Métodos Algorítmicos en Resolución de Problemas II

Grado en Ingeniería Informática

Doble Grado en Ingeniería Informática y Matemáticas

Hoja de ejercicios 1 Curso 2025–2026

Ejercicios de programación dinámica

Ejercicio 1 Tenemos n varillas distintas de longitudes enteras $l_1, \ldots, l_n$ y precios reales $c_1, \ldots, c_n$, respectivamente. Las varillas no se pueden cortar. Se desea soldar algunas de ellas para obtener una varilla de longitud total L (también entera).

Se pide escribir algoritmos de programación dinámica para resolver directamente cada uno de los siguientes problemas:

1. Indicar si es posible o no obtener la varilla deseada soldando algunas de las varillas dadas.
2. Calcular el número total de maneras de obtener la varilla deseada soldando algunas de las varillas dadas, sin que importe el orden de soldadura.
3. Minimizar el número de varillas utilizadas.
4. Minimizar el precio total de las varillas utilizadas.

Cada algoritmo tiene que resolver exactamente el problema pedido en cada caso y no hay que calcular en ningún caso las varillas necesarias. Por supuesto, hay que minimizar los costes en espacio y tiempo.

Ejercicio 2 (**El rapero con sueltecillo**) Un rapero lleva excesivo dinero suelto en el bolsillo de su pantalón, lo que contribuye a que lleve los pantalones demasiado caídos. Como consecuencia, cada vez que paga una cantidad C, su deseo es emplear el mayor número posible de monedas. Suponiendo que para todo i, $1 \leq i \leq n$, tiene n_i monedas del tipo m_i, diseñar un algoritmo que le diga cuáles monedas debe emplear.

Ejercicio 3 Inspirarse en el algoritmo de Floyd para diseñar un algoritmo que calcule el cierre reflexivo y transitivo de una relación binaria dada mediante un grafo dirigido.

Ejercicio 4 Un archipiélago consta de unas cuantas islas y varios puentes que unen ciertos pares de islas entre sí. Para cada puente (que puede ser de dirección única o doble), además de saber la isla de origen y la de destino, se conoce su anchura > 0. La anchura de un camino formado por una sucesión de puentes es la anchura mínima entre las anchuras de los puentes que lo forman. Para cada isla, se desea saber cuál es el camino de anchura máxima que las une (siempre que exista alguno).

Ejercicio 5 Jaimito, Juanito y Jorgito quieren construir una cabaña de exploradores en la playa. Con el objetivo de protegerla de la marea, van a levantar cuatro pilares sobre los que construirla y para ello disponen de un montón de ladrillos. Ahora bien, los ladrillos proceden de muy diversas fábricas y sus alturas no son las mismas. El objetivo de los jóvenes castores es determinar la distribución de los ladrillos para que el pilar más pequeño sea tan alto como sea posible. Suponiendo que las alturas de los ladrillos son valores enteros, resolver el problema mediante programación dinámica, indicando costes en tiempo y espacio.

Ejercicio 6 El grado de relación de dos genes se mide en función de hasta qué punto se pueden alinear. Para formalizar esta idea, piensa en un gen como en una cadena sobre el alfabeto $\Sigma = \{A, C, G, T\}$. Considera dos genes, $x = ATGCC$ e $y = TACGCA$. Una alineación de x e y es una forma de emparejar estas dos cadenas escribiéndolas en columnas; por ejemplo:

$$\begin{array}{c} -AT-GCC \\ TA-CGCA \end{array}$$

El guión "$-$" señala un hueco. Los caracteres de cada cadena deben aparecer en orden y cada columna debe contener un carácter de al menos una de las dos cadenas. La puntuación de un

alineamiento se calcula sumando las puntuaciones de los pares emparejados, utilizando para ello una matriz de puntuación P de tamaño 5×5, donde la columna y la fila extras se utilizan para acomodar los huecos. Por ejemplo, el alineamiento anterior tiene la siguiente puntuación:

$$P[-,T] + P[A,A] + P[T,-] + P[-,C] + P[G,G] + P[C,C] + P[C,A]$$

Escribir un algoritmo de programación dinámica que, dadas dos cadenas $x[1..n]$ y $y[1..m]$ y una matriz de puntuación P devuelva el alineamiento de mayor puntuación.

Ejercicio 7 El carpintero Ebanisto recibe el encargo de cortar un tronco en n trozos, con cortes que han sido previamente marcados sobre la madera. Llamaremos $l_1, \ldots, l_{n+1}$ a las longitudes de los fragmentos resultantes tras hacer los n cortes establecidos. Sabemos que el esfuerzo de cortar cada fragmento de tronco en dos es proporcional a su longitud. Ebanisto se da cuenta de que el orden en que realice los cortes influye en el esfuerzo total empleado en el proceso de corte del tronco.

