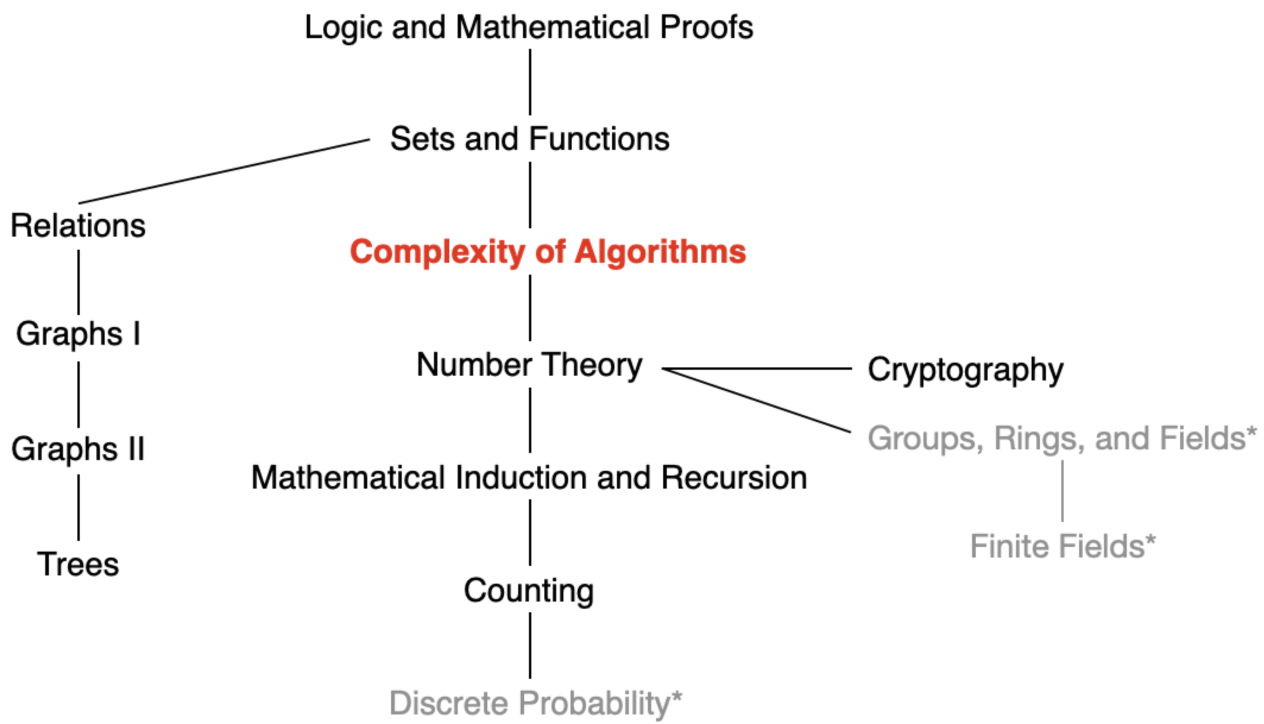
Discrete Mathematics for Computer Science

Lecture 7: Number Theory

Dr. Ming Tang

Department of Computer Science and Engineering
Southern University of Science and Technology (SUSTech)
Email: tangm3@sustech.edu.cn

This Lecture



The growth of functions, complexity of algorithm,
P and NP ...

Decision Problems and Optimization Problem

Definition: A decision problem is a question that has two possible answers: yes and no.

Decision Problems and Optimization Problem

Definition: A decision problem is a question that has two possible answers: yes and no.

Definition: An optimization problem requires an answer that is an optimal configuration.

- Decision variables
- Maximize or minimize certain objective subject to some constraints

Decision Problems and Optimization Problem

Definition: A decision problem is a question that has two possible answers: yes and no.

Definition: An optimization problem requires an answer that is an optimal configuration.

- Decision variables
- Maximize or minimize certain objective subject to some constraints

An optimization problem usually has a corresponding decision problem.

Decision Problems and Optimization Problem

Definition: A decision problem is a question that has two possible answers: yes and no.

Definition: An optimization problem requires an answer that is an optimal configuration.

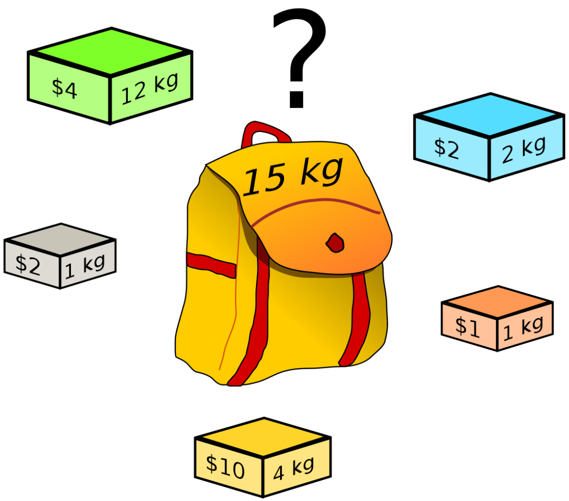
- Decision variables
- Maximize or minimize certain objective subject to some constraints

An optimization problem usually has a corresponding decision problem.

Examples:
Knapsack vs. Decision Knapsack (DKnapsack)

Knapsack V.S. DKnapsack

We have a knapsack of capacity W (a positive integer) and N objects with weights $w_1, \ldots, w_N$ and values $v_1, \ldots, v_N$, where v_n and w_n are positive integers.



Knapsack V.S. DKnapsack

We have a knapsack of capacity W (a positive integer) and N objects with weights $w_1, \ldots, w_N$ and values $v_1, \ldots, v_N$, where v_n and w_n are positive integers.

Optimization problem (Knapsack):

- Decision variable $x_n \in \{0, 1\}$: $x_n = 1$, object x is placed in the knapsack; $x_n = 0$, otherwise
- Maximize $\sum_{n=\{1,\ldots,N\}} x_n v_n$, subject to constraint $\sum_{n=\{1,\ldots,N\}} x_n w_n \leq W$.

Knapsack V.S. DKnapsack

We have a knapsack of capacity W (a positive integer) and N objects with weights $w_1, \ldots, w_N$ and values $v_1, \ldots, v_N$, where v_n and w_n are positive integers.

Optimization problem (Knapsack):

- Decision variable $x_n \in \{0,1\}$: $x_n = 1$, object x is placed in the knapsack; $x_n = 0$, otherwise
- Maximize $\sum_{n=\{1,\ldots,N\}} x_n v_n$, subject to constraint $\sum_{n=\{1,\ldots,N\}} x_n w_n \leq W$.

Decision problem (DKnapsack): Given V, is there a subset of the objects that fits in the knapsack and has total value at least V?

SUSTech Southern University of Science and Technology

Knapsack V.S. DKnapsack

We have a knapsack of capacity W (a positive integer) and N objects with weights $w_1, \ldots, w_N$ and values $v_1, \ldots, v_N$, where v_n and w_n are positive integers.

Optimization problem (Knapsack):

- Decision variable $x_n \in \{0, 1\}$: $x_n = 1$, object x is placed in the knapsack; $x_n = 0$, otherwise
- Maximize $\sum_{n=\{1,\ldots,N\}} x_n v_n$, subject to constraint $\sum_{n=\{1,\ldots,N\}} x_n w_n \leq W$.

Decision problem (DKnapsack): Given V, is there a subset of the objects that fits in the knapsack and has total value at least V?

The optimization problem is at least as hard as the decision problem.

Decision Problems and Optimization Problem

Given a subroutine for solving the optimization problem, solving the corresponding decision problem is usually trivial.

- First, solve the optimization problem
- Then, check the decision problem.

Thus, if we prove that a given decision problem is hard to solve efficiently, then it is obvious that the optimization problem must be (at least as) hard.

