Universidad del Valle de Guatemala
Departamento de Matemática
Licenciatura en Matemática Aplicada

Estudiante: Rudik Roberto Rompich
Correo: rom19857@uvg.edu.gt
Carné: 19857

Análisis y diseño de algoritmos - Catedrático: Tomás Galvéz
26 de mayo de 2023

Parcial 2

Considere el problema de "hacer sencillo": dado un monto, buscamos alcanzar dicho monto con la menor cantidad de monedas posible. Las monedas tienen las siguientes denominaciones: 1,5,10,25. Podemos tomar todas las monedas que necesitemos de cada denominación.

Problema 1. *Demuestre que este problema exhibe subestructura óptima.*

- *a. Este problema no exhibe subestructura óptima porque los subproblemas no son independientes.*
- *b. Debemos decidir qué denominación de moneda excluir de nuestro resultado, produciendo el subproblema de alcanzar el monto con las denominaciones restantes. Si la cantidad usada de monedas de las denominaciones restantes no es la óptima, habrá otra cantidad menor de monedas de esas denominaciones que podríamos incluir en nuestra solución para hacerla mejor.*
- *c. Debemos decidir qué denominación de moneda incluir en nuestro resultado. El subproblema que queda es cómo alcanzar el monto que falta con la menor cantidad de monedas posible (considerando las denominaciones originales). Si el monto que falta se alcanza con una cantidad de monedas subóptima, habrá una menor cantidad de monedas con las que se podría alcanzar ese monto faltante, y por tanto habrá una menor cantidad total de monedas con la que podríamos alcanzar el monto original.*
- *d. Este problema no presenta subestructura óptima porque las denominaciones no son las adecuadas para acumular cualquier monto.*

Solución. Para responder esta pregunta, primero debemos recordar el procedimiento para demostrar la subestructura óptima:

1. Identificar las decisiones.
2. Identificar los subproblemas.

3. Demostrar la optimalidad.

En primer lugar, definitivamente los incisos a. y d. son incorrectos, ya que a. no es cierto, ya que el problema se podría modelar con subestructura óptima y los problemas si son dependientes; para d. no es cierto, ya que sí se puede modelar cualquier monto, ya que tenemos la denominación 1, aunque las demás denominaciones son múltiples de 5. El inciso b. no tiene ningún sentido ¿por qué excluir una moneda? Cuando en la redacción del problema nos dicen que tenemos monedas ilimitadas, este inciso solo tendría sentido si hubiera una restricción en la cantidad de monedas por cada vez que seleccionados. La respuesta más lógica sería la c. ya que considera todas los parámetros que nos da el problema y además sigue los 3 pasos de la subestructura óptima. □

Problema 2. *Demuestre que este problema presenta subproblemas traslapados.*

- *a. Si el monto que queremos alcanzar es m, la solución a un subproblema acumulará un cierto monto $n < m$ que se sumará a la solución de un problema más general. Como las soluciones del problema general y del subproblema se combinan, estos problemas se traslapan. El criterio aplica en todos los niveles de la jerarquía de subproblemas.*
- *b. El problema no presenta subproblemas traslapados.*
- *c. Si el monto que queremos alcanzar es m, algunos montos $n < m$ podrán ser alcanzados con diferentes combinaciones de monedas. El resultado es un mismo monto restante $(m - n)$, que necesitaremos alcanzar con la menor cantidad de monedas posible sin preocuparnos por la combinación de monedas con la que hayamos alcanzado n.*
- *d. Los subproblemas no son independientes. Por lo tanto, son traslapados.*

Solución. Para responder esta pregunta, debemos considerar los subproblemas traslapados, en clase estudiamos dos tipos de métodos para resolver este tipo de problemas: top-down y bottom-up. La idea, es hacer más eficiente un algoritmo para no hacer problemas repetidos, en este caso, definitivamente no consideramos el inciso b., ya que el problema sí presenta problemas trasladapos, el incico d. queda descartado porque no es una demostración. Ahora bien, nos quedan los incisos a. y c. El inciso c. no se adecúa ni con top-down, ni bottom-up, además es raro que se mencione el $(m - n)$, ya que entonces estaríamos generando otro problema extra. Por lo tanto, me inclinaría por inciso a. que es una solución que pienso que sí se podría adaptar al top-down y bottom-up. □

Problema 3. *Describa qué estructura de datos usaría y cómo la usaría para resolver el problema de hacer sencillo con un acercamiento bottom-up.*

- *a. Un árbol, cuya raíz representaría el problema de optimizar el número de monedas para el monto original. Para un nodo, los hijos serían engendrados al probar cada denominación sobre el monto correspondiente al nodo, y representarían los subproblemas de optimizar la cantidad de monedas para el monto restante correspondiente al hijo.*
- *b. Un grafo, cuyo nodo inicial representaría el problema de optimizar el número de monedas para el monto original, y cuyos nodos adyacentes representarían los subproblemas de optimizar el número de monedas para los montos restantes resultantes*

de probar cada denominación. Si, sobre un nodo a, probar una denominación engendra un subproblema cuyo monto ya ha sido considerado en otro nodo b, se conecta a a b con la arista de la denominación probada en lugar de engendrar un hijo nuevo de a.

