Recursividad

Contenido

- Aproximación.
- Definición formal.
  - Propiedades.
  - Caso de Análisis: factorial “!”.
- Buena-mala práctica de recursividad.
- Algoritmos de rastreo Inverso.
  - Caso de Análisis: Ocho Reinas.
- Ejemplos:
  - Serie de Fibonacci. Torres de Hanoi. Salto del caballo.
- Empleo de árbol Binario.

Aproximación

- ¿Nunca tuvieron un sueño y dentro del mismo se encontraban soñando?
- ¿Vieron una película que en ella estaban preparando otra película?

Aproximación

- Una *función* que se *llama* a sí mismo se dice que es *recursiva*. La *recursividad* en las *funciones* puede darse de dos maneras diferentes:
  - ***Directa***.
    - *La función se llama directamente a sí misma.*
  - ***Indirecta***.
    - *La función llama a otra función y esta a su vez llama a la primera.*

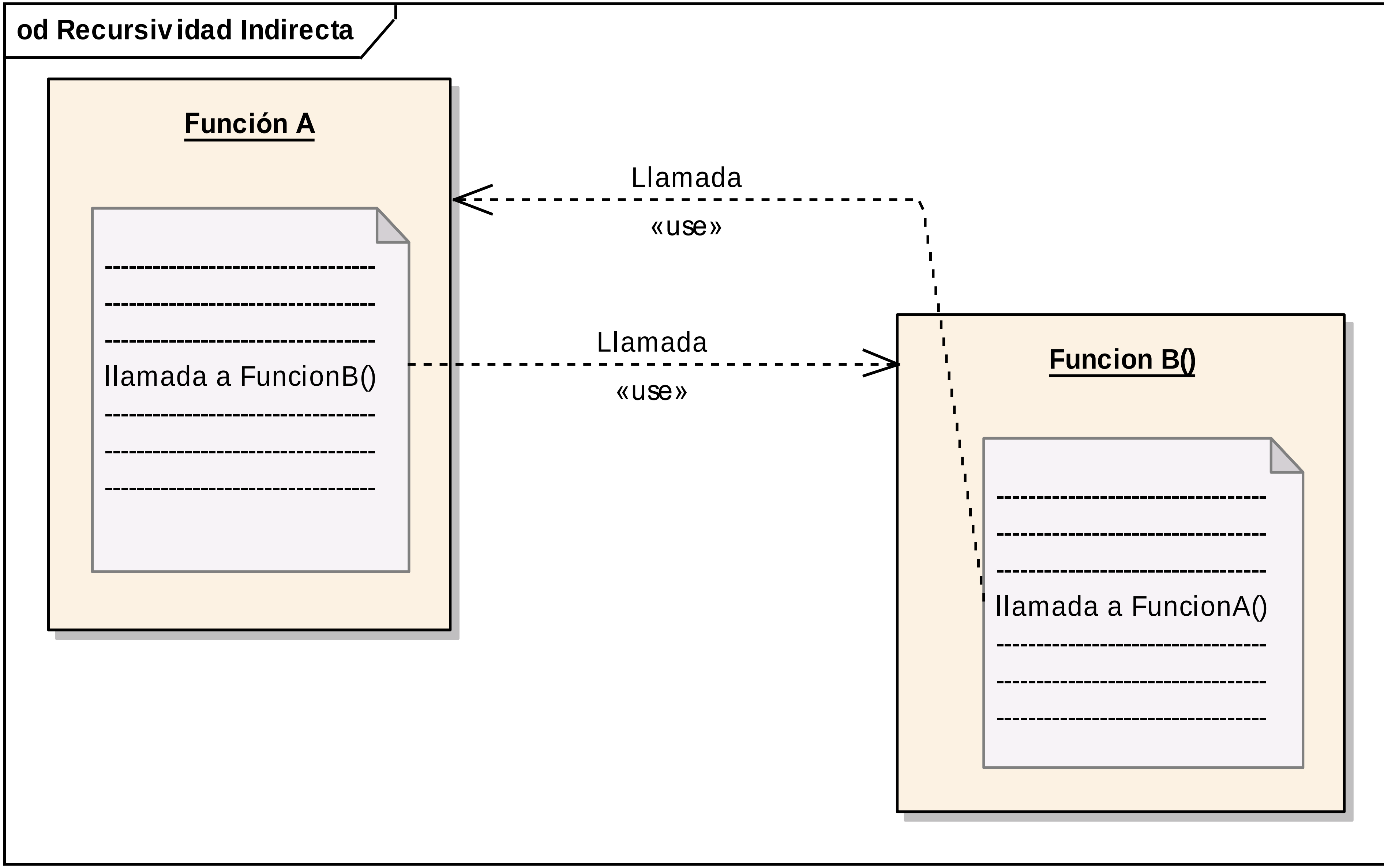


Figura 1: Llamada Indirecta de una función recursiva.

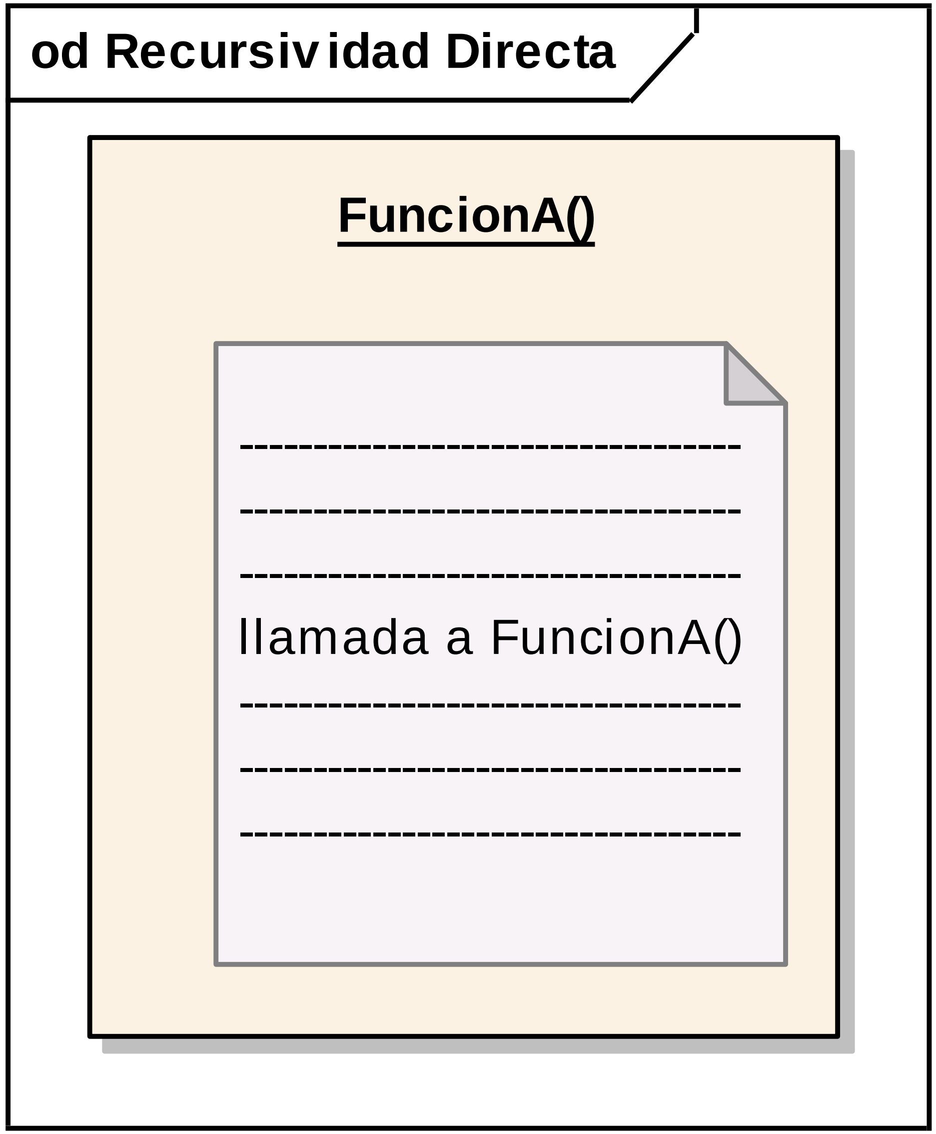


Figura 2: Llamada Directa de una función recursiva.

