26-2-2026

Semana 4

Taller en clase - Estudiantes

Laila Zareth Romano Guerrero & Juan Sebastian Garavito Mejia
DDYA-7

Problema

Suponga que hay n amigos, quienes pueden permanecer solos o ser emparejados con otro amigo. Cada amigo puede ser emparejado una sola vez. Encuentre el número de formas en que los amigos pueden quedar solos o emparejados

Descripción del problema

El sistema recibe un número entero n de amigos y calcula cuántas formas posibles existen para que estos se organicen, ya sea quedándose solos o formando parejas. Si tenemos pocos amigos, un cálculo recursivo daría una respuesta rápidamente, pero a lago plazo cuando el número de amigos aumente este procesamiento se quedaría atrás ya que dependiendo el n ingresado recalcularа los mismos subproblemas, haciendo que el tiempo de ejecución crezca. Por esto, el objetivo de este diseño es optimizar el algoritmo utilizando como técnica la programación dinámica, en específico la memoización. Al implementar una "memoria", el sistema almacena los resultados de cada cantidad de amigos ya calculada, evitando procesar cálculos ya hechos, para entregar el resultado final de forma rápida y eficiente.

Requerimientos (Historias de usuario)

H01- Entrada y validación de datos

- **Descripción:** El sistema debe permitir al usuario ingresar el número de amigos (n) para calcular de cuántas formas pueden estar solos o emparejados.
- **Caso exitoso:** El usuario ingresa un número entero positivo y el sistema procede a realizar el cálculo.
- **Caso no exitoso:** El usuario ingresa un valor no numérico, un número negativo o lo deja vacío.
- **Entrada:** Un número entero n que representa la cantidad de amigos.
- **Salida:** No hay salida, el dato ingresa y pasa a la lógica de cálculo.

H02- Cálculo de combinaciones

- **Descripción:** El sistema aplica como herramienta la memoización para resolver el problema de amigos, manejando casos base para tener como parámetro datos que no son realizados al utilizar la función de recurrencia encontrada, en caso de que el número de amigos ingresado por el usuario no sea un caso base y no exista su cálculo en nuestro diccionario "memo" aplicará la función de recursión $f(n) = f(n-1) + (n-1) \times f(n-2)$.
- **Caso exitoso:** El sistema utiliza un arreglo para almacenar los resultados previos, en caso de que el calculo de n no este almacenado, lo calcula y almacena para luego no tener que volver hacer el calculo por cada vez que se ingrese un dato.
- **Entrada:** El número entero n validado anteriormente.

- **Salida:** Un valor entero que representa el total de combinaciones posibles para los amigos.

H03 - Visualización de resultados

- **Descripción:** El sistema debe mostrar de forma clara al usuario el resultado final del cálculo.
- **Caso exitoso:** Se muestra en pantalla “Para n amigos, hay resultado formas posibles de estar solos o emparejados.”
- **Caso no exitoso:** El sistema finaliza de forma repentina sin mostrar el resultado debido a un error en el flujo de salida.
- **Entrada:** El resultado numérico obtenido del proceso de programación dinámica.
- **Salida:** Mensaje mostrando el resultado.

Requerimientos

- **R01:** El sistema debe capturar el número de amigos a través de una entrada de teclado**.**
- **R02:** El sistema debe implementar una solución de Programación Dinámica para evitar la redundancia de cálculos recursivos**.**
- **R03:** Debe definirse el caso base de forma correcta (para n=0, n=1 y n=2) dentro de la estructura de control.
- **R04:** El sistema debe utilizar una estructura de datos para el almacenamiento de los subproblemas resueltos.
- **R05:** La salida debe ser presentada en un formato de cadena de texto amigable para el usuario final.

