Dynamic Programming

– Saurabh Verma

Agenda

- What is DP?
- Memoizarion & Tabulation with Example
- 0/1 Knapsack Problem
- Subset Sum Problem
- Equal Sum Partition Problem
- Count of Subsets with Sum equal to X
- Unbounded Knapsack Problem
- Coin Change Problem 1
- Coin Change Problem 2
- Rod Cutting Problem
- Maximum Ribbon Cut Problem
- Longest Common Subsequences
- Longet Common Substring
- Minimum Insertion or Deletion to change A into B
- Minimum Insertion or Deletion to make Palindrome
- Longest Plindromic Subseuence

What is DP?

- Dynamic Programming (DP) is an algorithmic technique for solving an optimization problem by breaking it down into simpler subproblems and utilizing the fact that the optimal solution to the overall problem depends upon the optimal solution to its subproblems.

What is dp? (contd..)

- Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming.
- The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.

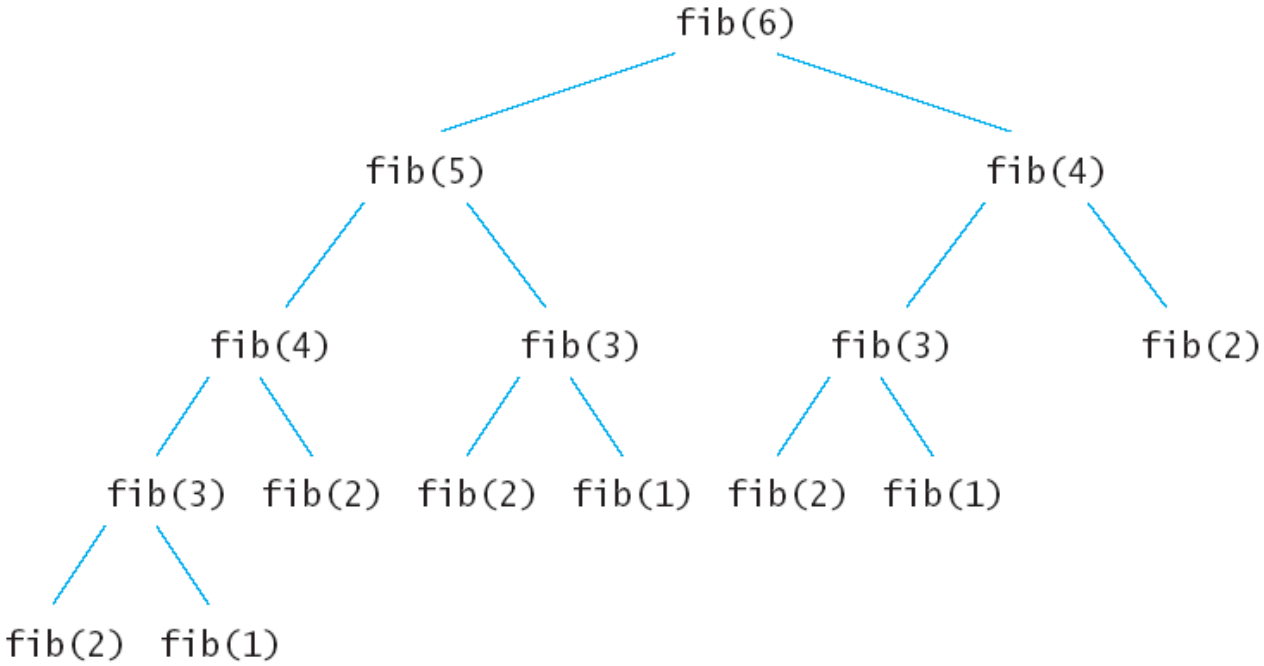
Example: N^{th} Fibonacci Number

- 0, 1, 1, 2, 3, 5, 8, 13, 21,..

```
int fib(int n){
    if(n<2)
        return n;
    return fib(n-1) + fib(n-2);
}
```

Time Complexity : $O(2^n)$

Example: Nth Fibonacci Number (Contd..)



