

a) Longest Common Subsequence

A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous. In LCS, we have to find the Longest Common Subsequence that is in the same relative order.

In the give question, let the name of Group Member 1 is **Danish** and Group member 2 is **Salman**, so:

X: {B, D, C, A, B, A} and Y: {A, B, C, B, D, A, B}

		j	0	1	2	3	4	5	6
				B	D	C	A	B	A
i	x_i	y_j							
0			0	0	0	0	0	0	0
1	A		0	↑	↑	↑	↖	←	↖
2	B		0	↖	←	←	↑	↖	←
3	C		0	↑	↑	↖	←	↑	↑
4	B		0	↖	↑	↑	↑	↖	←
5	D		0	↑	↖	↑	↑	↑	↑
6	A		0	↑	↑	↑	↖	↑	↖
7	B		0	↖	↑	↑	↑	↖	↑

LCS-LENGTH(X, Y)

```

1  m = X.length
2  n = Y.length
3  let b[1..m, 1..n] and c[0..m, 0..n] be new tables
4  for i = 1 to m
5      c[i, 0] = 0
6  for j = 0 to n
7      c[0, j] = 0
8  for i = 1 to m
9      for j = 1 to n
10         if  $x_i == y_j$ 
11             c[i, j] = c[i - 1, j - 1] + 1
12             b[i, j] = "↖"
13         elseif c[i - 1, j] ≥ c[i, j - 1]
14             c[i, j] = c[i - 1, j]
15             b[i, j] = "↑"
16         else c[i, j] = c[i, j - 1]
17             b[i, j] = "←"
18  return c and b

```

PRINT-LCS(b, X, i, j)

```

1  if i == 0 or j == 0
2      return
3  if b[i, j] == "↖"
4      PRINT-LCS(b, X, i - 1, j - 1)
5      print  $x_i$ 
6  elseif b[i, j] == "↑"
7      PRINT-LCS(b, X, i - 1, j)
8  else PRINT-LCS(b, X, i, j - 1)

```

(b) Shortest-Common-Supersequence

Example Problem

S1 = "abdul"

S2 = "muhammad"

Now we first need to find the LCS of both strings by dynamic programming approach.

The image shows a handwritten dynamic programming table for finding the Longest Common Subsequence (LCS) between the strings S1 = "abdul" and S2 = "muhammad". The table has 9 rows and 7 columns. The columns are labeled with the characters of S1: a, b, d, u, l. The rows are labeled with the characters of S2: m, u, h, a, m, m, a, d. The table contains numerical values representing the length of the LCS for each subproblem. Arrows indicate the path back to the source of the maximum value, showing the LCS is "ad".

		a	b	d	u	l
	0	0	0	0	0	0
m	1	0	0	0	0	0
u	2	0	0	0	1	1
h	3	0	0	0	1	1
a	4	0	1	1	1	1
m	5	0	1	1	1	1
m	6	0	1	1	1	1
a	7	0	1	1	1	1
d	8	0	1	1	2	2

So LCS is "ad"

Now for the shortest common supersequence we need to find the following;

1) Length of shortest common supersequence

Length of SCS = (length of S1 + length of S2) - Length of LCS

$$=(5+8) - 2$$

$$= 11$$

So in our example the length of SCS will be **11**

Find SCS :

Note:

1. Add LCS characters only once .
2. Assume first common character belongs to LCS character.
3. Add non LCS characters in order of either S1 then S2 or vice versa.

S1= a b d u l
↗ ↗ ↗ ↗ ↗

S2 = m u h a m m a d
↗ ↗ ↗ ↗ ↗ ↗ ↗ ↗

LCS = a d
↗ ↗

SCS = muhabmmadul

So “muhabmmadul” is the SCS in our example of length
11

Algorithm

```
int p1=0,p2=0;
for(char c: lcs)
    while(str1[p1]!=c) //Add all non-LCS chars from str1
        ans += str1[p1++];
    while(str2[p2]!=c) //Add all non-LCS chars from str2
        ans += str2[p2++];
    ans += c; //Add LCS-char and increment both ptrs
    ++p1
    ++p2
Return ans
```

c) Longest Increasing subsequence

Algo Assignment #3

Roll No of Member #1 = 18K-0264

Roll No of Member #2 = 18K-0350

Date: _____

Problem# longest increasing subsequence.

Q) 4 10 2 0 20

Algorithm used:-

$L(i) = 1 + \max(L(j))$ where $0 \leq j < i$ and $array[j] < array[i]$;
 $L(i) = 1$, if no j exists

Solution:- making new array of same size with all values 1

iteration #1(i) $i=1, j=0 \Rightarrow array[j] < array[i]$, add 1, $j++$,
 $i=i$, new iteration

