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*
1: i = 1	c_1	1
2: j = 1	c_1	1
3: **while** $i \leq p$	c_3	1
4: tmp = 0	c_1	1
5: **while** $A[i] \neq 1$	c_4	1
6: i += 1	c_5	0
7: **while** $A[i] \neq 0$	c_4	$p + 1$
8: i += 1	c_5	p
9: tmp += 1	c_5	p
10: **if** tmp $\neq 0$	c_6	1
11: X[j] = tmp	c_7	1
12: j += 1	c_5	1
13: return X	c_8	1

Tiempo de ejecución: Get-Blocks

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

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

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

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

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*
1: X = Get-Blocks(A,p)	.	.
2: Y = Get-Blocks(B,p)	.	.
3: n = X.size	c_1	1
4: m = Y.size	c_2	1
5: Match = $\emptyset$	c_3	1
6: $w = 0$	c_4	1
7: **for** $i = 1$ TO $min(n, m) - 1$	c_5	m
8: Match = Match $\cup$ $\{\{i, i\}\}$	c_6	$m - 1$
9: w += X[i]/Y[i]	c_7	$m - 1$
10: **if** $n > m$	c_8	1
11: **for** $i = 0$ TO $n - m$	c_5	$n - m + 1$
12: Match = Match $\cup$ $\{\{m + i, m\}\}$	c_6	$n - m$
13: w += X[m+i]/Y[m]	c_7	$n - m$
14: **else**		
15: **for** $i = 0$ TO $n - m$	c_5	0
16: Match = Match $\cup$ $\{\{n, n + i\}\}$	c_6	0
17: w += X[n]/Y[n+i]	c_7	0
18: 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*
1: Min $= \infty$	c_1	1
2: **for** $k = 1$ TO $i-1$	c_2	i
3: accum $= 0$	c_3	$i-1$

4:	**for** $p = k + 1$ TO i	c_4	$\sum_{k=1}^{i} i - k - 1$
5:	accum += $X[p]$	.	.
6:	accum /= $Y[j]$	.	.
7:	match, pmin = MIN-MATCHING-RECURSIVE$(X, Y, k, j - 1)$		
		.	.
8:	**if** Min>accum+pmin	.	.
9:	Min = accum + pmin	.	.
10:	Match = match	.	.
11:	**return** Match $\cup \{[i, j]\}$, Min	c_1	1

Algoritmo: Division

DIVISION(X, Y, i, j)	*cost*	*times*
1: Min = ∞	c_1	1
2: **for** $k = 1$ TO $j - 1$	i	.
3: accum = 0	.	.
4: **for** $p = k + 1$ TO j	.	.
5: accum += $Y[p]$	.	.
6: accum = $X[i]$/accum	.	.
7: match, pmin = MIN-MATCHING-RECURSIVE$(X, Y, i - 1, k)$		
	.	.
8: **if** Min>accum+pmin	.	.
9: Min = accum + pmin	.	.
10: Match = match	.	.
11: **return** Match $\cup \{[i, j]\}$, Min	.	.

Algoritmo: Min-Matching-Recursive

MIN-MATCHING-RECURSIVE(X, Y, i, j)	*cost*	*times*
1: **if** $i == 1$ **and** $j == 1$	c_1	1
2: **return** $\{[i, j]\}$, $\frac{X[i]}{Y[i]}$	c_2	1
3: **if** $i == 1$ **and** $j > 1$	c_3	1
4: match = {}	c_4	1
5: accumY = 0	c_5	1
6: **for** $p = 1$ TO j	.	.
7: accumY += Y[p]	.	.
8: Match = Match $\cup \{[i, p]\}$,	.	.
9: **return** Match, $\frac{X[i]}{accumY}$	.	.
10: **if** $i > 1$ **and** $j == 1$	.	.
11: match = {}	.	.
12: accumX = 0	.	.
13: **for** $p = 1$ TO i	.	.
14: accumX += X[p]	.	.
15: Match = Match $\cup \{[p, j]\}$,	.	.
16: **return** $Match$, $\frac{accumX}{Y[j]}$	.	.

17:	MatchG, MinG = GROUP(X, Y, i, j)	.	.
18:	MatchD, MinD = DIVISION(X, Y, i, j)	.	.
19:	**if** MinG>MinD	.	.
20:	**return** MatchD, MinD	.	.
21:	**return** MatchG, MinG	.	.

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*
1:	**if** $minMatch[i][j][2]! = \infty$	.	.
2:	**return** $minMatch[i][j][1],\ minMatch[i][j][2]$	c_1	1
3:	**if** $i == 1$ **and** $j == 1$	.	.
4:	$minMatch[i][j][1] = [(i, j)]$	c_2	1
5:	$minMatch[i][j][2] = \frac{X[i]}{Y[j]}$	c_3	1
6:	**return** $minMatch[i][j][1],\ minMatch[i][j][2]$	c_4	1
7:	**if** $i == 1$ **and** $j > 1$	.	.
8:	match = {}	c_5	1
9:	accumY = 0	c_6	1
10:	**for** $p = 1$ TO j	c_7	j
11:	accumY += Y[p]	c_8	1
12:	Match = Match $\cup$ $\{[i, p]\}$,	c_9	1
13:	$minMatch[i][j][1] = Match$	c_{10}	1
14:	$minMatch[i][j][2] = \frac{X[i]}{accumY}$	c_{11}	1
15:	**return** $minMatch[i][j][1],\ minMatch[i][j][2]$	c_{12}	1
16:	**if** $i > 1$ **and** $j == 1$	.	.
17:	match = {}	c_{13}	1
18:	accumX = 0	c_{14}	1
19:	**for** $p = 1$ TO i	c_{15}	i
20:	accumX += X[p]	c_{16}	1
21:	Match = Match $\cup$ $\{[p, j]\}$,	c_{17}	1
22:	$minMatch[i][j][1] = Match$	c_{18}	1

23:	$minMatch[i][j][2] = \frac{accumX}{Y[j]}$	c_{19}	1
24:	**return** $minMatch[i][j][1],\ minMatch[i][j][2]$	c_{20}	1
25:	MatchG, MinG = GROUP(X, Y, i, j)	$i * j$	1
26:	MatchD, MinD = DIVISION(X, Y, i, j)	$i * j$	1
27:	**if** MinG>MinD	.	.
28:	**return** MatchD, MinD	c_{21}	1
29:	**return** MatchG, MinG	c_{22}	1

Pregunta 5 (Programación Dinámica)