Discrete Mathematics

Shizhong Liao

szliao@tju.edu.cn

Computer Science Department, Tianjin University

October 22, 2012

Contents

Outline

Outline

Introduction

Definition 3.1 (Algorithm)

An *algorithm* is a finite set of precise instructions for performing a computation for solving a problem.

Remarks

Introduction

Definition 3.1 (Algorithm)

An *algorithm* is a finite set of precise instructions for performing a computation for solving a problem.

Remarks

An algorithm MUST terminate,
since it is a *recursive* (or Turing decidable) computational process.

Introduction

Definition 3.1 (Algorithm)

An *algorithm* is a finite set of precise instructions for performing a computation for solving a problem.

Remarks

- Input.
- Output.
- Definiteness.
- Correctness.
- Finiteness.
- Effectiveness.
- Generality.

Introduction

Example 3.2

Describe an algorithm for finding the maximum (largest) value in a finite sequence of integers.

Solution 3.2

Algorithm 3.1: *Finding the Maximum Element.*

procedure $max(a_1, a_2, \ldots, a_n$: integer)
$max := a_1$
for $i := 2$ **to** n **do**
  if $max < a_i$ **then**
    $max := a_i$
  end
end
{max is the largest element}

Introduction

Example 3.2

Describe an algorithm for finding the maximum (largest) value in a finite sequence of integers.

Solution 3.2

Algorithm 3.1: *Finding the Maximum Element.*

procedure $max(a_1, a_2, \ldots, a_n$: integer)
$max := a_1$
for $i := 2$ **to** n **do**
  if $max < a_i$ **then**
    $max := a_i$
  end
end
{max is the largest element}

Searching Algorithms

The Searching Problem

Locate an element x in a list of distinct elements $a_1, a_2, \ldots, a_n$, or determine that it is not in the list.

Searching Algorithms

The Searching Problem

Locate an element x in a list of distinct elements $a_1, a_2, \ldots, a_n$, or determine that it is not in the list.

Algorithm 3.2: The Linear Search Algorithm.

procedure *linearsearch*(x : integer, $a_1, \ldots, a_n$: distinct integers)
$i := 1$
while $i \leq n \wedge x \neq a_i$ **do**
  $i := i + 1$
end
if $i \leq n$ **then**
  location $:= i$
else
  location $:= 0$
end
{*location* is the subscript of the term that equal x,}
{or is 0 if x is not found}

Searching Algorithms

The Searching Problem

Locate an element x in a list of distinct elements $a_1, a_2, \ldots, a_n$, or determine that it is not in the list.

Algorithm 3.3: The Binary Search Algorithm.

procedure *binarysearch*(x : integer, $a_1, \ldots, a_n$: increasing integers)
$i := 1$ {i is left endpoit of search interval}
$j := n$ {j is right endpoint of search interval}
while $i < j$ **do**
  $m := \lfloor (i+j)/2 \rfloor$
  if $x > a_m$ **then** $i := m+1$ **else** $j := m$
end
if $x = a_i$ **then** *location* $:= i$ **else** *location* $:= 0$
{*location* is the subscript of the term that equal x,}
{or is 0 if x is not found}

Sorting

Sorting is putting elements into a list in which the elements are in increasing order.

Sorting

Sorting is putting elements into a list in which the elements are in increasing order.

Algorithm 3.4: The Bubble Sort.

procedure *bubblesort*($a_1, \ldots, a_n$: real numbers with $n \geq 2$)
for $i = 1$ **to** $n - 1$ **do**
  for $j := 1$ **to** $n - i$ **do**
    if $a_j > a_{j+1}$ **then** interchange a_j and a_{j+1}
  end
end
$\{a_1, \ldots, a_n$ is in increasing order$\}$

Sorting

Sorting is putting elements into a list in which the elements are in increasing order.

Algorithm 3.5: The Insertion Sort.

procedure *insertionsort*$(a_1, \ldots, a_n$: real numbers with $n \geq 2)$
for $j := 2$ **to** n **do**
  $i := 1$
  while $a_j > a_i$ **do** $i := i + 1$
  $m := a_j$
  for $k := 0$ **to** $j - i - 1$ **do**
    $a_{j-k} := a_{j-k-1}$
  end
  $a_j := m$
end
$\{a_1, \ldots, a_n$ is in increasing order$\}$

Greedy Algorithms

Optimization problems are to minimize or maximize the value of some parameter.

Greedy algorithms are algorithms that make what seem to be "best" choice at each step.

Once we know that a greedy algorithm finds a feasible solution, we need to determine whether it has found an optimal solution.

Greedy Algorithms

Example 3.3

Consider the problem of making n cents change with quarters, dimes, nickels, and pennies, and using the least total number of coins.

Algorithm 3.6: Greedy Change-Making Algorithm.

procedure *change*$(d_1, \ldots, d_r,\ n,\ c_1, \ldots, c_r)$
Data: d_i: values of denominations of coins,
$d_i > d_{i+1}, i = 1, \ldots, r$
Result: $c_1, \ldots, c_r : \sum_{i=1}^{r} c_i d_i = n$
for $i := 1$ **to** r **do**
  while $n \geq d_i$ **do**
    $c_i := c_i + 1$
    $n := n - d_i$
  end
end

Greedy Algorithms

Example 3.3

Consider the problem of making n cents change with quarters, dimes, nickels, and pennies, and using the least total number of coins.

Algorithm 3.6: Greedy Change-Making Algorithm.

procedure *change*$(d_1, \ldots, d_r,\ n,\ c_1, \ldots, c_r)$
Data: d_i: values of denominations of coins,
$d_i > d_{i+1}, i = 1, \ldots, r$
Result: $c_1, \ldots, c_r : \sum_{i=1}^{r} c_i d_i = n$
for $i := 1$ **to** r **do**
  while $n \geq d_i$ **do**
    $c_i := c_i + 1$
    $n := n - d_i$
  end
end

Greedy Algorithms

Lemma 3.4

If n is a positive integer, then n cents in change using quarters, dimes, nickels, and pennies using the fewest coins possible has at most two dimes, at most one nickel, at most four pennies,
and cannot have two dimes and a nickel.
The amount of change in dimes, nickels, and pennies cannot exceed 24 cents.

Proof.

Proof by contradiction.

