Count number of ways to cover a distance - GeeksforGeeks

Source: https://www.geeksforgeeks.org/count-number-of-ways-to-cover-a-distance/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Count number of ways to cover a distance Last Updated : 23 Jul, 2025 Given a distance 'dist', count total number of ways to cover the distance with 1, 2 and 3 steps. Examples: Input: n = 3 Output: 4 Explanation: Below are the four ways => 1 step + 1 step + 1 step => 1 step + 2 step => 2 step + 1 step => 3 step Input: n = 4 Output: 7 Explanation: Below are the four ways => 1 step + 1 step + 1 step + 1 step => 1 step + 2 step + 1 step => 2 step + 1 step + 1 step => 1 step + 1 step + 2 step => 2 step + 2 step => 3 step + 1 step => 1 step + 3 step Try it on GfG Practice Recommended Practice Count number of hops Try It! Count number of ways to cover a distance using Recursio n : There are n stairs, and a person is allowed to next step, skip one position or skip two positions. So there are n positions. The idea is standing at the ith position the person can move by i+1, i+2, i+3 position. So a recursive function can be formed where at current index i the function is recursively called for i+1, i+2 and i+3 positions. There is another way of forming the recursive function. To reach position i, a person has to jump either from i-1, i-2 or i-3 position where i is the starting position. Step by step approach: Create a recursive function (count(int n)) which takes only one parameter. Check the base cases. If the value of n is less than 0 then return 0 , and if value of n is equal to zero then return 1 as it is the starting position. Call the function recursively with values n-1, n-2 and n-3 and sum up the values that are returned, i.e. sum = count(n-1) + count(n-2) + count(n-3) . Return the value of sum . Below are the implementation of the above approach: C++ // A naive recursive C++ program to count number of ways to // cover a distance with 1, 2 and 3 steps #include <iostream> using namespace std ; // Returns count of ways to cover 'dist' int printCountRec (int dist) { // Base cases if (dist < 0) return 0 ; if (dist == 0) return 1 ; // Recur for all previous 3 and add the results return printCountRec (dist - 1) + printCountRec (dist - 2) + printCountRec (dist - 3); } // driver program int main () { int dist = 4 ; cout << printCountRec (dist); return 0 ; } Java // A naive recursive Java program to count number // of ways to cover a distance with 1, 2 and 3 steps import java.io.* ; class GFG { // Function returns count of ways to cover 'dist' static int printCountRec (int dist) { // Base cases if (dist < 0) return 0 ; if (dist == 0) return 1 ; // Recur for all previous 3 and add the results return printCountRec (dist - 1) + printCountRec (dist - 2) + printCountRec (dist - 3); } // driver program public static void main (String [] args) { int dist = 4 ; System . out . println (printCountRec (dist)); } } // This code is contributed by Pramod Kumar Python # A naive recursive Python3 program # to count number of ways to cover # a distance with 1, 2 and 3 steps # Returns count of ways to # cover 'dist' def printCountRec (dist): # Base cases if dist < 0 : return 0 if dist == 0 : return 1 # Recur for all previous 3 and # add the results return (printCountRec (dist - 1) + printCountRec (dist - 2) + printCountRec (dist - 3)) # Driver code dist = 4 print (printCountRec (dist)) # This code is contributed by Anant Agarwal. C# // A naive recursive C# program to // count number of ways to cover a // distance with 1, 2 and 3 steps using System ; class GFG { // Function returns count of // ways to cover 'dist' static int printCountRec (int dist) { // Base cases if (dist < 0) return 0 ; if (dist == 0) return 1 ; // Recur for all previous 3 // and add the results return printCountRec (dist - 1) + printCountRec (dist - 2) + printCountRec (dist - 3); } // Driver Code public static void Main () { int dist = 4 ; Console . WriteLine (printCountRec (dist)); } } // This code is contributed by Sam007. JavaScript // A naive recursive javascript program to count number of ways to cover // a distance with 1, 2 and 3 steps // Returns count of ways to cover 'dist' function printCountRec (

