Instituto Politécnico Nacional
Escuela Superior de Cómputo
Materia: GENETIC ALGORITHMS
Lab Session 3. Greedy Algorithm
Alumno: Torres Abonce Luis Miguel
Profesor: Jorge Luis Rosas Trigueros
Fecha de Realización: 24 Septiembre 2024
Fecha de Entrega: 24 Septiembre 2024

Marco Teórico.

El Problema del Cambio de Moneda (Coin Change Problem) es un problema clásico de optimización combinatoria en el que se busca determinar la menor cantidad de monedas necesarias para alcanzar un valor específico utilizando un conjunto de denominaciones de monedas disponibles. Este problema es de gran relevancia en la teoría de algoritmos y en la práctica, ya que tiene aplicaciones en sistemas financieros y en la planificación de recursos.
El problema puede ser formulado de la siguiente manera: dado un conjunto de denominaciones $D=\{d1,d2,\ldots,dn\}$ y una cantidad objetivo C, se debe encontrar el número mínimo de monedas de las denominaciones disponibles que sumen exactamente C. Este problema se puede resolver mediante diferentes enfoques, incluyendo algoritmos voraces (greedy), programación dinámica o algoritmos de retroceso (backtracking).
Una estrategia común para resolver el CMP es el uso de una heurística voraz. En este enfoque, se selecciona en cada paso la moneda de mayor denominación posible que no exceda la cantidad restante. Este método puede producir soluciones óptimas en ciertos casos, como cuando las denominaciones de las monedas siguen una estructura bien definida (por ejemplo, las monedas de muchos sistemas monetarios, como el dólar o el euro, donde las denominaciones están dispuestas para que este método sea efectivo).
Sin embargo, en algunos conjuntos de denominaciones, el enfoque voraz puede no ofrecer una solución óptima, dado que toma decisiones basadas únicamente en la mejor opción local (la moneda de mayor valor) sin considerar las combinaciones globales que podrían llevar a una solución con un menor número de monedas.

El Problema de la Mochila 0/1 (Knapsack Problem, KP) es otro problema fundamental en la teoría de optimización combinatoria. El objetivo de este problema es maximizar el valor total de los objetos que pueden ser incluidos en una mochila, sin exceder su capacidad de peso. Cada objeto tiene un valor vi y un peso wi, y se debe decidir si incluir o no cada objeto en la mochila, sujeto a una restricción de capacidad W. Este problema tiene aplicaciones en campos como la economía, la logística y la teoría de recursos, y es un problema NP-completo, lo que significa que no existe un algoritmo eficiente que garantice siempre encontrar la solución óptima en tiempo polinomial para todas las instancias.
Una de las aproximaciones más sencillas para el problema de la mochila es el algoritmo voraz. Este enfoque consiste en calcular la razón valor/peso para cada objeto y seleccionar primero los objetos con la mayor relación vi/wi , hasta que no se puedan añadir más objetos sin exceder la capacidad de la mochila. Aunque este enfoque voraz puede ser eficiente y fácil de implementar, no siempre garantiza la solución óptima. El problema radica en que seleccionar objetos basándose en la relación valor/peso no siempre maximiza el valor total cuando se consideran todas las combinaciones posibles. La solución óptima puede requerir incluir un objeto con menor relación valor/peso si permite optimizar la combinación total de objetos.

La heurística voraz es una técnica que se utiliza comúnmente en problemas de optimización. Un algoritmo voraz toma decisiones secuenciales basadas en la mejor opción local disponible en cada paso, con la esperanza de que esta estrategia conduzca a una solución globalmente óptima. Este tipo de algoritmo es particularmente útil cuando se busca una solución rápida y aproximada, y es conocido por su simplicidad y eficiencia computacional.
En un algoritmo voraz, las decisiones se toman de acuerdo con un criterio de optimización local (como seleccionar el elemento de mayor valor o menor costo), y una vez tomada una decisión, no se revisa ni se ajusta, lo que lo hace computacionalmente eficiente. Sin embargo, esta estrategia no siempre lleva a una solución óptima, ya que puede ser miope, es decir, optimiza el resultado inmediato sin considerar el impacto de las decisiones en el largo plazo o en la solución global.
Características de los Algoritmos Voraces:

- ❖ Selección local óptima: En cada paso del algoritmo, se elige la mejor opción disponible en ese momento, sin tener en cuenta las decisiones futuras.
- ❖ Irrevocabilidad: Una vez que se toma una decisión, no se revisa ni se cambia en pasos posteriores.
- ❖ Eficiencia: Los algoritmos voraces suelen tener una baja complejidad temporal, lo que los hace adecuados para grandes problemas donde se necesita una solución rápida.

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 se enfoca en seleccionar un subconjunto de objetos, cada uno con un valor y un peso, que maximice el valor total sin exceder una capacidad de peso predeterminada. Este es un problema de optimización combinatoria donde la relación valor/peso es un criterio fundamental.
El enfoque voraz en este problema utiliza la relación valor/peso de los objetos para priorizar la selección. Se eligen primero los objetos con la mayor relación valor/peso hasta que no se puedan agregar más objetos sin exceder la capacidad de la mochila.

