Note : There are 2 classes of DP problems :

[a]. Simple : At any step, finding the optimal solution requires an optimal solution to only one sub problem. eg. maximal sub-array problem.

[b.]. A more interesting problem : Finding an optimal solution requires solutions to multiple sub-problems. Eg. Rod cutting problem, longest increasing subsequence, box stacking, etc

1. **Max Subarray Sum(Kadane’s algo):** Given an array containing both negative and positive integers. Find the contiguous sub-array with maximum sum. Solve it using simple DP. Solve without using dp array.

**2. Subset sum problem** : Given an array, find the subset with a given sum.(try to do this using backtracking also.) Find the maximum subset with sum <= S.

3. **Balanced Partition.** You have a set of n integers each in the range 0 ... K. Partition these integers into two subsets such that both have equal sum. Now try to minimize |S1 - S2|, where S1 and S2 denote the sums of the elements in each of the two subsets. This is an application of subset sum problem. Code at - <http://algorithmsandme.in/2014/04/balanced-partition-problem/.> Learn solving subset sum problem using 1d dp by marking true for every array element a[i], the sums sum[a[i]+k]……k=0 to sum[n]-a[I] and sum[a[k]]=true .

**4.Longest Increasing Subsequence(LIS)** : Given sequence, find the length of the longest subsequence such that all elements of the subsequence are sorted in increasing order. Idea : this class of dp problem requires solving more than one sub-problem. Let dp(i) = lenght of longest subsequence ending with the element a[i]. The answer will be max os all such dp(i), 0<=I<n.

dp(i) = { 1 + Max ( dp(j) ) } where j < i and arr[j] < arr[i] and if there is no such j then dp(i) = 1

Ans : max(L(i)) where 0 < i < n.

Applications : Max chain length problem

(http://www.geeksforgeeks.org/dynamic-programming-set-20-maximum-length-chain-of-pairs/)

Box stacking problem, etc.

**3. Longest Common Subsequence(LCS)** : Given two sequences, find the length of longest subsequence present in both of them.

**4.Edit Distance** : Given two strings str1 and str2 and one of three operations(insert,delete, replace) that can performed on str1. Find minimum number of edits (operations) required to convert ‘str1’ into ‘str2’. All operations are of equal cost.

Approach : Let dp[i,j] be ans for str1 of length I and str2 of length j. Then recurrence relation can be written as :

dp[I,j] = dp[i-1,j-1] , str1(I)=str2(j)

= max { dp[I,j-1], dp[I-1,j], dp[I-1,j-1] } considering insert, delete, replace.

1. **LCS**(Longest Common Subsequence), **LIS**(Longest Increasing Subequence), **LPS**(Longest Palindromic Subsequence). Also printing the longest substring/subsequence(by backtracing the previous states in the dp array). Print all multiple such sequences. **Maximum subarray/subsequence sum**.

<http://www.geeksforgeeks.org/printing-longest-common-subsequence-set-2-printing/>.

Despite above approach, the below question gives tle :

<http://www.practice.geeksforgeeks.org/problem-page.php?pid=1448>

For longest common substring - <http://www.geeksforgeeks.org/longest-common-substring/>

**6**.. **Knapsack problem**(bounded,unbounded,0/1 knapsack with limits on weight, etc all variations) with repetition allowed/not allowed. Greedy if weight is binary/doesn't vary(see codecheff problem).

**7**. **Coin change problem** with infinite/limited supply of coins and when ordering matters and not matters case. Find minimum no. of coins to make a change and also all ways to make a change(note that ways doesn’t count the number of coins it simply searches for a path leading to the target sum….see 1d soln from 2d soln achieved by choosing coin and updating the sums beginning from the coin value and above till target amount).

**8**. **Interval-Scheduling Problem** and other variants(Based on Greedy and DP Approach).Do some problems.

9. Maximum subarray sum(using dp), max subsequence/subset sum with printing the subarray/subset. Subset sum problem(printing all subsets with a given sum http://www.edufyme.com/code/?id=45c48cce2e2d7fbdea1afc51c7c6ad26

**10**. **Matrix chain multiplication:** Given an array p[] which represents the chain of matrices such that the ith matrix Ai is of dimension p[i-1] x p[i]. We need to write a function MatrixChainOrder() that should return the minimum number of multiplications needed to multiply the chain. Application : rod-cutting problem, optimal palindrome partitioning.

**11**. **Box Stacking**. You are given a set of n types of rectangular 3-D boxes, where the i^th box has height h(i), width w(i) and depth d(i) (all real numbers). You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box. Of course, you can rotate a box so that any side functions as its base. It is also allowable to use multiple instances of the same type of box.

12.**Counting Boolean Parenthesizations.** You are given a boolean expression consisting of a string of the symbols 'true', 'false', 'and', 'or', and 'xor'. Count the number of ways to parenthesize the expression such that it will evaluate to true. For example, there are 2 ways to parenthesize 'true and false xor true' such that it evaluates to true.

13.**Optimal Strategy for a Game.** Consider a row of n coins of values v(1) ... v(n), where n is even. We play a game against an opponent by alternating turns. In each turn, a player selects either the first or last coin from the row, removes it from the row permanently, and receives the value of the coin. Determine the maximum possible amount of money we can definitely win if we move first.

14 **Building Bridges**. Consider a 2-D map with a horizontal river passing through its center. There are n cities on the southern bank with x-coordinates a(1) ... a(n) and n cities on the northern bank with x-coordinates b(1) ... b(n). You want to connect as many north-south pairs of cities as possible with bridges such that no two bridges cross. When connecting cities, you can only connect city i on the northern bank to city i on the southern bank. (Note: this problem was incorrectly stated on the paper copies of the handout given in recitation.)

**14. Google algo arccchive - <https://sites.google.com/site/indy256/algo>**