Basic and Extended Euclidean algorithms

Basic Euclidean Algorithm is used to find GCD of two numbers say a and b. Below is a recursive C function to evaluate gcd using Euclid's algorithm.

```
// C program to demonstrate Basic Euclidean Algorithm
#include <stdio.h>

// Function to return gcd of a and b
int gcd(int a, int b)
{
    if (a == 0)
        return b;
    return gcd(b%a, a);
}

// Driver program to test above function
int main()
{
    int a = 10, b = 15;
    printf("GCD(%d, %d) = %d\n", a, b, gcd(a, b));
    a = 35, b = 10;
    printf("GCD(%d, %d) = %d\n", a, b, gcd(a, b));
    a = 31, b = 2;
    printf("GCD(%d, %d) = %d\n", a, b, gcd(a, b));
    return 0;
}
```

Run on IDE

Output:

```
GCD(10, 15) = 5
GCD(35, 10) = 5
GCD(31, 2) = 1
```

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
(Note that 30*1 + 20*(-1) = 10)

Input: a = 35, b = 15
Output: gcd = 5
        x = 1, y = -2
(Note that 10*0 + 5*1 = 5)
```

The extended Euclidean algorithm updates results of gcd(a, b) using the results calculated by 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 below expressions.

$$x = y_1 - \lfloor b/a \rfloor * x_1$$
$$y = x_1$$

Below is C implementation based on above formulas.

```c
// C program to demonstrate working of extended
// Euclidean Algorithm
#include <stdio.h>

// C 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; // To store results of recursive call
    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;
}

// Driver Program
int main()
{
    int x, y;
    int a = 35, b = 15;
    int g = gcdExtended(a, b, &x, &y);
    printf("gcd(%d, %d) = %d, x = %d, y = %d",
           a, b, g, x, y);
    return 0;
}
```

Run on IDE

Output:

```
gcd(35, 15) = 5, x = 1, y = -2
```

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 x1 and y1 are results for inputs b%a and a
   (b%a).x1 + a.y1 = gcd

When we put b%a = (b - (⌊b/a⌋).a) in above,
we get following. Note that ⌊b/a⌋ is floor(a/b)

   (b - (⌊b/a⌋).a).x1 + a.y1  = gcd

Above equation can also be written as below
   b.x1 + a.(y1 - (⌊b/a⌋).x1) = gcd      ---(2)

After comparing coefficients of 'a' and 'b' in (1) and
(2), we get following
   x = y1 - ⌊b/a⌋ * x1
   y = x1
```

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.

References:

http://e-maxx.ru/algo/extended_euclid_algorithm
http://en.wikipedia.org/wiki/Euclidean_algorithm
http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

This article is contributed by **Ankur**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

2 Comments Category: Mathematical Tags: MathematicalAlgo

Related Posts:

- Program for Rank of Matrix
- Primality Test | Set 3 (Miller–Rabin)
- Chinese Remainder Theorem | Set 2 (Inverse Modulo based Implementation)
- Euclid's lemma
- Chinese Remainder Theorem | Set 1 (Introduction)
- Compute nCr % p | Set 2 (Lucas Theorem)
- Compute nCr % p | Set 1 (Introduction and Dynamic Programming Solution)
- Fibonacci Coding

(Login to Rate and Mark)

0 Average Difficulty : **0/5.0**
No votes yet.

- [ ] Add to TODO List
- [ ] Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

2 Comments GeeksforGeeks 1 Login

Recommend Share Sort by Newest

Join the discussion…

Mak • 7 months ago

'When we put b%a = (b - (⌊b/a⌋).a) in above'

can anyone explain why we put this ?

• Reply • Share ›

Vishal Arya → Mak • 4 months ago

Dividend = (Divisor * Quotient) + Remainder
Fill this equation when we divide b by a.
b = (⌊b/a⌋*a) + b%a

• Reply • Share ›