Práctico 9

Programación Dinámica

NOTA

Para todas las soluciones:

- definir una notación adecuada para representar los elementos del problema.
- encontrar una recurrencia (utilizando la notación definida) que resuelva el problema.
- demostrar que la solución del problema cumple con el principio de optimalidad.

EJERCICIO 1 (R)

Se tiene una secuencia de **N** números positivos y se desea intercalar entre ellos **N-1** pares de paréntesis para luego proceder a calcular las sumas parciales de cada pareja de paréntesis. Existen diferentes formas de intercalar los paréntesis en la secuencia.

Ejemplo:

Sea la secuencia de 4 elementos 4, 1, 2, 3.

Primera forma de intercalar los paréntesis: ((4+1) + (2+3))

Las sumas intermedias son: (4 + 1) = 5, (2 + 3) = 5 y (5 + 5) = 10

La sumatoria de estas es 5 + 5 + 10 = 20

Segunda forma de intercalar los paréntesis: (4 + ((1 + 2) + 3))

Las sumas intermedias son: (1 + 2) = 3, (3 + 3) = 6 y (4 + 6) = 10
La sumatoria de estas es 3 + 6 + 10 = 19

De todas las formas posibles de intercalar los paréntesis, se considera como óptima aquella para la cual, la sumatoria de sumas intermedia sea mínima. En el ejemplo sería el segundo caso.

Resolver utilizando *Programación Dinámica* el problema de determinar el **valor** de la sumatoria óptima de sumas intermedias para una secuencia de largo **N**.

Nota: NO se pide determinar cuál es la distribución de los paréntesis sino solo el valor de la sumatoria de sumas intermedias.

Se pide:

a) plantear una recurrencia que permita calcular el valor mencionado.
b) construir un algoritmo que implemente la recurrencia de la parte a).
c) calcular el orden del algoritmo de la parte b).

EJERCICIO 2 (I)

Una empresa procesadora de harina de maíz, vende su producto en forma mayoritaria y minoritaria. Por este motivo, dispone de distintos tipos de bolsas $b_1,\ldots, b_n$ de capacidades $k_1,\ldots, k_n$ kilos. Cada bolsa b_i tiene un precio asignado p_i y los precios varían dependiendo de la bolsa en una forma que no es lineal con los kilos.

Un cliente realiza compras en forma periódica por una cantidad q de kilos que varía de pedido en pedido siendo q entero.

Resolver utilizando *Programación Dinámica* el problema que determine en base a las capacidades y los precios de las bolsas, cuál es la combinación "ideal" de bolsas para la compra de esos q kilos. Se entiende por combinación "ideal", aquella que suma exactamente q kilos con un precio mínimo. Se sabe además que existe la bolsa de capacidad 1, por lo cuál cualquier cantidad q puede ser formada.

Ejemplo: si las capacidades son 1, 3 y 5, con precios 2, 4 y 9, la combinación ideal para comprar 7 kilos es comprar dos bolsas de 3 y una de 1.

Se pide:
a) definir un algoritmo que utilizando la técnica de *Programación Dinámica*, determine el **costo** de la combinación ideal para un q dado.
b) calcular el orden del tiempo de ejecución del algoritmo definido en a).
c) modificar el algoritmo de la parte a) de forma de obtener, no el costo, sino la combinación ideal para un q dado.

Nota: suponga que las capacidades están ordenadas en forma creciente.

EJERCICIO 3 (I)

Sea $G = (V, E)$ un grafo conexo dirigido, con costos asociados a sus aristas. Se quiere, dados dos vértices u y v, determinar la forma de ir de u a v, pasando por exactamente k aristas con el menor costo posible. El grafo está representado por su matriz de adyacencia (*ADY*) de la siguiente forma:

$ADY[u][v]$ = costo de la arista del vértice u al v.
$ADY[u][v] = +\infty$ si la arista no existe

Se pide:
a) escribir un algoritmo que, utilizando la técnica de *Programación Dinámica*, determine el **costo mínimo** de los caminos de u a v que recorren k aristas.
b) calcular el orden del algoritmo anterior en función de n y k.
c) Escriba un algoritmo que determine el **camino de costo mínimo** que va de u a v recorriendo k aristas si éste existe.
d) calcular el orden del algoritmo anterior en función de n y k.

EJERCICIO 4 (I)

Se desea resolver el problema de la devolución exacta de cambio usando una cantidad mínima de monedas.

