21BDS0340

Abhinav Dinesh Srivatsa

Design and Analysis of Algorithms

Digital Assignment – I

Question 1

A1 – 10 x 100
A2 – 100 x 5
A3 – 5 x 50
A4 – 50 x 7

From this:
$p_0 = 10$
$p_1 = 100$
$p_2 = 5$
$p_3 = 50$
$p_4 = 7$

Optimal multiplication with tabulation:

	1	2	3	4
1				
2	**X**			
3	**X**	**X**		
4	**X**	**X**	**X**	

Using the dynamic programming recursive formula:

$$M(i,j) = 0, if i = j$$

$$M(i,j) = min\big(M(i,k) + M(k+1,j) + p_{(i-1)}p_k p_j, otherwise\big), i \leq k < j$$

Filling the principal diagonal:

	1	2	3	4
1	**0**			
2	X	**0**		
3	X	X	**0**	
4	X	X	X	**0**

Filling the next diagonal:
M(1, 2) = M(1, 1) + M(2, 2) + $p_0p_1p_2$ = 0 + 0 + 5000 = **5000**

M(2, 3) = M(2, 2) + M(3, 3) + $p_1p_2p_3$ = 0 + 0 + 25000 = **25000**

M(3, 4) = M(3, 3) + M(4, 4) + $p_2p_3p_4$ = 0 + 0 + 1750 = **1750**

	1	2	3	4
1	0	**5000**		
2	X	0	**25000**	
3	X	X	0	**1750**
4	X	X	X	0

Filling the next diagonal:
M(1, 3) = M(1, 1) + M(2, 3) + $p_0p_1p_3$ = 0 + 25000 + 5000 = 30000
or = M(1, 2) + M(3, 3) + $p_0p_2p_3$ = 5000 + 0 + 2500 = **7500**

M(2, 4) = M(2, 2) + M(3, 4) + $p_1p_2p_4$ = 0 + 1750 + 3500 = **5250**
or = M(2, 3) + M(4, 4) + $p_1p_3p_4$ = 25000 + 0 + 35000 = 60000

	1	2	3	4
1	0	5000	**7500**	
2	X	0	25000	**5250**
3	X	X	0	1750
4	X	X	X	0

Filling the last element:
M(1, 4) = M(1, 1) + M(2, 4) + $p_0p_1p_4$ = 0 + 5250 + 7000 = 12250
or = M(1, 2) + M(3, 4) + $p_0p_2p_4$ = 5000 + 1750 + 350 = **7100**
or = M(1, 3) + M(4, 4) + $p_0p_3p_4$ = 7500 + 0 + 3500 = 11000

	1	2	3	4
1	0	5000	7500	**7100**
2	X	0	25000	5250
3	X	X	0	1750
4	X	X	X	0

Therefore, the least amount of operation is **<u>7100</u>**, and the calculations to reach it is to do:

$$\underline{(\boldsymbol{A1} * \boldsymbol{A2}) * (\boldsymbol{A3} * \boldsymbol{A4})}$$

Question 1: Algorithm

1. Create a square table with the number of matrices as dimensions N; in this case 4
2. Mark the bottom half of the table to be unused
3. Mark the principal diagonal values as 0
4. Iterate a value I from 1 to N and another value J from I + 1 to N
5. Find the minimum value of $M(i,k) + M(k+1,j) + p_{i-1}p_kp_j$ and assign it to $M(i,j)$, where $M(i,j)$ is the table value of (i,j) and $i \leq k < j$

6. Do this process till all the values in the table are found, the value at $M(1, n)$ will be the solution for the minimum number of multiplication steps; in this case 7100

Question 1: Time Complexity

The time complexity of the algorithm depends on the factors I, J and K. Since these values are in nested loops, the time complexity will be I x J x K = N x N x N, where N is the dimension of the matrices

Therefore, the time complexity is of $\underline{\boldsymbol{O(n^3)}}$

<u>Question 2</u>

Activity	1	2	3	4	5	6	7	8	9
Start	1	3	0	5	3	5	6	8	8
Finish	4	5	6	7	8	9	10	11	72

Sorting by end time:

Activity	1	2	3	4	5	6	7	8	9
Start	1	3	0	5	3	5	6	8	8
Finish	4	5	6	7	8	9	10	11	72

Creating solution set:
S = {A1}

Adding activities that start after the last end time:
S = {A1, A4, A8}

Therefore, the activities that can be done maximally are **<u>{A1, A4, A8}</u>**

Question 2: Algorithm

Let N be the number of activities

1. Sort the activities by descending order end times
2. Create a solution set and append the first activity to it
3. Iterate through the activities and append any that have their start time greater than or equal to the latest appended activity's end time
4. Display the solution set

Question 2: Time Complexity

The time complexity of the activity selection algorithm depends on 2 factors, the sorting algorithm, and the solution loop. The most efficient sorting algorithm's time complexity is $O(nlogn)$ and the solution loop's is $O(n)$.