Definición formal

- Se dice que un procedimiento es recursivo, si en parte está formado por sí mismo o se define en función de sí mismo.
- Una composición P compuesta de un conjunto de sentencias S (que pueden contener a P) y P.
- Expresión simbólica:

$$\boxed{P = P[S,P]}$$

Propiedades

- Se reserva espacio para almacenar el nuevo conjunto completo de las nuevas variables.
  - Hay variables con valores "presentes" y "pendientes".
- La profundidad de la recursión debe ser **finita**, y además **pequeña**.
- Están sujetas a una condición de **salida B.** En símbolos:

función P = if B then P[S,P] else fin;

Caso Análisis: factorial

- ***Factorial de un número:*** *La notación* ***n!*** *se lee* ***factorial*** *de* ***n*** *e indica el producto de los enteros positivos desde* ***1*** *hasta* ***n****. Por ejemplo:*

  $$3! = 1 \times 2 \times 3$$

  $$4! = 1 \times 2 \times 3 \times 4$$

  $$5! = 1 \times 2 \times 3 \times 4 \times 5$$

  $$n! = 1 \times 2 \times 3 \times \ldots \times (n-2) \times (n-1) \times (n)$$

- También ***definimos 1! = 1*** *y* ***0! = 1***..

Caso Análisis: factorial

- Si ***invertimos*** la definición de la fórmula tenemos:

$$n! = n\ (n-1)\ (n-2)\ \ldots\ 3 \times 2 \times 1$$

- Donde está la recursividad?

  En general, podemos definir ***n!*** como:

$$n! = n\ (n-1)!$$
$$(n-1)! = (n-1)\ (n-2)!$$
$$(n-2)! = (n-2)\ (n-3)!$$

.

.

Caso Análisis: factorial

```
#include <iostream>
using namespace std;

int fact (int n) {
if (n==0) return (1);
else return (n*fact(n-1));
}

int main ( ) {
int r, m;
cin >> m;
r = fact(m);
cout << r;
return (0);
}
```

```
3
6
```

