.UBA fiuba

FACULTAD DE INGENIERÍA

Recursividad

Fundamentos de programación

rev2.0

¿Qué es la recursividad?

Las funciones recursivas son funciones en las que, dentro de la definición, tienen por lo menos **un llamado a sí mismas.**

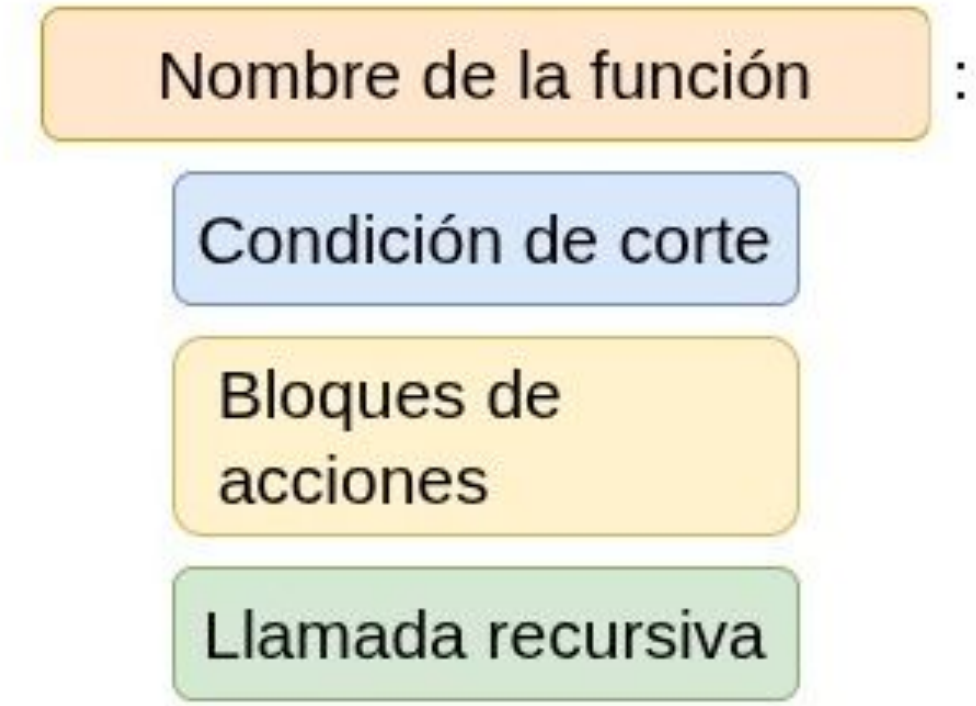
Iterativo vs recursivo

Todo problema que se pueda resolver recursivamente, también se puede resolver iterativamente.

Se prefiere el **enfoque recursivo** frente al iterativo, cuando la solución recursiva resulta más natural al problema y produce un programa más fácil de comprender y depurar.

Otra razón para elegir una solución recursiva es que una solución iterativa puede no ser clara ni evidente.

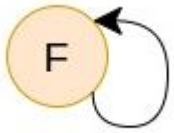
Partes de una función recursiva



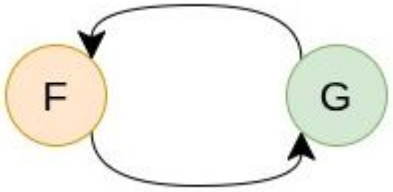
La **condición de corte** o finalización, debe evaluar si nos encontramos en presencia de un **caso base,** donde la recursividad debe terminar, o bien nos encontramos en un **caso recursivo** donde se debe realizar una llamada recursiva.

Clasificación:
Por la forma de invocación

Directa



Indirecta



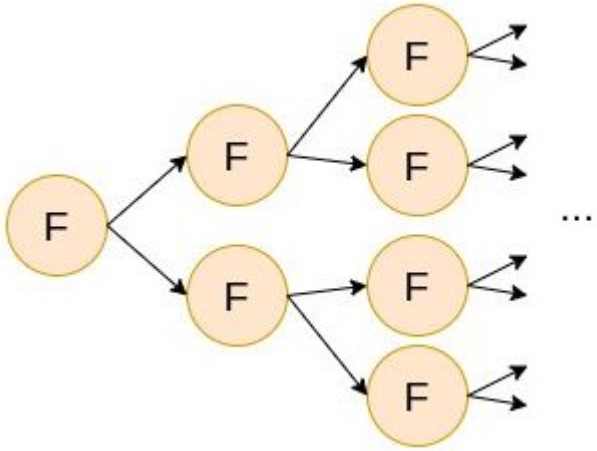
Anidada

F(n + F(n+3))

Clasificación:
Por la cantidad de llamadas

Simple

Múltiple



Desventajas de la recursividad

La recursividad tiene muchas desventajas. Se invoca repetidamente el mecanismo de recursividad y en consecuencia se necesita tiempo suplementario para realizar las mencionadas llamadas.