Por ejemplo, si se tienen monedas de 1, 2, 5 y 10 pesos y se deben devolver 28 pesos, entonces la cantidad mínima de monedas a devolver es cinco (dos de 10 pesos, una de 5 pesos, una de 2 pesos y una de 1 peso). Es fácil notar que si no se tienen monedas de 1 peso, este problema no siempre se puede resolver (por ejemplo, no se puede dar cambio de 3 pesos usando sólo monedas de 2, 5 y 10 pesos). Una manera de resolver este problema es usando un **algoritmo ávido**.

i) Se tiene un sistema monetario con n tipos de monedas diferentes y un vector $v[1..n]$ ordenado en forma creciente de valor monetario, donde $v[k]$ indica el valor del k-ésimo tipo de moneda.

   En el ejemplo anterior, este vector es: [1, 2, 5, 10].

Se pide escribir un **algoritmo ávido** para resolver el problema de devolver un total de *N* pesos usando la mínima cantidad de monedas.

ii) Ilustrar cómo funciona el algoritmo si se quiere devolver 39 pesos, usando monedas de 1, 2, 5 y 10 pesos.

iii) Lamentablemente el algoritmo ávido no siempre da la cantidad mínima de monedas. Esto depende del sistema monetario que se use.

Por ejemplo, si se tienen monedas de 1, 4 y 5 pesos, hay casos en que el **algoritmo ávido** no da la respuesta adecuada.

Encontrar un valor *N* tal que el algoritmo ávido calcule una cantidad de monedas mayor que el mínimo necesario para alcanzar el valor de *N*.

iv) Una solución apropiada al problema involucra el uso de **Programación Dinámica**. Para esta solución se utiliza una matriz c[1..N][0..N], donde c[i][j] es el número mínimo de monedas necesarias para pagar un valor de j ($0 \le j \le N$), empleando monedas sólo las monedas con valor v[k] ($1 \le k \le i$).

Si se tienen monedas de valor [1, 4, 5, 10, 12], entonces *c*[3][27] indica la cantidad mínima de monedas necesarias para pagar 27 pesos usando sólo monedas de valor 1, 4 o 5 (que se obtiene pagando cinco monedas de 5 pesos y dos monedas de un peso). Se pide dar una recurrencia para hallar *c*[*i*][*j*].

Ayuda: Considerar dos casos. En primer lugar, considerar que sólo se utilizan las monedas de valor *v*[*k*] ($1 \le k \le i$-1), y en segundo lugar que se utiliza alguna moneda de valor *v*[*i*].

v) Se tiene el sistema monetario con los valores [1, 4, 5] y se quiere calcular la cantidad mínima de monedas necesarias para pagar un valor de *j* con $0 \le j \le 8$. Construir la matriz c[1..3][0..8], resultante de usar el algoritmo de **Programación Dinámica**.

EJERCICIO 5 (R)

Una partición de un entero positivo *n*, es una secuencia de enteros positivos $m_1, ..., m_k$ tal que:

$$m_1 + \quad + m_k = n$$

Suponga que se tiene una tabla de pesos asociada a los enteros positivos del 1 al *n*, siendo cada peso un número real. El peso total en una partición $m_1, m_2, ..., m_k$ es la suma de los pesos asociados a cada entero m_j por el valor de m_j o sea

$$\sum_{i=1}^{k} m_i \quad \text{tiene como peso asociado} \quad \sum_{i=1}^{k} (p_{mi} \cdot m_i)$$

Se define como la mejor partición de *n* aquella cuyo peso total es mínimo.

Se pide:

a) escribir un algoritmo utilizando *Programación Dinámica* para calcular el peso total de la mejor partición.
b) estudiar el orden de su algoritmo.

EJERCICIO 6 (R)

Dos strings de caracteres pueden tener muchos substrings en común. Por ejemplo "fotógrafo" y "tomografía" tiene como substrings comunes las letras "o", "t" y "g" entre otras y además "to", "af", "ograf".

Se define como **MLSC** (máximo largo de substring común) a la cantidad de letras del substring común con más letras. Para el ejemplo visto, MLSC=5.

Sean $X = x_1\, x_2 \ldots x_n$ e $Y = y_1\, y_2 \ldots y_m$ dos strings de caracteres.

Se pide:
a) escribir un algoritmo que determine el MLSC de X e Y utilizando *Programación Dinámica*.
b) calcular el orden de su algoritmo.

EJERCICIO 7 (C)

Un sastre trabaja con una máquina de coser que se inspecciona cada año. A comienzos de cada año, el sastre puede decidir entre reparar o cambiar la máquina. El precio de reparación y cambio varía con el número de años de uso de la máquina y está dado por la siguiente tabla:

Años de uso	Reparar	Cambiar
1	70	100
2	30	150
3	90	180
4	-	200

Después de 4 años de uso la máquina no puede ser reparada. Estando a comienzos de año la máquina del sastre ya tiene dos años de uso. Se desea conocer un plan óptimo (de menor costo) de reparación y cambio en los próximos 4 años, sabiendo que a comenzar el quinto regalará la máquina que posea y cerrará la sastrería.

