4. RECURSIVIDAD

La recursividad forma parte del repertorio para resolver problemas en Computación y es de los métodos más poderosos y usados.

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

La recursividad es un concepto fundamental en matemáticas y en computación. Una definición recursiva dice cómo obtener conceptos nuevos empleando el mismo concepto que intenta describir.

En toda situación en la cual la respuesta pueda ser expresada como una secuencia de movimientos, pasos o transformaciones gobernadas por un conjunto de reglas no ambiguas, la fórmula recursiva es un buen candidado para resolver el problema.

Los razonamientos recursivos se encuentran en la base misma de las matemáticas porque son necesarios para describir conceptos centrales como el de número:

Ejemplo:

1. Factorial

Definición:

factorial (n) = n!   si n > 0
factorial (n) = n*n-1*n-2* ... * 1   si n > 0

el valor de 0! se define como

factorial (0) = 1

Su definición recursiva es:

factorial (n) = 1   si n = 0
factorial (n) = n* factorial (n-1)   si n > 0

así para calcular el factorial de 4:

factorial (4) = 4 * factorial (3) = 4 * 6 = 24
factorial (3) = 3 * factorial (2) = 3 *2 = 6
factorial (2) = 2 * factorial (1) = 2 * 1 = 2
factorial (1) = 1 * factorial (0) = 1 * 1 = 1

```
estática int factorial (int n)
comienza
    si n = 0 entonces
```

```
        regresa 1
    otro
        regresa factorial (n-1) * n
termina
```

Versión iterativa:

```
estática int factorial (int n)
comienza
    fact ← 1
    para i ← 1 hasta n
        fact ← i * fact
    regresa fact
termina
```

Las definiciones recursivas de funciones en matemáticas que tienen como argumentos números enteros, se llaman relaciones de recurrencia.

Forma de una ecuación de recurrencia:

$c_o a_r + c_1 a_{r-1} + c_2 a_{r-2} + \ldots + c_k a_{r-k} = f(r)$ función matemática discreta

donde c_i son constantes, es llamada una ecuación de recurrencia de coeficientes constantes de orden k, condicionada a que c_0 y $c_k = 0$.

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.

Ejemplo:

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.

Un conjunto de objetos está definido recursivamente siempre que:

(B) algunos elementos del conjunto se especifican explícitamente

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

donde

(B) se llama base

(R) se llama cláusula recursiva

Observaciones:

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

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.

Ejemplos:

1. Números de Fibonacci

Los números de Fibonacci se definen como:

$F_N = F_{N-1} + F_{N-2}$ para $N \geq 2$

$F_0 = F_1 = 1$

que definen la secuencia:

1,1,2,3,5,8,13,21,34,55,89,144, ...

Por ejemplo, si quisiéramos encontrar Fibonacci de 5, entonces:

Fibonacci (5) = Fibonacci (4) + Fibonacci (3)
Fibonacci (4) = Fibonacci (3) + Fibonacci (2)
Fibonacci (3) = Fibonacci (2) + Fibonacci (1)
Fibonacci (2) = Fibonacci (1) + Fibonacci (0)

De manera iterativa el algoritmo podría ser:

```
estática int Fibonacci (int n)
// versión iterativa
```

```
comienza
        fibo [0] ← 1
        fibo [1]← 1
        para i ←1 hasta n
                fibo [i] ← fibo [i-1]+ fibo [i-2]
        regresa fibo [n]
termina
```

Resolviendo el mismo problema pero de manera recursiva, obtenemos:

```
estática int Fibonacci (int n)
//versión recursiva
comienza
        si n ≤ 1 entonces
                regresa 1
        otro
                regresa Fibonacci (n-1) + Fibonacci (n-2)
termina
```

2. Considere la siguiente ecuación recurrente:

$$a_n = a_{n-1} + 2^n$$
$$a_0 = 1$$

```
estática int  a (int n)
comienza
        si n = 0 entonces
                regresa1
        otro
                regresa 2^n + a (n-1)
termina
```

Los valores de los términos de una sucesión pueden darse de manera explícita mediante fórmulas como:

$$S_n = n^4 - 3n^3 - 2n$$

pero también pueden definirse por medio de descripciones que involucren otros términos que les anteceden en la sucesión.