dist) { // Base cases if (dist < 0) return 0 ; if (dist == 0) return 1 ; // Recur for all previous 3 and add the results return printCountRec (dist - 1) + printCountRec (dist - 2) + printCountRec (dist - 3); } // driver program var dist = 4 ; console . log (printCountRec (dist)); Output 7 Time Complexity: O(3 n), the time complexity of the above solution is exponential, a close upper bound is O(3 n). From each state 3, a recursive function is called. So the upper bound for n states is O(3 n). Auxiliary Space: O(1), No extra space is required. Count number of ways to cover a distance using Dynamic Programming (Memoization) : The problem have overlapping subproblems , meaning that the same subproblems are encountered multiple times during the recursive computation. For example, when calculating the number of ways to cover distance 'dist', we need to calculate the number of ways to cover distances 'dist-1', 'dist-2', and 'dist-3'. These subproblems are also encountered when calculating the number of ways to cover distances 'dist-2' and 'dist-3'. Memoization solve this issue by storing the results of previously computed subproblems in a data structure, typically an array or a hash table. Before making a recursive call, we first check if the result for the current distance has already been computed and stored in the memo[] array. If it has, we directly return the stored result, avoiding the recursive call. This significantly reduces the number of recursive calls and improves the performance of the algorithm, especially for larger distances. Once the result for the current distance is computed, we store it in the memo[] array for future reference. Below is the implementation of the above approach: C++ #include <iostream> #include <vector> using namespace std ; // Returns count of ways to cover 'dist' using memoization int printCountRecMemo (int dist , vector < int >& memo) { // Base cases if (dist < 0) return 0 ; if (dist == 0) return 1 ; // Check if the value for 'dist' is already computed if (memo [dist] != -1) return memo [dist]; // Recur for all previous 3 and add the results int ways = printCountRecMemo (dist - 1 , memo) + printCountRecMemo (dist - 2 , memo) + printCountRecMemo (dist - 3 , memo); // Memoize the result for 'dist' for future use memo [dist] = ways ; return ways ; } // Function to calculate the count of ways with memoization int countWays (int dist) { vector < int > memo (dist + 1 , -1); // Initialize memoization array with -1 return printCountRecMemo (dist , memo); } // Driver program int main () { int dist = 4 ; cout << countWays (dist); return 0 ; } Java import java.util.Arrays ; public class Main { // Function to calculate the count of ways with memoization static int countWays (int dist) { int [] memo = new int [dist + 1] ; // Initialize memoization array with 0s Arrays . fill (memo , - 1); // Set all elements to -1 to indicate not computed yet return printCountRecMemo (dist , memo); } // Recursive function to calculate the count of ways to cover 'dist' using memoization static int printCountRecMemo (int dist , int [] memo) { // Base cases if (dist < 0) return 0 ; if (dist == 0) return 1 ; // Check if the value for 'dist' is already computed if (memo [dist] != - 1) return memo [dist] ; // Recur for all previous 3 and add the results int ways = printCountRecMemo (dist - 1 , memo) + printCountRecMemo (dist - 2 , memo) + printCountRecMemo (dist - 3 , memo); // Memoize the result for 'dist' for future use memo [dist] = ways ; return ways ; } // Driver program public static void main (String [] args) { int dist = 4 ; System . out . println (countWays (dist)); } } Python # Function to calculate the count of ways to cover 'dist' using memoization def printCountRecMemo (dist , memo): # Base cases if dist < 0 : return 0 if dist == 0 : return 1 # Check if the value for 'dist' is already computed if memo [dist] != - 1 : return memo [dist] # Recur for all previous 3 and add the results ways = printCountRecMemo (dist - 1 , memo) \ + printCountRecMemo (dist - 2 , memo) \ + printCountRecMemo (dist - 3 , memo) # Memoize the result for 'dist' for future use memo [dist] = ways return ways # Function to calculate the count of ways with memoization def countWays (dist): memo = [- 1] * (dist + 1) # Initialize memoization list with -1 return printCountRecMemo (dist , memo) # Driver program if __name__ == "__main__" : dist = 4 print (countWays (dist)) JavaScript // Function to calculate the count of ways with memoization function countWays (dist) { let memo = new Array (dist + 1). fill (- 1); // Initialize memoization array with -1s // Recursive function to calculate the count of ways to cover 'dist' using memoization function printCountRecMemo (dist , memo) { // Base cases if (dist < 0) return 0 ; if (dist === 0) return 1 ; // Check if the value for 'dist' is already computed if (memo [dist] !