

**Dynamic Programming**

**Very Easy:**

**1.** [**N-th Tribonacci Number**](https://leetcode.com/problems/n-th-tribonacci-number/)The Tribonacci sequence Tn is defined as follows:

T0 = 0, T1 = 1, T2 = 1, and Tn+3 = Tn + Tn+1 + Tn+2 for n >= 0.

Given n, return the value of Tn.

**Example 1:**

**Input: n = 4**

**Output: 4**

**Explanation:**

**T\_3 = 0 + 1 + 1 = 2**

**T\_4 = 1 + 1 + 2 = 4**

**Example 2: Input: n = 25**

**Output: 1389537**

**Constraints: 0 <= n <= 37**

**The answer is guaranteed to fit within a 32-bit integer, ie. answer <= 2^31 - 1**.

**URL-** [**https://leetcode.com/problems/n-th-tribonacci-number/description/?envType=problem-list-v2&envId=dynamic-programmin**](https://leetcode.com/problems/n-th-tribonacci-number/description/?envType=problem-list-v2&envId=dynamic-programmin)

**2.** [**Divisor Game**](https://leetcode.com/problems/divisor-game/)

Alice and Bob take turns playing a game, with Alice starting first. Initially, there is a number n on the chalkboard. On each player's turn, that player makes a move consisting of:

Choosing any x with 0 < x < n and n % x == 0.

Replacing the number n on the chalkboard with n - x.

Also, if a player cannot make a move, they lose the game.

Return true if and only if Alice wins the game, assuming both players play optimally.

**Example 1:**

**Input: n = 2**

**Output: true**

**Explanation: Alice chooses 1, and Bob has no more moves.**

**Example 2:**

**Input: n = 3**

**Output: false**

**Explanation: Alice chooses 1, Bob chooses 1, and Alice has no more moves.**

**Constraints: 1 <= n <= 1000**

**URL-** [**https://leetcode.com/problems/divisor-game/description/?envType=problem-list-v2&envId=dynamic-programming&favoriteSlug=&difficulty=EASY**](https://leetcode.com/problems/divisor-game/description/?envType=problem-list-v2&envId=dynamic-programming&favoriteSlug=&difficulty=EASY)

**3.** [**Maximum Repeating Substring**](https://leetcode.com/problems/maximum-repeating-substring/)

For a string sequence, a string word is k-repeating if word concatenated k times is a substring of sequence. The word's maximum k-repeating value is the highest value k where word is k-repeating in sequence. If word is not a substring of sequence, word's maximum k-repeating value is 0. Given strings sequence and word, return the maximum k-repeating value of word in sequence.

**Example 1:**

**Input: sequence = "ababc", word = "ab"**

**Output: 2**

**Explanation: "abab" is a substring in "ababc".**

**Example 2:**

**Input: sequence = "ababc", word = "ba"**

**Output: 1**

**Explanation: "ba" is a substring in "ababc". "baba" is not a substring in "ababc".**

**Example 3:**

**Input: sequence = "ababc", word = "ac"**

**Output: 0**

**Explanation: "ac" is not a substring in "ababc".**

**Constraints:**

**1 <= sequence.length <= 100**

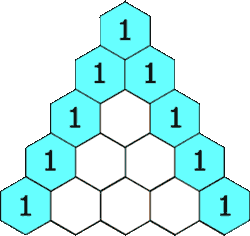
**1 <= word.length <= 100**

**sequence and word contains only lowercase English letters.**

**URL-** [**https://leetcode.com/problems/maximum-repeating-substring/description/?envType=problem-list-v2&envId=dynamic-programming&favoriteSlug=&difficulty=EASY**](https://leetcode.com/problems/maximum-repeating-substring/description/?envType=problem-list-v2&envId=dynamic-programming&favoriteSlug=&difficulty=EASY)

**4.** [**Pascal's Triangle II**](https://leetcode.com/problems/pascals-triangle-ii/)

Given an integer rowIndex, return the rowIndexth (0-indexed) row of the Pascal's triangle. In Pascal's triangle, each number is the sum of the two numbers directly above it as shown:



**Example 1:**

**Input: rowIndex = 3**

**Output: [1,3,3,1]**

**Example 2:**

**Input: rowIndex = 0**

**Output: [1]**

**Example 3:**

**Input: rowIndex = 1**

**Output: [1,1]**

**Constraints: 0 <= rowIndex <= 33**

**URL-** [**https://leetcode.com/problems/pascals-triangle-ii/description/?envType=problem-list-v2&envId=dynamic-programming&favoriteSlug=&difficulty=EASY**](https://leetcode.com/problems/pascals-triangle-ii/description/?envType=problem-list-v2&envId=dynamic-programming&favoriteSlug=&difficulty=EASY)

**5.** [**Maximum Repeating Substring**](https://leetcode.com/problems/maximum-repeating-substring/)

For a string sequence, a string word is k-repeating if word concatenated k times is a substring of sequence. The word's maximum k-repeating value is the highest value k where word is k-repeating in sequence. If word is not a substring of sequence, word's maximum k-repeating value is 0.

Given strings sequence and word, return the maximum k-repeating value of word in sequence.

**Example 1:**

**Input: sequence = "ababc", word = "ab"**

**Output: 2**

**Explanation: "abab" is a substring in "ababc".**

**Example 2:**

**Input: sequence = "ababc", word = "ba"**

**Output: 1**

**Explanation: "ba" is a substring in "ababc". "baba" is not a substring in "ababc".**

**Example 3:**

**Input: sequence = "ababc", word = "ac"**

**Output: 0**

**Explanation: "ac" is not a substring in "ababc".**

**Constraints:**

**1 <= sequence.length <= 100**

**1 <= word.length <= 100**

**sequence and word contains only lowercase English letters.**

URL- <https://leetcode.com/problems/maximum-repeating-substring/description/?envType=problem-list-v2&envId=dynamic-programming&favoriteSlug=&difficulty=EASY>

**Easy:   
  
 1.** [**Climbing Stairs**](https://leetcode.com/problems/climbing-stairs/)

1. You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

**Example 1: Input: n = 2**

**Output: 2**

**Explanation: There are two ways to climb to the top.**

**1. 1 step + 1 step**

**2. 2 steps**

**Example 2:Input: n = 3**

**Output: 3**

**Explanation: There are three ways to climb to the top.**

**1. 1 step + 1 step + 1 step**

**2. 1 step + 2 steps**

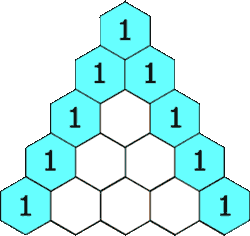
**3. 2 steps + 1 step**

**Constraints:1 <= n <= 45**

**URL-** [**https://leetcode.com/problems/climbing-stairs/description/?envType=problem-list-v2&envId=dynamic-programming**](https://leetcode.com/problems/climbing-stairs/description/?envType=problem-list-v2&envId=dynamic-programming)

**2. Pascal's Triangle**

1. Given an integer numRows, return the first numRows of Pascal's triangle.In Pascal's triangle, each number is the sum of the two numbers directly above it as shown:



**Example 1:Input: numRows = 5**

**Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]**

**Example 2:Input: numRows = 1**

**Output: [[1]]**

**Constraints:1 <= numRows <= 30**

**URL-** [**https://leetcode.com/problems/pascals-triangle/description/?envType=problem-list-v2&envId=dynamic-programming**](https://leetcode.com/problems/pascals-triangle/description/?envType=problem-list-v2&envId=dynamic-programming)

**3.** [**Best Time to Buy and Sell Stock**](https://leetcode.com/problems/best-time-to-buy-and-sell-stock/)

You are given an array prices where prices[i] is the price of a given stock on the ith day.You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