Diagrama de secuencia

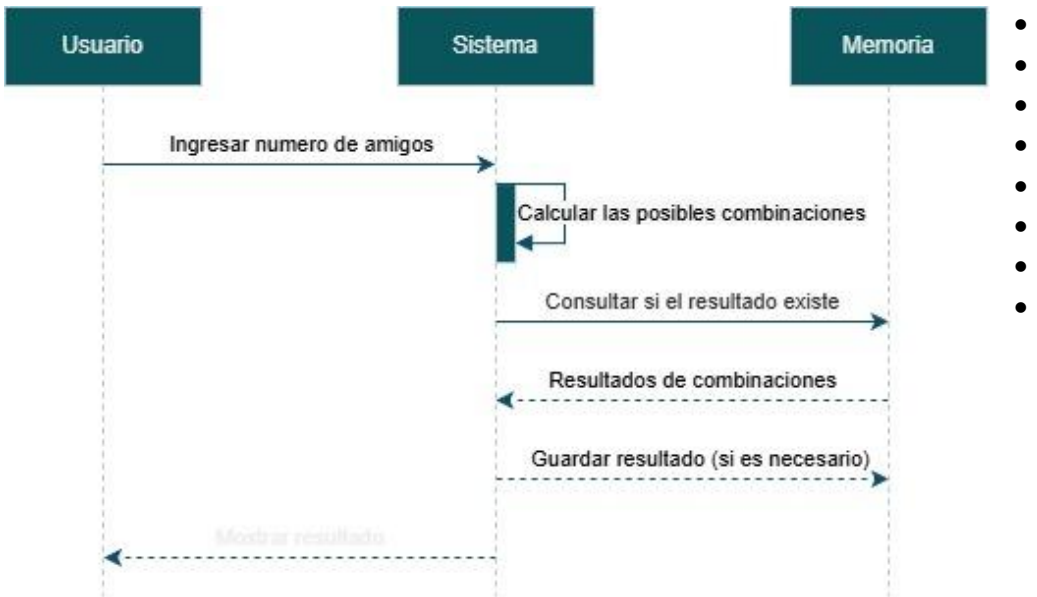


Diagrama de flujo

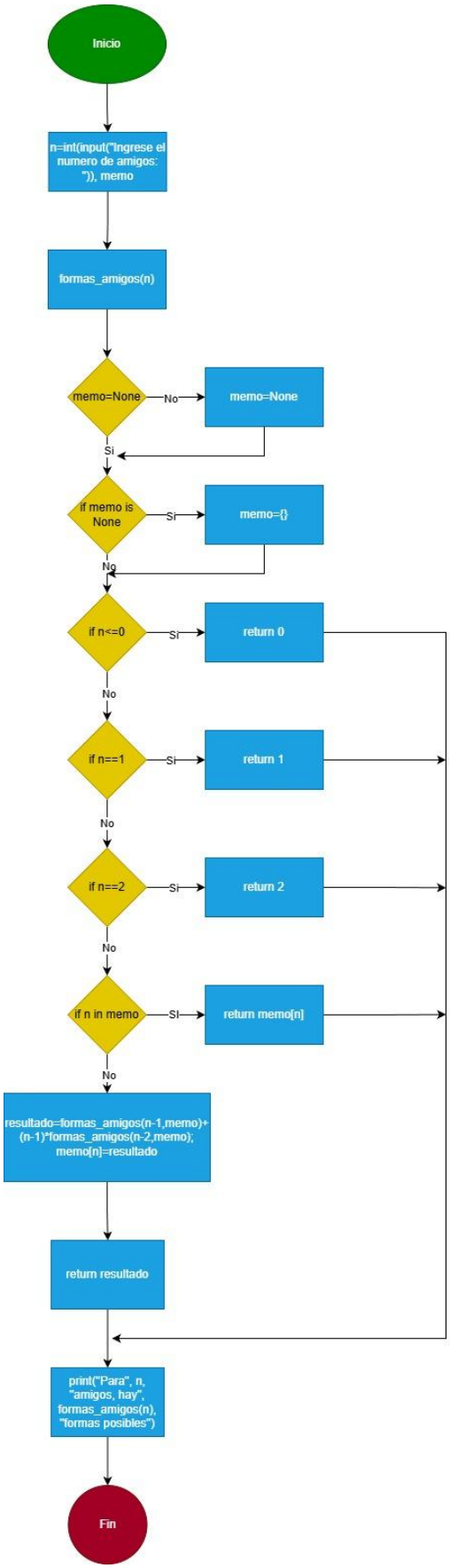


Diagrama de caso de uso

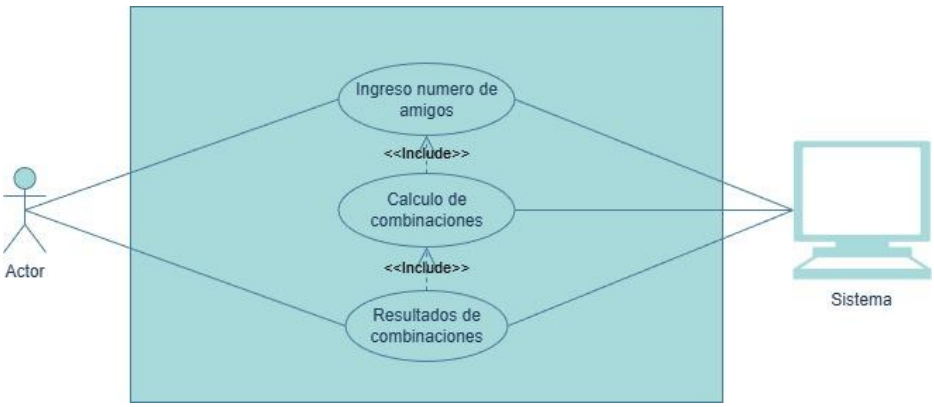


Tabla de complejidad

Línea	Cost	Times
`if memo is None:`	C_1	1 (memos is None)
`    memo = {}`	C_2	1
`if n <= 0:`	C_3	1 (n <= 0)
`    return 0`	C_4	1
`if n == 1:`	C_5	1(n == 1)
`    return 1`	C_6	1
`if n==2:`	C_6	1(n == 2)
`    return 2`	C_8	1
`if n in memo:`	C_9	n (n in memo)
`    return memo[n]`	C_9	n
`resultado = formas_amigos(n-1,memo) + (n-1)* formas_amigos(n-2,memo)`	C_{10}	N
`memo[n]= resultado`	C_{11}	n
`return resultado`	C_{12}	1
`n = int(input("Ingrese el numero de amigos:"))`	C_{13}	1
`print("Para", n, "amigos, hay", formas_amigos(n), "formas posibles de estar solor o emparejados."`	C_{14}	1

