Práctica 6 - Estrategias

Greedy

1. Problema del cambio. El problema del cambio consiste en determinar, dado un número entero C, una combinación de monedas de 1, 5, 10 y 20 centavos que sumen C y que use la mínima cantidad de monedas posible.

a Programar un algoritmo greedy para resolver este problema.

b ¿Este algoritmo resuelve el problema para cualquier denominación de monedas? Justificar.

2. Problema de selección de actividades. Sea $S = \{a_1, a_2, \ldots, a_n\}$ un conjunto de n actividades que necesitan hacer uso de un recurso, el cual solo puede ser usado por a lo sumo una actividad a la vez, por ejemplo: clases que deben dictarse en un mismo aula. Cada actividad a_i tiene una hora de inicio s_i y una hora de finalización f_i, donde $0 \leq s_i < f_i < \infty$. En caso de seleccionarse, la actividad a_i hará uso exclusivo del recurso durante el intervalo de tiempo semi-abierto $[s_i, f_i)$. Escribir un programa que determine la máxima cantidad de actividades que se pueden seleccionar sin que haya superposición entre ellas, utilizando un algoritmo greedy.

Dividir y conquistar

3. Mergesort. Definimos el algoritmo de ordenación Mergesort de la siguiente manera:

- Si la longitud del arreglo es 0 o 1, entonces ya está ordenado.
- Si la longitud del arreglo es mayor que 1, entonces se divide a la mitad y se realiza lo siguiente:
  - **a** Se ordena cada subarreglo recursivamente aplicando Mergesort.
  - **b** Se mezclan los dos subarreglos generando un arreglo ordenado.

Implementar Mergesort y calcular su complejidad con el Teorema Maestro.

4. Escribir un algoritmo basado en la estrategia dividir y conquistar que dado un arreglo busque los dos elementos más grandes del mismo. Hacer un análisis de su complejidad.

Programación dinámica

5. Secuencia de Fibonacci. Esta secuencia tiene los dos primeros valores 0 (cero) y 1 (uno), y los siguientes valores serán la suma de los dos valores precedentes. Por definición, la función para encontrar cualquier número de Fibonacci es:

$$fib(0) = 0$$
$$fib(1) = 1$$
$$fib(n) = fib(n-1) + fib(n-2)$$

Implementar la función de Fibonacci utilizando la técnica de programación dinámica.

6. Problema de la mochila 0-1. Se quiere encontrar la manera óptima de cargar una mochila con objetos. Existen n objetos disponibles. El i-ésimo objeto cuesta v_i pesos, y pesa w_i kilos, donde v_i y w_i son enteros. Se pretende cargar el mayor valor en pesos posibles pero con un máximo W de

kilos. Cada objeto puede ser elegido o no, y no se puede tomar una parte de un objeto ni se puede elegir un objeto más de una vez. Programar un algoritmo dinámico para resolver este problema.

7. Problema de la subsecuencia común más larga (LCS). Dada una secuencia $X = \langle x_1, x_2, \ldots, x_m \rangle$, se se dice que una secuencia $Z = \langle z_1, z_2, \ldots, z_k \rangle$ es una subsecuencia de X si existe una secuencia estrictamente creciente de índices $\langle i_1, i_2, \ldots, i_k \rangle$ de X tal que $\forall j = 1, \ldots, k \cdot x_{i_j} = z_j$. Por ejemplo, $Z = \langle B, C, D, B \rangle$ es una subsecuencia de $X = \langle A, B, C, B, D, A, B \rangle$ con la correspondiente secuencia de índices $\langle 2, 3, 5, 7 \rangle$. Dadas dos secuencias X e Y, se dice que una secuencia Z es una subsecuencia en común de ambas si es una subsecuencia tanto de X como de Y.
El problema de la subsecuencia en común más larga (LCS) consiste en encontrar una subsecuencia en común de máxima longitud entre dos secuencias X e Y.
Sea $X = \langle x_1, x_2, \ldots, x_n \rangle$ una secuencia, se define X_i como el prefijo $\langle x_1, x_2, \ldots, x_i \rangle$ de X. Sean $X = \langle x_1, x_2, \ldots, x_m \rangle$ e $Y = \langle y_1, y_2, \ldots, y_n \rangle$ dos secuencias y $Z = \langle z_1, z_2, \ldots, z_k \rangle$ una LCS de ambas, se puede demostrar que:

- $x_m = y_n \Rightarrow z_k = x_m = y_n$ y Z_{k-1} es una LCS de X_{m-1} e Y_{n-1}.
- $x_m \neq y_n \wedge z_k \neq x_m \Rightarrow Z$ es una LCS de X_{m-1} e Y.
- $x_m \neq y_n \wedge z_k \neq y_n \Rightarrow Z$ es una LCS de X e Y_{n-1}.

