

Recursión

- ✓ Un método recursivo es un método que, directa o indirectamente, se hace una llamada a sí mismo.

¿Cómo puede un método resolver un problema llamándose a si mismo?

La **clave** está en que el método se llama a sí mismo pero con **instancias diferentes**, más simples, en algún sentido.

Ejemplos:

- ✓ Ficheros de un ordenador que se almacenan en directorios.
- ✓ Un diccionario enciclopédico
(Al ser finito llegaremos a un punto en el que se entenderá una definición dada).

Fundamentos. Inducción matemática.

Demostraciones por inducción matemática.

Una demostración por inducción muestra que un teorema es cierto en algunos casos simples y después demuestra como extender esa certeza indefinidamente.

En una demostración por inducción se llama **caso base** al caso sencillo que se puede demostrar directamente. La **hipótesis de inducción** supone que el teorema es cierto para un caso arbitrario. Y el **paso inductivo** de la demostración consiste en demostrar que también es cierto para el caso siguiente

Ejemplo:

Teorema:

Para todo entero $N \geq 1$, la suma de los N primeros enteros, dada por

$$\sum_{i=1}^N i = 1 + 2 + 3 + \dots + N \text{ es igual a } N(N+1)/2$$

Demostración:

El teorema es cierto para $N=1$;

Supongamos que es cierto para todo $1 \leq N \leq k$, entonces:

$$\sum_{i=1}^{k+1} i = (k+1) + \sum_{i=1}^k i \text{ de donde, sustituyendo } \sum_{i=1}^{k+1} i = (k+1) + k(k+1)/2 \text{ donde}$$

desarrollando y agrupando, tenemos $\sum_{i=1}^{k+1} i = (k+1) + k(k+1)/2$ o también

$$\sum_{i=1}^{k+1} i = (k+1)(1 + k/2) \text{ y finalmente } \sum_{i=1}^{k+1} i = (k+1)(k+2)/2$$

Que demuestra que es cierto para el caso $k+1$

Suma N enteros

Definimos la suma de los N primeros enteros de manera recursiva como:

$$\begin{aligned} \text{Suma}(1) &= 1 \\ \text{Suma}(N) &= \text{Suma}(N-1) + N \end{aligned}$$

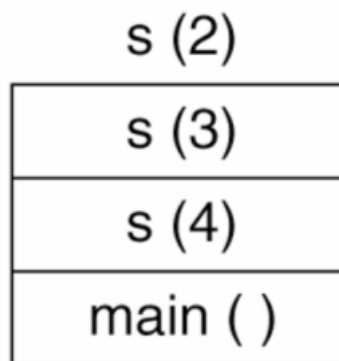
```
public class SumaRec
{
    /**
     * public static long suma(int n)
     * {
     *     if (n == 1)
     *         return 1;
     *     else
     *         return suma(n - 1) + n;
     * }
     * }
```

↗ Caso base

↗ Caso recursivo

¿Como funciona?

A bajo nivel la recursión es resuelta por **la pila de registros de activación**



```

public class SumatoriaRecLimites {

    public static long llamadas = 0;

    public static long suma(int n) {
        if (n == 1) {
            llamadas += 1;
            return 1;
        } else {
            llamadas += 1;
            return suma(n - 1) + n;
        }
    }

    public static void main(String[] args) {

        int inf = 0;
        int sup = 9841;
        int med = 0;
        long val;

        while (inf < sup) {
            med = (inf + sup + 1) / 2;
            try {
                llamadas = 0;
                long t1=System.currentTimeMillis();
                val=suma(med);
                long t2=System.currentTimeMillis();
                long t3=t2-t1;
                inf = med;
                System.out.println(inf + " " + med + " " + sup
                    + " Número llamadas: " + llamadas
                    + " Milisegundos= "+t3);
            } catch (Throwable e) {
                sup = med - 1;
                System.out.println(e);
            }
        }
        System.out.println("Maximo n que funciona: " + inf);
    }
}

