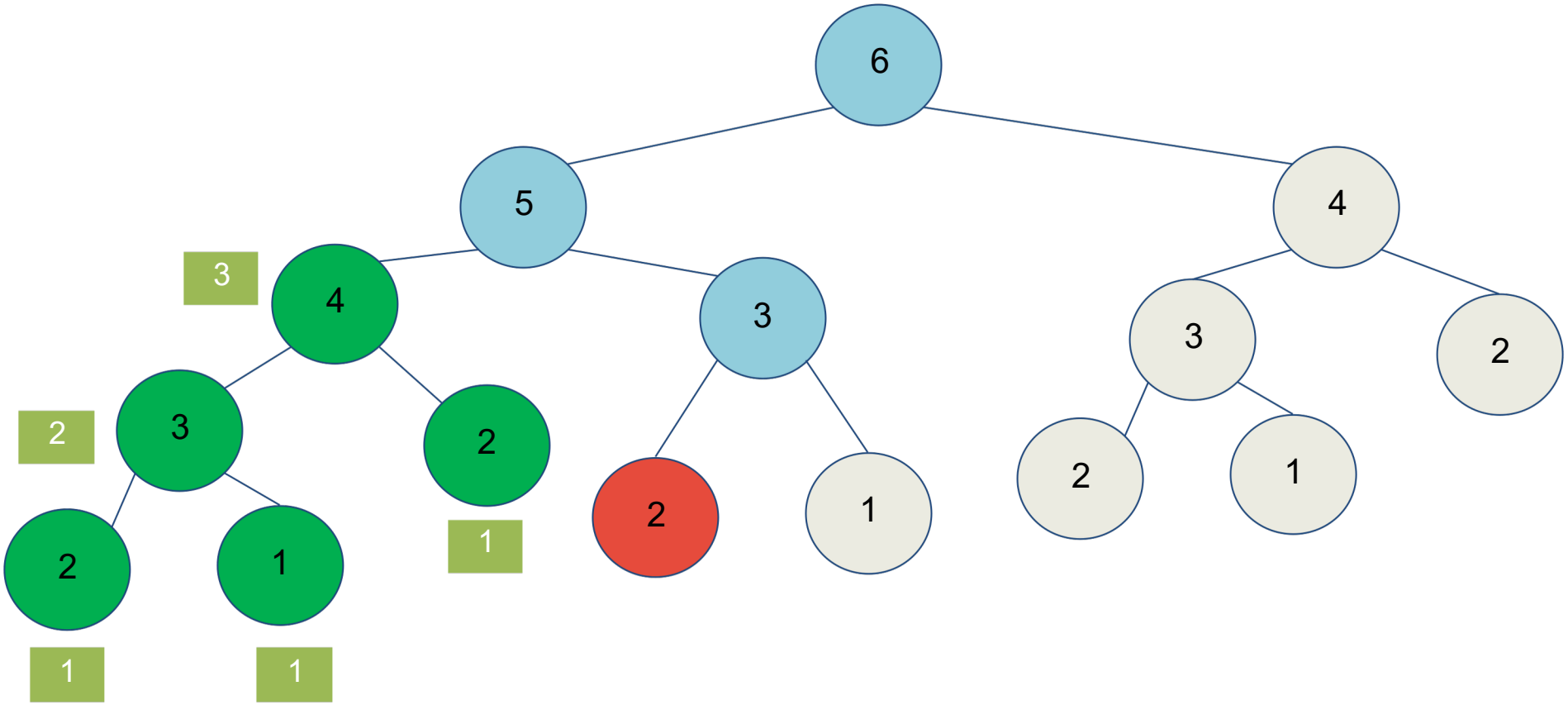
Fibonacci

Ejemplo: Fibonacci(6)



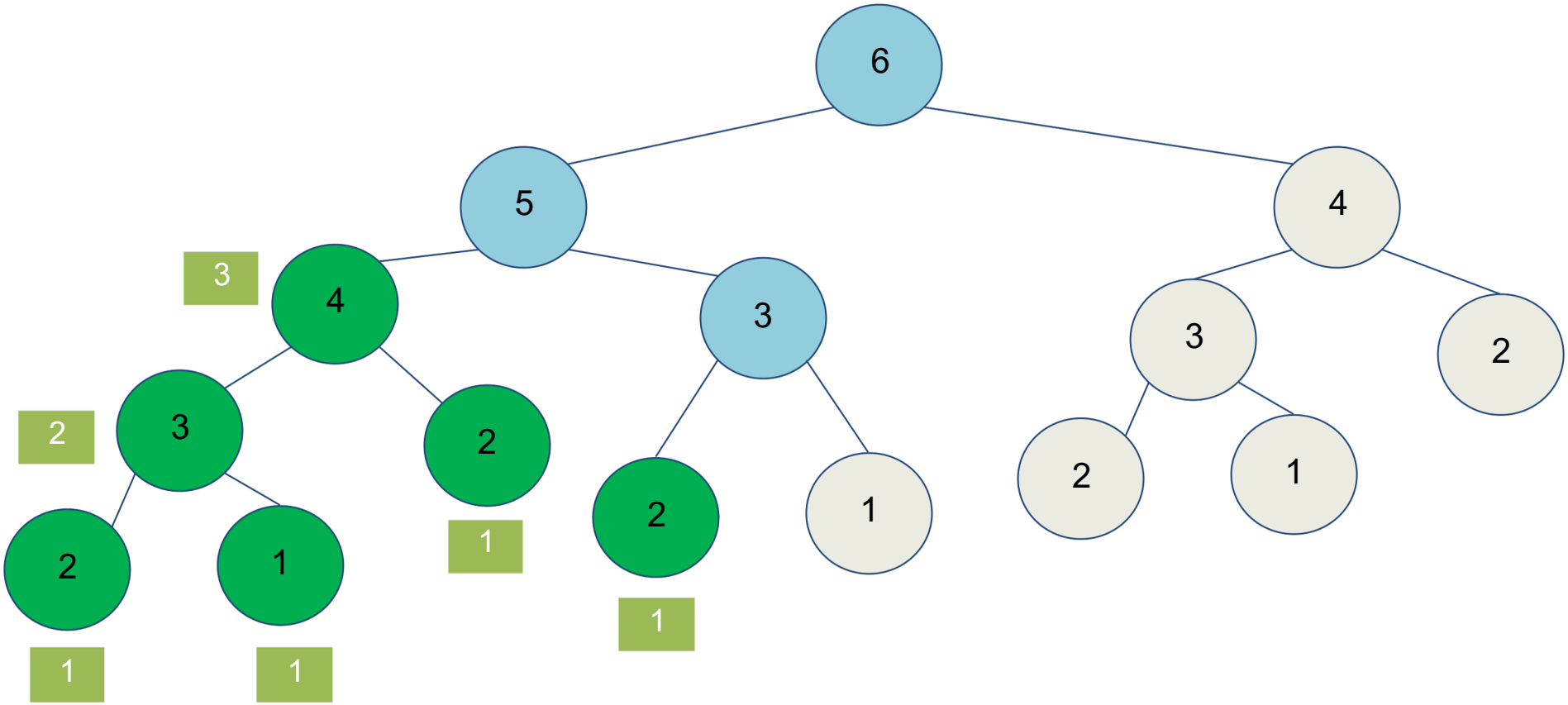
Fibonacci

Ejemplo: Fibonacci(6)



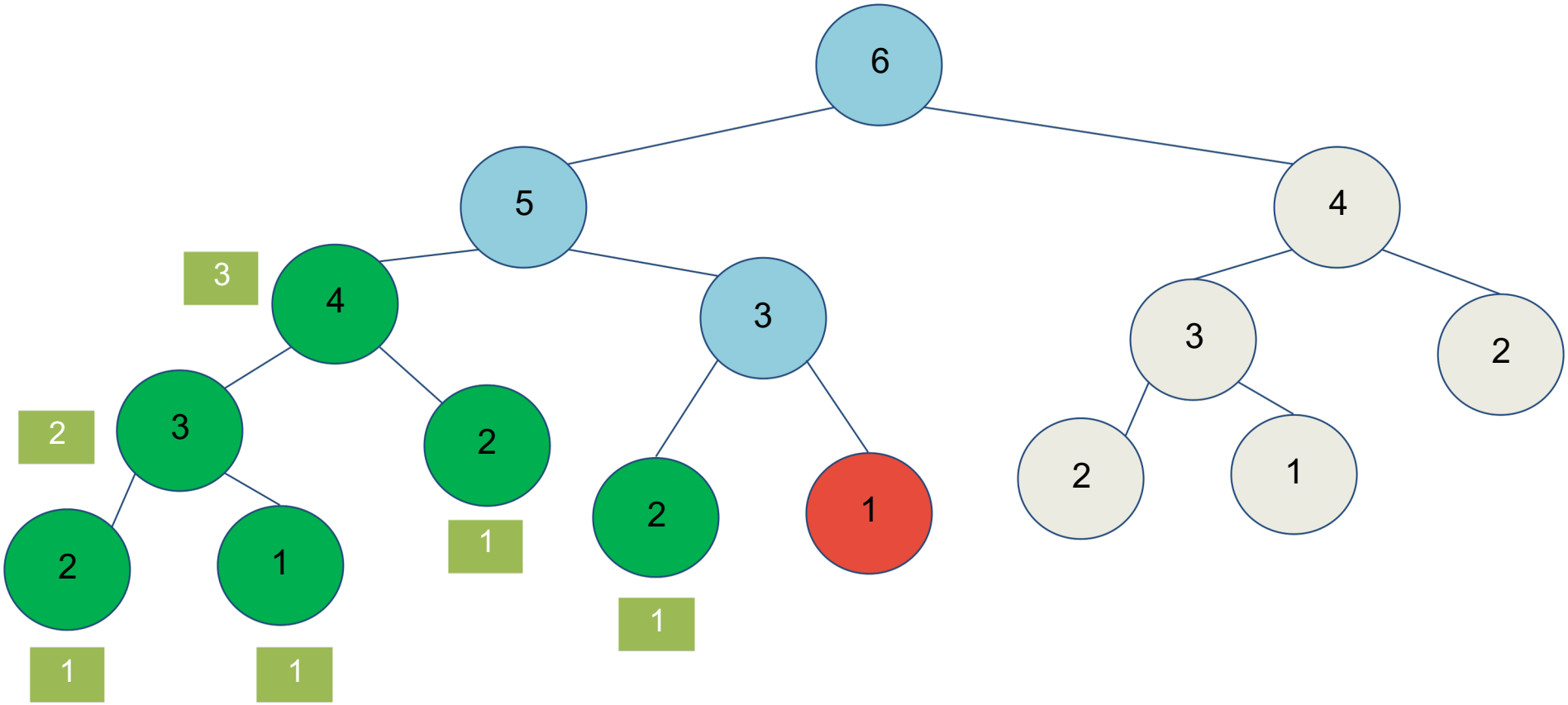
Fibonacci

Ejemplo: Fibonacci(6)