Complexity Classes

Theory of Complexity deals with

1. the classification of certain "decision problems" into several classes:
   - the class of "easy" problems
   - the class of "hard" problems
   - the class of "hardest" problems
2. relations among the three classes
3. properties of problems in the three classes

Complexity Classes

Theory of Complexity deals with

1. the classification of certain "decision problems" into several classes:
   - the class of "easy" problems
   - the class of "hard" problems
   - the class of "hardest" problems
2. relations among the three classes
3. properties of problems in the three classes

Question: How to classify decision problems?

Complexity Classes

Theory of Complexity deals with

1. the classification of certain "decision problems" into several classes:
   - the class of "easy" problems
   - the class of "hard" problems
   - the class of "hardest" problems
2. relations among the three classes
3. properties of problems in the three classes

Question: How to classify decision problems?

Answer: Use polynomial-time algorithms.

Complexity Classes

Theory of Complexity deals with

1. the classification of certain "decision problems" into several classes:
   - the class of "easy" problems
   - the class of "hard" problems
   - the class of "hardest" problems
2. relations among the three classes
3. properties of problems in the three classes

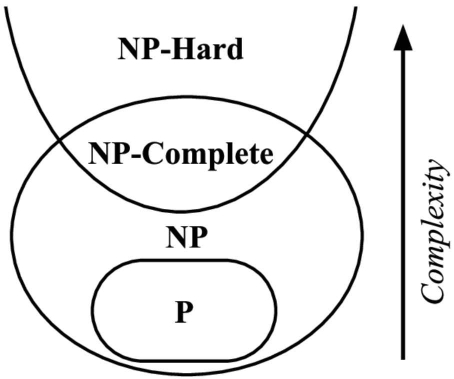
Question: How to classify decision problems?

Answer: Use polynomial-time algorithms.

P problem, NP problem, ...

To Be Discussed

- Polynomial-time algorithms
- P problem and NP problem



Polynomial-Time Algorithms

Definition: An algorithm is polynomial-time if its running time is $O(n^k)$, where k is a constant independent of n, and n is the input size of the problem that the algorithm solves.

Polynomial-Time Algorithms

Definition: An algorithm is polynomial-time if its running time is $O(n^k)$, where k is a constant independent of n, and n is the input size of the problem that the algorithm solves.

Whether we use n or n^a (for a fixed $a > 0$) as the input size, it will not affect the conclusion of whether an algorithm is polynomial-time.

Polynomial-Time Algorithms

Definition: An algorithm is polynomial-time if its running time is $O(n^k)$, where k is a constant independent of n, and n is the input size of the problem that the algorithm solves.

Whether we use n or n^a (for a fixed $a > 0$) as the input size, it will not affect the conclusion of whether an algorithm is polynomial-time.

Example:
The standard multiplication algorithm has time $O(m_1 m_2)$, where m_1 and m_2 denote the number of digits in the two integers, respectively.

Nonpolynomial-Time Algorithms

Definition: An algorithm is nonpolynomial-time if the running time is not $O(n^k)$ for any fixed $k \geq 0$.

Nonpolynomial-Time Algorithms

Definition: An algorithm is nonpolynomial-time if the running time is not $O(n^k)$ for any fixed $k \geq 0$.

Example (Composite): The naive algorithm for determining whether n is composite compares n with the first $n - 1$ numbers to see if any of them divides n.

Nonpolynomial-Time Algorithms

Definition: An algorithm is nonpolynomial-time if the running time is not $O(n^k)$ for any fixed $k \geq 0$.

Example (Composite): The naive algorithm for determining whether n is composite compares n with the first $n-1$ numbers to see if any of them divides n.

- Let $m = \log_2 n$ be the input size of this problem

Nonpolynomial-Time Algorithms

Definition: An algorithm is nonpolynomial-time if the running time is not $O(n^k)$ for any fixed $k \geq 0$.

Example (Composite): The naive algorithm for determining whether n is composite compares n with the first $n-1$ numbers to see if any of them divides n.

- Let $m = \log_2 n$ be the input size of this problem
- Thus, the complexity if $\Theta(n) = \Theta(2^{(\log_2 n)})$, which is $\Theta(2^m)$

Nonpolynomial-Time Algorithms

Definition: An algorithm is nonpolynomial-time if the running time is not $O(n^k)$ for any fixed $k \geq 0$.

Example (Composite): The naive algorithm for determining whether n is composite compares n with the first $n-1$ numbers to see if any of them divides n.

- Let $m = \log_2 n$ be the input size of this problem
- Thus, the complexity if $\Theta(n) = \Theta(2^{(\log_2 n)})$, which is $\Theta(2^m)$
- The algorithm is nonpolynomial!

Polynomial- vs. Nonpolynomial-Time

Nonpolynomial-time algorithms are impractical.

- 2^n for $n = 100$: it takes billions of years!!!

Polynomial- vs. Nonpolynomial-Time

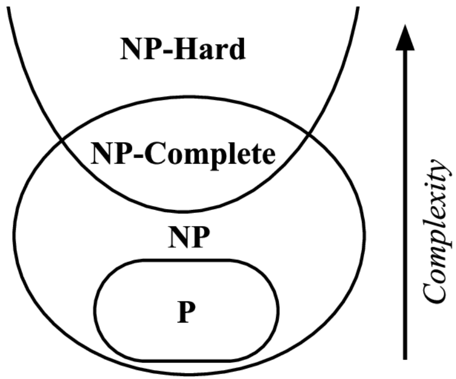
Nonpolynomial-time algorithms are impractical.

- 2^n for $n = 100$: it takes billions of years!!!

In reality, an $O(n^{20})$ algorithm is not really practical.

To Be Discussed

- Polynomial-time algorithms
- P problem and NP problem



The Class P

Definition: A problem is solvable in polynomial time (or more simply, the problem is in polynomial time) if there exists an algorithm which solves the problem in polynomial time

- This problem is called tractable.

Definition (The Class P): The class P consists of all decision problems that are solvable in polynomial time. That is, there exists an algorithm that will decide in polynomial time if any given input is a yes-input or a no-input.

The Class P

Question: How to prove that a decision problem is in P?

The Class P

Question: How to prove that a decision problem is in P?

Answer: Find a polynomial-time algorithm.

The Class P

Question: How to prove that a decision problem is in P?

Answer: Find a polynomial-time algorithm.

Question: How to prove that a decision problem is not in P?

The Class P

Question: How to prove that a decision problem is in P?

Answer: Find a polynomial-time algorithm.

Question: How to prove that a decision problem is not in P?

Answer: You need to prove that there is no polynomial-time algorithm for this problem. (much much harder)

The Class P

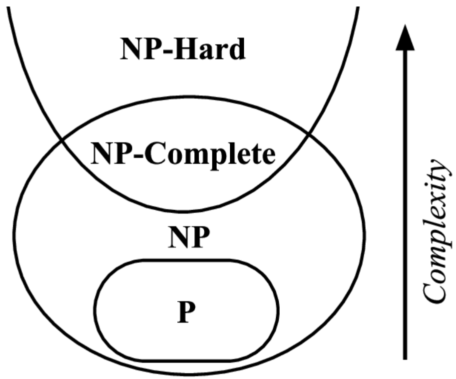
Question: How to prove that a decision problem is in P?

Answer: Find a polynomial-time algorithm.

Question: How to prove that a decision problem is not in P?

Answer: You need to prove that there is no polynomial-time algorithm for this problem. (much much harder)

- Some other definitions for potentially harder problems ....



Certificates and Verifying Certificates

Before introduce NP Problem, some new definitions ...

Certificates and Verifying Certificates

Before introduce NP Problem, some new definitions ...

A decision problem is usually formulated as:

Is there an object satisfying some conditions?

Certificates and Verifying Certificates

Before introduce NP Problem, some new definitions ...

