Departamento de Cs. e Ingeniería de la Computación
Universidad Nacional del Sur

Algoritmos y Complejidad

Trabajo Práctico 2

Programación Dinámica

primer semestre de 2023

1. Principio de Optimalidad.

   (a) Mostrar que el principio de optimalidad **no** se aplica al problema de encontrar el camino simple *más largo* entre dos ciudades. Se dice que un camino es simple si no pasa dos veces por el mismo lugar.

2. Números de Fibonacci. Explicar cómo se puede resolver el problema de encontrar el n-ésimo número de Fibonacci utilizando la técnica de programación dinámica. *Pista:* revisar los algoritmos vistos, y ver cuále se adapta a esta estrategia)

3. *Subsecuencia creciente más larga*

   Dado una lista de números $A[1 \dots N]$, una subsecuencia es una lista de valores $A[i_1], A[i_2], \dots, A[i_k]$, donde $1 \leq i_1 < i_2 < \dots < i_k \leq N$, y se dice que es creciente si $A[i_1] < A[i_2] < \dots < A[i_k]$. Se pide encontrar un algoritmo con un tiempo en $O(N^2)$ que halle la longitud de la subsecuencia creciente más larga.

   (a) Utilizando las instancias $LIS(i)$ = "longitud de la subsecuencia creciente más larga cuyo primer índice es i". Plantear una recurrencia que defina $LIS(i)$ en función de los valores de $LIS(j)$ para $i + 1 \leq j \leq N$.

   (b) ¿Cómo se calcula la solución al problema a partir de los valores $LIS(1), LIS(2), \dots, LIS(N)$?.

   (c) Dar pseudocódigo que resuelva el problema y analizar orden asintótico de espacio y tiempo de ejecución.

4. Carrera de Relevos.

   Un equipo de corredores debe participar en una carrera de relevos que consiste de m tramos diferentes. El entrenador del equipo dispone una tabla de tiempos V tal que $V[i, j]$ contiene el tiempo promedio que le lleva a cualquier corredor (se asume que todos corren a la misma velocidad) realizar los tramos desde i hasta j y desea encontrar una secuencia de relevos que minimice el tiempo total de carrera. Un corredor puede dejar la carrera al inicio de uno de los tramos para ser relevado por otro corredor que continuará la carrera. Cada relevo insume un tiempo determinado de acuerdo al punto de la carrera donde se encuentre. Se dispone de una tabla con los tiempos que consume cada relevo, $R[i]$ indica el costo, en tiempo, que insume realizar un relevo al inicio del tramo i. Se desea encontrar una secuencia de relevos que minimice el tiempo total de carrera.

   (a) Explicar cómo se puede resolver el problema utilizando programación dinámica. Mostrar la recurrencia que codifica la solución.

   (b) Dar un algoritmo basado en la recurrencia del inciso anterior que resuelva el problema y retorne la secuencia de relevos que minimiza el tiempo total de la carrera.

   (c) Analizar el orden del tiempo de ejecución y el espacio consumido por el algoritmo.

5. Un sapo se mueve sobre una línea recta dando saltos. Inicialmente está parado en la coordenada 0 de la recta y se quiere mover hasta un punto en la coordenada entera D.

   Para esto, puede dar saltos de longitud entera eligiendo alguna de las posibles longitudes $s_1, s_2, \dots, s_N$. Los saltos siempre avanzarán hacia su objetivo en la coordenada D, no es posible retroceder.

Se pide determinar la cantidad de maneras diferentes en que puede llegar el sapo de 0 a D utilizando los saltos permitidos.

Por ejemplo, si $D = 3$ y $s_1 = 1, s_2 = 2$, hay 3 maneras posibles de saltar:

- $1 + 1 + 1$
- $1 + 2$
- $2 + 1$

(a) Plantear una recursión que resuelva el problema, utilizando subinstancias adecuadas e identificando el caso base.

(b) Dar pseudocódigo de un algoritmo que resuelva el problema.

(c) Hallar el orden asintótico de espacio y tiempo de ejecución.

6. Dado un arreglo de enteros no necesariamente positivos $A[1 \dots N]$, se desea hallar el subarreglo $A[i \dots j]$ cuya suma sea máxima. Es decir, se desea hallar un par de índices $1 \leq i \leq j \leq N$ tal que la suma $A[i] + A[i + 1] + \dots + A[j]$ sea la máxima posible.

   Por ejemplo, para el arreglo $A = [3, -4, 5, -2, -2, 6, -3, 5, -3, 2]$, el subarreglo de suma máxima es $[5, -2, -2, 6, -3, 5]$ cuya suma es 9.

   (a) Encontrar una estrategia de Programación Dinámica que resuelva el problema en $O(N)$.

   (b) Dar pseudocódigo del algoritmo.

7. *Problema de la mochila*

   Se disponen de N items numerados del 1 al N. El item número i tiene un valor v_i y un peso w_i, ambos enteros. Se los quiere transportar en una mochila que soporta un peso entero W. Los items no pueden ser fraccionados, es decir que cada uno de ellos debe ser transportado íntegramente o no ser utilizado en absoluto. Se desea maximizar el valor total de los items transportados con la restricción que el peso total sea menor o igual a W.