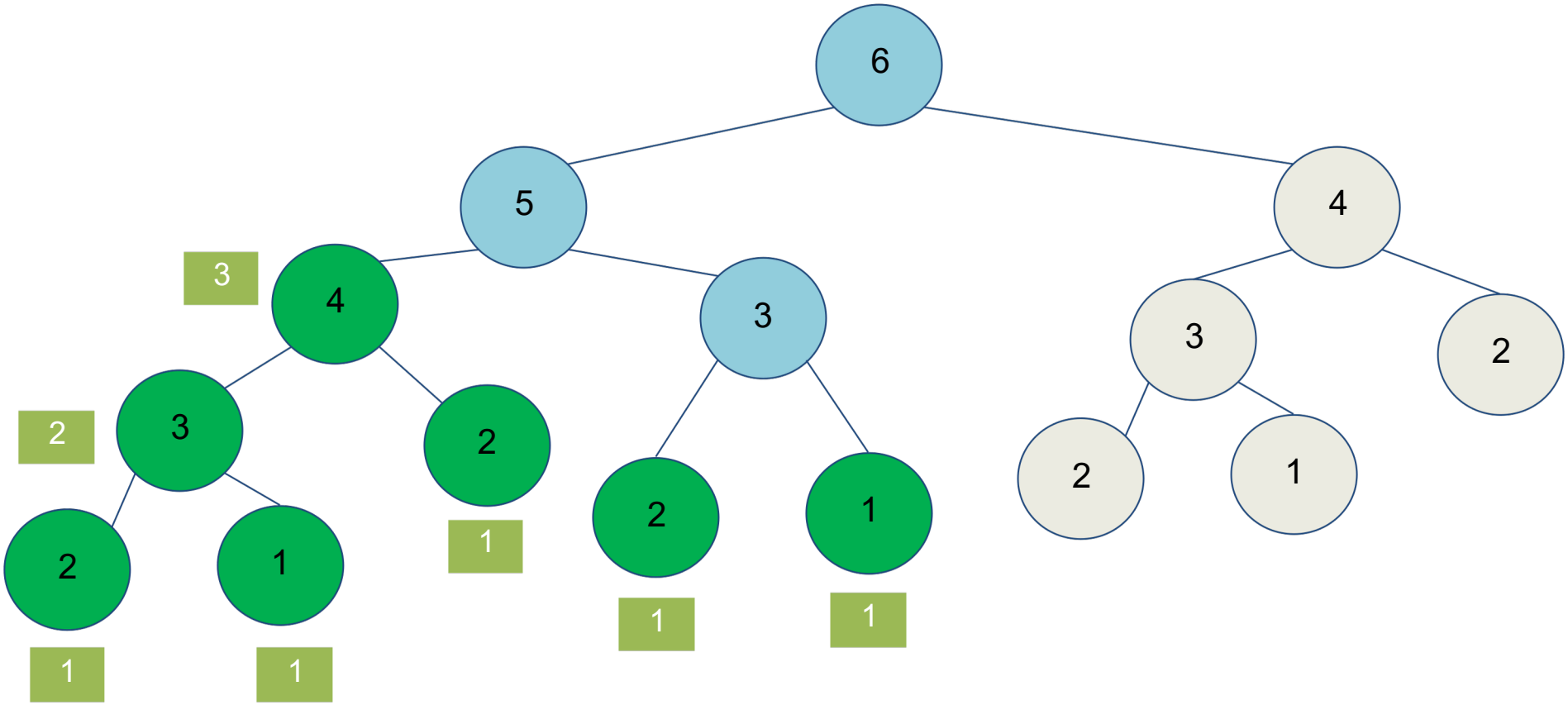
Fibonacci

Ejemplo: Fibonacci(6)



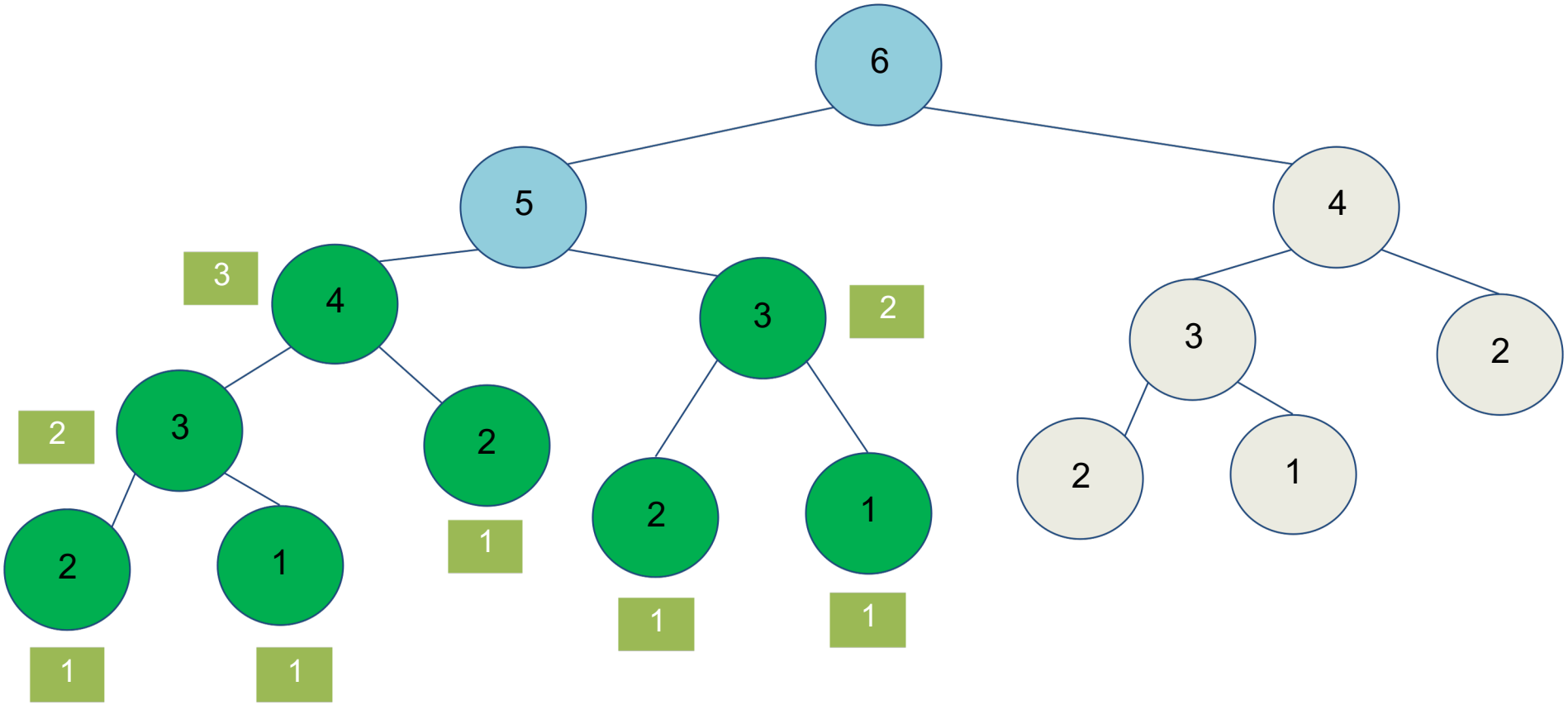
Fibonacci

Ejemplo: Fibonacci(6)



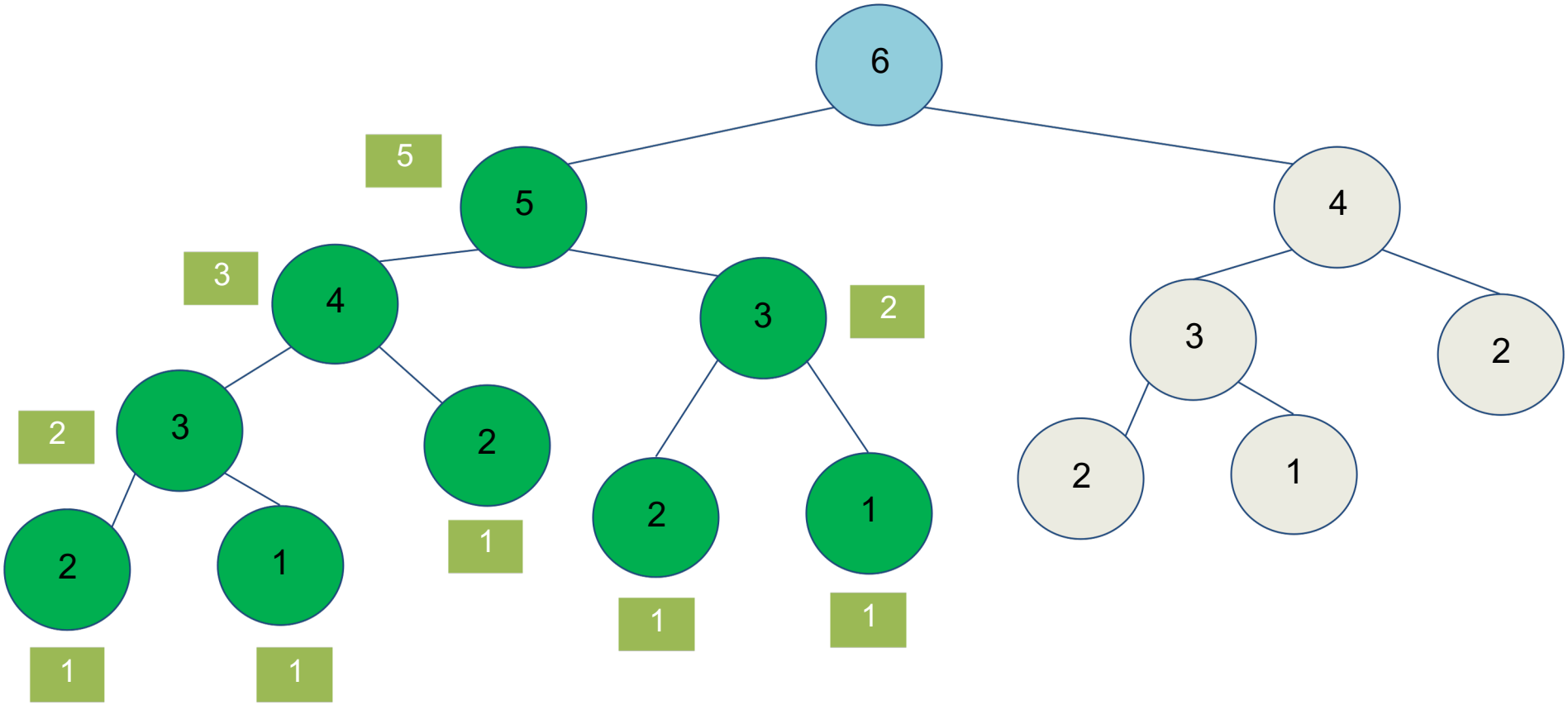
Fibonacci

Ejemplo: Fibonacci(6)



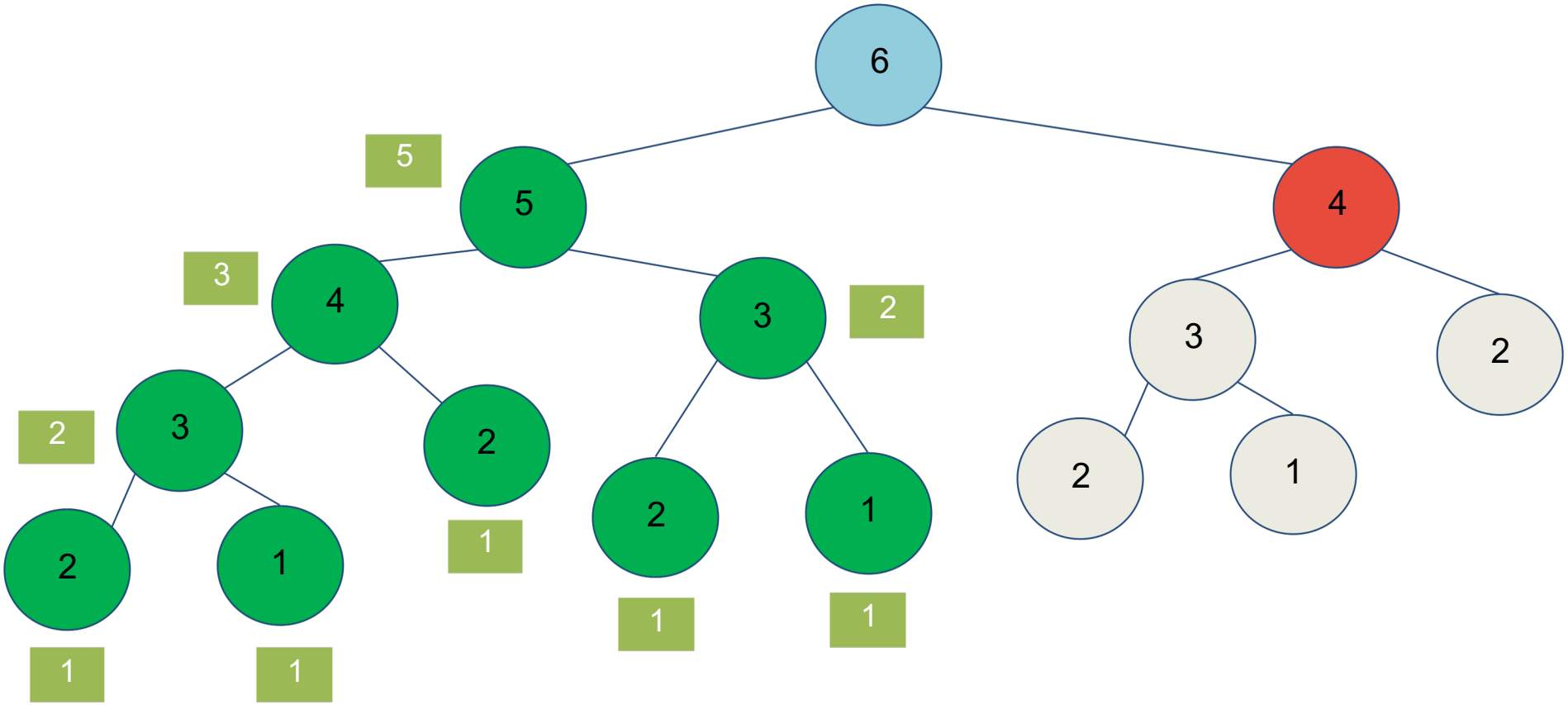
Fibonacci

Ejemplo: Fibonacci(6)