**Example 1:Input: prices = [7,1,5,3,6,4]**

**Output: 5**

**Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.**

**Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.**

**Example 2:Input: prices = [7,6,4,3,1]**

**Output: 0**

**Explanation: In this case, no transactions are done and the max profit = 0.**

**Constraints:1 <= prices.length <= 105**

**0 <= prices[i] <= 104**

**URL-** [**https://leetcode.com/problems/best-time-to-buy-and-sell-stock/description/?envType=problem-list-v2&envId=dynamic-programming**](https://leetcode.com/problems/best-time-to-buy-and-sell-stock/description/?envType=problem-list-v2&envId=dynamic-programming)

**4.** [**Counting Bits**](https://leetcode.com/problems/counting-bits/)

1. Given an integer n, return an array ans of length n + 1 such that for each i (0 <= i <= n), ans[i] is the number of 1's in the binary representation of i.

**Example 1:Input: n = 2 | Output: [0,1,1]**

**Explanation:0 --> 0**

**1 --> 1**

**2 --> 10**

**Example 2:Input: n = 5**

**Output: [0,1,1,2,1,2]**

**Explanation:0 --> 0 1 --> 1 2 --> 10 3 --> 11 4 --> 100 5 --> 101**

**Constraints:0 <= n <= 105**

**Follow up:It is very easy to come up with a solution with a runtime of O(n log n). Can you do it in linear time O(n) and possibly in a single pass?**

**Can you do it without using any built-in function (i.e., like \_\_builtin\_popcount in C++)?**

**URL-**[**https://leetcode.com/problems/counting-bits/description/?envType=problem-list-v2&envId=dynamic-programming**](https://leetcode.com/problems/counting-bits/description/?envType=problem-list-v2&envId=dynamic-programming)

**5.** [**Is Subsequence**](https://leetcode.com/problems/is-subsequence/)

Given two strings s and t, return true if s is a subsequence of t, or false otherwise.A subsequence of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., "ace" is a subsequence of "abcde" while "aec" is not).

**Example 1:**

**Input: s = "abc", t = "ahbgdc"**

**Output: true**

**Example 2:Input: s = "axc", t = "ahbgdc"**

**Output: false**

**Constraints:0 <= s.length <= 100**

**0 <= t.length <= 104**

**s and t consist only of lowercase English letters**.

**URL-** [**https://leetcode.com/problems/is-subsequence/description/?envType=problem-list-v2&envId=dynamic-programming**](https://leetcode.com/problems/is-subsequence/description/?envType=problem-list-v2&envId=dynamic-programming)

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

Given a string s, return the longest palindromic substring in s.

**Example 1: Input: s = "babad"**

**Output: "bab"**

**Explanation: "aba" is also a valid answer.**

**Example 2: Input: s = "cbbd"**

**Output: "bb"**

**Constraints: 1 <= s.length <= 1000**

**s consist of only digits and English letters.**

**URL-** [**https://leetcode.com/problems/longest-palindromic-substring/description/?envType=problem-list-v2&envId=dynamic-programming**](https://leetcode.com/problems/longest-palindromic-substring/description/?envType=problem-list-v2&envId=dynamic-programming)

**2.** [**Generate Parentheses**](https://leetcode.com/problems/generate-parentheses/)

1. Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

**Example 1: Input: n = 3**

**Output: ["((()))","(()())","(())()","()(())","()()()"]**

**Example 2: Input: n = 1**

**Output: ["()"]**

**Constraints: 1 <= n <= 8**

**URL-** [**https://leetcode.com/problems/generate-parentheses/description/?envType=problem-list-v2&envId=dynamic-programming**](https://leetcode.com/problems/generate-parentheses/description/?envType=problem-list-v2&envId=dynamic-programming)

**3.** [**Jump Game**](https://leetcode.com/problems/jump-game/)

You are given an integer array nums. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position.

Return true if you can reach the last index, or false otherwise.