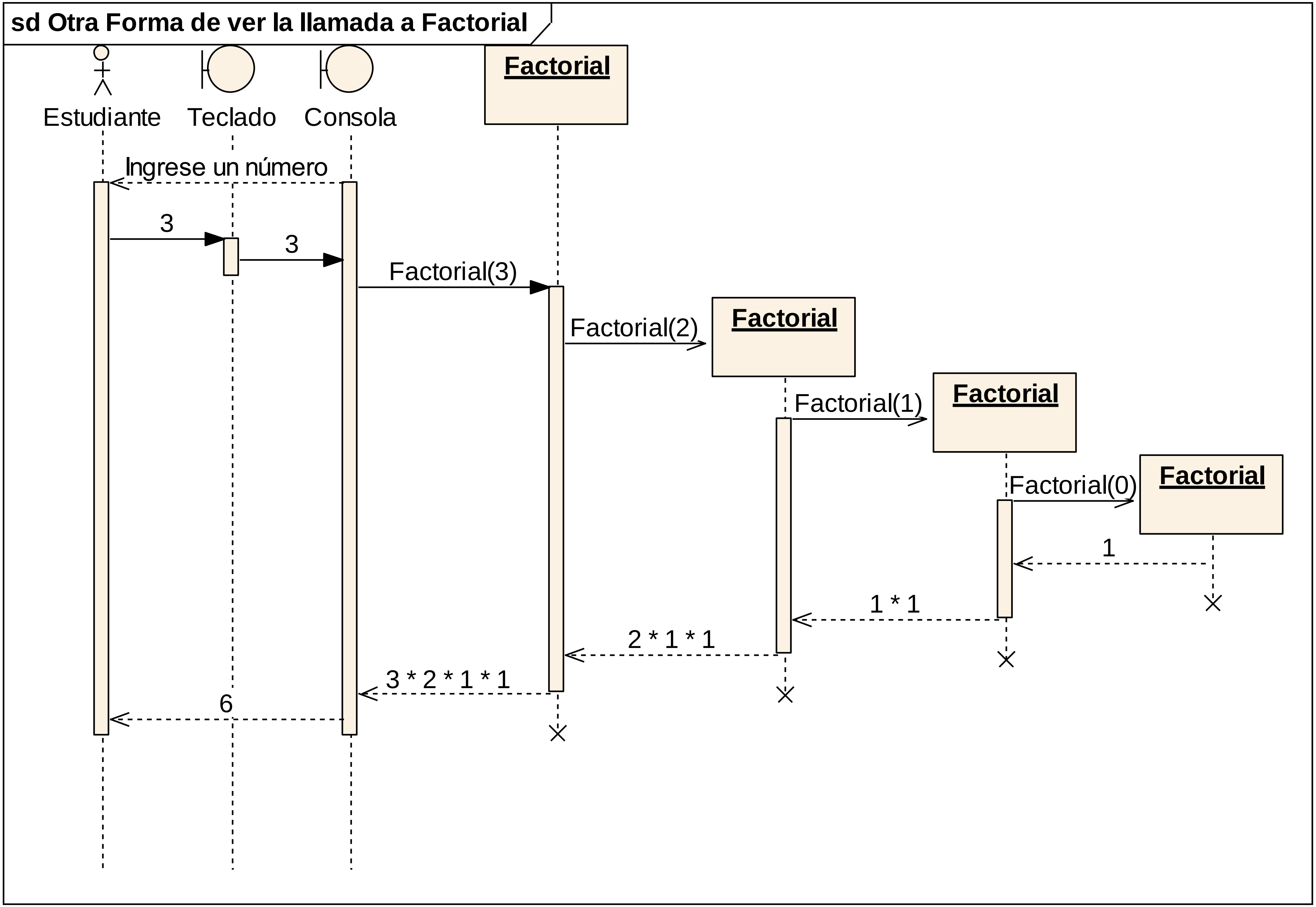


Figura 3: Otro diagrama de secuencia para llamadas recursivas de factorial. Observe las variables pendientes y las presentes.