3. Considere las siguientes sucesiones:

a) Sum (n) = $\sum_{i=0}^{i=n} \frac{1}{i!}$

b) Sum (0) = 1
Sum (n+1) = Sum (n) + 1/(1/(n+1)!)

c) Seq (0) = 1
Seq (n+1) = (n+1) / Seq(n)

d) T (1) = 1
T (n) = 2T (n div 2)

e) $$\sum_{i=a}^{i=n} i = \begin{cases} i & \text{si } i = a \\ n + \sum_{i=a}^{n-1} i & \text{si } a < n \end{cases}$$

por ejemplo:

$$\sum_{i=1}^{i=n} i = 1$$

$$\sum_{i=2}^{i=5} i = 5 + \sum_{i=2}^{i=4} i = 5 + 4 + \sum_{i=2}^{i=3} i = 5 + 4 + 3 + \sum_{i=1}^{i=2} i = 5 + 4 + 3 + 2 + \sum_{i=1}^{i=1} i = 5 + 4 + 3 + 2 + 1 = 15$$

```
estática int suma (int i, int n)
comienza
        si i = n entonces
                regresa i
        otro
                regresa n + suma (i,n-1)
termina
```

4. *Definición recursiva para elevar un número a una potencia:* Un número elevado a la potencia cero produce la unidad; la potencia de un número se obtiene multiplicándolo por sí mismo elevando a la potencia menos uno. Por ejemplo:

$3^2 = 3*(3^1) = 3*(3*3^0) = 3*(3*1) = 3*(3) = 9$

```
estática int potencia (int n, int k)
comienza
        si k=0 entonces
                regresa 1
        otro
                regresa n * potencia (n,k-1)
termina
```

5. Número de Combinaciones

Recursivamente, podemos definir el número de combinaciones de m objetos tomados de n, denotado (n,m) para $n \geq 1$ y $0 \leq m \leq n$ por:

(n,m) = 1	si m = 0 ó m = n ó n = 1
(n,m) = (n-1, m) + (n-1,m-1)	en otro caso

```
estática int  combinaciones (int n, int m)
comienza
        si m = 0 o m = n o n = 1 entonces
                regresa 1
        otro
                regresa combinaciones (n-1,m) + combinaciones (n-1,m-1)
termina
```

6. Algoritmo de Euclides

Este algoritmo es considerado el más viejo no trivial.

El paso esencial que garantiza la validez del algoritmo consiste en mostrar que el máximo común divisor (mcd) de a y b, ($a > b \geq 0$), es igual a a si b es cero, en otro caso es igual al mcd de b y el remanente de a dividido por b, si $b > 0$.

```
estática int  mcd (int a,b)
comienza
        si b = 0
                regresa a
        otro
                regresa mcd (b, a mod b)
termina
```

Ejemplos:

mcd (25, 5) = mcd (5,0) = 5
mcd (18,6) = mcd (6,0) = 6
mcd (57, 23) = mcd (23, 1) = mcd (1,0) = 1
mcd (35, 16) = mcd (16,3) = mcd (3, 1) = mcd (1, 0) = 1

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.

4.1 Técnica de diseño "dividir para vencer"

Muchas veces es posible dividir un problema en subproblemas más pequeños, generalmente del mismo tamaño, resolver los subproblemas y entonces combinar sus soluciones para obtener la solución del problema original.

Como los subproblemas obtenidos generalmente son del mismo tipo que el problema original, la estrategia "**dividir para vencer**" generalmente se implementa como un algoritmo recursivo.

```
Solución DPV (Problema T)
comienza
        si T ≤ pequeño entonces
                regresa soluciónTrivial (T)
        otro
                comienza
                        divide (T,T1,T2, …, Ta)
                        para  i ← 1 hasta a
                                Solucióni a ← DPV (Ti)
                        regresa combina (Solución1, Solución2, … , Solucióna)
                termina
termina
```

Dividir para vencer es una técnica natural para las estructuras de datos, ya que por definición están compuestas por piezas. Cuando una estructura de tamaño finito se divide, las últimas piezas ya no podrán ser divididas.