```

```

9226 9226 9841 Número llamadas: 9226 Milisegundos= 0
9534 9534 9841 Número llamadas: 9534 Milisegundos= 0
9688 9688 9841 Número llamadas: 9688 Milisegundos= 0
9765 9765 9841 Número llamadas: 9765 Milisegundos= 0
9803 9803 9841 Número llamadas: 9803 Milisegundos= 0
9822 9822 9841 Número llamadas: 9822 Milisegundos= 0
9832 9832 9841 Número llamadas: 9832 Milisegundos= 0
9837 9837 9841 Número llamadas: 9837 Milisegundos= 0
9839 9839 9841 Número llamadas: 9839 Milisegundos= 0
9840 9840 9841 Número llamadas: 9840 Milisegundos= 0
java.lang.StackOverflowError
Maximo n que funciona: 9840

```

Las cuatro reglas de la recursión

1. Caso base. Se debe tener al menos un caso base que pueda resolverse sin recursión
2. Progreso. Cualquier llamada recursiva debe progresar hacia el caso base.
3. “Puede creerlo”. Asuma siempre que cualquier llamada recursiva funciona correctamente.
4. Regla del interés compuesto. Nunca duplique trabajo resolviendo la misma instancia del problema en llamadas recursivas paralelas

Factorial

Definimos que $N!$ es el producto de los N primeros enteros, expresado de manera recursiva:

$$1! = 1$$

$$N! = N \cdot (N-1)!$$

```
public class FactorialRec
{
    /**
    public static long factorial(int n)
    {
        if (n <= 1)
            return 1;
        else
            return n*factorial(n - 1);
    }
}
```

Caso base

Caso recursivo

```

public class FactorialRecLimites {

    public static long llamadas = 0;

    public static long factorial(int n) {
        if ( n == 1) {
            llamadas += 1;
            return 1;
        } else {
            llamadas += 1;
            return factorial(n - 1) * n;
        }
    }

    public static void main(String[] args) {

        int inf = 0;
        int sup = 9841;
        int med = 0;
        long val;

        while (inf < sup) {
            med = (inf + sup + 1) / 2;
            try {
                llamadas = 0;
                long t1=System.currentTimeMillis();
                val=factorial(med);
                long t2=System.currentTimeMillis();
                long t3=t2-t1;
                inf = med;
                System.out.println(inf + " " + med + " " + sup
                    + " Número llamadas: " + llamadas
                    + " Milisegundos= " + t3);
            } catch (Throwable e) {
                sup = med - 1;
                System.out.println(e);
            }
        }
        System.out.println("Maximo n que funciona: " + inf);
    }
}

```

```

9226 9226 9841 Número llamadas: 9226 Milisegundos= 0
9534 9534 9841 Número llamadas: 9534 Milisegundos= 0
9688 9688 9841 Número llamadas: 9688 Milisegundos= 0
9765 9765 9841 Número llamadas: 9765 Milisegundos= 0
9803 9803 9841 Número llamadas: 9803 Milisegundos= 0
9822 9822 9841 Número llamadas: 9822 Milisegundos= 0
9832 9832 9841 Número llamadas: 9832 Milisegundos= 0
9837 9837 9841 Número llamadas: 9837 Milisegundos= 0
9839 9839 9841 Número llamadas: 9839 Milisegundos= 0
9840 9840 9841 Número llamadas: 9840 Milisegundos= 0
java.lang.StackOverflowError
Maximo n que funciona: 9840

```

Fibonacci

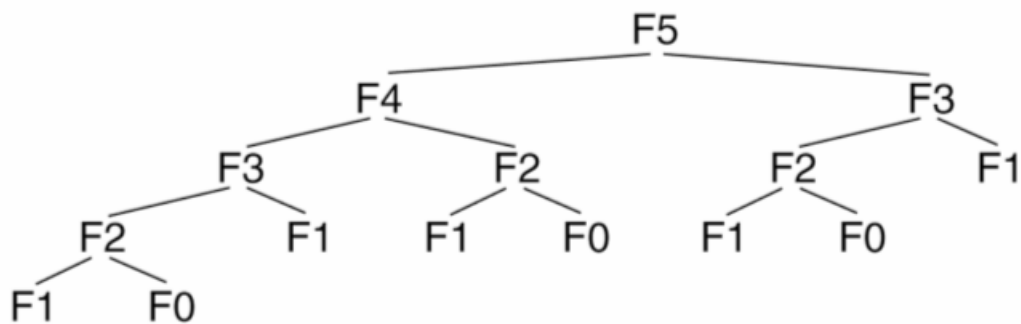
La función de Fibonacci definida en términos recursivos, sería:

$$F_0 = 1$$

$$F_1 = 1$$

$$F_i = F_{i-1} + F_{i-2}$$

```
public class FibonacciRec
{
    /**
     * public static long fib(int n)
     * {
     *     if (n <= 1)
     *         return n;
     *     else
     *         return fib(n - 1) + fib(n - 2);
     * }
}
```




```

public class FibonacciRecLimites {

    public static long llamadas = 0;

    public static long fibonacci(int n) {
        if (n == 0 || n == 1) {
            llamadas += 1;
            return 1;
        } else {
            llamadas += 1;
            return fibonacci(n - 1) + fibonacci(n - 2);
        }
    }

    public static void main(String[] args) {

        int inf = 0;
        int sup = 40;
        int med = 0;
        long val;

        while (inf < sup) {
            med = (inf + sup + 1) / 2;
            try {
                llamadas = 0;
                long t1 = System.currentTimeMillis();
                val = fibonacci(med);
                long t2 = System.currentTimeMillis();
                long t3 = t2 - t1;
                inf = med;
                System.out.println(inf + " " + med + " " + sup
                    + " numero llamadas: " + llamadas
                    + " Milisegundos= " + t3);
            } catch (Throwable e) {
                sup = med - 1;
                System.out.println(e);
            }
        }
        System.out.println("Maximo n que funciona: " + inf);
    }
}

```

```

22 22 43 numero llamadas: 57313 Milisegundos= 0
33 33 43 numero llamadas: 11405773 Milisegundos= 203
38 38 43 numero llamadas: 126491971 Milisegundos= 2010
41 41 43 numero llamadas: 535828591 Milisegundos= 8572
42 42 43 numero llamadas: 866988873 Milisegundos= 13903
43 43 43 numero llamadas: 1402817465 Milisegundos= 22460
Maximo n que funciona: 43

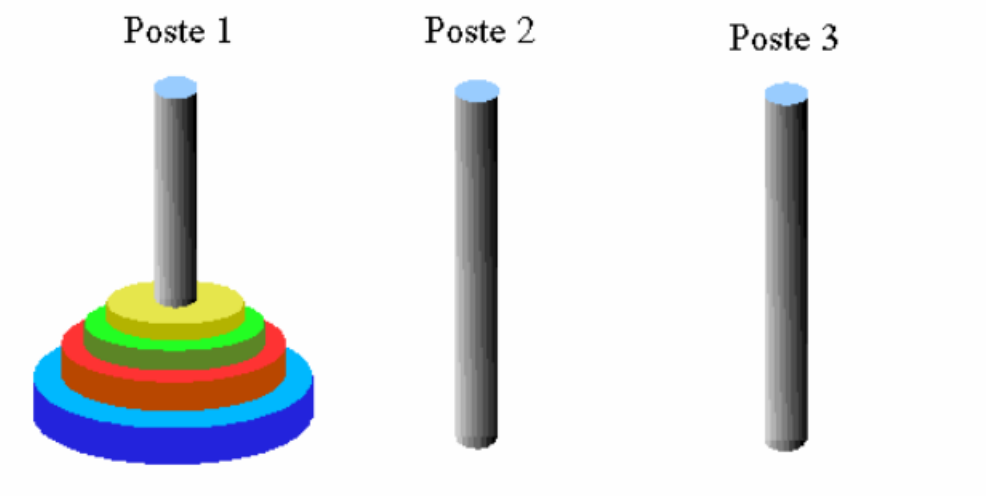
```