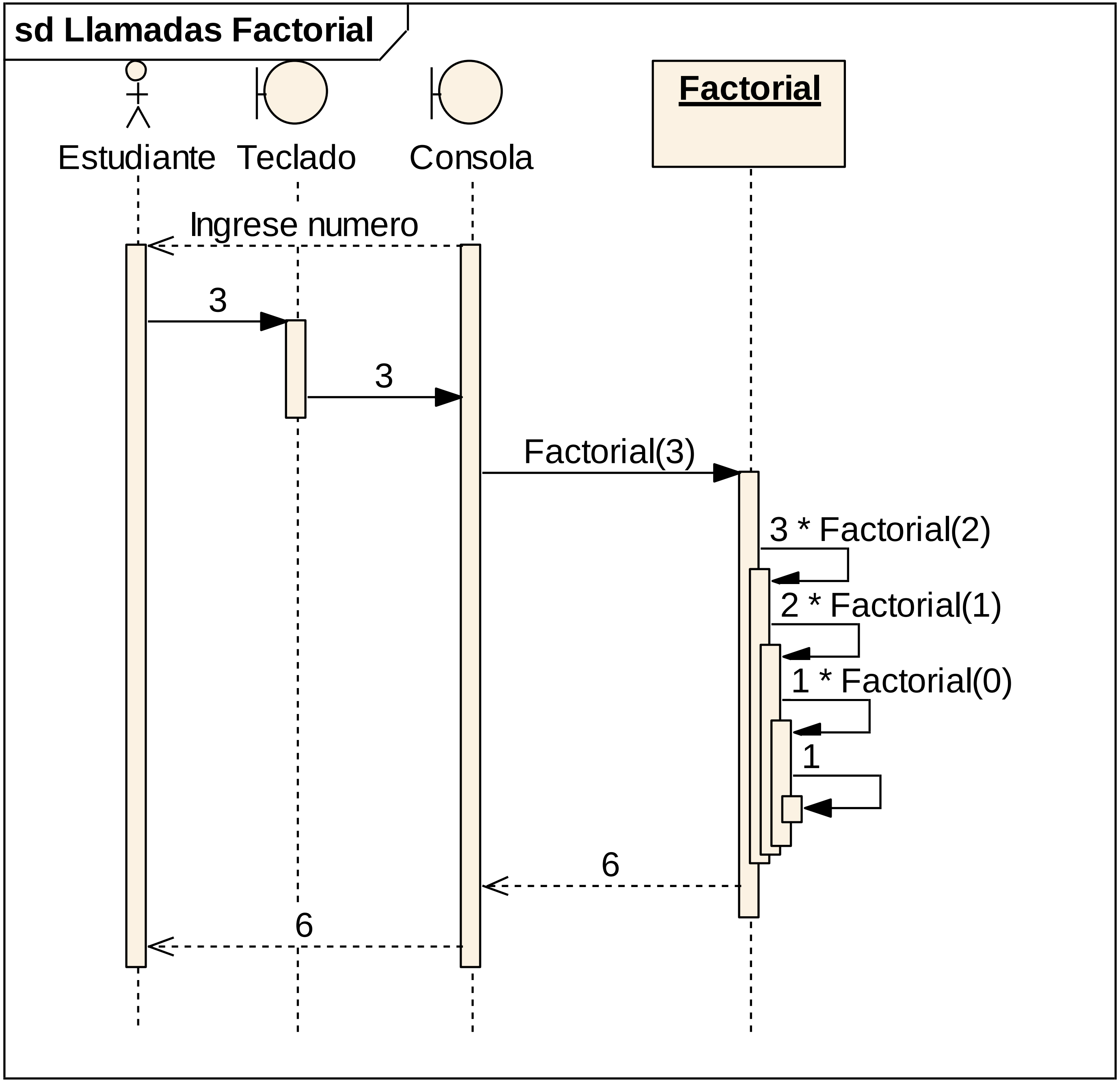


Figura 4: Diagrama de secuencia para llamadas recursivas de factorial.

Caso de Análisis: Factorial Iterativo

```
int FactorialIterativo(unsigned int n)
{unsigned int producto = 1;
  while(n != 0)
     {producto *= n; //n! = n*(n-1)*(n-2)*..*1 si n>0
       n--;}
  return producto;
}
```

¿Cuanta memoria utilizamos ahora?

Buena-mala práctica de recursividad

- Una solución recursiva puede no ser la mejor manera de resolver el problema.
- Siempre que pueda ser implementado iterativo, debe hacerse así.
- Esquemas donde hay sólo una llamada P al final o al inicio de la composición, que se puede representar de la siguiente manera:

  Función P = if (B) S else P

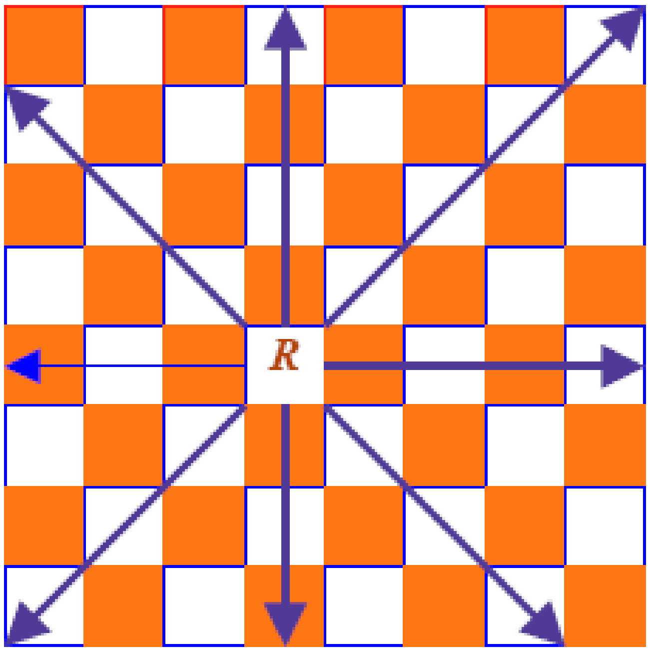
  Función P = S; if (B) P |

  Función P =[inicializar; while(B) S]

- Estos esquemas están asociados con funciones elementales de recurrencia.

Caso de Análisis: Ocho Reinas

- Objetivo:
  - Colocar ocho reinas dentro de un tablero de ajedrez, sin que se ataquen la una con la otra.



•Recordar que una reina pude comer moviendose:

> Horizontalmente.
> Verticalmente.
> Diagonalmente.