A decision problem is usually formulated as:

Is there an object satisfying some conditions?

A certificate (or witness) is a specific object corresponding to a yes-input, such that it can be used to show that the input is indeed a yes-input.

Certificates and Verifying Certificates

Before introduce NP Problem, some new definitions ...

A decision problem is usually formulated as:

Is there an object satisfying some conditions?

A certificate (or witness) is a specific object corresponding to a yes-input, such that it can be used to show that the input is indeed a yes-input.

Example (DKnapsack): Given V, is there a subset of the objects that fits in the knapsack and has total value at least V?

To show V is a yes-input, a certificate is a subset of the objects that

- fit in the knapsack (i.e., the sum weight does not exceed the capacity)
- have a total value at least V

Certificates and Verifying Certificates

A certificate (or witness) is a specific object corresponding to a yes-input, such that it can be used to show that the input is indeed a yes-input.

Verifying a certificate: Given a presumed yes-input and its corresponding certificate, by making use of the given certificate, we verify that the input is actually a yes-input.

Certificates and Verifying Certificates

A certificate (or witness) is a specific object corresponding to a yes-input, such that it can be used to show that the input is indeed a yes-input.

Verifying a certificate: Given a presumed yes-input and its corresponding certificate, by making use of the given certificate, we verify that the input is actually a yes-input.

Proposition: The problem **LongPath(G,k)** is in **NP**.
Proof: (PARTIAL!)

1. Note that **LongPath(G,k)** is a decision problem, as the definition of NP requires!
2. Here's my notion of certificate: A certificate is a list of vertices comprising a path of length at least k
3. Here's my algorithm for verifying a certificate:

```
Verify(G,k,C)
  1. Read G, k, store graph G in an adjacency matrix
  2. Read certificate C into an array
  3. if m < k, where m is the length of C, return FALSE
  4. for i = 1 to m - 1 do
        if G has no edge from vertex C[i-1] to C[i] return FALSE
  5. for i = 0 to m - 1 do
       for j = i + 1 to m - 1 do
         if C[i] == C[j] return FALSE
  6. return TRUE
```

The Class NP

Definition: The class NP consists of all decision problems such that, for each yes-input, there exists a certificate which allows one to verify in polynomial time that the input is indeed a yes-input.

The Class NP

Definition: The class NP consists of all decision problems such that, for each yes-input, there exists a certificate which allows one to verify in polynomial time that the input is indeed a yes-input.

NP – “nondeterministic polynomial-time”

The Class NP

Definition: The class NP consists of all decision problems such that, for each yes-input, there exists a certificate which allows one to verify in polynomial time that the input is indeed a yes-input.

NP – "nondeterministic polynomial-time"

Example (DKnapsack): Given V, is there a subset of the objects that fits in the knapsack and has total value at least V?

To show V is a yes-input, a certificate is a subset of the objects that

- fit in the knapsack (i.e., the sum weight does not exceed the capacity)
- have a total value at least V

The Class NP

Definition: The class NP consists of all decision problems such that, for each yes-input, there exists a certificate which allows one to verify in polynomial time that the input is indeed a yes-input.

NP – "nondeterministic polynomial-time"

Example (DKnapsack): Given V, is there a subset of the objects that fits in the knapsack and has total value at least V?

To show V is a yes-input, a certificate is a subset of the objects that

- fit in the knapsack (i.e., the sum weight does not exceed the capacity)
- have a total value at least V

DKnapsack is an NP problem.

$P = NP$?

One of the most important problems in CS is

$$\text{Whether } P = NP \text{ or } P \neq NP?$$

P = NP?

One of the most important problems in CS is

Whether P = NP or P $\neq$ NP?

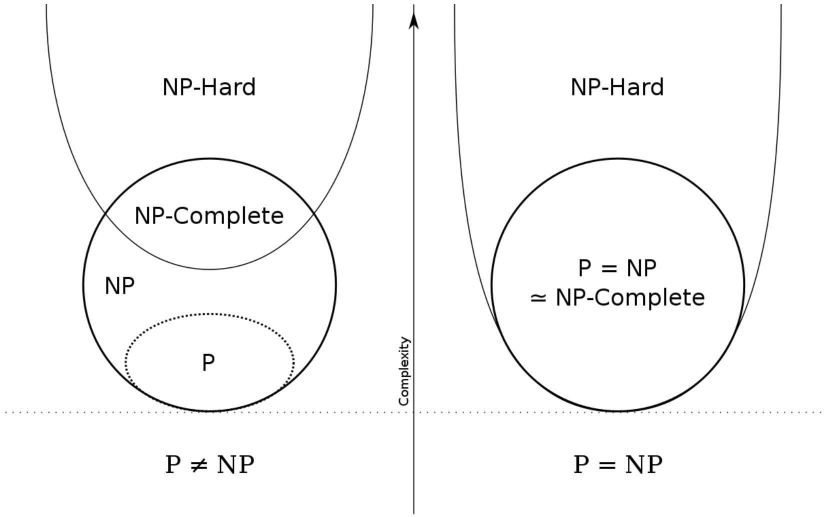
- Observe that $P \subseteq NP$.
- Intuitively, $NP \subseteq P$ is doubtful.

P = NP?

One of the most important problems in CS is

Whether P = NP or P $\neq$ NP?

- Observe that $P \subseteq NP$.
- Intuitively, $NP \subseteq P$ is doubtful.



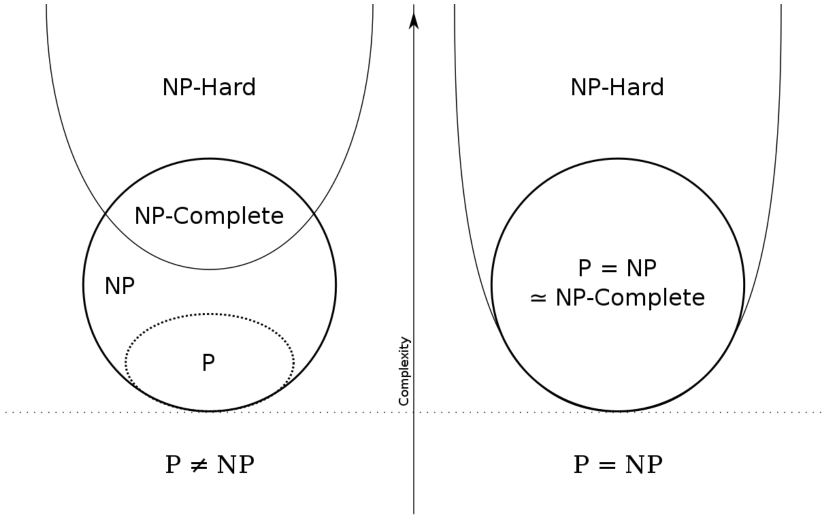
- NP-Hard: informally "at least as hard as the hardest problems in NP"
- NP-Complete: If the problem is NP and all other NP problems are polynomial-time reducible to it.

P = NP?

One of the most important problems in CS is

Whether P = NP or P $\neq$ NP?

- Observe that $P \subseteq NP$.
- Intuitively, $NP \subseteq P$ is doubtful.



- NP-Hard: informally "at least as hard as the hardest problems in NP"
- NP-Complete: If the problem is NP and all other NP problems are polynomial-time reducible to it.

