Proyecto para la casa

Análisis y Diseño de Algoritmos (ADA)

16 de mayo de 2020

1. Introducción

Este es un proyecto grupal para el curso de Análisis y Diseño de Algoritmos, sirve como nota dentro del rubro "proyectos". El grupo es de máximo tres personas.

2. Secuencias

Dado un arreglo A de ceros y unos, un *bloque* en A es un subarreglo de unos. Por ejemplo, $[0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0]$ tiene tres bloques. Cada bloque puede ser denotado por $[i, j]$, donde i es el índice inicial del bloque y j es el índice final. Si A tiene m bloques, entonces podemos ordenar dichos bloques por índice inicial, denotamos $A_1, A_2, \ldots A_m$ a este conjunto de bloques. Por ejemplo, el arreglo A tendría tres bloques $A_1 = [3, 5], A_2 = [7, 7], A_3 = [9, 10]$.

Considere un segundo arreglo B, de ceros y unos, con sus correspondientes bloques $B_1, B_2, \ldots, B_n$. Queremos asociar los segmentos de A con los segmentos de B. Más formalmente, un *matching* entre A y B es un conjunto M de pares ordenados (i, j) que cumple las siguientes condiciones.

(1) Todo índice entre 1 y m aparece alguna vez en la primera coordenada de algún par ordenado en M. Todo índice entre 1 y n aparece alguna vez en la segunda coordenada de algún par ordenado en M.

(2) Si $(i_1, j_2) \in M$, $i_1 < i_2$ y $j_1 < j_2$, entonces $(i_2, j_1) \notin M$

(3) Si $(i_1, j_1), (i_2, j_2) \in M$ con $i_1 < i_2$ y $j_1 < j_2$, entonces $(i_1, j_2), (i_2, j_1) \notin M$

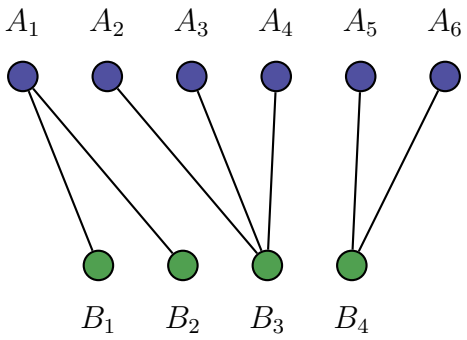
En palabras, un matching corresponde a una transformación entre de bloques de A hacia los bloques de B, tal que algunos bloques de A son divididos y otros bloques de son agrupados.

La primera condición asegura que todos los bloques de A y todos los bloques de B se encuentran en el matching. Es decir, todos los bloques de A serán transformados en algún bloque de B (ya sea por una división o un agrupamiento de bloques).

La segunda condición asegura que no es posible hacer un matching "cruzado". Es decir, la transformación de A en B sigue un orden, ya que no pueden haber bloques que se transforman de manera cruzada.

La tercera condición asegura que un bloque de A no puede ser partido y agrupado a la vez. Solo se puede hacer una operación con dicho bloque.

A continuación mostramos un ejemplo de matching.



Dicho ejemplo podría estar asociado al siguiente par de vectores

$$A = [0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0]$$
$$B = [0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0]$$

Se trata de una transformación de los bloques de A en los bloques de B. En el ejemplo, el bloque $A_1 = [2, 4]$ ha sido transformado en dos bloques: $B_1 = [3, 4]$ y $B_2 = [6, 8]$ mediante un proceso de división. Mientras que los bloques $A_2 = [2, 4], A_3 = [7, 7], A_4 = [9, 10]$ han sido transformados en un bloque: $B_3 = [11, 15]$ mediante un proceso de agrupación.