Note that

$$3 \text{ dimes} = 1 \text{ quarter} + 1 \text{ nickel}$$

$$2 \text{ nikels} = 1 \text{ dime}$$

$$5 \text{ pennies} = 1 \text{ nickel}$$

So we can have at most two dimes, one nickel, and four pennies, but we cannot have two dimes and five pennies, it follows that 24 cents is the most money we can have in dimes, nickels, and pennies when we make change using the fewest number of coins for n cents. □

Greedy Algorithms

Lemma 3.4

If n is a positive integer, then n cents in change using quarters, dimes, nickels, and pennies using the fewest coins possible has at most two dimes, at most one nickel, at most four pennies,
and cannot have two dimes and a nickel.
The amount of change in dimes, nickels, and pennies cannot exceed 24 cents.

Proof.

Proof by contradiction.

Note that

$$3 \text{ dimes} = 1 \text{ quarter} + 1 \text{ nickel}$$

$$2 \text{ nikels} = 1 \text{ dime}$$

$$5 \text{ pennies} = 1 \text{ nickel}$$

So we can have at most two dimes, one nickel, and four pennies, but we cannot have two dimes and five pennies, it follows that 24 cents is the most money we can have in dimes, nickels, and pennies when we make change using the fewest number of coins for n cents. □

Greedy Algorithms

Theorem 3.5

Algorithm 3.6 produces change using the fewest coins possible.

Proof.

Proof by contradiction.

□

Greedy Algorithms

Theorem 3.5

Algorithm 3.6 produces change using the fewest coins possible.

Proof.

Proof by contradiction.

□

Greedy Algorithms

Theorem 3.5

Algorithm 3.6 produces change using the fewest coins possible.

Proof.

Proof by contradiction.

The optimal way and the greedy algorithm use the same number of quarters.
First, the optimal uses no more quarters.
Since the greedy algorithm uses the most quarters possible.

However, the optimal uses no less quarters.
If it were, 25 cents would be made up from dimes, nickels and pennies.
But this is impossible by Lemma 3.4.

Greedy Algorithms

Theorem 3.5

Algorithm 3.6 produces change using the fewest coins possible.

Proof.

Proof by contradiction.

Because there must be the same number of quarters in the two ways to make change, the value of the dimes, nickels, and pennies in these two ways must be the same, and these coins are worth no more than 24 cents.

There must be the same number of dimes, because the greedy algorithm used the most dimes possible and by Lemma 3.4, when change is made using the fewest coins possible, at most one nickel and at most four pennies are used, so that the most dimes possible are also used in the optimal way to make change.
Similarly, we have the same number of nickels and, finally, the same number of pennies.

□

The Halting Problem

Example 3.6

The halting problem:

Is there an algorithm that determines whether a program terminates?

Solution 3.6

Assume there exists such algorithm called $H(P, I)$, where P is a program and I is the input to P. $H(P, I)$ outputs "halt" if H determines that P stops when given I as input. Otherwise, $H(P, I)$ generates "loop forever".

Construct an algorithm $K(P)$ as follow.

$$K(P) = \begin{cases} \text{“halt”}, & \text{if } H(P,P) = \text{“loop forever”}; \\ \text{“loop forever”}, & \text{if } H(P,P) = \text{“halt”}. \end{cases}$$

Consider $K(K)$,

$$K(K) = \text{“loop forever”} \iff K(K) = \text{“halt”}.$$

Contradiction.

Therefore, there is no algorithm that solves the halting problem.

The Halting Problem

Example 3.6

The halting problem:

Is there an algorithm that determines whether a program terminates?

Solution 3.6

Assume there exists such algorithm called $H(P, I)$, where P is a program and I is the input to P. $H(P, I)$ outputs "halt" if H determines that P stops when given I as input. Otherwise, $H(P, I)$ generates "loop forever".

Construct an algorithm $K(P)$ as follow.

$$K(P) = \begin{cases} \text{“halt”}, & \text{if } H(P,P) = \text{“loop forever”}; \\ \text{“loop forever”}, & \text{if } H(P,P) = \text{“halt”}. \end{cases}$$

Consider $K(K)$,

$$K(K) = \text{“loop forever”} \iff K(K) = \text{“halt”}.$$

Contradiction.

Therefore, there is no algorithm that solves the halting problem.

Outline

Definitions

Lef f and g be functions from the set of integers or the set of real numbers to the set of real numbers, C and k are constants.

Definition 3.7 (O, Big-O)

We say that $f(x)$ is $O(g(x))$ if there are C and k such that

$$|f(x)| \leq C|g(x)|$$

where $x > k$.

Definition 3.8 (Ω, Big-Ω)

We say that $f(x)$ is $\Omega(g(x))$ if there are C and k such that

$$|f(x)| \geq C|g(x)|$$

where $x > k$.

Definition 3.9 (Θ, Big-Θ)

We say that $f(x)$ is $\Theta(g(x))$ if $f(x)$ is $O(g(x))$ and $f(x)$ is $\Omega(g(x))$. We also say that $f(x)$ is of *order* $g(x)$.

Definitions

Lef f and g be functions from the set of integers or the set of real numbers to the set of real numbers, C and k are constants.

Definition 3.7 (O, Big-O)

We say that $f(x)$ is $O(g(x))$ if there are C and k such that

$$|f(x)| \leq C|g(x)|$$

where $x > k$.

Definition 3.8 (Ω, Big-Ω)

We say that $f(x)$ is $\Omega(g(x))$ if there are C and k such that

$$|f(x)| \geq C|g(x)|$$

where $x > k$.

Definition 3.9 (Θ, Big-Θ)

We say that $f(x)$ is $\Theta(g(x))$ if $f(x)$ is $O(g(x))$ and $f(x)$ is $\Omega(g(x))$. We also say that $f(x)$ is of *order* $g(x)$.

Definitions

Lef f and g be functions from the set of integers or the set of real numbers to the set of real numbers, C and k are constants.

Definition 3.7 (O, Big-O)

We say that $f(x)$ is $O(g(x))$ if there are C and k such that

$$|f(x)| \leq C|g(x)|$$

where $x > k$.

Definition 3.8 (Ω, Big-Ω)

We say that $f(x)$ is $\Omega(g(x))$ if there are C and k such that

$$|f(x)| \geq C|g(x)|$$

where $x > k$.

Definition 3.9 (Θ, Big-Θ)

We say that $f(x)$ is $\Theta(g(x))$ if $f(x)$ is $O(g(x))$ and $f(x)$ is $\Omega(g(x))$. We also say that $f(x)$ is of *order* $g(x)$.

