Binomial Coefficient - GeeksforGeeks

Source: https://www.geeksforgeeks.org/binomial-coefficient-dp-9/

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 Binomial Coefficient Last Updated : 23 Jul, 2025 Given an integer values n and k, the task is to find the value of Binomial Coefficient C(n, k). A binomial coefficient C(n, k) can be defined as the coefficient of x^k in the expansion of (1 + x)^n. A binomial coefficient C(n, k) also gives the number of ways, disregarding order, that k objects can be chosen from among n objects more formally, the number of k-element subsets (or k-combinations) of a n-element set. Examples Input: n = 4, k = 2 Output: 6 Explanation: The value of 4C2 is (4 × 3) / (2 × 1) = 6 . Input: n = 5, k = 2 Output: 10 Explanation: The value of 5C2 is (5 × 4) / (2 × 1) = 10 . Input: n = 6, k = 3 Output: 20 Explanation: The value of 6C3 is (6 × 5 × 4) / (3 × 2 × 1) = 20 . Try it on GfG Practice Table of Content Using recursion - O(2 ^ n) Time and O(n) Space Top-Down DP (Memoization) - O(n * k) Time and O(n * k) Space Using Bottom-Up DP (Tabulation) - O(n * k) Time and O(n * k) Space Using Space Optimized DP - O(n * k) Time and O(k) Space Using recursion - O(2 ^ n) Time and O(n) Space The idea is to use recursion to find C(n, k). The value of C(n, k) can be recursively calculated using the following standard formula for Binomial Coefficients . C(n, k) = C(n-1, k-1) + C(n-1, k) . C(n, 0) = C(n, n) = 1. So we just need to make recursive calls of C(n-1, k-1) and C(n - 1, k). The base conditions will be when k = 0 or value of k and n be equal. C++ // C++ implementation to find // Binomial Coefficient using recursion #include <bits/stdc++.h> using namespace std ; // Returns value of Binomial Coefficient C(n, k) int binomialCoeff (int n , int k) { // k can not be grater then k so we return 0 here if (k > n) return 0 ; // base condition when k and n are equal or k = 0 if (k == 0 || k == n) return 1 ; // Recurvie add the value return binomialCoeff (n - 1 , k - 1) + binomialCoeff (n - 1 , k); } int main () { int n = 5 , k = 2 ; cout << binomialCoeff (n , k); return 0 ; } C // C implementation to find // Binomial Coefficient using recursion #include <stdio.h> // Returns value of Binomial Coefficient C(n, k) int binomialCoeff (int n , int k) { // k can not be grater then k so we return 0 here if (k > n) return 0 ; // base condition when k and n are equal or k = 0 if (k == 0 || k == n) return 1 ; // Recursive add the value return binomialCoeff (n - 1 , k - 1) + binomialCoeff (n - 1 , k); } int main () { int n = 5 , k = 2 ; printf ("%d" , binomialCoeff (n , k)); return 0 ; } Java // Java implementation to find // Binomial Coefficient using recursion class GfG { // Returns value of Binomial Coefficient C(n, k) static int binomialCoeff (int n , int k) { // k can not be grater then k so we // return 0 here if (k > n) return 0 ; // base condition when k and n are // equal or k = 0 if (k == 0 || k == n) return 1 ; // Recursive add the value return binomialCoeff (n - 1 , k - 1) + binomialCoeff (n - 1 , k); } public static void main (String [] args) { int n = 5 , k = 2 ; System . out . println (binomialCoeff (n , k)); } } Python # Python implementation to find # Binomial Coefficient using recursion # Returns value of Binomial Coefficient C(n, k) def binomialCoeff (n , k): # k can not be grater then k so we # return 0 here if k > n : return 0 # base condition when k and n are equal # or k = 0 if k == 0 or k == n : return 1 # Recursive add the value return binomialCoeff (n - 1 , k - 1) + binomialCoeff (n - 1 , k) n = 5 k = 2 print (binomialCoeff (n , k)) C# // C# implementation to find // Binomial Coefficient using recursion using System ; class GfG { // Returns value of Binomial Coefficient C(n, k) static int BinomialCoeff (int n , int k) { // k can not be grater then k so we // return 0 here if (k > n) return 0 ; // base condition when k and n are // equal or k = 0 if (k == 0 || k == n) return 1 ; // Recursive add the value return BinomialCoeff (n - 1 , k - 1) + BinomialCoeff (n - 1 , k); } static void Main (string [] args) { int n = 5 , k = 2 ; Console . WriteLine (BinomialCoeff (n , k)); } } JavaScript // Javascript implementation to find // Binomial Coefficient using recursion // Returns value of Binomial Coefficient C(n, k) function binomialCoeff (n , k) { // k can not be grater then k so we // return 0 here if (k > n) return 0 ; // base condition when k and n are equal // or k = 0 if (k === 0 || k === n) return 1 ; //

