CSE 551: Foundations of Algorithms

Assignment 3 Solutions

1. The optimal order of multiplying $A1 * A2 * A3 * A4$ is $((A1 * (A2 * A3)) * A4)$. The fewest number of multiplications needed is 2856.

   Given array = [13, 5, 89, 3, 34]

   The steps of computation:

   $M[1, 2] = 13 * 5 * 89 = 5785$
   $M[2, 3] = 5 * 89 * 3 = 1335$
   $M[3, 4] = 89 * 3 * 34 = 9078$

   $M[1, 3] = \min\{$

   $M[2, 3] + 13 * 5 * 3 = 1530,$
   $M[1, 2] + 13 * 89 * 3 = 9256$

   $\} = 1530.$

   $M[2, 4] = \min\{$
   $M[3, 4] + 5 * 89 * 34 = 24208,$
   $M[2, 3] + 5 * 3 * 34 = 1845$
   $\} = 1845.$

   $M[1, 4] = \min\{$
   $M[2,4] + 13 \times 5 \times 34,$
   $M[1,2] + M[3,4] + 13 \times 89 \times 34,$
   $M[1,3] + 13 \times 3 \times 34$
   $\} = 2856$

2. Longest Increasing Subsequences of the given strings are: BDAB, BCAB, BCBA. Atleast one of them needs to be found using the below matrix. (see lecture slides)

	Yj	B	D	C	A	B	A
Xi	0	0	0	0	0	0	0
A	0	0	0	0	1	1	1
B	0	1	1	1	1	2	2
C	0	1	1	2	2	2	2
B	0	1	1	2	2	3	3
D	0	1	2	2	2	3	3
A	0	1	2	2	3	3	4
B	0	1	2	2	3	4	4

3. We maintain three indices i, j, k for the given 3 strings S1, S2, and S3. We iterate from the end of the strings (string length) to the beginning (index 1) and check different cases when there is a match.

   IL(i, j, k) is a Boolean value indicating if S3[0..k-1] is an interleaving string of S1[0..i-1] and S2[0..j-1].

$$IL(i,j,k) = \begin{cases} S2[j-1] == S3[k-1] \ and \ IL(i,j-1,k-1) \ \ if \ i=0 \\ S1[i-1] == S3[k-1] \ and \ IL(i-1,j,k-1) \ \ if \ j=0 \\ \big(S2[j-1] == S3[k-1] \ and \ IN(i,j-1,k-1)\big) \ or \ \big(S1[i-1] == S3[k-1] \ and \ IN(i-1,j,k-1)\big) \\ \qquad otherwise \end{cases}$$

   The base cases:

   Return False if Len(S3) != Len(S1) + Len(S2).
   IN(i, j, k) = True if (i = 0 and j = 0)

   The final result would be IN(L1, L2, L3) where L1, L2, L3 are the lengths of strings S1, S2, S3. Note that at every point value of k = i+j so the above formulation can also be written by replacing k with i+j. So, we can use a 2D DP matrix. Here is the pseudocode.

```
boolean isInterleaving(S1, S2, S3):
        n1 = S1.length
        n2 = S2.length
        n3 = S3.length

        if n3 != n1 + n2:
                return false

        boolean[][]dp = new dp[n1+1][n2+1]

        for i = 0 to n1:
                for j = 0 to n2:
                        if i==0 and j==0:
                                dp[i][j] = true
                        else if i==0:
                                dp[i][j] = S2[j-1] == S3[i+j-1] and dp[i][j-1]
                        else if j==0:
                                dp[i][j] = S1[i-1] == S3[i+j-1] and dp[i-1][j]
                        else:
                                dp[i][j] = (S1[i-1] == S3[i+j-1] and dp[i-1][j]) or
                                           (S2[j-1] == S3[i+j-1] and dp[i][j-1])
                End for
        End for
        Return dp[n1][n2]
```

4. We need to find the longest palindromic subsequence of the given string. So, we maintain two indices i, j one from beginning of the string and other from end.

   LPS(i, j): Longest palindromic subsequence between indices i and j of the given string.

$$LPS(i,j) = \begin{cases} 1 \; if \; i == j \\ 2 + LPS(i+1, j-1) \; if \; str[i] = str[j] \\ Max\big(LPS(i+1,j), LPS(i,j-1)\big) \; otherwise \end{cases}$$

   If str[i] = str[j] we can add these two characters to our result and then find LPS in substring from i+1 to j-1.

   If str[i] != str[j] we need to ignore of the characters to construct the palindrome. So we check both substrings str(i+1, j), str(i, j-1) and then take the maximum value among them.

   Pseudocode:

```
LongestPalindromicString(str, i, j, dp):
        If i > j:
                return 0

        if i == j:
                return 1

        if dp[i][j] is not empty:
                return dp[i][j]

        if str[i] == str[j]:
                dp[i][j] = 2 + LongestPalindromicString(str, i+1, j-1, dp)
        else:
                dp[i][j] = max(
                        LongestPalindromicString(str, i+1, j, dp),
                        LongestPalindromicString(str, i, j-1, dp):
                )
        Return dp[i][j]


LongestPalindromicString(str):
         n = Length(str)
        dp[N][N] = {empty}

        return LongestPalindromicString(str, 0, n-1, dp)
```

Note: students may also use the bottom-up approach for Q4 and top-down approach for Q3. Indexes in the DP formulation might change if student assumes 1-indexing. Another approach for Q4 is to get the reverse of the given string and find the longest common subsequence between the string and its reverse.