Proyecto de ADA - Primer Avance

Alejandro Goicochea Diego Linares Ariana Villegas

16 de Junio del 2020

Secuencias

Pregunta 1 (Voraz)

Algoritmo: Get-Blocks

Recibe: Un arreglos A de ceros y unos.
Devuelve: Un arreglo X de bloques de unos.

GET-BLOCKS(A, p)	*cost*	*times*
i = 1	c_1	1
j = 1	c_1	1
while $i \leq p$	c_3	1
tmp = 0	c_1	1
while $A[i] \neq 1$	c_4	1
i += 1	c_5	0
while $A[i] \neq 0$	c_4	$p+1$
i += 1	c_5	p
tmp += 1	c_5	p
if tmp $\neq 0$	c_6	1
X[j] = tmp	c_7	1
j += 1	c_5	1
return X	c_8	1

Tiempo de ejecución: Get-Blocks

Para T(p) el tiempo de ejecución con un array de tamaño p como entrada, tenemos que

$$\begin{aligned} T(p) &= c_1 + c_1 + c_3 + c_1 + c_4 + c_4 \cdot (p+1) + c_5 \cdot p + c_5 \cdot p + c_6 + c_7 + c_5 + c_8 \\ T(p) &= c_4 \cdot p + 2c_5 \cdot p + 3c_1 + c_3 + 2c_4 + c_5 + c_6 + c_7 + c_8 \end{aligned}$$

Si $a = c_4 + 2c_5$ y $b = 3c_1 + c_3 + 2c_4 + c_5 + c_6 + c_7 + c_8$, entonces

$$T(p) = ap + b$$

Nota: Para el análisis de los algoritmos propuestos en los siguientes apartados no se tomará en cuenta el tiempo de generar los bloques en los array de entrada.

Algoritmo: Min-Matching-Greedy

Recibe: 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).
Devuelve: Un matching entre A y B, no necesariamente óptimo, y su peso.

MIN-MATCHING-GREEDY(A, B)	*cost*	*times*
X = Get-Blocks(A,p)	.	.
Y = Get-Blocks(B,p)	.	.
n = X.size	c_1	1
m = Y.size	c_2	1
Match = $\emptyset$	c_3	1
$w = 0$	c_4	1
for $i = 1$ TO $min(n, m) - 1$	c_5	m
Match = Match $\cup$ $\{\{i, i\}\}$	c_6	$m - 1$
w += X[i]/Y[i]	c_7	$m - 1$
if $n > m$	c_8	1
for $i = 0$ TO $n - m$	c_5	$n - m + 1$
Match = Match $\cup$ $\{\{m + i, m\}\}$	c_6	$n - m$
w += X[m+i]/Y[m]	c_7	$n - m$
else		
for $i = 0$ TO $n - m$	c_5	0
Match = Match $\cup$ $\{\{n, n + i\}\}$	c_6	0
w += X[n]/Y[n+i]	c_7	0
return w, Match	c_9	1

Tiempo de ejecución: Min-Matching-Greedy

Para T(n,m) el tiempo de ejecución con un array de tamaño p como entrada, tenemos que

$$\begin{aligned}
T(n,m) &= c_1 + c_2 + c_3 + c_4 + c_5 \cdot m + c_6 \cdot (m-1) + c_7 \cdot (m-1) + c_8 + c_5 \cdot (n - m + 1) + \\
&\quad c_6 \cdot (n-m) + c_7 \cdot (n-m) + c_9 \\
T(n,m) &= c_5 \cdot m + c_6 \cdot m + c_7 \cdot m - c_5 \cdot m - c_6 \cdot m - c_7 \cdot m + c_5 \cdot n + c_6 \cdot n + c_7 \cdot n + c_1 \\
&\quad + c_2 + c_3 + c_5 - c_6 - c_7 + c_8 + c_9
\end{aligned}$$

