Instituto Politécnico Nacional
Escuela superior de Cómputo
Algoritmos Bioinspirados
Práctica No. 3: Introducción a la Programación Dinámica
Realizada por: Calva Guzmán Alan Alexis
Profesor Jorge Luis Rosas Trigueros
Realizada: 20/03/2025
Fecha de entrega: 24/03/2025

Marco teórico:

La programación dinámica es un método poderoso para resolver problemas complejos dividiéndolos en partes más pequeñas y manejables. Se basa en la idea de "divide y vencerás", donde un problema grande se descompone en subproblemas más simples. La diferencia clave es que en la programación dinámica, estos subproblemas a menudo se repiten, por lo que guardar sus soluciones permite ahorrar tiempo de cálculo. Esto significa que aunque usamos más memoria para almacenar resultados intermedios, ganamos mucha eficiencia en velocidad, ya que evitamos recalcular las mismas cosas una y otra vez.

Este enfoque es especialmente útil para problemas donde las decisiones óptimas en cada paso conducen a la solución global óptima. Por ejemplo, al elegir qué objetos llevar en una mochila para maximizar su valor sin exceder su capacidad, cada decisión de incluir o no un objeto afecta las posibilidades futuras. La programación dinámica nos permite explorar sistemáticamente todas estas posibilidades sin repetir trabajo innecesario.

Existen dos formas principales de implementar la programación dinámica. La primera es la memorización (o enfoque "top-down"), que funciona resolviendo el problema de manera recursiva y guardando cada resultado nuevo en una estructura de datos para usarlo más tarde si se necesita. La segunda es la tabulación (o enfoque "bottom-up"), que es la que usamos en esta práctica. Este método es más directo: construimos una tabla desde los casos más simples hasta llegar al problema completo, llenando cada celda según una relación clara entre los subproblemas.

El método de tabulación es particularmente interesante porque sigue un orden lógico y sistemático. Comenzamos resolviendo los casos más pequeños y triviales, luego usamos esos resultados para resolver problemas un poco más grandes, y así sucesivamente hasta llegar a la solución que buscamos originalmente. Para hacer esto, necesitamos identificar claramente cómo se relacionan los subproblemas entre sí, lo que se expresa mediante lo que llamamos una "ecuación de recurrencia". Esta ecuación no es más que una regla que nos dice cómo calcular la solución de un problema usando las soluciones de problemas más pequeños que ya hemos resuelto.

Para implementar correctamente este método, es crucial prestar atención a cómo inicializamos la tabla y en qué orden la llenamos. Un error común es intentar acceder a resultados que aún no hemos calculado, lo que lleva a respuestas incorrectas. Por eso, el enfoque "bottom-up" es tan valioso: al proceder sistemáticamente desde los casos más pequeños, nos aseguramos de que siempre tendremos los datos necesarios cuando los necesitemos.

Material y equipo:

Fué utilizado como auxiliar en la escritura de código el programa Codium, fueron generados entornos de python con Anaconda, utilizando la versión 3.9 de python; todo esto corriendo en Ubuntu 24.04.2 en un equipo de escritorio con 8 Gb de memoria RAM y un procesador Ryzen de 64 bits como características principales relevantes.

No fueron requeridas librerías especiales de Python.

Desarrollo de la práctica:

El desarrollo requirió un análisis profundo del problema, y sin embargo para estos ya se contaba con la ecuación de recurrencia a priori, lo que simplificó significativamente el trabajo. No obstante, este análisi culminó con un código muy corto y con las soluciones esperadas de los problemas.

No obstante, hay una consideración importante en esta solución. Estos códigos sólo hallan la mejor solución para los parámetros dados al final de la tabla, es decir, si existe más de una solución óptima, esto no es señalado. Si bien esto se puede solucionar facilmente recorriendo la tabla al final de la ejecución y buscando soluciones distintas, esto ya es un problema diferente, por lo que decidí no hacer esta parte, asumiendo que es algo que se tiene que tomar como contraste para las futuras prácticas con algoritmos genéticos. Aún así parecía algo digno de recalcar.

Implementar una solucón a travéz de una tabla es algo bastante abstracto y muy curioso de implementar, pero así se hizo. Utilizando matrices (o arreglos) de dos dimensiones se almacenaron los valores y se exploraron los valores repetidos con las respectivas ecuaciones de recurrencia para cada problema. Ciertamente no sé si haya mucho que explicar sobre esta parte.

A continuación se ve una ejecución del programa, con los mismos valroes que se obtuvieron analíticamente para cada caso

```
Mínimo monedas: 2
(base) alan@Desk:~/Documents/portfolio/Algorigmos Bioinspirados$ python Practica3.py
Valor máximo mochila (ejemplo de la tarea): 13
Mínimo monedas den 1, 3, 5, M=6: 2
Mínimo monedas den 1, 2, 5, M=7: 2
```

Figura 1: Resultado de la ejecución del programa con los ejemplos previamente vistos y resueltos analíticamente

```python
#parametros

if __name__ == "__main__":
    weights = [1,2,3,4,5]
    values = [2,4,4,5,7]
    capacity = 9
    print(f"Valor máximo mochila (ejemplo de la tarea): {knapsack(weights, values, capacity)}")


    coins = [1, 3, 5]
    amount = 6
    print(f"Mínimo monedas den 1, 3, 5, N=6: {coin_change(coins, amount)}")

    coins = [1, 2, 5]
    amount = 7
    print(f"Mínimo monedas den 1, 2, 5, N=7: {coin_change(coins, amount)}")
```

Figura 2: Parámetros de cada instancia vista en clase para las salidas anteriores

Conclusiones y recomendaciones:

En la práctica, la programación dinámica puede parecer complicada al principio, especialmente al tratar de identificar los subproblemas correctos o al derivar la ecuación de recurrencia adecuada. Sin embargo, la demanda de este tipo de problemas por un entendimiento y dominio del problema nos lleva a concluir en una solución muy adecuada para el caso. Esto combina un muy buen manejo de la memoria y del procesamiento a la hora de ejecutar un código. Aunque me parece algo complicado escalar este tipo de soluciones si de pronto se tienen que enfrentar una variedad diversa de problemas con el mismo algoritmo, pues no parece factible hacer un análisis tan profundo y personalizar una ecuación de recurrencia en este caso.