Diseñar e implementar un algoritmo que indique a Ebanisto un orden posible en el que debe realizar los cortes para que el esfuerzo total sea mínimo. Indicar en particular los tipos de datos concretos que se utilizan en la implementación. Analizar el coste en tiempo y en espacio del algoritmo propuesto.

Ejercicio 8 Nos dan una secuencia $x_1, \ldots, x_n$ de n números naturales cuyo orden no podremos alterar. Se pide decidir si es posible intercalar entre cada dos de ellos los signos $+$ o $-$, de manera que el valor de la expresión resultante, evaluada de izquierda a derecha, y sin ningún tipo de precedencia entre sus operadores, sea finalmente 0. Por ejemplo, dada la secuencia $8, 2, 4, 3, 1$ podríamos en este caso obtener 0 así: $8 - 2 - 4 - 3 + 1 = 6 - 4 - 3 + 1 = 2 - 3 + 1 = -1 + 1 = 0$.

Indicación: Como paso previo a la resolución, y fijada la secuencia dada, se sugiere calcular el valor máximo que podría obtenerse al evaluar cualquiera de las expresiones resultantes.

Ejercicio 9 Sean $X = x_1, \ldots, x_n$ e $Y = y_1, \ldots, y_m$ dos secuencias formadas ambas con letras del alfabeto $\{A, C, G, T\}$, que representan dos hebras de ADN que se desea comparar. En concreto, se pretende encontrar alguna subsecuencia común a ambas secuencias de la máxima longitud posible. Entendemos por subsecuencia de una secuencia dada X a toda secuencia que resulte de eliminar de X cero o más de sus letras. Por ejemplo, $\langle C, G, A, C\rangle$ es una subsecuencia de $\langle T, A, C, A, G, C, A, C\rangle$.

Desarrollar un algoritmo de programación dinámica que resuelva el problema. Indicar la complejidad del mismo, tanto en tiempo como en memoria.

Ejercicio 10 Dada una secuencia de valores A, una supersecuencia suya es otra secuencia obtenida a partir de A añadiéndole cero o más valores; por ejemplo, ALGORITMIA es una supersecuencia de GRITA y de ARMA.

Desarrollar un algoritmo de programación dinámica que, dadas dos secuencias $A[1..n]$ y $B[1..m]$, encuentre una supersecuencia común más corta; por ejemplo, dadas las secuencias GRITA y ARMA, supersecuencias comunes más cortas son GARITMA y AGRIMTA.

Se valorarán todos los pasos: definición de la recurrencia, implementación del algoritmo y análisis de los costes en tiempo de ejecución y memoria adicional del mismo.

Ejercicio 11 Durante la fiesta de Halloween, y con el fin de recaudar dinero para el viaje de fin de curso, Alicia tiene P prendas disponibles para vender a sus vecinos para que se confeccionen sus disfraces. Puede vender a cada uno de ellos un número variable k de prendas, con $0 \leq k \leq h$, siendo h una constante dada, y para cada vecino, $i \in \{1 \cdots n\}$, Alicia sabe qué beneficio b_{ik} obtendría si le vendiera k prendas.

Implementar un algoritmo que nos diga cuántas prendas habrá de vender Alicia a cada uno de sus vecinos, de forma que el beneficio total obtenido sea máximo. Indicar justificadamente su coste en tiempo y en espacio.

Ejercicio 12 Diseñar un algoritmo que encuentre el máximo valor que puede obtenerse colocando paréntesis de forma apropiada en la expresión

$$x_1/x_2/x_3/\cdots/x_{n-1}/x_n,$$

donde $x_1, x_2, \ldots, x_n$ son números reales positivos y '/' denota su división. El algoritmo debe mostrar también la forma en la que los paréntesis deben colocarse para obtener dicho valor máximo. Por ejemplo, si la expresión es 3/1/2, colocando los paréntesis de la forma (3/1)/2 obtenemos $1,5$ mientras que colocando los paréntesis de la forma 3/(1/2) obtenemos 6.