Si $a = c_5 + c_6 + c_7 - c_5 - c_6 - c_7 = 0$, $b = c_5 + c_6 + c_7$ y $c = c_1 + c_2 + c_3 + c_5 - c_6 - c_7 + c_8 + c_9$, entonces si $n > m$

$$T(n,m) = bn + c$$

Y si $m > n$, tenemos que

$$T(n,m) = am + c$$

Entonces, podemos generalizarlo como

$$T(n,m) = max(am, bn) + c$$

Prueba que $T(n) = O(max(n, m))$

Note que para $n_0 \geq max(a,b)+1$, tenemos que

$$\begin{aligned} max(am,bn)+c &\leq max(max(a,b)\cdot m, max(a,b)\cdot n)+c \\ &\leq max(a,b)\cdot max(m,n)+c \\ &\leq (max(a,b)+1)\cdot max(m,n) \\ &\leq n_0 \cdot max(m,n) \end{aligned}$$

Y como la función max es creciente, tenemos que

$$0 \leq max(am,bn) \leq n_0 \cdot max(m,n)$$

Entonces por definición, concluimos que $T(n) = O(max(n,m))$

Pregunta 2 (Recurrencia)

Asumimos que:
$X = \text{Get-Blocks}(A,p)$
$Y = \text{Get-Blocks}(B,p)$

$$OPT(i,j) = \begin{cases} \frac{X_i}{Y_j} & i=1 \quad \wedge \quad j=1 \\ \frac{X_i}{\sum_{p=1}^{j} Y_p} & i=1 \quad \wedge \quad j>1 \\ \frac{\sum_{p=1}^{i} X_p}{Y_j} & i>1 \quad \wedge \quad j=1 \\ min(M_g(i,j), M_d(i,j)) & i>1 \quad \wedge \quad j>1 \end{cases}$$

$$M_g(i,j) = min_{k=1}^{i-1}\{\frac{\sum_{p=k+1}^{i} X_p}{Y_j} + OPT(k,j-1)\}$$

$$M_d(i,j) = min_{k=1}^{j-1}\{\frac{X_i}{\sum_{p=k+1}^{j} Y_p} + OPT(i-1,k)\}$$

Pregunta 3 (Recursivo)

Asumimos que:
$X = \text{Get-Blocks}(A,p)$
$Y = \text{Get-Blocks}(B,p)$

Algoritmo: Group

Group(X,Y,i,j)	*cost*	*times*
Min $= \infty$	c_1	1
for $k = 1$ TO $i-1$	c_2	i
accum $= 0$	c_3	$i-1$

for $p = k+1$ TO i	c_4	$\frac{(i-2)(i-1)}{2}$
accum += $X[p]$	c_5	$\frac{(i-2)(i-1)}{2}$
accum /= $Y[j]$	c_6	$i-1$
match, pmin = MIN-MATCHING-RECURSIVE$(X, Y, k, j-1)$	$T_m(i-1, j-1)$	2
if Min>accum+pmin	c_7	$i-1$
Min = accum + pmin	c_8	$i-1$
Match = match	c_9	$i-1$
return Match $\cup \{[i, j]\}$, Min	c_1	1

Tiempo de ejecución: Group

Para $T_g(i, j)$ el tiempo de ejecución con un array de tamaño p como entrada, tenemos que

$$\begin{aligned}
T_g(i,j) &= c_1 + c_2 + c_3 \cdot (i-1) + c_4 \cdot \frac{(i-2)(i-1)}{2} + c_5 \cdot \frac{(i-2)(i-1)}{2} + c_6 \cdot (i-1) + \\
&\quad 2 \cdot T_m(i-1, j-1) + c_7 \cdot (i-1) + c_8 \cdot (i-1) + c_9 \cdot (i-1) + c_1 \\
T_g(i,j) &= 2 \cdot T_m(i-1, j-1) + \frac{c_4}{2} \cdot i^2 + \frac{c_5}{2} \cdot i^2 - 3c_4 \cdot i - 3c_5 \cdot i + c_6 \cdot i + c_7 \cdot i + \\
&\quad c_8 \cdot i + c_9 \cdot i + 2c_1 + c_2 - c_3 + 3c_4 + 3c_5 - c_6 - c_7 - c_8 - c_9
\end{aligned}$$