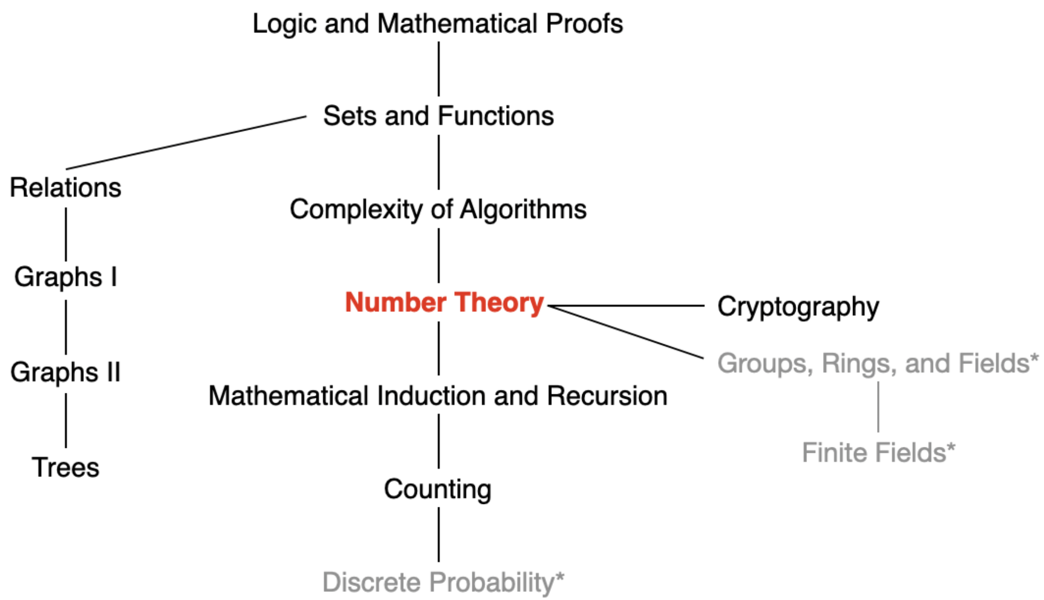
However, we are still no closer to solving it.

SUSTech Southern University of Science and Technology

What We Covered

- Decision problem and optimization
- Polynomial-time algorithms
- P problem and NP problem

Number Theory



Number Theory: divisibility and modular arithmetic, integer representations, primes, greatest common divisors, ...

Number Theory

Number theory is a branch of mathematics that explores integers and their properties, is the basis of cryptography, coding theory, computer security, e-commerce, etc.

Division

If a and b are integers with $a \neq 0$,

- we say that a divides b if there is an integer c such that $b = ac$, or equivalently b/a is an integer.

In this case, we say that a is a factor or divisor of b, and b is a multiple of a. (We use the notations $a|b$, $a \nmid b$)

Division

If a and b are integers with $a \neq 0$,

- we say that a divides b if there is an integer c such that $b = ac$, or equivalently b/a is an integer.

In this case, we say that a is a factor or divisor of b, and b is a multiple of a. (We use the notations $a|b$, $a \nmid b$)

Example:

- $4|24$
- $4 \nmid 5$

Divisibility

All integers divisible by $d > 0$ can be enumerated as:

$$..., -kd, ..., -2d, -d, 0, d, 2d, ..., kd, ...$$

Divisibility

All integers divisible by $d > 0$ can be enumerated as:

$$\ldots, -kd, \ldots, -2d, -d, 0, d, 2d, \ldots, kd, \ldots$$

Question: Let n and d be two positive integers. How many positive integers not exceeding n are divisible by d?

Divisibility

All integers divisible by $d > 0$ can be enumerated as:

$$..., -kd, ..., -2d, -d, 0, d, 2d, ..., kd, ...$$

Question: Let n and d be two positive integers. How many positive integers not exceeding n are divisible by d?

Answer: Count the number of integers such that $0 < kd \leq n$. Therefore, there are $\lfloor n/d \rfloor$ such positive integers.

Divisibility: Properties

Let a, b, c be integers. Then the following hold:

(i) if $a|b$ and $a|c$, then $a|(b+c)$

(ii) if $a|b$ then $a|bc$ for all integers c

(iii) if $a|b$ and $b|c$, then $a|c$

Divisibility: Properties

Let a, b, c be integers. Then the following hold:

(i) if $a|b$ and $a|c$, then $a|(b+c)$

(ii) if $a|b$ then $a|bc$ for all integers c

(iii) if $a|b$ and $b|c$, then $a|c$

Proof: Suppose that $a|b$ and $a|c$. Then, from the definition of divisibility, it follows that there are integers s and t with $b = as$ and $c = at$. Hence,

$$b + c = as + at = a(s + t).$$

Therefore, a divides $b + c$.

Divisibility

Corollary If a, b, c are integers, where $a \neq 0$, such that $a | b$ and $a | c$, then $a | (mb + nc)$ whenever m and n are integers.

Divisibility

Corollary If a, b, c are integers, where $a \neq 0$, such that $a|b$ and $a|c$, then $a|(mb + nc)$ whenever m and n are integers.

Proof: By part (ii) and part (i) of Properties.

The Division Algorithm

If a is an integer and d a positive integer, then there are unique integers q and r, with $0 \leq r < d$, such that

$$a = dq + r.$$

In this case, d is called the divisor, a is called the dividend, q is called the quotient, and r is called the remainder.

The Division Algorithm

If a is an integer and d a positive integer, then there are unique integers q and r, with $0 \leq r < d$, such that

$$a = dq + r.$$

In this case, d is called the divisor, a is called the dividend, q is called the quotient, and r is called the remainder.

In this case, we use the notations $q = a \text{ div } d$ and $r = a \bmod d$.

The Division Algorithm

If a is an integer and d a positive integer, then there are unique integers q and r, with $0 \leq r < d$, such that

$$a = dq + r.$$

In this case, d is called the divisor, a is called the dividend, q is called the quotient, and r is called the remainder.

In this case, we use the notations $q = a \text{ div } d$ and $r = a \bmod d$.

Example: The quotient and remainder when 101 is divided by 11?

$$101 = 11 \times 9 + 2$$

Hence, the quotient is $9 = 101 \text{ div } 11$, and the remainder is $2 = 101 \bmod 11$.

Congruence Relation

If a and b are integers and m is a positive integer, then a is congruent to b modulo m if m divides $a - b$, denoted by $a \equiv b \pmod{m}$. This is called congruence and m is its modulus.

Congruence Relation

If a and b are integers and m is a positive integer, then a is congruent to b modulo m if m divides $a - b$, denoted by $a \equiv b \pmod{m}$. This is called congruence and m is its modulus.

Example:

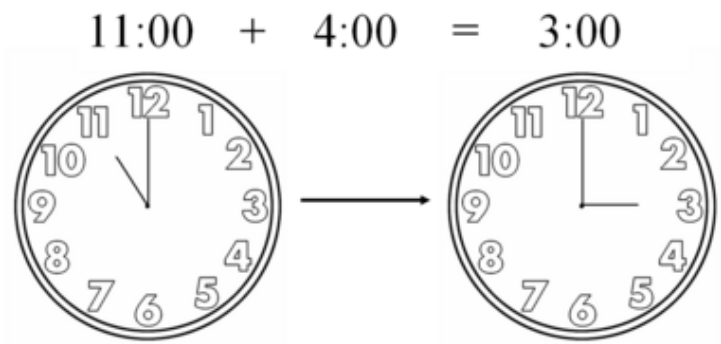
- $15 \equiv 3 \pmod{12}$
- $-1 \equiv 11 \pmod{6}$

Congruence Relation

If a and b are integers and m is a positive integer, then a is congruent to b modulo m if m divides $a - b$, denoted by $a \equiv b \pmod{m}$. This is called congruence and m is its modulus.

Example:

- $15 \equiv 3 \pmod{12}$
- $-1 \equiv 11 \pmod{6}$



Congruence Relation

Let m be a positive integer. The integers a and b are congruent modulo m if and only if there is an integer k such that

$$a = b + km$$

.

Congruence Relation

Let m be a positive integer. The integers a and b are congruent modulo m if and only if there is an integer k such that

$$a = b + km$$

.

