Tiling Problem - GeeksforGeeks

Source: https://www.geeksforgeeks.org/tiling-problem/

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 Tiling Problem Last Updated : 6 Nov, 2025 Given a 2 × n board and tiles of size 2 × 1 , find the total number of ways to completely fill the board using these tiles. Each tile can be placed either horizontally i.e., as a 1 x 2 tile or vertically i.e., as 2 x 1 tile. Two tiling arrangements are considered different if the placement of at least one tile differs. Examples: Input: n = 4 Output: 5 Explanation: For a 2 x 4 board, there are 5 ways. Input: n = 3 Output: 3 Explanation: For or a 2 x 3 board, there are 3 ways. Try it on GfG Practice Table of Content [Naive Approach] Using Recursion - O(2^n) Time and O(n) Space [Better Approach 1] Using Top-Down DP (Memoization) - O(n) Time and O(n) Space [Better Approach 2] Using Bottom-Up DP (Tabulation) - O(n) Time and O(n) Space [Expected Approach] Using Space Optimized DP - O(n) Time and O(1) Space [Naive Approach] Using Recursion - O(2^n) Time and O(n) Space The idea is to use Recursion and breaking the larger problem (size n) into smaller subproblems (sizes n-1 and n-2). At any point, we have two choices: Place one vertical tile (2 × 1): This tile will cover one column of the board, leaving a smaller board of size 2 × (n - 1) to be filled. Place two horizontal tiles (1 × 2): These two tiles will cover two columns together, leaving a smaller board of size 2 × (n - 2) to be filled. So, to find the total number of ways to fill a 2 × n board, we just need to sum the number of ways to fill the smaller boards obtained by these two choices. C++ //Driver Code Starts #include <iostream> #include <vector> using namespace std ; //Driver Code Ends int numberOfWays (int n) { // Base Case if (n < 0) return 0 ; if (n == 0) return 1 ; int ans = 0 ; // if one tile is placed vertically ans = numberOfWays (n -1); // if two tiles are placed horizontly. ans += numberOfWays (n -2); return ans ; } //Driver Code Starts int main () { int n = 4 ; cout << numberOfWays (n); return 0 ; } //Driver Code Ends Java //Driver Code Starts class GFG { //Driver Code Ends static int numberOfWays (int n) { // Base Case if (n < 0) return 0 ; if (n == 0) return 1 ; int ans = 0 ; // if one tile is placed vertically ans = numberOfWays (n - 1); // if two tiles are placed horizontally. ans += numberOfWays (n - 2); return ans ; } //Driver Code Starts public static void main (String [] args) { int n = 4 ; System . out . println (numberOfWays (n)); } } //Driver Code Ends Python def numberOfWays (n): # Base Case if n < 0 : return 0 if n == 0 : return 1 ans = 0 # if one tile is placed vertically ans = numberOfWays (n - 1) # if two tiles are placed horizontally. ans += numberOfWays (n - 2) return ans #Driver Code Starts if __name__ == "__main__" : n = 4 print (numberOfWays (n)) #Driver Code Ends C# //Driver Code Starts using System ; class GFG { //Driver Code Ends static int numberOfWays (int n) { // Base Case if (n < 0) return 0 ; if (n == 0) return 1 ; int ans = 0 ; // if one tile is placed vertically ans = numberOfWays (n - 1); // if two tiles are placed horizontally. ans += numberOfWays (n - 2); return ans ; } //Driver Code Starts static void Main () { int n = 4 ; Console . WriteLine (numberOfWays (n)); } } //Driver Code Ends JavaScript function numberOfWays (n) { // Base Case if (n < 0) return 0 ; if (n === 0) return 1 ; let ans = 0 ; // if one tile is placed vertically ans = numberOfWays (n - 1); // if two tiles are placed horizontally. ans += numberOfWays (n - 2); return ans ; } //Driver Code //Driver Code Starts let n = 4 ; console . log (numberOfWays (n)); //Driver Code Ends Output 5 [Better Approach 1] Using Top-Down DP (Memoization) - O(n) Time and O(n) Space In the recursive approach, we saw that many subproblems are solved repeatedly, leading to redundant computations. To handle this, we use Dynamic Programming with Memoization. We create a 1D array dp[] of size n + 1, where each element dp[i] represents the number of ways to completely fill a 2 × i board. Since there is only one changing parameter (i) in our recursive function, a 1D array is sufficient. Before solving a subproblem, we first check if it has already been computed. If yes, we return the stored value from the dp array. If not, we compute it recursively and store the result in dp[i] for future use. C++ //Driver Code Starts #include <iostream> #include <vector> using namespace std ; //Driver Code Ends int countRecur (int n , vector < int > & dp) { // Base Case if (n < 0) return 0 ; if (n == 0) return 1 ; // If value is memoized if (dp [n] != -1) return dp [n]; int ans = 0 ; // if one tile is placed vertically ans = countRecur (n -1 , dp); //if two tiles are placed horizontly. ans += countRecur (n -2 , dp); return dp [n] = ans ; } int numberOfWays (int n) { vector < int > dp (n + 1 , -1); return countRecur (n , dp); } //Driver Code Starts int main () { int n = 4 ; cout << numberOfWays (n); return 0 ; } //Driver Code Ends Java //Driver Code Starts import java.util.Arrays ; class GFG { //Driver Code Ends static int countRecur (int n , int [] dp) { // Base Case if (n < 0) return 0 ; if (n == 0) return 1 ; // If value is memoized if (dp [n] != - 1) return dp [n] ; int ans = 0 ; // if one tile is placed vertically ans = countRecur (n - 1 , dp); //if two tiles are placed horizontly. ans += countRecur (n - 2 , dp); return dp [n] = ans ; } static int numberOfWays (int n) { int [] dp = new int [n + 1] ; Arrays . fill (dp , - 1); return countRecur (n , dp); } //Driver Code Starts public static void main (String [] args) { int n = 4 ; System . out . println (numberOfWays (n)); } } //Driver Code Ends Python def countRecur (n , dp): # Base Case if n < 0 : return 0 if n == 0 : return 1 # If value is memoized if dp [n] != - 1 : return dp [n] ans = 0 # if one tile is placed vertically ans = countRecur (n - 1 , dp) # if two tiles are placed horizontly. ans += countRecur (n - 2 , dp) dp [n] = ans return dp [n] def numberOfWays (n): dp = [- 1] * (n + 1) return countRecur (n , dp) if __name__ == "__main__" : #Driver Code Starts n = 4 print (numberOfWays (n)) #Driver Code Ends C# //Driver Code Starts using System ; class GFG { //Driver Code Ends static int countRecur (int n , int [] dp) { // Base Case if (n < 0) return 0 ; if (n == 0) return 1 ; // If value is memoized if (dp [n] != - 1) return dp [n]; int ans = 0 ; // if one tile is placed vertically ans = countRecur (n - 1 , dp); //if two tiles are placed horizontly. ans += countRecur (n - 2 , dp); return dp [n] = ans ; } static int numberOfWays (int n) { int [] dp = new int [n + 1]; for (int i = 0 ; i <= n ; i ++) dp [i] = - 1 ; return countRecur (n , dp); } //Driver Code Starts static void Main () { int n = 4 ; Console . WriteLine (numberOfWays (n)); } } //Driver Code Ends JavaScript function countRecur (n , dp) { // Base Case if (n < 0) return 0 ; if (n === 0) return 1 ; // If value is memoized if (dp [n] !== - 1) return dp [n]; let ans = 0 ; // if one tile is placed vertically ans = countRecur (n - 1 , dp); //if two tiles are placed horizontly. ans += countRecur (n - 2 , dp); return dp [n] = ans ; } function numberOfWays (n) { const dp = new Array (n + 1). fill (- 1); return countRecur (n , dp); } //Driver Code //Driver Code Starts let n = 4 ; console . log (numberOfWays (n)); //Driver Code Ends Output 5 [Better Approach 2] Using Bottom-Up DP (Tabulation) - O(n) Time and O(n) Space In this approach, instead of using recursion, we build the solution iteratively from bottom to top. We create a 1D DP array dp[n+1] to store the results of already computed subproblems, since there is only one parameter — n — that changes in the recursive solution. First, we will solve some smaller problems for which we define the base cases: dp[0] = 1 — There is one way to fill a 2×0 board. dp[1] = 1 — There is only one way to fill a 2×1 board. We start from these base cases, and then we have two choices: Place one vertical tile, leaving a 2×(i–1) board - dp[i–1] Place two horizontal tiles, leaving a 2×(i–2) board - dp[i–2] After filling the DP table iteratively, dp[n] will give the total number of ways to completely fill the 2×n board C++ //Driver Code Starts #include <iostream> #include <vector> using namespace std ; //Driver Code Ends int numberOfWays (int n) { if (n == 0 || n == 1) return 1 ; // Create a 1D DP array to store results of subproblems vector < int > dp (n + 1); // Initialize base cases dp [0] = 1 ; dp [1] = 1 ; // Build the solution iteratively (Bottom-Up DP) for (int i = 2 ; i <= n ; i ++) { // Two choices: dp [i] = dp [i - 1] + dp [i - 2]; } return dp [n]; } //Driver Code Starts int main () { int n = 4 ; cout << numberOfWays (n); return 0 ; } //Driver Code Ends Java //Driver Code Starts import java.util.Arrays ; public class GFG { //Driver Code Ends static int numberOfWays (int n) { if (n == 0 || n == 1) return 1 ; // Create a 1D DP array to store results of subproblems int [] dp = new int [n + 1] ; // Initialize base cases dp [0] = 1 ; dp [1] = 1 ; // Build the solution iteratively (Bottom-Up DP) for (int i = 2 ; i <= n ; i ++) { // Two choices: dp [i] = dp [i - 1] + dp [i - 2] ; } return dp [n] ; } //Driver Code Starts public static void main (String [] args) { int n = 4 ; System . out . println (numberOfWays (n)); } } //Driver Code Ends Python def numberOfWays (n): if n == 0 or n == 1 : return 1 # Create a 1D DP array to store