Instituto Politécnico Nacional

Escuela Superior de Cómputo

Unidad de académica: Análisis y Diseño de Algoritmos

Actividad:

“Recursividad plana vs Memoization”

Equipo:

Solares Velasco Arturo Misael

Solis Lugo Mayra

Grupo: 3CV1

Profesor: García Floriano Andrés

Fecha:

13 de Mayo de 2024

Reporte de Práctica: Recursividad Plana y Memoization

Introducción:

En el código proporcionado, se exploran dos enfoques para abordar un problema común en informática: calcular el número de formas únicas de subir una escalera de N escalones, pudiendo avanzar de 1, 2 o 3 escalones a la vez. Estos enfoques utilizan la técnica de recursividad plana y memoization para optimizar el rendimiento y la eficiencia del cálculo.

Recursividad Plana:

La función contar_formas(n, memo={}) implementa el enfoque de recursión simple para resolver el problema. La función recibe como entrada el número de escalones n que se desea subir y un diccionario memo que almacena los resultados previamente calculados para evitar recálculos innecesarios.

- La función primero verifica si n es igual a 0, lo que significa que ya hemos subido todos los escalones y solo hay una forma de hacerlo: no subir ninguno.
- Si n es menor que 0, significa que no hay formas válidas de subir los escalones y devuelve 0.
- Si n ya está presente en el diccionario memo, devuelve el valor almacenado correspondiente.
- En caso contrario, calcula el número de formas de subir n escalones recursivamente sumando el número de formas de subir n-1, n-2 y n-3 escalones, almacenando el resultado en el diccionario memo para su uso futuro y retorna el valor calculado.

Algoritmo usando la recursividad plana

main.py > ...

```python
def contar_formas(n, memo={}):
    if n == 0:
        return 1
    elif n < 0:
        return 0
    elif n in memo:
        return memo[n]
    else:
        memo[n] = contar_formas(n - 1, memo) + contar_formas(n - 2, memo) + contar_formas(n - 3, memo)
        return memo[n]
```

Memoization:

La función contar_formas_memoization(n, memo={}) utiliza el enfoque de memoization, que es una técnica de optimización que almacena los resultados de cálculos previos para evitar recalcularlos.

- En lugar de calcular recursivamente las formas de subir n escalones cada vez que se llama a la función, esta versión verifica si n está presente en el diccionario memo.
- Si está presente, devuelve el valor almacenado correspondiente.

- De lo contrario, realiza los cálculos recursivos como en el enfoque de recursividad plana, almacenando los resultados en memo y devolviendo el valor calculado.

Algoritmo usando memoization

```python
# Código usando memorización
def contar_formas_memoization(n, memo={}):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    elif n == 2:
        return 2
    elif n == 3:
        return 4
    elif n in memo:
        return memo[n]
    else:
        memo[n] = contar_formas_memoization(n-1, memo) + contar_formas_memoization(n-2, memo) + contar_formas_memoization(n-3, memo)
        return memo[n]
```

Comparación y Ejemplos:

Ambas funciones proporcionan resultados idénticos, pero la versión de memoization es más eficiente en términos de tiempo de ejecución, ya que evita recalcular valores previamente calculados. En el código de ejemplo, se prueba la función de memoization con una serie de casos de prueba para calcular el número de formas de subir una escalera de 2 a 6 escalones.

Resultado del algoritmo de memoization

```
Ejemplo de uso:
        Utilizando el método de memoization para contar las formas de subir n escalones

Formas de subir 2 escalones (Memoization): 2
Formas de subir 3 escalones (Memoization): 4
Formas de subir 4 escalones (Memoization): 7
Formas de subir 5 escalones (Memoization): 13
Formas de subir 6 escalones (Memoization): 24
```

Conclusión:

El uso de técnicas como la recursión y la memoization es fundamental en la programación para resolver problemas complejos de manera eficiente. La memoization, en particular, es una técnica valiosa para optimizar algoritmos recursivos al almacenar resultados previamente calculados. En este caso, ambos enfoques proporcionan la solución deseada, pero la versión de memoization es preferible en términos de rendimiento en escenarios donde se requiere calcular el resultado para un conjunto grande de valores de entrada.