- Subestructura Óptima (enfoque greedy): La estrategia voraz selecciona los objetos que proporcionan el mayor valor por unidad de peso. Esto supone que priorizar los objetos más "eficientes" en términos de valor maximizará el valor total de la mochila.
- Decisiones locales versus globales: Al igual que en el CMP, el enfoque voraz toma decisiones locales en cada paso. Selecciona el objeto con la mejor relación valor/peso en cada iteración, pero puede fallar al combinar objetos de manera óptima para maximizar el valor total bajo la restricción de peso.

Proceso de Resolución:

1. Calcular la relación valor/peso para cada objeto.
2. Ordenar los objetos por su relación valor/peso de mayor a menor.
3. Seleccionar los objetos comenzando por el de mayor relación valor/peso, añadiendo tantos como sea posible sin exceder la capacidad de la mochila.
4. Parar cuando no se pueda añadir más objetos debido a la restricción de peso.
5. El resultado final es el conjunto de objetos seleccionados y su valor total.

La principal dificultad del enfoque voraz es que seleccionar siempre la moneda de mayor valor puede llevar a soluciones subóptimas, ya que no considera combinaciones de monedas más pequeñas que podrían resultar en una menor cantidad total de monedas. Este problema se presenta especialmente cuando las denominaciones no están estructuradas de manera que el enfoque greedy siempre funcione, lo que genera soluciones que no son óptimas.

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 un conjunto de denominaciones disponibles. Este problema es un desafío clásico de optimización que busca la combinación más eficiente de monedas que sumen el monto deseado.

El enfoque voraz se aplica seleccionando la moneda de mayor valor que no exceda la cantidad restante en cada paso. Sin embargo, este método no siempre garantiza una solución óptima, ya que puede ignorar combinaciones más eficientes de monedas.

- Subestructura Óptima (enfoque greedy): El algoritmo voraz intenta construir la solución seleccionando localmente la moneda de mayor denominación disponible que sea menor o igual al monto restante. Se basa en la idea de que elegir la moneda más grande minimizará el número de monedas en la solución final.
- Decisiones locales y globales: El enfoque voraz toma decisiones locales en cada paso sin considerar el impacto global de dichas decisiones. Esto significa que la selección de una moneda de alto valor puede impedir la creación de una solución óptima en algunos casos.

Proceso de Resolución:

1. Ordenar las denominaciones de mayor a menor valor.
2. Iterar sobre las denominaciones y, para cada una, seleccionar tantas monedas de ese valor como sea posible sin exceder el monto restante.
3. Reducir el monto restante en función de las monedas seleccionadas y repetir el proceso hasta que no queden monedas o se alcance el monto objetivo.
4. La solución final es la lista de monedas seleccionadas que suman exactamente la cantidad deseada o indican que no es posible obtener el monto con las monedas disponibles.

La dificultad radica en que el enfoque voraz, al priorizar los objetos con la mejor relación valor/peso, puede fallar en encontrar la combinación óptima de objetos. Aunque seleccionar los objetos más "eficientes" en términos de valor por peso parece una buena estrategia, no siempre produce el valor máximo posible, ya que no toma en cuenta cómo la combinación de objetos con menor relación valor/peso puede generar un mayor valor total dentro de la capacidad limitada de la mochila.

Diagramas, gráficas y pantallas.

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

```python
# Ejemplo de donde falla el algoritmo greedy (solución subóptima)
items_fallo = [Item(60, 10), Item(100, 20), Item(120, 30)]  # Ordenado por valor/peso
capacidad_fallo = 50
resultado_fallo, seleccion_fallo = mochila_greedy(capacidad_fallo, items_fallo)
print("Solución Greedy (Fallo):", resultado_fallo)  # Resultado incorrecto: 160, óptimo: 220

# Ejemplo donde el algoritmo greedy tiene éxito (solución óptima)
items_exito = [Item(100, 10), Item(60, 20), Item(120, 30)]
capacidad_exito = 50
resultado_exito, seleccion_exito = mochila_greedy(capacidad_exito, items_exito)
print("Solución Greedy (Éxito):", resultado_exito)  # Resultado correcto esperado: 220
```

Figura 1.

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

```
Solución Greedy (Fallo): 160
Solución Greedy (Éxito): 220
```

Figura 2.

