CMPS 3120

Algorithm Analysis

Dr. Chengwei Lei
CEECS
California State University, Bakersfield
Problem: Find gcd\((m,n)\), the greatest common divisor of two nonnegative, not both zero integers \(m\) and \(n\)
Euclid’s Algorithm

Problem:

Examples: \( \gcd(60, 24) = 12 \), \( \gcd(60, 0) = 60 \), \( \gcd(0, 0) = ? \)
Euclid’s Algorithm

Problem:

Examples: \( \gcd(60, 24) = 12, \quad \gcd(60, 0) = 60, \quad \gcd(0, 0) = ? \)

Euclid’s algorithm:

Euclid’s algorithm is based on repeated application of equality

\[
\gcd(m, n) = \gcd(n, m \mod n)
\]

until the second number becomes 0, which makes the problem trivial.

Example: \( \gcd(60, 24) = \gcd(24, 12) = \gcd(12, 0) = 12 \)
Two descriptions of Euclid’s algorithm

Step 1  If $n = 0$, return $m$ and stop; otherwise go to Step 2
Step 2  Divide $m$ by $n$ and assign the value fo the remainder to $r$
Step 3  Assign the value of $n$ to $m$ and the value of $r$ to $n$. Go to Step 1.
Two descriptions of Euclid’s algorithm

Step 1  If \( n = 0 \), return \( m \) and stop; otherwise go 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 1.

while \( n \neq 0 \) do
    \( r \leftarrow m \mod n \)
    \( m \leftarrow n \)
    \( n \leftarrow r \)
return \( m \)
while $n \neq 0$ do
  $r \leftarrow m \mod n$
  $m \leftarrow n$
  $n \leftarrow r$
return $m$
while $n \neq 0$ do
    $r \leftarrow m \mod n$
    $m \leftarrow n$
    $n \leftarrow r$
return $m$

int gcd(int m, int n)
{
    while(n!=0)
    {
        int $r = m \% n$;
        $m = n$;
        $n = r$;
    }
    return m;
}
#include <stdio.h>

int gcd(int m, int n) {
    while (n != 0) {
        int r = m % n;
        m = n;
        n = r;
    }
    return m;
}

int main() {
    int input1, input2, result;

    printf("Enter two positive integers: ");
    scanf("%d %d", &input1, &input2);

    result = gcd(input1, input2);

    printf("GCD = %d\n", result);

    return 0;
}
Other methods for computing gcd($m,n$)

Consecutive integer checking algorithm

Step 1  Assign the value of min{$m,n$} to $t$
Step 2  Divide $m$ by $t$. If the remainder is 0, go to Step 3; otherwise, go to Step 4
Step 3  Divide $n$ by $t$. If the remainder is 0, return $t$ and stop; otherwise, go to Step 4
Step 4  Decrease $t$ by 1 and go to Step 2
Other methods for computing gcd($m,n$)

Consecutive integer checking algorithm

```c
int gcd(int m, int n)
{
    int t=m>n?n:m;
    step2:
        if(m%t==0)
            if(n%t==0)
                return t;
        t=t-1;
        goto step2;
}
```
Other methods for $\gcd(m, n)$ [cont.]

Middle-school procedure

Step 1  Find the prime factorization of $m$
Step 2  Find the prime factorization of $n$
Step 3  Find all the common prime factors
Step 4  Compute the product of all the common prime factors and return it as $\gcd(m, n)$

Is this an algorithm?
Sieve of Eratosthenes

Input: Integer $n \geq 2$
Output: List of primes less than or equal to $n$

for $p \leftarrow 2$ to $n$ do $A[p] \leftarrow p$
for $p \leftarrow 2$ to $\lfloor \sqrt{n} \rfloor$ do
  if $A[p] \neq 0$ // $p$ hasn't been previously eliminated from the list
    $j \leftarrow p \cdot p$
    while $j \leq n$ do
      $A[j] \leftarrow 0$ // mark element as eliminated
      $j \leftarrow j + p$
Sieve of Eratosthenes

Input: Integer \( n \geq 2 \)

Output: List of primes less than or equal to \( n \)

\[
\begin{align*}
\text{for } p & \leftarrow 2 \text{ to } n \text{ do } A[p] \leftarrow p \\
\text{for } p & \leftarrow 2 \text{ to } \lfloor \sqrt{n} \rfloor \text{ do} \\
& \quad \text{if } A[p] \neq 0 \quad /\!/p \text{ hasn’t been previously eliminated from the list} \\
& \quad \quad j \leftarrow p \cdot p \\
& \quad \quad \text{while } j \leq n \text{ do} \\
& \quad \quad \quad A[j] \leftarrow 0 \quad /\!/\text{mark element as eliminated} \\
& \quad \quad \quad j \leftarrow j + p
\end{align*}
\]

Example: 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Two main issues related to algorithms

- How to design algorithms
- How to analyze algorithm efficiency
Algorithm design techniques/strategies

- Brute force
- Divide and conquer
- Decrease and conquer
- Transform and conquer
- Space and time tradeoffs
- Greedy approach
- Dynamic programming
- Iterative improvement
- Backtracking
- Branch and bound
Analysis of algorithms

- How good is the algorithm?
  - time efficiency
  - space efficiency

- Does there exist a better algorithm?
  - lower bounds
  - optimality
Important problem types

- sorting
- searching
- string processing
- graph problems
- combinatorial problems
- geometric problems
- numerical problems
Fundamental data structures

- list
- array
- linked list
- string
- stack
- queue
- priority queue

- graph
- tree
- set and dictionary