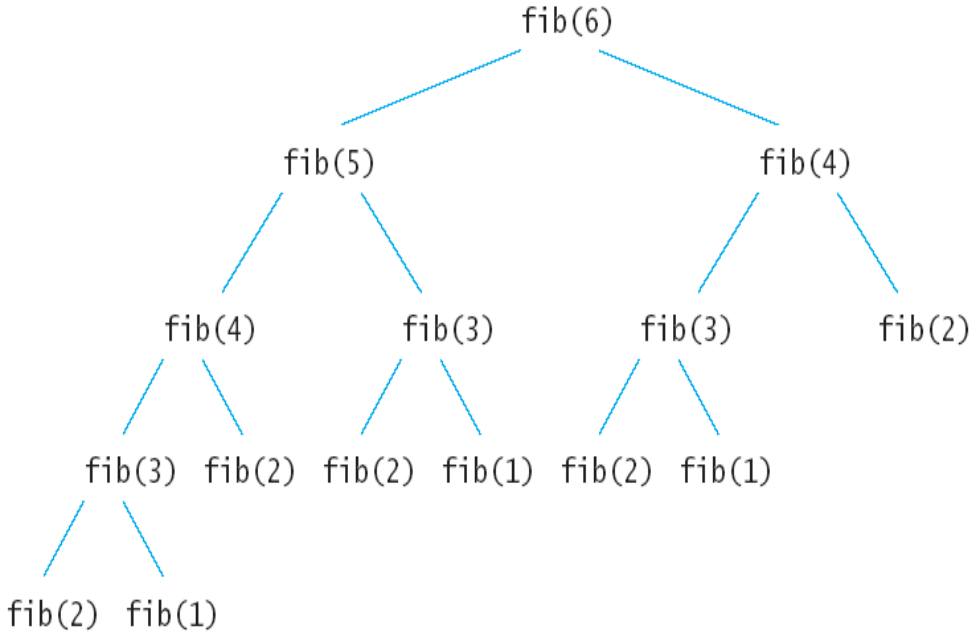
Example: Nth Fibonacci Number (Contd..)

```
static int fib(int n){
    if(n<2)
        return n;
    if(a[n] != -1)
        return a[n];
    a[n] = fib(n-1) + fib(n-2);
    return a[n];
}
```

Time Complexity : O(n)

Memoization Technique

Example: Nth Fibonacci Number (Contd..)



0	1	2	3	4	5	6
0	1	-1	-1	-1	-1	-1

Example: Nth Fibonacci Number (Contd..)

```java
static int fibTabulation(int n) {
    if (n==0) return 0;
    int dp[] = new int[n+1];

    //base cases
    dp[0] = 0;
    dp[1] = 1;

    for(int i=2; i<=n; i++)
      dp[i] = dp[i-1] + dp[i-2];

    return dp[n];
  }
```

Time Complexity : O(n)

Tabulation Technique

Memoization V/S Tabulation

```
static int fib(int n){
    if(n<2)
        return n;
    if(a[n] != -1)
        return a[n];
    a[n] = fib(n-1) + fib(n-2);
    return a[n];
}
```

Time Complexity : O(n)

```
static int fibTabulation(int n) {
    if (n==0) return 0;
    int dp[] = new int[n+1];

    //base cases
    dp[0] = 0;
    dp[1] = 1;

    for(int i=2; i<=n; i++)
      dp[i] = dp[i-1] + dp[i-2];

    return dp[n];
  }
```