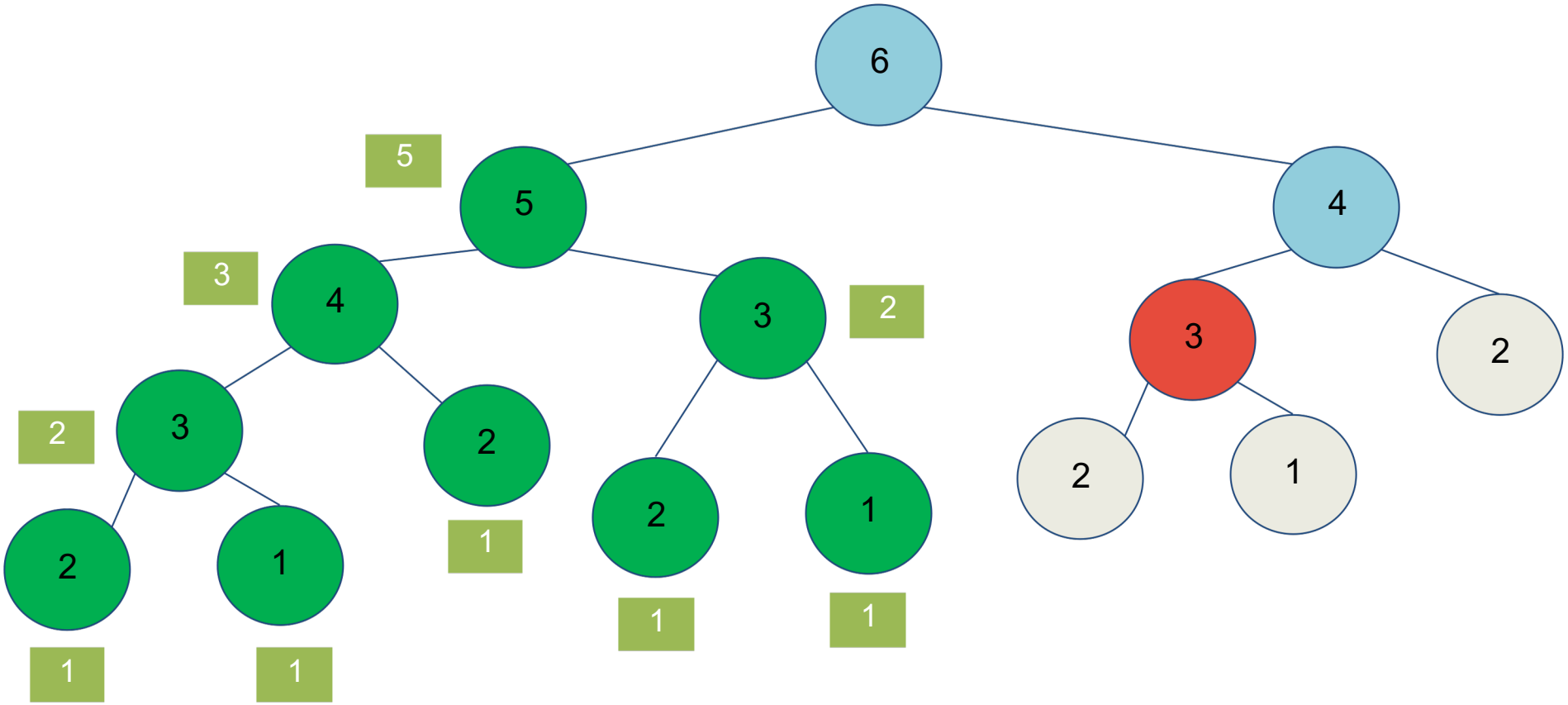
Fibonacci

Ejemplo: Fibonacci(6)



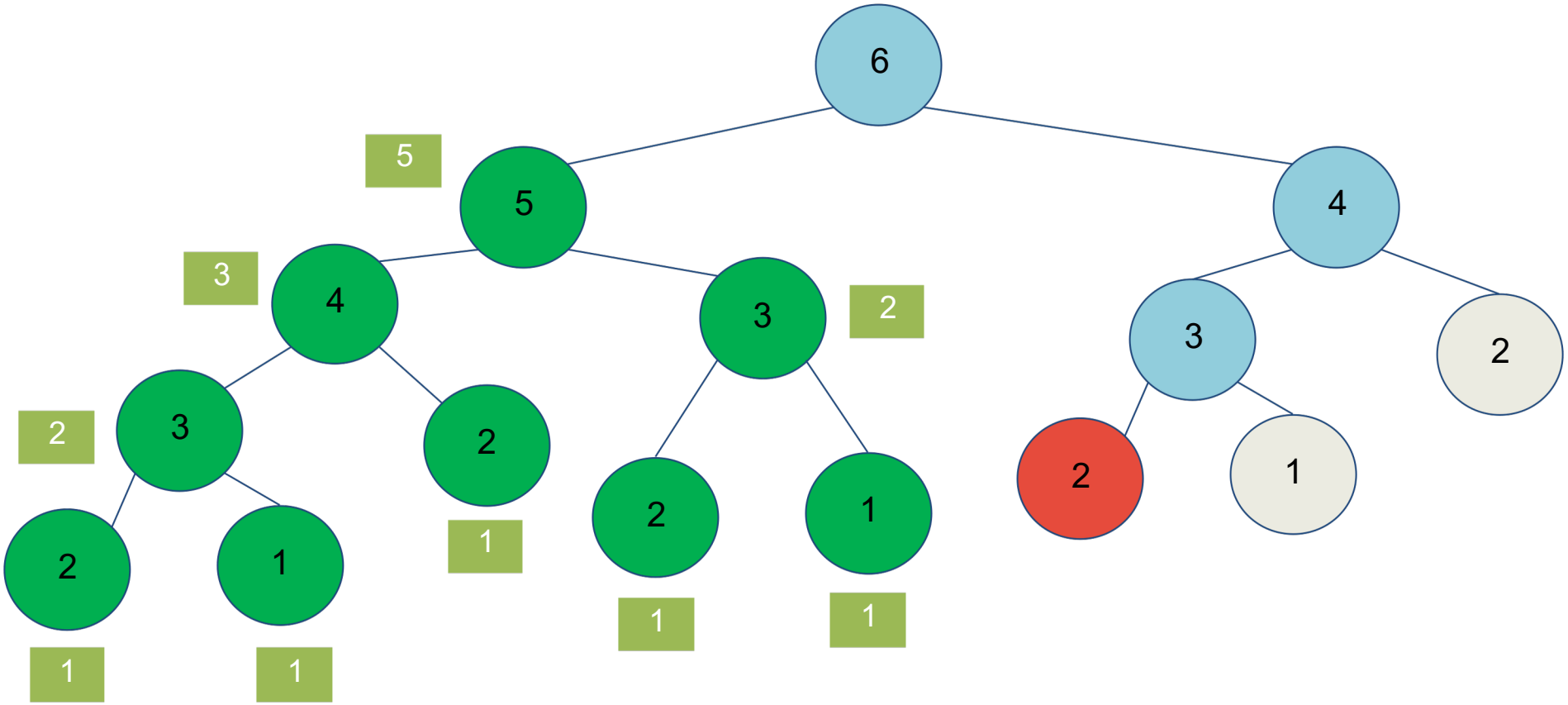
Fibonacci

Ejemplo: Fibonacci(6)



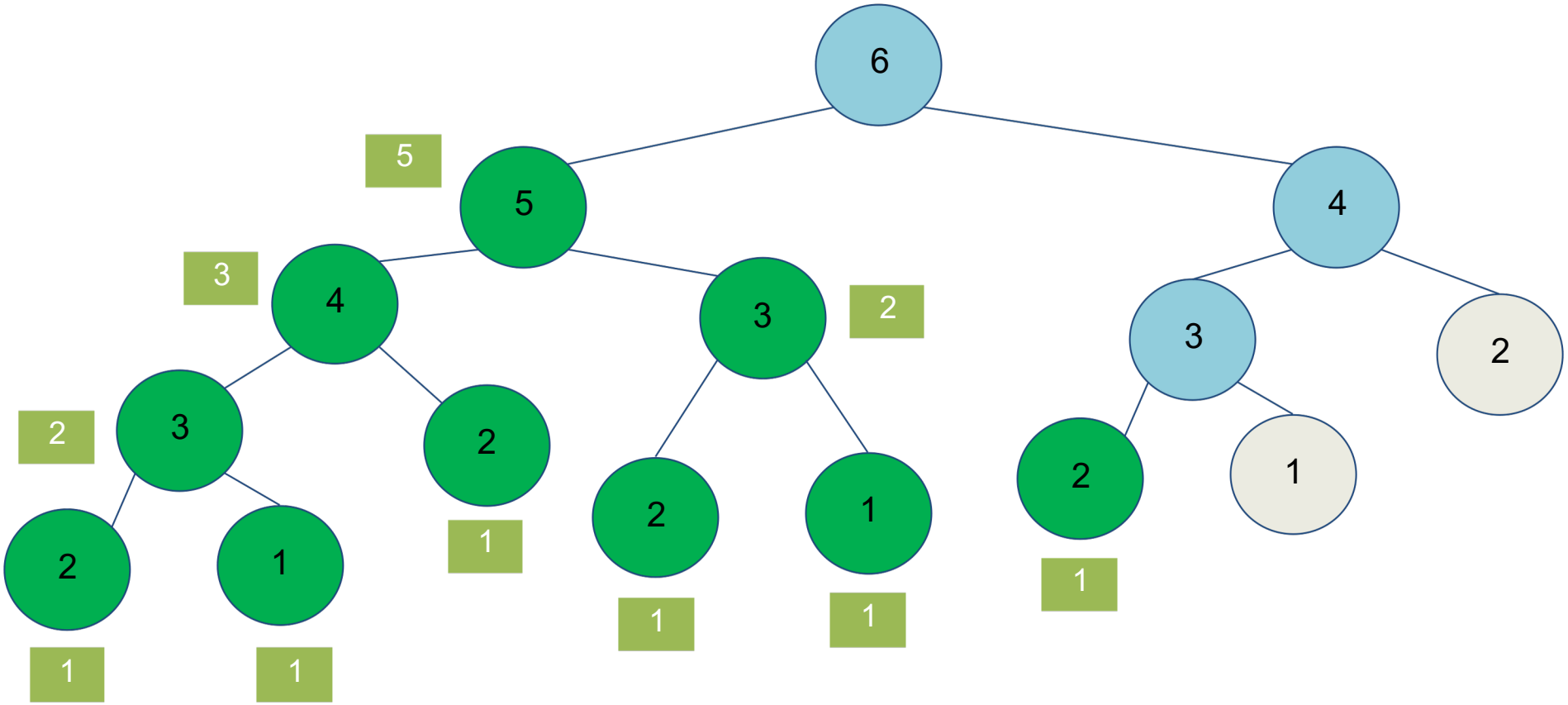
Fibonacci

Ejemplo: Fibonacci(6)



