PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

ALGORITMOS AVANZADOS
Hoja de Enunciados
Programación Dinámica

Compilado por Andrés Melgar

---

1. Problema: *Coin Row*

Contexto

Considérese el caso de un ladrón que se enfrenta a una fila de casas dispuestas en línea. Cada casa contiene una cantidad determinada de monedas, pero el sistema de seguridad impide que se roben dos casas consecutivas. Si el ladrón roba dos casas contiguas, la alarma se activa. El objetivo del ladrón es maximizar la cantidad total de monedas robadas sin activar el sistema de seguridad.

Este problema representa un caso clásico de optimización con restricciones, y es adecuado para ser resuelto utilizando técnicas de **programación dinámica**, bajo un enfoque *bottom-up*, lo cual permite minimizar la complejidad computacional en comparación con soluciones de fuerza bruta.

Enunciado

Se dispone de un arreglo de enteros no negativos $C = [C_1, C_2, \ldots, C_n]$, donde C_i representa la cantidad de monedas disponibles en la casa i-ésima. Se desea determinar el valor máximo de monedas que pueden ser robadas sin que se seleccionen dos casas adyacentes.

Restricciones:

- No se permite elegir dos elementos consecutivos del arreglo.
- Se puede elegir ninguna, una o varias casas, siempre que no sean adyacentes.
- Se busca maximizar la suma total de las monedas seleccionadas.

Entrada:

Un arreglo de enteros C de tamaño n.

Salida esperada:

Un entero que representa la suma máxima de monedas que se puede obtener sin seleccionar dos casas consecutivas.

Ejemplo

Entrada:

$$C = [5, 1, 2, 10, 6, 2]$$

Posible solución: Seleccionando las casas con 5, 10 y 2 monedas (índices 0, 3 y 5).
Salida:

$$5 + 10 + 2 = 17$$

2. Problema: Subsecuencia creciente más larga (*Longest Increasing Subsequence*)

Contexto

En diversos problemas computacionales, es frecuente enfrentarse a secuencias de datos donde se requiere identificar estructuras internas con propiedades específicas. Una de estas estructuras, de amplio uso en teoría de algoritmos y análisis de secuencias, es la *subsecuencia creciente más larga* (*Longest Increasing Subsequence*, o LIS por sus siglas en inglés). Este problema tiene aplicaciones en análisis de series temporales, bioinformática (por ejemplo, comparación de cadenas de ADN), y optimización de procesos, entre otros.

Enunciado

Dada una secuencia de n números enteros $A = [a_1, a_2, a_3, \ldots, a_n]$, se desea encontrar la longitud de la subsecuencia creciente más larga contenida en A. Una **subsecuencia creciente** es una secuencia $[a_{i_1}, a_{i_2}, \ldots, a_{i_k}]$ tal que $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ y $a_{i_1} < a_{i_2} < \ldots < a_{i_k}$.

Objetivo

Diseñar e implementar un algoritmo que determine la longitud de la subsecuencia creciente más larga de una secuencia de números enteros. Para ello, se deberá aplicar la técnica de **programación dinámica en su enfoque *bottom-up***, almacenando resultados parciales para evitar cálculos redundantes.

Ejemplo

- Entrada: $A = [10, 22, 9, 33, 21, 50, 41, 60]$
- Salida esperada: 5
- Explicación: Una posible subsecuencia creciente es $[10, 22, 33, 50, 60]$, cuya longitud es 5.

3. Problema de la Mochila (*Knapsack Problem*)

Contexto

El problema de la mochila es un problema clásico de optimización combinatoria que se presenta en una amplia variedad de contextos, tales como la selección de inversiones, la planificación de recursos, o la carga de equipaje. En todos estos escenarios, se debe tomar una decisión óptima sobre qué elementos seleccionar entre un conjunto, bajo una restricción de capacidad, con el fin de maximizar un beneficio total.

Enunciado

Se cuenta con una mochila que tiene una capacidad máxima de carga W (un número entero positivo que representa el peso máximo que puede contener). Además, se dispone de n objetos, donde cada objeto i tiene un peso w_i y un valor asociado v_i.

El objetivo es seleccionar un subconjunto de estos objetos para colocarlos en la mochila, de modo que:

- La suma total de los pesos de los objetos seleccionados no exceda la capacidad W.
- La suma total de los valores de los objetos seleccionados sea la máxima posible.