Definitions

The constants C and k in the definitions are called *witnesses*.
To establish that $f(x)$ is $O(g(x))$ (or $f(x)$ is $\Omega(g(x))$),
we need only one pair of witnesses to this relationship.

Note that there are infinitely many pairs of witnesses.

The facts that

$$f(x) \text{ is } O(g(x)),$$
$$f(x) \text{ is } \Omega(g(x)),$$
$$f(x) \text{ is } \Theta(g(x)),$$

is sometimes written

$$f(x) = O(g(x)),$$
$$f(x) = \Omega(g(x)),$$
$$f(x) = \Theta(g(x)).$$

It is acceptable to write

$$f(x) \in O(g(x)),$$
$$f(x) \in \Omega(g(x)),$$
$$f(x) \in \Theta(g(x)).$$

Definitions

The constants C and k in the definitions are called *witnesses*.
To establish that $f(x)$ is $O(g(x))$ (or $f(x)$ is $\Omega(g(x))$),
we need only one pair of witnesses to this relationship.

Note that there are infinitely many pairs of witnesses.

The facts that

$$f(x) \text{ is } O(g(x)),$$
$$f(x) \text{ is } \Omega(g(x)),$$
$$f(x) \text{ is } \Theta(g(x)),$$

is sometimes written

$$f(x) = O(g(x)),$$
$$f(x) = \Omega(g(x)),$$
$$f(x) = \Theta(g(x)).$$

It is acceptable to write

$$f(x) \in O(g(x)),$$
$$f(x) \in \Omega(g(x)),$$
$$f(x) \in \Theta(g(x)).$$

Definitions

Example 3.10

Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$.

Solution 3.10

It follows that

$$0 \leq x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2$$

whenever $x > 1$.
Consequently, we can take $C = 4$ *and* $k = 1$ *as witnesses to show that* $f(x)$ *is* $O(x^2)$.

Definitions

Example 3.10

Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$.

Solution 3.10

It follows that

$$0 \leq x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2$$

whenever $x > 1$.
Consequently, we can take $C = 4$ *and* $k = 1$ *as witnesses to show that* $f(x)$ *is* $O(x^2)$.

Definitions

Example 3.11

Show that n^2 is not $O(n)$.

Solution 3.11

Observe that when $n > 0$ we can divide both sides of $n^2 \leq Cn$ by n, and obtain $n \leq C$.
We now see that no matter what C and k are,
the inequality $n \leq C$ cannot hold for all n with $n > k$.

In particular, once we set a value of k,
we see that when n is larger than the maximum of k and C,
it is not true that $n \leq C$ even though $n > k$.

Definitions

Example 3.11

Show that n^2 is not $O(n)$.

Solution 3.11

Observe that when $n > 0$ we can divide both sides of $n^2 \leq Cn$ by n, and obtain $n \leq C$.
We now see that no matter what C and k are,
the inequality $n \leq C$ cannot hold for all n with $n > k$.

In particular, once we set a value of k,
we see that when n is larger than the maximum of k and C,
it is not true that $n \leq C$ even though $n > k$.

Important Results

Theorem 3.12

Let $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$, where $a_0, a_1, \ldots, a_{n-1}, a_n$ are real numbers. Then $f(x)$ is $O(x^n)$.

Proof.

If $x > 1$ we have

$$\begin{aligned} |f(x)| &= |a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0| \\ &\leq |a_n| x^n + |a_{n-1}| x^{n-1} + \cdots + |a_1| x + |a_0| \\ &= x^n(|a_n| + |a_{n-1}|/x + \cdots + |a_1|/x^{n-1} + |a_0|/x^n) \\ &\leq x^n(|a_n| + |a_{n-1}| + \cdots + |a_1| + |a_0|). \end{aligned}$$

This shows that

$$|f(x)| \leq Cx^n,$$

where $C = |a_n| + |a_{n-1}| + \cdots + |a_1| + |a_0|$ whenever $x > 1$.
Hence, the witnesses $C = |a_n| + |a_{n-1}| + \cdots + |a_1| + |a_0|$ and $k = 1$ show that $f(x)$ is $O(x^n)$.

Important Results

Theorem 3.12

Let $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$,
where $a_0, a_1, \ldots, a_{n-1}, a_n$ *are real numbers.*
Then $f(x)$ *is* $O(x^n)$.

Proof.

If $x > 1$ we have

$$\begin{aligned} |f(x)| &= |a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0| \\ &\leq |a_n|x^n + |a_{n-1}|x^{n-1} + \cdots + |a_1|x + |a_0| \\ &= x^n(|a_n| + |a_{n-1}|/x + \cdots + |a_1|/x^{n-1} + |a_0|/x^n) \\ &\leq x^n(|a_n| + |a_{n-1}| + \cdots + |a_1| + |a_0|). \end{aligned}$$

This shows that

$$|f(x)| \leq Cx^n,$$

where $C = |a_n| + |a_{n-1}| + \cdots + |a_1| + |a_0|$ whenever $x > 1$.
Hence, the witnesses $C = |a_n| + |a_{n-1}| + \cdots + |a_1| + |a_0|$ and $k = 1$ show that $f(x)$ is $O(x^n)$.

Important Results

Theorem 3.13

Suppose that $f_1(x)$ *is* $O(g_1(x))$ *and* $f_2(x)$ *is* $O(g_2(x))$. *Then* $(f_1 + f_2)(x)$ *is* $O(\max(|g_1(x)|, |g_2(x)|))$.

Proof.

□

Important Results

Theorem 3.13

Suppose that $f_1(x)$ *is* $O(g_1(x))$ *and* $f_2(x)$ *is* $O(g_2(x))$. *Then* $(f_1 + f_2)(x)$ *is* $O(\max(|g_1(x)|, |g_2(x)|))$.

Proof.

From the definition of big-O,
there are constants C_1, C_2, k_1, and k_2 such that

$$|f_1(x)| \leq C_1|g_1(x)|$$

when $x > k_1$, and

$$|f_2(x)| \leq C_2|g_2(x)|$$

when $x > k_2$.
Note that

$$\begin{aligned} |(f_1 + f_2)(x)| &= |f_1(x) + f_2(x)| \\ &\leq |f_1(x)| + |f_2(x)|. \end{aligned}$$

Important Results

Theorem 3.13

Suppose that $f_1(x)$ *is* $O(g_1(x))$ *and* $f_2(x)$ *is* $O(g_2(x))$. *Then* $(f_1 + f_2)(x)$ *is* $O(\max(|g_1(x)|, |g_2(x)|))$.