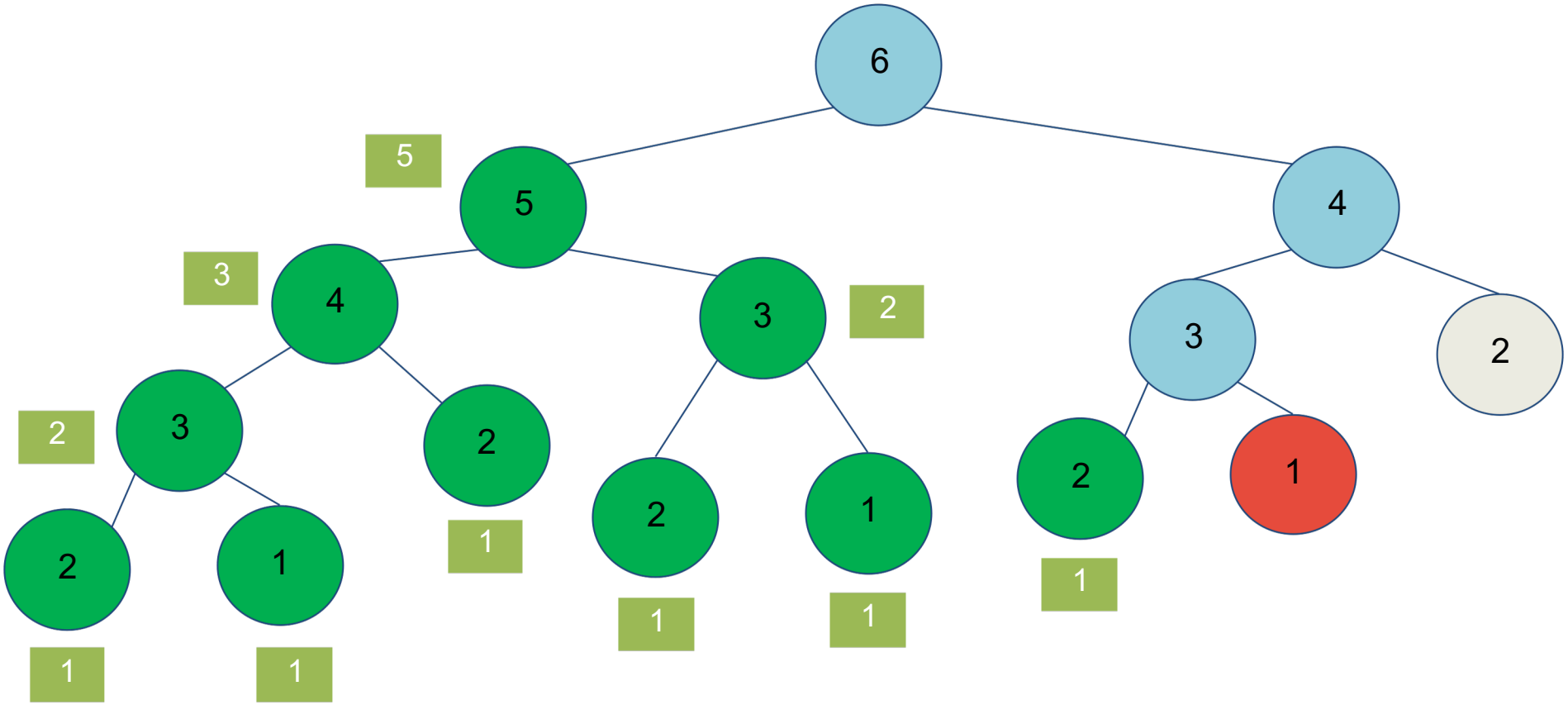
Fibonacci

Ejemplo: Fibonacci(6)



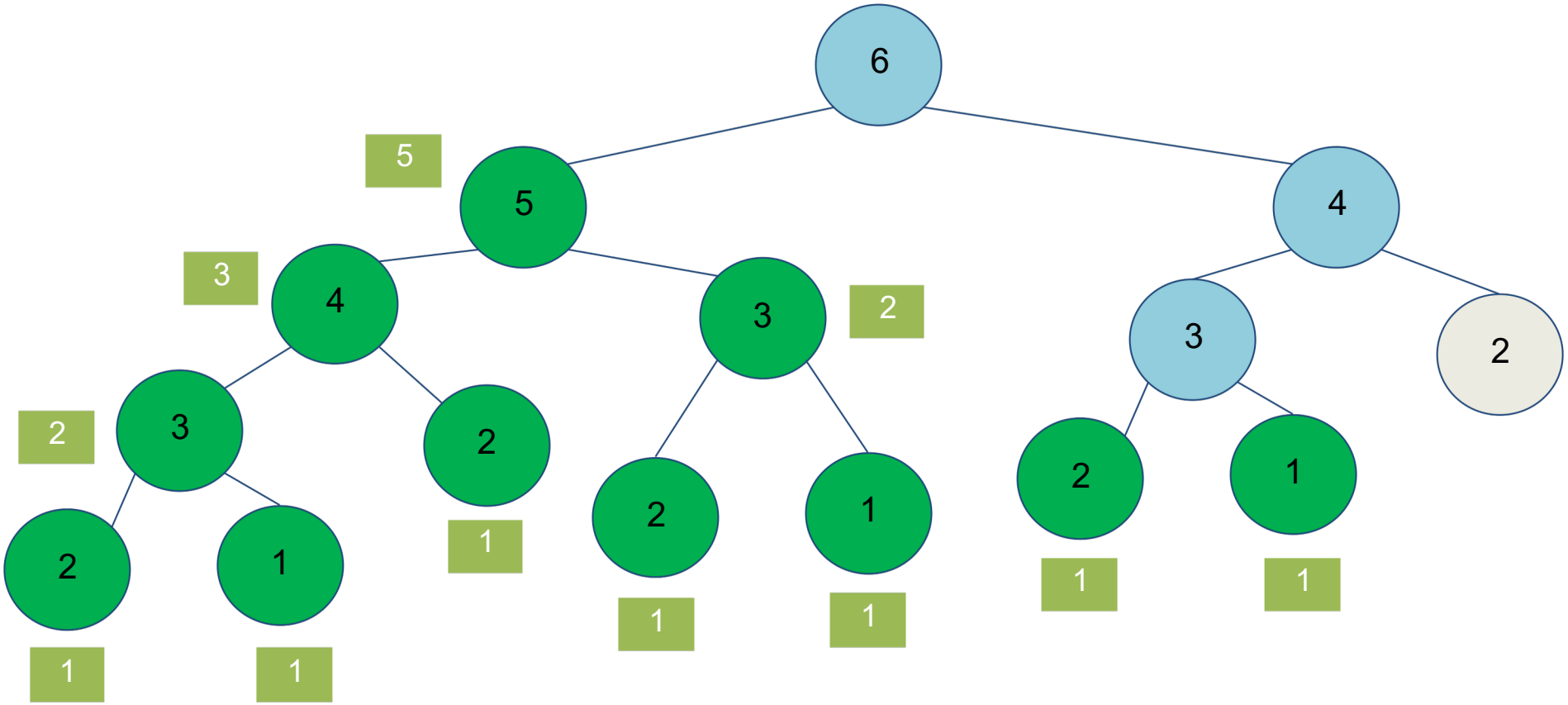
Fibonacci

Ejemplo: Fibonacci(6)



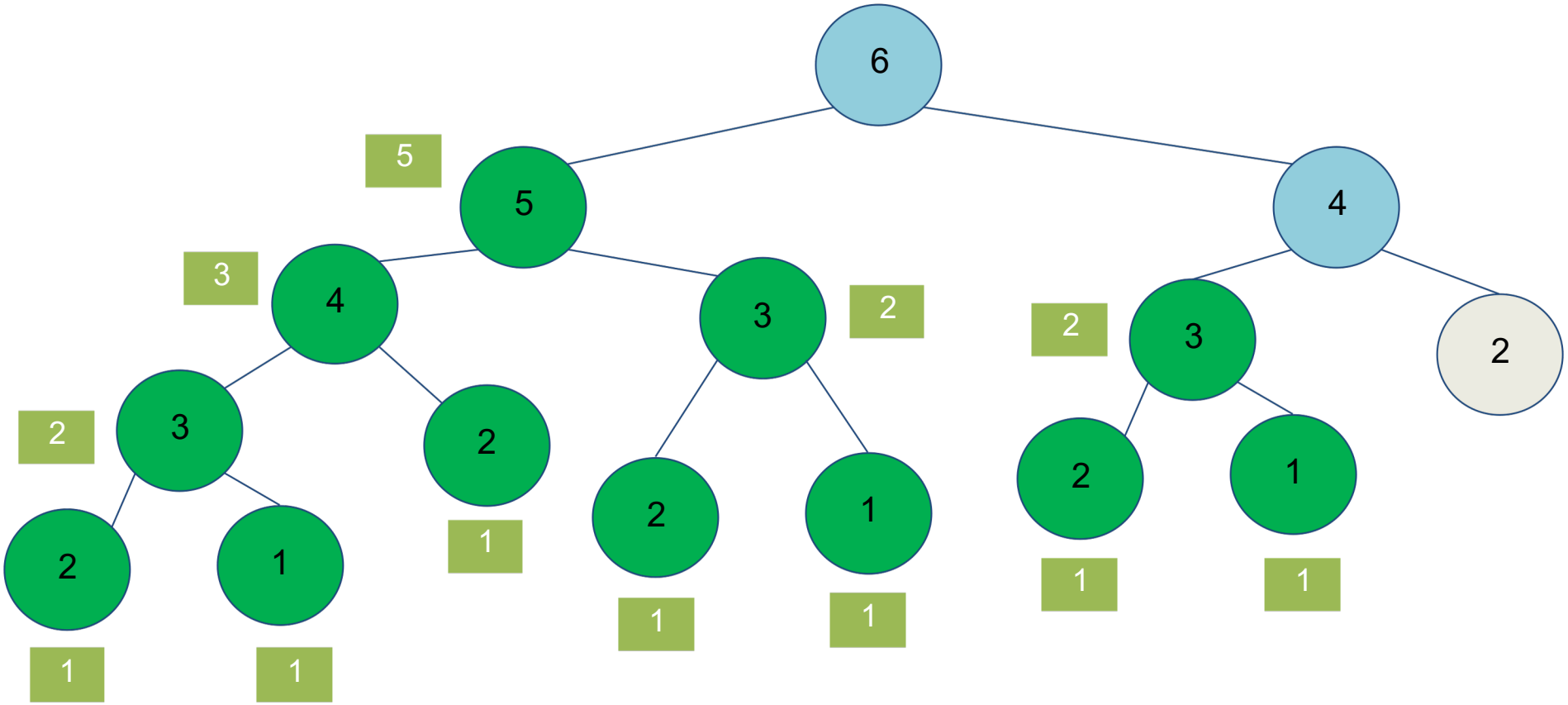
Fibonacci

Ejemplo: Fibonacci(6)