**Example 1: Input: nums = [2,3,1,1,4]**

**Output: true**

**Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.**

**Example 2: Input: nums = [3,2,1,0,4]**

**Output: false**

**Explanation: You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.**

**Constraints: 1 <= nums.length <= 104**

**0 <= nums[i] <= 105**

**URL-** [**https://leetcode.com/problems/jump-game/description/?envType=problem-list-v2&envId=dynamic-programming**](https://leetcode.com/problems/jump-game/description/?envType=problem-list-v2&envId=dynamic-programming)

**4.** [**Minimum Path Sum**](https://leetcode.com/problems/minimum-path-sum/)

Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

**Example 1:**

****

**Input: grid = [[1,3,1],[1,5,1],[4,2,1]]**

**Output: 7**

**Explanation: Because the path 1 → 3 → 1 → 1 → 1 minimizes the sum.**

**Example 2: Input: grid = [[1,2,3],[4,5,6]] Output: 12**

**Constraints:**

**m == grid.length**

**n == grid[i].length**

**1 <= m, n <= 200**

**0 <= grid[i][j] <= 200**

URL- <https://leetcode.com/problems/minimum-path-sum/?envType=problem-list-v2&envId=dynamic-programming>

**5 Given an integer n, return the least number of perfect square numbers that sum to n.**

**A perfect square is an integer that is the square of an integer; in other words, it is the product of some integer with itself. For example, 1, 4, 9, and 16 are perfect squares while 3 and 11 are not.**

**Example 1:**

**Input: n = 12**

**Output: 3**

**Explanation: 12 = 4 + 4 + 4.**

**Example 2:**

**Input: n = 13**

**Output: 2**

**Explanation: 13 = 4 + 9.**

**Constraints:**

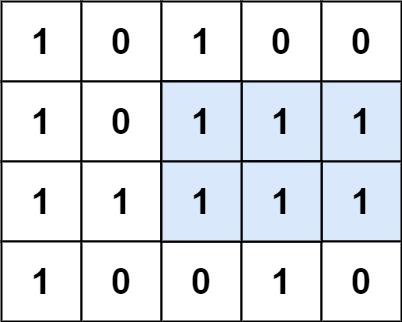
**1 <= n <= 104**

**URL-**[**https://leetcode.com/problems/perfect-squares/description/**](https://leetcode.com/problems/perfect-squares/description/)

**Hard:   
  
 1.** [**Maximal Rectangle**](https://leetcode.com/problems/maximal-rectangle/)

1. Given a rows x cols binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area.

Example-



**Input: matrix = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]**

**Output: 6**

**Explanation: The maximal rectangle is shown in the above picture.**

**Example 2: Input: matrix = [["0"]]**

**Output: 0**

**Example 3: Input: matrix = [["1"]]**

**Output: 1**

**Constraints:**

**rows == matrix.length**

**cols == matrix[i].length**

**1 <= row, cols <= 200**

**matrix[i][j] is '0' or '1'.**

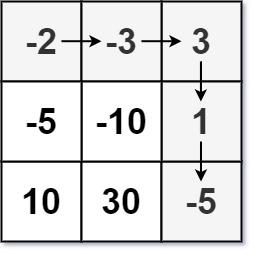
**URL-** [**https://leetcode.com/problems/maximal-rectangle/description/?envType=problem-list-v2&envId=dynamic-programming**](https://leetcode.com/problems/maximal-rectangle/description/?envType=problem-list-v2&envId=dynamic-programming)

[**2. Dungeon Game**](https://leetcode.com/problems/dungeon-game/)

1. The demons had captured the princess and imprisoned her in the bottom-right corner of a dungeon. The dungeon consists of m x n rooms laid out in a 2D grid. Our valiant knight was initially positioned in the top-left room and must fight his way through dungeon to rescue the princess. The knight has an initial health point represented by a positive integer. If at any point his health point drops to 0 or below, he dies immediately Some of the rooms are guarded by demons (represented by negative integers), so the knight loses health upon entering these rooms; other rooms are either empty (represented as 0) or contain magic orbs that increase the knight's health (represented by positive integers). To reach the princess as quickly as possible, the knight decides to move only rightward or downward in each step. Return the knight's minimum initial health so that he can rescue the princess.

