* ***Dice Throw***

Given n dice each with m faces, numbered from 1 to m, find the number of ways to get sum X. X is the summation of values on each face when all the dice are thrown.

(<http://www.geeksforgeeks.org/dice-throw-problem/>)

* ***Edit Distance***

In [computer science](https://en.wikipedia.org/wiki/Computer_science), **edit distance** is a way of quantifying how dissimilar two [strings](https://en.wikipedia.org/wiki/String_(computing)) (e.g., words) are to one another by counting the minimum number of operations required to transform one string into the other.

Edit distances find applications in [natural language processing](https://en.wikipedia.org/wiki/Natural_language_processing), where automatic [spelling correction](https://en.wikipedia.org/wiki/Spell_checker) can determine candidate corrections for a misspelled word by selecting words from a dictionary that have a low distance to the word in question. In [bioinformatics](https://en.wikipedia.org/wiki/Bioinformatics), it can be used to quantify the similarity of [DNA](https://en.wikipedia.org/wiki/DNA) sequences, which can be viewed as strings of the letters A, C, G and T. (<https://en.wikipedia.org/wiki/Edit_distance>)

* ***~~Longest Common Subsequence~~***

The **longest common subsequence** (**LCS**) **problem** is the problem of finding the longest [subsequence](https://en.wikipedia.org/wiki/Subsequence) common to all sequences in a set of sequences (often just two sequences). It differs from problems of finding common [substrings](https://en.wikipedia.org/wiki/Substring): unlike substrings, subsequences are not required to occupy consecutive positions within the original sequences. The longest common subsequence problem is a classic [computer science](https://en.wikipedia.org/wiki/Computer_science) problem, the basis of [data comparison](https://en.wikipedia.org/wiki/Data_comparison) programs such as the [diff utility](https://en.wikipedia.org/wiki/Diff_utility), and has applications in [bioinformatics](https://en.wikipedia.org/wiki/Bioinformatics). It is also widely used by [revision control systems](https://en.wikipedia.org/wiki/Revision_control) such as [Git](https://en.wikipedia.org/wiki/Git_(software)) for [reconciling](https://en.wikipedia.org/wiki/Merge_(revision_control)) multiple changes made to a revision-controlled collection of files.

(<https://en.wikipedia.org/wiki/Longest_common_subsequence_problem>)

* ***~~Boolean Parenthesization~~*** *(like)*

Given a boolean expression with symbols(T or F) and operators (&,|,^):Count the number of ways we can parenthesize the expression so that the value of expression evaluates to true.

Let the input be in form of two arrays one contains the symbols (T and F) in order and other contains operators (&, | and ^)

(<http://www.geeksforgeeks.org/dynamic-programming-set-37-boolean-parenthesization-problem/>)

* ***Building Bridges***

**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.

(<http://www.geeksforgeeks.org/dynamic-programming-set-14-variations-of-lis/>)

8 1 4 3 5 2 6 7

<---- Cities on the other bank of river---->

--------------------------------------------

<--------------- River--------------->

--------------------------------------------

1 2 3 4 5 6 7 8

<------- Cities on one bank of river------->

* ***Rod Cutting***

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. For example, if length of the rod is 8 and the values of different pieces are given as following, then the maximum obtainable value is 22 (by cutting in two pieces of lengths 2 and 6)

length | 1 2 3 4 5 6 7 8

--------------------------------------------

price | 1 5 8 9 10 17 17 20

And if the prices are as following, then the maximum obtainable value is 24 (by cutting in eight pieces of length 1)

length | 1 2 3 4 5 6 7 8

--------------------------------------------

price | 3 5 8 9 10 17 17 20

The naive solution for this problem is to generate all configurations of different pieces and find the highest priced configuration. This solution is exponential in term of time complexity. Let us see how this problem possesses both important properties of a Dynamic Programming (DP) Problem and can efficiently solved using Dynamic Programming.

(<http://www.geeksforgeeks.org/dynamic-programming-set-13-cutting-a-rod/>)

* ***~~Boolean Knapsack~~***

We have computed data files that we want to store, and we have available bytes of storage.  
File has size bytes and takes minutes to re- compute.

We want to avoid as much recomputing as possible, so we want to find a subset of files to store such that

The files have combined size at most .

The total computing time of the stored files is as large as possible.  
We can not store parts of files, it is the whole file or nothing.

How should we select the files?