Se pide: representar el problema en términos de grafos e indicar como resolverlo.

EJERCICIO 8 (C)

Se tienen n matrices M_1,..., M_n de índices naturales y elementos de tipo r. Consideramos definidas las siguientes funciones:

filas (M_i) = "número de filas de la matriz M_i"
columnas (M_i) = "número de columnas de la matriz M_i"

Llamaremos $M_{i\,..\,j}$ a la matriz resultado del producto $M_i \cdot \ldots \cdot M_j$, con $i \leq j$.

El costo de calcular $M_{i\,..\,i+1}$ es el número de productos necesarios es decir,

costo ($M_{i..i+1}$) = filas (M_i) · columnas (M_i) · columnas (M_{i+1}),

siendo columnas (M_i) = filas (M_{i+1}).

El costo de calcular $M_{i..j}$ (siendo $j > i+1$) depende de la asociación de las matrices $M_i...M_j$ que se utilice en el cálculo, pues el producto de matrices es asociativo y por lo tanto para calcular $M_{i..j}$ se puede elegir la asociación que involucre menos operaciones.

Por ejemplo, sean las matrices M_1 de dimensión 10×20, M_2 de 20×30 y M_3 de 30×30, resulta:
costo $((M_1 \cdot M_2) \cdot M_3) = 10 \cdot 20 \cdot 30 + 10 \cdot 30 \cdot 30 = 15000$, mientras que
costo $(M_1 \cdot (M_2 \cdot M_3)) = 20 \cdot 30 \cdot 30 + 10 \cdot 20 \cdot 30 = 24000$.

Se desea un algoritmo para calcular la asociación que tiene el mínimo costo de obtener $M_{1..n}$

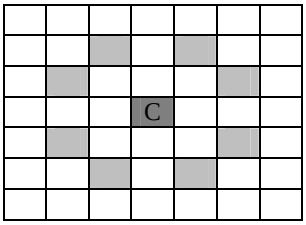
Se pide:

a) elegir una estructura de datos adecuada para almacenar una asociación de matrices.

b) desarrollar un algoritmo, usando ***Programación Dinámica***, que calcule y retorne la asociación que tiene el mínimo costo de obtener $M_{1..n}$. ¿Cuál es el orden de tiempo de ejecución del algoritmo?

EJERCICIO 9 (C)

Sea un tablero de ajedrez de n casillas de lado. Los posibles movimientos del caballo C son los que están marcados en la siguiente figura:



Dado un tablero de ajedrez, una casilla inicial *u* y una casilla final *v*, el objetivo es, en el caso que sea posible ir de la casilla *u* a la *v* con una secuencia de movimientos válidos del caballo, determinar la mínima cantidad de movimientos necesarios para ir de la casilla *u* a la *v*.

Se pide:
a) diseñar un algoritmo basado en *Programación Dinámica* formulando el principio de Optimalidad.
b) dar la fórmula de recurrencia que decida aplicar, explicandola detalladamente.
c) escribir el algoritmo que calcule la recurrencia explicando detalladamente las estructuras de datos utilizadas.

EJERCICIO 10 (C)

Una empresa de alquiler de canoas tiene **N** puestos a lo largo de un río. En cada uno de estos puestos se alquilan canoas que pueden ser devueltas en cualesquiera de los puestos que hay más abajo en el río (la corriente del río es lo suficientemente fuerte como para que nadie haya logrado nunca devolver una canoa más arriba). La empresa tiene una tarifa que indica el costo del alquiler entre cada posible punto de partida y cada posible punto de llegada. Puede ocurrir que el costo de alquiler de un puesto *A* a un puesto *B*, sea mayor que el costo total de una serie de alquileres, caso en el cual puede ser conveniente retornar la canoa en algún punto intermedio *X* y continuar el viaje en una segunda canoa. La empresa no cobra ningún tipo de sobrecargo por este tipo de cambio.

Se pide:

a) utilizando *Programación Dinámica* escribir la recurrencia y el algoritmo que permitan determinar el mínimo costo de un viaje en canoa desde cada posible puesto de partida a cada posible puesto de llegada.

b) calcular la cantidad de comparaciones que realiza su algoritmo.

