Number Theory

Content

Factors of a number

Introduction

A Naive Solution would be to iterate all the numbers from 1 to n, checking if that number divides n and printing it.

Implementation

Visit the link - https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Number%20Theory/Codes/Factors/findFactorsOfN_naiveApproach.cpp

Time Complexity - O(n)

Space Complexity - O(1)

Can we improve the above solution?

If we look carefully, all the divisors are present in pairs. For example if n = 100, then the various pairs of divisors are: (1,100), (2,50), (4,25), (5,20), (10,10)

Using this fact we could speed up our program significantly.

We, however, have to be careful if there are two equal divisors as in the case of (10, 10). In such a case, we'd print only one of them.

Implementation

Visit the link -

https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Number%20Theory/Codes/Factors/findFactorsOfN_betterApproach.cpp

Time Complexity - O(sqrt(n))

Space Complexity- O(1)

Water and Jug Problem

You are given two jugs with capacities jug1Capacity and jug2Capacity liters. There is an infinite amount of water supply available. Determine whether it is possible to measure exactly targetCapacity liters using these two jugs.

If targetCapacity liters of water are measurable, you must have targetCapacity liters of water contained within one or both buckets by the end.

Operations allowed:

- Fill any of the jugs with water.
- Empty any of the jugs.
- Pour water from one jug into another till the other jug is completely full, or the first jug itself is empty.

Approach

Solution exists onlye for integer values of X, Y in aX + bY = c

where a =Jug1Cap, b = jug2Cap, c= target Capacity

We simply have to find GCD of both jug capacity and have to check whether the gcd is divisible by targetCapacity or not.

Implementation

Visit link -

https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Number%20Theory/Codes/Misc/waterAndJug.cpp

Euclid's algorithm for GCD of 1 numbers

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.

- If we subtract a smaller number from a larger (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 smaller number, the algorithm stops when we find remainder 0
- Implementation Visit the link - https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Number%20Theory/Codes/Misc/gcdUsingEuclid.cpp
- Time Complexity - O(Log min(a, b))

Extended Euclidean Algorithm

Introduction

Extended Euclidean algorithm also finds integer coefficients x and y such that:

$ax + by = gcd(a, b)$

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 x1 and y1. x and y are updated using the below expressions.

$x = y1 - \lfloor b/a \rfloor * x1$

$y = x1$

Implementation

Visit the link - https://github.com/lavishabhambri/Weekly-Algo-Newsletter/blob/main/Number%20Theory/Codes/Misc/gcdUsingExtendedEuclid.cpp

How does Extended Algorithm Work?

As seen above, x and y are results for inputs a and b,

$$a.x + b.y = gcd \qquad \text{----(1)}$$

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(b/a)

$$(b - (\lfloor b/a \rfloor).a).x1 + a.y1 = gcd$$

Above equation can also be written as below

$$b.x1 + a.(y1 - (\lfloor b/a \rfloor).x1) = gcd \qquad \text{---(2)}$$

After comparing coefficients of 'a' and 'b' in (1) and (2), we get following

$$x = y1 - \lfloor b/a \rfloor * x1$$

$$y = x1$$

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 the RSA public-key encryption method.