Si $a = \frac{c_4}{2} + \frac{c_5}{2}$, $b = -3c_4 - 3c_5 + c_6 + c_7 + c_8 + c_9$ y $c = 2c_1 + c_2 - c_3 + 3c_4 + 3c_5 - c_6 - c_7 - c_8 - c_9$, entonces

$$T_g(i,j) = 2 \cdot T_m(i-1, j-1) + ai^2 + bi + c$$

Algoritmo: Division

DIVISION(X, Y, i, j)	*cost*	*times*
Min = ∞	c_1	1
for $k = 1$ TO $j-1$	c_2	j
accum = 0	c_3	$j-1$
for $p = k+1$ TO j	c_4	$\frac{(j-2)(j-1)}{2}$
accum += $Y[p]$	c_5	$\frac{(j-2)(j-1)}{2}$
accum = $X[i]$/accum	c_6	$j-1$
match, pmin = MIN-MATCHING-RECURSIVE$(X, Y, i-1, k)$	$T_m(i-1, j-1)$	2
if Min>accum+pmin	c_7	$j-1$
Min = accum + pmin	c_8	$j-1$
Match = match	c_9	$j-1$
return Match $\cup \{[i, j]\}$, Min	c_1	1

Tiempo de ejecución: Division

Para $T_d(i,j)$ el tiempo de ejecución con un array de tamaño p como entrada, tenemos que

$$\begin{aligned}
T_d(i,j) &= c_1 + c_2 + c_3 \cdot (j-1) + c_4 \cdot \frac{(j-2)(j-1)}{2} + c_5 \cdot \frac{(j-2)(j-1)}{2} + c_6 \cdot (j-1) + \\
& \quad 2 \cdot T_m(i-1, j-1) + c_7 \cdot (j-1) + c_8 \cdot (j-1) + c_9 \cdot (j-1) + c_1 \\
T_d(i,j) &= 2 \cdot T_m(i-1,j-1) + \frac{c_4}{2} \cdot j^2 + \frac{c_5}{2} \cdot j^2 - 3c_4 \cdot j - 3c_5 \cdot j + c_6 \cdot j + c_7 \cdot j + \\
& \quad c_8 \cdot j + c_9 \cdot j + 2c_1 + c_2 - c_3 + 3c_4 + 3c_5 - c_6 - c_7 - c_8 - c_9
\end{aligned}$$

Si $a = \frac{c_4}{2} + \frac{c_5}{2}$, $b = -3c_4 - 3c_5 + c_6 + c_7 + c_8 + c_9$ y $c = 2c_1 + c_2 - c_3 + 3c_4 + 3c_5 - c_6 - c_7 - c_8 - c_9$, entonces

$$T_d(i,j) = 2 \cdot T_m(i-1,j-1) + aj^2 + bj + c$$

Algoritmo: Min-Matching-Recursive

MIN-MATCHING-RECURSIVE(X, Y, i, j)	*cost*	*times*
if $i == 1$ **and** $j == 1$	c_1	1
return $\{[i,j]\}$, $\frac{X[i]}{Y[i]}$	c_2	1
if $i == 1$ **and** $j > 1$	c_3	1
match = {}	c_4	1
accumY = 0	c_5	1
for $p = 1$ TO j	c_6	$j+1$
accumY += Y[p]	c_7	j
Match = Match $\cup \{[i,p]\}$,	c_8	j
return Match, $\frac{X[i]}{accumY}$	c_2	1
if $i > 1$ **and** $j == 1$	c_3	1
match = {}	c_4	1
accumX = 0	c_5	1
for $p = 1$ TO i	c_6	$i+1$
accumX += X[p]	c_7	i
Match = Match $\cup \{[p,j]\}$,	c_8	i
return $Match$, $\frac{accumX}{Y[j]}$	c_2	1
MatchG, MinG = GROUP(X, Y, i, j)	$T_g(i,j)$	1
MatchD, MinD = DIVISION(X, Y, i, j)	$T_d(i,j)$	1
if MinG>MinD	c_3	1
return MatchD, MinD	c_2	1
return MatchG, MinG	c_2	1

