



Design and analysis of algorithms

Lecture 38 & 39 & 40

Edyta Szymańska

edyta@cc.gatech.edu

Euclid's algorithm for greatest common divisor

Given two integers a and b , compute the largest integer which divides both of them.

Euclid's algorithm for greatest common divisor

Given two integers a and b , compute the largest integer which divides both of them.

Example: $\text{GCD}(1035, 759) = ?$

Euclid's algorithm for greatest common divisor

Given two integers a and b , compute the largest integer which divides both of them.

Example: $\text{GCD}(1035, 759) = ?$

If we can factorize, then $1035 = 3^2 \cdot 5 \cdot 23$ and $759 = 3 \cdot 11 \cdot 23$ and thus $\text{GCD}(1035, 759) = 3 \cdot 23 = 69$.

Euclid's algorithm for greatest common divisor

Given two integers a and b , compute the largest integer which divides both of them.

Example: $\text{GCD}(1035, 759) = ?$

If we can factorize, then $1035 = 3^2 \cdot 5 \cdot 23$ and $759 = 3 \cdot 11 \cdot 23$ and thus $\text{GCD}(1035, 759) = 3 \cdot 23 = 69$.

No polynomial procedure is known for factoring integers. Different method must be used.

Euclid's algorithm for greatest common divisor

Idea:



Euclid's algorithm for greatest common divisor

Idea:

Proposition *If $a > b$ then $\gcd(a, b) = \gcd(a \bmod b, b)$*

Euclid's algorithm for greatest common divisor

Idea:

Proposition *If $a > b$ then $\gcd(a, b) = \gcd(a \bmod b, b)$*

function `EUCLID(a, b)`

Input: two positive integers a, b with $a \geq b$

Output: $\gcd(a, b)$

if $b = 0$ then return a

return `EUCLID($b, a \bmod b$)`

Running time ?

Euclid's algorithm for greatest common divisor

Idea:

Proposition *If $a > b$ then $\gcd(a, b) = \gcd(a \bmod b, b)$*

function EUCLID(a, b)

Input: two positive integers a, b with $a \geq b$

Output: $\gcd(a, b)$

if $b = 0$ then return a

return EUCLID($b, a \bmod b$)

Running time ?

Claim *If $a \geq b$ then $a \bmod b \leq \frac{a}{2}$.*

Euclid's algorithm for greatest common divisor

Idea:

Proposition *If $a > b$ then $\gcd(a, b) = \gcd(a \bmod b, b)$*

function EUCLID(a, b)

Input: two positive integers a, b with $a \geq b$

Output: $\gcd(a, b)$

if $b = 0$ then return a

return EUCLID($b, a \bmod b$)

Running time ?

Claim *If $a \geq b$ then $a \bmod b \leq \frac{a}{2}$.*

Number of recursive calls: $2 \lceil \log_2 b \rceil$.

Since $n = \lceil \log_2 a \rceil + \lceil \log_2 b \rceil$, we have $O(n)$ rounds and the total time is $O(n \cdot n^2 = n^3)$ (n^2 for division.)

An extension to Euclid's algorithm

How can we check that d is the greatest common divisor of a and b ?

An extension to Euclid's algorithm

How can we check that d is the greatest common divisor of a and b ?

Claim *If $d|a$ and $d|b$ and $d = ax + by$ for some integers x, y , then $d = \gcd(a, b)$.*

An extension to Euclid's algorithm

How can we check that d is the greatest common divisor of a and b ?

Claim *If $d|a$ and $d|b$ and $d = ax + by$ for some integers x, y , then $d = \gcd(a, b)$.*

Claim *For any inputs a, b , the Extended Euclid algorithm returns integers x, y, d such that $\gcd(a, b) = d = ax + by$.*

An extension to Euclid's algorithm

How can we check that d is the greatest common divisor of a and b ?

Claim *If $d|a$ and $d|b$ and $d = ax + by$ for some integers x, y , then $d = \gcd(a, b)$.*

Claim *For any inputs a, b , the Extended Euclid algorithm returns integers x, y, d such that $\gcd(a, b) = d = ax + by$.*

function EXTENDED-EUCLID(a, b)

Input: two positive integers a, b with $a \geq b$

Output: integers x, y, d such that $d = \gcd(a, b) = ax + by$

if $b = 0$ then return $(1, 0, a)$

$(x', y', d) = \text{EXTENDED-EUCLID}(b, a \bmod b)$

return $(y', x' - \lfloor \frac{a}{b} \rfloor y', d)$

Modular division

In regular arithmetic: $\forall a \neq 0, a$ has an inverse $\frac{1}{a}$.