Esta característica puede resultar cara en tiempo de procesador y espacio de memoria.

Error de recursividad infinita

Si el algoritmo está planteado incorrectamente de forma tal que la condición del caso base nunca es verdadera, o bien el caso recursivo no reduce el problema, pueden darse llamadas recursivas indeterminadamente.
En la práctica el código se ejecutará hasta que la computadora agote la memoria disponible y se produzca una terminación anormal del programa por desbordamiento de pila (stack overflow).

El flujo de control de un algoritmo recursivo requiere de tres elementos para funcionar correctamente:

- Una evaluación para detener o continuar la recursión.
- Una llamada recursiva para continuar la recursión.
- Un caso final para terminar la recursión (caso base).

Ejemplo: Potencia

Para todo exponente entero positivo.

$$base^{exponente} = \underbrace{base * base * \ldots * base}_{exponente\ veces}$$

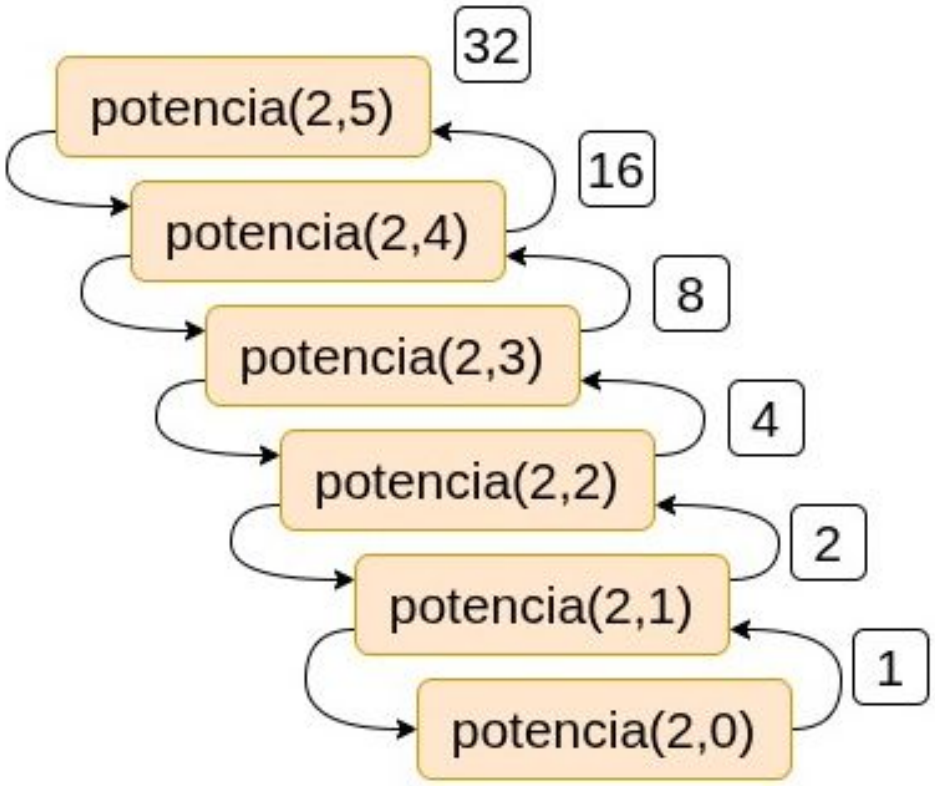
Definición

$$potencia(x, n) = \begin{cases} 1 & \text{si } n = 0 \text{ (caso base)} \\ x * potencia(x, n-1) & \text{si } n > 0 \text{ (caso recursivo)} \end{cases}$$

Ejemplo: Potencia (Pseudocódigo)

```
funcion potencia(base, exponente)
   si exponente igual a 0
      retornar 1
   retornar base * potencia(base, exponente-1)
```