Recursive add the value return binomialCoeff (n - 1 , k - 1) + binomialCoeff (n - 1 , k); } let n = 5 , k = 2 ; console . log (binomialCoeff (n , k)); Output 10 Top-Down DP (Memoization) - O(n * k) Time and O(n * k) Space It should be noted that the above function computes the same subproblems again and again. And have two properties of Dynamic Programming : 1. Optimal Substructure : The value of C(n, k) depends on the optimal solutions of the subproblems C(n-1, k-1) and C(n-1, k) . By adding these optimal substrutures, we can efficiently calculate the total value of C(n, k). 2. Overlapping Subproblems : While applying a recursive approach in this problem, we notice that certain subproblems are computed multiple times. Recursion tree for n = 5 and k = 2. The function C(3, 1) is called two times. For large values of n, there will be many common subproblems. The Binomial Coefficient C(n, k) is computed recursively, but to avoid redundant calculations, dynamic programming with memoization is used. A 2D table stores previously computed values, allowing efficient lookups instead of recalculating. If a value is already computed, it is returned directly; otherwise, it is computed recursively and stored for future use. C++ // C++ implementation to find // Binomial Coefficient using memoization #include <bits/stdc++.h> using namespace std ; // Returns value of Binomial Coefficient C(n, k) int getnCk (int n , int k , vector < vector < int >> & memo) { // k can not be grater then k so we // return 0 here if (k > n) return 0 ; // base condition when k and n are // equal or k = 0 if (k == 0 || k == n) return 1 ; // Check if pair n and k is already // calculated then return it from here if (memo [n][k] != -1) return memo [n][k]; // Recurvie add the value and store to memorize table return memo [n][k] = getnCk (n - 1 , k - 1 , memo) + getnCk (n - 1 , k , memo); } int binomialCoeff (int n , int k) { // Create table for memorization vector < vector < int >> memo (n + 1 , vector < int > (k + 1 , -1)); return getnCk (n , k , memo); } int main () { int n = 5 , k = 2 ; cout << binomialCoeff (n , k); return 0 ; } Java // Java implementation to find // Binomial Coefficient using memoization import java.util.Arrays ; class GfG { // Returns value of Binomial Coefficient C(n, k) static int getnCk (int n , int k , int [][] memo) { // k cannot be greater than n so we return 0 here if (k > n) return 0 ; // base condition when k and n are equal or k = 0 if (k == 0 || k == n) return 1 ; // Check if pair n and k is already // calculated then return it from here if (memo [n][k] != - 1) return memo [n][k] ; // Recursive add the value and store to memo table return memo [n][k] = getnCk (n - 1 , k - 1 , memo) + getnCk (n - 1 , k , memo); } static int binomialCoeff (int n , int k) { // Create table for memoization int [][] memo = new int [n + 1][k + 1] ; for (int [] row : memo) Arrays . fill (row , - 1); return getnCk (n , k , memo); } public static void main (String [] args) { int n = 5 , k = 2 ; System . out . println (binomialCoeff (n , k)); } } Python # Python implementation to find # Binomial Coefficient using memoization def getnCk (n , k , memo): # k cannot be greater than n so we return 0 here if k > n : return 0 # base condition when k and n are equal or k = 0 if k == 0 or k == n : return 1 # Check if pair n and k is already # calculated then return it from here if memo [n][k] != - 1 : return memo [n][k] # Recursive add the value and store to memo table memo [n][k] = getnCk (n - 1 , k - 1 , memo) + \ getnCk (n - 1 , k , memo) return memo [n][k] def binomialCoeff (n , k): # Create table for memoization memo = [[- 1 for _ in range (k + 1)] for _ in range (n + 1)] return getnCk (n , k , memo) n , k = 5 , 2 print (binomialCoeff (n , k)) C# // C# implementation to find // Binomial Coefficient using memoization using System ; class GfG { // Returns value of Binomial // Coefficient C(n, k) static int GetnCk (int n , int k , int [,] memo) { // k cannot be greater than n so we // return 0 here if (k > n) return 0 ; // base condition when k and n are // equal or k = 0 if (k == 0 || k == n) return 1 ; // Check if pair n and k is already // calculated then return it from here if (memo [n , k] != - 1) return memo [n , k]; // Recursive add the value and store to memo table return memo [n , k] = GetnCk (n - 1 , k - 1 , memo) + GetnCk (n - 1 , k , memo); } static int BinomialCoeff (int n , int k) { // Create table for memoization int [,] memo = new int [n + 1 , k + 1]; for (int i = 0 ; i <= n ; i ++) for (int j = 0 ; j <= k ; j ++) memo [i , j] = - 1 ; return GetnCk (n , k , memo); } static void Main () { int n = 5 , k = 2 ; Console . WriteLine (BinomialCoeff (n , k)); } } JavaScript // Javascript implementation to find // Binomial Coefficient using memoization function getnCk (n , k , memo) { // k cannot be greater than n so we // return 0 here if (k > n) return 0 ; // base condition when k and n are // equal or k = 0 if (k === 0 || k === n) return 1 ; // Check if pair n and k is already // calculated then return it from here if (memo [n][k] !== - 1) return memo [n][k]; // Recursive add the value and store to memo table return memo [n][k] = getnCk (n - 1 , k - 1 , memo) + getnCk (n - 1 , k , memo); } function binomialCoeff (n , k) { // Create table for memoization const memo = Array . from ({ length : n + 1 }, () => Array (k + 1). fill (- 1)); return getnCk (n , k , memo); } const n = 5 , k = 2 ; console . log (binomialCoeff (n , k)); Output 10 Using Bottom-Up DP (Tabulation) - O(n * k) Time and O(n * k) Space The approach is similar to the previous one. just instead of breaking down the problem recursively , we iteratively build up the solution by calculating in bottom-up manner. Maintain a dp[][] table such that dp[i][j] stores the count all unique possible paths to reach the cell (i, j) . Base Case: For i = j and 0 <= i <= n , dp[i][j] = 1 for j = 0 and 0 <= j

<= min(i, k) , dp[i][j] = 1 Recursive Case: For i > 1 and j > 1 , dp[i][j] = dp[i-1][j-1] + dp[i-1][j] C++ // C++ implementation to find // Binomial Coefficient using tabulation #include <bits/stdc++.h> using namespace std ; // Returns value of Binomial Coefficient C(n, k) int binomialCoeff (int n , int k) { vector < vector < int >> dp (n + 1 , vector < int > (k + 1)); // Calculate value of Binomial Coefficient // in bottom up manner for (int i = 0 ; i <= n ; i ++) { for (int j = 0 ; j <= min (i , k); j ++) { // Base Cases if (j == 0 || j == i) dp [i][j] = 1 ; // Calculate value using previously // stored values else dp [i][j] = dp [i - 1][j - 1] + dp [i - 1][j]; } } return dp [n][k]; } int main () { int n = 5 , k = 2 ; cout << binomialCoeff (n , k); } C // C implementation to find // Binomial Coefficient using tabulation #include <stdio.h> // Returns value of Binomial Coefficient C(n, k) int binomialCoeff (int n , int k) { int dp [n + 1][k + 1]; // Calculate value of Binomial Coefficient // in bottom up manner for (int i = 0 ; i <= n ; i ++) { for (int j = 0 ; j <= (i < k ? i : k); j ++) { if (j == 0 || j == i) dp [i][j] = 1 ; // Calculate value using previously // stored values else dp [i][j] = dp [i - 1][j - 1] + dp [i - 1][j]; } } return dp [n][k]; } int main () { int n = 5 , k = 2 ; printf ("%d" , binomialCoeff (n , k)); return 0 ; } Java // Java implementation to find // Binomial Coefficient using tabulation class GfG { // Returns value of Binomial Coefficient C(n, k) static int binomialCoeff (int n , int k) { int [][] dp = new int [n + 1][k + 1] ; // Calculate value of Binomial Coefficient // in bottom up manner for (int i = 0 ; i <= n ; i ++) { for (int j = 0 ; j <= Math . min (i , k); j ++) { if (j == 0 || j == i) dp [i][j] = 1 ; // Calculate value using previously // stored values else dp [i][j] = dp [i - 1][j - 1] + dp [i - 1][j] ; } } return dp [n][k] ; } public static void main (String [] args) { int n = 5 , k = 2 ; System . out . println (binomialCoeff (n , k)); } } Python # Python implementation to find # Binomial Coefficient using tabulation # Returns value of Binomial Coefficient C(n, k) def binomialCoeff (n , k): dp = [[0 for _ in range (k + 1)] for _ in range (n + 1)] # Calculate value of Binomial Coefficient # in bottom up manner for i in range (n + 1): for j in range (min (i , k) + 1): # Base Cases if j == 0 or j == i : dp [i][j] = 1 # Calculate value using previously # stored values else : dp [i][j] = dp [i - 1][j - 1] + dp [i - 1][j] return dp [n][k] n = 5 k = 2 print (binomialCoeff (n , k)) C# // C3 implementation to find // Binomial Coefficient using tabulation using System ; class GfG { // Returns value of Binomial Coefficient C(n, k) public static int BinomialCoeff (int n , int k) { int [,] dp = new int [n + 1 , k + 1]; // Calculate