Figura 5: Movimientos posibles de una reina.

Caso de Análisis: Ocho Reinas

	0	1	2	3	4	5	6	7
0							6	
1						6		
2					6			9
3				R			9	
4			6			9		
5		6			9			
6	6			9				
7			9					

	0	1	2	3	4	5	6	7
0	0							
1		0						
2	-2		0					
3		-2		0				
4			-2		0			
5				-2		0		
6					-2		0	
7						-2		0

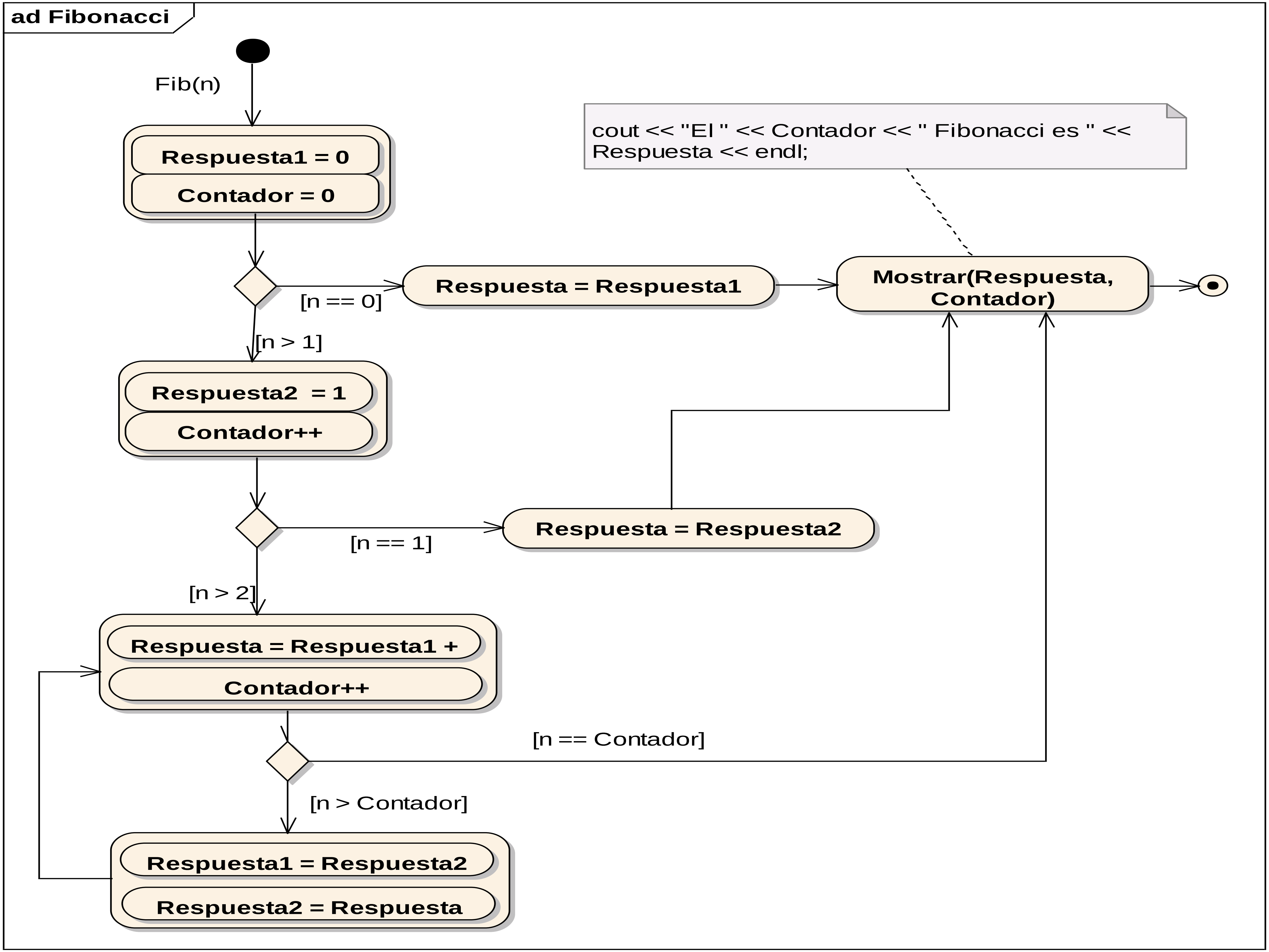
Figura : Diagonales del tablero.