Modular division

In regular arithmetic: $\forall a \neq 0, a$ has an inverse $\frac{1}{a}$.

Modular division

In regular arithmetic: $\forall a \neq 0$, a has an inverse $\frac{1}{a}$.

In modular arithmetic: x is a multiplicative inverse of a if $ax \equiv 1 \pmod{m}$.

We denote $x := a^{-1}$ (at most one such x modulo m exists (why?)).

Modular division

In regular arithmetic: $\forall a \neq 0$, a has an inverse $\frac{1}{a}$.

In modular arithmetic: x is a multiplicative inverse of a if $ax \equiv 1 \pmod{m}$.

We denote $x := a^{-1}$ (at most one such x modulo m exists (why?)).

Proposition *For any $a < m$, a has a multiplicative inverse modulo m if and only if it is relatively prime to m . When this inverse exists then it can be found in time $O(\log^3 m)$.*

Modular division

In regular arithmetic: $\forall a \neq 0$, a has an inverse $\frac{1}{a}$.

In modular arithmetic: x is a multiplicative inverse of a if $ax \equiv 1 \pmod{m}$.

We denote $x := a^{-1}$ (at most one such x modulo m exists (why?)).

Proposition *For any $a < m$, a has a multiplicative inverse modulo m if and only if it is relatively prime to m . When this inverse exists then it can be found in time $O(\log^3 m)$.*

Use Extended-Euclid's algorithm to find the inverse of a if a and m are relatively prime.

Primality testing

Theorem (Fermat's Little Theorem) *If m is prime then for every $1 \leq a < m$, we have*

$$a^{m-1} \equiv 1 \pmod{m}.$$

Primality testing

Theorem (Fermat's Little Theorem) *If m is prime then for every $1 \leq a < m$, we have*

$$a^{m-1} \equiv 1 \pmod{m}.$$

*It is promising, but notice that it is not **an equivalence** statement.*

Primality testing

Theorem (Fermat's Little Theorem) *If m is prime then for every $1 \leq a < m$, we have*

$$a^{m-1} \equiv 1 \pmod{m}.$$

*It is promising, but notice that it is not **an equivalence** statement.*

However, if a is picked randomly, it is unlikely that

$$a^{m-1} \equiv 1 \pmod{m}.$$

This motivates a randomized algorithm PRIMALITY1.

Primality testing

Theorem (Fermat's Little Theorem) *If m is prime then for every $1 \leq a < m$, we have*

$$a^{m-1} \equiv 1 \pmod{m}.$$

*It is promising, but notice that it is not **an equivalence** statement.*

However, if a is picked randomly, it is unlikely that

$$a^{m-1} \equiv 1 \pmod{m}.$$

This motivates a randomized algorithm PRIMALITY1.

Primality testing

PRIMALITY1(m)

Input: a positive integer m

Output: yes/no

Pick an integer $a < m$ at random

if $a^{m-1} \equiv 1 \pmod{m}$

 then return yes

 else return no

Primality testing

PRIMALITY1(m)

Input: a positive integer m

Output: yes/no

Pick an integer $a < m$ at random

if $a^{m-1} \equiv 1 \pmod{m}$

 then return yes

 else return no

Primality testing - analysis

PRIMALITY 1 is a randomized algorithm.

Primality testing - analysis

PRIMALITY 1 is a randomized algorithm.

Properties:

- ⑥ it may give an incorrect answer (when?)
- ⑥ one of the two possible outputs (yes/no) is always correct (which one?)

Primality testing - analysis

PRIMALITY 1 is a randomized algorithm.

Properties:

- ⑥ it may give an incorrect answer (when?)
- ⑥ one of the two possible outputs (yes/no) is always correct (which one?)

The algorithm belongs to a class of **Monte Carlo** algorithms with *one-sided error*.

Primality testing - analysis

PRIMALITY 1 is a randomized algorithm.

Properties:

- ⑥ it may give an incorrect answer (when?)
- ⑥ one of the two possible outputs (yes/no) is always correct (which one?)

The algorithm belongs to a class of **Monte Carlo** algorithms with *one-sided error*.