Tiempo de ejecución: Min-Matching-Recursive

$$T_m(i,j) = \begin{cases} 1 & i = 1 \quad \wedge \quad j = 1 \\ \\ j & i = 1 \quad \wedge \quad j > 1 \\ \\ i & i > 1 \quad \wedge \quad j = 1 \\ \\ 4T_m(i-1, j-1) + i^2 + i + j^2 + j & i > 1 \quad \wedge \quad j > 1 \end{cases}$$

Resolver la recurrencia: Min-Matching-Recursive

El análisis será para el caso en el que $i = j$ debido a que es el peor caso. Entonces, tenemos que

$$\begin{aligned}
T_m(i,i) &= 4T_m(i-1, i-1) + i(i+1) + i(i+1) \\
&= 4T_m(i-1, i-1) + 2i(i+1) \\
&= 4(4T_m(i-2, i-2) + 2(i-1)i) + 2i(i+1) \\
&= 4^2 T_m(i-2, i-2) + 4 \cdot 2(i-1)i + 2i(i+1) \\
&= 4^2(4T_m(i-3, i-3) + 2(i-2)(i-1)) + 4 \cdot 2(i-1)i + 2i(i+1) \\
&= 4^3 T_m(i-3, i-3) + 4^2 \cdot 2(i-2)(i-1) + 4 \cdot 2(i-1)i + 2i(i+1) \\
&= 4^3 T_m(i-3, i-3) + 2\sum_{k=0}^{2} 4^k \cdot (i-k-1)(i-k) \\
&= 4^l T_m(i-l, i-l) + 2\sum_{k=0}^{l-1} 4^k \cdot (i-k-1)(i-k) \\
&= 4^{i-1} T_m(i-(i-1), i-(i-1)) + 2\sum_{k=0}^{i-2} 4^k \cdot (i-k-1)(i-k) \\
&= 4^{i-1} + 2\sum_{k=0}^{i-2} 4^k \cdot (i^2 + k^2 - 2ki - i + k)
\end{aligned}$$

Ahora suponemos que $i = 2^p$

$$\begin{aligned}
T_m(i,i) &= 4^{i-1} + 2\sum_{k=0}^{i-2} 4^k \cdot (2^{2p} + 2^k - 2\cdot 2^k \cdot 2^p - 2^p + 2^k) \\
&= 4^{i-1} + 2\sum_{k=0}^{i-2} 2^{2k} \cdot (2^{2p} + 2^k - 2\cdot 2^k \cdot 2^p - 2^p + 2^k) \\
&= 4^{i-1} + 2\sum_{k=0}^{i-2} 2^{2k} \cdot 2^{2p} + 2\sum_{k=0}^{i-2} (2^k - 2\cdot 2^k \cdot 2^p - 2^p + 2^k) \\
&= 4^{i-1} + 2\cdot 2^{2p}\sum_{k=0}^{i-2} 2^{2k} + 2\sum_{k=0}^{i-2} 2^{3k} - 2^2\cdot 2^p\sum_{k=0}^{i-2} 2^{3k} - 2\cdot 2^p\sum_{k=0}^{i-2} 2^k + 2\sum_{k=0}^{i-2} 2^{2k} \\
&= 4^{i-1} + 2\cdot 2^{2p}\sum_{k=0}^{i-2} 4^k + 2\sum_{k=0}^{i-2} 8^k - 2^2\cdot 2^p\sum_{k=0}^{i-2} 8^k - 2\cdot 2^p\sum_{k=0}^{i-2} 2^k + 2\sum_{k=0}^{i-2} 4^k \\
&= 4^{i-1} + 2\cdot 2^{2p}\frac{4^{i-2}-1}{3} + 2\frac{8^{i-2}-1}{7} - 2^2\cdot 2^p\frac{8^{i-2}-1}{7} - 2\cdot 2^p\frac{2^{i-2}-1}{1} + 2\frac{4^{i-2}-1}{3} \\
&= 4^{i-1} + 2i^2\frac{4^{i-2}-1}{3} + 2\frac{8^{i-2}-1}{7} - 4i\frac{8^{i-2}-1}{7} - 2i\frac{2^{i-2}-1}{1} + 2\frac{4^{i-2}-1}{3}
\end{aligned}$$