Torres de Hanoi

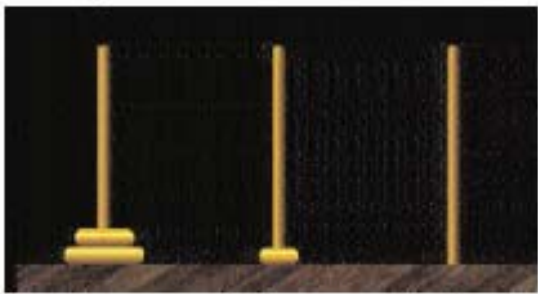
En el Gran Templo de Benarés, en la India, bajo el domo que marca el centro del mundo, descansa un plato de bronce, en el cual se encuentran tres agujas de diamantes, cada una con un "cubit" de alto (un "cubit" equivale a aproximadamente 50 centímetros) y tan gruesas como el cuerpo de una abeja. La leyenda cuenta que en una de estas agujas, durante la creación, Dios colocó 64 discos de oro puro, el disco más grande descansa en el fondo del plato de bronce y los otros se ordena de mayor a menor hasta llegar a la cima. A ésta se le conoce como la Torre de Brahma.

Día y noche sin cesar, el sacerdote responsable debía transferir los discos de una aguja de diamante a otra, bajo las leyes fijadas e inmutable de Brahma, en donde el sacerdote sólo puede mover un disco a la vez, y debe colocar estos discos en la aguja, de tal manera que un disco pequeño nunca este debajo de uno más grande.

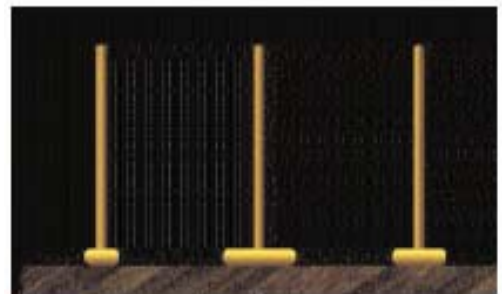
Cuando todos los 64 discos hayan sido transferidos de la aguja en la cual, durante la creación, Dios los colocó, a una de las otras agujas, entonces torre, templo y Brahmas se convertirá en polvo, y en un abrir y cerrar de ojos el mundo se terminará.



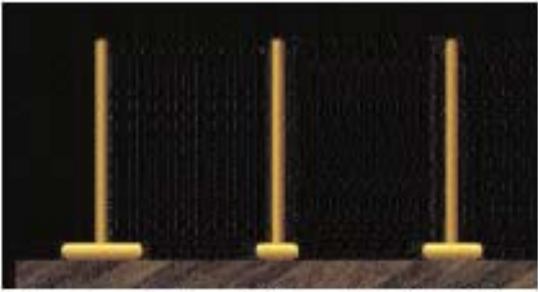
El juego (puzzle) de las Torres de Hanoi fué inventado por el matemático francés Edouard Lucas en 1883



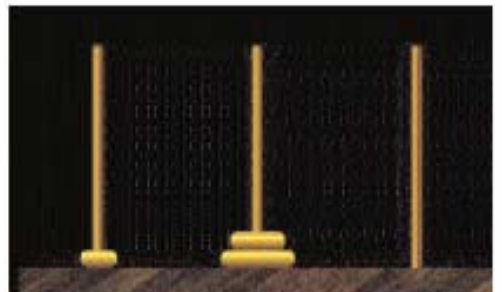
El primer movimiento es obvio.



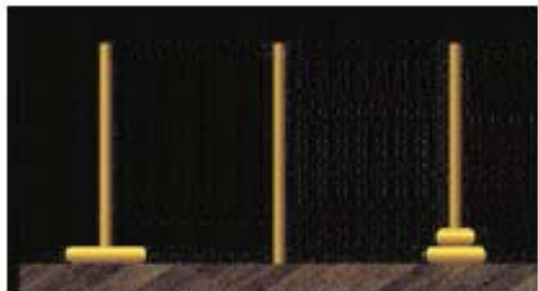
Ahora volvemos al paso uno.



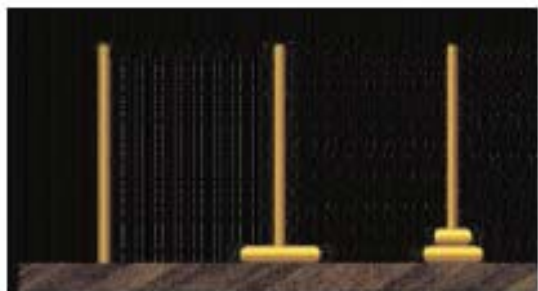
El segundo también está decidido.






