Instituto Politécnico Nacional
Escuela Superior de Cómputo
Materia: GENETIC ALGORITHMS
Lab Session 2. Introduction to Dynamic Programming
Alumno: Torres Abonce Luis Miguel
Profesor: Jorge Luis Rosas Trigueros
Fecha de Realización: 16 Septiembre 2024
Fecha de Entrega: 17 Septiembre 2024

Marco Teórico.

La Programación Dinámica es una técnica de optimización y solución de problemas que se utiliza ampliamente en algoritmos y estructuras de datos. Esta técnica se basa en el principio de descomponer un problema complejo en subproblemas más simples y resolver cada uno de ellos una sola vez, almacenando sus soluciones para evitar cálculos redundantes en el futuro. La programación dinámica es especialmente efectiva en problemas que tienen subestructuras óptimas y subproblemas superpuestos, lo que significa que el problema grande puede ser resuelto a partir de soluciones óptimas de sus subproblemas.

Características de la Programación Dinámica:

1. Subestructura Óptima: Un problema exhibe subestructura si la solución óptima del problema se puede obtener a partir de las soluciones óptimas de sus subproblemas. Esto permite construir la solución global de manera incremental.
2. Subproblemas Superpuestos: Los subproblemas superpuestos se presentan cuando un problema puede ser dividido en subproblemas más pequeños que son resueltos varias veces. La programación dinámica almacena las soluciones de estos subproblemas en estructuras de datos como matrices o tablas para evitar la recalculación.
3. Memorización y Tabulación: La memorización es un enfoque de "arriba hacia abajo" que guarda los resultados de subproblemas ya calculados en una tabla y los reutiliza cuando se necesita resolver el mismo subproblema. La tabulación, por otro lado, es un enfoque de "abajo hacia arriba" que construye soluciones de subproblemas más pequeños para llegar a la solución del problema principal.
4. Complejidad Temporal y Espacial: Los algoritmos de programación dinámica suelen tener una complejidad temporal y espacial que depende del tamaño de la tabla utilizada. Aunque pueden consumir más espacio en memoria, son significativamente más rápidos que las soluciones recursivas ingenuas.

Aplicaciones de la Programación Dinámica

La programación dinámica se aplica en una variedad de problemas en áreas como la optimización, la teoría de juegos, la bioinformática, la economía, la ingeniería, entre otros. Algunos de los problemas más conocidos que se resuelven mediante programación dinámica son:

- Problemas de Rutas Óptimas: Como el problema del camino más corto o el problema del viajante de comercio.
- Problemas de Secuencias: Como el problema de la subsecuencia común más larga.
- Problemas de Particionamiento: Como el problema de particionar un conjunto de números en dos subconjuntos con sumas iguales.
- Problemas de Optimización de Recursos: Como la asignación de trabajos o tareas para maximizar la ganancia o minimizar el tiempo.

Material y Equipo

- **Hardware:** Computadora con capacidad para ejecutar Python.
- **Software:**
  - Python 3.x
  - Google Colab.

Desarrollo de la practica

En esta práctica se desarrollan soluciones para dos problemas clásicos utilizando la técnica de Programación Dinámica: el problema de la Mochila 0/1 y el problema del Cambio de Monedas.

1. Problema de la Mochila 0/1

El problema de la Mochila 0/1 consiste en determinar la combinación de objetos que maximiza el valor total que se puede llevar en una mochila con una capacidad limitada. Cada objeto tiene un peso y un valor. El objetivo es maximizar el valor total sin exceder la capacidad de la mochila. Se llama 0/1 porque cada objeto solo puede ser incluido o excluido, sin fraccionamiento.

La programación dinámica es útil aquí porque permite descomponer el problema principal en subproblemas más pequeños, donde cada subproblema representa una decisión de incluir o no un objeto en la mochila.

- Subestructura Óptima: Cada subproblema se resuelve determinando la mejor decisión posible (incluir o excluir un objeto) basándose en las soluciones a subproblemas previos.
- Subproblemas Superpuestos: Muchos subproblemas se resuelven varias veces con las mismas combinaciones de capacidad y elementos restantes, lo cual se gestiona eficientemente almacenando las soluciones intermedias en una tabla (dp).