Proof.

When x is greater than both k_1 and k_2, it follows that

$$\begin{aligned} |f_1(x)| + |f_2(x)| &\leq C_1|g_1(x)| + C_2|g_2(x)| \\ &\leq C_1|g(x)| + C_2|g(x)| \\ &= (C_1 + C_2)|g(x)| \\ &= C|g(x)| \end{aligned}$$

where $C = C_1 + C_2$ and $g(x) = \max(|g_1(x)|, |g_2(x)|)$.
This inequality shows that $|(f_1 + f_2)(x)| \leq C|g(x)|$ whenever $x > k$, where $k = \max(k_1, k_2)$. □

Important Results

Corollary 3.14

Suppose that $f_1(x)$ and $f_2(x)$ are both $O(g(x))$.
Then $(f_1 + f_2)(x)$ is $O(g(x))$.

Theorem 3.15

Suppose that $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$.
Then $(f_1 f_2)(x)$ is $O(g_1(x)g_2(x))$.

Theorem 3.16

Let $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$,
where $a_0, a_1, \ldots, a_{n-1}, a_n$ are real numbers.
Then $f(x)$ is of order x^n, that is, $f(x)$ is $\Theta(x^n)$.

Important Results

Corollary 3.14

Suppose that $f_1(x)$ and $f_2(x)$ are both $O(g(x))$.
Then $(f_1 + f_2)(x)$ is $O(g(x))$.

Theorem 3.15

Suppose that $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$.
Then $(f_1 f_2)(x)$ is $O(g_1(x)g_2(x))$.

Theorem 3.16

Let $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$,
where $a_0, a_1, \ldots, a_{n-1}, a_n$ are real numbers.
Then $f(x)$ is of order x^n, that is, $f(x)$ is $\Theta(x^n)$.

Important Results

Corollary 3.14

Suppose that $f_1(x)$ *and* $f_2(x)$ *are both* $O(g(x))$.
Then $(f_1 + f_2)(x)$ *is* $O(g(x))$.

Theorem 3.15

Suppose that $f_1(x)$ *is* $O(g_1(x))$ *and* $f_2(x)$ *is* $O(g_2(x))$.
Then $(f_1 f_2)(x)$ *is* $O(g_1(x)g_2(x))$.

Theorem 3.16

Let $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$,
where $a_0, a_1, \ldots, a_{n-1}, a_n$ *are real numbers.*
Then $f(x)$ *is of order* x^n, *that is,* $f(x)$ *is* $\Theta(x^n)$.

Outline

Complexity of Algorithms

Computational Complexity

- Time Complexity
- Space Complexity

Analysis Approach

- Problem Size n
- Complexity Measure $\Theta(f(n))$ $(\Omega(f(n)), O(f(n)))$.

Complexity Types

- Worst-Case Complexity
- Average-Case Complexity

Complexity of Algorithms

Computational Complexity

- Time Complexity
- Space Complexity

Analysis Approach

- Problem Size n
- Complexity Measure $\Theta(f(n))$ $(\Omega(f(n)), O(f(n)))$.

Complexity Types

- Worst-Case Complexity
- Average-Case Complexity

Complexity of Algorithms

Computational Complexity

- Time Complexity
- Space Complexity

Analysis Approach

- Problem Size n
- Complexity Measure $\Theta(f(n))$ $(\Omega(f(n)), O(f(n)))$.

Complexity Types

- Worst-Case Complexity
- Average-Case Complexity

Complexity of Algorithms

$$\text{Problems} = \begin{cases} \text{Unsolvable} & \begin{cases} \text{Uncomputable} \\ \text{Recursively Enumerable (r. e.)} \end{cases} \\ \text{Solvable} & \begin{cases} \text{Tractable} \\ \text{Intractable} \end{cases} \end{cases}$$

- Solvable: Computable, Recursive
  Problems that can be solved using algorithms.
- Recursively enumerable: Semi-computable
  Problems that can be tackled using programs.
- Tractable: **P**
  Having polynomial complexity algorithm solution.
- **NP**
  Checkable using polynomial complexity algorithms,
  Having polynomial complexity in nondeterministic TM.
- **NP**-complete
  In **NP** and if solvable in polynomial complexity,
  then **NP** = **P**.
- Conjecture: **NP** $\neq$ **P**.

Complexity of Algorithms

$$\text{Problems} = \begin{cases} \text{Unsolvable} & \begin{cases} \text{Uncomputable} \\ \text{Recursively Enumerable (r. e.)} \end{cases} \\ \text{Solvable} & \begin{cases} \text{Tractable} \\ \text{Intractable} \end{cases} \end{cases}$$

- Solvable: Computable, Recursive
  Problems that can be solved using algorithms.
- Recursively enumerable: Semi-computable
  Problems that can be tackled using programs.
- Tractable: **P**
  Having polynomial complexity algorithm solution.
- **NP**
  Checkable using polynomial complexity algorithms,
  Having polynomial complexity in nondeterministic TM.
- **NP**-complete
  In **NP** and if solvable in polynomial complexity,
  then **NP** = **P**.
- Conjecture: **NP** $\neq$ **P**.

Complexity of Algorithms

Table 3.1: Commonly Used Terminology for the Complexity of Algorithms

Complexity	Terminology
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n \log n)$	$n \log n$ complexity
$\Theta(n^b)$	Polynomial complexity
$\Theta(b^n), b > 1$	Exponential complexity
$\Theta(n!)$	Factorial complexity

Outline

Division

Definition 3.17

If a and b are integers with $a \neq 0$,
we say that a *divides* b if there is an integer c such that $b = ac$.

When a divides b we say that a is a *factor* of b and that b is a *multiple* of a.

The notation $a \mid b$ denotes that a divides b.
We write $a \nmid b$ when a does not divide b.

Division

Theorem 3.18

Let a, b, and c be integers. Then

1. *if $a \mid b$ and $a \mid c$, then $a \mid (b+c)$;*
2. *if $a \mid b$, then $a \mid bc$ for all integers c;*
3. *if $a \mid b$ and $b \mid c$, then $a \mid c$.*

Corollary 3.19

If a, b, and c are integers such that $a \mid b$ and $a \mid c$, then $a \mid mb+nc$ whenever m and n are integers.

