# Dynamic Programming

|  |  |  |
| --- | --- | --- |
| S. no. | Type of Question | # Alternatives |
| 1 | 0-1 Knapsack | 6 |
| 2 | Unbounded Knapsack | 5 |
| 3 | Fibonacci | 7 |
| 4 | Longest common subsequence | 15 |
| 5 | Longest increasing subsequence | 10 |
| 6 | Kadane’s Algorithm | 6 |
| 7 | Matrix chain multiplication | 7 |
| 8 | DP on trees | 4 |
| 9 | DP on grid | 14 |
| 10 | Others | 5 |
|  |  | **79** |

## 0-1 Knapsack

|  |  |
| --- | --- |
| S.no. | Question |
| 1 | Subset Sum |
| 2 | Equal sum partition |
| 3 | Count of subset sum |
| 4 | Minimum subset sum difference |
| 5 | Target sum |
| 6 | No of subset with given difference |

# Longest Common Subsequence

|  |  |
| --- | --- |
| S.no. | Question |
| 1 | Longest common substring |
| 2 | Print LCS |
| 3 | Shortest common subsequence |
| 4 | Print SCS |
| 5 | Min no of insertion and deletion a -> b |
| 6 | Longest repeating subsequence |
| 7 | Length of largest subsequence of ‘a’ which is a substring in ‘b’ |
| 8 | Subsequence pattern matching |
| 9 | Count how many times ‘a’ appear as subsequence in ‘b’ |
| 10 | Longest palindromic subsequence |
| 11 | Longest palindromic substring |
| 12 | Count of palindromic substring |
| 13 | Min no of deletion in a string to make it a palindrome |
| 14 | Min no of insertion in a string to make it a palindrome |

## Matrix Chain Multiplication

|  |  |
| --- | --- |
| S.no. | Question |
| 1 | Matrix chain Multiplication |
| 2 | Printing MCM |
| 3 | Evaluate expression to true |
| 4 | Min/Max value of expression |
| 5 | Palindrome Partitioning |
| 6 | Scramble string |
| 7 | Egg dropping Problem |

## Trees

|  |  |
| --- | --- |
| S.no. | Question |
| 1 | Diameter of Binary Tree |
| 2 | Max path sum from any node to any node |
| 3 | Max path sum from leaf node to leaf node |
| 4 | Diameter of N array tree |

3 things to see while thinking recursion solution:

1. Smaller input
2. Base case
3. Choice diagram

How to check if a question is of DP or not?

1. Choice – options to choose from
2. Optimal -> Max, Min, Longest, shortest , etc.

Always First think of recursive solution

Then, convert that recursive solution to top down

How to check if 2 questions are similar. Use **Pattern Matching Algo:**

1. Compare I/P, Ques, O/P
2. If u get 2/3 or 3/3, then same ques.

Memoization -> Recursion + matrix

Top - Down -> Matrix only

TO Check->

1. Min no of Insert, delete and replace to make string a-> string b
2. Print shortest common super sequence

**Note for Q 32 –**

1) If the node has no children (it is a leaf node) you simply return the node's data

2) If the node has any one child you return the sum of the node's data with the child's maximum path sum (l or r)

def maxPathSum(root, result):

if root == None:

return 0

if root.left == None and root.right == None:

return root.data

l = maxPathSum(root.left, result)

r = maxPathSum(root.right, result)

if root.right == None:

return l + root.data

if root.left == None:

return r + root.data

tempans = root.data + max(l, r)

ans = root.data + l + r

if ans > result[0]:

result[0] = ans

return tempans

3) We have one last edge case to take care of. If the tree is completely skewed (Every node in the tree only has one child), our result will remain unchanged as we just return l + root.data or r + root.data in each recursive call without updating result. So we check if the result has changed after calling maxPathSum. If it hasn't changed (the tree is skewed), we print tempans, which was returned by maxPathSum instead of printing result variable.

class Solution:

def maxPathSum(self, root):

# code here

result = [-99999999999]

tempans = maxPathSum(root, result)

# if the tree is skewed

if result[0] == -99999999999:

return tempans

else:

return result[0]

Hope this helped :)