Proof:

- If part:
- Only if part:

Congruence Relation

Let m be a positive integer. The integers a and b are congruent modulo m if and only if there is an integer k such that

$$a = b + km$$

.

Proof:

- If part: If there is an integer k such that $a = b + km$, then $km = a - b$. Hence, m divides $a - b$, so that $a \equiv b \pmod{m}$.
- Only if part:

Congruence Relation

Let m be a positive integer. The integers a and b are congruent modulo m if and only if there is an integer k such that

$$a = b + km$$

.

Proof:

- If part: If there is an integer k such that $a = b + km$, then $km = a - b$. Hence, m divides $a - b$, so that $a \equiv b \pmod{m}$.
- Only if part: If $a \equiv b \pmod{m}$, by the definition of congruence, we know that $m|(a - b)$. This means that there is an integer k such that $a - b = km$, so that $a = b + km$.

(mod m) and mod m Notations

Notations $a \equiv b \pmod{m}$ and $a \bmod m$ are different.

- $a \equiv b \pmod{m}$ is a relation on the set of integers
- In $a \bmod m$, the notation mod denotes a function

(mod m) and mod m Notations

Notations $a \equiv b \pmod{m}$ and $a \bmod m$ are different.

- $a \equiv b \pmod{m}$ is a relation on the set of integers
- In $a \bmod m$, the notation mod denotes a function

Let a and b be integers, and let m be a positive integer. Then, $a \equiv b \pmod{m}$ if and only if

$$a \bmod m = b \bmod m$$

.

Congruence: Properties

Theorem: Let m be a positive integer. If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then

$$a + c \equiv b + d \pmod{m}$$

$$ac \equiv bd \pmod{m}$$

Proof:

Congruence: Properties

Theorem: Let m be a positive integer. If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then

$$a + c \equiv b + d \pmod{m}$$

$$ac \equiv bd \pmod{m}$$

Proof: We use a direct proof. Since $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, there are integers s and t with $a = b + sm$ and $c = d + tm$. Hence,

$$b + d = (a - sm) + (c - tm) = (a + c) + m(-s - t)$$

$$bd = (a - sm)(c - tm) = ac + m(-at - cs + stm)$$

Hence, $a + c \equiv b + d \pmod{m}$, $ac \equiv bd \pmod{m}$.

Algebraic Manipulation of Congruence

Question: If $ca \equiv cb \pmod{m}$, then $a \equiv b \pmod{m}$?

Algebraic Manipulation of Congruence

Question: If $ca \equiv cb \pmod{m}$, then $a \equiv b \pmod{m}$?

Answer: No. $14 \equiv 8 \pmod{6}$, but $7 \not\equiv 4 \pmod{6}$

Algebraic Manipulation of Congruence

Question: If $ca \equiv cb \pmod{m}$, then $a \equiv b \pmod{m}$?

Answer: No. $14 \equiv 8 \pmod{6}$, but $7 \not\equiv 4 \pmod{6}$

Question: If $a \equiv b \pmod{m}$ and c is an integer, then

- $ca \equiv cb \pmod{m}$?
- $c + a \equiv c + b \pmod{m}$?
- $a/c \equiv b/c \pmod{m}$?

Algebraic Manipulation of Congruence

Question: If $ca \equiv cb \pmod{m}$, then $a \equiv b \pmod{m}$?

Answer: No. $14 \equiv 8 \pmod{6}$, but $7 \not\equiv 4 \pmod{6}$

Question: If $a \equiv b \pmod{m}$ and c is an integer, then

- $ca \equiv cb \pmod{m}$? Yes
- $c + a \equiv c + b \pmod{m}$? Yes
- $a/c \equiv b/c \pmod{m}$? No

Computing the mod Function

Corollary: Let m be a positive integer and let a and b be integers. Then,

$$(a+b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$$

$$ab \bmod m = ((a \bmod m)(b \bmod m)) \bmod m$$

Computing the mod Function

Corollary: Let m be a positive integer and let a and b be integers. Then,

$$(a+b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$$

$$ab \bmod m = ((a \bmod m)(b \bmod m)) \bmod m$$

Proof: By the definitions of $\bmod m$ and of congruence modulo m, we know that $a \equiv (a \bmod m) (\mathrm{mod}\ m)$ and $b \equiv (b \bmod m) (\mathrm{mod}\ m)$. Hence,

$$a + b \equiv (a \bmod m) + (b \bmod m) (\mathrm{mod}\ m)$$

$$ab \equiv (a \bmod m)(b \bmod m) (\mathrm{mod}\ m).$$

According to the theorem that $a \equiv b \pmod{m}$ if and only if $a \bmod m = b \bmod m$, we obtain the above equalities.

Arithmetic Modulo m

Let Z_m be the set of nonnegative integers less than m: $\{0, 1, ..., m-1\}$.

- $+_m$: $a +_m b = (a + b) \bmod m$
- $\cdot_m$: $a \cdot_m b = ab \bmod m$

Arithmetic Modulo m

Let Z_m be the set of nonnegative integers less than m: $\{0, 1, ..., m-1\}$.

- $+_m$: $a +_m b = (a + b) \bmod m$
- $\cdot_m$: $a \cdot_m b = ab \bmod m$

Example:

- $7 +_{11} 9 =?$
- $7 \cdot_{11} 9 =?$

Arithmetic Modulo m

Let Z_m be the set of nonnegative integers less than m: $\{0, 1, ..., m-1\}$.

- $+_m$: $a +_m b = (a + b) \bmod m$
- $\cdot_m$: $a \cdot_m b = ab \bmod m$

Example:

- $7 +_{11} 9 =?$ 5
- $7 \cdot_{11} 9 =?$ 8

Arithmetic Modulo m

The operations $+_m$ and $\cdot_m$ satisfy many of the same properties of ordinary addition and multiplication of integers:

Arithmetic Modulo m

The operations $+_m$ and $\cdot_m$ satisfy many of the same properties of ordinary addition and multiplication of integers:

Closure: If a and b belong to Z_m, then $a +_m b$ and $a \cdot_m b$ belong to Z_m.

Arithmetic Modulo m

The operations $+_m$ and $\cdot_m$ satisfy many of the same properties of ordinary addition and multiplication of integers:

Closure: If a and b belong to Z_m, then $a +_m b$ and $a \cdot_m b$ belong to Z_m.

Associativity: If a, b, and c belong to Z_m, then $(a +_m b) +_m c = a +_m (b +_m c)$ and $(a \cdot_m b) \cdot_m c = a \cdot_m (b \cdot_m c)$.

Arithmetic Modulo m

The operations $+_m$ and $\cdot_m$ satisfy many of the same properties of ordinary addition and multiplication of integers:

Closure: If a and b belong to Z_m, then $a +_m b$ and $a \cdot_m b$ belong to Z_m.

Associativity: If a, b, and c belong to Z_m, then $(a +_m b) +_m c = a +_m (b +_m c)$ and $(a \cdot_m b) \cdot_m c = a \cdot_m (b \cdot_m c)$.

Identity elements: $a +_m 0 = a$ and $a \cdot_m 1 = a$.

Arithmetic Modulo m

The operations $+_m$ and $\cdot_m$ satisfy many of the same properties of ordinary addition and multiplication of integers:

Closure: If a and b belong to Z_m, then $a +_m b$ and $a \cdot_m b$ belong to Z_m.

Associativity: If a, b, and c belong to Z_m, then $(a +_m b) +_m c = a +_m (b +_m c)$ and $(a \cdot_m b) \cdot_m c = a \cdot_m (b \cdot_m c)$.

Identity elements: $a +_m 0 = a$ and $a \cdot_m 1 = a$.

Additive inverses: If $a \neq 0$ and $a \in \mathrm{Z}_m$, then $m - a$ is an additive inverse of a modulo m. That is, $a +_m (m - a) = 0$ and $0 +_m 0 = 0$.