Note that any room can contain threats or power-ups, even the first room the knight enters and the bottom-right room where the princess is imprisoned.

**Example-**

****

**Input: dungeon = [[-2,-3,3],[-5,-10,1],[10,30,-5]]**

**Output: 7**

**Explanation: The initial health of the knight must be at least 7 if he follows the optimal path: RIGHT-> RIGHT -> DOWN -> DOWN.**

**Example 2: Input: dungeon = [[0]]**

**Output: 1**

**Constraints:**

**m == dungeon.length**

**n == dungeon[i].length**

**1 <= m, n <= 200**

**-1000 <= dungeon[i][j] <= 1000**

**URL-** [**https://leetcode.com/problems/dungeon-game/description/?envType=problem-list-v2&envId=dynamic-programming**](https://leetcode.com/problems/dungeon-game/description/?envType=problem-list-v2&envId=dynamic-programming)

[**3. Number of Digit One**](https://leetcode.com/problems/number-of-digit-one/)Given an integer n, count the total number of digit 1 appearing in all non-negative integers less than or equal to n.

**Example 1: Input: n = 13**

**Output: 6**

**Example 2: Input: n = 0**

**Output: 0**

**Constraints: 0 <= n <= 109**

**URL-** [**https://leetcode.com/problems/number-of-digit-one/description/?envType=problem-list-v2&envId=dynamic-programming**](https://leetcode.com/problems/number-of-digit-one/description/?envType=problem-list-v2&envId=dynamic-programming)

**4.** [**Burst Balloons**](https://leetcode.com/problems/burst-balloons/)

4.You are given n balloons, indexed from 0 to n - 1. Each balloon is painted with a number on it represented by an array nums. You are asked to burst all the balloons. If you burst the ith balloon, you will get nums[i - 1] \* nums[i] \* nums[i + 1] coins. If i - 1 or i + 1 goes out of bounds of the array, then treat it as if there is a balloon with a 1 painted on it. Return the maximum coins you can collect by bursting the balloons wisely.

**Example 1: Input: nums = [3,1,5,8]**

**Output: 167**

**Explanation:**

**nums = [3,1,5,8] --> [3,5,8] --> [3,8] --> [8] --> []**

**coins = 3\*1\*5 + 3\*5\*8 + 1\*3\*8 + 1\*8\*1 = 167**

**Example 2: Input: nums = [1,5]**

**Output: 10**

**Constraints:**

**n == nums.length**

**1 <= n <= 300**

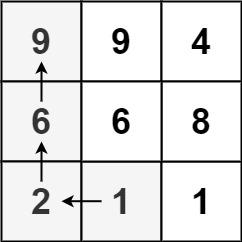
**0 <= nums[i] <= 100**

URL- <https://leetcode.com/problems/burst-balloons/description/?envType=problem-list-v2&envId=dynamic-programming>

[**5. Longest Increasing Path in a Matrix**](https://leetcode.com/problems/longest-increasing-path-in-a-matrix/)

5.Given an m x n integers matrix, return the length of the longest increasing path in matrix. From each cell, you can either move in four directions: left, right, up, or down. You may not move diagonally or move outside the boundary (i.e., wrap-around is not allowed).

**Example 1:**

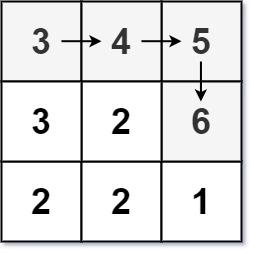
****

**Input: matrix = [[9,9,4],[6,6,8],[2,1,1]]**

**Output: 4**

**Explanation: The longest increasing path is [1, 2, 6, 9].**

**Example 2:**

****

**Input: matrix = [[3,4,5],[3,2,6],[2,2,1]]**

**Output: 4**

**Explanation: The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not allowed.**

**Example 3: Input: matrix = [[1]]**

**Output: 1**

**Constraints:**

**m == matrix.length**

**n == matrix[i].length**

**1 <= m, n <= 200**

**0 <= matrix[i][j] <= 231 – 1**

URL- <https://leetcode.com/problems/longest-increasing-path-in-a-matrix/description/?envType=problem-list-v2&envId=dynamic-programming>

**Very Hard:   
  
 1.** [**Cherry Pickup**](https://leetcode.com/problems/cherry-pickup/)

1. You are given an n x n grid representing a field of cherries, each cell is one of three possible integers.

0 means the cell is empty, so you can pass through,

1 means the cell contains a cherry that you can pick up and pass through, or

-1 means the cell contains a thorn that blocks your way.