Ejercicio 13 Nos dan un polígono P convexo de n vértices, numerados $1, \ldots, n$ en sentido horario. Queremos descomponerlo en triángulos, trazando suficientes diagonales de forma que estas no se corten entre sí. Nos dan también una matriz d simétrica $n \times n$ tal que d_{ij} contiene la longitud de la diagonal que une los vértices i y j. Por conveniencia, si esos dos vértices son el mismo, o son adyacentes—es decir, no forman una diagonal, sino un lado del polígono—, la matriz contiene $d_{ij} = 0$. Desarrollar un algoritmo que calcule la descomposición en triángulos de un polígono P dado, cuya suma de longitudes de sus diagonales sea mínima.

Ejercicio 14 La empresa *SlowTravel* ofrece un recorrido en burro por la Alcarria. La ruta consiste en visitar n pueblos en un orden preestablecido $1, 2, \ldots, n$. En cada pueblo, el viajero tiene la posiblidad de cambiar de burro o de seguir con el que tiene. Se conoce el coste p_{ij} de alquilar un pollino en cada pueblo i y dejarlo en cualquier pueblo j situado más adelante en la ruta. Desarrollar un algoritmo que encuentre la secuencia de alquileres de pollinos para hacer la ruta completa, que tenga coste mínimo. Los costes en tiempo y en espacio del algoritmo no han de exceder respectivamente $O(n^2)$ y $O(n)$.

Ejercicio 15 Dado n, se desea calcular el número de árboles binarios de búsqueda que se pueden formar con n claves. Se pide desarrollar un algoritmo de programación dinámica para resolver este problema. Mostrar la ejecución del algoritmo para $n = 5$.

Ejercicio 16 Dada una ecuación lineal de la forma $a_1x_1 + \cdots + a_nx_n = c$, con coeficientes $a_i > 0, 1 \leq i \leq n, c \geq 0$ y variables x_i que toman valores en los naturales, se pide un algoritmo de programación dinámica que devuelva el número de tuplas $(x_1, \ldots, x_n)$ que son solución a dicha ecuación. Dar el coste en tiempo del algoritmo y conseguir un coste en espacio en $O(c)$.

Ejercicio 17 Dada una secuencia de enteros $\langle v_1, \ldots, v_n \rangle$, se desea encontrar la subsecuencia de suma máxima que no contenga elementos adyacentes. Por ejemplo, en la secuencia $\langle 0, 1, 2, 3 \rangle$, la subsecuencia de suma máxima sin elementos adyacentes es $\langle 1, 3 \rangle$. Se pide un algoritmo de programación dinámica que devuelva dicha subsecuencia. Dar el coste en tiempo y en espacio del algoritmo.

Ejercicio 18 Tenemos un array de n naturales S que en cada posición i ($1 \leq i \leq n$) contiene el máximo salto que puede darse hacia el final del array desde la posición i y un 0 en caso de que no se pueda saltar a dicha posición. Un salto k permite avanzar de golpe de la posición i a la posición $i + k$ siempre que en dicha posición no haya un 0. Se desea calcular cuál es el menor número de saltos necesarios para llegar desde la primera posición del array a la última, así como conocer la secuencia de posiciones del array por las que se ha pasado. Por ejemplo, si el array es $[1, 3, 8, 0, 1, 1, 1]$, el menor número de saltos es 3 y la secuencia de posiciones es $1, 2, 3, 7$.

Se pide desarrollar un algoritmo de programación dinámica para resolver este problema. Se valorarán todos los pasos: definición de la recurrencia, implementación del algoritmo y análisis de los costes en tiempo y espacio.

Ejercicio 19 Un número natural está formado por dígitos no decrecientes si todos los dígitos más significativos que cada dígito son menores o iguales que dicho dígito. Por ejemplo, los números 1234 y 111 están formados por dígitos no decrecientes, mientras que 98 no. Se pide desarrollar un algoritmo de programación dinámica para calcular cuántos números de hasta n dígitos están formados por dígitos no decrecientes. Por ejemplo, para $n = 1$ hay 10 y para $n = 2$ hay 55. Se valorarán todos los pasos: definición y explicación de la recurrencia, implementación del algoritmo, optimización en espacio y análisis de los costes en tiempo y espacio.

Nota: Ayuda considerar que los números de menos de n dígitos tienen n dígitos porque llevan 0s a la izquierda del más significativo.

Ejercicio 20 Disponemos de una colección de libros numerados (de 1 a n) de de diferentes alturas $h_1, \ldots, h_n$ y anchuras $w_1, \ldots, w_n$, que deseamos colocar en estanterías apiladas unas encima de otras. Se desea colocar los libros en estanterías siempre respetando la numeración dada y siguiendo

las siguientes normas: (1) el primer libro se coloca en la estantería superior a la izquierda del todo, (2) dentro de una misma estantería se van colocando de izquierda a derecha, (3) cada nueva estantería se añade por debajo de las anteriores y (4) se admiten estanterías vacías pero todas estarán debajo de las ocupadas. En ningún caso un libro con número i puede tener a su derecha otro que no sea el $i+1$, y si el último libro (el situado más a la derecha) de una estantería tiene número i entonces el primer libro (el situado más a la izquierda) de la primera estantería por debajo que contenga algún libro será el $i+1$.