value of Binomial Coefficient // in bottom up manner for (int i = 0 ; i <= n ; i ++) { for (int j = 0 ; j <= Math . Min (i , k); j ++) { // Base Cases if (j == 0 || j == i) dp [i , j] = 1 ; // Calculate value using previously // stored values else dp [i , j] = dp [i - 1 , j - 1] + dp [i - 1 , j]; } } return dp [n , k]; } static void Main (string [] args) { int n = 5 , k = 2 ; Console . WriteLine (BinomialCoeff (n , k)); } } JavaScript // Javascript implementation to find // Binomial Coefficient using tabulation // Returns value of Binomial Coefficient C(n, k) function binomialCoeff (n , k) { let dp = Array . from ({ length : n + 1 }, () => Array (k + 1). fill (0)); // Calculate value of Binomial Coefficient // in bottom up manner for (let i = 0 ; i <= n ; i ++) { for (let j = 0 ; j <= Math . min (i , k); j ++) { // Base Cases if (j === 0 || j === i) { dp [i][j] = 1 ; } // Calculate value using previously // stored values else { dp [i][j] = dp [i - 1][j - 1] + dp [i - 1][j]; } } } return dp [n][k]; } let n = 5 , k = 2 ; console . log (binomialCoeff (n , k)); Output 10 Using Space Optimized DP - O(n * k) Time and O(k) Space In the previous approach using dynamic programming , we derived a relation between states as follows: dp[i][j] = dp[i-1][j-1]+dp[i-1][j] We do not need to maitain whole matrix for this. We can just maintain one array of length k and add dp[j-1] every time to dp[j]; Use a 1D array dp[] of size k+1 to store binomial coefficients, reducing space complexity to O(k) . Set dp[0] = 1, representing nC0=1. Update dp[j] in reverse order, using the previous values from the same array. Each entry dp[j] is updated as dp[j] + dp[j-1] for each row. The final value of dp(n,k) is stored in dp[k], and returned. C++ // C++ program for space optimized Dynamic Programming // Solution of Binomial Coefficient #include <bits/stdc++.h> using namespace std ; int binomialCoeff (int n , int k) { vector < int > dp (k + 1); // nC0 is 1 dp [0] = 1 ; for (int i = 1 ; i <= n ; i ++) { // Compute next row of pascal triangle using // the previous row for (int j = min (i , k); j > 0 ; j --) dp [j] = dp [j] + dp [j - 1]; } return dp [k]; } int main () { int n = 5 , k = 2 ; cout << binomialCoeff (n , k); return 0 ; } Java // Java program for space optimized Dynamic Programming // Solution of Binomial Coefficient class GfG { // Returns value of Binomial Coefficient C(n, k) static int binomialCoeff (int n , int k) { int [] dp = new int [k + 1] ; // nC0 is 1 dp [0] = 1 ; for (int i = 1 ; i <= n ; i ++) { // Compute next row of pascal triangle using // the previous row for (int j = Math . min (i , k); j > 0 ; j --) dp [j] = dp [j] + dp [j - 1] ; } return dp [k] ; } public static void main (String [] args) { int n = 5 , k = 2 ; System . out . println (binomialCoeff (n , k)); } } Python # Python program for space optimized Dynamic Programming # Solution of Binomial Coefficient def binomialCoeff (n , k): dp = [0] * (k + 1) # nC0 is 1 dp [0] = 1 for i in range (1 , n + 1): # Compute next row of pascal triangle using # the previous row for j in range (min (i , k), 0 , - 1): dp [j] = dp [j] + dp [j - 1] return dp [k] n = 5 k = 2 print (binomialCoeff (n , k)) C# // C# program for space optimized Dynamic Programming // Solution of Binomial Coefficient using System ; class GfG { // Returns value of

Binomial Coefficient C(n, k) static int BinomialCoeff (int n , int k) { int [] dp = new int [k + 1]; // nC0 is 1 dp [0] = 1 ; for (int i = 1 ; i <= n ; i ++) { // Compute next row of pascal triangle using // the previous row for (int j = Math . Min (i , k); j > 0 ; j --) dp [j] = dp [j] + dp [j - 1]; } return dp [k]; } static void Main (string [] args) { int n = 5 , k = 2 ; Console . WriteLine (BinomialCoeff (n , k)); } } JavaScript // JavaScript program for space optimized Dynamic // Programming Solution of Binomial Coefficient function binomialCoeff (n , k) { let dp = Array (k + 1). fill (0); // nC0 is 1 dp [0] = 1 ; for (let i = 1 ; i <= n ; i ++) { // Compute next row of pascal triangle using // the previous row for (let j = Math . min (i , k); j > 0 ; j --) { dp [j] = dp [j] + dp [j - 1]; } } return dp [k]; } let n = 5 , k = 2 ; console . log (binomialCoeff (n , k)); Output 10 Related articles: Space and time efficient Binomial Coefficient Program to calculate value of nCr Comment Article Tags: Article Tags: Dynamic Programming Mathematical DSA binomial coefficient