== - 1) return memo [dist]; // Recur for all previous 3 and add the results let ways = printCountRecMemo (dist - 1 , memo) + printCountRecMemo (dist - 2 , memo) + printCountRecMemo (dist - 3 , memo); // Memoize the result for 'dist' for future use memo [dist] = ways ; return ways ; } // Call recursive function with memoization return printCountRecMemo (dist , memo); } // Driver program let dist = 4 ; console . log (countWays (dist)); Output 7 Time Complexity: O(n) Auxiliary Space: O(n) Count number of ways to cover a distance using Dynamic Programming (Tabluation): We start by initializing the base cases for covering distances 0, 1, and 2. For distance 0, there is only one way to cover it (do nothing). For distance 1, there is also only one way to cover it (take a step of size 1). For distance 2, there are two ways to cover it (take two steps of size 1 or take a step of size 2). We use a bottom-up approach to fill in the count[]

array. For each entry count[i] , we compute the number of ways to cover distance i by adding the number of ways to cover distances i-1, i-2 , and i-3. Finally, we return the value of count[dist], which represents the number of ways to cover the given distance. Below is the implementation of the above approach: C++ // A Dynamic Programming based C++ program to count number of ways // to cover a distance with 1, 2 and 3 steps #include <iostream> using namespace std ; int printCountDP (int dist) { int count [dist + 1]; // Initialize base values. There is one way to cover 0 and 1 // distances and two ways to cover 2 distance count [0] = 1 ; if (dist >= 1) count [1] = 1 ; if (dist >= 2) count [2] = 2 ; // Fill the count array in bottom up manner for (int i = 3 ; i <= dist ; i ++) count [i] = count [i -1] + count [i -2] + count [i -3]; return count [dist]; } // driver program int main () { int dist = 4 ; cout << printCountDP (dist); return 0 ; } Java // A Dynamic Programming based Java program // to count number of ways to cover a distance // with 1, 2 and 3 steps import java.io.* ; class GFG { // Function returns count of ways to cover 'dist' static int printCountDP (int dist) { int [] count = new int [dist + 1] ; // Initialize base values. There is one way to // cover 0 and 1 distances and two ways to // cover 2 distance count [0] = 1 ; if (dist >= 1) count [1] = 1 ; if (dist >= 2) count [2] = 2 ; // Fill the count array in bottom up manner for (int i = 3 ; i <= dist ; i ++) count [i] = count [i - 1] + count [i - 2] + count [i - 3] ; return count [dist] ; } // driver program public static void main (String [] args) { int dist = 4 ; System . out . println (printCountDP (dist)); } } // This code is contributed by Pramod Kumar Python # A Dynamic Programming based on Python3 # program to count number of ways to # cover a distance with 1, 2 and 3 steps def printCountDP (dist): count = [0] * (dist + 1) # Initialize base values. There is # one way to cover 0 and 1 distances # and two ways to cover 2 distance count [0] = 1 if dist >= 1 : count [1] = 1 if dist >= 2 : count [2] = 2 # Fill the count array in bottom # up manner for i in range (3 , dist + 1): count [i] = (count [i - 1] + count [i - 2] + count [i - 3]) return count [dist]; # driver program dist = 4 ; print (printCountDP (dist)) # This code is contributed by Sam007. C# // A Dynamic Programming based C# program // to count number of ways to cover a distance // with 1, 2 and 3 steps using System ; class GFG { // Function returns count of ways // to cover 'dist' static int printCountDP (int dist) { int [] count = new int [dist + 1]; // Initialize base values. There is one // way to cover 0 and 1 distances // and two ways to cover 2 distance count [0] = 1 ; count [1] = 1 ; count [2] = 2 ; // Fill the count array // in bottom up manner for (int i = 3 ; i <= dist ; i ++) count [i] = count [i - 1] + count [i - 2] + count [i - 3]; return count [dist]; } // Driver Code public static void Main () { int dist = 4 ; Console . WriteLine (printCountDP (dist)); } } // This code is contributed by Sam007. JavaScript // A Dynamic Programming based Javascript program // to count number of ways to cover a distance // with 1, 2 and 3 steps // Function returns count of ways // to cover 'dist' function printCountDP (dist) { let count = new Array (dist + 1); // Initialize base values. There is one // way to cover 0 and 1 distances // and two ways to cover 2 distance count [0] = 1 ; if (dist >= 1) count [1] = 1 ; if (dist >= 2) count [2] = 2 ; // Fill the count array // in bottom up manner for (let i = 3 ; i <= dist ; i ++) count [i] = count [i - 1] + count [i - 2] + count [i - 3]; return count [dist]; } // Driver code let dist = 4 ; console . log (printCountDP (dist)); // This code is contributed by divyeshrabadiya07 PHP <?php // A Dynamic Programming based PHP program // to count number of ways to cover a // distance with 1, 2 and 3 steps function printCountDP ($dist) { $count = array (); // Initialize base values. There is // one way to cover 0 and 1 distances // and two ways to cover 2 distance $count [0] = 1 ; $count [1] = 1 ; $count [2] = 2 ; // Fill the count array // in bottom up manner for ($i = 3 ; $i <= $dist ; $i ++) $count [$i] = $count [$i - 1] + $count [$i - 2] + $count [$i - 3]; return $count [$dist]; } // Driver Code $dist = 4 ; echo printCountDP ($dist); // This code is contributed by anuj_67. ?