RECURSION

La recursión es una de las herramientas más poderosas para resolver problemas en Computación.

Una definición recursiva dice cómo obtener conceptos nuevos empleando el mismo concepto que intenta describir. (Esto se llama definición **INDUCTIVA**)

Los algoritmos recursivos ofrecen soluciones estructuradas, modulares y elegantemente simples.

RECURSION: un ejemplo

Definición:

factorial (n) = n!

n! = n * n-1 * n-2 * ... * 1 si n > 0

el valor de 0! se define como

factorial (0) = 0! = 1 si n > 0

O sea que

4! = 4 * 3 * 2 * 1

Algoritmo ITERATIVO

```
prod = 1
repetir desde n hasta 1 (-1)
    prod = prod * n
```

Algoritmo RECURSIVO

```
Si  n == 0
    prod = 1
Sino
    prod = n * Factorial( n – 1 )
```

RECURSION

Una definición recursiva dice cómo obtener conceptos nuevos empleando el mismo concepto que intenta definir.

El poder de la recursividad es que los procedimientos o conceptos complejos pueden expresarse de una forma simple.

Un razonamiento recursivo tiene dos partes: la base y la regla recursiva de construcción. La base no es recursiva y es el punto tanto de partida como de terminación de la definición.

RECURSION

- Definición: Cuando una llamada recursiva es la última posición ejecutada del procedimiento se llama recursividad de cola, recursividad de extremo final o recursión de extremo de cola.
- Definición: Cuando un procedimiento incluye una llamada a si mismo se conoce como recursión directa.
- Definición: Cuando un procedimiento llama a otro procedimiento y este causa que el procedimiento original sea invocado, se conoce como recursión indirecta.
- Al principio algunas personas se sienten un poco incómodas con la recursividad, tal vez porque da la impresión de ser un ciclo infinito, pero en realidad es menos peligrosa una recursión infinita que un ciclo infinito, ya que una recursividad infinita pronto se queda sin espacio y termina el programa, mientras que la iteración infinita puede continuar mientras no se termine en forma manual.
- Cuando un procedimiento recursivo se llama recursivamente a si mismo varias veces, para cada llamada se crean copias independientes de las variables declaradas en el procedimiento.

RECURSION

Un conjunto de objetos está definido recursivamente siempre que:

algunos elementos del conjunto se especifican explícitamente y aseguren el final o corte (base)

el resto de los elementos del conjunto se definen en términos de los elementos ya definidos (regla recursiva)

Base: La secuenciación, iteración condicional y selección son estructuras válidas de control que pueden ser consideradas como enunciados.

Regla recursiva: Las estructuras de control que se pueden formar combinando de manera válida la secuenciación iteración condicional y selección también son válidos.

RECURSION

1. El procedimiento se llama a si mismo
2. El problema se resuelve, resolviendo el mismo problema pero de tamaño menor
3. La manera en la cual el tamaño del problema disminuye asegura que el caso base eventualmente se alcanzará

Definir por ejemplo:

$$n! = (n + 1)! / (n + 1)$$

Sumar de manera continua 1 a la variable n no conduce a un final definido.

RECURSION: otro ejemplo

Multiplicación de números naturales:

$a * b = a$ si $b = 1$

$a * b = a * (b - 1) + a$ si $b > 1$

O sea que:

$6 * 3 = 6 * 2 + 6 = 6 * 1 + 6 + 6 = 6 + 6 + 6 = 18$

Algoritmo RECURSIVO

```
FUNCION Multiplic( a, b )
Si  b = 1
    prod = a
Sino
    prod = Multiplic( a, b – 1 ) + a
```

RECURSION: otro ejemplo

La secuencia de Fibonacci

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Cada elemento es esta secuencia es la suma de los dos precedentes.

Ej. 0 + 1 = 1, 1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, ...

Entonces podemos definir la secuencia como:

fib(n) = n si n = 0 o si n = 1

fib(n) = fib(n – 2) + fib(n – 1) si n >= 2

Algoritmo ITERATIVO

```
FUNCION Fib( n )
Si  n <= 1
    Salida = n;
loFib = 0;
hiFib = 1;
Repetir i = 2; i <= n
    x = loFib;
    loFib = hiFib;
    hiFib = x + loFib;
Salida = hiFib
```

Algoritmo RECURSIVO

```
FUNCION Fib( n )
Si  n <= 1
    Fib = n
Sino
    Fib = Fib( n – 2 ) + Fib( n – 1 )
```

Importante: el cálculo recursivo es mucho más caro que el iterativo.