1	1	1	1	1
---	---	---	---	---

iteration #2(i) $i=2, j=0 \Rightarrow array[j] > array[i]$, $j++$

iteration #2(ii) $i=2, j=1 \Rightarrow array[j] > array[i]$, $j++$, $i=i$, new iteration

iteration #3(i) $i=3, j=0 \Rightarrow array[j] > array[i]$, $j++$

iteration #3(ii) $i=3, j=1 \Rightarrow array[j] > array[i]$, $j++$

iteration #3(iii) $i=3, j=2 \Rightarrow array[j] > array[i]$, $j++$, $i=i$, new iteration

iteration #4(i) $i=4, j=0 \Rightarrow array[j] < array[i]$, add 1, $j++$

1	2	1	1	2
---	---	---	---	---

iteration #4(ii) $i=4, j=1 \Rightarrow array[j] < array[i]$, add 1 to i th position, $j++$

1	2	1	1	3
---	---	---	---	---

iteration #4(iii) $i=4, j=2 \Rightarrow array[j] < array[i]$, Pick from j th position, but we didn't add because already $j++$, max value present

iteration #4(iv) $i=4, j=3 \Rightarrow array[j] < array[i]$, we didn't add because max value already present, $j++$, new iteration

So longest common subsequence is = 3

d) Levenshtein-Distance

The minimum edit distance between two strings is the minimum number of editing operations:

- Insertion
- Deletion
- Substitution

Needed to transform one into the other.

Examples include Spell correction, computational biology etc.

Normally, each operation has cost of 1. However, in Levenshtein, the substitution cost can be taken as 2.

For two strings X of length n Y of length m. We define $D(i,j)$ the edit distance between $X[1..i]$ and $Y[1..j]$ i.e., the first i characters of X and the first j characters of Y. The edit distance between X and Y is thus $D(n,m)$.

Algorithm

```
//Initialization
D(i,0) = i
D(0,j) = j
//Recurrence Relation:
For each i = 1...n
    For each j = 1...m
        D(i,j)= min(
            D(i-1, j) + 1,
            D(i, j-1) + 1,
            D(i-1, j-1) + 2,  if X(i) ≠ Y(j)
            D(i-1, j-1) + 0   if X(i) = Y(j) )
//Termination:
D(n,m) is distance
```

Example: Transform "PLASMA" into "ALTRUISM"

The steps are

Step	Comparison	Edit necessary	Total Editing
1.	"" to ""	no edits necessary!	"" to "" in 0 edits
2.	"P" to "A"	replace: +1 edit	"P" to "A" in 1 edit
3.	"L" to "L"	no edits necessary!	"PL" to "AL" in 1 edit
4.	"A" to "T"	replace: +1 edit	"PLA" to "ALT" in 2 edits
5.	"A" to "R"	insert: +1 edit	"PLA" to "ALTR" in 3 edits
6.	"A" to "U"	insert: +1 edit	"PLA" to "ALTRU" in 4 edits
7.	"A" to "I"	insert: +1 edit	"PLA" to "ALTRUI" in 5 edits
8.	"S" to "S"	no edits necessary!	"PLAS" to "ALTRUIS" in 5 edits
9.	"M" to "M"	no edits necessary!	"PLASM" to "ALTRUISM" in 5 edits
10.	"A" to "M"	delete: +1 edit	"PLASMA" to "ALTRUISM" in 6 edits

The above steps are shown as the yellow blocks below with the solid red arrows pointing in the opposite direction from the process above:

	""	A	L	T	R	U	I	S	M
""	0	1	2	3	4	5	6	7	8
P	1	1	2	3	4	5	6	7	8
L	2	2	1	2	3	4	5	6	7
A	3	2	2	2	3	4	5	6	7
S	4	3	3	3	3	4	5	5	6
M	5	4	4	4	4	4	5	6	5
A	6	5	5	5	5	5	5	6	6

e) Matrix Chain Multiplication

Input: $p[] = \{40, 20, 30, 10, 30\}$

Output: 26000

There are 4 matrices of dimensions 40×20 , 20×30 , 30×10 and 10×30 .

Let the input 4 matrices be A, B, C and D. The minimum number of multiplications are obtained by putting parenthesis in following way
 $(A(BC))D \rightarrow 20 \times 30 \times 10 + 40 \times 20 \times 10 + 40 \times 10 \times 30$

Input: $p[] = \{10, 20, 30, 40, 30\}$

Output: 30000

There are 4 matrices of dimensions 10×20 , 20×30 , 30×40 and 40×30 .

Let the input 4 matrices be A, B, C and D. The minimum number of multiplications are obtained by putting parenthesis in following way
 $((AB)C)D \rightarrow 10 \times 20 \times 30 + 10 \times 30 \times 40 + 10 \times 40 \times 30$

Input: $p[] = \{10, 20, 30\}$

Output: 6000

There are only two matrices of dimensions 10×20 and 20×30 . So there is only one way to multiply the matrices, cost of which is $10 \times 20 \times 30$

```
#include <bits/stdc++.h>
using namespace std;

// Matrix Ai has dimension p[i-1] x p[i]
// for i = 1..n
int MatrixChainOrder(int p[], int i, int j)
{
    if (i == j)
        return 0;
    int k;
    int min = INT_MAX;
    int count;

    // place parenthesis at different places
    // between first and last matrix, recursively
    // calculate count of multiplications for
    // each parenthesis placement and return the
    // minimum count
    for (k = i; k < j; k++)
    {
        count = MatrixChainOrder(p, i, k)
                + MatrixChainOrder(p, k + 1, j)
                + p[i - 1] * p[k] * p[j];

        if (count < min)
            min = count;
    }
}
```

```
        // Return minimum count
        return min;
    }

    // Driver Code
    int main()
    {
        int arr[] = { 1, 2, 3, 4, 3 };
        int n = sizeof(arr) / sizeof(arr[0]);

        cout << "Minimum number of multiplications is "
              << MatrixChainOrder(arr, 1, n - 1);
    }
```


f) Knapsack Problem

Given a set 'S' of 'n' items,

such that each item i has a positive value v_i and positive weight w_i

The goal is to find maximum-benefit subset that does not exceed the given weight W .