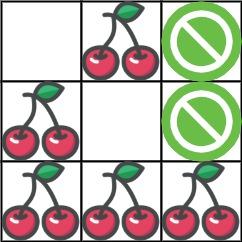
Return the maximum number of cherries you can collect by following the rules below:

Starting at the position (0, 0) and reaching (n - 1, n - 1) by moving right or down through valid path cells (cells with value 0 or 1).

After reaching (n - 1, n - 1), returning to (0, 0) by moving left or up through valid path cells.

When passing through a path cell containing a cherry, you pick it up, and the cell becomes an empty cell 0.

If there is no valid path between (0, 0) and (n - 1, n - 1), then no cherries can be collected.



**Input: grid = [[0,1,-1],[1,0,-1],[1,1,1]]**

**Output: 5**

**Explanation: The player started at (0, 0) and went down, down, right right to reach (2, 2).**

**4 cherries were picked up during this single trip, and the matrix becomes [[0,1,-1],[0,0,-1],[0,0,0]].**

**Then, the player went left, up, up, left to return home, picking up one more cherry.**

**The total number of cherries picked up is 5, and this is the maximum possible.**

**Example 2: Input: grid = [[1,1,-1],[1,-1,1],[-1,1,1]]**

**Output: 0**

**Constraints:**

**n == grid.length**

**n == grid[i].length**

**1 <= n <= 50**

**grid[i][j] is -1, 0, or 1.**

**grid[0][0] != -1**

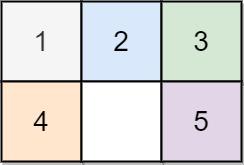
**grid[n - 1][n - 1] != -1**

URL- <https://leetcode.com/problems/cherry-pickup/description/?envType=problem-list-v2&envId=dynamic-programming>

**2.** [**Sliding Puzzle**](https://leetcode.com/problems/sliding-puzzle/)

On an 2 x 3 board, there are five tiles labeled from 1 to 5, and an empty square represented by 0. A move consists of choosing 0 and a 4-directionally adjacent number and swapping it. The state of the board is solved if and only if the board is [[1,2,3],[4,5,0]]. Given the puzzle board board, return the least number of moves required so that the state of the board is solved. If it is impossible for the state of the board to be solved, return -1.

**Example 1:**

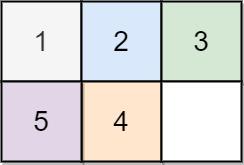
****

**Input: board = [[1,2,3],[4,0,5]]**

**Output: 1**

**Explanation: Swap the 0 and the 5 in one move.**

**Example 2:**

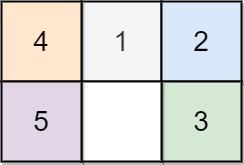
****

**Input: board = [[1,2,3],[5,4,0]]**

**Output: -1**

**Explanation: No number of moves will make the board solved.**

**Example 3:**

****

**Input: board = [[4,1,2],[5,0,3]]**

**Output: 5**

**Explanation: 5 is the smallest number of moves that solves the board.**

**An example path:**

**After move 0: [[4,1,2],[5,0,3]]**

**After move 1: [[4,1,2],[0,5,3]]**

**After move 2: [[0,1,2],[4,5,3]]**

**After move 3: [[1,0,2],[4,5,3]]**

**After move 4: [[1,2,0],[4,5,3]]**

**After move 5: [[1,2,3],[4,5,0]]**

**Constraints:**

**board.length == 2**

**board[i].length == 3**

**0 <= board[i][j] <= 5**

**Each value board[i][j] is unique.**

URL- <https://leetcode.com/problems/sliding-puzzle/description/?envType=problem-list-v2&envId=dynamic-programming>