RECURSION: búsqueda

Intentemos un ejemplo más interesante.

El objetivo es buscar un nombre dentro de una guía telefónica.

¿Cómo sería nuestro algoritmo de búsqueda?

El método de búsqueda menos refinado es el de búsqueda lineal o secuencial. (si la lista no está ordenada, quizás este puede ser el único método para encontrar algo)

Si la guía se encuentra ordenada cual puede ser un método más adecuado para realizar la búsqueda.

```
FUNCION Búsqueda( bajo, alto, x )
Si bajo >= alto
        salida = -1
medio = entero( (bajo + alto) / 2 )
Si x = a[ medio ]
        salida = medio
Si x < a[ medio ]
        Búsqueda( bajo, medio, x )
Sino
        Búsqueda( medio, alto, x )
```

RECURSION
Propiedades de las definiciones o algoritmos recursivos.

- NO debe generar una llamada infinita a sí mismo.
- Una función recursiva f debe definirse en términos que no impliquen a f al menos en un argumento o grupo de argumentos.
- Debe existir una salida de la secuencia de llamadas recursivas.
- Cualquier caso de definición recursiva tiene que reducirse a la larga a alguna manipulación de uno o casos más simples no recursivos.

RECURSION: algoritmo de Euclides para MCD

Deseamos obtener el MCD entre dos números positivos mayores a 0.

Esto se logra fijando las siguientes reglas:

mcd(a, b) = a si b = 0

mcd(a, b) = mcd(b, c) si b > 0

donde c = resto de la división entre a y b

Algoritmo RECURSIVO

```
FUNCION mcd( a, b )
Si  b = 0
    mcd = a
Sino
    div = entero( a / b )
    c = a – div * b
    mcd( b, c )
```

Resultados Ejemplo

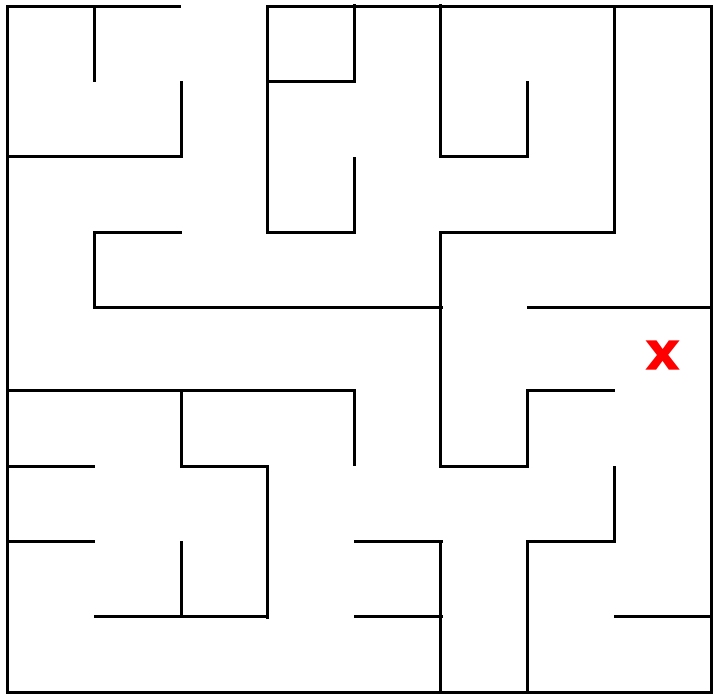
mcd (25, 5) = mcd (5, 0) = 5

mcd (21, 6) = mcd (6, 3) = mdc(3, 0) = 3

mcd (57, 23) = mcd (23, 1) = mcd (1, 0) = 1

mcd (35, 16) = mcd (16, 3) = mcd (3, 1) = mcd (1, 0) = 1

PROBLEMA: encontrar la salida a un laberinto



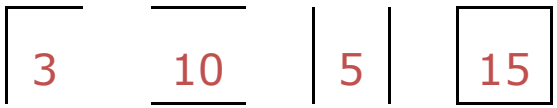
El objetivo del juego consiste en desplazar la persona (x) de una a otra celda del laberinto hasta encontrar la salida.

Se asume que las rayas negras se corresponden con paredes y no pueden ser atravesadas por la persona.

No es objetivo encontrar el camino mas corto, solo importa encontrar un camino.