Proceso de Resolución:

1. Se crea una tabla dp donde dp[i][w] representa el valor máximo que se puede obtener utilizando los primeros i objetos y una capacidad w.
2. Se inicializa la tabla con ceros, pues sin capacidad o sin objetos, el valor máximo es 0.
3. Se rellena la tabla iterativamente, evaluando para cada objeto si es mejor incluirlo o no, basándose en el valor acumulado y la capacidad restante.
4. Al final, la solución óptima se encuentra en dp[n][capacity], donde n es el número total de objetos.

Las principales dificultades encontradas fueron el manejo de la tabla dp, especialmente en problemas de gran escala debido al uso elevado de memoria y tiempo de procesamiento, así como la comprensión de cuándo incluir o excluir un objeto y cómo estas decisiones impactan en el valor total. Estas dificultades se resolvieron mediante un uso más eficiente de la tabla, y la incorporación de comentarios explicativos en el código para clarificar la lógica de inclusión y exclusión de objetos.

2. Problema del Cambio de Monedas

El problema del Cambio de Monedas consiste en determinar el número mínimo de monedas necesarias para alcanzar una cantidad específica utilizando denominaciones disponibles. Es un problema de optimización que busca la combinación más eficiente de monedas.

La programación dinámica se aplica dividiendo el problema en subproblemas que buscan la cantidad mínima de monedas para montos más pequeños y almacenando estas soluciones para construir la solución final.

- Subestructura Óptima: Cada subproblema se resuelve encontrando la combinación óptima de monedas que forma un monto particular, basándose en las soluciones de los subproblemas menores.
- Subproblemas Superpuestos: Los mismos subproblemas (mismos montos con las mismas monedas) se resuelven repetidamente. Al almacenar los resultados en una tabla, se evita recalcular soluciones, mejorando la eficiencia.

Proceso de Resolución:

1. Se crea una lista dp donde dp[i] representa el número mínimo de monedas necesarias para formar el monto i.
2. Se inicializa dp[0] en 0 (ya que no se necesitan monedas para formar el monto 0) y los demás valores en infinito, indicando que inicialmente no hay una combinación válida.
3. Se actualiza la lista iterativamente para cada moneda y cada monto, minimizando el número de monedas necesarias para cada cantidad.
4. El resultado final se encuentra en dp[amount], que representa el número mínimo de monedas para formar el monto deseado.

Los desafíos incluyeron la correcta inicialización de la tabla dp con valores infinitos para gestionar montos inalcanzables, la optimización en la minimización del número de monedas necesarias, y el manejo de casos en los que no era posible formar el monto con las monedas disponibles. Estas dificultades se superaron estableciendo condiciones claras para la actualización de dp, optimizando el cálculo del mínimo de monedas y añadiendo verificaciones finales para asegurar que los resultados reflejen si un cambio es posible o no.

Diagramas, gráficas y pantallas.

Figura 1. Se muestra los 2 ejemplos de ejecución del problema de la mochila 0/1.

```
# Ejemplos de ejecución
pesos = [1, 2, 3, 4]
valores = [10, 20, 30, 40]
capacidad = 5
print("Valor máximo en la mochila:", mochila(pesos, valores, capacidad))

pesos = [2, 3, 4]
valores = [3, 4, 5]
capacidad = 5
print("Valor máximo en la mochila:", mochila(pesos, valores, capacidad))
```

Figura 1.

Figura 2. Se muestra la pantalla de ejecución de los ejemplos del problema de la mochila 0/1.

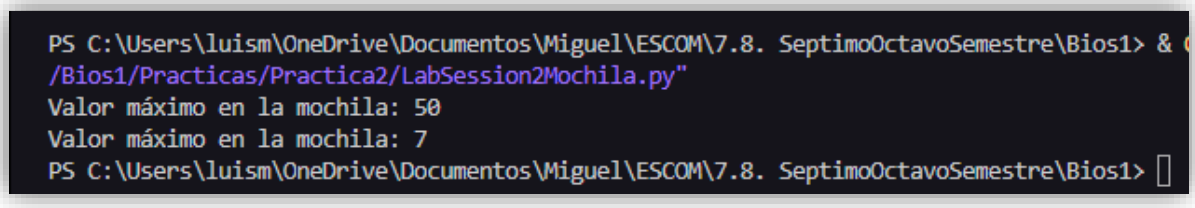

Figura 2.