01 Knapsack Problem

Problem Statement: Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

Input: wt[], val[], n, W

Output: Max Profit

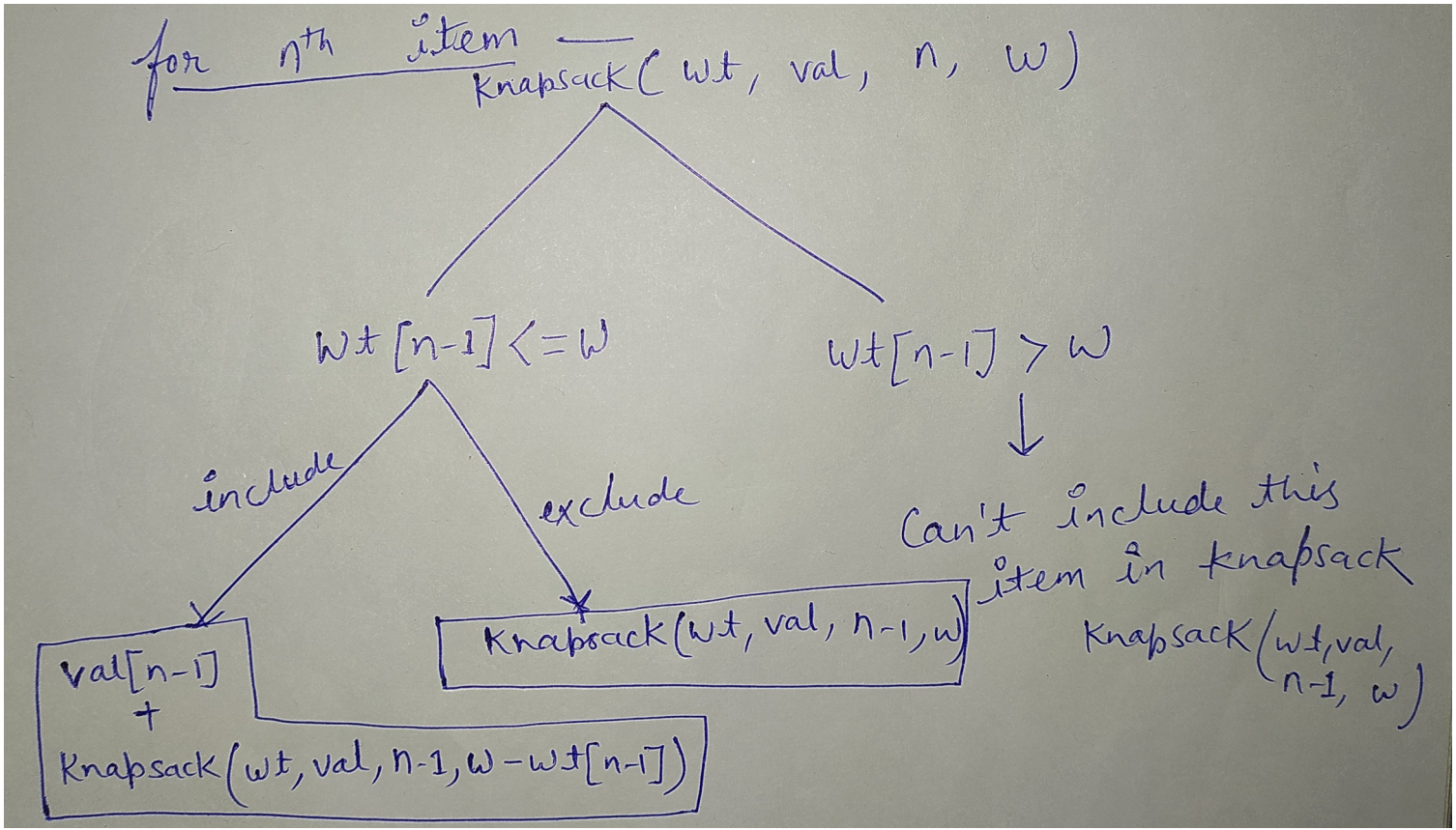
Example

value[] = {60, 100, 120};
weight[] = {10, 20, 30};
W = 50;

Solution: 220

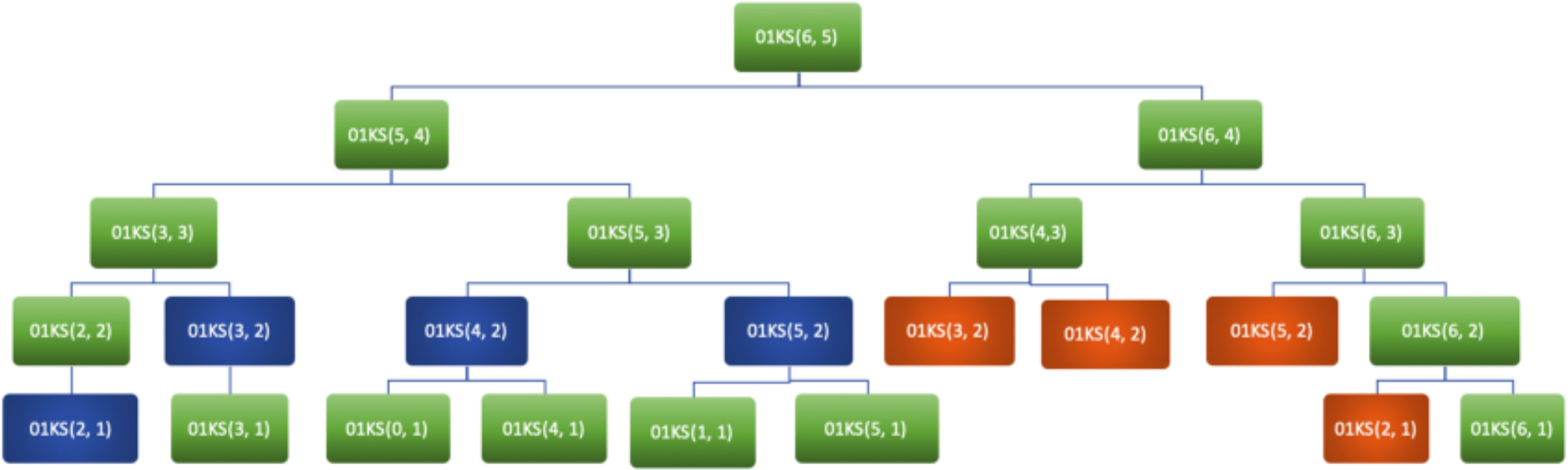
Weight = 10; Value = 60;
Weight = 20; Value = 100;
Weight = 30; Value = 120;
Weight = (20+10); Value = (100+60);
Weight = (30+10); Value = (120+60);
Weight = (30+20); Value = (120+100);
Weight = (30+20+10) > 50