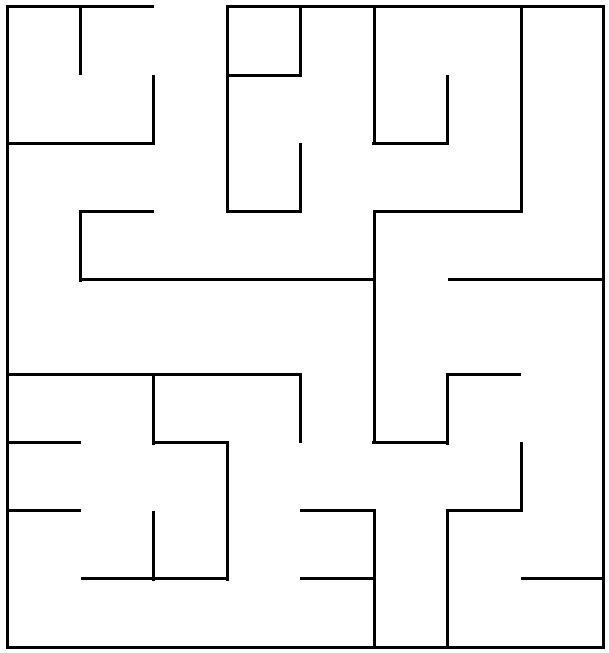
Paso 1: acordar un modo de representación del modelo.

Proponemos llamar a las celdas que tienen una pared cerrada con un número.



Las celdas que tienen más de una pared cerrada, se les asigna un número correspondiente a la sumatoria de las paredes cerradas.

PROBLEMA: encontrar la salida a un laberinto



7	3	4	15	7	3	6	7
9	12	5	3	4	13	5	5
3	10	4	13	1	10	12	5
5	11	8	10	12	3	10	12
9	10	10	10	2	1	10	12
11	6	11	6	5	13	3	4
11	0	6	1	8	2	12	5
3	12	12	1	14	5	3	12
9	10	10	8	14	13	9	14

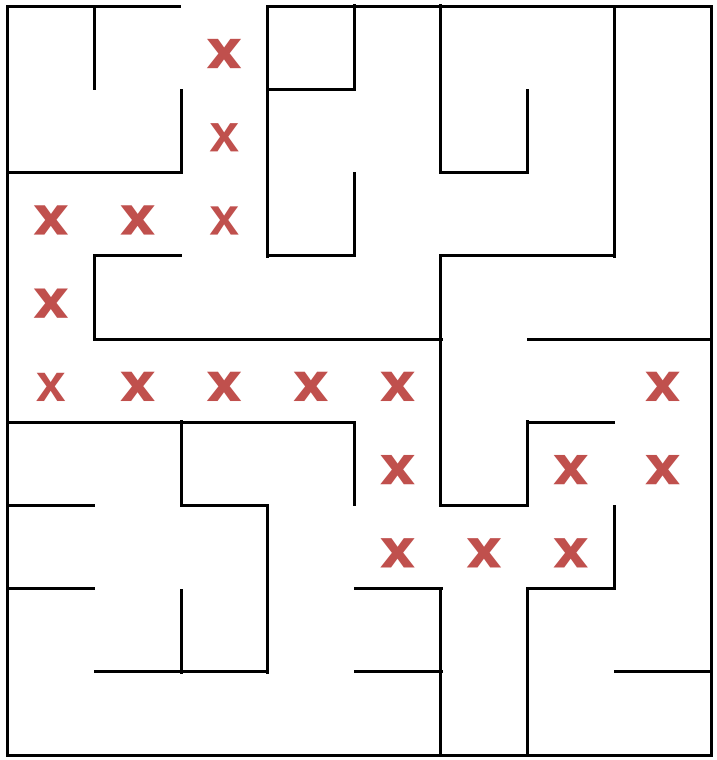
De acuerdo con la representación seleccionada.

SOLUCION: encontrar la salida a un laberinto

El Programa: Recorrer(8, 5, { })

```
Recorrer( x, y, visitadas )
SI (x < 0)  O  (x > 8)  O  (y < 0 )  O  ( y > 8 )  ENTONCES
    MostrarSalida( visitadas )
SINO
    SI FueVisitada( visitadas, x, y )
        RETORNO
    visitadas = visitadas + { x, y }
    SI matriz( x, y ) esta en lista( 1, 2, 3, 4, 5, 6, 7) ENTONCES
        Recorrer( X, Y-1, visitadas)          MOVER HACIA ARRIBA
    SI matriz( x, y ) esta en lista( 1, 4, 5, 8, 9, 12, 13) ENTONCES
        Recorrer( X, Y+1, visitadas)          MOVER HACIA ABAJO
    SI matriz( x, y ) esta en lista( 2, 4, 6, 8, 10, 12, 14) ENTONCES
        Recorrer( X-1, Y, visitadas)          MOVER HACIA IZQUIERDA
    SI matriz( x, y ) esta en lista( 1, 2, 3, 8, 9, 10, 11) ENTONCES
        Recorrer( X+1, Y, visitadas)          MOVER HACIA DERECHA
```

