Euclidean algorithms (Basic and Extended) - GeeksforGeeks

Source: https://www.geeksforgeeks.org/euclidean-algorithms-basic-and-extended/

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 Euclidean algorithms (Basic and Extended) Last Updated : 17 Feb, 2025 The Euclidean algorithm is a way to find the greatest common divisor of two positive integers. GCD of two numbers is the largest number that divides both of them. A simple way to find GCD is to factorize both numbers and multiply common prime factors. Examples: input: a = 12, b = 20 Output: 4 Explanation: The Common factors of (12, 20) are 1, 2, and 4 and greatest is 4. input: a = 18, b = 33 Output: 3 Explanation: The Common factors of (18, 33) are 1 and 3 and greatest is 3. Try it on GfG Practice Table of Content Basic Euclidean Algorithm for GCD Extended Euclidean Algorithm How does Extended Algorithm Work? How is Extended Algorithm Useful? Basic Euclidean Algorithm for GCD The algorithm is based on the below facts. If we subtract a smaller number from a larger one (we reduce a larger number), GCD doesn't change. So if we keep subtracting repeatedly the larger of two, we end up with GCD. Now instead of subtraction, if we divide the larger number, the algorithm stops when we find the remainder 0. CPP // C++ program to demonstrate working of // extended Euclidean Algorithm #include <bits/stdc++.h> using namespace std ; // Function to return // gcd of a and b int findGCD (int a , int b) { if (a == 0) return b ; return findGCD (b % a , a); } int main () { int a = 35 , b = 15 ; int g = findGCD (a , b); cout << g << endl ; return 0 ; } C // C program to demonstrate working of // extended Euclidean Algorithm #include <stdio.h> // Function to return // gcd of a and b int findGCD (int a , int b) { if (a == 0) return b ; return findGCD (b % a , a); } int main () { int a = 35 , b = 15 ; int g = findGCD (a , b); printf ("%d \n " , g); return 0 ; } Java // Java program to demonstrate working of // extended Euclidean Algorithm class GFG { // Function to return // gcd of a and b static int findGCD (int a , int b) { if (a == 0) return b ; return findGCD (b % a , a); } public static void main (String [] args) { int a = 35 , b = 15 ; int g = findGCD (a , b); System . out . println (g); } } Python # Python program to demonstrate working of # extended Euclidean Algorithm # Function to return # gcd of a and b def findGCD (a , b): if a == 0 : return b return findGCD (b % a , a) # Main function def main (): a , b = 35 , 15 g = findGCD (a , b) print (g) if __name__ == "__main__" : main () C# // C# program to demonstrate working of // extended Euclidean Algorithm using System ; class GFG { // Function to return // gcd of a and b static int FindGCD (int a , int b) { if (a == 0) return b ; return FindGCD (b % a , a); } public static void Main () { int a = 35 , b = 15 ; int g = FindGCD (a , b); Console . WriteLine (g); } } JavaScript // JavaScript program to demonstrate working of // extended Euclidean Algorithm // Function to return // gcd of a and b function findGCD (a , b) { if (a === 0) return b ; return findGCD (b % a , a); } function main () { let a = 35 , b = 15 ; let g = findGCD (a , b); console . log (g); } // Run the main function main (); Output GCD(10, 15) = 5 GCD(35, 10) = 5 GCD(31, 2) = 1 Time Complexity: O(log min(a, b)) Auxiliary Space: O(log (min(a,b))) Extended Euclidean Algorithm Extended Euclidean algorithm also finds integer coefficients x and y such that: ax + by = gcd(a, b) Examples: Input: a = 30, b = 20 Output: gcd = 10, x = 1, y = -1 Explanation: 30*1 + 20*(-1) = 10 Input: a = 35, b = 15 Output: gcd = 5, x = 1, y = -2 Explanation: 35*1 + 15*(-2) = 5 The extended Euclidean algorithm updates the results of gcd(a, b) using the results calculated by the recursive call gcd(b%a, a). Let values of x and y calculated by the recursive call be x 1 and y 1 . x and y are updated using the below expressions. ax + by = gcd(a, b) gcd(a, b) = gcd(b%a, a) gcd(b%a, a) = (b%a)x 1 + ay 1 ax + by = (b%a)x 1 + ay 1 ax + by = (b - [b/a] * a)x 1 + ay 1 ax + by = a(y 1 - [b/a] * x 1) + bx 1 Comparing LHS and RHS, x = y 1 - \lfloor b/a \rfloor * x