Además, sabemos que si $i \neq j$

$$T_m(i,j) \leq T_m(max(i,j), max(i,j))$$

Entonces, tenemos que

$$0 \leq 2^{max(i,j)} \leq T_m(i,j)$$

Entonces por definición, concluimos que

$$T(n,m) = \Omega(2^{max(n,m)})$$

Pregunta 4 (Memoizado)

Los algoritmos para las funciones group y división permanecen igual. La diferencia con el algoritmo recursivo es que almacena los datos ya calculados en una matriz de tamaño $m * n$ para evitar llamadas que calculen datos que ya sabemos. Dado esto y la definición de nuestra recurrencia en la pregunta 2, esta claro que el algoritmo va a tener tiempo de ejecución $\Theta(n * m)$ ya que es el tiempo que toma llenar toda la matriz. El algoritmo funciona de la siguiente manera:

Recibe: Dos arreglos de bits, X y Y con la cantidad de pesos que tienen i y j respectivamente.
Devuelve: El matching de peso mínimo junto con su peso.

Algoritmo: Min-Matching-Memoization

Min-Matching-Memoization(X, Y, i, j)	*cost*	*times*
if $minMatch[i][j][2]! = \infty$	c_1	1
return $minMatch[i][j][1],\ minMatch[i][j][2]$	c_2	1
if $i == 1$ **and** $j == 1$	c_3	1
$minMatch[i][j][1] = [(i, j)]$	c_4	1
$minMatch[i][j][2] = \frac{X[i]}{Y[j]}$	c_5	1
return $minMatch[i][j][1],\ minMatch[i][j][2]$	c_6	1
if $i == 1$ **and** $j > 1$	c_7	1
match = {}	c_8	1
accumY = 0	c_9	1
for $p = 1$ TO j	c_{10}	j
accumY += Y[p]	c_{11}	1
Match = Match $\cup$ $\{[i, p]\}$,	c_{12}	1
$minMatch[i][j][1] = Match$	c_{13}	1
$minMatch[i][j][2] = \frac{X[i]}{accumY}$	c_{14}	1
return $minMatch[i][j][1],\ minMatch[i][j][2]$	c_{15}	1
if $i > 1$ **and** $j == 1$	c_{16}	1
match = {}	c_{17}	1
accumX = 0	c_{18}	1
for $p = 1$ TO i	c_{19}	i
accumX += X[p]	c_{20}	1
Match = Match $\cup$ $\{[p, j]\}$,	c_{21}	1
$minMatch[i][j][1] = Match$	c_{22}	1
$minMatch[i][j][2] = \frac{accumX}{Y[j]}$	c_{23}	1
return $minMatch[i][j][1],\ minMatch[i][j][2]$	c_{24}	1
MatchG, MinG = Group(X, Y, i, j)	$i * j$	1
MatchD, MinD = Division(X, Y, i, j)	$i * j$	1
if MinG>MinD	c_{25}	1
return MatchD, MinD	c_{26}	1
return MatchG, MinG	c_{27}	1

Pregunta 5 (Programación Dinámica)

La primera diferencia que notamos con el algoritmo memoizado es que es iterativo. El algoritmo funciona de la siguiente manera: Tenemos una matriz de tamaño $n * m$ la cual va a ser llenada mediante se ejecuta el algoritmo. El algoritmo cada vez va añadiendo mas bloques con cada iteración que pasa y decide que se va a hacer con este nuevo bloque seleccionado. Con esta decisión se va llenando la matriz para que la decisión pueda ser usada en las siguientes iteraciones. Por esto es que el algoritmo tiene tiempo de ejecución $\Theta(mn^2)$. El código es el siguiente:

Recibe: Dos arreglos de bits, X y Y.
Devuelve: El matching de peso mínimo junto con su peso.