Formalmente, dado un matching entre dos vectores A y un índice i, una *i-división* es un subconjunto $(i, j_1), (i, j_2), \ldots, (i, j_k)$ del matching original. Note que, debido a la definición de matching, $j_1, j_2, \ldots, j_k$ son índices consecutivos. Y dado un índice j, un *j-agrupamiento* es un subconjunto $(i_1, j), (i_2, j), \ldots, (i_\ell, j)$ del matching original. Note que, debido a la definición de matching, $i_1, i_2, \ldots i_\ell$ son índices consecutivos.

Si $D = \{(i, j_1), (i, j_2), \ldots, (i, j_k)\}$ es una división, entonces el *peso* asociado a dicha división es

$$w(D) = \frac{|A_i|}{|B_{j_1}| + |B_{j_2}| + \cdots + |B_{j_k}|}$$

En el ejemplo, el peso de la división $(1, 1), (1, 2)$ es igual a $|A_1|/(|B_1|+|B_2|) = 3/(2+2) = 3/4$.

Si $D = \{(i_1, j), (i_2, j), \ldots, (i_\ell, j)\}$ es una agrupación, entonces el *peso* asociado a dicha agrupación es

$$w(D) = \frac{|A_{i_1}| + |A_{i_2}| + \cdots + |A_{i_\ell}|}{|B_j|}.$$

En el ejemplo, el peso de la agrupación $(2, 3), (3, 3), (4, 3)$ es igual a $(|A_2|+|A_3|+|A_4|)/|B_3| = (1 + 2 + 2)/5 = 1$.

El peso de un matching M, denotado por $w(M)$, es definido como la suma de los pesos de las agrupaciones y las divisiones en M. Esto es,

$$w(M) = \sum_{D: D \text{ es una division o agrupacion en } M} w(D).$$

En el ejemplo de la figura, el peso del matching es $\frac{3}{4} + 1 + \frac{4}{2} = 3{,}75$.

Problema Min-Matching. Dados dos vectores A y B de ceros y unos, encontrar un matching entre A y B de peso mínimo.

Diseñaremos algoritmos greedy y de programación dinámica para el problema Min-Matching.

Pregunta 1 (Voraz). Analise, diseñe e implemente un algoritmo voraz con complejidad lineal para el problema Min-Matching. Su algoritmo no deberá encontrar necesariamente el matching de peso mínimo.

Entrada del algoritmo: Dos arreglos A y B de ceros y unos de tamaño p, con n bloques y m bloques respectivamente (los valores de n y m no son recibidos como entrada).

Salida del algoritmo: Un matching entre A y B, no necesariamente óptimo, y su peso.

Tiempo de ejecución del algoritmo: $O(\text{máx}\{m, n\})$.

El algoritmo anterior, aunque es rápido, no resuelve el problema de manera exacta. El siguiente paso es diseñar un algoritmo recursivo, para ello encontraremos un subproblema al problema original. Sea $OPT(i, j)$ el peso de una solución óptima para el subproblema que solo considera a los i primeros bloques de A y a los j primeros bloques de B.

Pregunta 2 (Recurrencia). Plantee una recurrencia para $OPT(i, j)$.

Basada en la recurrencia hallada en la Pregunta 2, diseñaremos tres algoritmos.

Pregunta 3 (Recursivo). Analise, diseñe e implemente un algoritmo recursivo con complejidad exponencial para el problema Min-Matching. Su algoritmo deberá encontrar el matching de peso mínimo.

Entrada del algoritmo: Dos arreglos A y B de ceros y unos de tamaño p, con n bloques y m bloques respectivamente (los valores de n y m no son recibidos como entrada).

Salida del algoritmo: Un matching entre A y B, de peso mínimo, y su peso.

Tiempo de ejecución del algoritmo: $\Omega(2^{(\text{máx}\{m,n\})})$.

Pregunta 4 (Memoizado). Analise, diseñe e implemente un algoritmo memoizado para el problema Min-Matching. Su algoritmo deberá encontrar el matching de peso mínimo.

Entrada del algoritmo: Dos arreglos A y B de ceros y unos de tamaño p, con n bloques y m bloques respectivamente (los valores de n y m no son recibidos como entrada).

