# TSP - PROGRAMACION II

RECURSIVIDAD

# RECURSIVIDAD

Un algoritmo se dice recursivo si calcula instancias de un problema en función de otras instancias del mismo problema hasta llegar a un caso base, que suele ser una instancia pequeña del problema, cuya respuesta generalmente está dada en el algoritmo y no es necesario calcularla.

En programación:

- La recursividad es la propiedad que tienen los procedimientos y funciones de llamarse a si mismos para resolver un problema.
- Permite describir un número infinito de operaciones de cálculo mediante un programa recursivo finito  sin implementar de forma explícita estructuras repetitivas.

**Conjuntos definidos de forma recurrente**

Un ejemplo de conjunto definido de forma recurrente es el de los números naturales, es decir, el conjunto de los números enteros no negativos:

1. $0$ pertenece a $\mathbb{N}$.
2. Si $n$ pertenece a $\mathbb{N}$, entonces $n + 1$ pertenece a $\mathbb{N}$.
3. Si $x$ verifica las anteriores condiciones, entonces $x$ está incluido en $\mathbb{N}$

# RECURSIVIDAD - Ejemplos (cont.)

**Funciones definidas de forma recurrente**

Un ejemplo conocido es la definición recurrente de la función factorial n!:

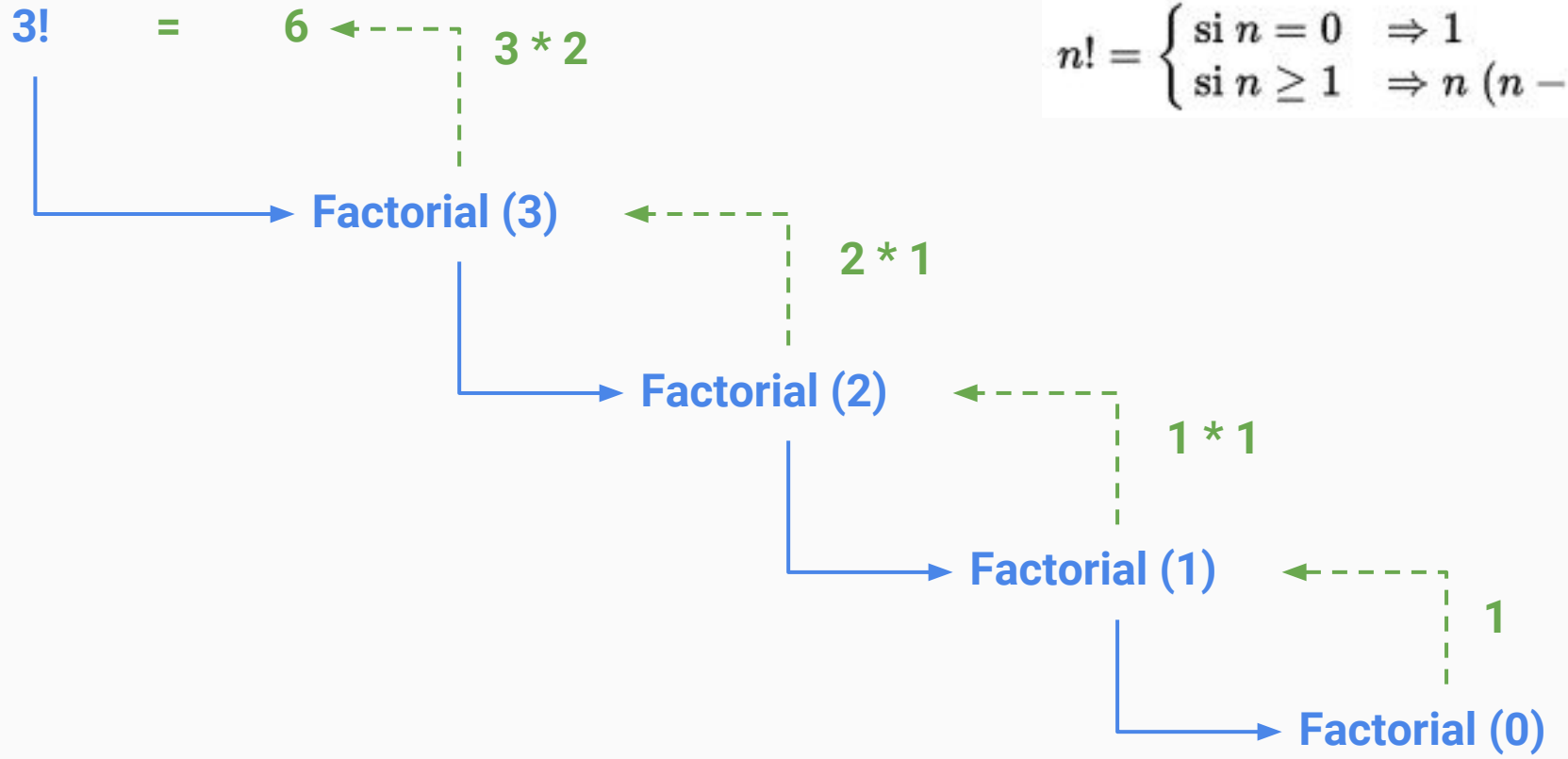
$$n! = \begin{cases} \text{si } n = 0 & \Rightarrow 1 \\ \text{si } n \geq 1 & \Rightarrow n\,(n-1)! \end{cases}$$