Ejemplos:

1. Encontrar el número mayor de un arreglo de enteros:

```
estática int  mayor1 (objeto [ ] A, int limIzq, int limDer)
comienza
        si limIzq = limDer entonces
                regresa  A[limIzq]
        otro
                comienza
                        m ← (limIzq + limDer) div 2
                        mayorIzq ← mayor1 (A,limIzq,m)
                        mayorDer ← mayor1 (A, m +1, limDer)
                        si mayorIzq > mayorDer entonces
                                mayor1 ← mayorIzq
                        otro
                                regresa mayorDer
                termina
termina
```

Otra forma de resolver este problema es la siguiente:

```
estática int  mayor1 (objeto [ ] A, int limIzq, int limDer)
//versión 2
comienza
        si limIzq = limDer entonces
                regresa A[limIzq]
        otro
                comienza
                        m ← (limIzq + limDer) div 2
                        mayorIzq ← mayor1 (A, limIzq, m)
                        mayorDer ← mayor1 (A, m +1, limDer)
                        si mayorIzq > mayorDer entonces
                                regresa mayorIzq
                        otro
                                regresa mayorDer
                termina
termina

estática int  mayor2 (objeto [ ] A, int limIzq)
comienza
        limDer = A.longitud-1
        si limIzq = limDer entonces
                regresa A[limIzq]
        otro
                si limDer-limIzq = 1 entonces
                        si A[limDer] ≥ A[limIzq] entonces
                                regresa A[limDer]
                        otro
                                regresa A[limIzq]
                otro
                        comienza
                                m ← mayor2 (A,limIzq+1)
                                si A[limIzq] ≥ m entonces
                                        regresa  A[limIzq]
                                otro
                                        regresa m
                        termina
termina
```

2. Inversión de una cadena

```
estática Cad invierte (Cad cadena, int limIzq, int limDer)
comienza
        si limDer = limIzq entonces
                regresa cadena
        otro
```

```
                regresa invierte (cadena, limDer,limIzq+1) + cadena [limIzq]
termina
```

3. Cadenas de la forma $a^n b^n$

La notación $a^n b^n$ se utiliza para denotar cadenas que consisten de n a's consecutivas seguidas de n b's consecutivas.

Es decir denota al lenguaje:

L = {w| w es de la forma $a^n b^n$ para $n \geq 0$ }

La gramática es parecida a la de los palíndromes. Debemos checar que la primera sea una a y la última una b.

<palabraLegal> := cadena vacía | a<palabraLegal>b

El algoritmo para reconocer cadenas de esta forma, podría ser:

```
estática bool reconocer  (Cad w, int limIzq, int limDer)
comienza
        si limIzq > limDer entonces
                regresa verdadero
        otro
                si w[limIzq] = 'a' & w[limDer] = 'b' entonces
                        regresa reconocer (w, limIzq+1, limDer-1)
                otro
                        regresa falso
termina
```

4. Palindromes

Un palindrome es una cadena que se lee (escribe, en este caso) igual de izquierda a derecha que de derecha a izquierda. Escribir una función que determine cuando una cadena es o no un palindrome.

```
estática bool palindrome (Cad c, int limIzq)
comienza
        si limIzq > c.longitud   entonces
                regresa verdadero
        otro
                si c [limIzq] = c [limDer] entonces
                        regresa palindrome (c, limIzq+1)
                otro
                        regresa falso
termina
```

5. Búsqueda binaria

```
estática bool busbin (objeto [ ] A, int limIzq, int limDer, objeto valor)
comienza
        si limIzq = limDer entonces
                regresa A [limDer].igual (valor)
        otro
                comienza
                        m ← (limIzq + limDer) div 2
                        si A [m].igual (valor)  entonces
                                regresa verdadero
                        otro
                                si valor.mayor(A [m])  entonces
                                        regresa BusBin (A,m+1,limDer, valor)
                                otro
                                        regresa BusBin (A,limIzq,m-1, valor)
                termina
termina
```