Derecha: Suma de los índices fila y columna para las diagonales ascendentes.

Izquierda: Resta de los índices fila y columna para las diagonales desscendentes.

Ejemplo: Serie de Fibonacci

- *Otro caso clásico de problemas definidos recursivamente es el cálculo de la serie de* ***Fibonacci*: *0, 1, 1, 2, 3, 5, 8, 13, 21, …***
- *Recuérdese que el Fibonacci de un número se obtiene de la suma de los dos números Fibonacci anteriores.*
- *Por definición:*
  - ***Fibonacci(0) = 0***
  - ***Fibonacci(1) = 1***


ad Fibonacci
Fib(n)
Respuesta1 = 0
Contador = 0
cout << "El " << Contador << " Fibonacci es " << Respuesta << endl;
Respuesta = Respuesta1
[n == 0]
Mostrar(Respuesta, Contador)
[n > 1]
Respuesta2 = 1
Contador++
Respuesta = Respuesta2
[n == 1]
[n > 2]
Respuesta = Respuesta1 +
Contador++
[n == Contador]
[n > Contador]
Respuesta1 = Respuesta2
Respuesta2 = Respuesta

Ejemplo: Torres de Hanoi

- Objetivos:
  - Dadas tres torres "A", "B" y "C", con una cantidad de N discos apilados en el torre "A".
  - Los discos son de diferentes tamaños e inicialmente se encuentran apilados desde el más grande hasta el más pequeño.
  - Se pide lograr que los N discos de la torre "A" pasen a la "C" según las reglas:
    - Sólo se puede mover el disco superior de cualquier torre.
    - Un disco más grande no puede estar encima de uno más pequeño.