Division

Theorem 3.18

Let a, b, and c be integers. Then

1. *if $a \mid b$ and $a \mid c$, then $a \mid (b + c)$;*
2. *if $a \mid b$, then $a \mid bc$ for all integers c;*
3. *if $a \mid b$ and $b \mid c$, then $a \mid c$.*

Corollary 3.19

If a, b, and c are integers such that $a \mid b$ and $a \mid c$, then $a \mid mb + nc$ whenever m and n are integers.

The Division Algorithm

Theorem 3.20

Let a be an integer and d a positive integer.
Then there are unique integers q and r, with $0 \leq r < d$,
such that $a = dq + r$.

Definition 3.21

In Theorem 3.20,
d is called the *divisor*,
a is called the *dividend*,
q is called the *quotient*,
and r is called the *remainder*.

This notation is used to express the quotient and remainder:

$$q = a \operatorname{div} d, \quad r = a \bmod d.$$

The Division Algorithm

Theorem 3.20

Let a be an integer and d a positive integer.
Then there are unique integers q and r, with $0 \leq r < d$,
such that $a = dq + r$.

Definition 3.21

In Theorem 3.20,
d is called the *divisor*,
a is called the *dividend*,
q is called the *quotient*,
and r is called the *remainder*.

This notation is used to express the quotient and remainder:

$$q = a \operatorname{div} d, \quad r = a \bmod d.$$

Modular Arithmetic

Definition 3.22

If a and b are integers and m is a positive integer,
then a is *congruent to b modulo m* if m divides $a - b$.

We use the notation $a \equiv b \pmod{m}$ to indicate that a is congruent to b modulo m.
If a and b are not congruent modulo m, we write $a \not\equiv b \pmod{m}$.

Theorem 3.23

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

Theorem 3.24

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 $a = b + km$.

Modular Arithmetic

Definition 3.22

If a and b are integers and m is a positive integer,
then a is *congruent to b modulo m* if m divides $a - b$.

We use the notation $a \equiv b \pmod{m}$ to indicate that a is congruent to b modulo m.
If a and b are not congruent modulo m, we write $a \not\equiv b \pmod{m}$.

Theorem 3.23

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

Theorem 3.24

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 $a = b + km$.

Modular Arithmetic

Definition 3.22

If a and b are integers and m is a positive integer,
then a is *congruent to b modulo m* if m divides $a - b$.

We use the notation $a \equiv b \pmod{m}$ to indicate that a is congruent to b modulo m.
If a and b are not congruent modulo m, we write $a \not\equiv b \pmod{m}$.

Theorem 3.23

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

Theorem 3.24

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 $a = b + km$.

Modular Arithmetic

Theorem 3.25

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

$$a + c \equiv b + d \pmod{m} \quad \text{and} \quad ac \equiv bd \pmod{m}.$$

Proof.

Because $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$,
there are integers s and t with

$$b = a + sm \quad \text{and} \quad d = c + tm.$$

Hence,

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

and

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

Hence,

$$a + c \equiv b + d \pmod{m} \quad \text{and} \quad ac \equiv bd \pmod{m}.$$

Modular Arithmetic

Theorem 3.25

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

$$a + c \equiv b + d \pmod{m} \quad \text{and} \quad ac \equiv bd \pmod{m}.$$

Proof.

Because $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$,
there are integers s and t with

$$b = a + sm \quad \text{and} \quad d = c + tm.$$

Hence,

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

and

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

Hence,

$$a + c \equiv b + d \pmod{m} \quad \text{and} \quad ac \equiv bd \pmod{m}.$$

Corollary 3.26

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

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

and

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

Proof.

By the definitions, we know that

$$a \equiv (a \bmod m) \pmod m \quad \text{and} \quad b \equiv (b \bmod m) \pmod m.$$

Hence, Theorem 3.25 tells us that

$$a+b \equiv (a \bmod m)+(b \bmod m) \pmod m$$

and

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

The equalities in this corollary follow from these last two congruences by Theorem 3.23. □

Corollary 3.26

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

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

and

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

Proof.

By the definitions, we know that

$$a \equiv (a \bmod m) \pmod m \quad \text{and} \quad b \equiv (b \bmod m) \pmod m.$$

Hence, Theorem 3.25 tells us that

$$a+b \equiv (a \bmod m)+(b \bmod m) \pmod m$$

and

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

The equalities in this corollary follow from these last two congruences by Theorem 3.23. □

Applications of Congruences

Hasing Functions

Hashing function $h : K \longrightarrow A$,
where K is *key* word set, A is address set.
It assigns memory location $h(k)$ to the record that has k as its key.

Since $|K| \gg |A|$, h should be uniform and *collisions* should be resolved.

Let $|A| = m$, one of the most common hashing function is

$$h(k) = k \bmod m.$$

Applications of Congruences

Pseudorandom Number

Randomly chosen numbers are crucial for random algorithms and often needed for computer simulations.
The most commonly used procedure for generating pseudorandom numbers is the linear congruential method.

We choose four integers:
the modulus m,
multiplier a,
increment c,
and seed x_0,
with $2 \leq a < m, 0 \leq c < m$, and $0 \leq x_0 < m$.

$$x_{n+1} = (ax_n + c) \bmod m.$$

Applications of Congruences

Cryptology

- Encryption: The process making a message secret.
- Decryption: The process determining the original message from the encrypted message.

Caesar cipher

- Encryption: $f(p) = (p+3) \bmod 26$.
- Decryption: $f^{-1}(p) = (p-3) \bmod 26$.

Example 3.27

What is the secret message produced from the message "MEET YOU IN THE PARK" using Caesar cipher?

Letters to numbers	12 4 4 19	24 14 20	8 13	19 7 4	15 0 17 10
Encryption	15 7 7 22	1 17 23	11 16	22 10 7	18 3 20 13
Encrypted message	"PHHW BRX LQ WKH SDUN"				

Outline

Primes

Definition 3.28 (prime and composite)

A positive integer p greater than 1 is called *prime* if the only positive factors of p are 1 and p.
A positive integer that is greater than 1 and is not prime is called *composite*.

Theorem 3.29 (The Fundamental Theorem of Arithmetic)

Every positive integer greater than 1 can be written uniquely as a prime or as the the product of two or more primes where the prime factors are written in order of nondecreasing size.

Theorem 3.30

If n is a composite integer, then n has a prime divisor less than or equal to $\sqrt{n}$.