> Output 7 Time Complexity: O(n), Only one traversal of the array is needed. Auxiliary Space: O(n), To store the values in a DP O(n) extra space is needed. Count number of ways to cover a distance using Contant Space O(1): From the above solution, we can observe that only three previous states are required to compute the value of current state. So, creates an array of size 3, where each element represents the number of ways to reach a certain distance. The base cases are initialized to 1 for distance 0 and 1, and 2 for distance 2. The remaining elements are filled using the recurrence relation, which states that the number of ways to reach a certain distance is the sum of the number of ways to reach the previous three distances. The final answer is the number of ways to reach the given distance, which is stored in the last element of the array. Below is the implementation of the above approach: C++ // A Dynamic Programming based C++ program to count number of ways #include <iostream> using namespace std ; int printCountDP (int dist) { //Create the array of size 3. int ways [3] , n = dist ; //Initialize the bases cases ways [0] = 1 ; ways [1] = 1 ; ways [2] = 2 ; //Run a loop from 3 to n //Bottom up approach to fill the array for (int i = 3 ; i <= n ; i ++) ways [i % 3] = ways [(i -1) % 3] + ways [(i -2) % 3] + ways [(i -3) % 3]; return ways [n % 3]; } // driver program int main () { int dist = 4 ; cout << printCountDP (dist); return 0 ; } Java // A Dynamic Programming based Java

program to count number of ways import java.util.* ; class GFG { static int printCountDP (int dist) { // Create the array of size 3. int [] ways = new int [3] ; int n = dist ; // Initialize the bases cases ways [0] = 1 ; ways [1] = 1 ; ways [2] = 2 ; // Run a loop from 3 to n // Bottom up approach to fill the array for (int i = 3 ; i <= n ; i ++) ways [i % 3] = ways [(i - 1) % 3] + ways [(i - 2) % 3] + ways [(i - 3) % 3] ; return ways [n % 3] ; } // driver program public static void main (String arg []) { int dist = 4 ; System . out . print (printCountDP (dist)); } } // this code is contributed by shivanisinghss2110 Python # A Dynamic Programming based C++ program to count number of ways def prCountDP (dist): # Create the array of size 3. ways = [0] * 3 n = dist # Initialize the bases cases ways [0] = 1 ways [1] = 1 ways [2] = 2 # Run a loop from 3 to n # Bottom up approach to fill the array for i in range (3 , n + 1): ways [i % 3] = ways [(i - 1) % 3] + ways [(i - 2) % 3] + ways [(i - 3) % 3] return ways [n % 3] # driver program dist = 4 print (prCountDP (dist)) # This code is contributed by shivanisinghss2110 C# // A Dynamic Programming based C# // program to count number of ways using System ; class GFG { static int printCountDP (int dist) { // Create the array of size 3. int [] ways = new int [3]; int n = dist ; // Initialize the bases cases ways [0] = 1 ; ways [1] = 1 ; ways [2] = 2 ; // Run a loop from 3 to n // Bottom up approach to fill the array for (int i = 3 ; i <= n ; i ++) ways [i % 3] = ways [(i - 1) % 3] + ways [(i - 2) % 3] + ways [(i - 3) % 3]; return ways [n % 3]; } // Driver code public static void Main (String [] arg) { int dist = 4 ; Console . Write (printCountDP (dist)); } } // This code is contributed by shivanisinghss2110 JavaScript // A Dynamic Programming based javascript program to count number of ways function printCountDP (dist) { //Create the array of size 3. var ways = [] , n = dist ; ways . length = 3 ; //Initialize the bases cases ways [0] = 1 ; ways [1] = 1 ; ways [2] = 2 ; //Run a loop from 3 to n //Bottom up approach to fill the array for (var i = 3 ; i <= n ; i ++) ways [i % 3] = ways [(i - 1) % 3] + ways [(i - 2) % 3] + ways [(i - 3) % 3]; return ways [n % 3]; } // driver code var dist = 4 ; console . write (printCountDP (dist)); Output 7 Time Complexity: O(n). Auxiliary Space: O(1) Comment Article Tags: Article Tags: Dynamic Programming DSA Amazon