# PROBLEM SET 5 – Manav Bilakhia


CSc 250, Spring 2022 | Aaron G. Cass
Due: Thursday, Week 7 | Department of Computer Science
Union College


**Note:** For each algorithm design question, you must **prove** that the algorithm works.

## ALGORITHM DESIGN AND ANALYSIS

1. **Longest Common Subsequence.** [From [1] §16.3] A *subsequence* of a given sequence is the given sequence with some (possibly zero) elements removed. In other words, if $X = \langle x_1, x_2, ..., x_m \rangle$ is a sequence and $Z = \langle z_1, z_2, ..., z_k \rangle$ is another sequence, $Z$ is a subsequence of $X$ if there exists a strictly increasing sequence $\langle i_1, i_2, ..., i_k \rangle$ of indices of $X$ such that for all $j = 1, 2, ..., k$, we have $x_{i_j} = z_j$. For example, $Z = \langle B, C, D, B \rangle$ is a subsequence of $X = \langle A, B, C, B, D, A, B \rangle$ with index sequence $\langle 2, 3, 5, 7 \rangle$.

   Given two sequences $X$ and $Y$, $Z$ is a *common subsequence* of $X$ and $Y$ if $Z$ is a subsequence of both $X$ and $Y$. A *longest common subsequence* (LCS) of $X$ and $Y$ is a common subsequence with length at least as large as any other common subsequence of $X$ and $Y$.

   **The goal, then**, of the LCS problem is to find an LCS of given sequences $X = \langle x_1, x_2, ..., x_m \rangle$ and $Y = \langle y_1, y_2, ..., y_n \rangle$.

   (a) Prove that the LCS problem has optimal substructure. In other words, given $X = \langle x_1, x_2, ..., x_m \rangle$ and $Y = \langle y_1, y_2, ..., y_n \rangle$, and assuming $Z = \langle z_1, z_2, ..., z_k \rangle$ is an LCS of $X$ and $Y$, prove each of the following claims:

      i. If $x_m = y_n$, then $z_k = x_m = y_n$ and $Z_{k-1}$ is an LCS of $X_{m-1}$ and $Y_{n-1}$. Note that $Q_p$, where $Q = \langle q_1, q_2, ..., q_r \rangle$ is a sequence and $p$ is an index, is defined as the sequence $Q_p = \langle q_1, q_2, ..., q_p \rangle$.

      ii. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that $Z$ is an LCS of $X_{m-1}$ and $Y$.

      iii. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that $Z$ is an LCS of $X$ and $Y_{n-1}$.

   (b) Using the fact that the LCS problem has optimal substructure as shown above, give a dynamic programming algorithm for solving LCS with a worst-case running time in $\Theta(mn)$.

   (c) Prove that the running time is in $\Theta(mn)$.

   (d) Determine the space efficiency class of the algorithm.

   (e) Describe how to improve the space efficiency of your algorithm if the goal is only to determine the length of the longest common subsequence instead of determining an actual LCS.

**Answer:**

[From [1] §16.3]

(a) *Proof.* Case 1: By contradiction
Let us assume that $z_k \neq x_m$. Therefore, We can append $x_m = y_n$ to Z, obtaining a common subsequence of X and Y. Now we know we have $x_m = y_n = z_k$. We wish to show that the prefix $Z_{k-1}$ is a length-$(k-1)$ common subsequence of $X_{m-1}$ and $Y_{n-1}$. For contradiction, let us assume there exists a common subsequence F of $X_{m-1}$ and $Y_{n-1}$ that has a length greater than $k-1$. If we append $x_m = y_n$ to F then the length of the subsequence will be greater than $k-1$ which is a contradiction.

Case 2: By contradiction
Let us assume that $z_k \neq x_m$. Therefore we know that Z is a common subsequence of $X_{m-1}$ and

$Y$. Let us assume there is a common subsequence F of $X_{m-1}$ and $Y$ which has length greater than k. hence F would be also a subsequence of $X$ and $Y$ which is a contradiction.
Case 3: This case is symmetric to case 2.