Brute Force Approach(Recursive Soln)



Knapsack Capacity = 6

Objects	B1	B2	B3	B4	B5
Profit	25	125	15	68	19
Weight	2	4	1	2	1



2d Matrix of Memoization Analysis

Capacity of knapsack (w) : 10

	0	1	2	3
wt[]	1	3	4	5
val[]	1	4	5	7

	0	1	2	3	4	5	6	7	8	9	10
0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	1	1	1	-1	1	1	1	-1	-1	1
2	-1	1	-1	-1	-1	5	5	-1	-1	-1	5
3	-1	-1	-1	-1	-1	6	-1	-1	-1	-1	10
4	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	13

Understand Optimal Subproblems

Capacity of knapsack (w) : 10

	0	1	2	3
wt[]	1	3	4	5
val[]	1	4	5	7

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1	1
2	0	1	1	4	5	5	5	5	5	5	5
3	0	1	1	4	5	6	6	9	10	10	10
4	0	1	1	4	5	7	8	9	11	12	13

Subset Sum Problem

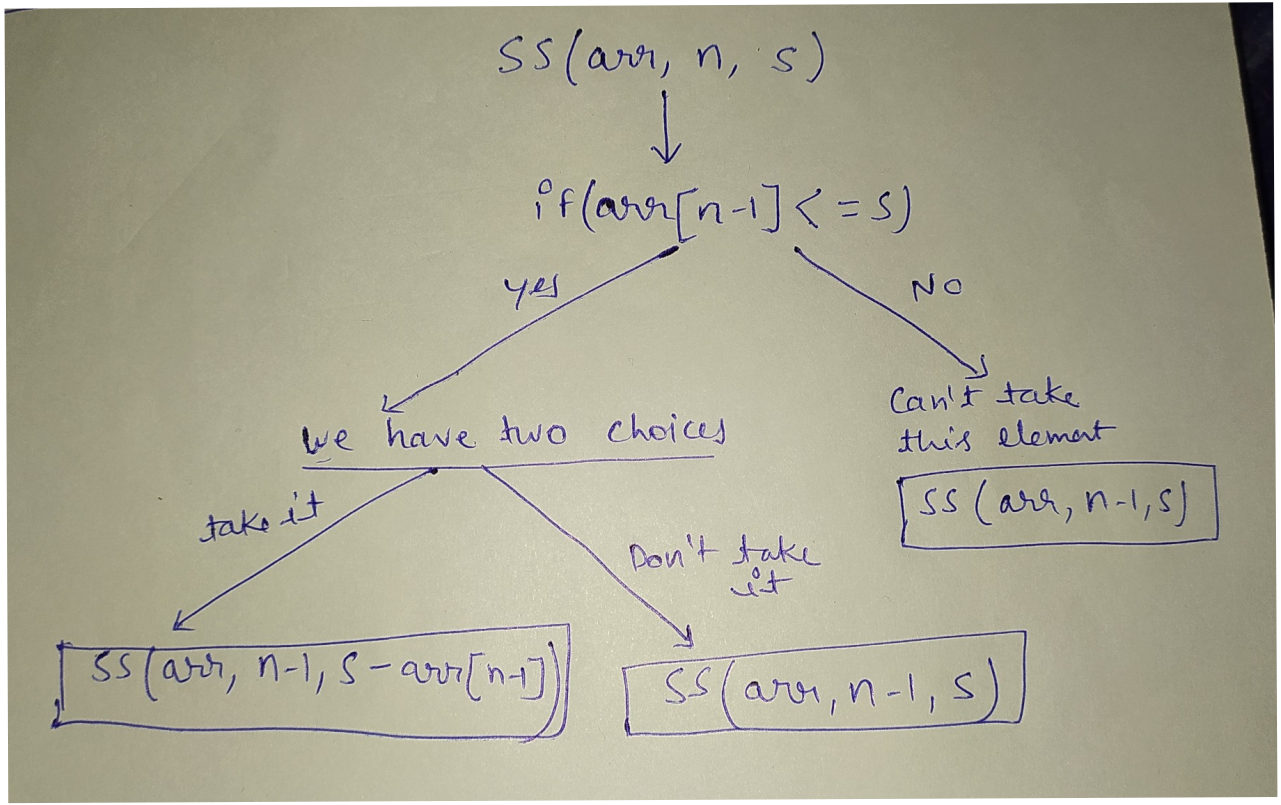
Given a set of non-negative integers, and a value *sum*, determine if there is a subset of the given set with sum equal to given *sum*.