- *c. Si el monto requerido es m, un arreglo A con m + 1 posiciones. Guardaremos las cantidades mínimas de monedas que permiten alcanzar cada monto desde 0 hasta m, comenzando con $A[0] = 0$. En la posición $A[i]$ se almacena el más pequeño entre los resultados de $1 + A[i - d]$ para cada denominación $d \leq i$.*

- *d. Este acercamiento es de programación dinámica y este problema no se puede resolver con programación dinámica por no presentar las propiedades necesarias.*

Solución. Para considerar una solución para bottom-up, primero debemos definir que esta forma de resolver problemas traslapados se basa en ordenar por dificultad el nivel de los subproblemas, primero resolvemos los subproblemas triviales y con esos resultados vamos resolviendo los problemas más difíciles y generales. Ahora bien, desde el principio descartamos el inciso d. ya que es una oración contradictoria ¿programación dinámica pero no se puede resolver con programación dinámica? El inciso a. lo descartaría también ya que no hay indicios de que se esté usando bottom-up. Ahora bien, quedan los incisos b. y c. Descartaría la c, ya que ese sería más bien un top-down ya que tenemos una tabla que guarda los valores. Entonces, la respuesta es la b. ya que exhibe la forma de bottom-up al resolver los problemas triviales y luego irse a buscar los problemas generales. □

Problema 4. *El problema de hacer sencillo posee la greedy choice property. Explique la diferencia principal que habría entre un algoritmo greedy y uno de programación dinámica para resolver este problema, de modo que quede claro y se justifique cuál algoritmo sería mejor.*

- *a. El algoritmo greedy decidiría siempre incluir la moneda de mayor denominación en el resultado, siempre que incluirla no supere el monto requerido. El algoritmo de programación dinámica probaría todas las combinaciones de monedas posibles para alcanzar el monto requerido. Luego compararía las cantidades de monedas usadas en todos los casos para elegir la menor. Greedy se ahorra probar todas las combinaciones al elegir una directamente, por lo que es más eficiente.*

- *b. El algoritmo greedy decidiría siempre incluir la moneda de mayor denominación en el resultado, siempre que incluirla no supere el monto requerido. El algoritmo de programación dinámica probaría una moneda de cada denominación y, para el monto restante por cubrir, probaría nuevamente cada denominación, y así sucesivamente. Luego compararía las cantidades de monedas usadas en todos los casos para elegir la menor. Sin embargo, el algoritmo de programación dinámica es mejor porque garantiza una solución óptima, mientras que el algoritmo greedy no.*

- *c. No se puede construir ninguno de estos algoritmos porque este problema no presenta las propiedades necesarias. Se tendría que usar Divide and Conquer.*

- *d. El algoritmo greedy decidiría siempre incluir la moneda de mayor denominación en el resultado, siempre que incluirla no supere el monto requerido. El algoritmo de programación dinámica probará todas las combinaciones de monedas posibles para alcanzar el monto requerido, pero que sea mejor o peor al greedy dependerá de si el acercamiento es bottom-up o top-down.*

Solución. De entrada, tenemos que la greedy-choice-property, nos quiere decir que posee una estructura greedy y que de subestructura óptima. Sin embargo, comparado a la programación dinámica, la greedy-choice-property es algo más heurístico (no siempre es la solución óptima) y no tan riguroso como los métodos de programación dinámica (que incluso nos garantiza la solución óptima). Por lo tanto: c. queda descartado desde el principio, evidentemente el problema si presenta las propiedades necesarias. El inciso d. queda descartado, que sea mejor o peor, el greedy no depende de los métodos, ya desde el principio no es un método bueno. Entonces queda a. y b.; aunque el inciso b. es más o menos preciso, pero siento que tiene muchas dilaciones. Entonces, pienso que la respuesta es la a. ya que es más concisa: el algoritmo de programación dinámica irá a probar todas las combinaciones hasta encontrar la solución óptima (aunque sea tardado); mientras que el greedy quizás sea muy eficiente, pero su respuesta no será la mejor. □

Problema 5. *Modele el knapsack problem fraccionado como una matroide ponderada (suponga que la unidad mínima de peso son libras).*

- *a. La familia de conjuntos independientes la conforman los diferentes conjuntos de artículos enteros cuyo peso acumulado es aguantado por la bolsa.*
- *b. La familia de conjuntos independientes la conforman las acumulaciones de libras de artículos arbitrarios tales que el peso total no supera la capacidad de la bolsa.*
- *c. La familia de conjuntos independientes la conforman las acumulaciones de libras de artículos cuyo valor total es el máximo para cada cantidad de libras.*
- *d. El knapsack problem fraccionado no puede modelarse como una matroide porque no posee la greedy-choice property.*

Solución. Para modelarlo como una matroide debemos considerar su defínción directa $M = (S, I)$, en donde $S \neq \varnothing$(que tiene una función de $w : \mathbb{R}^+ \to S$), en donde I es una familia de subconjuntos de S (se le conoce como independiente). Además se debe cumplir la herencia y el intercambio,

- Herencia. Sea $A \in I, B \subset A \implies B \in I$.
- Intercambio. Sea $A, B \in I, |B| > |A| \implies \exists x \in B - A \ni A \cup \{x\} \in I$

Entonces, al hacer el análisis del inciso, d. queda descartado. Ahora bien el knapsack problem era el problema del caco, que asaltaba un lugar y tenía que hacer cálculos para robar la mayor cantidad de artículos que costaran más y que fueran de un peso específico y como es fraccionado, se pueden repetir varias veces los artículos. Por lo tanto, el inciso b. y c. quedan descartados porque no se visualizan como conjuntos indepedientes, entonces me inclinaría por el inciso a, ya que son los diferentes conjuntos de artículos enteros. □