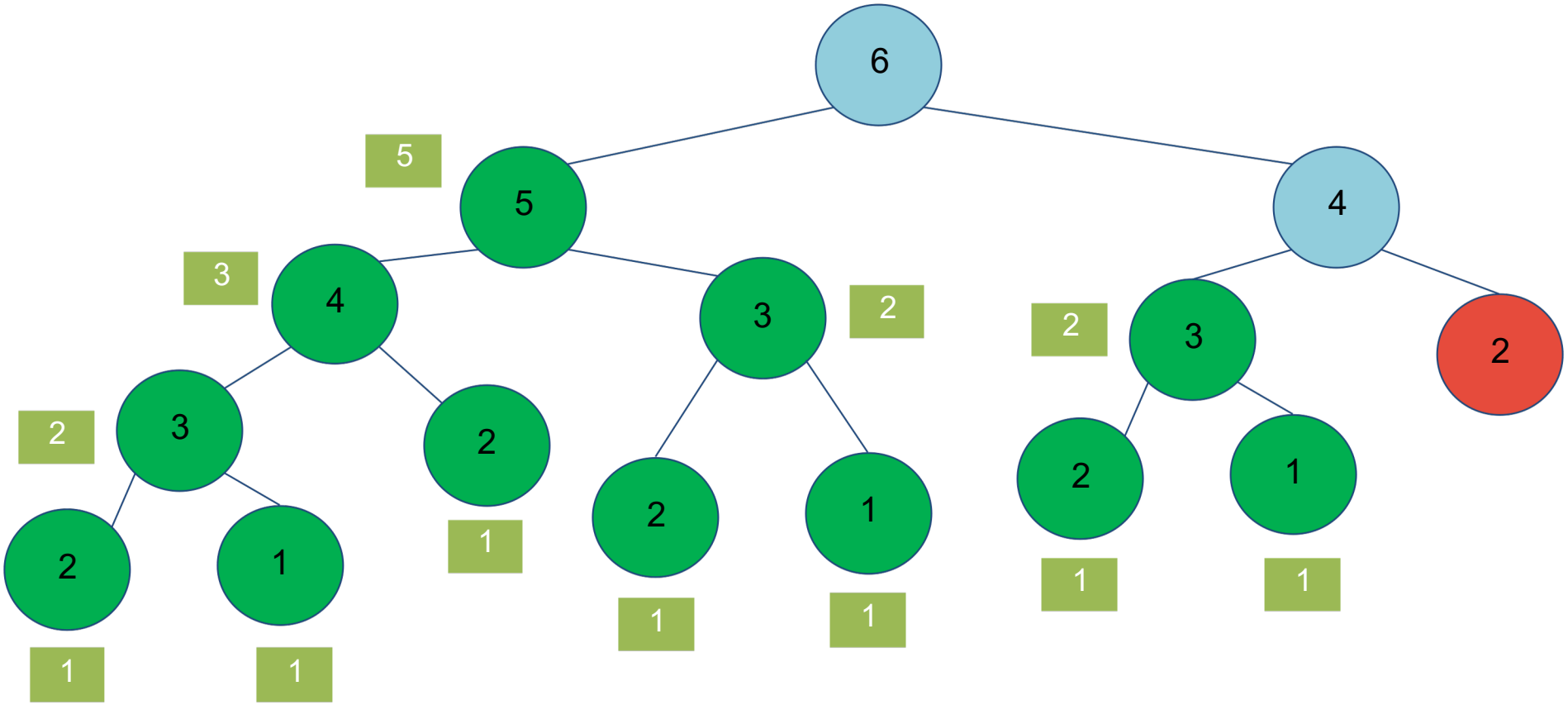
Fibonacci

Ejemplo: Fibonacci(6)



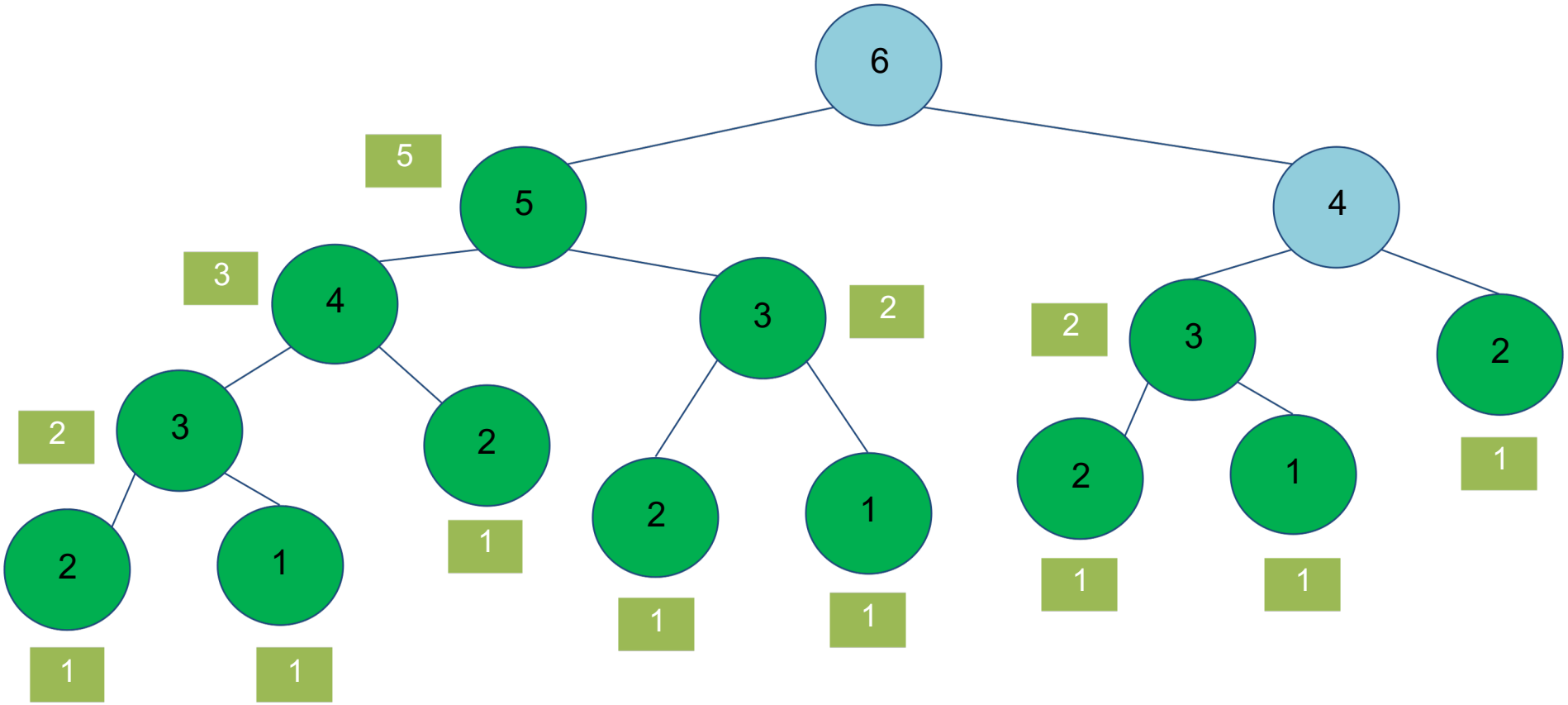
Fibonacci

Ejemplo: Fibonacci(6)



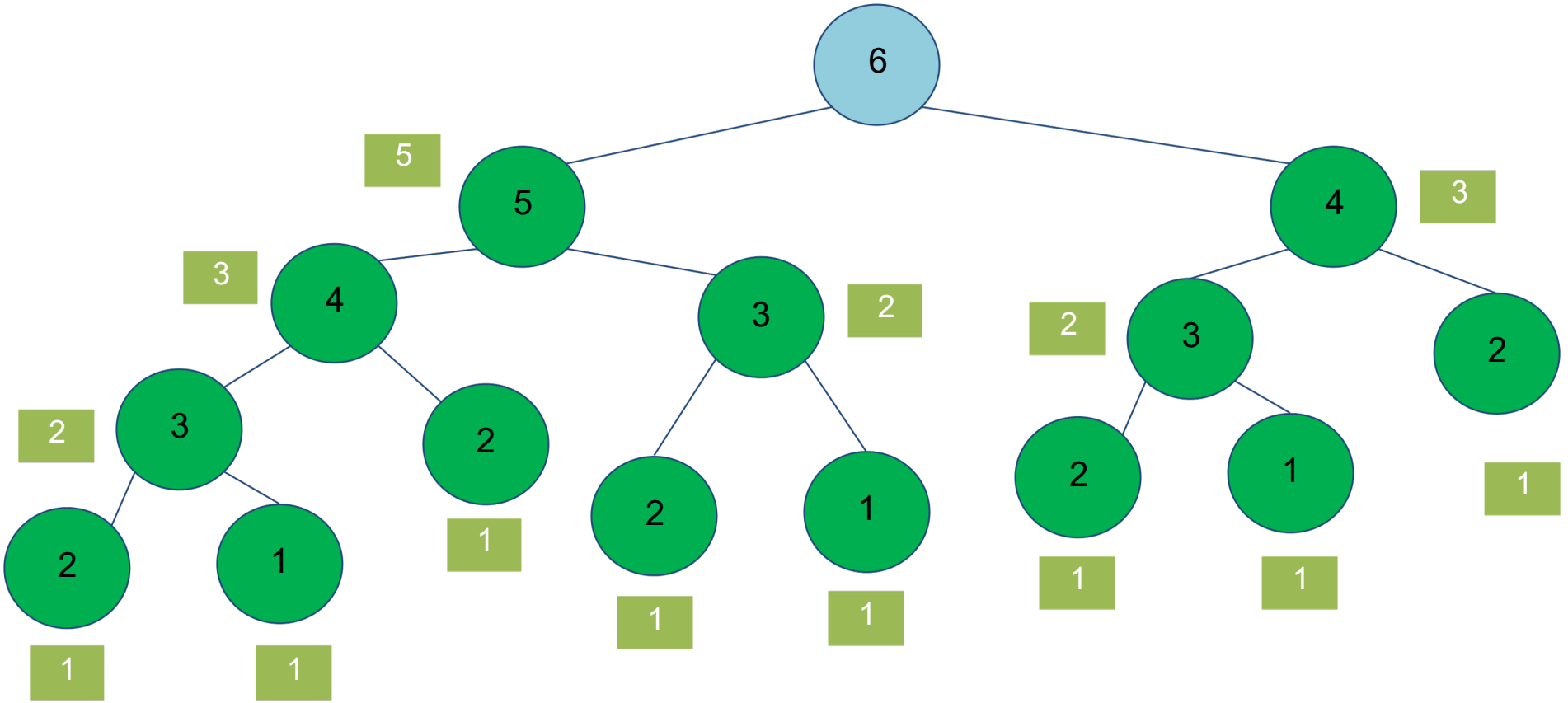
Fibonacci

Ejemplo: Fibonacci(6)