Salida del algoritmo: Un matching entre A y B, de peso mínimo, y su peso.

Tiempo de ejecución del algoritmo: $O(mn)$.

Memoria gastada por el algoritmo: $O(mn)$.

Pregunta 5 (Programación Dinámica). Analise, diseñe e implemente un algoritmo de programación dinámica para el problema Min-Matching. Su algoritmo deberá encontrar el matching de peso mínimo.

Entrada del algoritmo: Dos arreglos A y B de ceros y unos de tamaño p, con n bloques y m bloques respectivamente (los valores de n y m no son recibidos como entrada).

Salida del algoritmo: Un matching entre A y B, de peso mínimo, y su peso.

Tiempo de ejecución del algoritmo: $O(mn)$.

Memoria gastada por el algoritmo: $O(mn)$.

3. Transformación de imágenes

Dadas dos matrices $A[1..p, 1..q], B[1..p, 1..q]$ de ceros y unos, una *transformación* de A en B es un conjunto $M = \{M_1, M_2, \ldots, M_p\}$, donde cada M_i es un matching entre los vectores $A[i]$ y $B[i]$. El peso de M, denotado por $w(M)$ es igual a la suma de pesos de cada M_i es decir

$$w(M) = \sum_{i=1}^{n} w(M_i).$$

Problema Min-Transformacion. Dadas dos matrices A y B de ceros y unos, encontrar una transformación entre A y B de peso mínimo.

Es claro que para resolver el problema MIN-TRANSFORMACION de manera óptima basta invocar varias veces a alguna de las subrutinas implementadas en la subsección anterior.

Pregunta 6 (Transformación Voraz). Analise, diseñe e implemente un algoritmo voraz con complejidad cuadrática para el problema MIN-TRANSFORMACION. Su algoritmo no deberá encontrar necesariamente la transformación de peso mínimo. Debe usar como subrutina al algoritmo implementado en la Pregunta 1.

Entrada del algoritmo: Dos matrices A y B de ceros y unos de tamaño $p \times q$.

Salida del algoritmo: Una transformación entre A y B, no necesariamente óptima, y su peso.

Tiempo de ejecución del algoritmo: $O(pq)$.

Pregunta 7 (Transformación Prog. Dinámica). Analise, diseñe e implemente un algoritmo de programación dinámica con complejidad cúbica para el problema MIN-TRANSFORMACION. Su algoritmo deberá devolver una transformación de peso mínimo. Debe usar como subrutina al algoritmo implementado en la Pregunta 5.

Entrada del algoritmo: Dos matrices A y B de ceros y unos de tamaño $p \times q$.

Salida del algoritmo: Una transformación óptima entre A y B y su peso.

Tiempo de ejecución del algoritmo: $O(pq^2)$.

La motivación de hacer transformación de una matriz hacia otra es poder transformar una imagen en otra mediante una curva suave.

Una manera de hacerlo es codificar cada pixel como un 0 o un 1. Para ello, puede tomar cada pixel en la escala RGB (r, g, b) y transformarlo a una escala de grises, para posteriormente escoger los que están más cerca de ser pixeles blancos (0) y los que están más cerca

de ser negros (1). Existen muchos métodos para hacer esta transformación, dependiendo de los coeficientes escogidos. [1]

Pregunta 8 (Lectura de imágenes). Implementar una función de lectura de imágenes. Deberá recibir como entrada una imagen y devolver una matriz de 0s y 1s que codifica a la imagen leída. Esta función deberá ser lo suficientemente flexible (recibir tres parámetros adicionales) para poder usar uno u otro método de transformación. Así también, deberá ser flexible para decidir cual es el umbral que decide si se transforma a 0 o a 1.