Se pide desarrollar algoritmos de programación dinámica para resolver los dos problemas siguientes:

1. Suponiendo que están disponibles un número ilimitado de estanterías de anchura fija W y alturas cualesquiera, determinar cómo colocar los libros en estanterías (según lo explicado) de forma que se minimice la altura total de la pila de estanterías. El coste esperado en espacio es $\Theta(n)$.
2. Suponiendo disponibles $K \geq 1$ estanterías de altura suficiente para colocar cualquier libro, determinar cómo colocar los libros en estanterías (según lo explicado) de forma que se minimice la anchura de la estantería más ancha. El coste esperado en espacio está en $\Theta(nK)$.

Sugerencia: es conveniente decidir de golpe todos los libros a colocar en la estantería en curso.

Ejercicio 21 Un viajero debe realizar una ruta por el desierto de Kalahari sin morir de sed. Conoce la posición de $n \geq 2$ oasis en los que puede parar a rellenar su cantimplora de C litros de capacidad con agua fresca, y sabe que su consumo de agua es de L litros por kilómetro. Los oasis están numerados de 1 a n, siendo el oasis 1 el de salida, en el que llena completamente su cantimplora, y n el de destino. El vector $D[1..n]$ indica para cada oasis i, la distancia en kilómetros desde el oasis 1 hasta el oasis i. Cuando para, vacía completamente su cantimplora antes de llenarla de agua fresca. Un guardia le hace pagar una multa por el cuadrado de los litros del agua desperdiciada cada vez que para. En el oasis 1 no paga nada pero en el n paga por lo que lleve al llegar a destino. Se pide desarrollar un algoritmo de programación dinámica que ayude al viajero a decidir las paradas que debe hacer de forma que el dinero total en multas por agua desperdiciada sea lo menor posible. Se valorarán todos los pasos: definición de la recurrencia, implementación del algoritmo, reconstrucción de la solución y análisis de los costes en tiempo de ejecución y memoria adicional del mismo.

Ejercicio 22 Robbie es un ladrón profesional que está planeando saquear algunas de las $n \geq 1$ casas situadas consecutivamente en una larga avenida. Tras una intensa labor de vigilancia ha calculado la cantidad de dinero que podrá obtener con la venta de lo robado en cada casa. Esta información la tiene almacenada en un vector d de n naturales en el que $d[i]$ representa el dinero que podrá obtener si roba en la i-ésima casa. El objetivo de Robbie es maximizar la cantidad total de dinero obtenido teniendo en cuenta que no puede robar en dos casas adyacentes (para cada $1 \leq i \leq n-1$, las casas i e $i+1$ son adyacentes) porque esto provocaría que el sistema de alarma contacte automáticamente con la policía.

Desarrollar un algoritmo de programación dinámica que ayude a Robbie a determinar en qué casas robar para maximizar el dinero total sin que le pille la policía. Se valorarán todos los pasos: definición de la recurrencia, implementación del algoritmo y análisis de los costes en tiempo de ejecución y memoria adicional del mismo.

Ejercicio 23 Supongamos que disponemos de una función *EsPalabra* que, dada una secuencia de caracteres $A[1..n]$ y dos posiciones i, j tales que $1 \leq i \leq j \leq n$, nos devuelve *cierto* si la secuencia $A[i..j]$ corresponde a una palabra y *falso* en caso contrario.

Queremos saber si una secuencia de caracteres $A[1..n]$ se puede separar en una secuencia de palabras. Por ejemplo, la secuencia $A =$ *"lacamalo"* se puede separar en [*"la"*, *"cama"*, *"lo"*] si *EsPalabra*$(A, 1, 2)$, *EsPalabra*$(A, 3, 6)$ y *EsPalabra*$(A, 7, 8)$ devuelven *cierto*.

Diseñar e implementar un algoritmo de programación dinámica que resuelva el problema de determinar si existe o no alguna forma de separar una secuencia dada en palabras y, en caso de que exista, devolver una de las posibles. Se valorarán todos los pasos: definición de la recurrencia, implementación del algoritmo y análisis de los costes en tiempo de ejecución y memoria adicional del mismo.