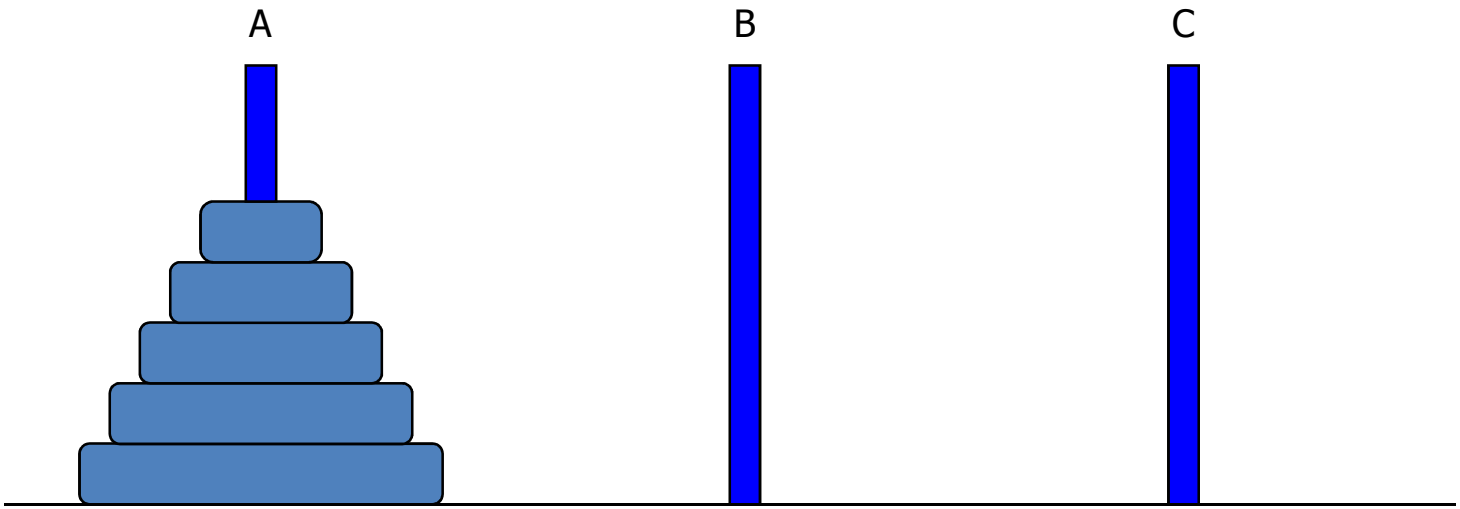
SOLUCION: encontrar la salida a un laberinto



{ 8, 5 }
{ 8, 6 }
{ 7, 6 }
{ 7, 7 }
{ 6, 7 }
{ 5, 7 }
{ 5, 6 }
{ 5, 5 }
{ 4, 5 }
....

{ 3, 5 }
{ 2, 5 }
{ 1, 5 }
{ 1, 4 }
{ 1, 3 }
{ 2, 3 }
{ 3, 3 }
{ 3, 2 }
{ 3, 1 }

RECURSION: TORRES DE HANOI



El objetivo del juego es desplazar de a una las piezas del poste A al poste C, usando como soporte el poste B, de manera tal que nunca una pieza de tamaño menor se encuentre debajo de una pieza de tamaño mayor.

El modelo se plantea con 5 piezas pero deberíamos generalizarlo para n piezas.

Deberíamos lograr una solución general, aunque aún no sabemos si realmente tiene alguna solución.