Primes

Definition 3.28 (prime and composite)

A positive integer p greater than 1 is called *prime* if the only positive factors of p are 1 and p.
A positive integer that is greater than 1 and is not prime is called *composite*.

Theorem 3.29 (The Fundamental Theorem of Arithmetic)

Every positive integer greater than 1 can be written uniquely as a prime or as the the product of two or more primes where the prime factors are written in order of nondecreasing size.

Theorem 3.30

If n is a composite integer, then n has a prime divisor less than or equal to $\sqrt{n}$.

Primes

Definition 3.28 (prime and composite)

A positive integer p greater than 1 is called *prime* if the only positive factors of p are 1 and p.
A positive integer that is greater than 1 and is not prime is called *composite*.

Theorem 3.29 (The Fundamental Theorem of Arithmetic)

Every positive integer greater than 1 can be written uniquely as a prime or as the the product of two or more primes where the prime factors are written in order of nondecreasing size.

Theorem 3.30

If n is a composite integer, then n has a prime divisor less than or equal to $\sqrt{n}$.

Primes

Theorem 3.31

There are infinitely many primes.

Proof.

Proof by contradiction.

Assume that there are only finitely many primes, $p_1, p_2, \ldots, p_n$.

Let

$$Q = p_1 p_2 \cdots p_n + 1.$$

Note that $p_i \nmid Q, i = 1, 2, \ldots, n$,

there is a prime not in the list $p_1, p_2, \ldots, p_n$,

which is either Q or a prime factor of Q.

Contradiction.

Hence, there are infinitely many prime. $\square$

Primes

Theorem 3.31

There are infinitely many primes.

Proof.

Proof by contradiction.
Assume that there are only finitely many primes, $p_1, p_2, \ldots, p_n$.
Let

$$Q = p_1 p_2 \cdots p_n + 1.$$

Note that $p_i \nmid Q, i = 1, 2, \ldots, n$,
there is a prime not in the list $p_1, p_2, \ldots, p_n$,
which is either Q or a prime factor of Q.
Contradiction.
Hence, there are infinitely many prime. $\square$

Primes

Mersenne Primes: After Marin Mersenne (1588–1648)
The prime of the form

$$M_p = 2^p - 1$$

where p is a prime.

Lucas-Lehmer test:
After E. Lucas (1842–1891) and D. Lehmer (1905–1991).

Algorithm 3.7: Lucas-Lehmer Test.

procedure $Lucas - Lehmer(p : \text{prime integer}, r : \text{test result})$
$s := 4$
$M := 2^p - 1$
for $i := 1$ **to** $p - 2$ **do**
  $s := (s^2 - 2) \bmod M$
end
if $s = 0$ **then** $r :=$ Prime **else** $r :=$ Composite

Time complexity: $O(p^2 \log p \log\log p)$.

Primes

Mersenne Primes: After Marin Mersenne (1588–1648)
The prime of the form

$$M_p = 2^p - 1$$

where p is a prime.
Lucas-Lehmer test:
After E. Lucas (1842–1891) and D. Lehmer (1905–1991).

Algorithm 3.7: Lucas-Lehmer Test.

procedure $Lucas - Lehmer(p : \text{prime integer}, r : \text{test result})$
$s := 4$
$M := 2^p - 1$
for $i := 1$ **to** $p - 2$ **do**
  $s := (s^2 - 2) \bmod M$
end
if $s = 0$ **then** $r :=$ Prime **else** $r :=$ Composite

Time complexity: $O(p^2 \log p \log\log p)$.

Primes

Theorem 3.32

The ratio of the number of primes not exceeding x and $x/\ln x$ approaches 1 as x grows without bound.

Using the Prime Number Theorem and calculus,
it can be shown that the probability that an integer n is prime is also approximately $1/\ln n$.

For example, the odds that an integer near 10^{1000} is prime are approximately $1/\ln 10^{1000}$,
which is approximately $1/2300$.

Primes

Theorem 3.32

The ratio of the number of primes not exceeding x and $x/\ln x$ approaches 1 as x grows without bound.

Using the Prime Number Theorem and calculus,
it can be shown that the probability that an integer n is prime is also approximately $1/\ln n$.

For example, the odds that an integer near 10^{1000} is prime are approximately $1/\ln 10^{1000}$,
which is approximately $1/2300$.

Conjectures and Open Problems About Primes

Example 3.33

Find a function $f(n)$ such that $f(n)$ is prime for all positive integers n.

$$f(n) = n^2 - n + 41.$$

Correct for $1 \leq n \leq 40$.

For every polynomial $f(n)$ with integer coefficients,
there is a positive integer y such that $f(y)$ is composite.

Example 3.34

There are infinitely many primes of the form $n^2 + 1$, where n is a positive integer.

Example 3.35 (Goldbach's Conjecture)

Every even integer $n, n > 2$, is the sum of two primes (1742).

Example 3.36 (The Twin Prime Conjecture)

Twin primes are primes that differ by 2.
There are infinitely many twin primes.

Conjectures and Open Problems About Primes

Example 3.33

Find a function $f(n)$ such that $f(n)$ is prime for all positive integers n.

$$f(n) = n^2 - n + 41.$$

Correct for $1 \leq n \leq 40$.

For every polynomial $f(n)$ with integer coefficients,
there is a positive integer y such that $f(y)$ is composite.

Example 3.34

There are infinitely many primes of the form $n^2 + 1$, where n is a positive integer.

Example 3.35 (Goldbach's Conjecture)

Every even integer $n, n > 2$, is the sum of two primes (1742).

Example 3.36 (The Twin Prime Conjecture)

Twin primes are primes that differ by 2.
There are infinitely many twin primes.

Conjectures and Open Problems About Primes

Example 3.33

Find a function $f(n)$ such that $f(n)$ is prime for all positive integers n.

$$f(n) = n^2 - n + 41.$$

Correct for $1 \leq n \leq 40$.

For every polynomial $f(n)$ with integer coefficients,
there is a positive integer y such that $f(y)$ is composite.

Example 3.34

There are infinitely many primes of the form $n^2 + 1$, where n is a positive integer.

Example 3.35 (Goldbach's Conjecture)

Every even integer $n, n > 2$, is the sum of two primes (1742).

Example 3.36 (The Twin Prime Conjecture)

Twin primes are primes that differ by 2.
There are infinitely many twin primes.

