Dynamic Programming

1. [Fibonacci](https://leetcode.com/problems/fibonacci-number/)

Text

Description automatically generated

1. [Coin Change](https://leetcode.com/problems/coin-change/)

Text

Description automatically generated

1. Sum Possible

Graphical user interface, text, application

Description automatically generated

1. [Palindromic Substrings](https://leetcode.com/problems/palindromic-substrings/)

Must Refer: <https://www.youtube.com/watch?v=XmSOWnL6T_I>

Diagram

Description automatically generated

Here is the sample framework for our dynamic programming solution:

* Define the dynamic programming state. The is the result that gets reused in further computations

Let’s define our state dp(i,j) which tells us whether the substring composed of the ith and jth characters of the input string , is a palindrome or not

Thus, the answer to our problem lies in counting all substrings whose state is true.

* Identify the base cases. There are essentially two base-cases:
* Single letter substrings are palindromes by definition i.e

Dp(i,i) = true

* Double letter substrings composed of the same character are palindromes

i.e dp(i, i+1) = {true if si = si+1

false otherwise

* Identify the optimal substructure. A string is considered a palindrome if:
  + Its first and last characters are equal and
  + The rest of the string(excluding the boundary characters) is also a palindrome

The optimal substructure can be formulated into a recurrence rule.

Dp(i, j) = {true if dp(i+1, j-1) ^ (si== sj)

False otherwise

* Identify overlapping sub-problems and compute them only once. If we compute the states for all smaller strings first, larger strings can be processed by reusing previously saved states.
* The answer is found by counting all states that evaluate true.

Text

Description automatically generated

1. [Longest Palindromic Substring](https://leetcode.com/problems/longest-palindromic-substring/)

Text, whiteboard

Description automatically generated

1. [Ones and Zeros](https://leetcode.com/problems/ones-and-zeroes/)

For Every function call

* Count the total number of 1’s and 0’s in a string.
* Include the current string in the subset currently being considered, if we include the current the current string, we will need to deduct number of 0’s and 1’s in the current string from total available respective counts. We also increment the total number of strings considered so far by 1.
* Not include the current string in the current subset, we don’t have to update the count 1’s and 0’s.
* Math.Max(taken, notTaken) result to be returned from the current function call.

**Complexity Analysis:**

Time Complexity: O(l\*m\*n), memo object of size l\*m\*n filled. where l is the length of strs, m and n are number of 0’s and 1’s respectively

Space: O(l\*m\*n)

Text

Description automatically generated

1. [Longest Common Subsequence](https://leetcode.com/problems/longest-common-subsequence/)

Graphical user interface, text, application

Description automatically generated

1. [Longest Palindromic Subsequence](https://leetcode.com/problems/longest-palindromic-subsequence/)

Graphical user interface, text, application

Description automatically generated

1. [Russian Dull Envelopes](https://leetcode.com/problems/russian-doll-envelopes/)

We know that each envelope must be increasing in w, thus our best arrangement must be a subsequence of all of all our envelopes sorted on w.

After we sort our envelopes, we can simply find the length of the longest increasing subsequence on the second dimension(h). Note that we use clever trick to solve edge cases.

Consider an input [[1,3], [1,4],[1,5],[2,3]], if we simply sort and extract the second dimension we get [3, 4, 5 ,3], which implies that we can fit three envelopes (3,4,5). The problem is that we can only fit one envelope, since envelopes that are equal in the first dimension can’t put into each other.

To fix this, we don’t just sort increasing in the first dimension, we also sort decreasing on the second dimension, so two envelopes that are equal in the first dimension can never be in the same increasing subsequence.

Now when we sort and extract the second element from the input we get [5,4,3,3], which correctly reflects an LIS of one.