1 y = x 1 C++ // C++ program to demonstrate working of // extended Euclidean Algorithm #include <bits/stdc++.h> using namespace std ; // Function for extended Euclidean Algorithm int gcdExtended (int a , int b , int & x , int & y) { // Base Case if (a == 0) { x = 0 ; y = 1 ; return b ; } int x1 , y1 ; int gcd = gcdExtended (b % a , a , x1 , y1); // Update x and y using results of // recursive call x = y1 - (b / a) * x1 ; y = x1 ; return gcd ; } int findGCD (int a , int b) { int x = 1 , y = 1 ; return gcdExtended (a , b , x , y); } int main () { int a = 35 , b = 15 ; int g = findGCD (a , b); cout << g << endl ; return 0 ; } C // C program to demonstrate working of // extended Euclidean Algorithm #include <stdio.h> // Function for extended Euclidean Algorithm int gcdExtended (int a , int b , int * x , int * y) { // Base Case if (a == 0) { * x = 0 ; * y = 1 ; return b ; } int x1 , y1 ; int gcd = gcdExtended (b % a , a , & x1 , & y1); // Update x and y using results of // recursive call * x = y1 - (b / a) * x1 ; * y = x1 ; return gcd ; } int findGCD (int a , int b) { int x = 1 , y = 1 ; return gcdExtended (a , b , & x , & y); } int main () { int a = 35 , b = 15 ; int g = findGCD (a , b); printf ("%d \n " , g); return 0 ; } Java // Java program to demonstrate working of // extended Euclidean Algorithm class GFG { // Function for extended Euclidean Algorithm static int gcdExtended (int a , int b , int [] x , int [] y) { // Base Case if (a == 0) { x [0] = 0 ; y [0] = 1 ; return b ; } int [] x1 = { 0 }, y1 = { 0 }; int gcd = gcdExtended (b % a , a , x1 , y1); // Update x and y using results of // recursive call x [0] = y1 [0] - (b / a) * x1 [0] ; y [0] = x1 [0] ; return gcd ; } static int findGCD (int a , int b) { int [] x = { 1 }, y = { 1 }; return gcdExtended (a , b , x , y); } public static void main (String [] args) { int a = 35 , b = 15 ; int g = findGCD (a , b); System . out . println (g); } } Python # Python program to demonstrate working of # extended Euclidean Algorithm # Function for extended Euclidean Algorithm def gcdExtended (a , b , x , y): # Base Case if a == 0 : x [0] = 0 y [0] = 1 return b x1 , y1 = [0], [0] gcd = gcdExtended (b % a , a , x1 , y1) # Update x and y using results of # recursive call x [0] = y1 [0] - (b // a) * x1 [0] y [0] = x1 [0] return gcd def findGCD (a , b): x , y = [1], [1] return gcdExtended (a , b , x , y) # Main function def main (): a , b = 35 , 15 g = findGCD (a , b) print (g) if __name__ == "__main__" : main () C# // C# program to demonstrate working of // extended Euclidean Algorithm using System ; class GFG { // Function for extended Euclidean Algorithm static int GcdExtended (int a , int b , ref int x , ref int y) { // Base Case if (a == 0) { x = 0 ; y = 1 ; return b ; } int x1 = 0 , y1 = 0 ; int gcd = GcdExtended (b % a , a , ref x1 , ref y1); // Update x and y using results of // recursive call x = y1 - (b / a) * x1 ; y = x1 ; return gcd ; } static int FindGCD (int a , int b) { int x = 1 , y = 1 ; return GcdExtended (a , b , ref x , ref y); } public static void Main () { int a = 35 , b = 15 ; int g = FindGCD (a , b); Console . WriteLine (g); } } JavaScript // JavaScript program to demonstrate working of // extended Euclidean Algorithm // Function for extended Euclidean Algorithm function gcdExtended (a , b , x , y) { // Base Case if (a === 0) { x [0] = 0 ; y [0] = 1 ; return b ; } let x1 = [0], y1 = [0]; let gcd = gcdExtended (b % a , a , x1 , y1); // Update x and y using results of // recursive call x [0] = y1 [0] - Math . floor (b / a) * x1 [0]; y [0] = x1 [0]; return gcd ; } function findGCD (a , b) { let x = [1], y = [1]; return gcdExtended (a , b , x , y); } // Main function function main () { let a = 35 , b = 15 ; let g = findGCD (a , b); console . log (g); } // Run the main function main (); Output 5 Time Complexity: O(log min(a, b)) Auxiliary Space: O(log (min(a,b))) How does Extended Algorithm Work? As seen above, x and y are results for inputs a and b, a.x + b.y = gcd ----(1) And x 1 and y 1 are results for inputs b%a and a (b%a).x 1 + a.y 1 = gcd When we put b%a = (b - (\lfloor b/a \rfloor).a) in above, we get following. Note that \lfloor b/a\rfloor is floor(b/a) (b - (\lfloor b/a \rfloor).a).x1 + a.y1 = gcd Above equation can also be written as below b.x1 + a.(y1 - (\lfloor b/a \rfloor).x1) = gcd ---(2) After comparing coefficients of 'a' and 'b' in (1) and (2), we get following, x = y1 - \lfloor b/a \rfloor * x1 y = x 1 How is Extended Algorithm Useful? The extended Euclidean algorithm is particularly useful when a and b are coprime (or gcd is 1). Since x is the modular multiplicative inverse of "a modulo b", and y is the modular multiplicative inverse of "b modulo a". In particular, the computation of the modular multiplicative inverse is an essential step in RSA public-key encryption method. Comment Article Tags: Article Tags: Mathematical DSA GCD-LCM