Ejemplo: Potencia



Caso Base

Ejemplo: Fibonacci

valor	0	1	1	2	3	5	8	13	21	34	55	89	…
orden	0	1	2	3	4	5	6	7	8	9	10	11	…

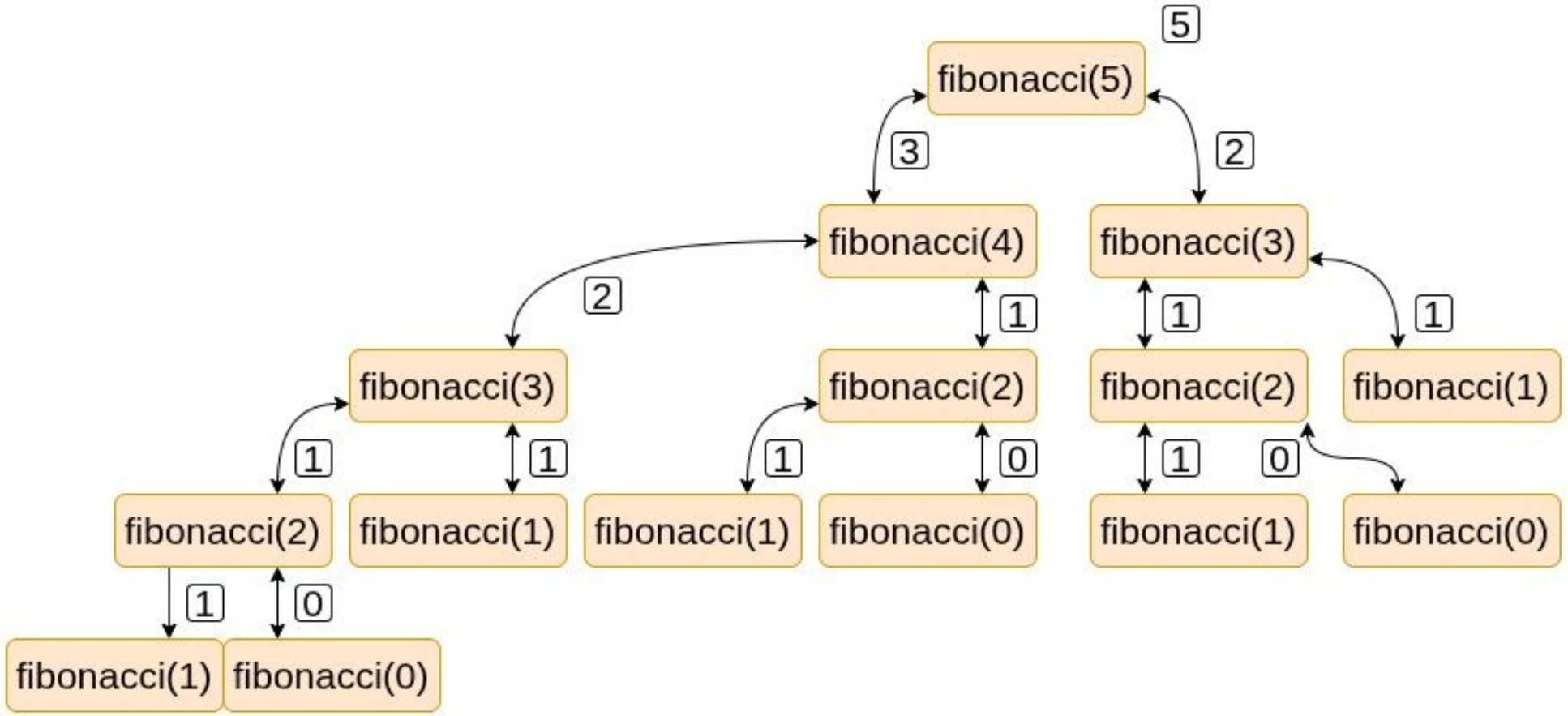
Definición

$$\text{fibonacci(orden)} = \begin{cases} 0 & \text{si orden} = 0 \ \text{(caso base)} \\ 1 & \text{si orden} = 1 \ \text{(caso base)} \\ \text{fibonacci(orden-1)} + \text{fibonacci(orden-2)} & \text{si orden} > 1 \ \text{(caso recursivo)} \end{cases}$$

Ejemplo: Fibonacci (Pseudocódigo)

```
funcion fibonacci(orden)
   si orden igual a 0
      return 0
   si orden igual a 1
      retornar 1
   retornar fibonacci(orden-1) + fibonacci(orden-2)
```

Ejemplo: Fibonacci



Práctica: Factorial

Plantear la solución del cálculo del factorial considerando su definición recursiva.

Definición

$$\text{factorial}(n) = \begin{cases} 1 & \text{si } n = 0 \text{ (caso base)} \\ n * \text{factorial}(n-1) & \text{si } n > 0 \text{ (caso recursivo)} \end{cases}$$

Bibliografía

- **Capítulo 14** - Recursividad - **Fundamentos de programación - Algoritmos, estructuras de datos y objetos** - [Joyanes Aguilar][Mc Graw Hill][5ta ed]