Veamos cómo se usa esta definición para hallar el valor del factorial de 3:

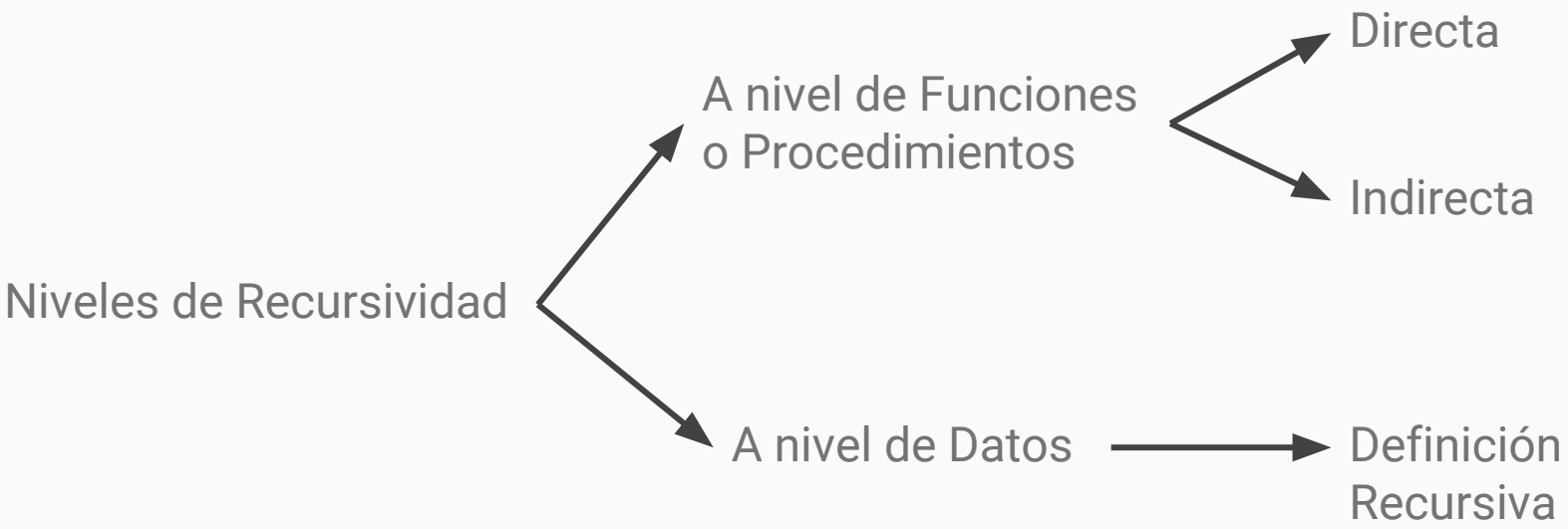
$$\begin{aligned} 3! &= 3 \cdot (3-1)! \\ &= 3 \cdot 2! \\ &= 3 \cdot 2 \cdot (2-1)! \\ &= 3 \cdot 2 \cdot 1! \\ &= 3 \cdot 2 \cdot 1 \cdot (1-1)! \\ &= 3 \cdot 2 \cdot 1 \cdot 0! \\ &= 3 \cdot 2 \cdot 1 \cdot 1 \\ &= 6 \end{aligned}$$

# RECURSIVIDAD - Ejemplos (cont.)

$$n! = \begin{cases} \text{si } n = 0 & \Rightarrow 1 \\ \text{si } n \geq 1 & \Rightarrow n\,(n-1)! \end{cases}$$



