UADE

Programación 3

Clase 5

Temas de la clase 4

- Definición Técnica de Diseño Greedy
- Resolución de problemas Greedy
- Análisis general de la complejidad de los problemas Greedy.

Definición Técnica de Diseño Greedy

Técnica de diseño de algoritmos donde se toma la mejor decisión localmente óptima en cada paso con la esperanza de encontrar una solución globalmente óptima.

Propiedades

Optimal Substructure: La solución óptima del problema contiene la solución óptima de sus subproblemas.

Greedy Choice Property: Una solución globalmente óptima se puede obtener mediante decisiones localmente óptimas.

Ejemplo 1: Problema del Cambio de Monedas

Problema del cambio de monedas: Dado un conjunto de denominaciones de monedas y un monto total, el objetivo es encontrar la cantidad mínima de monedas necesarias para hacer ese monto.

Problema del Cambio de Monedas

Problema del cambio de monedas: Dado un conjunto de denominaciones de monedas y un monto total, el objetivo es encontrar la cantidad mínima de monedas necesarias para hacer ese monto.

Supongamos que tenemos las siguientes denominaciones de monedas: {1, 5, 10, 25} y queremos hacer un monto total de 36.

Un algoritmo greedy para este problema seleccionará siempre la moneda de mayor denominación que no exceda el monto restante.

Pasos del Algoritmo

Ordenar las monedas de mayor a menor.

Inicializar el monto restante al monto total.

Mientras el monto restante sea mayor que 0:

- Seleccionar la moneda de mayor denominación que no exceda el monto restante.
- Restar el valor de esa moneda del monto restante.
- Añadir la moneda a la solución.

Conjunto canónico

Un conjunto canónico de monedas es aquel en el que el enfoque greedy siempre produce una solución óptima para cualquier monto. El conjunto {1,5,10,25} es un ejemplo de un conjunto canónico.

Dado que tenemos una moneda de 1 centavo, es posible generar cualquier monto entero positivo. Esto significa que no existen valores que no se puedan formar utilizando este conjunto de monedas.

Pseudocódigo

```
función encontrarMinimoMonedas(monedas, monto)
    ordenar monedas de menor a mayor
    lista resultado = lista vacía

    para i desde el índice más alto de monedas hasta el índice 0 (de mayor a menor)
        mientras monto sea mayor o igual a monedas[i]
            restar monedas[i] a monto
            añadir monedas[i] a resultado
        fin mientras
    fin para

    devolver resultado
fin función
```

Implementación en Java

```
public class CambioMonedas {
    // Función para encontrar la cantidad mínima de monedas
    public static List<Integer> encontrarMinimoMonedas(int[] monedas, int monto) {
        Arrays.sort(monedas);
        List<Integer> resultado = new ArrayList<>();
        for (int i = monedas.length - 1; i >= 0; i--) {
            while (monto >= monedas[i]) {
                monto -= monedas[i];
                resultado.add(monedas[i]);
            }
        }
        return resultado;
    }
    public static void main(String[] args) {
        int[] monedas = {1, 5, 10, 25};
        int monto = 36;
        List<Integer> resultado = encontrarMinimoMonedas(monedas, monto);
        System.out.println("Monedas usadas para hacer " + monto + ": " + resultado);
    }
}
```

Descripción del algoritmo

En cualquier etapa individual, un algoritmo greedy, selecciona la opción que se “localmente óptimal” . Por ejemplo si tenemos monedas $25,$10,$5,$1 y de desea dar un cambio de $63.

El algoritmo ordena de mayor a menor, y selecciona la moneda de mayor valor que no supere los $63, la agrega a la lista y resta el valor de los $63, quedando $38. De nuevo, se seleccionó la moneda mas grande cuyo valor no fuera mayor a $38 y se se añadió a la lista y así sucesivamente.

Nota: si las monedas tuvieran valores de $1,$5 y $11 y se requiere dar cambio de $15, el algoritmo greedy selecciona $11 y cuatro de $1 , en vez de 3 de $5.

Prueba de escritorio

63-25=38 es mayor a 25

38-25=13 no es mayor a 25, siguiente moneda $10

13-10= 3 no es mayor a 25, siguiente moneda $ 5, siguiente moneda $1

3-1 = 2 es mayor $1

2-1 = 1 fin

Conclusiones

Complejidad Temporal: O(n log n) debido a la ordenación inicial de las monedas. (selección monedas O(n) entonces complejidad: 0(n) + O(n log n) y por propiedad de algebra de órdenes, tomamos el mayor en este caso O(n log n)

El enfoque greedy no siempre produce una solución óptima para todos los conjuntos de denominaciones de monedas. Por ejemplo, si las denominaciones son {1, 3, 4} y el monto es 6, el algoritmo greedy daría {4, 1, 1} (3 monedas) en lugar de la solución óptima {3, 3} (2 monedas).

