ESTRUCTURA DE DATOS

Recursividad

RECURSIVIDAD

Se dice que una función es recursiva cuando se define en función de si misma.

No todas la funciones pueden llamarse a si mismas, sino que deben estar diseñadas especialmente para que sean recursivas, de otro modo podrían conducir a bucles infinitos, o a que el programa termine inadecuadamente.

Tampoco todos los lenguajes de programación permiten usar recursividad.

C permite la recursividad. Cada vez que se llama a una función, se crea un juego de variables locales, de este modo, si la función hace una llamada a si misma, se guardan sus variables y parámetros, usando la pila, y la nueva instancia de la función trabajará con su propia copia de las variables locales. Cuando esta segunda instancia de la función retorna, recupera las variables y los parámetros de la pila y continua la ejecución en el punto en que había sido llamada.

Es una alternativa diferente para implementar estructuras de repetición (ciclos). Los módulos se hacen llamadas recursivas.

Caso recursivo:

Una solución que involucra volver a utilizar la función original, con parámetros que se acercan más al caso base.

Los pasos que sigue el caso recursivo son los siguientes:

1. El procedimiento se llama a sí 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

¿Cuándo usar recursividad?

- Para simplificar el código.
- Cuando la estructura de datos es recursiva ejemplo : ordenamientos, búsquedas, árboles.

¿Cuándo no usar recursividad?

- Cuando los métodos usen arreglos largos.
- Cuando el método cambia de manera impredecible de campos.
- Cuando las iteraciones sean la mejor opción.

Ejemplo de Recursividad

Prodríamos crear una función recursiva para calcular el factorial de un número entero.
El factorial se simboliza como n!, se lee como "n factorial", y la definición es:

$$n! = n * (n-1) * (n-2) * ... * 1$$

Hay algunas limitaciones:

- No es posible calcular el factorial de números negativos, no está definido.
- El factorial de cero es 1.

De modo que una función bien hecha para cálculo de factoriales debería incluir un control para esos casos:

```
Funcion Factorial(n)
      if     n == 1 entonces              return 1
     else     return n * factorial(n-1)
```

Función factorial (n!)

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

Ejemplos:

0! = 1
1! = 1 ➔ 1! = 1 * 0!
2! = 2 ➔ 2! = 2 * 1!
3! = 6 ➔ 3! = 3 * 2!
4! = 24 ➔ 4! = 4 * 3!
5! = 120 ➔ 5! = 5 * 4!

Producto de los números comprendidos entre 1 y n.

Función factorial (n!)

```
int factorial(int n)
{
   if (n == 0)
     return 1;
   else
     return  n * factorial(n – 1);
}
```

Caso(s) base: punto(s) donde existe una condición para la cual no se requiere volver a llamar a la misma función.

Caso(s) recursivo(s): punto(s) donde se requiere volver a llamar a la misma función.

Función factorial (n!)

```
int factorial(int n)
{
    if (n < 2) // ya que 1! = 1
        return 1;
    return n * factorial(n - 1);
}
```

Simplificación de la función factorial.

Función factorial (n!)

```
int factorial (int n)
{
    int factorial = 1;
    for (int i = n; i > 1; i--) // for (int i = 2; i <=n; i++)
        factorial *= i;
    return factorial;
}
```

Solución iterativa (no recursiva).