From this, the time complexity is **<u>$\boldsymbol{O(n)}$ when the list is sorted and $\boldsymbol{O(nlogn)}$ when the list is not.</u>**

Question 3
X = (C, R, O, S, S)
Y = (R, O, A, D, S)

Using the dynamic programming recursive formula:

$$M(x, y) = \max\big(M(x-1, y), M(x, y-1)\big), if\ x \neq y$$
$$M(x, y) = 1 + M(x-1, y-1), if\ x = y$$

Optimisation table:

	''	C	R	O	S	S
''	**0**	**0**	**0**	**0**	**0**	**0**
R	**0**					
O	**0**					
A	**0**					
D	**0**					
S	**0**					

Row 'R':

	''	C	R	O	S	S
''	0	0	0	0	0	0
R	0	**0**	**1**	**1**	**1**	**1**
O	0					
A	0					
D	0					
S	0					

Row 'O':

	''	C	R	O	S	S
''	0	0	0	0	0	0
R	0	0	1	1	1	1
O	0	**0**	**1**	**2**	**2**	**2**
A	0					
D	0					
S	0					

Row 'A':

	''	C	R	O	S	S
''	0	0	0	0	0	0
R	0	0	1	1	1	1
O	0	0	1	2	2	2
A	0	**0**	**1**	**2**	**2**	**2**
D	0					
S	0					

Row 'D':

	"	C	R	O	S	S
"	0	0	0	0	0	0
R	0	0	1	1	1	1
O	0	0	1	2	2	2
A	0	0	1	2	2	2
D	0	**0**	**1**	**2**	**2**	**2**
S	0					

Row 'S':

	"	C	R	O	S	S
"	0	0	0	0	0	0
R	0	0	1	1	1	1
O	0	0	1	2	2	2
A	0	0	1	2	2	2
D	0	0	1	2	2	2
S	0	**0**	**1**	**2**	**3**	**3**

Therefore, the longest common subsequence is of length **3**. We can find the subsequence by analysing where the values in the optimisation table were incremented. The values changed at R, O, S. From this, the longest common subsequence is **ROS**.

Question 3: Algorithm

1. Create a table with the letters of each word as either of the axes
2. Iterate through all the elements in the table from top to bottom, these represent the string building one letter at a time
3. Assign $M(x, y)$ as $1 + M(x-1, y-1)$ if the letters at (x, y) match
4. Assign $M(x, y)$ as $\max\left(M(x-1, y),\ M(x, y-1)\right)$ if the letters do not match
5. Display the value at the last entry

Question 3: Time Complexity

The algorithm goes through each entry in the table without repeating any because of dynamic programming. From this, the time complexity is $\boldsymbol{O(m * n)}$, where m and n are the length of the two words.

<u>Question 4</u>

N = 3
M = 20
Prices = {25, 24, 15}
Weights = {18, 15, 10}

Calculating price per weight (value):

N1 = $\frac{25}{18}$ = 1.389

N2 = $\frac{24}{15}$ = 1.6

N3 = $\frac{15}{10}$ = 1.5

Sorting by value:
Items = {N2, N3, N1}
Prices = {24, 15, 25}
Weights = {15, 10, 18}

Selecting N2:
Knapsack weight remaining = 20 – 15 = 5
Price = 0 + 24 = 24

Selecting N3:
Knapsack weight remaining = 5 – 10 = -5 (cannot be -ve)
Therefore, a fractional amount of N3 must be selected, 5 units to fill the knapsack.
Price = 24 + 5 x N3's value
= 24 + 5 x 1.5
= 24 + 7.5
= **<u>31.5</u>**

Question 4: Algorithm

1. Create a new array from the given arrays called value, which is price per weight
2. Sort by ascending value of value
3. Iterate through the items and fill the knapsack with each until space runs out, and add the price, fractional too, to accumulate the total knapsack price
4. Display the price

Question 4: Time Complexity

The time complexity of the fractional knapsack problem can be divided into two parts, the sorting, and the calculation, each with a time complexity as $O(nlogn)$ and $O(n)$.

From this, the time complexity is **<u>$O(n)$ when the value list is sorted and $O(nlogn)$ when the value list is not.</u>**

Question 5
N = 8

First solution: 1s are queens
1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0

Question 5: Algorithm

1. If the row value = dimension of chessboard, then display the board
2. Iterate through rows and columns in the chessboard
3. Check if no other queen is on the same row, column, or diagonal
4. If there are no queens, then place a queen there and repeat from step 1 with the next row value, then remove the queen to complete the backtracking process
5. If there are queens in the way, continue with step 1's next iteration

Question 5: Time Complexity

The time complexity of the n-queens problem is the time complexity of the row and column iterations coupled with the recursive function. Compounding these, the total time complexity is $\underline{\boldsymbol{O(n^2 n!)}}$