Formalmente, se desea maximizar:

$$\sum_{i=1}^{n} v_i \cdot x_i$$

sujeto a:

$$\sum_{i=1}^{n} w_i \cdot x_i \leq W, \quad x_i \in \{0,1\}, \quad \forall i \in \{1,2,\ldots,n\}$$

donde $x_i = 1$ si el objeto i es incluido en la mochila, y $x_i = 0$ en caso contrario.

Objetivo

Diseñar un algoritmo que determine el valor máximo que se puede obtener al seleccionar un subconjunto de objetos bajo la restricción de peso. Se espera que el algoritmo utilice el enfoque de programación dinámica *bottom-up*, construyendo una tabla de soluciones parciales que permita llegar a la solución óptima del problema.

Ejemplo

Dado $n = 4$ (número de ítems), $W = 5$ (capacidad máxima de la mochila).
Los ítems:

Ítem	Peso (w_i)	Valor (v_i)
1	2	12
2	1	10
3	3	20
4	2	15

El valor máximo se logra seleccionando los ítems 1, 2 y 4, con pesos $2+1+2 = 5$ y valores $12+10+15 = 37$.

4. Problema: *Hangover* (Adaptado de ex1 2012-1 Algoritmia)

Problema adaptado de la URL `http://poj.org/problem?id=1003` ¿Hasta dónde puede hacer que una pila de cartas, cada una de tamaño n, sobresalga fuera de una mesa? Si Ud. posee una sola carta, a lo más la mitad de la carta podrá quedar fuera de la mesa (ver figura 1). (*Asumimos que las cartas deben estár de forma perpendicular a la mesa*).

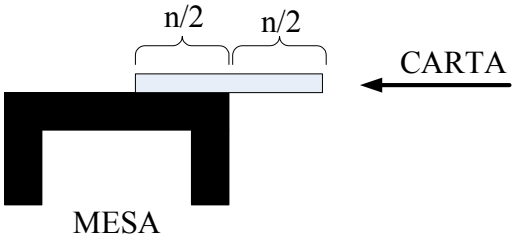

Figura 1: Mesa con una sola carta.

Con dos cartas Ud. puede hacer que la carta de encima sobresalga de la de abajo por la mitad de la longitud de la carta, y la de abajo sobresalga de la mesa por un tercio de la longitud de la carta, haciendo que en total las dos cartas sobresalgan una distancia igual a $1/2 + 1/3 = 5/6$ del tamaño de la carta (ver figura 2).

En general Ud. puede hacer que k cartas sobresalgan de la mesa a una distancia de $1/2 + 1/3 + 1/4 + \ldots + 1/(k+1)$ del tamaño de una carta, en donde la carta de la cima se encuentra separada por una distancia de $1/2$ de la de carta que se encuentra debajo de ella, la segunda carta a partir de la cima se cuentra separada por una distancia de $1/3$ de la carta, la tercera carta a partir de la cima se encuentra separada por una distancia de $1/4$, etc, y la carta de abajo se encuentra a una distancia $1/(k+1)$ (ver figura 3).

Se le pide que elabora un programa en C++ que dado una lista de m números reales ($0.01 \leq m \leq 5.20$) que representa una distancia en donde 1 representa al tamaño n de un carta, retorne para cada número,

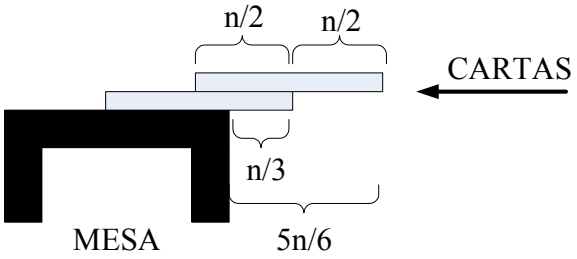

Figura 2: Mesa con dos cartas.

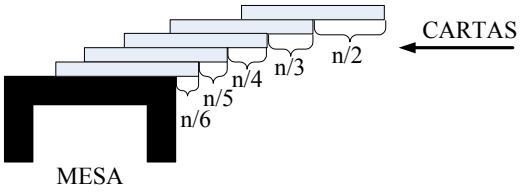

Figura 3: Mesa con varias cartas.

la cantidad de cartas necesarias para alcanzar la distancia solicitada. ***Una restricción del problema es que este debe ser ejecutado rápidamente (i.e. 1000 milisegundos) para una lista grande de m números, además no podrá usar mucha memoria (i.e. 10000K)***

Por ejemplo si la lista de m números fuera

```
1.00
3.71
0.04
5.19
```

el programá debería retornar

```
3 carta(s)
61 carta(s)
1 carta(s)
273 carta(s)
```

respectivamente