[**3. Race Car**](https://leetcode.com/problems/race-car/)

Your car starts at position 0 and speed +1 on an infinite number line. Your car can go into negative positions. Your car drives automatically according to a sequence of instructions 'A' (accelerate) and 'R' (reverse):

When you get an instruction 'A', your car does the following:

position += speed

speed \*= 2

When you get an instruction 'R', your car does the following:

If your speed is positive then speed = -1

otherwise speed = 1

Your position stays the same.

For example, after commands "AAR", your car goes to positions 0 --> 1 --> 3 --> 3, and your speed goes to 1 --> 2 --> 4 --> -1.

Given a target position target, return the length of the shortest sequence of instructions to get there.

**Example 1:**

**Input: target = 3**

**Output: 2**

**Explanation:**

**The shortest instruction sequence is "AA".**

**Your position goes from 0 --> 1 --> 3.**

**Example 2:**

**Input: target = 6**

**Output: 5**

**Explanation:**

**The shortest instruction sequence is "AAARA".**

**Your position goes from 0 --> 1 --> 3 --> 7 --> 7 --> 6.**

**Constraints: 1 <= target <= 104**

URL- <https://leetcode.com/problems/race-car/description/?envType=problem-list-v2&envId=dynamic-programming>

**4.** [**Super Egg Drop**](https://leetcode.com/problems/super-egg-drop/)

You are given k identical eggs and you have access to a building with n floors labeled from 1 to n. You know that there exists a floor f where 0 <= f <= n such that any egg dropped at a floor higher than f will break, and any egg dropped at or below floor f will not break. Each move, you may take an unbroken egg and drop it from any floor x (where 1 <= x <= n). If the egg breaks, you can no longer use it. However, if the egg does not break, you may reuse it in future moves. Return the minimum number of moves that you need to determine with certainty what the value of f is.

**Example 1: Input: k = 1, n = 2**

**Output: 2**

**Explanation:**

**Drop the egg from floor 1. If it breaks, we know that f = 0.**

**Otherwise, drop the egg from floor 2. If it breaks, we know that f = 1.**

**If it does not break, then we know f = 2.**

**Hence, we need at minimum 2 moves to determine with certainty what the value of f is.**

**Example 2: Input: k = 2, n = 6**

**Output: 3**

**Example 3:**

**Input: k = 3, n = 14**

**Output: 4**

**Constraints:**

**1 <= k <= 100**

**1 <= n <= 104**

URL- <https://leetcode.com/problems/super-egg-drop/description/?envType=problem-list-v2&envId=dynamic-programming>

**5.** [**Number of Music Playlists**](https://leetcode.com/problems/number-of-music-playlists/)

Your music player contains n different songs. You want to listen to goal songs (not necessarily different) during your trip. To avoid boredom, you will create a playlist so that: Every song is played at least once. A song can only be played again only if k other songs have been played.

Given n, goal, and k, return the number of possible playlists that you can create. Since the answer can be very large, return it modulo 109 + 7.

**Example 1:**

**Input: n = 3, goal = 3, k = 1**

**Output: 6**

**Explanation: There are 6 possible playlists: [1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], and [3, 2, 1].**

**Example 2:**

**Input: n = 2, goal = 3, k = 0**

**Output: 6**

**Explanation: There are 6 possible playlists: [1, 1, 2], [1, 2, 1], [2, 1, 1], [2, 2, 1], [2, 1, 2], and [1, 2, 2].**

**Example 3:**

**Input: n = 2, goal = 3, k = 1**

**Output: 2**

**Explanation: There are 2 possible playlists: [1, 2, 1] and [2, 1, 2].**

**Constraints: 0 <= k < n <= goal <= 100**

URL- <https://leetcode.com/problems/number-of-music-playlists/description/?envType=problem-list-v2&envId=dynamic-programming>