Example:

```
Input: set[] = {3, 34, 4, 12, 5, 2}, sum = 9
Output: True
There is a subset (4, 5) with sum 9.

Input: set[] = {3, 34, 4, 12, 5, 2}, sum = 30
Output: False
There is no subset that add up to 30.
```

Choice Diagram



Equal Sum Partition Problem

Partition problem is to determine whether a given set can be partitioned into two subsets such that the sum of elements in both subsets is the same.

Examples:

```
arr[] = {1, 5, 11, 5}
Output: true
The array can be partitioned as {1, 5, 5} and {11}

arr[] = {1, 5, 3}
Output: false
The array cannot be partitioned into equal sum sets.
```

Count of subsets with sum equal to X

Given an array **arr[]** of length **N** and an integer **X**, the task is to find the number of subsets with sum equal to **X**.

Examples:

Input: *arr[] = {1, 2, 3, 3}, X = 6*
Output: *3*
All the possible subsets are {1, 2, 3},
{1, 2, 3} and {3, 3}

Input: *arr[] = {1, 1, 1, 1}, X = 1*
Output: *4*

Unbounded 0/1 Knapsack Problem

Problem Statement:

Given a knapsack weight **W** and a set of **n** items with certain value $\mathbf{val}_i$ and weight $\mathbf{wt}_i$, we need to calculate the maximum amount that could make up this quantity exactly. This is different from classical Knapsack problem, here we are **allowed to use unlimited number of instances** of an item.

```
Input : W = 8
        val[] = {10, 40, 50, 70}
        wt[]  = {1, 3, 4, 5}
Output : 110
We get maximum value with one unit of
weight 5 and one unit of weight 3.
```

Rod Cutting Problem

Problem Statement:

Given a rod of length n inches and an array of prices that contains prices of all pieces of size smaller than n. Determine the maximum value obtainable by cutting up the rod and selling the pieces. Example, let say N=8 and length[] and price[] are given below-

Length	1	2	3	4	5	6	7	8
Price	1	5	8	9	10	17	17	20

Max Profit : 22 (1 piece of 2, 1 peice of 6)

Coin Change Problem (max. no. of ways)

Given a value N, if we want to make change for N cents, and we have infinite supply of each of S = { S1, S2, .. , Sm} valued coins, how many ways can we make the change? The order of coins doesn't matter.

For example, for N = 4 and S = {1,2,3}, there are four solutions: {1,1,1,1},{1,1,2},{2,2}, {1,3}. So output should be 4. For N = 10 and S = {2, 5, 3, 6}, there are five solutions: {2,2,2,2,2}, {2,2,3,3}, {2,2,6}, {2,3,5} and {5,5}. So the output should be 5.

Longest Common Subsequence (LCS)

Problem Statement: Given two sequences, find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.

Input: A = “AGGTAB” , B = “GXTXAYB”
Output:
4
Explanation:
longest subsequence is GTAB.

	0	a	b	c	d	e	f
0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1
f	0	1 (2, 1)	1 (2,2)				
c	0			2			
h	0						
d	0						
k	0						
e	0						result

Longest Common Substring

Problem Statement: Given two strings X and Y, find the length of the longest common substring.