Sucesión Fibonacci

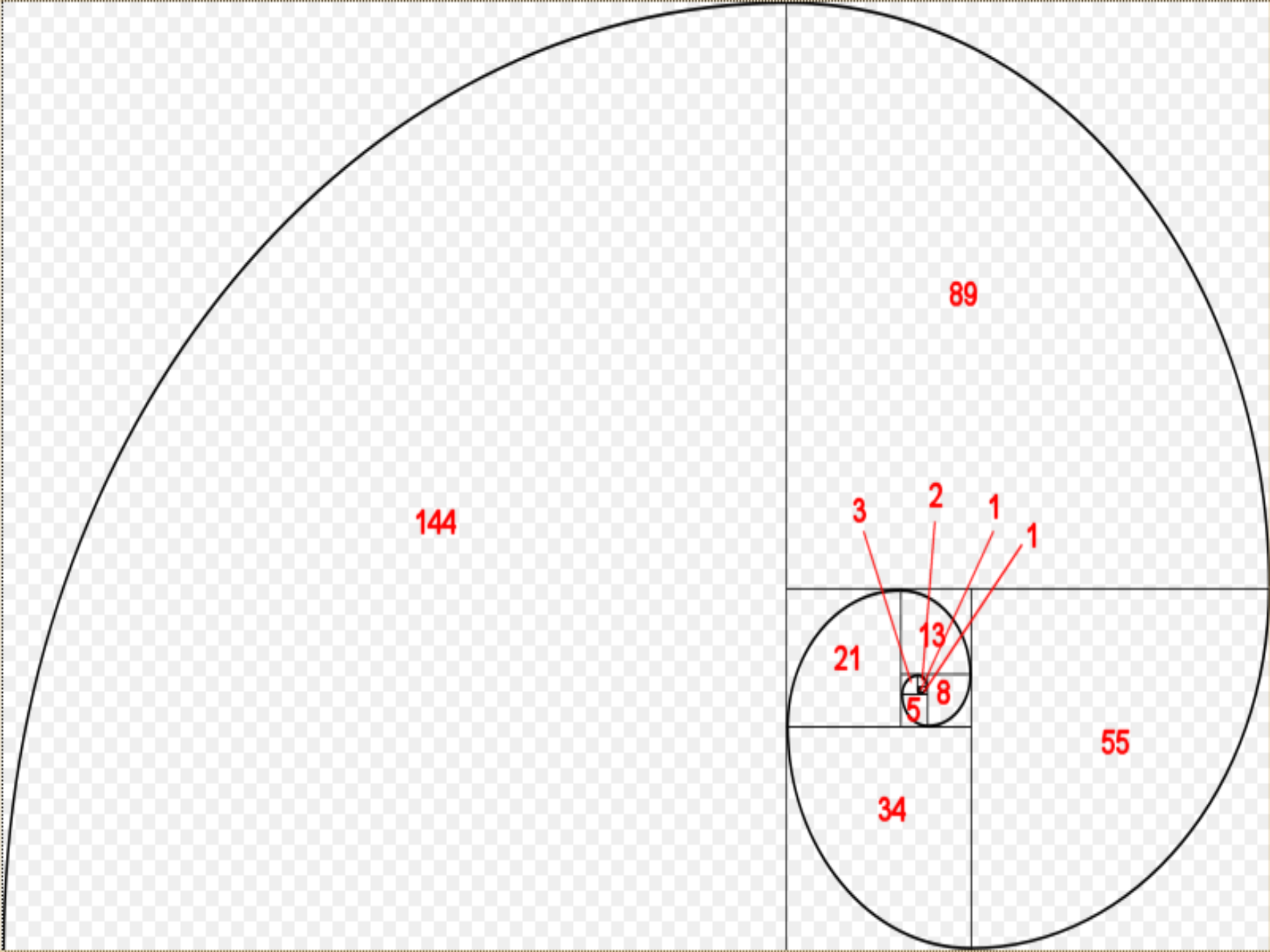
En matemáticas, la **sucesión de Fibonacci** es la siguiente sucesión infinita de números naturales:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 \ldots$$

El primer elemento es 0, el segundo es 1 y cada elemento restante es la suma de los dos anteriores:

$$f_i = \begin{cases} 0 & \text{si } i = 0 \\ 1 & \text{si } i = 1 \\ f_{(i-2)} + f_{(i-1)} & \text{si } i > 1 \end{cases}$$

A cada elemento de esta sucesión se le llama **número de Fibonacci**


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

Torres de Hanoi

1.Sólo se puede mover un disco cada vez.
2.Un disco de mayor tamaño no puede descansar sobre uno más pequeño que él mismo.
3.Sólo puedes desplazar el disco que se encuentre arriba en cada varilla.