Ejemplo: Torres de Hanoi

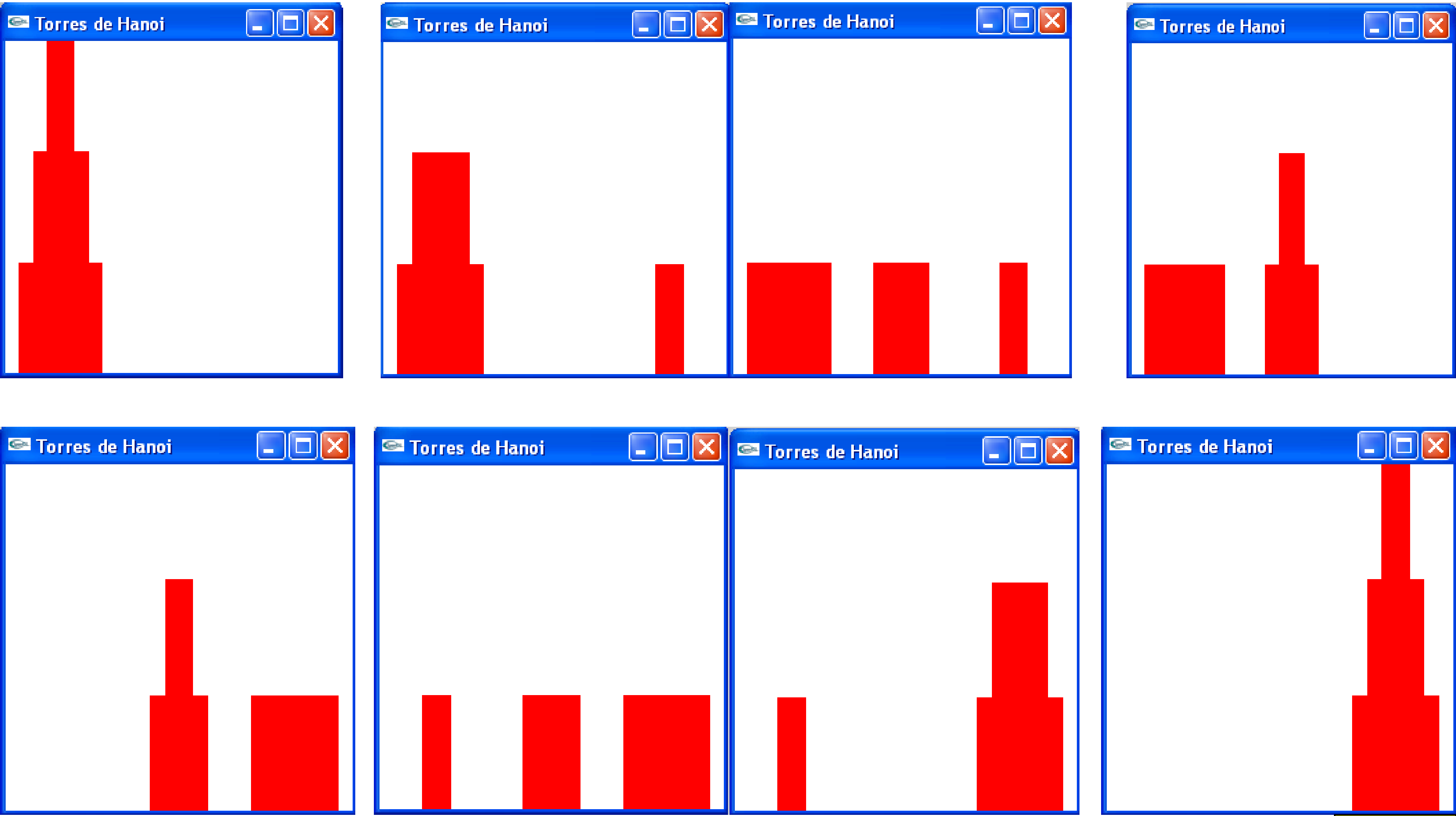


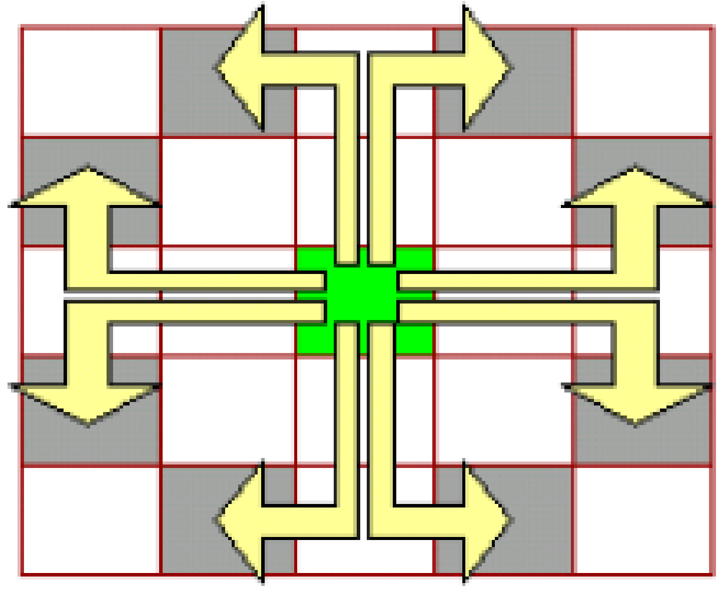
Figura : Pasos necesarios para resolver las Torres de Hanoi con 3 discos.

Ejemplo: Torres de Hanoi

```
void MoverDiscos(unsigned short DiscoN, stack<Disco>& TorreOrigen,
    stack<Disco>& TorreAuxiliar, stack<Disco>& TorreDestino)
{if(DiscoN == 1)
   {if(TorreOrigen.size())
        {TorreDestino.push(TorreOrigen.top());
        TorreOrigen.pop();}
   }
 else
   { MoverDiscos(DiscoN-1, TorreOrigen, TorreDestino, TorreAuxiliar);
     if(TorreOrigen.size())
        {TorreDestino.push(TorreOrigen.top());
        TorreOrigen.pop();}
MoverDiscos(DiscoN-1, TorreAuxiliar, TorreOrigen, TorreDestino);
    }
}
```

Ejemplo: Salto del caballo

- Objetivo:
  - Visitar todas las casillas de un tablero de ajedrez por medio de los movimientos de un caballo.
  - Se debe pasar solamente una vez por cada casilla.



Empleo de árbol Binario