Input : X = “GeeksforGeeks”, Y = “GeeksQuiz”
Output : 5
Explanation: The longest common substring is “Geeks” and is of length 5.

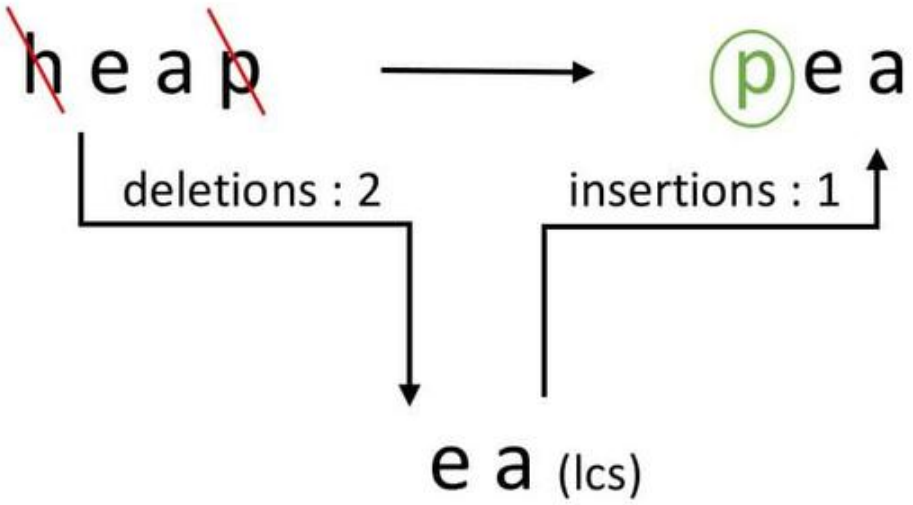
Input : X = “abcdxyz”, Y = “xyzabcd”
Output : 4
Explanation: The longest common substring is “abcd” and is of length 4.

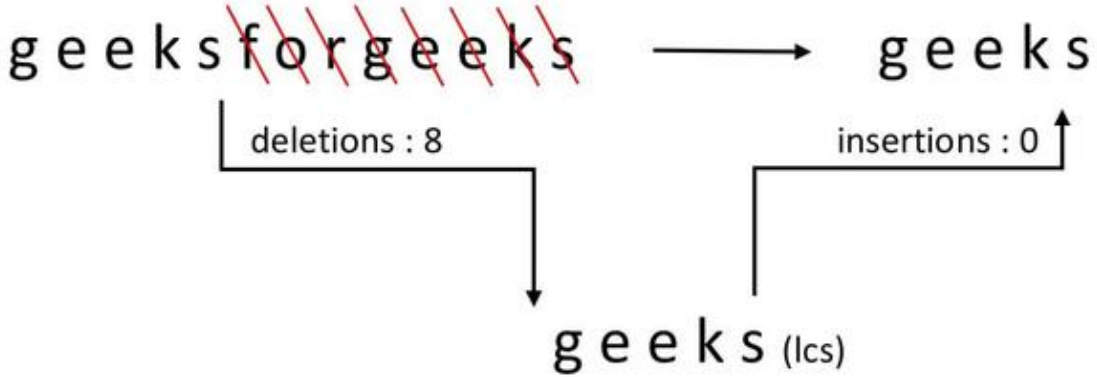
Minimum number of deletions and insertions to transform one string into another

Problem Statement: Given two strings X and Y of size m and n respectively. The task is to remove/delete and insert the minimum number of characters from/in X to transform it into Y.

Input: X= “heap” , Y = “pea”
Output: Deletion: 2, Insertion: 1

Explanation:
p and **h** deleted from heap
Then, **p** is inserted at the beginning.


h e a p
p e a
deletions : 2
insertions : 1
e a (lcs)


g e e k s f o r g e e k s
g e e k s
deletions : 8
insertions : 0
g e e k s (lcs)

Longest Palindromic Subsequence

Problem Statement:
Given a sequence, find the length of the longest palindromic subsequence in it.

Input: geekforgeeks
Output: 5

Explanation: longest palindromic subsequences are **“eekee”, “eesee”, “eefee”**

lcs(x, reverse(x))

Minimum number of deletions to make a string palindrome

Problem Statement:
Given a string of size ‘n’. The task is to remove or delete the minimum number of characters from the string so that the resultant string is a palindrome.

Input : aebcbda
Output : 2
Remove characters 'e' and 'd'
Resultant string will be **'abcba'**
which is a palindromic string