Finalmente, haremos una animación que ejemplifica una transformación de una imagen en otra. Para ello debemos analizar cada matching encontrado en la solución. Si en el matching entre $A[i]$ y $B[i]$ hay una división, entonces es razonable que el bloque correspondiente en $A[i]$ se divida en subloques proporcionales a los tamaños de los bloques correspondientes en $B[i]$. Por ejemplo, si un bloque, de tamaño 14 mediante una división termina en tres bloques de tamaño 10, 20, y 5. Entonces dicho bloque será dividido en subloques de $4, 8, 2$, cada uno de los cuales ser irá transformando progresivamente en su correspondiente bloque en $B[i]$. Un razonamiento análogo occurre en el caso de una agrupación.

Pregunta 9 (Animación). Implementar una animación para transformar imágenes. Deberá recibir como entrada dos imágenes y mostrar una animación que transforma la primera imagen en la segunda. Aunque internamente se trabaje con matrices de ceros y unos, esta animación deberá mostrar la transformación en colores.

Esta función deberá recibir adicionalmente un parámetro toma en cuenta el algoritmo a utilizar:

1. Utiliza la subrutina implementada en la Pregunta 6 (Voraz).
2. Utiliza la subrutina implementada en la Pregunta 7 (Prog. Dinámica).
3. Utiliza la subrutina implementada en la Pregunta 10 (Prog. Dinámica Mejorada).

La subrutina mejorada (item 3) es descrita a continuación. Una mejor medida de la distorsión entre dos vectores de pixels, es decir el peso de un matching, viene dado por la varianza. Sean $a_1, a_2, \ldots, a_m$ los tamaños de los bloques de un arreglo $A[x]$. Sean $b_1, b_2, \ldots, b_n$ los tamaños de los bloques de un arreglo $B[x]$. Sea $\mu = \sum_{i=1}^{n} a_i / \sum_{i=1}^{m} b_i$.

Si D es una división o agrupación en un matching entre $A[x]$ y $B[x]$, el *peso promedio* asociado a dicha división o agrupación, denotado por $\bar{w}(D)$ es

$$\bar{w}(D) = |w(D) - \mu|.$$

Análogamente, el peso promedio de un matching M, denotado por $\bar{w}(M)$ es definido como la suma de los pesos promedios de las agrupaciones y las divisiones en M.

[1] `https://en.wikipedia.org/wiki/Luma_(video)`

Pregunta 10 (Transformación Prog. Dinámica). Analise, diseñe e implemente un algoritmo de programación dinámica con complejidad cúbica para el problema MIN-TRANSFORMACION. Su algoritmo deberá devolver la transformación de peso **promedio** mínimo. Debe usar como subrutina un algoritmo que encuentra el peso promedio mínimo de un matching.

Entrada del algoritmo: Dos matrices A y B de ceros y unos de tamaño $p \times q$.
Salida del algoritmo: Una transformación óptima entre A y B y su peso.
Tiempo de ejecución del algoritmo: $O(pq^2)$.

4. Indicaciones finales

Fechas de entrega

Habrán dos fechas de entrega.

- Entrega parcial: Semana 11. Preguntas 1–4.
- Entrega final: Semana 15. Habrá exposición. Preguntas 5–10.

Entregables

Se deberá entregar una monografía hecha en Latex describiendo el análisis diseño e implementación de cada algoritmo pedido. El análisis del algoritmo debe incluir análisis de tiempo de ejecución (si es el caso plantear una recurrencia y resolverla o mostrar por inducción). El diseño del algoritmo deberá incluir el pseudocódigo del algoritmo. La implementación del algoritmo podrá ser hecha en C,C++,Python o Java. Se deberá compartir el código de las implementaciones en algún repositorio Git.

Grupos

Los grupos son de máximo 3 personas. Pueden hacer grupo si son de diferentes horarios, sin embargo, en la presentaciónes deben estar todos los integrantes del grupo. En el caso que el grupo tenga integrantes de más de un horario, la decisión de en qué horario (1.01,1.02) serán revisados será tomada por los profesores del curso, independientemente cuantos integrantes de cada horario conformen el grupo.