Arithmetic Modulo m

The operations $+_m$ and $\cdot_m$ satisfy many of the same properties of ordinary addition and multiplication of integers:

Closure: If a and b belong to Z_m, then $a +_m b$ and $a \cdot_m b$ belong to Z_m.

Associativity: If a, b, and c belong to Z_m, then $(a +_m b) +_m c = a +_m (b +_m c)$ and $(a \cdot_m b) \cdot_m c = a \cdot_m (b \cdot_m c)$.

Identity elements: $a +_m 0 = a$ and $a \cdot_m 1 = a$.

Additive inverses: If $a \neq 0$ and $a \in \mathrm{Z}_m$, then $m - a$ is an additive inverse of a modulo m. That is, $a +_m (m - a) = 0$ and $0 +_m 0 = 0$.

Commutativity: If $a, b \in \mathrm{Z}_m$, then $a +_m b = b +_m a$.

Arithmetic Modulo m

The operations $+_m$ and $\cdot_m$ satisfy many of the same properties of ordinary addition and multiplication of integers:

Closure: If a and b belong to Z_m, then $a +_m b$ and $a \cdot_m b$ belong to Z_m.

Associativity: If a, b, and c belong to Z_m, then $(a +_m b) +_m c = a +_m (b +_m c)$ and $(a \cdot_m b) \cdot_m c = a \cdot_m (b \cdot_m c)$.

Identity elements: $a +_m 0 = a$ and $a \cdot_m 1 = a$.

Additive inverses: If $a \neq 0$ and $a \in \mathrm{Z}_m$, then $m - a$ is an additive inverse of a modulo m. That is, $a +_m (m - a) = 0$ and $0 +_m 0 = 0$.

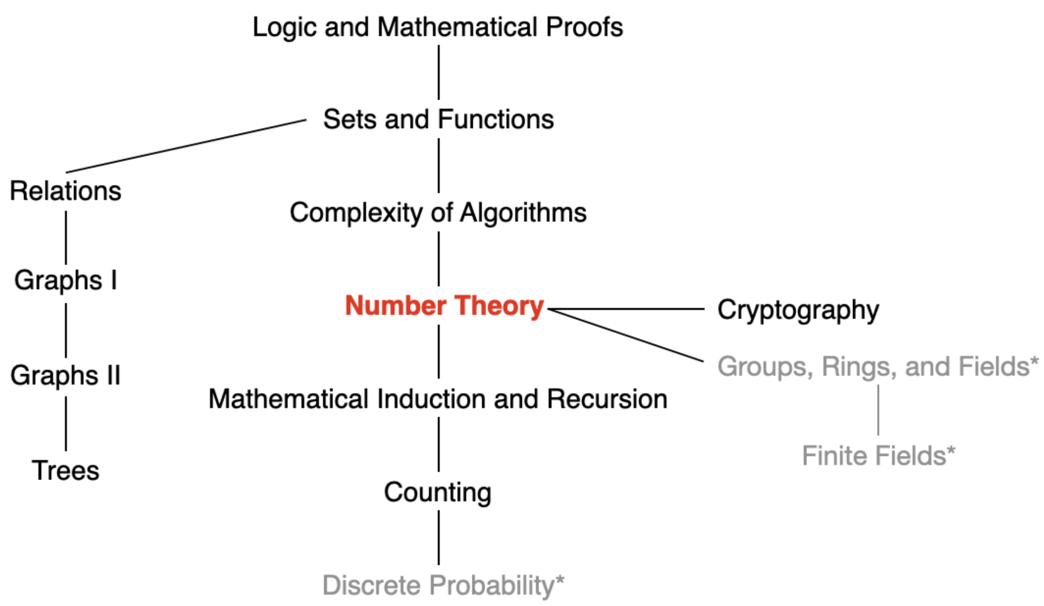
Commutativity: If $a, b \in \mathrm{Z}_m$, then $a +_m b = b +_m a$.

Distributivity: If $a, b, c \in \mathrm{Z}_m$, then

$$a \cdot_m (b +_m c) = (a \cdot_m b) +_m (a \cdot_m c)$$

$$(a +_m b) \cdot_m c = (a \cdot_m c) +_m (b \cdot_m c)$$

Number Theory



Number Theory: divisibility and modular arithmetic, integer representations, primes, greatest common divisors, ...

Representations of Integers

We may use decimal (base 10), binary, octal, hexadecimal, or other notations to represent integers.

Representations of Integers

We may use decimal (base 10), binary, octal, hexadecimal, or other notations to represent integers.

Let $b > 1$ be an integer. Then if n is a positive integer, it can be expressed uniquely in the form

$$n = a_k b^k + a_{k-1} b^{k-1} + \quad + a_1 b + a_0,$$

where k is nonnegative, a_k's are nonnegative integers less than b. The representation of n is called the base-b expansion of n and is denoted by $(a_k a_{k-1} \ldots a_1 a_0)_b$.

Base-b Expansions

From binary, octal, hexadecimal expansions to the decimal expansion:

Base-b Expansions

From binary, octal, hexadecimal expansions to the decimal expansion:

Example

- $(101011111)_2 = 2^8 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 351$
- $(7016)_8 = 7 \cdot 8^3 + 1 \cdot 8 + 6 = 3598$

Base-b Expansions

From binary, octal, hexadecimal expansions to the decimal expansion:

Example

- $(101011111)_2 = 2^8 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 351$
- $(7016)_8 = 7 \cdot 8^3 + 1 \cdot 8 + 6 = 3598$

Conversions between binary and octal (or hexadecimal) expansions:

Base-b Expansions

From binary, octal, hexadecimal expansions to the decimal expansion:

Example

- ◇ $(101011111)_2 = 2^8 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 351$
- ◇ $(7016)_8 = 7 \cdot 8^3 + 1 \cdot 8 + 6 = 3598$

Conversions between binary and octal (or hexadecimal) expansions:

Example

- ◇ $(101011111)_2 = (\underline{101}\overline{011}\underline{111}) = (537)_8$
- ◇ $(7016)_8 = (\underline{111}\overline{000}\underline{001}\overline{110})_2$
  $= (\underline{1110}\overline{0000}\underline{1110})_2 = (E0E)_{16}$

Base-b Expansions

From decimal expansion to the base-b expansion:

Base-b Expansions

From decimal expansion to the base-b expansion:

$$
\begin{aligned}
n &= a_k b^k + a_{k-1} b^{k-1} + a_{k-2} b^{k-2} + \cdots + a_2 b^2 + a_1 b + a_0 \\
&= b(a_k b^{k-1} + a_{k-1} b^{k-2} + a_{k-2} b^{k-3} + \cdots + a_2 b + a_1) + a_0 \\
&= b(b(a_k b^{k-2} + a_{k-1} b^{k-3} + a_{k-2} b^{k-4} + \cdots + a_2) + a_1) + a_0 \\
&= \cdots
\end{aligned}
$$

Base-b Expansions

From decimal expansion to the base-b expansion:

$$\begin{aligned} n &= a_k b^k + a_{k-1} b^{k-1} + a_{k-2} b^{k-2} + \cdots + a_2 b^2 + a_1 b + a_0 \\ &= b(a_k b^{k-1} + a_{k-1} b^{k-2} + a_{k-2} b^{k-3} + \cdots + a_2 b + a_1) + a_0 \\ &= b(b(a_k b^{k-2} + a_{k-1} b^{k-3} + a_{k-2} b^{k-4} + \cdots + a_2) + a_1) + a_0 \\ &= \cdots \end{aligned}$$