El problema del cambio de monedas es un ejemplo para ilustrar los algoritmos greedy, mostrando cómo se pueden utilizar para resolver problemas de manera eficiente en muchos casos, aunque no siempre producen la solución óptima para todos los conjuntos de entrada.

Actividad 1

Dada una lista de monedas con denominaciones convencionales (10,1,5,2,10,10,5,2,5,5,5,5,5,5,10), implementar una función greedy que devuelva o genere una lista de monedas para dar cambio exacto utilizando una lista de monedas disponible para un importe de $33. Devolver una lista nula o lanzar una excepción, si no se puede dar el cambio.

Realizar pseudocódigo e implementación en Java. Indicar la complejidad algorítmica.

Actividad 2: Cambio de Moneda Extranjera con Múltiples Tipos de Comprobantes

Descripción del Problema:

Un sistema de tesorería tiene a disposición una variedad de comprobantes que incluyen monedas, cheques, bonos y otros documentos financieros. Cada comprobante tiene un valor específico. El objetivo es realizar una compra de moneda extranjera minimizando el número de comprobantes utilizados.

Resolver mediante pseudocódigo e implementación java.

Indicar la complejidad algorítmica.

Ejemplo 2: Problema Mochila Fraccional

En la versión fraccional del problema de la mochila (Fractional Knapsack), el objetivo es maximizar el valor total de los objetos seleccionados para incluir en una mochila con una capacidad limitada. Dado que se permite tomar fracciones de objetos, este problema se resuelve de manera óptima utilizando un algoritmo voraz (greedy).

Ejemplo 2: Problema Mochila Fraccional

Método greedy:

Calcular la relación valor/peso: Para cada objeto i , se calcula la relación Vi/Wi donde Vi es el valor del objeto y Wi es su peso.

Ordenar los objetos: Se ordenan los objetos en orden descendente según su relación valor/peso.

Selección: Se seleccionan los objetos en el orden determinado, llenando la mochila con tanto peso como sea posible de cada objeto. Si la mochila no puede contener el objeto completo, se toma la fracción del objeto que llena la capacidad restante.

Ejemplo 2: Problema Mochila Fraccional

Este método tiene una complejidad temporal de O(n log n) debido al paso de ordenamiento, seguido de un paso lineal O(n) para seleccionar los objetos, lo que lo hace eficiente para esta variante del problema.

Ejemplo 2: pseudocódigo

Entrada:

- Un conjunto de n objetos, cada uno con un valor v[i] y un peso w[i].
- Capacidad total de la mochila W.

Salida:

- Valor máximo que se puede obtener en la mochila.

Ejemplo 2: pseudocódigo

Procedimiento FractionalKnapsack(n, v, w, W)

1. Para i de 1 a n
    a. Calcular la relación valor/peso para cada objeto: ratio[i] = v[i] / w[i]
2. Ordenar los objetos en orden descendente según ratio[i]
3. Inicializar maxValue = 0
4. Para i de 1 a n
    a. Si W == 0, terminar
    b. Si w[i] <= W
        i. maxValue = maxValue + v[i]
        ii. W = W - w[i]
    c. Sino
        i. maxValue = maxValue + (v[i] * (W / w[i]))
        ii. W = 0
5. Devolver maxValue

Ejemplo 2: implementación en java

La implementación está en el repo de la materia.

Prueba de escritorio

Objeto 1: Valor = 30, Peso = 10 (Relación valor/peso = 3.0)
Objeto 2: Valor = 50, Peso = 20 (Relación valor/peso = 2.5)
Objeto 3: Valor = 60, Peso = 30 (Relación valor/peso = 2.0)
Capacidad de la mochila: 50

Orden descendente (correcto): Objeto 1, Objeto 2, Objeto 3

Tomar Objeto 1 completo (Valor = 30, Peso = 10, Capacidad restante = 40)
Tomar Objeto 2 completo (Valor = 50, Peso = 20, Capacidad restante = 20)
Tomar 2/3 del Objeto 3 (Valor = 40, Peso = 20, Capacidad restante = 0)
Valor total = 30 + 50 + 40 = 120

Actividad 3

Una empresa distribuidora necesita cargar un camión con mercancía que se puede fraccionar. Indicar con una lista los elementos a subir al camión para maximizar el valor total, dado que el camión tiene una capacidad limitada.

Actividad 4

Ingresar a https://www.hackerrank.com/challenges/greedy-florist

Resolver el desafío y luego subir el código fuente a teams y github.