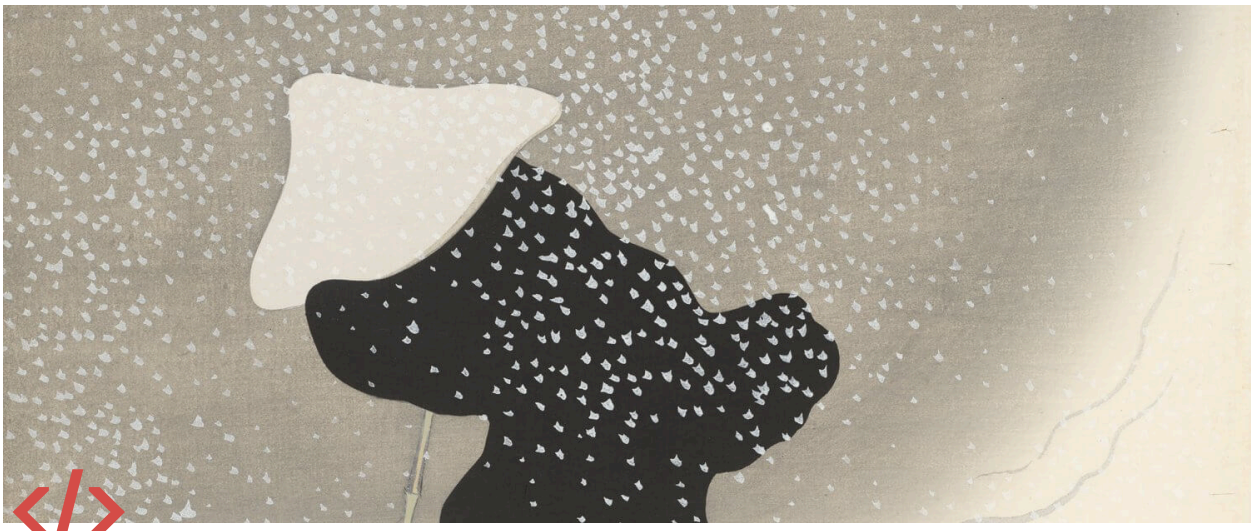

# Recursividad de Cola

Etiquetas

## Diferencias entre Recursividad de Pila y Recursividad de Cola

Antes de entrar al código en *Wolfram Mathematica*, es útil diferenciar estos dos tipos de recursión:

## 1. Recursividad de Pila (o Recursión Normal)

- Cada llamada recursiva genera un nuevo marco en la pila de llamadas.
- La función sigue ejecutando instrucciones **después** de la llamada recursiva.
- Esto puede generar un alto consumo de memoria si la profundidad de recursión es grande.
- Ejemplo típico: el factorial sin optimización.

$$\text{fact}(n) = n \times \text{fact}(n-1)$$

- En código de *Wolfram Mathematica*:

```
fact[n_] := If[n == 0, 1, n * fact[n - 1]]
```

- Problema: A medida que `n` crece, se acumulan muchas llamadas en la pila.

---

### 2. Recursividad de Cola

- La llamada recursiva es la **última operación** en la función.
- Se usa un **parámetro adicional** para llevar la acumulación de resultados.
- Ejemplo con el factorial optimizado:

```
factTail[n_, acc_] := If[n == 0, acc, factTail[n - 1, n * acc]]
factTail[n_] := factTail[n, 1]
```

Aquí, la variable `acc` lleva el resultado acumulado, permitiendo que *Mathematica* elimine la pila de llamadas.

---

# Conceptos Teóricos de la Recursión de Cola

La **recursión de cola** es un tipo especial de recursión en la que la llamada recursiva es la **última operación** que realiza la función antes de devolver el resultado.

## ¿Por qué es importante?

- **Eficiencia**: Evita el problema de **desbordamiento de pila (stack overflow)** que ocurre en la recursión normal cuando la cantidad de llamadas es grande.
- **Estructura**: Se usa una **variable acumuladora** para llevar el resultado parcial y evitar cálculos adicionales después de la recursión.

## Comparación con la Recursión Normal

### Recursión Normal (de Pila)

```
Factorial[n_] := If[n == 0, 1, n * Factorial[n - 1]]
```

Aquí, la llamada recursiva `Factorial[n - 1]` está **dentro de una multiplicación**, lo que significa que *Mathematica* debe recordar `n` en cada paso.

**Recursión de Cola**

```
FactorialTail[n_] := Module[{factAux},
  factAux[k_, acc_] := If[k == 0, acc, factAux[k - 1, k * acc]];
  factAux[n, 1]
]
```

Aquí, la **última operación** en cada llamada es la recursión en `factAux[k - 1, k * acc]`, permitiendo que Mathematica *optimice* la ejecución.

---

# 📝 Ejemplo 1: Factorial con Recursión de Cola

Vamos a definir la **versión recursiva de cola** del factorial usando un **acumulador** para mantener el resultado parcial.

```
FactorialTail[n_] := Module[{factAux},
  factAux[k_, acc_] := If[k == 0, acc, factAux[k - 1, k * acc]];
  factAux[n, 1]
]
```

📌 **Explicación del código:**

1. **Se usa** `Module` para definir una función auxiliar `factAux` dentro de la función principal.
2. **Parámetro** `acc` **(acumulador)**: lleva el resultado parcial del cálculo.
3. **Condición base**: cuando `k == 0`, devolvemos `acc`, el resultado final.
4. **Llamada recursiva en cola**: `factAux[k - 1, k * acc]` es la última operación, asegurando que no haya cálculos pendientes.

🔷 **Ejemplo de uso:**

```
FactorialTail[5] (* Devuelve 120 *)
FactorialTail[10] (* Devuelve 3628800 *)
```

# 🏁 Resumen

| Método | Recursión Normal | Recursión de Cola |
| --- | --- | --- |
| **Cómo funciona** | Acumula llamadas en la pila | Reemplaza cada llamada |
| **Optimización (TCO)** | No se optimiza fácilmente | Puede optimizarse eliminando la pila |
| **Uso de memoria** | Puede ser grande para `n` grande | Constante |
| **Ejemplo** | `Factorial[n] = n * Factorial[n - 1]` | `FactorialTail[n] = factAux[n, 1]` |

Algoritmos Recursivos de Cola

Antes de entrar en tema, es importante entender que es la recursión “normal”, para eso te recomiendo leer el siguiente contenido...

https://medium.com/@marcoantonio.blancas/algoritmos-recursivos-de-cola-da0479a893ab

C++ Recursion (Recursive Function)

C++ Recursion (Recursive Function) - Recursion is a programming technique where a function calls itself over again and again with modified arguments until it reaches its base

https://www-tutorialspoint-com.translate.goog/cplusplus/cpp_recursion.htm?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc

Recursividad pero Optimizada!!! Recursión de Cola

La recursión de cola o tail recursion, es una técnica que optimiza el espacio en el Stack del programa para que nuestros algoritmos recursivos tengan un mejor desempeño a la hora de

https://www.youtube.com/watch?v=SIgfSYyWVjo