- Divide n by b to obtain $n = bq_0 + a_0$, with $0 \leq a0 < b$

Base-b Expansions

From decimal expansion to the base-b expansion:

$$\begin{aligned} n &= a_k b^k + a_{k-1} b^{k-1} + a_{k-2} b^{k-2} + \cdots + a_2 b^2 + a_1 b + a_0 \\ &= b(a_k b^{k-1} + a_{k-1} b^{k-2} + a_{k-2} b^{k-3} + \cdots + a_2 b + a_1) + a_0 \\ &= b(b(a_k b^{k-2} + a_{k-1} b^{k-3} + a_{k-2} b^{k-4} + \cdots + a_2) + a_1) + a_0 \\ &= \cdots \end{aligned}$$

- Divide n by b to obtain $n = bq_0 + a_0$, with $0 \le a0 < b$
- The remainder a_0 is the rightmost digit in the base-b; expansion of n. Then divide q_0 by b to get $q_0 = bq_1 + a_1$ with $0 \le a1 < b$;

Base-b Expansions

From decimal expansion to the base-b expansion:

$$\begin{aligned} n &= a_k b^k + a_{k-1} b^{k-1} + a_{k-2} b^{k-2} + \cdots + a_2 b^2 + a_1 b + a_0 \\ &= b(a_k b^{k-1} + a_{k-1} b^{k-2} + a_{k-2} b^{k-3} + \cdots + a_2 b + a_1) + a_0 \\ &= b(b(a_k b^{k-2} + a_{k-1} b^{k-3} + a_{k-2} b^{k-4} + \cdots + a_2) + a_1) + a_0 \\ &= \cdots \end{aligned}$$

- Divide n by b to obtain $n = bq_0 + a_0$, with $0 \leq a0 < b$
- The remainder a_0 is the rightmost digit in the base-b; expansion of n. Then divide q_0 by b to get $q_0 = bq_1 + a_1$ with $0 \leq a1 < b$;
- a_1 is the second digit from the right; continue by successively dividing the quotients by b until the quotient is 0

Base-b Expansions

procedure *base b expansion*(n, b: positive integers with $b > 1$)
$q := n$
$k := 0$
while ($q \neq 0$)
  $a_k := q$ **mod** b
  $q := q$ **div** b
  $k := k + 1$
return($a_{k-1}, \ldots, a_1, a_0$){($a_{k-1} \ldots a_1 a_0)_b$ is base b expansion of n}

Base-b Expansions

Example: Find the hexadecimal expansion of $(177130)_{10}$.

Base-b Expansions

Example: Find the hexadecimal expansion of $(177130)_{10}$.

Solution: First divide 177130 by 16 to obtain

$$177130 = 16 \cdot 11070 + 10.$$

Base-b Expansions

Example: Find the hexadecimal expansion of $(177130)_{10}$.

Solution: First divide 177130 by 16 to obtain

$$177130 = 16 \cdot 11070 + 10.$$

Successively dividing quotients by 16 gives

$$11070 = 16 \cdot 691 + 14,$$

$$691 = 16 \cdot 43 + 3,$$

$$43 = 16 \cdot 2 + 11,$$

$$2 = 16 \cdot 0 + 2.$$

The successive remainders that we have found, 10, 14, 3, 11, 2. It follows that $(177130)_{10} = (2B3EA)_{16}$.

Binary Addition of Integers

$a = (a_{n-1} a_{n-2} ... a_1 a_0)$, $b = (b_{n-1} b_{n-2} ... b_1 b_0)$

Binary Addition of Integers

$$a = (a_{n-1}a_{n-2}...a_1a_0),\ b = (b_{n-1}b_{n-2}...b_1b_0)$$

procedure *add*(a, b: positive integers)
{the binary expansions of a and b are $(a_{n-1}, a_{n-2}, \ldots, a_0)_2$ and $(b_{n-1}, b_{n-2}, \ldots, b_0)_2$, respectively}
$c := 0$
for $j := 0$ to $n - 1$
  $d := \lfloor (a_j + b_j + c)/2 \rfloor$
  $s_j := a_j + b_j + c - 2d$
  $c := d$
$s_n := c$
return$(s_0, s_1, \ldots, s_n)${the binary expansion of the sum is $(s_n, s_{n-1}, \ldots, s_0)_2$}

Binary Addition of Integers

$a = (a_{n-1}a_{n-2}...a_1a_0)$, $b = (b_{n-1}b_{n-2}...b_1b_0)$

procedure *add*(a, b: positive integers)
{the binary expansions of a and b are $(a_{n-1},a_{n-2},...,a_0)_2$ and $(b_{n-1},b_{n-2},...,b_0)_2$, respectively}
$c := 0$
for $j := 0$ to $n - 1$
    $d := \lfloor (a_j + b_j + c)/2 \rfloor$
    $s_j := a_j + b_j + c - 2d$
    $c := d$
$s_n := c$
return$(s_0, s_1, ..., s_n)${the binary expansion of the sum is $(s_n, s_{n-1}, ..., s_0)_2$}

$O(n)$ bit additions

Algorithm: Binary Multiplication of Integers

$a = (a_{n-1}a_{n-2}...a_1a_0)_2, \ b = (b_{n-1}b_{n-2}...b_1b_0)_2$

$ab = a(b_0 2^0 + b_1 2^1 + \ + b_{n-1}2^{n-1}) = a(b_0 2^0) + a(b_1 2^1) + \ + a(b_{n-1}2^{n-1})$

Algorithm: Binary Multiplication of Integers

$a = (a_{n-1}a_{n-2}...a_1a_0)_2, \ b = (b_{n-1}b_{n-2}...b_1b_0)_2$

$ab = a(b_0 2^0 + b_1 2^1 + \ + b_{n-1}2^{n-1}) = a(b_0 2^0) + a(b_1 2^1) + \ + a(b_{n-1}2^{n-1})$

procedure *multiply*(a, b: positive integers)
{the binary expansions of a and b are $(a_{n-1}, a_{n-2}, ..., a_0)_2$ and $(b_{n-1}, b_{n-2}, ..., b_0)_2$, respectively}
for $j := 0$ to $n-1$
    if $b_j = 1$ **then** $c_j = a$ shifted j places
    else $c_j := 0$
{$c_0, c_1, ..., c_{n-1}$ are the partial products}
$p := 0$
for $j := 0$ to $n-1$
  $p := p + c_j$
return p {p is the value of ab}

Algorithm: Binary Multiplication of Integers

$a = (a_{n-1}a_{n-2}...a_1a_0)_2, \; b = (b_{n-1}b_{n-2}...b_1b_0)_2$

$ab = a(b_0 2^0 + b_1 2^1 + \; + b_{n-1}2^{n-1}) = a(b_0 2^0) + a(b_1 2^1) + \; + a(b_{n-1}2^{n-1})$

procedure *multiply*(a, b: positive integers)
{the binary expansions of a and b are $(a_{n-1},a_{n-2},...,a_0)_2$ and $(b_{n-1},b_{n-2},...,b_0)_2$, respectively}
for $j := 0$ to $n - 1$
  if $b_j = 1$ **then** $c_j = a$ shifted j places
  else $c_j := 0$
{$c_0, c_1, ..., c_{n-1}$ are the partial products}
$p := 0$
for $j := 0$ to $n - 1$
  $p := p + c_j$
return p {p is the value of ab}

$O(n^2)$ shifts and $O(n^2)$ bit additions

SUSTech Southern University of Science and Technology

Algorithm: Computing div and mod

Compute $q = a$ div d and $r = a$ mod d:

procedure *division algorithm* (a: integer, d: positive integer)
$q := 0$
$r := |a|$
while $r \geq d$
    $r := r - d$
    $q := q + 1$
if $a < 0$ and $r > 0$ **then**
    $r := d - r$
    $q := -(q+1)$
return (q, r) $\{q = a$ **div** d *is the quotient*, $r = a$ **mod** d *is the remainder* $\}$