Utilizando estos resultados, escribir un algoritmo que encuentre la longitud de una LCS entre dos secuencias X e Y usando la estrategia de programación dinámica.

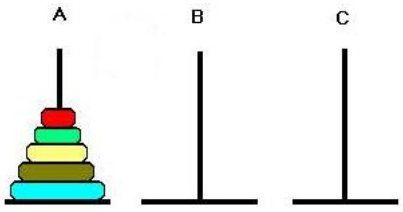
Mezclados

8. Función potencia. La potencia natural de un número se define como:

$$x^0 = 1$$
$$x^{n+1} = x * x^n$$

Es fácil ver que con esta definición calcular la potencia x^y requiere de $O(y)$ productos. Proponer una función para calcular la potencia x^y que requiera $O(\log_2 y)$ productos.

9. Torres de Hanoi. Este juego consiste de tres postes verticales, uno de los cuales tiene apilados n discos de tamaño decreciente, ubicándose el disco más chico en el tope de la pila, y los restantes se encuentran vacíos.



El juego consiste en pasar todos los discos desde el poste ocupado a uno de los otros postes vacíos, siguiendo tres simples reglas:

- Solo se puede mover un disco a la vez.
- Solo se puede desplazar el disco que se encuentra en el tope de la pila.

- Un disco de mayor tamaño no puede estar sobre uno más pequeño.

Escribir un programa para resolver este problema.

10. Considere las siguientes variantes del problema de la mochila 0-1:

- **Problema de la mochila fraccionaria.** En este caso, está permitido tomar fracciones de objetos (incrementando el precio y peso de manera proporcional).
- **Problema de la mochila no acotada.** En este caso, podemos poner en la mochila un mismo objeto tantas veces como queramos (incrementando el precio y peso de manera proporcional).

Escribir programas para resolver estas nuevas variantes.

11. Problema de la multiplicación de matrices. Se desea obtener el producto de n matrices $A_1, A_2, \ldots, A_n$. Dado que el producto de matrices es asociativo, se puede elegir diferentes maneras para obtener el resultado final. Por ejemplo, si $n = 4$ tenemos, entre otras, las siguientes formas:

- $(A_1 \times A_2) \times (A_3 \times A_4)$
- $A_1 \times (A_2 \times (A_3 \times A_4))$

Asumiendo que un algoritmo para multiplicar dos matrices $A_{m\times p}$ y $B_{p\times n}$ realiza $m \times p \times n$ multiplicaciones, podemos observar que la elección de asociatividad que realicemos será determinante para la complejidad de nuestro algoritmo.
Volviendo al ejemplo, supongamos que A_1, A_2, A_3, A_4 son de orden 10×100, 100×5, 5×3 y 3×30 respectivamente. Entonces la cantidad de multiplicaciones para cada una de las formas de asociatividad vistas arriba son:

- $10 \times 100 \times 5 + 5 \times 3 \times 30 + 10 \times 5 \times 30 = 6950$
- $5 \times 3 \times 30 + 100 \times 5 \times 30 + 10 \times 100 \times 30 = 31950$

Como vemos, es importante decidir de qué forma vamos a multiplicar la matrices. Podemos expresar el problema de multiplicación de matrices como sigue:
Dadas n matrices $A_1, A_2, \ldots, A_n$, donde A_i tiene dimensiones $p_{i-1} \times p_i$, debemos dar una expresión con todos los paréntesis de $A_1 \times A_2 \times \ldots A_n$. de forma tal que el número de multiplicaciones se minimice al calcular el producto final. Escribir un programa para resolver este problema.