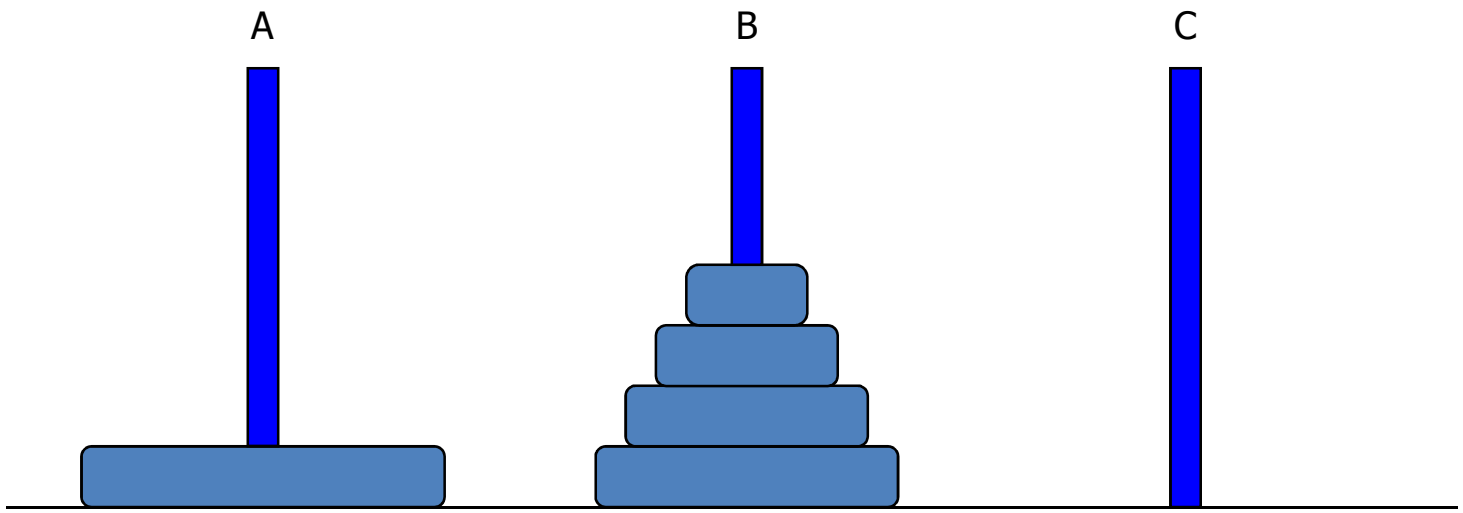
RECURSION: TORRES DE HANOI

Tratemos el problema para el caso general de n piezas.

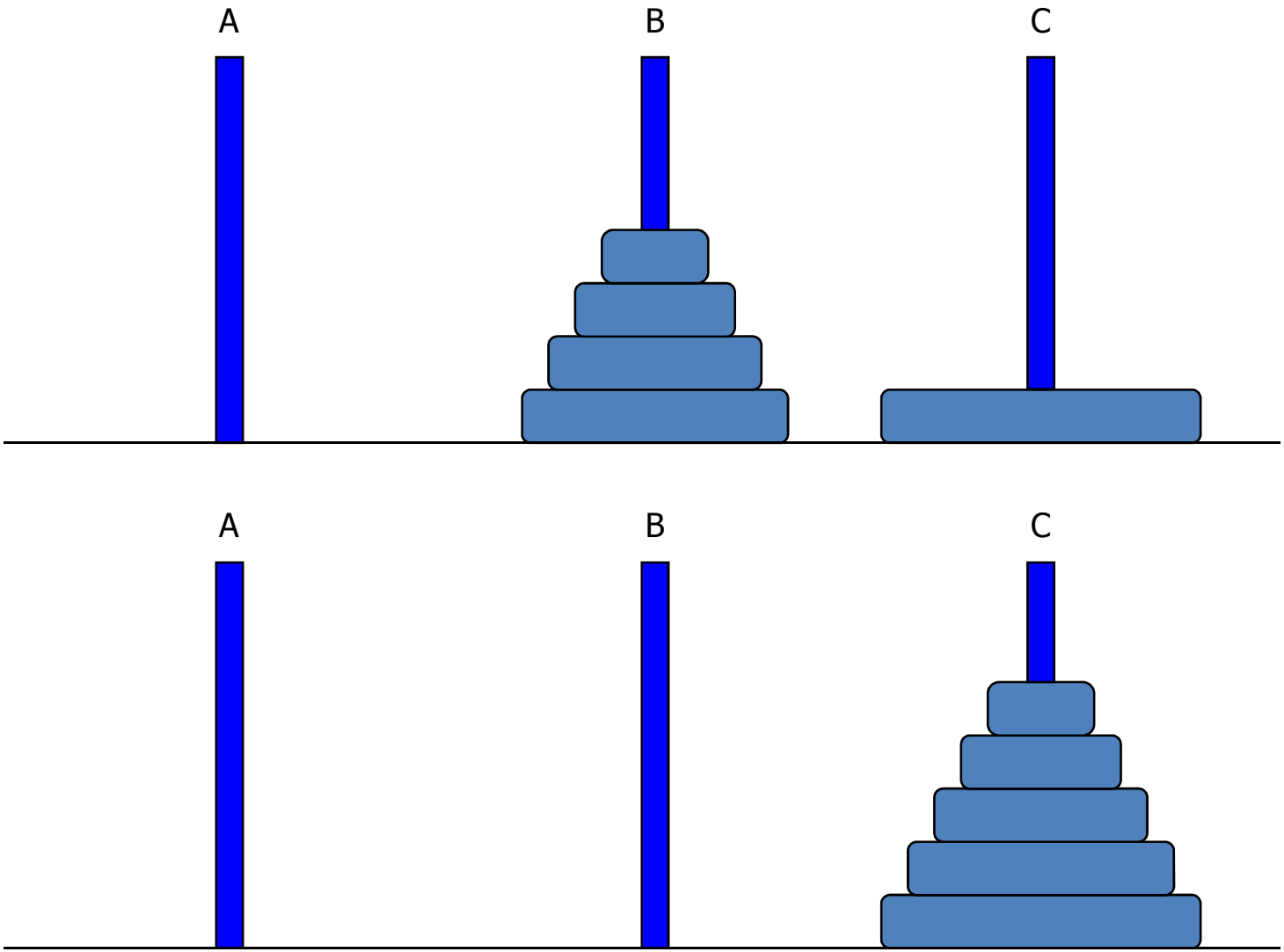
Supongamos si tengo 1 pieza, mi única acción debería ser mover la pieza del poste A al poste C.