Carmichael numbers

Bad news for PRIMALITY1 algorithm:

Carmichael numbers

Bad news for PRIMALITY1 algorithm:

There exist composite numbers m (named after Carmichael), which satisfy $a^{m-1} \equiv 1 \pmod{m}$ for all values of a !

Carmichael numbers

Bad news for PRIMALITY1 algorithm:

There exist composite numbers m (named after Carmichael), which satisfy $a^{m-1} \equiv 1 \pmod{m}$ for all values of a !

Fortunately, Carmichael numbers are extremely rare (first three are 561, 1105, 1729) and we will ignore them for a moment.

Carmichael numbers

Bad news for PRIMALITY1 algorithm:

There exist composite numbers m (named after Carmichael), which satisfy $a^{m-1} \equiv 1 \pmod{m}$ for all values of a !

Fortunately, Carmichael numbers are extremely rare (first three are 561, 1105, 1729) and we will ignore them for a moment.

Claim *If $a^{m-1} \not\equiv 1 \pmod{m}$ for some a relatively prime to m , then it must hold for at least half the choices of $a < m$.*

Primality testing - analysis

Assuming that there are no Carmichael numbers, we have

Primality testing - analysis

Assuming that there are no Carmichael numbers, we have

- ⑥ if m is a **prime** then $a^{m-1} \equiv 1 \pmod{m}$ for all $a < m$
- ⑥ if m is **composite** then $a^{m-1} \equiv 1 \pmod{m}$ for at most half the values of $a < m$

Primality testing - analysis

Assuming that there are no Carmichael numbers, we have

- ⑥ if m is a **prime** then $a^{m-1} \equiv 1 \pmod{m}$ for all $a < m$
- ⑥ if m is **composite** then $a^{m-1} \equiv 1 \pmod{m}$ for at most half the values of $a < m$

The algorithm PRIMALITY1 has the following guarantee:

Primality testing - analysis

Assuming that there are no Carmichael numbers, we have

- ⑥ if m is a **prime** then $a^{m-1} \equiv 1 \pmod{m}$ for all $a < m$
- ⑥ if m is **composite** then $a^{m-1} \equiv 1 \pmod{m}$ for at most half the values of $a < m$

The algorithm `PRIMALITY1` has the following guarantee:

- ⑥ $\Pr(\text{PRIMALITY1 returns yes when } m \text{ is a prime}) = 1$
- ⑥ $\Pr(\text{PRIMALITY1 returns yes when } m \text{ is composite}) \leq \frac{1}{2}$.

Lower the error probability

Independent repetitions can reduce the error probability:

Lower the error probability

Independent repetitions can reduce the error probability:

PRIMALITY2(m)

Input: a positive integer m

Output: yes/no

Pick integers $a_1, a_2, \dots, a_k < m$ at random

if $a_i^{m-1} \equiv 1 \pmod{m}$ for all $i = 1, 2, \dots, k$

then return yes

else return no

Lower the error probability

Independent repetitions can reduce the error probability:

PRIMALITY2(m)

Input: a positive integer m

Output: yes/no

Pick integers $a_1, a_2, \dots, a_k < m$ at random

if $a_i^{m-1} \equiv 1 \pmod{m}$ for all $i = 1, 2, \dots, k$

then return yes

else return no

$\Pr(\text{PRIMALITY2 returns yes when } m \text{ is composite}) \leq \frac{1}{2^k},$

can be made arbitrarily small by choosing k large enough.

Miller-Rabin primality test

It is an algorithm which handles the Carmichael numbers - see Advanced Notes for details.

Complexity of Primes



Recent advances:

Complexity of Primes

Recent advances:

The problem of deciding whether or not an integer is a prime can be solved in time polynomial in the number of bits necessary to represent the integer at hand (that is, roughly the logarithm of the integer itself)

Complexity of Primes

Recent advances:

The problem of deciding whether or not an integer is a prime can be solved in time polynomial in the number of bits necessary to represent the integer at hand (that is, roughly the logarithm of the integer itself)

References:

*M. Agrawal, N. Kayal and N. Saxena, **PRIMES is in P***

Complexity of Primes

Recent advances:

The problem of deciding whether or not an integer is a prime can be solved in time polynomial in the number of bits necessary to represent the integer at hand (that is, roughly the logarithm of the integer itself)

References:

*M. Agrawal, N. Kayal and N. Saxena, **PRIMES is in P***

The fact that PRIME has been found to be in P cannot be used to break any cryptographic algorithms. (Moreover, the randomized algorithms are still faster.)