   Si queremos aplicar una estrategia de Programación Dinámica para resolver el problema:

   (a) ¿Cuáles son las subinstancias a resolver?

   (b) Plantear una recursión para la solución de cada instancia a partir de subinstancias más pequeñas. No olvidar los casos base.

   (c) Escribir el pseudocódigo del algoritmo que resuelve el problema utilizando esta estrategia.

   (d) Analizar el espacio y tiempo de ejecución.

8. Alineamiento de Secuencias

   Considere las siguientes secuencias de letras

   EAKKLNDAQAPKDN
   EAKSDEAEALKSDE

   Y considere ahora, estas mismas dos secuencias con algunas separaciones ("-"), las cuales permiten obtener una determinada alineación.

   EAK**K**LND**A**Q**A**P**K-D**N
   EA-KSDE**A**E**A**L**K**S**D**E

   El puntaje que puede asignarse a una alineación es la suma de la cantidad de pares de letras iguales alineadas, menos una penalidad w por cada separación ("-") insertada, menos una penalidad p por cada par de letras distintas que quedan alineadas. La alineación anterior tiene un puntaje igual a: $7 - 2w - 6p$. El problema consiste en encontrar el mayor puntaje que puede obtenerse con una alineación.

   (a) Plantear una estrategia de Programación Dinámica para resolver el problema. Mostrar la recurrencia que modela la estrategia y explicarla claramente.

   (b) Dar un algoritmo de Programación Dinámica que resuelva el problema en base a la estrategia planteada.

   (c) Analizar el tiempo de ejecución de este algoritmo.

9. Multiplicación de matrices en cadena.

   El problema de la multiplicación encadenada de matrices consiste en encontrar la mejor manera de realizar el producto de n matrices: $M = M_1 M_2 \dots M_n$.

   (a) Modificar el pseudocódigo visto en teoría para que devuelva no solamente la mínima cantidad de productos necesarios, sino también la manera en que este producto óptimo debe ser calculado.

   (b) Encontrar una "parentización" optimal de una multiplicación de matrices, cuya secuencia de dimensiones es $\langle 5, 10, 3, 12, 5, 50, 6 \rangle$.

10. Triangularización optimal de polígonos.

    Considere el problema de encontrar una triangularización optimal de un polígono convexo de n lados.

    (a) Determinar qué relación existe entre este problema y el problema de la multiplicación encadenada de matrices.

    (b) Especificar el pseudocódigo completo que resuelva el problema de minimizar la suma total de los perímetros determinados por los triángulos de la parentización, utilizando la técnica de PD.

    (c) Modificar su pseudocódigo para que devuelva la manera en la que este producto óptimo debe ser calculado.

    (d) Probar que toda triangularización de un polígono convexo de n vértices tiene n-3 cuerdas y divide al polígono en n-2 triángulos.

11. Subsecuencia Común Más Larga.

    En el problema de la subsecuencia común más larga (SCML) se consideran dos secuencias $X =< x_1, ..., x_m >$ e $Y =< y_1, ..., y_n >$ y se desea encontrar una SCML de X e Y. Se dice que $Z =< z_1, ..., z_k >$ es una subsecuencia de X si existe una secuencia de índices de X, $i_1, ..., i_k$ tal que $x_{i_j} = z_j$ para $j = 1, 2, ..., k$. Z será común a X e Y si es subsecuencia de ambas y será una SCML si no existe otra de longitud mayor.

    (a) Explicar cómo se puede solucionar el problema de encontrar una SCML usando programación dinámica.

(b) Mostrar la recurrencia que define recursivamente el valor de la solución optimal y el valor de las soluciones a los casos básicos.

(c) Escribir el algoritmo de programación dinámica que implementa la solución descripta en el inciso anterior.

12. *Juego con monedaas*

Alice y Bob decidieron jugar a un juego sencillo con monedas. Inicialmente, se disponen en una hilera N monedas con valores $v_1, v_2, \ldots, v_N$ en ese orden de izquierda a derecha.

El juego se desarrolla por turnos, jugando alternativamente cada uno de los dos jugadores y empezando el juego Alice. En su turno el jugador decide tomar una moneda y retirarla de la hilera, sus únicas dos opciones son tomar la moneda del extremo izquierdo o la moneda del extremo derecho. El juego finaliza cuando ya no quedan más monedas.

Llamemos S_A a la suma de valores de las monedas tomadas por Alice y S_B a la suma de valores de las monedas tomadas por Bob. El objetivo de Alice es maximizar $S_A - S_B$ y el de Bob es maximizar $S_B - S_A$.

Se pide desarrollar un algoritmo que determine cuál es el máximo valor de $S_A - S_B$ que puede lograr Alice asumiendo que Bob juega de manera optimal. El algoritmo también deberá poder reconstruir la estrategia optimal de Alice.

(a) Identificar la subinstancias del problema más adecuadas para resolver el problema con una estrategia de Programación Dinámica.

(b) Mostrar la recurrencia que define la solución a una instancia en función de subinstancias más pequeñas.

(c) Escribir el pseudocódigo del algoritmo que resuelve el problema. El algoritmo deberá poder responder cuál es el valor de $S_A - S_B$ en una estrategia optimal de Alice así como también determinar qué movimiento debe hacer Alice en cada subinstancia del problema.