Si logramos una solución para n piezas en función a n-1 piezas el problema estará resuelto.

Supongamos que se como mover 4 postes de A a C, también los puedo mover a B, usando C como auxiliar, en ese caso sólo me resta mover la pieza, de A a C, y luego trasladar las cuatro piezas de B a C usando como auxiliar al poste A.



RECURSION: TORRES DE HANOI



RECURSION: TORRES DE HANOI

En concreto nuestra solución será la siguiente:

1. Si n == 1, mover la única pieza del poste A al poste C.
2. Mover la pieza superior de A a B, n-1 veces, usando el poste C como auxiliar.
3. Mover la pieza restante del poste A al poste C.
4. Mover las piezas n-1 del poste B al poste C, usando A como auxiliar.

¿ Cómo transformamos esto en un programa ?

Primero debemos ponernos de acuerdo con el usuario en el nombre de las cosas, llamaremos a los postes A, B, C, y a las piezas mediante un número desde 1 hasta n, consideramos 1 a la pieza de menor tamaño y n la de mayor tamaño.

El usuario generará una entrada que será n, cantidad de piezas.

Nuestra salida indicará: MOVER LA PIEZA n DESDE EL POSTE x AL POSTE y

RECURSION: TORRES DE HANOI

El programa:

```
Torres( n, desde, hacia, auxiliar )
Si n == 1
        "MOVER PIEZA n DESDE POSTE desde HACIA POSTE hacia"
        FIN
Torres( n – 1, desde, auxiliar, hacia )
"MOVER PIEZA n DESDE POSTE desde HACIA POSTE hacia"
Torres( n – 1, auxiliar, hacia, desde )
```

La salida para n = 3

```
MOVER PIEZA 1 DESDE POSTE A HACIA POSTE C
MOVER PIEZA 2 DESDE POSTE A HACIA POSTE B
MOVER PIEZA 1 DESDE POSTE C HACIA POSTE B
MOVER PIEZA 3 DESDE POSTE A HACIA POSTE C
MOVER PIEZA 1 DESDE POSTE B HACIA POSTE A
MOVER PIEZA 2 DESDE POSTE B HACIA POSTE C
MOVER PIEZA 1 DESDE POSTE A HACIA POSTE C
```

RECURSION

La recursividad es un método poderoso usado en inteligencia artificial, su poder es que algunos conceptos complejos pueden expresarse en una forma simple. Una definición recursiva difiere de una definición circular en que tiene una forma de escapar de su expansión infinita. Este escape se encuentra en la definición o porción no recursiva o terminal de la definición.

Las fórmulas recursivas pueden aplicarse a situaciones tales como prueba de teoremas, solución de problemas combinatorios, algunos acertijos, etc.