UNIT-2

What is Dynamic programming?

- Dynamic programming is a programming paradigm and algorithmic technique used to solve complex problems by breaking them down into simpler, overlapping subproblems.

- It is particularly useful for optimization problems, where the goal is to find the best solution among a set of feasible solutions**.**

Key Idea:

- The key idea behind dynamic programming is to solve each subproblem only once and store the results (usually in an array or table) to avoid redundant work when the same subproblem is encountered again.

- This approach significantly improves the efficiency and speed of solving the overall problem.

Here are the 4 basic steps involved in solving a problem using dynamic programming:

Characterize the structure of an optimal solution:

- Define the problem in terms of smaller subproblems**.**

Define the value of an optimal solution recursively:

- Express the value of an optimal solution in terms of the values of smaller subproblems.

Compute the value of an optimal solution in a bottom-up fashion:

- Use an iterative approach, solving subproblems in a bottom-up manner and storing the results in a table.

Construct an optimal solution:

- Build the optimal solution from the computed information in a bottom-up or top-down manner.

Top-Down Approach (Memoization):

Start from the top and break the problem into smaller subproblems. Recursively solve these subproblems and store their results in a memoization table to avoid redundant calculations.

Bottom-Up Approach (Tabulation):

Start from the bottom (smallest subproblems) and iteratively solve larger subproblems by using the results of smaller subproblems. Store the results in a table (usually an array) and build up to the final solution.

Example:

Longest common subsequence:

Algorithm

```
LCS-LENGTH(X, Y)
m ← length[X]
n ← length[Y]
for i ← 1 to m
     do c[i, 0] ← 0
for j ← 0 to n
     do c[0, j] ← 0
for i ← 1 to m
     do for j ← 1 to n
            do if x_i = y_j
                  then c[i, j] ← c[i - 1, j - 1] + 1
                       b[i, j] ← "□"
                  else if c[i - 1, j] ≥ c[i, j - 1]
                          then c[i, j] ← c[i - 1, j]
                               b[i, j] ← "↑"
                          else c[i, j] ← c[i, j - 1]
                               b[i, j] ← ←
return c and b
```

Example:

Q. Let X=<A B C B D A B>, Y=<B D C A B A>. Apply LCS Algorithm and calculate the LCS Number and elements.

Solution:

	0	**1**	**2**	**3**	**4**	**5**	**6**
	Y_j	**B**	**D**	**C**	**A**	**B**	**A**
0X_i							
1A							
2B							
3C							
4B							
5D							
6A							
7B							

Time Complexity:

m is the length of the first input sequence X.

n is the length of the second input sequence Y.

The time complexity is O(m×n) because we need to fill up a 2D table of size m×n using a nested loop structure, where m represents the length of the first sequence and n represents the length of the second sequence.

Applications of LCS

Bioinformatics:

LCS is widely used in bioinformatics for sequence alignment and comparison of DNA, RNA, or protein sequences. It helps identify similarities between genetic sequences, aiding in evolutionary analysis, disease detection, and drug design.

Plagiarism Detection:

In academic and professional settings, LCS can be employed to detect plagiarism by finding similarities between submitted texts and existing documents. It helps educators and institutions ensure academic integrity.

Natural Language Processing (NLP):

LCS is used in NLP for tasks like text summarization, where the longest common subsequence represents the core content shared between multiple documents, aiding in the extraction of essential information.

Image Comparison:

When comparing images, especially in video processing or computer vision, LCS can be adapted to identify the longest common subsequence of image features, aiding in image recognition and similarity analysis.

Optical Character Recognition (OCR):

LCS is used in OCR systems to compare recognized text with the original image to improve accuracy by identifying the longest common subsequence.