# RECURSIVIDAD - Clasificación



# RECURSIVIDAD - Directa: Serie Fibonacci

**Directa:** Una función o método se llama a sí mismo una o varias veces.

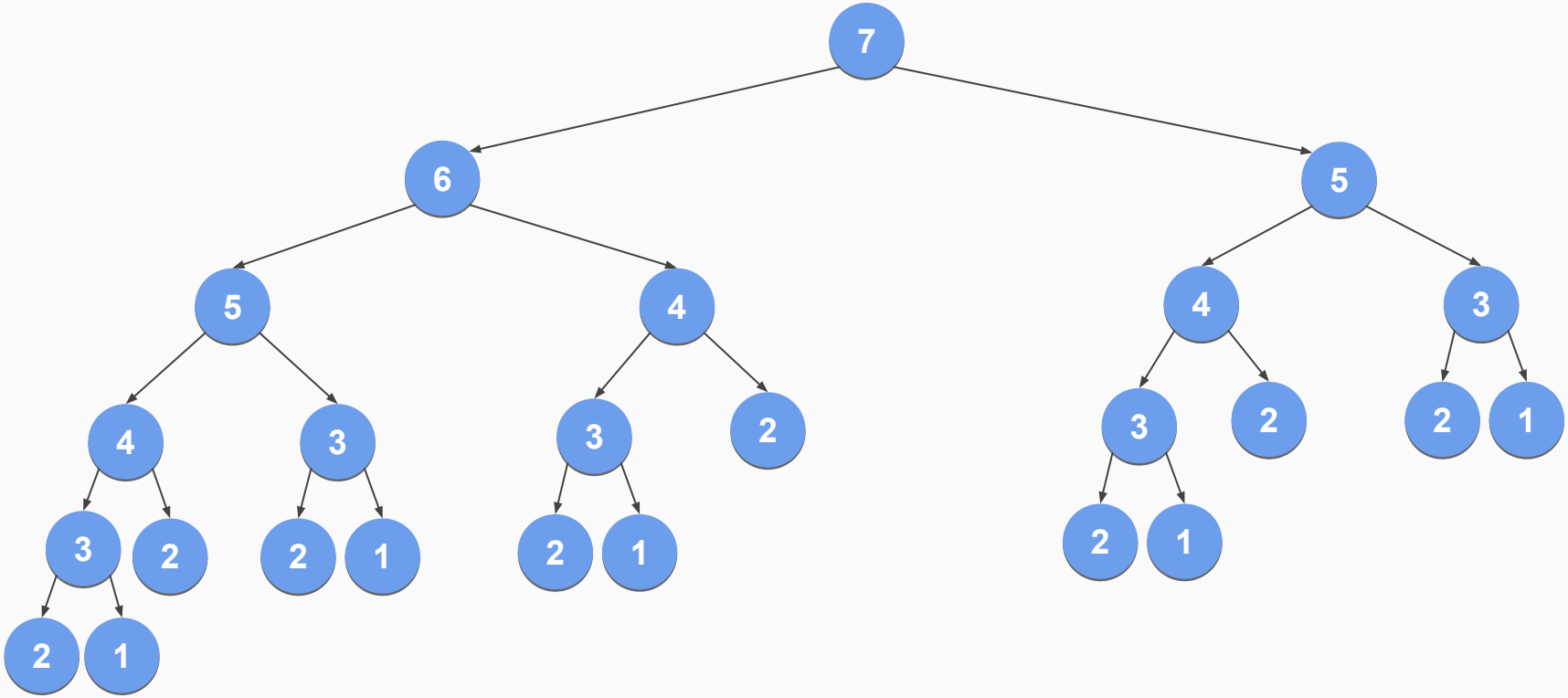
**Serie de Fibonacci**

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

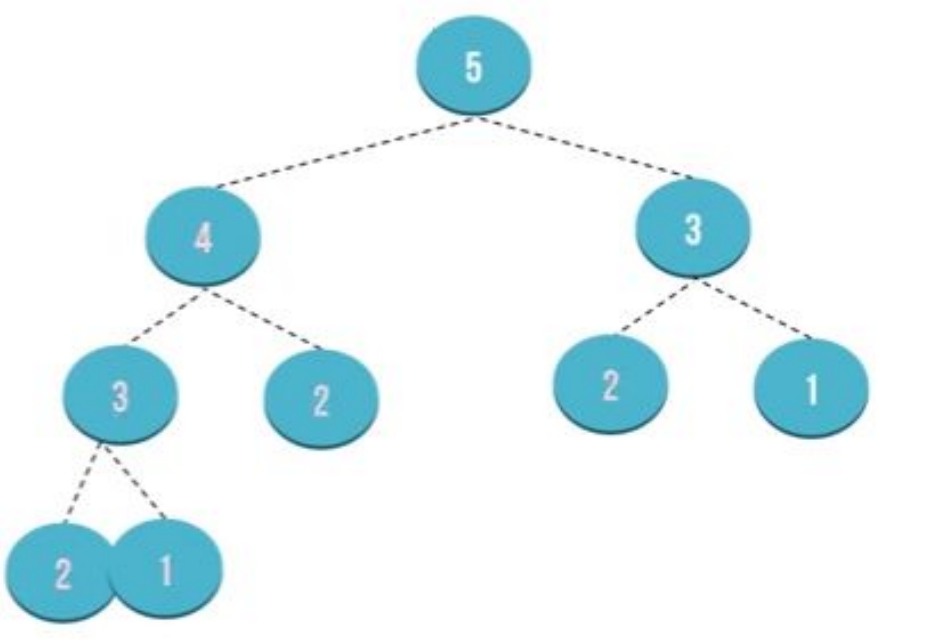
$$F_n = F_{n-1} + F_{n-2}$$

$$Fib(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ Fib(n-1) + Fib(n-2) & \text{si } n > 1 \end{cases}$$

# RECURSIVIDAD - Directa: Serie Fibonacci



# RECURSIVIDAD - Directa: Serie Fibonacci (cont.)




```
FUNC fib(↓n: NATURAL): NATURAL
INICIO
    SI n = 0 ENTONCES
        DEVOLVER 0
    SI NO, SI n = 1 ENTONCES
        DEVOLVER 1
    SI NO
        devolver fib(n-1) + fib(n-2)
    FIN SI
FIN
```

# RECURSIVIDAD - Indirecta: Números Positivos y Negativos

**Indirecta:** Una función o método “A” llama a otro “B” y este a su vez llama a “A”.

```
public boolean positivo(int n){
      if(n>0) return true;
      else return negativo(n);
 }

 public boolean negativo(int n){
      if(n<0) return false;
      else return  positivo(n);
 }
```

# RECURSIVIDAD - Características

- Una llamada o invocación a sí misma.
- Una condición de fin.
- Tiende a reemplazar a una estructura repetitiva.
- Secuencia de muy pocos pasos.
- **Es un proceso lento, ya que al apilar los datos en memoria, luego es necesario desapilarlos.**
- **Se requiere de mucha memoria para almacenar datos en una pila.**

# RECURSIVIDAD - Comparación con Estructuras Iterativas

| Recursividad | Iteración |
| --- | --- |
