Reporte: Puente Conceptual - Escaleras (Programación Dinámica)

Descripción del Reto

El objetivo era modificar el problema clásico de "Subir Escaleras" (donde se permiten pasos de 1 o 2 escalones) para permitir también pasos de **3 escalones**.

Implementación

Relación de Recurrencia

Originalmente (Fibonacci): `f(n) = f(n-1) + f(n-2)`

Nueva relación (Tribonacci-like): `f(n) = f(n-1) + f(n-2) + f(n-3)`

Esto significa que para llegar al escalón `n`, podemos haber venido desde:

- El escalón `n-1` (dando 1 paso)
- El escalón `n-2` (dando 2 pasos)
- El escalón `n-3` (dando 3 pasos)

Casos Base

Para que la lógica funcione correctamente:

- `f(0) = 1` (Una forma de "estar" en el suelo: no moverse)
- `f(n) = 0` si `n < 0` (No se puede venir de escalones negativos)

Esto resulta en:

- `f(1) = f(0) + f(-1) + f(-2) = 1 + 0 + 0 = 1`
- `f(2) = f(1) + f(0) + f(-1) = 1 + 1 + 0 = 2`
- `f(3) = f(2) + f(1) + f(0) = 2 + 1 + 1 = 4`

Resultados (n=35)

Método	Resultado	Tiempo (ms)
Memoization (Top-Down)	1,132,436,852	8.8113
Tabulación (Bottom-Up)	1,132,436,852	0.7126

Observaciones:

- Ambos métodos producen el mismo resultado correcto.
- La **Tabulación** es significativamente más rápida (~10x) en este caso. Esto se debe a que utiliza un arreglo simple (`long[]`) para almacenar los resultados, lo cual es mucho más eficiente en acceso a memoria y caché que el `Dictionary<int, long>` utilizado en la versión de Memoización.

- La complejidad temporal es O(n) en ambos casos, pero la constante oculta es menor en la tabulación.

Conclusión

La transición de recursividad a programación dinámica (DP) es esencial para este tipo de problemas. La versión recursiva ingenua (sin memoización) tendría una complejidad de O(3^n), lo cual sería impracticable para n=35. Al usar DP, reducimos esto a O(n), haciendo el cálculo instantáneo.