Figura 3. Se muestra los 2 ejemplos de ejecución del problema de cambio de monedas.

```
# Ejemplos de ejecución
monedas = [1, 2, 5]
cantidad = 11
print("Número mínimo de monedas:", cambio_monedas(monedas, cantidad))

monedas = [2, 5, 10]
cantidad = 7
print("Número mínimo de monedas:", cambio_monedas(monedas, cantidad))
```

Figura 3.

Figura 4. Se muestra la pantalla de ejecución de los ejemplos del problema de cambio de monedas.

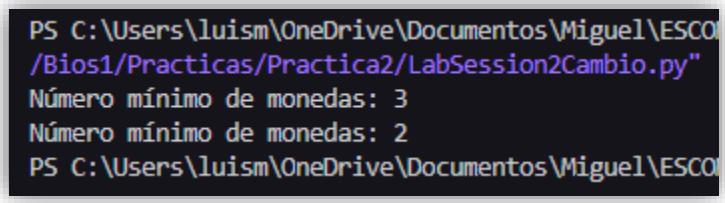

Figura 4.

Conclusiones

En esta práctica, se resolvieron los problemas de la Mochila 0/1 y el Cambio de Monedas utilizando programación dinámica, demostrando cómo esta técnica optimiza la resolución de problemas complejos al descomponerlos en subproblemas más simples y reutilizar sus soluciones. Ambos problemas ilustraron la eficacia de almacenar resultados intermedios para evitar cálculos repetitivos, logrando así soluciones óptimas de manera eficiente. La práctica destaca la importancia de la programación dinámica en la optimización y toma de decisiones en diversos contextos.

Bibliografía

-Knapsack Problem - GeeksforGeeks

- Autor: Equipo de GeeksforGeeks
- Fecha: 2024
- Fuente: https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/

- Coin Change Problem - GeeksforGeeks

- Autor: Equipo de GeeksforGeeks
- Fecha: 2024
- Fuente: https://www.geeksforgeeks.org/coin-change-dp-7/

- https://colab.research.google.com/

Anexo:

Código #1 Problema de la mochila 0/1.

```python
def mochila(weights, values, capacidad):
    n = len(weights)
    # Crear una matriz 2D para almacenar el valor máximo en cada capacidad
    dp = [[0 for _ in range(capacidad + 1)] for _ in range(n + 1)]

    # Rellenar la tabla dp
    for i in range(1, n + 1):
        for w in range(1, capacidad + 1):
            if weights[i - 1] <= w:
                # Si el peso del objeto actual es menor o igual a la capacidad actual,
                # elegimos entre incluir o no incluir el objeto
                dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w])
            else:
                # Si el peso del objeto es mayor que la capacidad actual, no lo incluimos
                dp[i][w] = dp[i - 1][w]

    return dp[n][capacidad]

# Ejemplos de ejecución
pesos = [1, 2, 3, 4]
valores = [10, 20, 30, 40]
capacidad = 5
print("Valor máximo en la mochila:", mochila(pesos, valores, capacidad))

pesos = [2, 3, 4]
valores = [3, 4, 5]
capacidad = 5
print("Valor máximo en la mochila:", mochila(pesos, valores, capacidad))
```

Código #2 Problema del cambio de monedas.

```python
def cambio_monedas(monedas, cantidad):
    # Crear una lista para almacenar el número mínimo de monedas para cada cantidad
    dp = [float('inf')] * (cantidad + 1)
    dp[0] = 0  # Se necesitan 0 monedas para la cantidad 0

    # Rellenar la tabla dp
    for i in range(1, cantidad + 1):
        for moneda in monedas:
            if moneda <= i:
                dp[i] = min(dp[i], dp[i - moneda] + 1)

    # Si la cantidad no puede alcanzarse con las monedas disponibles, devolver -1
    return dp[cantidad] if dp[cantidad] != float('inf') else -1

# Ejemplos de ejecución
monedas = [1, 2, 5]
cantidad = 11
print("Número mínimo de monedas:", cambio_monedas(monedas, cantidad))

monedas = [2, 5, 10]
cantidad = 7
print("Número mínimo de monedas:", cambio_monedas(monedas, cantidad))
```