Algorithm

```

Knapsack(j,w)
  for i ← 0 to n
    M[i,0] ← 0
  for w ← 0 to W
    M[0,w] ← 0

  for j ← 1 to n
    for w ← 0 to W
      if  $w_j > w$ 
        M[j,w] = M[j - 1, w]
      else M[j,w] ← MAX( $v_j + M[j - 1, w - w_j]$ ,
        M[j - 1, w])
  return M[n,W]
```

Problem Question:

Available weights

0 kg

1 kg

2 kg

3 kg

4 kg

5 kg

Values

\$ 0

\$ 3

\$ 5

\$ 4

\$ 8

\$ 10

Indices

0

1

2

3

4

5

5 kg

Capacity of sack

Sack capacity

→

Weights Available

0 kg

1 kg

2 kg

3 kg

4 kg

5 kg

(0 kg)

0

\$ 0

\$ 0

\$ 0

\$ 0

\$ 0

\$ 0

(0 kg, 1 kg)

1

\$ 0

\$ 3

\$ 3

\$ 3

\$ 3

\$ 3

(0 kg, 1 kg, 2 kg)

2

\$ 0

\$ 3

\$ 5

\$ 8

\$ 8

\$ 8

(0 kg, 1 kg, 2 kg, 3 kg)

3

\$ 0

\$ 3

\$ 5

\$ 8

\$ 8

\$ 9

(0 kg, 1 kg, 2 kg, 3 kg, 4 kg)

4

\$ 0

\$ 3

\$ 5

\$ 8

\$ 8

\$ 11

(0 kg, 1 kg, 2 kg, 3 kg, 4 kg, 5 kg)

5

\$ 0

\$ 3

\$ 5

\$ 8

\$ 8

\$ 11

Base Case Values

Base Case Values

The maximum benefit/value = 11

The selected items' indexes are: $s_i = [1,0,0,1,0]$

g) Partition Problem

Given a set of positive integers, check if it can be divided into two subsets with equal sum.

First. The sum of all elements in the set is calculated. If sum is odd, we can't divide the array into two sets. If sum is even, check if a subset with $\text{sum}/2$ exists or not.

Let the first three alphabets converted to numbers:

Group member 1: Abid 1,2,9

Group member 2: Adam 1,4,1

Set= 1,2,9,1,4,1

Sum=18 (Since the sum is even, so partition might be possible)

$\text{Sum}/2=9$

We will make table of size 9, and if the last element results in True, then the partition is possible, otherwise not.

This somewhat resembles to the 0/1 knapsack problem.

	0	1	2	3	4	5	6	7	8	9
0	T	F	F	F	F	F	F	F	F	F
1	T	T	F	F	F	F	F	F	F	F
2	T	T	T	F	F	F	F	F	F	F
9	T	T	T	F	F	F	F	F	F	T
1	T	T	T	T	F	F	F	F	F	T
4	T	T	T	T	T	T	T	T	T	T
1	T	T	T	T	T	T	T	T	T	T

The last box of the table is true, that shows that the partition is possible.

The result showed that set can be partitioned into $S1 = \{9\}$, and $S2 = \{1,2,1,4,1\}$, with each subset having a sum of 9. However, note that the size of sets are not equal.

h) Rod Cutting Problem

Rod cutting problem is a type of allocation problem. Allocation problem involves the distribution of resources among the competing alternatives in order to minimize the total costs or maximize total return (profit).

In the rod cutting problem we have given a rod of length n , and an array that contains the prices of all the pieces smaller than n , determine the maximum profit you could obtain from cutting up the rod and selling its pieces.

Length[] = {1,2,3,4,5,6,7,8}

price[] = {1,5,8,9,10,16,18,20}

Rod Length: 8

Price (i)	Length (i)	0	1	2	3	4	5	6	7	8
1	1	0	1	2	3	4	5	6	7	8
5	2	0	1	5	6	10	11	15	16	20
8	3	0	1	5	8	10	13	16	17	21
6	4	0	1	5	8	10	13	16	17	21
10	5	0	1	5	8	10	13	16	17	21
16	6	0	1	5	8	10	13	17	17	21
18	7	0	1	5	8	10	13	17	18	19
20	8	0	1	5	8	10	13	17	18	20

Maximum Profit = 20

Selected Pieces = 8 pieces of Rod Length 2, to get maximum profit for rod length 20.