Complejidad

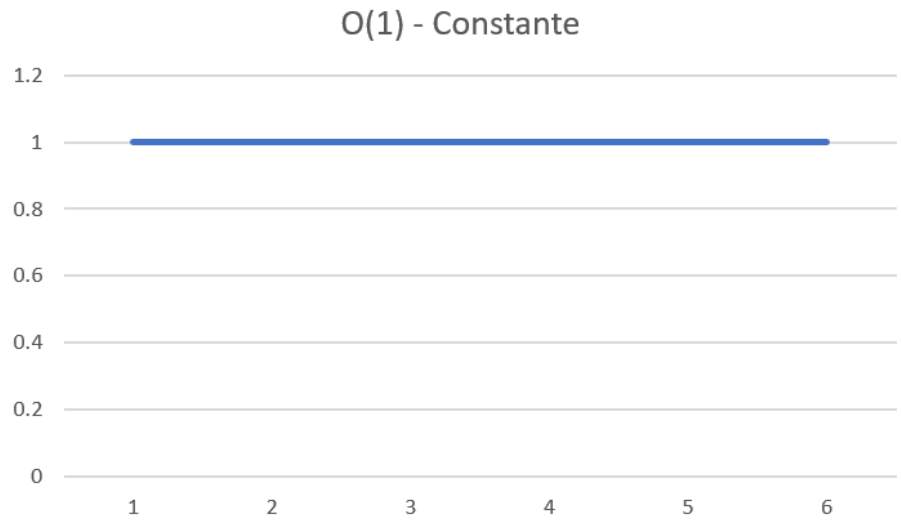
$$T(n) = T(n-1) + O(1)$$

$$T(n) = O(n)$$

Analisis de casos

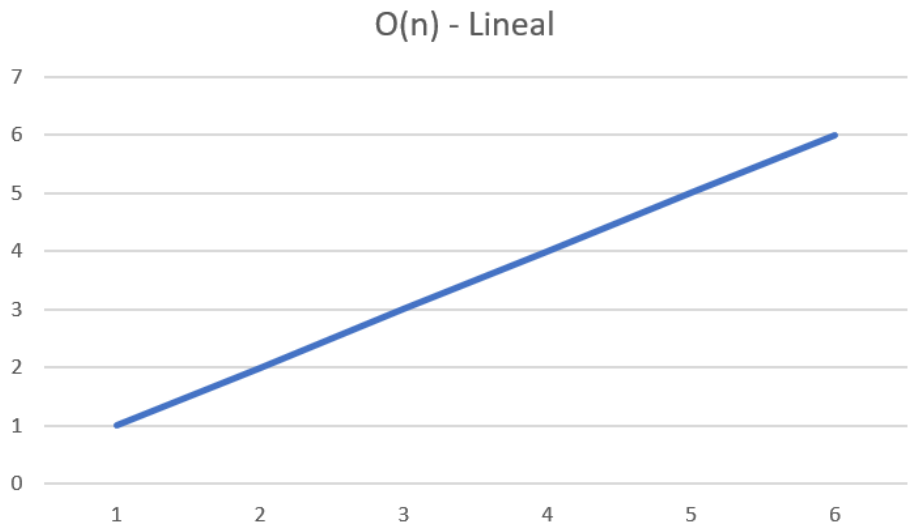
Mejor caso: $O(1)$ constante

Ocurre cuando el valor n ya ha sido calculado previamente y se encuentra en el diccionario “memo”.



Peor caso: $O(n)$ lineal

Ocurre la primera vez que se ejecuta el programa para un valor n, con el diccionario “memo” vacío.



Analisis

La implementación de programación dinámica transforma un problema de complejidad exponencial en uno de complejidad lineal $O(n)$. Esto garantiza que incluso para un número elevado de amigos que ingresen, el sistema responderá de manera eficiente al evitar recalcular los subproblemas ya resueltos.