□

(b) LCS($A, B$)
Input: Takes in 2 arrays A and B
Output: returns tables c and d

```
m ← A.length
n ← B.length
c[1...m,1....n], d[0...m][0...n]
for i ← 1 to m do
    d[i,0] ← 0
for j ← 0 to n do
    d[0,j] ← 0
for i ← 1 to m do
    for j ← 1 to n do
        if X[i] = Y[J] then
            d[i,j] ← c[i − 1, j − 1] + 1
            c[i,j] ← "go diagonally left up"
        else if d[i − 1, j] ≥ d[i, j − 1] then
            d[i,j] ← c[i − 1, j]
            c[i,j] ← "go up"
        else
            d[i,j] ← c[i, j − 1]
            c[i,j] ← "go left"
return c and d
```

PRINT($c, X, length1, length2$)
Input: takes in the table c, array X length of the first array and length of the second array
Output: returns tables c and d

```
if length1 = length2 = 0 then
    return
else if c[length1,length2]="go diagonally left up" then
    Print(c, X, length1-1 length2-1)
    print (X[length1])
else if c[length1, length2]="go up" then
    Print(c, X, length1-1, length2)
else
    Print(c, X, length1, length2-1)
```

(c) Each table entry in the LCS function take $\Theta(1)$ time to compute hence the overall function takes $\Theta(mn)$. The print function takes $\Theta(m + n)$ which is smaller than $\Theta(mn)$.

(d) This algorithm is using two tables each one of size $m \times n$. Therefore the space efficiency of class of this algorithm is $\Theta(mn)$.

(e) If we were to only determine the length of the LCS, we could use a 1-D Array just like the change maker problem where instead of counting the numbers of coins then we could count the length of the longest common subsequence.

2. **Matrix Chain Multiplication.** [From [2] §15.2] Recall that multiplying a $p \times q$ matrix by a $q \times r$ matrix produces a $p \times r$ matrix using $pqr$ multiplications (assuming brute-force matrix multiplication).

In general multiplying matrices $(A_1 \cdot A_2) \cdot A_3$ will take a different number of multiplications than multiplying $A_1 \cdot (A_2 \cdot A_3)$, even though both ways produce the same result.

The Matrix Chain Multiplication problem is:

> **Given**: an array $P[0..n]$ that gives the dimensions of a chain $< A_1, A_2, ..., A_n >$ of $n$ matrices — each matrix $A_i$, for $i = 1, 2, ..., n$, has dimensions $p[i-1] \times p[i]$.
>
> **Return**: The *minimum* number of multiplications needed to compute $A_1 \cdot A_2 \cdots A_n$, assuming pairs of matrices are multiplied using the brute force algorithm. In other words, return the number of multiplications needed by the best *parenthesization* of the matrix chain.

(a) Show that the Matrix Chain Multiplication problem has optimal substructure by giving a recurrence that evaluates to the number of multiplications of the optimal parenthesization. You will need to determine what parameters the optimal substructure takes.

(b) Write a dynamic programming algorithm based on the optimal substructure you just demonstrated, and determine the running time of the algorithm.

**Answer:**

[From [2] §15.2]

(a) Recurrence that evaluates counting parenthesization:

$$P(n) = \begin{cases} 1, & n = 1 \\ \sum_{k=1}^{n-1} p(k)p(n-k), & n \geq 2 \end{cases}$$

Let m[i,j] be the cost of counting parenthesisation such that the most optimal solution is at m[1,n]. This gives us the optimal substructure:

$$m[i,j] = \begin{cases} 0, & i = j \\ min(m[i,k] + m[k+1,j] + p_{i-1}p_k p_j), & i \leq k < j \text{ where k runs from i to j-1} \end{cases}$$

Here, we already know that if i = j then the cost would be 0 as there is only one matrix. But when $i \leq j$, we know that in order the calculate m[i,j] fr a matrix product of $j - i + 1$ matrices only depends on the cost of calculating the matrix product fewer than those. Such that for all $k = i, i+1...., j-1$ we know that the matrix $A_{i...k}$ is a product of $ki + 1 < ji + 1$ matrices and the matrix $A_{k+1...j}$ is a product of $jk < ji + 1$ matrices. Therefore our algorithm which uses this optimal substructure should fill in the table $m$ which stores the cost of parenthesization problem on matrix chains of increasing length.

(b) MCM($p$)
Input: takes in an array p
Output: returns the most optimal cost of parenthesizing

```
n ← length[p] − 1
for i ← 0 to n do
    m[i,i] ← 0
for l ← 2 to n do
    for i ← 1 to n − l + 1 do
        j ← i + l − 1
        m[i,j] ← ∞
        for k ← i to j − 1 do
            q ← m[i,k] + m[k+1,j] + p_{i−1} p_k p_j
            if q¡m[i,j] then
                m[i,j] ← q
return m[1,n]
```

Each of the three nested loops runs at most n-1 times giving this algorithm a running time complexity of $\Theta(n^3)$.

## References


[1] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, second edition, 2001.


## Honor Code Affirmation

I affirm that I have carried out my academic endeavors with full academic honesty
Manav Bilakhia