Fibonacci

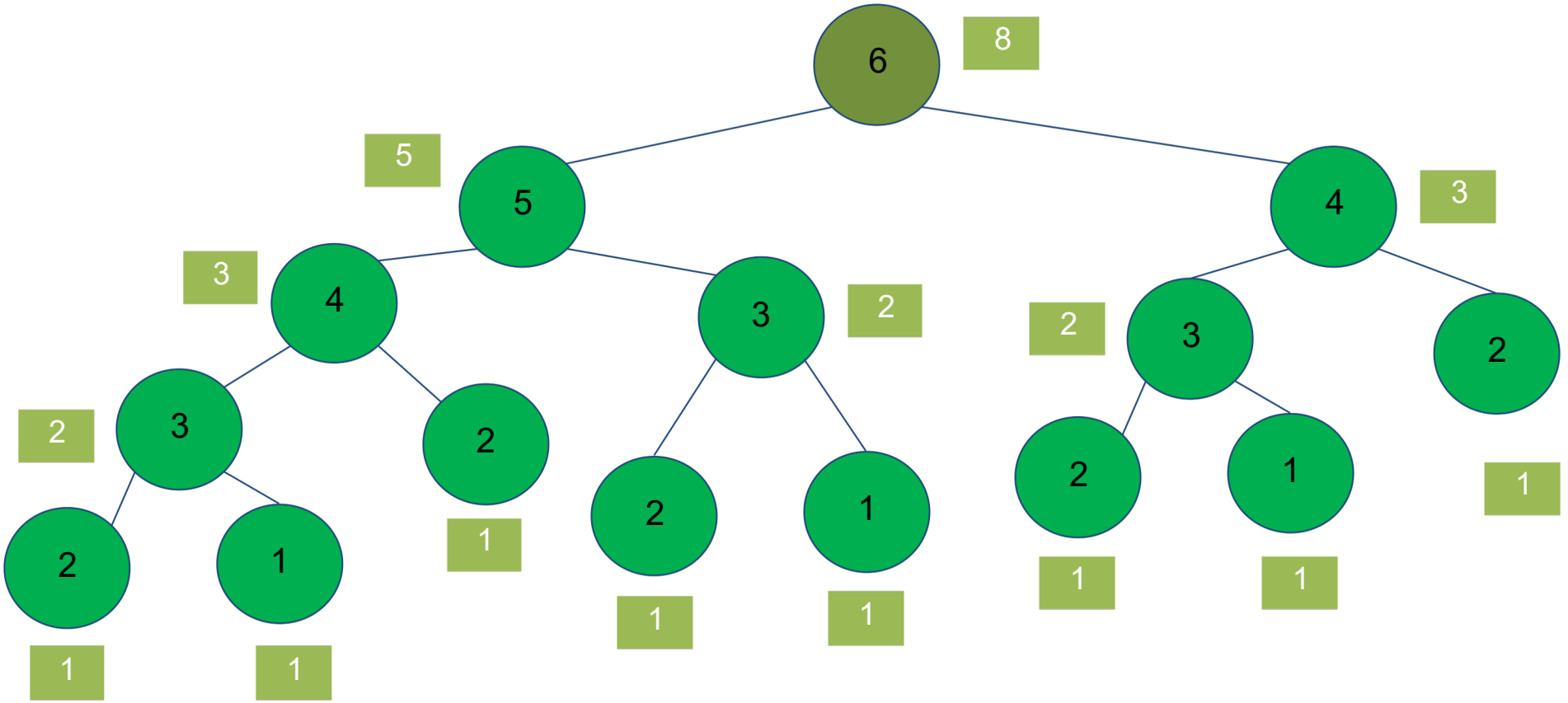
Ejemplo: Fibonacci(6)



Fibonacci

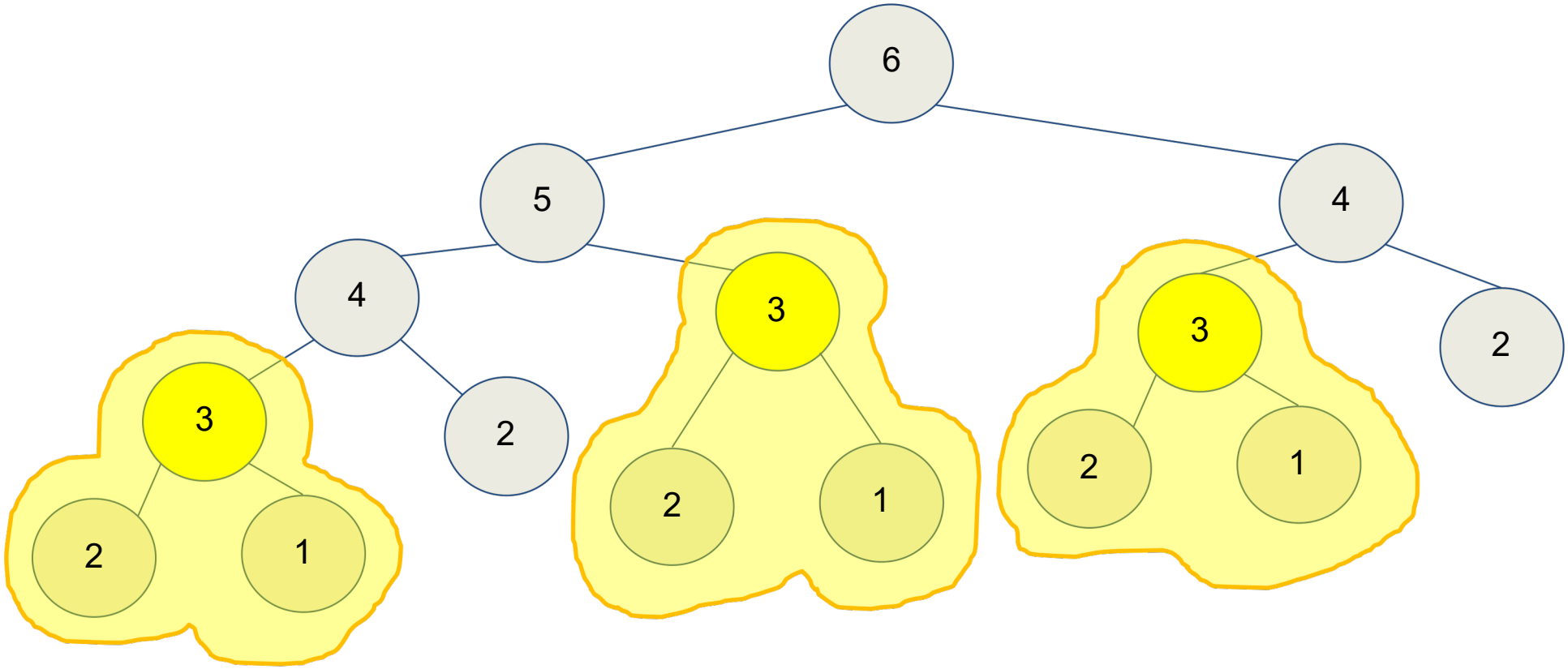
Ejemplo: Fibonacci(6)

Complejidad **exponencial**



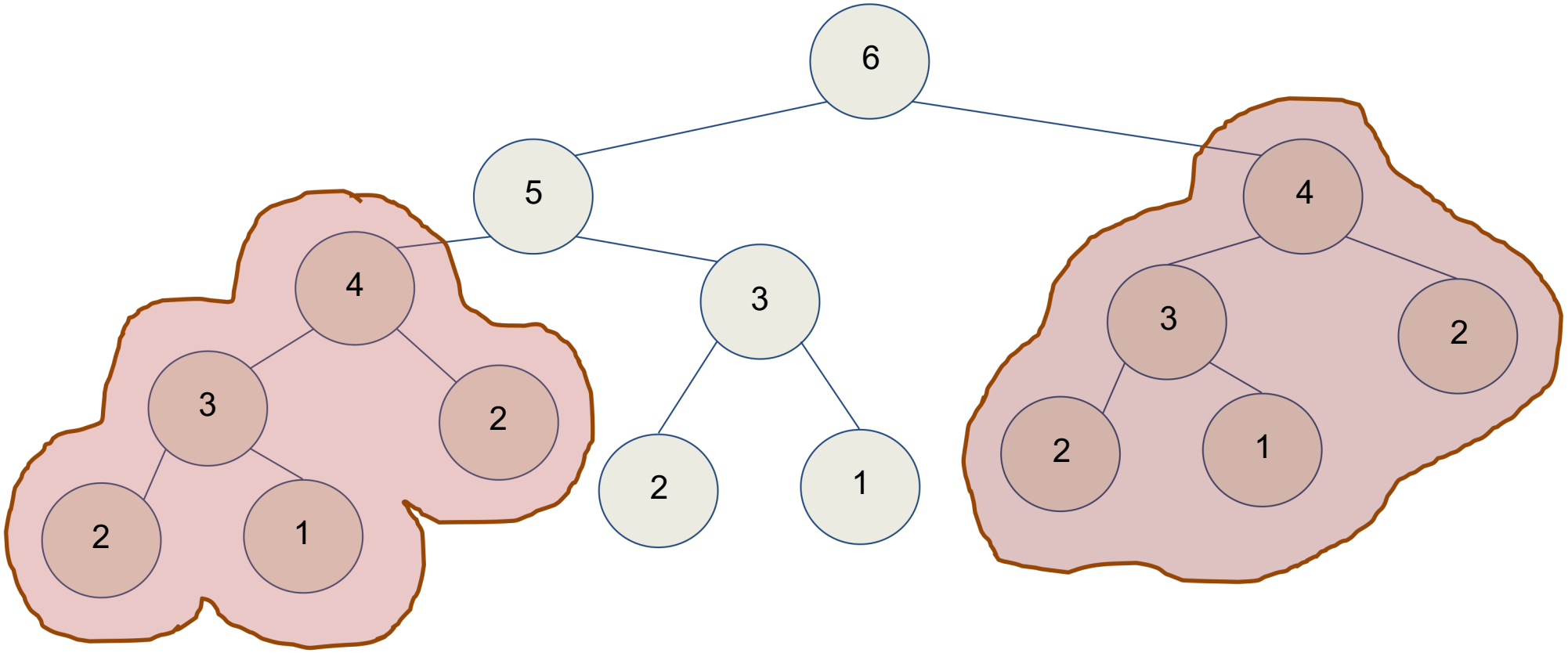
Fibonacci

Ejemplo: Fibonacci(6)



Fibonacci

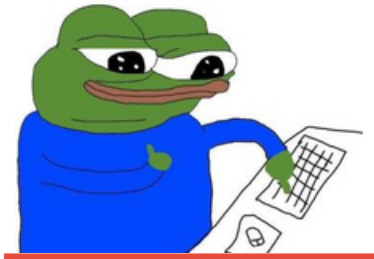
Ejemplo: Fibonacci(6)



Estados repetidos!

- Estamos calculando **varias veces** el mismo número de Fibonacci

¿Qué podemos hacer?



Estados repetidos!

- Estamos calculando **varias veces** el mismo número de Fibonacci

¿Qué podemos hacer? 

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

¿Qué es la Programación Dinámica?

- Paradigmas de la Programación:

  >> Fuerza Bruta
  >> Greedy (voraz)
  >> Divide y Vencerás
  >> **Programación Dinámica**

- **Memorizar** resultados calculados para reutilizarlos
- Problemas clásicos: maximizar, minimizar, contar caminos, …

DP desde un approach recursivo (Top-Down)

- Desde este enfoque, un algoritmo de programación dinámica es:

DP = **RECURSIÓN** + **MEMORIZACIÓN**

DP desde un approach recursivo (Top-Down)

- ¿Cómo construir un algoritmo de DP?

DP desde un approach recursivo (Top-Down)

- ¿Cómo construir un algoritmo de DP?

  1. Crear una solución **recursiva** del problema

DP desde un approach recursivo (Top-Down)

- ¿Cómo construir un algoritmo de DP?

1. Crear una solución **recursiva** del problema

solucion(x_1, x_2,…,xn):
if(caso base):
return ------
llamadas recursivas

DP desde un approach recursivo (Top-Down)

- ¿Cómo construir un algoritmo de DP?

  1. Crear una solución **recursiva** del problema.
  2. Identificar los **estados** que se repiten (para los cuales la solución es idéntica)

DP desde un approach recursivo (Top-Down)

- ¿Cómo construir un algoritmo de DP?

  1. Crear una solución **recursiva** del problema.
  2. Identificar los **estados** que se repiten (para los cuales la solución es idéntica)
  3. Añadir la **memoria** en la que almacenar las soluciones ya calculadas y utilizarla en caso de estar en un estado precalculado.

DP desde un approach recursivo (Top-Down)

- ¿Cómo construir un algoritmo de DP?

solucion(x_1, x_2,…,xn):
  if(**caso base**):
    return ------
  if(estado ya calculado):
    return **memo(estado)**
  llamadas recursivas
  actualizar **memo**

1. Crear una solución **recursiva**.
2. Identificar los **estados** que se repiten.
3. Añadir la **memorización**.

DP desde un approach recursivo (Top-Down)

- ¿Cuál es la complejidad de un algoritmo de DP?

O(nºestados * nº llamadas recursivas)

Ejemplo clásico: la sucesión de Fibonacci

- Cómo utilizaríamos DP para resolver el problema de Fibonacci?