Algoritmo: Min-Matching-DP

MIN-MATCHING-DP(X, Y, mu)	*cost*	*times*
$accumX = []$	c_1	1
$accumY = []$	c_2	1
$minMatch = [[[[], math.inf]\ for\ j\ in\ range(len(Y))]\ for\ i\ in\ range(len(X))]$	$n * m$	1
$n = len(X)$	c_3	1
$m = len(Y)$	c_4	1
$accumX.append(X[0])$	c_5	1
$accumY.append(Y[0])$	c_6	1
for $i = 1$ TO n	1	n
$accumX.append(accumX[i-1]\ +\ X[i])$	c_7	1
for $j = 1$ TO m	1	m
$accumY.append(accumY[j-1]\ +\ Y[j])$	c_7	1
$minMatch[0][0] = [[(0,0)], X[0]/Y[0]]$	c_8	1
for $i = 1$ TO n	n	1
$minMatch[i][0][0] = minMatch[i-1][0][0] + [(i,0)]$	c_9	1
$minMatch[i][0][1] = abs(accumX[i]/Y[0] - mu)$	c_{10}	1
for $j = 1$ TO m	m	1
$minMatch[0][j][0] = minMatch[0][j-1][0] + [(0,j)]$	c_{11}	1
$minMatch[0][j][1] = abs(X[0]/accumY[j] - mu)$	c_{12}	1
for $i = 1$ TO n	1	n
for $j = 1$ TO m	1	m
for $k = 1$ TO i	1	i
$accum = (accumX[i] - accumX[k])/Y[j]$	c_{13}	1
if $minMatch[i][j][1] > abs(accum + pmin - mu)$	c_{14}	1
$tmatch = [(p,j)\ for\ p\ in\ range(k+1, i+1)]$	$i - k$	1
$minMatch[i][j][0] = match + tmatch$	c_{15}	1
$minMatch[i][j][1] = abs(accum + pmin - mu)$	c_{16}	1
for $k = 0$ TO j	1	j
$accum = X[i]/(accumY[j] - accumY[k])$	c_{17}	1

$[match, pmin] = minMatch[i-1][k]$ $\quad c_{18} \quad 1$
if $minMatch[i][j][1] > abs(accum + pmin - mu)$ $\quad c_{19} \quad 1$
$tmatch = [(i, p)\ for\ p\ in\ range(k+1, j+1)]$ $\quad i-k \quad 1$
$minMatch[i][j][0] = match + tmatch$ $\quad c_{20} \quad 1$
$minMatch[i][j][1] = abs(accum + pmin - mu)$ $\quad c_{21} \quad 1$
return $minMatch[n-1][m-1][0], minMatch[n-1][m-1][1]$ $\quad c_{22} \quad 1$

Lectura de imágenes

En esta parte del proyecto, utilizamos la luminosidad de la imagen para crear la matriz que contiene los bloques de bits. Gracias a la libreria imageio de python, podemos cargar una imagen junto con los valores RGB de cada pixel. Cuando ya tenemos nuestras dos imágenes, podemos utilizar uno de los tres coeficientes de luma disponibles junto con un valor (el umbral) que se pasa a la funcion para ver cuales de los pixeles de las imágenes van a volverse un 0 o 1. Los valores mayores o iguales al umbral se vuelven 1 y el resto 0. Una vez completada nuestra funcion procedemos a aplicar los algoritmos que veremos en la siguiente sección.

Algoritmos de transformación

Para las preguntas de transformaciones de imágenes, estos reciben las dos imágenes como parámetros y llaman la funcion luma que devuelve una matriz con los bits correspondientes a cada imagen donde están los bloques. Una vez tenemos ambas matrices, se llaman los algoritmos anteriores por cada fila para sacar un matching por file de toda la imagen. Como se esta trabajando con imágenes del mismo tamaño, el tiempo de ejecución seria el mismo que los algoritmos anteriores multiplicado por la cantidad de filas que tienen las imágenes.

Link del repositorio

Github link.