Repetimos el paso dos y ¡ya está!

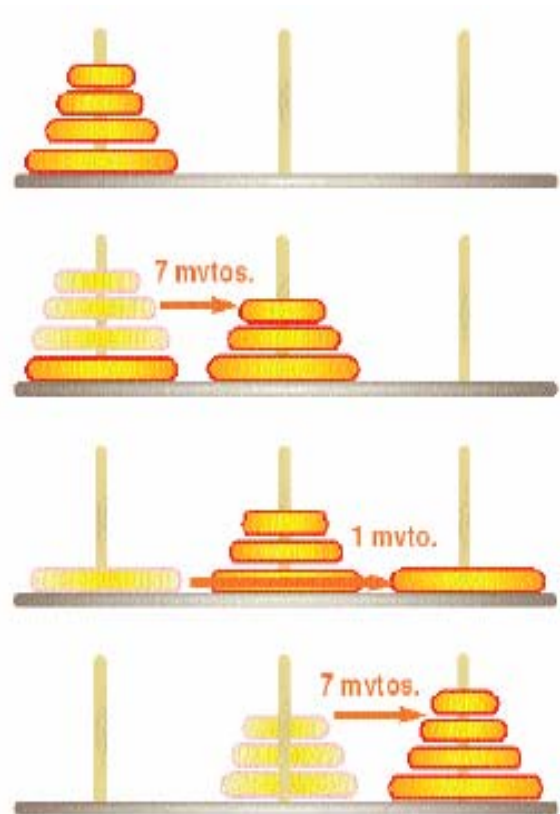


Hacemos sitio para mover el mayor.



Movemos el disco mayor. ¡Por fin!

Nº DE DISCOS	Nº MÍNIMO DE MOVIMIENTOS
	$1=2^0$ 2 TORRES DE 1 + 1 MOVIMIENTO DEL DISCO MAYOR
	$1+1+1=3=2^2-1$ 2 TORRES DE 2 + 1 MOVIMIENTO DEL DISCO MAYOR
	$3+1+3=7=2^3-1$ 2 TORRES DE 3 + 1 MOVIMIENTO DEL DISCO MAYOR
	$7+1+7=15=2^4-1$ 2 TORRES DE 4 + 1 MOVIMIENTO DEL DISCO MAYOR
	$15+1+15=31=2^5-1$



$$T_N = 2T_{N-1} + 1 \text{ for } N > 0$$

$$T_N = 2^N - 1 \text{ para } N \geq 0.$$

Si los discos son 64, como en la leyenda, se necesitan $2^{64}-1=18.446.744.073.709.551.615$ movimientos. Invirtiendo un segundo por movimiento y dedicando 24 horas al día se necesitarían casi **6.000 millones de siglos**.

n- discos desde 1 a 3 utilizando 2



```
//public static void hanoi (int n, String from, String to, String temp)
```

```
private void hanoi(int n, int a, int c, int b)
{
    if (n==1)
        System.out.println("Mover disco "+n+" desde "+a+" hasta "+c);
    else
    {
        hanoi(n-1,a,b,c);
        System.out.println("Mover disco "+n+" desde "+a+" hasta "+c);
        hanoi(n-1,b,c,a);
    }
}
```

=====

```
public static void main(String[] args) throws IOException
{
    System.out.println("Introduzca el numero de anillos:");
    int n=leer();
    TorresDeHanoi juego=new TorresDeHanoi();
    juego.solucionRec(n,1,3,2);
}
```

Introduzca el numero de anillos:

3

Mover disco 1 desde 1 hasta 3

Mover disco 2 desde 1 hasta 2

Mover disco 1 desde 3 hasta 2

Mover disco 3 desde 1 hasta 3

Mover disco 1 desde 2 hasta 1

Mover disco 2 desde 2 hasta 3

Mover disco 1 desde 1 hasta 3

```
private void hanoi(int n, int a, int c, int b)
{
    if(n==1)
        System.out.println("Mover disco "+n+" desde "+a+" hasta "+c);
    else
    {
        hanoi(n-1,a,b,c);
        System.out.println("Mover disco "+n+" desde "+a+" hasta "+c);
        hanoi(n-1,b,c,a);
    }
}
```