EJERCICIO 11 (C)

Un grupo de extraterrestres secuestra a un humano y lo lleva a un lugar en donde encuentra *n* pepitas de metales preciosos. Cada pepita *i* tiene su peso $w_i > 0$ y un valor $v_i > 0$. Asombrado ante tanto potencial económico, decide llevarse de regreso a la Tierra la mayor cantidad de pepitas que pueda. Sin embargo, como recipiente sólo tiene una mochila que puede cargar hasta un peso máximo *W* sin romperse. Entonces su objetivo es llenar la mochila de tal forma que maximice el valor de las pepitas incluidas, respetando la limitación de la capacidad de la mochila.

Matemáticamente el problema se puede enunciar de la siguiente manera: se tienen dos vectores *Pesos* = $[w_1, w_2, \ldots w_n]$ y *Valores* = $[v_1, v_2, \ldots v_n]$.
Se pide hallar un vector $X = [x_1, x_2, \ldots x_n]$ (donde x_i tiene valor igual a 1 si se decide colocar la pepita i en la mochila y valor igual a 0 si no se coloca la pepita i en la mochila), tal que se maximice

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

con la restricción de

$$\sum_{i=1}^{n} x_i w_i \leq W$$

donde $v_i > 0$, $w_i > 0$ y $x_i \in \{0,1\}$. Este problema es conocido como el *Problema de la Mochila*.

i) Una manera posible de resolver este problema es utilizando un algoritmo ávido. El valor por unidad de peso de la pepita i se define como el cociente $c_i = [v_i / w_i]$. La idea del algoritmo ávido consiste en ir examinando los objetos en orden decreciente de valor por unidad de peso, es decir, la idea es aprovechar al máximo el valor por cada unidad de peso que ingresa a la mochila.

   Por ejemplo, si se tienen 3 objetos definidos por los siguientes vectores: *Pesos = [6, 5, 5]* y *Valores [8, 5, 5]* entonces el vector C de valores por unidad de peso es: $C = [4/3, 1, 1]$.

   Se pide escribir dicho **algoritmo ávido**.

   Nota: asuma que está definida la función

```
int maximo (double * C, bool *X);
```

   que dado el vector C y el vector X devuelve el índice i del máximo elemento de C que tenga $X[i]$=0, es decir, devuelve el índice del máximo elemento de C que aún no ha sido elegido. Si todos los elementos ya fueron elegidos, entonces devuelve el valor 0.

ii) Ilustrar cómo funciona dicho algoritmo si se tienen las pepitas mencionadas en la parte anterior y una mochila de capacidad $W = 10$ unidades. Concluir que este algoritmo no da siempre la solución óptima, y por lo tanto este camino es incorrecto.

iii) Una solución apropiada al problema involucra el uso de ***Programación Dinámica***. Para esta solución se utiliza una matriz $V[1..n][0..W]$, donde $V[i]][j]$ es el valor máximo de los objetos que se pueden transportar en la mochila si el límite de peso es j ($0 \leq j \leq W$), y si solamente se incluye los objetos numerados del 1 al i ($1 \leq i \leq n$). La solución en este caso se encuentra en el lugar $V[n][W]$. Como condiciones de borde se puede definir $V[0][j]$ igual a 0 cuando $j \geq 0$ y $V[i][j] = -\infty$ para todo i si $j < 0$.

   Se pide dar una recurrencia para calcular los valores $V[i][j]$.

iv) Se tienen cuatro pepitas descritas con los vectores *Pesos = [1,2,5,6]*, *Valores = [1,6,18,22]*, y una mochila con capacidad $W = 8$.

   Se pide calcular el valor máximo que se puede cargar, mostrando la matriz $V[1..4][0..8]$ resultante de usar el algoritmo de ***Programación Dinámica***.

EJERCICIO 12 (C)

Sea $G = (V, A)$ un grafo dirigido con costos c_{ij} asociados a cada arista $<i,j>$. c_{ij} es tal que $c_{ij} > 0$ si $<i,j>$ es una arista, y $c_{ij} = \infty$ si $<i,j>$ no pertenece a A. Sea $|V| = n$, (n>1). El costo de un camino en el grafo es la suma de los costos de sus aristas.

Se pide encontrar un camino que pase por todos los vértices y tenga costo mínimo, este ejercicio es conocido como "El problema del agente viajero".

Escribir una recurrencia que exprese el **costo** de un camino con las características indicadas si es que existe (si no existe, retornar ∞).

Sugerencia: buscar una fórmula de recurrencia para $g(i,S)$ = costo mínimo del camino que parte del vértice i, pasa por todos los vértices de S una sola vez y termina en el vértice 1, siendo 1 el vértice inicial.