Name of Faculty: Alka Gulati

Designation: Associate Professor

Department: MCA

Subject: Design and Analysis of Algorithm

Unit: IV

Topic: Dynamic Programming , LCS

Dynamic Programming

Dynamic Programming - It is a method for solving complex problems by breaking them down into simpler subproblems.

Dynamic Programming solves problems by combining the solutions of subproblems. Dynamic Programming is used to solve optimization problems. Dynamic Programming algorithm solves each sub-problem just once and then saves its answer in a table, thereby avoiding the work of re-computing the answer every time. This technique of storing value of subproblems is called memoization.

Two properties of Dynamic programming are :-

(i) Overlapping sub-problems – This property is used when solution of one sub problem is needed again and again. The computed solutions are stored in a table.

(ii) Optimal substructure – this property is used if the optimal solution of the given problem can be obtained using optimal solutions of its sub-problems.

Steps to be followed in dynamic programming

1. Define subproblems
2. Write down the recurrence that relates subproblems
3. Recognize and solve the base cases

Applications of Dynamic Programming :-

- Shortest path in graph
- Matrix Chain Multiplication
- Longest Common Subsequence
- Travelling Salesman Problem

Comparison between Dynamic Programming and Divide and conquer method

S.No.	Dynamic Programming	Divide & Conquer method
1)	Dynamic Programming works by storing the result of each sub problem in table for future use. It solves the sub problems once only.	Divide and Conquer works by dividing the problem into sub-problems, conquer ea.ch sub-problem recursively and combine these solutions.
2)	The sub-problems are not independent.	The sub-problems are independent of each other.

Longest Common Subsequence (LCS)

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. For example, "abc", "abg", "bdf", "aeg", '"acefg", ….etc are subsequences of "abcdefg".

Examples:

1) LCS for input Sequences "ABCDGH" and "AEDHR" is "ADH" of length 3.

2) LCS for input Sequences "AGGTAB" and "GXTAB" is "GTAB"

of length 4.

3) LCS for input Sequences "LONGEST" and "PRONE" is "ONE" of length 3.

The solution for this problem is to generate all subsequences of both given sequences and find the longest matching subsequence.

Let two sequences be defined as follows: $X = (x_1, x_2, ..., x_m)$ and $Y = (y_1, y_2, ..., y_n)$, Function is defined as follows:-

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 , \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j , \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j . \end{cases}$$

Algorithm LCS-LENGTH takes two sequences $X = (x_1, x_2, ..., x_m)$ and $Y = (y_1, y_2, ..., y_n)$ as inputs. It stores the $c[i, j]$ values in a table $c[0 .. m, 0 .. n]$ whose entries are computed in row-major order. (That is, the first row of c is filled in from left to right, then the second row, and so on.) It also maintains the table $b[1 .. m, 1 .. n]$ to simplify construction of an optimal solution. Intuitively, $b[i, j]$ points to the table entry corresponding to the optimal subproblem solution chosen when computing $c[i, j]$. The procedure returns the b and c tables; $c[m, n]$ contains the length of an LCS of X and Y.

Algorithm LCS-LENGTH(X, Y)

{

$m \leftarrow length[X]$

$n \leftarrow length[Y]$

for $i \leftarrow 1$ **to** m

do $c[i, 0] \leftarrow 0$

for $j \leftarrow 0$ **to** n

do $c[0, j] \leftarrow 0$

for $i \leftarrow 1$ **to** m

do for $j \leftarrow 1$ **to** n

do if $x_i = y_j$

then $c[i, j] \leftarrow c[i - 1, j - 1] + 1$

$b[i, j] \leftarrow$ "↖"

else if $c[i - 1, j] \geq c[i, j - 1]$

then $c[i, j] \leftarrow c[i - 1, j]$

$b[i, j] \leftarrow$ "↑"

else $c[i, j] \leftarrow c[i, j - 1]$

$b[i, j] \leftarrow$" ←"

return c and b

}

Complexity : O(mn)

Example :

Let the sequences be

X = (*A*, *B*, *C*, *B*, *D*, *A*, *)* and Y =(*B*, *D*, *C*, *A*, *B*, *A)*. Find the longest common subsequence using dynamic programming.

Sol. Using memoization find the LCS.

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

The square in row *i* and column *j* contains the value of *c*[*i*, *j*] and the appropriate arrow for the value of *b*[*i*, *j*]. The entry 4 in *c*[7, 6]-the lower right-hand corner of the table-is the length of an LCS *(B,C,B,A)* of *X* and *Y* . For *i*, *j* > 0, entry *c*[*i*, *j*] depends only on whether $x_i = y_j$ and the values in entries *c*[*i* - 1, *j*], *c*[*i*, *j* - 1], and *c*[*i* - 1, *j* - 1], which are computed before *c*[*i*, *j*]. To reconstruct the elements of an LCS, follow the *b*[*i*, *j*] arrows from the lower right-hand corner; the path is shaded. Each "↖" on the path corresponds to an entry (highlighted) for which $x_i = y_j$ is a member of an LCS.

Ans : The subsequence is : BCBA and length is 4

Assignments

1) Determine an LCS of (1, 0, 0, 1, 0, 1, 0, 1) and (0, 1, 0, 1, 1, 0, 1,1) using dynamic Programming.

2) Determine an LCS of (A,B,C,D,B,A,C,D,F) and (C,A,B,A,F) using dynamic programming.

3) In which types of problem dynamic is better than divide and conquer? Give an example.