Conjectures and Open Problems About Primes

Example 3.33

Find a function $f(n)$ such that $f(n)$ is prime for all positive integers n.

$$f(n) = n^2 - n + 41.$$

Correct for $1 \leq n \leq 40$.

For every polynomial $f(n)$ with integer coefficients,
there is a positive integer y such that $f(y)$ is composite.

Example 3.34

There are infinitely many primes of the form $n^2 + 1$, where n is a positive integer.

Example 3.35 (Goldbach's Conjecture)

Every even integer $n, n > 2$, is the sum of two primes (1742).

Example 3.36 (The Twin Prime Conjecture)

Twin primes are primes that differ by 2.
There are infinitely many twin primes.

Greatest Common Divisors and Least Common Multiples

Definition 3.37 (gcd)

Let a and b be integers, not both zero. The largest integer d such that $d \mid a$ and $d \mid b$ is called the *greatest common divisor* of a and b, denoted by $\gcd(a, b)$.

Definition 3.38 (lcm)

The *least common multiple* of the positive integers a and b is the smallest positive integer that is divisible by both a and b, denoted by $\operatorname{lcm}(a, b)$.

Greatest Common Divisors and Least Common Multiples

Definition 3.37 (gcd)

Let a and b be integers, not both zero. The largest integer d such that $d \mid a$ and $d \mid b$ is called the *greatest common divisor* of a and b, denoted by $\gcd(a, b)$.

Definition 3.38 (lcm)

The *least common multiple* of the positive integers a and b is the smallest positive integer that is divisible by both a and b, denoted by $\operatorname{lcm}(a, b)$.

Greatest Common Divisors and Least Common Multiples

Remarks

Suppose that the prime factorizations of the integers a and b, neither equal to zero, are

$$a = p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n}, \quad b = p_1^{b_1} p_2^{b_2} \cdots p_n^{b_n},$$

where each exponent is a nonnegative integer, and where all primes occurring in the prime factorization of either a or b are included in both factorization, with zero exponents if necessary.
Then

$$\begin{aligned} \gcd(a,b) &= p_1^{\min\{a_1,b_1\}} p_2^{\min\{a_2,b_2\}} \cdots p_n^{\min\{a_n,b_n\}}, \\ \operatorname{lcm}(a,b) &= p_1^{\max\{a_1,b_1\}} p_2^{\max\{a_2,b_2\}} \cdots p_n^{\max\{a_n,b_n\}}, \\ ab &= \gcd(a,b) \cdot \operatorname{lcm}(a,b). \end{aligned}$$

Relatively Prime

Definition 3.39

The integers a and b are *relatively prime* if $\gcd(a, b) = 1$.

Definition 3.40

The integers $a_1, a_2, \ldots, a_n$ are *pairwise relatively prime* if $\gcd(a_i, a_j) = 1$, whenever $1 \leq i < j \leq n$.

Relatively Prime

Definition 3.39

The integers a and b are *relatively prime* if $\gcd(a, b) = 1$.

Definition 3.40

The integers $a_1, a_2, \ldots, a_n$ are *pairwise relatively prime* if $\gcd(a_i, a_j) = 1$, whenever $1 \leq i < j \leq n$.

Outline

Representations of Integers

Theorem 3.41

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

$$n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0, \tag{3.1}$$

where k is a nonnegative integer, $a_0, a_1, \ldots, a_k$ are nonnegative integers less than b, and $a_k \neq 0$.

The representation of n in formula 3.1 is called the base b expansion of n, denoted by $(a_k a_{k-1} \ldots a_1 a_0)_b$.

Remark

The optimal base b expansion of n can be derived by

$$b^* = \underset{b}{\operatorname{argmin}}\, b \log_b n,$$

hence, $b^* = \mathrm{e}$.

Representations of Integers

Base Conversion:

From formula 3.1 we can see that

$$a_i = q_{i-1} \bmod b, q_i = q_{i-1} \operatorname{div} b, \quad i = 0, \ldots, k-1, q_{-1} = n.$$

The Algorithm 3.8 gives the base b expansion $(a_{k-1} \cdots a_1 a_0)_b$ of the integer n.

Algorithm 3.8: Constructing Base b Expansions.

procedure *baseexpansion*(n : positive integer)
$q := n, \quad k := 0$
while $q \neq 0$ **do**
  $a_k := q \bmod b$
  $q := \lfloor q/b \rfloor$
  $k := k + 1$
end
{the base b expansion of n is $(a_{k-1} \cdots a_1 a_0)_b$}

Algorithms for Integer Operations

Addition: $a = (a_{n-1}a_{n-2}\cdots a_1a_0)_2$ and $b = (b_{n-1}b_{n-2}\cdots b_1b_0)_2$,
$a+b = (s_ns_{n-1}\cdots s_1s_0)_2$,
$a_i + b_i + c_i = c_{i+1}\cdot 2 + s_i, c_0 = 0$.
where c_i is the carry, $i = 0, 1, \ldots, n$.

Algorithm 3.9: Addition of Integers

procedure $add((a_{n-1}a_{n-2}\cdots a_1a_0)_2, (b_{n-1}b_{n-2}\cdots b_1b_0)_2)$
$c := 0$
for $i := 0$ **to** $n-1$ **do**
  $d := \lfloor (a_i + b_i + c)/2 \rfloor$
  $s_i := a_i + b_i + c - 2d$
  $c := d$
end
$s_n := c$
{the binary expansion of the sum is $(s_ns_{n-1}\cdots s_1s_0)_2$}

Algorithms for Integer Operations

Multiplication:

$$\begin{aligned} ab &= a(b_0 2^0 + b_1 2^1 + \cdots + b_{n-1} 2^{n-1}) \\ &= a(b_0 2^0) + a(b_1 2^1) + \cdots + a(b_{n-1} 2^{n-1}). \end{aligned}$$

We can obtain $(ab_j)2^j$ by shifting the binary expansion of ab_j j places to the left, and obtain ab by adding the n integers $ab_j 2^j$, $j = 0, 1, \ldots, n-1$.

