Number of Ways: https://leetcode.com/problems/decode-ways/

```
class Solution {
    public int numDecodings(String s) {

        if(s.length() == 0) return 0;

        int[] dp = new int[s.length() + 1];
        dp[s.length()] = 1;
        char ch = s.charAt(s.length() - 1);
        if(ch != '0') dp[s.length() - 1] = 1;

        for(int i = s.length() - 2; i >= 0; i--) {
            ch = s.charAt(i);
            if(ch != '0') dp[i] = dp[i + 1];
            int val = (ch - '0') * 10 + (s.charAt(i + 1) - '0');
            if(val > 9 && val <= 26) dp[i] = dp[i + 1] + dp[i + 2];
        }

        return dp[0];
    }
}
```

Longest Common Subsequence: https://leetcode.com/problems/longest-common-subsequence/

```
class Solution {
    public int longestCommonSubsequence(String text1, String text2) {

        char[] ch1 = text1.toCharArray();
        char[] ch2 = text2.toCharArray();

        int rows = ch1.length, cols = ch2.length;
        int[][] dp = new int[rows + 1][cols + 1];

        for(int i = 1; i <= rows; i++)
            for(int j = 1; j <= cols; j++) {
                if(ch1[i - 1] == ch2[j - 1]) dp[i][j] = dp[i - 1][j - 1] + 1;
                else dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
            }

        return dp[rows][cols];
    }
}
```

Maximum Problem https://leetcode.com/problems/maximum-product-subarray/

```
class Solution {
    public int maxProduct(int[] nums) {

        int max = nums[0], min = nums[0], result = nums[0];
        for(int i = 1; i < nums.length; i++) {
            int temp = max;
            max = Math.max(Math.max(nums[i] * max, nums[i] * min), nums[i]);
            min = Math.min(Math.min(nums[i] * temp, nums[i] * min), nums[i]);
            if(max > result)
                result = max;
        }
        return result;
    }
}
```

Minimum Number of Operations: https://leetcode.com/problems/edit-distance/

```
class Solution {

    public int minDistance(String word1, String word2) {

        if(word1.equals(word2)) return 0;
        int[] dp = new int[word2.length() + 1];
        for(int i = 0; i < dp.length; i++) dp[i] = i;

        for(int i = 1; i<= word1.length(); i++) {
            int prev = dp[0];
            ++dp[0];
            for(int j = 1; j <= word2.length(); j++) {
                int temp = dp[j];
                int val1 = Math.min(dp[j - 1], dp[j]) + 1;
                int val2 = word1.charAt(i - 1) == word2.charAt(j - 1) ? prev : prev + 1;
                dp[j] = Math.min(val1, val2);
                prev = temp;
            }
        }

        return dp[word2.length()];
    }
}
```