Ejemplo clásico: la sucesión de Fibonacci

Enlace en Telegram

Ejemplo clásico: la sucesión de Fibonacci

- Solución de programación dinámica:

Ejemplo clásico: la sucesión de Fibonacci

- Solución de programación dinámica:

1. Crear una solución **recursiva**.

```
fibonacci(i):
    if(i==1 OR i==2):
        return 1
    return fibonacci(i-1) + fibonacci(i-2)
```

Ejemplo clásico: la sucesión de Fibonacci

- Solución de programación dinámica:

  2. Identificar los **estados** que se repiten.

     i-ésimo nº de Fibonacci

Ejemplo clásico: la sucesión de Fibonacci

- Solución de programación dinámica:

3. Añadir la **memorización**.

```
fibonacci(i):
    if(i==1 OR i==2):
        return 1
    if(memo[i]!=-1):
        return memo[i]
    memo[i]= fibonacci(i-1) + fibonacci(i-2)
    return memo[i]
```

Ejemplo clásico: la sucesión de Fibonacci

- ¿Cuál es la complejidad del algoritmo?

Ejemplo clásico: la sucesión de Fibonacci

- ¿Cuál es la complejidad del algoritmo?

O(nº estados * nº llamadas recursivas)

- Para calcular el n-ésimo nº de Fibonacci, pasamos por los n primeros números: **n estados**
- Cada llamada realiza **2 llamadas recursivas**

↓

O(2n) ~ O(n)

- Hemos pasado de una complejidad **exponencial** a **lineal**

El problema de la Mochila

El problema de la Mochila

- Dados:

1. una mochila  con **capacidad máxima** w
2. n objetos con un **valor** y **peso** determinados

- ¿Cuál es la suma de valores máxima que podemos meter en la mochila?

El problema de la Mochila

- **Ejemplo:**

1. Mochila**:**

Peso máximo: 5

2. Objetos disponibles:

peso	2	3	2	4
valor	3	2	1	4

El problema de la Mochila

- **¿Enfoque Greedy? (voraz):**

El problema de la Mochila

- **¿Enfoque Greedy? (voraz):**

  - 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

- **¿Enfoque de fuerza bruta?:**
  - Probar todas las posibles combinaciones
- **¿Cómo?**
  - Colocando los objetos en fila, decidir si **COGER** o **NO COGER** cada uno de ellos.