Algorithm: Computing div and mod

Compute $q = a$ div d and $r = a$ mod d:

procedure *division algorithm* (a: integer, d: positive integer)
$q := 0$
$r := |a|$
while $r \geq d$
    $r := r - d$
    $q := q + 1$
if $a < 0$ and $r > 0$ **then**
    $r := d - r$
    $q := -(q+1)$
return (q, r) {$q = a$ **div** d *is the quotient*, $r = a$ **mod** d *is the remainder* }

$O(q \log a)$ bit operations. But there exist more efficient algorithms with complextiy $O(n^2)$, where $n = \max(\log a, \log d)$

Algorithm: Binary Modular Exponentiation

Compute $b^n \bmod m$:

Algorithm: Binary Modular Exponentiation

Compute b^n mod m: Let $n = (a_{k-1} \ldots a_1 a_0)_2$.

$$b^n = b^{a_{k-1} \cdot 2^{k-1} + \cdots + a_1 \cdot 2 + a_0} = b^{a_{k-1} \cdot 2^{k-1}} \cdots b^{a_1 \cdot 2} \cdot b^{a_0}$$

Algorithm: Binary Modular Exponentiation

Compute $b^n \bmod m$: Let $n = (a_{k-1}...a_1 a_0)_2$.

$$b^n = b^{a_{k-1}\cdot 2^{k-1}+\cdots+a_1\cdot 2+a_0} = b^{a_{k-1}\cdot 2^{k-1}} \cdots b^{a_1\cdot 2} \cdot b^{a_0}$$

Successively finds $b \bmod m$, $b^2 \bmod m$, $b^4 \bmod m$, . . . , $b^{2^{k-1}} \bmod m$, and multiplies together the terms b^{2^j}, where $a_j = 1$.

Algorithm: Binary Modular Exponentiation

Compute b^n mod m: Let $n = (a_{k-1}...a_1a_0)_2$.

$$b^n = b^{a_{k-1}\cdot 2^{k-1}+\cdots+a_1\cdot 2+a_0} = b^{a_{k-1}\cdot 2^{k-1}}\cdots b^{a_1\cdot 2}\cdot b^{a_0}$$

Successively finds b mod m, b^2 mod m, b^4 mod m, . . . , $b^{2^{k-1}}$ mod m, and multiplies together the terms b^{2^j}, where $a_j = 1$.

```
procedure modular exponentiation(b: integer, n = (a_{k-1}a_{k-2}...a_1a_0)_2 , m: positive
    integers)
x := 1
power := b mod m
for i := 0 to k − 1
    if a_i = 1 then x := (x · power) mod m
    power := (power · power) mod m
return x {x equals b^n mod m}
```

Recall that

$$ab \equiv ((a \bmod m)(b \bmod m))(\bmod\ m).$$

Algorithm: Binary Modular Exponentiation

Use the algorithm to find 3^{644} mod 645:

```
procedure modular exponentiation(b: integer, n = (a_{k-1}a_{k-2}...a_1a_0)_2 , m: positive
    integers)
x := 1
power := b mod m
for i := 0 to k − 1
    if a_i = 1 then x := (x · power) mod m
    power := (power · power) mod m
return x {x equals b^n mod m }
```

Algorithm: Binary Modular Exponentiation

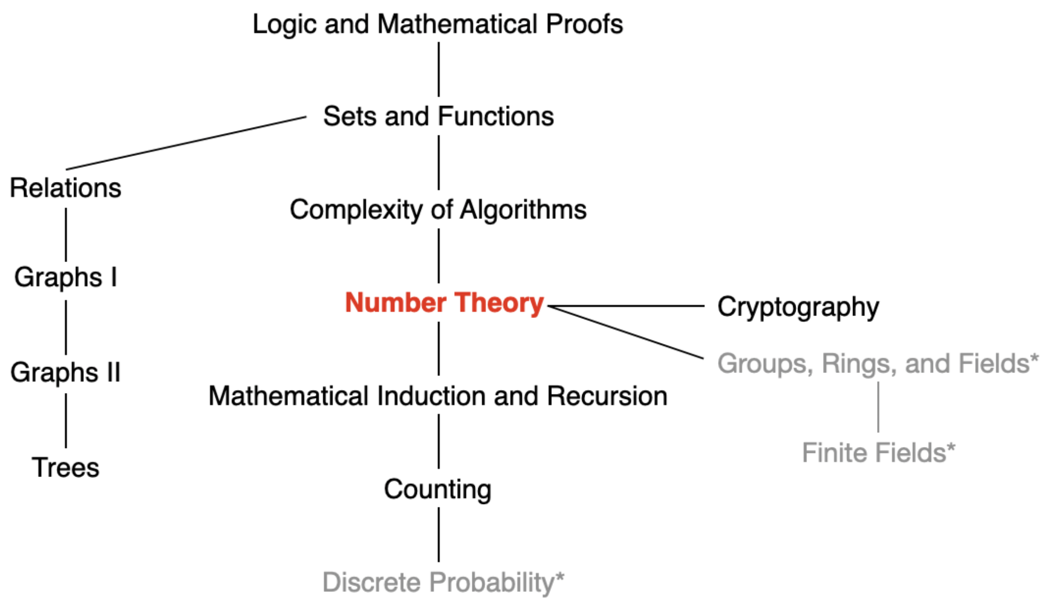
Use the algorithm to find 3^{644} mod 645:

procedure *modular exponentiation*(b: integer, $n = (a_{k-1}a_{k-2}\ldots a_1a_0)_2$, m: positive integers)
$x := 1$
$power := b$ **mod** m
for $i := 0$ to $k - 1$
  if $a_i = 1$ **then** $x := (x \cdot power)$ **mod** m
  $power := (power \cdot power)$ **mod** m
return x {x equals b^n **mod** m }

The algorithm initially sets $x = 1$ and $power = 3 \bmod 645 = 3$. The binary expansion of 644 is $(1010000100)_2$. Here are the steps used:

$i = 0$: Because $a_0 = 0$, we have $x = 1$ and $power = 3^2 \textbf{ mod } 645 = 9 \textbf{ mod } 645 = 9$;
$i = 1$: Because $a_1 = 0$, we have $x = 1$ and $power = 9^2 \textbf{ mod } 645 = 81 \textbf{ mod } 645 = 81$;
$i = 2$: Because $a_2 = 1$, we have $x = 1 \cdot 81 \textbf{ mod } 645 = 81$ and $power = 81^2 \textbf{ mod } 645 = 6561 \textbf{ mod } 645 = 111$;
$i = 3$: Because $a_3 = 0$, we have $x = 81$ and $power = 111^2 \textbf{ mod } 645 = 12{,}321 \textbf{ mod } 645 = 66$;
$i = 4$: Because $a_4 = 0$, we have $x = 81$ and $power = 66^2 \textbf{ mod } 645 = 4356 \textbf{ mod } 645 = 486$;
$i = 5$: Because $a_5 = 0$, we have $x = 81$ and $power = 486^2 \textbf{ mod } 645 = 236{,}196 \textbf{ mod } 645 = 126$;
$i = 6$: Because $a_6 = 0$, we have $x = 81$ and $power = 126^2 \textbf{ mod } 645 = 15{,}876 \textbf{ mod } 645 = 396$;
$i = 7$: Because $a_7 = 1$, we find that $x = (81 \cdot 396) \textbf{ mod } 645 = 471$ and $power = 396^2 \textbf{ mod } 645 = 156{,}816 \textbf{ mod } 645 = 81$;
$i = 8$: Because $a_8 = 0$, we have $x = 471$ and $power = 81^2 \textbf{ mod } 645 = 6561 \textbf{ mod } 645 = 111$;
$i = 9$: Because $a_9 = 1$, we find that $x = (471 \cdot 111) \textbf{ mod } 645 = 36$.

Next Lecture



Number Theory: divisibility and modular arithmetic, integer representations, primes, greatest common divisors, ...