| La repetición de las tareas se controla desde adentro. | La repetición de tareas se controla desde afuera. |
| Se ejecuta el conjunto de acciones y antes de finalizar se evalúa si se ha llegado a la condición de salida, sino se continúa adelantando para ejecutar un nuevo conjunto de acciones. | Se ejecuta un conjunto de acciones, en forma completa, se verifica la condición de fin y si se necesita, se vuelve a ejecutar el conjunto entero de acciones. |
| Suele ser lento y ocupa mayor espacio en memoria, pero es más entendible por ser el mecanismo de razonamiento que usa el ser humano. | Normalmente es más sencillo y más eficiente, pues usa menos recursos. |
| Todo proceso recursivo suele ser iterativo. | No todo proceso iterativo puede ser recursivo. |

# RECURSIVIDAD - Comparación con Estructuras Iterativas (cont.)

| Recursividad | Iteración |
|---|---|
| Procedimiento **P (parámetros)**<br>Si <CONDICIÓN><br>entonces <ACCIÓN><br>**P(parámetros)** | **Mientras** <CONDICIÓN> hacer<br><ACCIÓN> |
| Procedimiento **P (parámetros)**<br><SENTENCIA><br>Si no<CONDICIÓN><br>entonces <ACCIÓN><br>**P(parámetros)** | **Repetir**<br><SENTENCIA><br>hasta <CONDICIÓN> |
| Procedimiento **P (i)**<br>Si i<= n<br>entonces <SENTENCIA><br>**P(i+1)** | **Para** y:=1 a n hacer<br><SENTENCIA> |

# RECURSIVIDAD - Usos

**¿Cuándo usar recursividad?**

- Para simplificar el código.
- Cuando se necesita utilizar estructuras de datos que son recursivas. Ej: Árboles.

**¿Cuándo no usar recursividad?**

- Cuando los métodos usan arreglos largos.
- Cuando el método cambia de manera  impredecible de campos.
- Cuando las iteraciones sean la mejor opción.