El problema de la Mochila

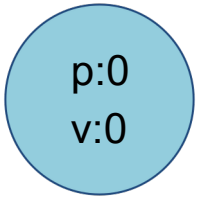
	1	2	3	4
				
p	2	2	4	4
v	3	2	5	4

Peso : 0

Valor : 0

El problema de la mochila

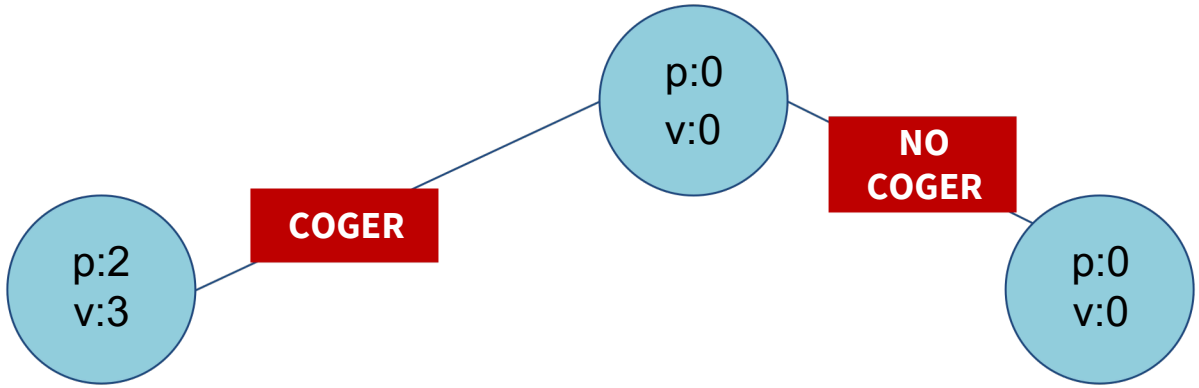
1



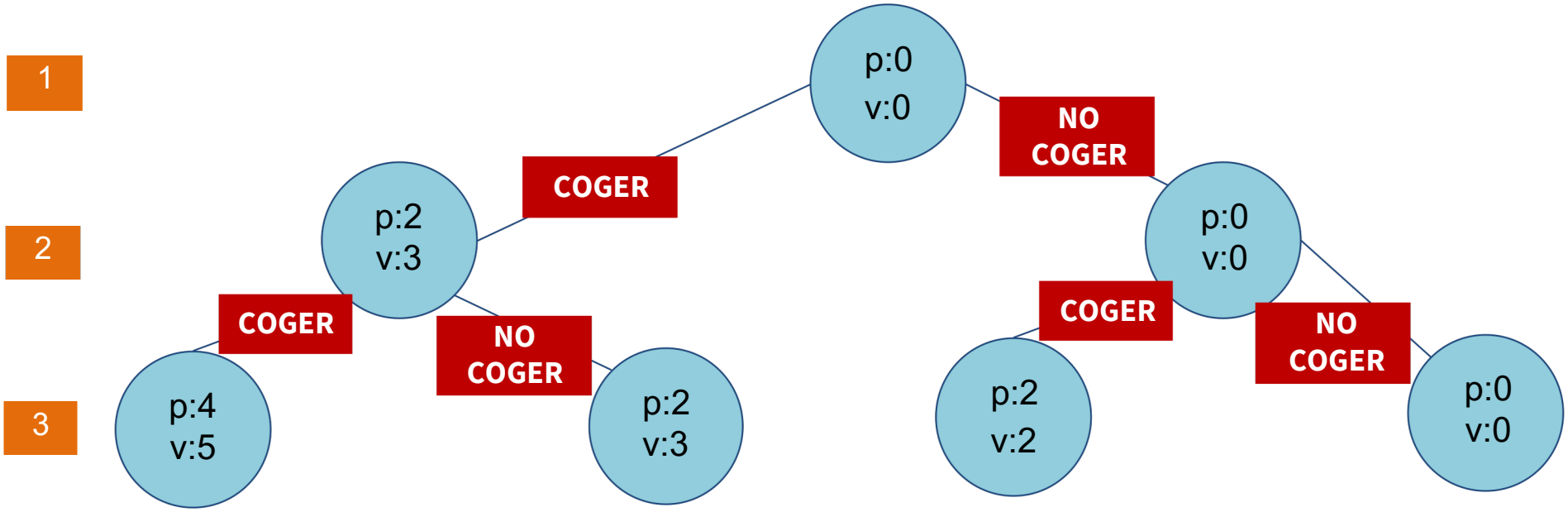
El problema de la mochila

1

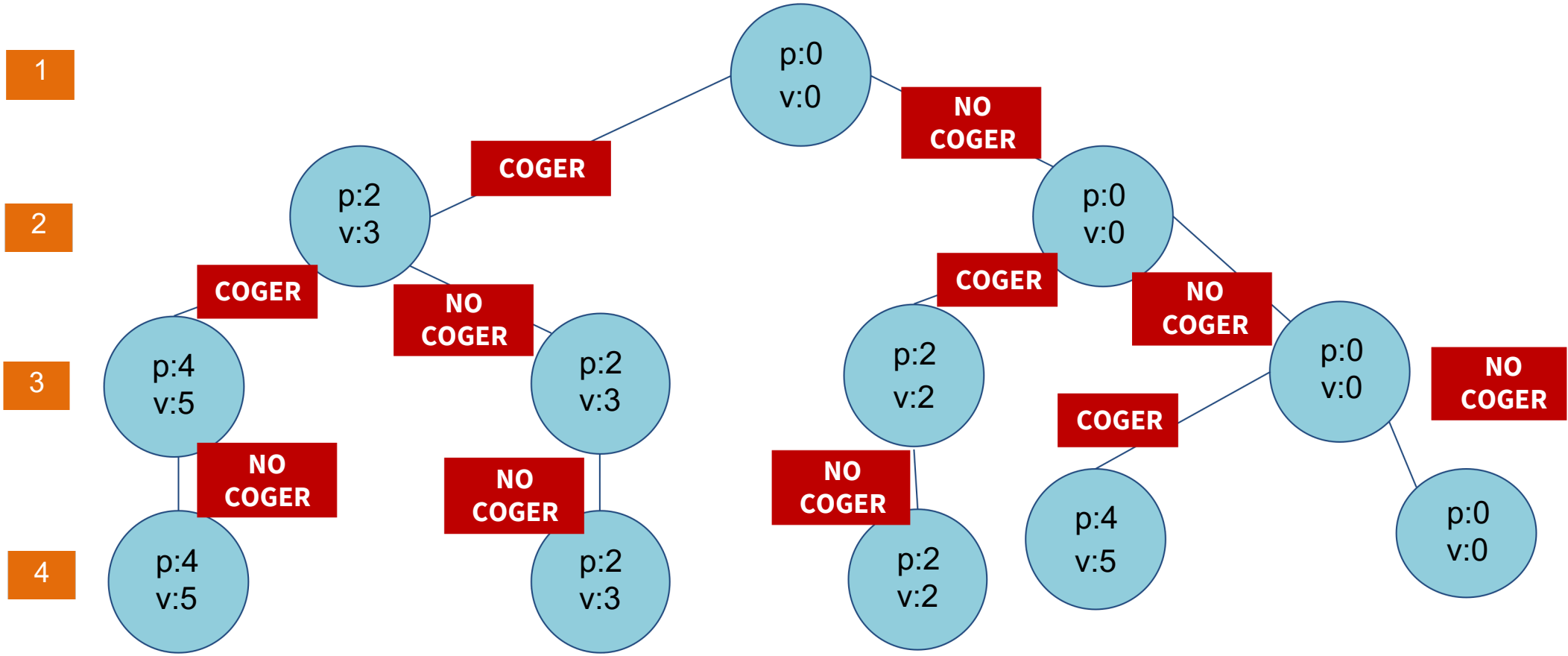
2



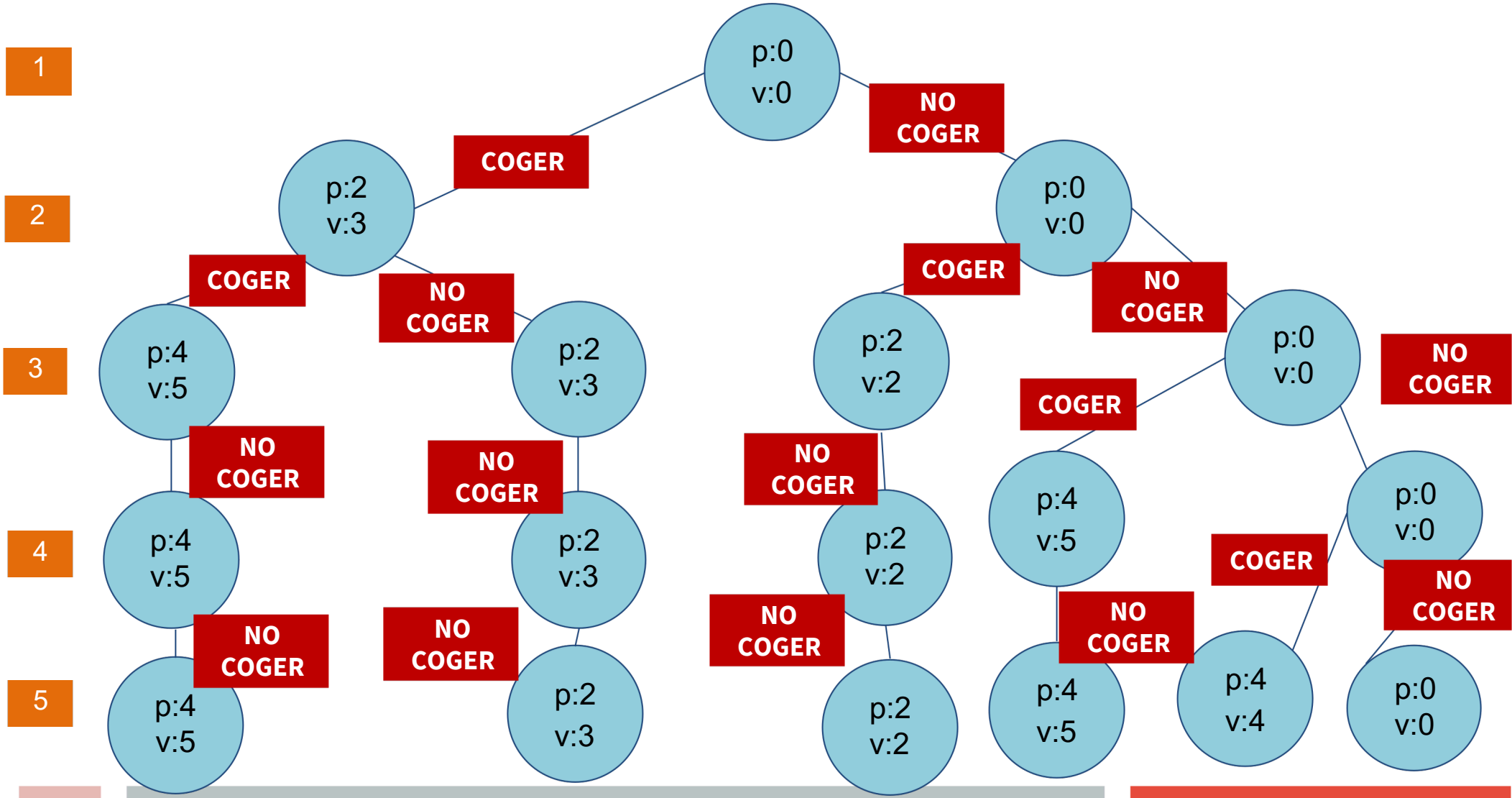
El problema de la mochila



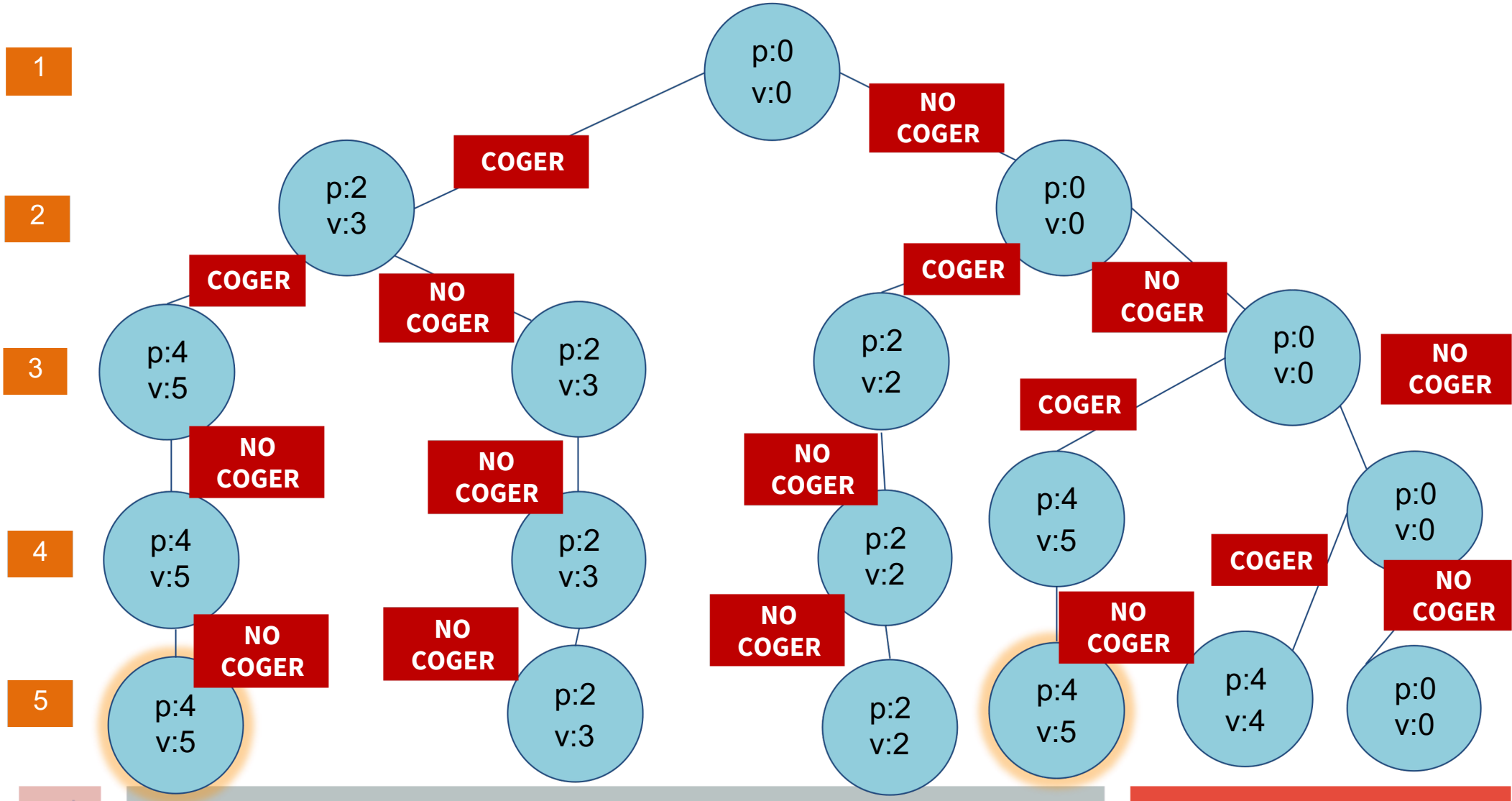
El problema de la mochila



El problema de la mochila



El problema de la mochila



El problema de la mochila

- ¿Cómo construimos la recursión?

El problema de la mochila

- ¿Cómo construimos la recursión?
  1. parámetros necesarios: índice, peso acumulado

El problema de la mochila

- ¿Cómo construimos la recursión?
  1. parámetros necesarios: índice, peso acumulado
  2. **casos base**: hemos terminado la fila (índice=n+1)

El problema de la mochila

- ¿Cómo construimos la recursión?
  1. parámetros necesarios: índice, peso acumulado
  2. **casos base:** hemos terminado la fila (índice=n+1)
  3. **llamadas recursivas**: **coger** o **no coger** el objeto y pasar al siguiente

El problema de la mochila

- ¿Cómo construimos la recursión?

```
mochila(i, acu):
    if(i==n+1):
        return 0
    if(cabe en la mochila)
        coger=valor[i]+mochila(i+1,acu+peso[i])
    no_coger=mochila(i+1,acu)
    return Max(coger, no_coger)
```

El problema de la mochila

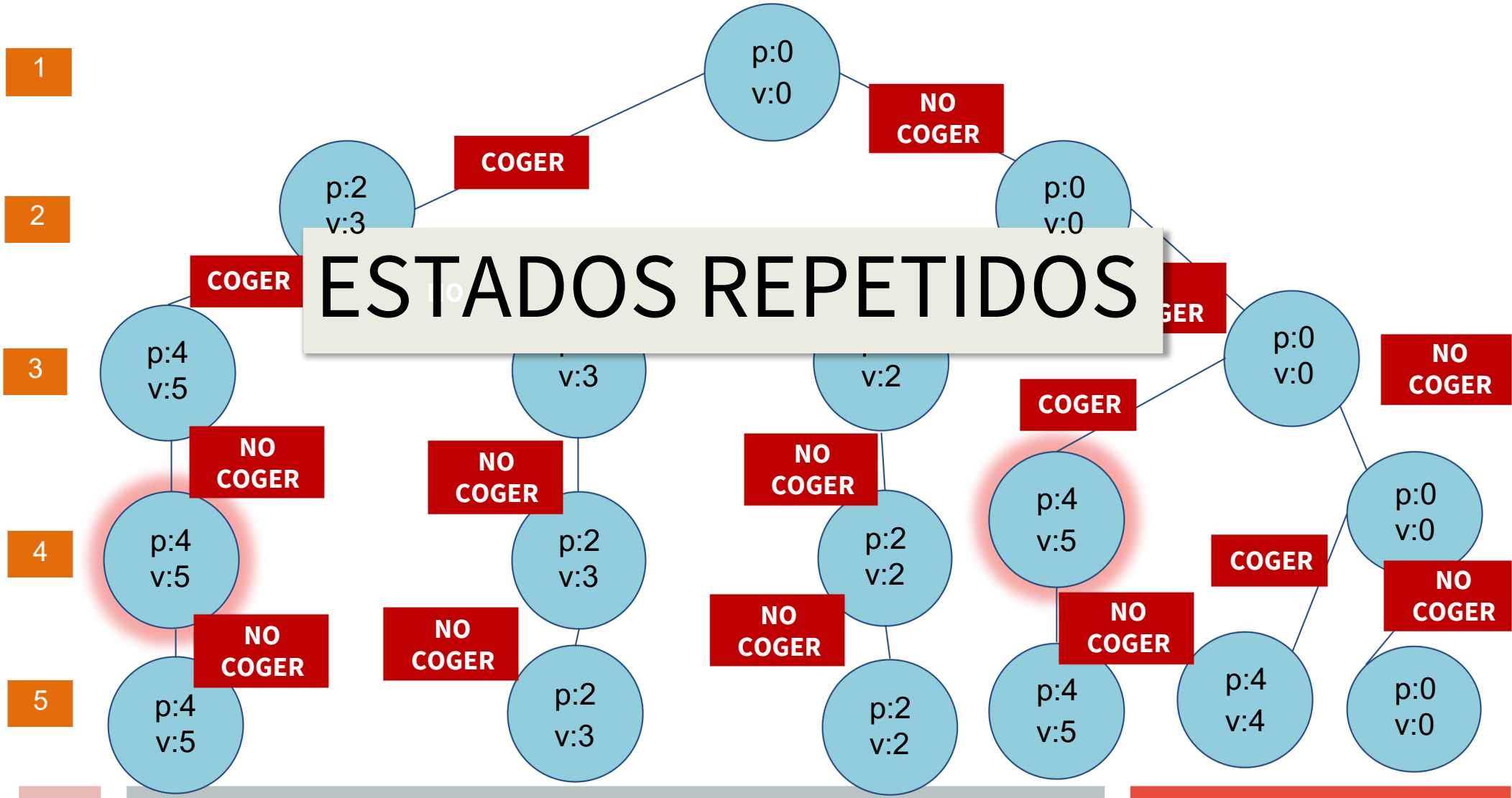
Enlace en Telegram

El problema de la mochila

- Estamos generando todos los subconjuntos posibles

complejidad exponencial

El problema de la mochila



ESTADOS REPETIDOS

El problema de la mochila

- ¿Podemos utilizar programación dinámica?

1. Crear una solución **recursiva**. ✔
2. Identificar los **estados** que se repiten. ✔ (índice, acu)
3. Añadir la **memorización**. ❓

El problema de la mochila

Enlace en Telegram

El problema de la mochila

- Solución de programación dinámica:

```
mochila(i, acu):
    if(i==n+1):
        return 0
    if(memo[i][acu]!=-1):
        return memo[i][acu]
    if(cabe en la mochila)
        coger=valor[i]+mochila(i+1,acu+peso[i])
    no_coger=mochila(i+1,acu)
    memo[i][acu]= Max(coger, no_coger)
    return memo[i][acu]
```

El problema de la mochila

- ¿Cuál es la complejidad del algoritmo?

O(nºestados * nº llamadas recursivas)

- **estado: (i,peso)** ⟶ **n*w** estados posibles
- Cada llamada realiza **2 llamadas recursivas**

↓

O(2n*w) ~ O(n*w)

Más allá...

- Otros **problemas clásicos**
  - Longest Increasing Subsequence (LIS)
  - Longest Common Subsequence (LCS)
  - Coin Problem

- Otros **enfoques**
  - Enfoque iterativo (Bottom-Up)

Dudas, preguntas, comentarios

- A vuestra disposición 💋