6. MergeSort

Suponga dos arreglos ordenados A[1..n] y B[1..m] que se desean juntar en un solo arreglo ordenado C[1..n+m]

```
estática void Merge (Objeto [ ] A, Objeto [ ] B, Objeto [ ] C)
comienza
        i ← 1; j ←1; k ←1;
        mientras i ≤ n & j ≤ m
                comienza
                        si A[i] < B[j] entonces
                                C[k] ← A[i]; i++
                        otro
                                C[k] ← B[j]; j++
                        k++
                termina
        Copia (C,k,A,i)
        Copia (C,k,B,j)
termina
```

```
void MergeSort (Objeto [] A, int limIzq, int limDer)
comienza
        si limIzq < limDer entonces
                comienza
                        i  ← (limIzq+limDer) div 2
                        MergeSort (A, limIzq, i)
                        MergeSort (A, i+1, limDer)
                        Merge (A,A,A)
                termina
termina
```

7. Fractal cuadrado

Dibujar una figura complicada con absoluto realismo empleando una computadora puede resultar en un trabajo bastante complicado para cualquier programador o artista. En 1975, un matemático francés llamado *Bernoit Mandelbrot* terminó una serie de investigaciones que duraron cerca de 20 años. Mandelbrot llamó al resultado de sus investigaciones *Geometría Fractal*. Un fractal es un dibujo formado por una geometría igual que la del dibujo pero de un tamaño menor. Un fractal se crea por cálculos repetitivos similares que los que se necesitaron para efectuar la parte inicial del dibujo. El fractal debe de seguir cierto orden para que no se produzcan dibujos que al final resultarían en un amontonamiento de líneas. Utilizando este método se pueden llegar a producir dibujos de una gran calidad y similitud con la realidad.

```
estática void cuadrado (int x,int y,int r)
comienza
        si r > 0 entonces
                comienza
                        pintaCuadrado (x,y,r)
                        cuadrado (x-r,y+r, r div 2)
                        cuadrado (x+r,y+r, r div 2)
                        cuadrado (x-r,y-r, r div 2)
                        cuadrado (x+r,y-r, r div 2)
                termina
termina
```

8. Dibujar recursivamente las marcas de una regla:

```
estática  void regla (int ini, int fin, int n)
comienza
        si n > 0 entonces
                comienza
                        m ← (ini+fin) div 2
                        marca (m,n)
                        regla (ini, m, n-1)
                        regla (m, fin, n-1)
                termina
termina
```

```
regla (0,4,2)
        marca (2,2)
        regla (0,2,1)
                marca (1,1)
                regla (0,1,0)
                regla (1,2,0)
        regla (2,4,1)
                marca (3,1)
                regla (2,3,0)
```

regla (3,4,0)

10. Torres de Hanoi

```
estática void hanoi (Torre A, Torre B, Torre C, int n)
// se desean mover n discos de la torre A a la B utilizando a C como auxiliar
comienza
        si n = 1 entonces
                A --> B                         //se mueve un disco de A a B
        otro
                comienza
                        hanoi (A,C,B,n-1)
                        A --> B
                        hanoi (C,A,B,n-1)
                termina
termina
```

Ejercicios:

1. Desarrolle una función para calcular recursivamente el número de ocurrencias del elemento *elem* en el vector *A*.
2. Desarrolle una función recursiva para saber si un carácter dado se encuentra presente en una cadena de caracteres.
3. Escriba una función recursiva para indicar si una cadena *st1* ocurre dentro de una cadena *st2*.
4. Desarrolle una función recursiva para determinar si dos vectores son iguales.
5. Escriba el algortimo para la función de Ackermann, que está definida como:

$A(0,n) = n+1$ para $n \geq 0$
$A(m,0) = A(m-1, 1)$ para $m > 0$
$A(m,n) = A(m-1, A(m,n-1))$ para $m > 0$ y $n > 0$

6. Suponiendo que solo existe una rutina *escribeDigito* para escribir en la pantalla un dígito (0-9) pasado como parámetro, desarrolle un procedimiento recursivo que imprima un entero de cualquier cantidad de dígitos.
7. Multiplicación de números naturales. El producto de a*b donde a y b son enteros positivos, se puede definir como a sumando a sí mismo un número de veces igual a b.

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

8. Considere un arreglo de enteros. Escriba algoritmos recursivos para calcular:
   (a) la suma de los elementos del arreglo
   (b) el producto de los elementos del arreglo
   (c) el promedio de los elementos del arreglo

9. Determine qué calcula la siguiente función recursiva y escriba una función iterativa que realice lo mismo:

```
estática  int f (int n)
comienza
        si n= 0
                regresa 0
        otro
                regresa n + f (n-1)
termina
```