results of subproblems dp = [0] * (n + 1) # Initialize base cases dp [0] = 1 dp [1] = 1 # Build the solution iteratively (Bottom-Up DP) for i in range (2 , n + 1): # Two choices: dp [i] = dp [i - 1] + dp [i - 2] return dp [n] if __name__ == "__main__" : #Driver Code Starts n = 4 print (numberOfWays (n)) #Driver Code Ends C# //Driver Code Starts using System ; class GFG { //Driver Code Ends static int numberOfWays (int n) { if (n == 0 || n == 1) return 1 ; // Create a 1D DP array to store results of subproblems int [] dp = new int [n + 1]; // Initialize base cases dp [0] = 1 ; dp [1] = 1 ; // Build the solution iteratively (Bottom-Up DP) for (int i = 2 ; i <= n ; i ++) { // Two choices: dp [i] = dp [i - 1] + dp [i - 2]; } return dp [n]; } //Driver Code Starts static void Main

() { int n = 4 ; Console . WriteLine (numberOfWays (n)); } } //Driver Code Ends JavaScript function numberOfWays (n) { if (n === 0 || n === 1) return 1 ; // Create a 1D DP array to store results of subproblems let dp = new Array (n + 1); // Initialize base cases dp [0] = 1 ; dp [1] = 1 ; // Build the solution iteratively (Bottom-Up DP) for (let i = 2 ; i <= n ; i ++) { // Two choices: dp [i] = dp [i - 1] + dp [i - 2]; } return dp [n]; } //Driver Code //Driver Code Starts let n = 4 ; console . log (numberOfWays (n)); //Driver Code Ends Output 5 [Expected Approach] Using Space Optimized DP - O(n) Time and O(1) Space In the previous dynamic programming approach, we used a 1D array dp[] to store all computed results up to n. However, if we observe carefully, to calculate the current state dp[i], we only need the last two states — dp[i-1] and dp[i-2]. This means we don’t need to maintain the entire DP array; instead, we can optimize the space by storing only these two values. We can use two variables, say prev1 and prev2, where: prev1 represents the number of ways for dp[i-1] prev2 represents the number of ways for dp[i-2] This way, we iteratively move forward while using only constant space instead of an entire array. C++ //Driver Code Starts #include <iostream> #include <vector> using namespace std ; //Driver Code Ends int numberOfWays (int n) { // Base cases if (n == 0 || n == 1) return 1 ; int prev2 = 1 ; int prev1 = 1 ; // Iterate from 2 to n to build the result for (int i = 2 ; i <= n ; i ++) { // Current state depends on the sum of previous two states int curr = prev1 + prev2 ; // Move the variables one step ahead prev2 = prev1 ; prev1 = curr ; } return prev1 ; } //Driver Code Starts int main () { int n = 4 ; cout << numberOfWays (n); return 0 ; } //Driver Code Ends Java //Driver Code Starts import java.util.Arrays ; class GFG { //Driver Code Ends static int numberOfWays (int n) { // Base cases if (n == 0 || n == 1) return 1 ; int prev2 = 1 ; int prev1 = 1 ; // Iterate from 2 to n to build the result for (int i = 2 ; i <= n ; i ++) { // Current state depends on the sum of previous two states int curr = prev1 + prev2 ; // Move the variables one step ahead prev2 = prev1 ; prev1 = curr ; } return prev1 ; } //Driver Code Starts public static void main (String [] args) { int n = 4 ; System . out . println (numberOfWays (n)); } } //Driver Code Ends Python def numberOfWays (n): # Base cases if n == 0 or n == 1 : return 1 prev2 = 1 prev1 = 1 # Iterate from 2 to n to build the result for i in range (2 , n + 1): # Current state depends on the sum of previous two states curr = prev1 + prev2 # Move the variables one step ahead prev2 = prev1 prev1 = curr return prev1 #Driver Code Starts if __name__ == "__main__" : n = 4 print (numberOfWays (n)) #Driver Code Ends C# //Driver Code Starts using System ; class GFG { //Driver Code Ends static int numberOfWays (int n) { // Base cases if (n == 0 || n == 1) return 1 ; int prev2 = 1 ; int prev1 = 1 ; // Iterate from 2 to n to build the result for (int i = 2 ; i <= n ; i ++) { // Current state depends on the sum of previous two states int curr = prev1 + prev2 ; // Move the variables one step ahead prev2 = prev1 ; prev1 = curr ; } return prev1 ; } //Driver Code Starts public static void Main () { int n = 4 ; Console . WriteLine (numberOfWays (n)); } } //Driver Code Ends JavaScript function numberOfWays (n) { // Base cases if (n === 0 || n === 1) return 1 ; let prev2 = 1 ; let prev1 = 1 ; // Iterate from 2 to n to build the result for (let i = 2 ; i <= n ; i ++) { // Current state depends on the sum of previous two states let curr = prev1 + prev2 ; // Move the variables one step ahead prev2 = prev1 ; prev1 = curr ; } return prev1 ; } // Driver code //Driver Code Starts let n = 4 ; console . log (numberOfWays (n)); //Driver Code Ends Output 5 Comment Article Tags: Article Tags: Dynamic Programming Mathematical DSA Amazon Fibonacci + 1 More