* ***~~Matrix Chain Multiplication~~***

**Matrix chain multiplication** (or Matrix Chain Ordering Problem, MCOP) is an [optimization problem](https://en.wikipedia.org/wiki/Optimization_problem) that can be solved using [dynamic programming](https://en.wikipedia.org/wiki/Dynamic_programming). Given a sequence of matrices, the goal is to find the most efficient way to [multiply these matrices](https://en.wikipedia.org/wiki/Matrix_multiplication). The problem is not actually to *perform* the multiplications, but merely to decide the sequence of the matrix multiplications involved.

Here are many options because matrix multiplication is [associative](https://en.wikipedia.org/wiki/Associativity). In other words, no matter how the product is [parenthesized](https://en.wikipedia.org/wiki/Bracket_(mathematics)), the result obtained will remain the same. For example, for four matrices *A*, *B*, *C*, and *D*, we would have:

((*AB*)*C*)*D* = ((*A*(*BC*))*D*) = (*AB*)(*CD*) = *A*((*BC*)*D*) = *A*(*B*(*CD*)).

However, the order in which the product is parenthesized affects the number of simple arithmetic operations needed to compute the product, or the *efficiency*. For example, if *A* is a 10 × 30 matrix, *B* is a 30 × 5 matrix, and *C* is a 5 × 60 matrix, then

computing (*AB*)*C* needs (10×30×5) + (10×5×60) = 1500 + 3000 = 4500 operations, while

computing *A*(*BC*) needs (30×5×60) + (10×30×60) = 9000 + 18000 = 27000 operations.

Clearly the first method is more efficient. With this information, the problem statement can be refined as "how to determine the optimal parenthesization of a product of *n* matrices?" Checking each possible parenthesization ([brute force](https://en.wikipedia.org/wiki/Brute-force_search)) would require a [run-time](https://en.wikipedia.org/wiki/Time_complexity) that is exponential in the number of matrices, which is very slow and impractical for large *n*. A quicker solution to this problem can be achieved by breaking up the problem into a set of related subproblems. By solving subproblems once and reusing the solutions, the required run-time can drastically reduced. This concept is known as [dynamic programming](https://en.wikipedia.org/wiki/Dynamic_programming).

(<https://en.wikipedia.org/wiki/Matrix_chain_multiplication>)

* ***~~Longest Palindromic Subsequence~~***

Given a sequence, find the length of the longest palindromic subsequence in it. For example, if the given sequence is “BBABCBCAB”, then the output should be 7 as “BABCBAB” is the longest palindromic subsequence in it. “BBBBB” and “BBCBB” are also palindromic subsequences of the given sequence, but not the longest ones.

The naive solution for this problem is to generate all subsequences of the given sequence and find the longest palindromic subsequence. This solution is exponential in term of time complexity. Let us see how this problem possesses both important properties of a Dynamic Programming (DP) Problem and can efficiently solved using Dynamic Programming.

(<http://www.geeksforgeeks.org/dynamic-programming-set-12-longest-palindromic-subsequence/>)

* ***~~Coin Change~~***

Given a set of coins and amount, Write an algo­rithm to find out how many ways we can make the change of the amount using the coins given.

(<http://algorithms.tutorialhorizon.com/dynamic-programming-coin-change-problem/>)

* ***~~Minimum Sum Partition~~***
* ***~~Word Break~~***

Given an input string and a dictionary of words, find out if the input string can be segmented into a space-separated sequence of dictionary words. See following examples for more details.

This is a famous Google interview question, also being asked by many other companies now a days.

Consider the following dictionary

{ i, like, sam, sung, samsung, mobile, ice,

cream, icecream, man, go, mango}

Input: ilike

Output: Yes

The string can be segmented as "i like".

Input: ilikesamsung

Output: Yes

The string can be segmented as "i like samsung" or "i like sam sung".

(<http://www.geeksforgeeks.org/dynamic-programming-set-32-word-break-problem/>)

* ***~~Minimum Number of Jumps~~***

Given an array of integers where each element represents the max number of steps that can be made forward from that element. Write a function to return the minimum number of jumps to reach the end of the array (starting from the first element). If an element is 0, then cannot move through that element.

Example:

Input: arr[] = {1, 3, 5, 8, 9, 2, 6, 7, 6, 8, 9}

Output: 3 (1-> 3 -> 8 ->9)

First element is 1, so can only go to 3. Second element is 3, so can make at most 3 steps eg to 5 or 8 or 9.

*(*[*http://www.geeksforgeeks.org/minimum-number-of-jumps-to-reach-end-of-a-given-array/*](http://www.geeksforgeeks.org/minimum-number-of-jumps-to-reach-end-of-a-given-array/)*)*

*(*[*http://practice.geeksforgeeks.org/problems/minimum-number-of-jumps/0*](http://practice.geeksforgeeks.org/problems/minimum-number-of-jumps/0)*)*