Figura 3. Se muestra los 2 ejemplos de ejecución del problema de cambio de monedas, uno donde falla en encontrar la solución óptima.

```python
# Ejemplo donde el algoritmo greedy falla (solución subóptima)
monedas_fallo = [25, 10, 5, 1]
cantidad_fallo = 30  # Greedy seleccionará 25 + 5, pero lo óptimo sería 10 + 10 + 10
resultado_fallo, num_monedas_fallo = cambio_greedy(monedas_fallo, cantidad_fallo)
print("Solución Greedy (Fallo):", resultado_fallo, "Número de monedas:", num_monedas_fallo)  # El resultado óptimo sería [10, 10, 10], 3 monedas.

# Ejemplo donde el algoritmo greedy tiene éxito (solución óptima)
monedas_exito = [25, 10, 5, 1]
cantidad_exito = 37  # Greedy dará 25 + 10 + 1 + 1
resultado_exito, num_monedas_exito = cambio_greedy(monedas_exito, cantidad_exito)
print("Solución Greedy (Éxito):", resultado_exito, "Número de monedas:", num_monedas_exito)  # El resultado correcto esperado es [25, 10, 1, 1], 4 monedas.
```

Figura 3.

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

```
Solución Greedy (Fallo): [25, 5] Número de monedas: 2
Solución Greedy (Éxito): [25, 10, 1, 1] Número de monedas: 4
```

Figura 4.

Conclusiones

En esta práctica, se utilizaron los algoritmos voraces para resolver los problemas del Cambio de Monedas y la Mochila 0/1, demostrando que, aunque son eficientes, no siempre llegan a las soluciones óptimas. Al modificar las entradas en ambos problemas, se observó cómo el rendimiento del algoritmo varía según la estructura del problema. Los algoritmos voraces son útiles cuando se necesita una solución rápida, pero en casos donde se requiere llegar a la solución óptima, deberíamos de usar otras técnicas de programación.

Bibliografía

Coin Change Problem - GeeksforGeeks

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

Greedy Algorithms - Programiz

- Autor: Equipo de Programiz
- Fecha: 2024
- Fuente: https://www.programiz.com/dsa/greedy-algorithm

Knapsack Problem - GeeksforGeeks

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

Greedy Algorithms in Optimization - Brilliant

- Autor: Equipo de Brilliant
- Fecha: 2023
- Fuente: https://brilliant.org/wiki/greedy-algorithm/

Anexo:

Código #1 Problema de la mochila 0/1.

```python
# Mochila 0/1 con heurística voraz
class Item:
    def __init__(self, valor, peso):
        self.valor = valor
        self.peso = peso
        self.ratio = valor / peso

def mochila_greedy(capacidad, items):
    items.sort(key=lambda x: x.ratio, reverse=True)  # Ordenar por valor/peso
    peso_actual = 0
    valor_total = 0
    seleccion = []

    for item in items:
        if peso_actual + item.peso <= capacidad:
            seleccion.append(item)
            peso_actual += item.peso
            valor_total += item.valor

    return valor_total, seleccion

# Ejemplo de donde falla el algoritmo greedy (solución subóptima)
items_fallo = [Item(60, 10), Item(100, 20), Item(120, 30)]  # Ordenado por valor/peso
capacidad_fallo = 50
resultado_fallo, seleccion_fallo = mochila_greedy(capacidad_fallo, items_fallo)
print("Solución Greedy (Fallo):", resultado_fallo)  # Resultado incorrecto: 160, óptimo:
220

# Ejemplo donde el algoritmo greedy tiene éxito (solución óptima)
items_exito = [Item(100, 10), Item(60, 20), Item(120, 30)]
capacidad_exito = 50
resultado_exito, seleccion_exito = mochila_greedy(capacidad_exito, items_exito)
print("Solución Greedy (Éxito):", resultado_exito)  # Resultado correcto esperado: 220
```

Código #2 Problema del cambio de monedas.

```python
# Heurística greedy para el problema de cambio de monedas (CMP)
def cambio_greedy(monedas, cantidad):
    monedas.sort(reverse=True)  # Ordenar las monedas de mayor a menor valor
    cambio = []
    total_monedas = 0

    for moneda in monedas:
        while cantidad >= moneda:
            cantidad -= moneda
            cambio.append(moneda)
            total_monedas += 1

    if cantidad != 0:
        return None, total_monedas  # Si no es posible dar el cambio exacto
    return cambio, total_monedas

# Ejemplo donde el algoritmo greedy falla (solución subóptima)
monedas_fallo = [25, 10, 5, 1]
cantidad_fallo = 30  # Greedy seleccionará 25 + 5, pero lo óptimo sería 10 + 10 + 10
resultado_fallo, num_monedas_fallo = cambio_greedy(monedas_fallo, cantidad_fallo)
print("Solución Greedy (Fallo):", resultado_fallo, "Número de monedas:", num_monedas_fallo)  # El
resultado óptimo sería [10, 10, 10], 3 monedas.

# Ejemplo donde el algoritmo greedy tiene éxito (solución óptima)
monedas_exito = [25, 10, 5, 1]
cantidad_exito = 37  # Greedy dará 25 + 10 + 1 + 1
resultado_exito, num_monedas_exito = cambio_greedy(monedas_exito, cantidad_exito)
print("Solución Greedy (Éxito):", resultado_exito, "Número de monedas:", num_monedas_exito)  # El
resultado correcto esperado es [25, 10, 1, 1], 4 monedas.
```