SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF AIML

19ITB201 – DESIGN AND ANALYSIS OF ALGORITHMS

II YEAR IV SEM

UNIT-I-Introduction

TOPIC: Euclid Algorithm

Prepared by
C.PARKAVI,AP/AIML

EUCLID'S ALGORITHM

Subject :Design and Analysis of Algorithm
Unit :I

GCD(Greatest common divisor)

- The greatest common divisor of two nonnegative, not-both-zero integers *m* and *n*, denoted gcd*(m, n)*, is defined as the largest integer that divides both *m* and *n* evenly, i.e., with a remainder of zero
- Euclid of Alexandria (third century B.c.) outlined an algorithm for solving this problem in one of the volumes of his *Elements* most famous for its systematic exposition of geometry

Euclid's algorithm

Euclid's algorithm is based on applying repeatedly the equality

gcd(m, n) = gcd$(n, m \bmod n)$,

Where *m* mod *n* is the remainder of the division of *m* by *n*, until *m* mod *n* is equal to 0. Since gcd$(m, 0) = m$ (why?), the last value of *m* is also the greatest common divisor of the initial *m* and *n*.

For example, gcd(60, 24) can be computed as follows:

gcd(60, 24) = gcd(24, 12) = gcd(12, 0) = 12.

Euclid's algorithm for computing gcd*(m, n)*

Step 1 If *n*=0, return the value of *m* as the answer and stop;

otherwise, proceed to Step 2.

Step 2 Divide *m* by *n* and assign the value of the remainder to *r*.

Step 3 Assign the value of *n* to *m* and the value of *r* to *n*. Go to Step

Alternatively, we can express the same algorithm in pseudocode:

ALGORITHM *Euclid(m, n)*

//Computes gcd*(m, n)* by Euclid's algorithm

//Input: Two nonnegative, not-both-zero integers *m* and *n*

//Output: Greatest common divisor of *m* and *n*

while *n!*= 0 **do**

r ← *m* mod *n*

m ← *n*

n ←*r*

return *m*

Consecutive integer checking algorithm

Consecutive integer checking algorithm for computing gcd*(m, n)*

Step 1 Assign the value of min{*m*, *n*} to *t*.

Step 2 Divide *m* by *t*. If the remainder of this division is 0, go to Step 3; otherwise, go to Step 4.

Step 3 Divide *n* by *t*. If the remainder of this division is 0, return the value of *t* as the answer and stop; otherwise, proceed to Step 4.

Step 4 Decrease the value of *t* by 1. Go to Step 2.

Middle-school procedure

Middle-school procedure for computing gcd*(m, n)*

Step 1 Find the prime factors of *m*.

Step 2 Find the prime factors of *n*.

Step 3 Identify all the common factors in the two prime expansions found in Step 1 and Step 2. (If *p* is a common factor occurring *pm* and *pn* times in *m* and *n*, respectively, it should be repeated min{*pm*, *pn*} times.)

Step 4 Compute the product of all the common factors and return it as the greatest common divisor of the numbers given.

Thus, for the numbers 60 and 24, we get

60 = 2 . 2 . 3 . 5

24 = 2 . 2 . 2 . 3

gcd*(*60, 24*)* = 2 . 2 . 3 = 12.

sieve of Eratosthenes

As an example, consider the application of the algorithm to finding the list of primes not exceeding $n = 25$:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
2	3		5		7		9		11		13		15		17		19		21		23		25
2	3		5		7				11		13				17		19				23		25
2	3		5		7				11		13				17		19				23		

ALGORITHM *Sieve(n)*

//Implements the sieve of Eratosthenes

//Input: A positive integer *n*> 1

//Output: Array *L* of all prime numbers less than or equal to *n*

```
for p ← 2 to n do A[p] ← p
for p ← 2 to ⌊√n⌋ do      //see note before pseudocode
    if A[p] ≠ 0           //p hasn't been eliminated on previous passes
        j ← p * p
        while j ≤ n do
            A[j] ← 0      //mark element as eliminated
            j ← j + p
//copy the remaining elements of A to array L of the primes
i ← 0
for p ← 2 to n do
    if A[p] ≠ 0
        L[i] ← A[p]
        i ← i + 1
return L
```


Thank you!