Algorithm 3.10: Multiplying Integers.

procedure *multiply*$(a_{n-1}a_{n-2}\cdots a_1 a_0)_2, (b_{n-1}b_{n-2}\cdots b_1 b_0)_2)$
for $i := 0$ **to** $n-1$ **do**
  if $b_i = 1$ **then** $c_i := a$ shifted i places **else** $c_i := 0$
end
$\{c_0, c_1, \ldots, c_{n-1}$ are the partial products$\}$
$p := 0$
for $i := 0$ **to** n_1 **do** $p := p + c$

Algorithms for Integer Operations

Div and Mod:

Algorithm 3.11: Computing div and mod.

procedure *division*(a : integer, d : positive integer)
$q := 0, r := |a|$
while $r \geq d$ **do**
    $r := r - d$
    $q := q + 1$
end
if $a < 0$ **then**
    if $r > 0$ **then**
        $r := d - r$
        $q := -(q+1)$
    else
        $q := -q$
    end
end
$\{q = a \div d, r = a \bmod d$. Revised according to Chang Feng.$\}$

Algorithms for Integer Operations

Modular Exponentiation:

Let $n = (a_{k-1} \dots a_1 a_0)_2$,
then $b^n = b^{a_{k-1} \cdot 2^{k-1} + \dots + a_1 \cdot 2 + a_0} = b^{a_{k-1} \cdot 2^{k-1}} \cdot \dots \cdot b^{a_1 \cdot 2} \cdot b^{a_0}$.
Multiplying together b^{2^i} mod m where $a_i \neq 0$, obtain b^n mod m.

Algorithm 3.12: Modular Exponentiation.

procedure *modularexponentiation*$(b, (a_{k-1} \dots a_1 a_0)_2, m)$
$x := 1$
$power := b \bmod m$
for $i := 0$ **to** $k-1$ **do**
  if $a_i = 1$ **then** $x := (x \cdot power) \bmod m$
  $power := (power \cdot power) \bmod m$
end
$\{x = b^n \bmod m\}$

Algorithms for Integer Operations

The Euclidean Algorithm

Lemma 3.42

Let $a = bq + r$, where a, b, q and r are integers. Then $\gcd(a, b) = \gcd(b, r)$.

Algorithms for Integer Operations

The Euclidean Algorithm

Lemma 3.42

Let $a = bq + r$, where a, b, q and r are integers. Then $\gcd(a, b) = \gcd(b, r)$.

Proof.

Suppose that d divides both a and b.
Then it follows that d also divides $a - bq = r$.
Hence, any common divisor of a and b is also a common divisor of b and r.
Likewise, suppose that d divides both b and r.
Then d also divides $bq + r = a$.
Hence, any common divisor of b and r is also a common divisor of a and b.
We have shown that the common divisors of a and b are the same as the common divisors of b and r.
Consequently, $\gcd(a, b) = \gcd(b, r)$. □

Algorithms for Integer Operations

The Euclidean Algorithm

Lemma 3.42

Let $a = bq + r$, where a, b, q and r are integers. Then $\gcd(a, b) = \gcd(b, r)$.

Suppose that a and b are positive integers with $a \geq b$.
Let $r_0 = a$ and $r_1 = b$.
When we successively apply the division algorithm, we obtain

$$\begin{aligned} r_0 &= r_1 q_1 + r_2 && 0 \leq r_2 < r_1, \\ r_1 &= r_2 q_2 + r_3 && 0 \leq r_3 < r_2, \\ &\vdots && \vdots \\ r_{n-2} &= r_{n-1} q_{n-1} + r_n && 0 \leq r_n < r_{n-1}, \\ r_{n-1} &= r_n q_n. \end{aligned}$$

Eventually a remainder of zero occurs in this sequences of successive divisions, and it follows that

$$\begin{aligned} \gcd(a, b) &= \gcd(r_0, r_1) = \gcd(r_1, r_2) = \cdots = \gcd(r_{n-1}, r_n) \\ &= \gcd(r_n, 0) = r_n. \end{aligned}$$

Algorithms for Integer Operations

The Euclidean Algorithm

Algorithm 3.13: The Euclidean Algorithm.

procedure $gcd(a, b$: positive integers)
$x := a$
$y := b$
while $y \neq 0$ **do**
    $r := x \bmod y$
    $x := y$
    $y := r$
end
$\{x = \gcd(a, b)\}$

Outline

Some Useful Results

Theorem 3.43

If a and b are positive integers, then there exist integers s and t such that $\gcd(a,b) = sa + tb$.

Example 3.44

Express $\gcd(252, 198) = 18$ as a linear combinations of 252 and 198.

Solution 3.44

To show that $\gcd(252, 198) = 18$, *the Euclidean algorithm uses these divisions:*

$$252 = 1 \cdot 198 + 54 \qquad 198 = 3 \cdot 54 + 36$$

$$54 = 1 \cdot 36 + 18 \qquad 36 = 2 \cdot 18$$

Some Useful Results

Theorem 3.43

If a and b are positive integers, then there exist integers s and t such that $\gcd(a,b) = sa + tb$.

Example 3.44

Express $\gcd(252, 198) = 18$ as a linear combinations of 252 and 198.

Solution 3.44

To show that $\gcd(252, 198) = 18$, *the Euclidean algorithm uses these divisions:*

$$252 = 1 \cdot 198 + 54 \qquad 198 = 3 \cdot 54 + 36$$

$$54 = 1 \cdot 36 + 18 \qquad 36 = 2 \cdot 18$$

Some Useful Results

Theorem 3.43

If a and b are positive integers, then there exist integers s and t such that $\gcd(a,b) = sa + tb$.

Example 3.44

Express $\gcd(252, 198) = 18$ as a linear combinations of 252 and 198.

Solution 3.44

To show that $\gcd(252, 198) = 18$, *the Euclidean algorithm uses these divisions:*

$$252 = 1 \cdot 198 + 54 \qquad 198 = 3 \cdot 54 + 36$$

$$54 = 1 \cdot 36 + 18 \qquad 36 = 2 \cdot 18$$

Some Useful Results

Theorem 3.43

If a and b are positive integers, then there exist integers s and t such that $\gcd(a,b) = sa + tb$.

Example 3.44

Express $\gcd(252, 198) = 18$ as a linear combinations of 252 and 198.

Solution 3.44

To show that $\gcd(252, 198) = 18$, *the Euclidean algorithm uses these divisions:*

$$252 = 1 \cdot 198 + 54 \qquad 198 = 3 \cdot 54 + 36$$

$$54 = 1 \cdot 36 + 18 \qquad 36 = 2 \cdot 18$$

We find that

$$18 = 54 - 1 \cdot 36,$$

$$36 = 198 - 3 \cdot 54,$$

we have

$$18 = 54 - 1 \cdot 36 = 54 - 1 \cdot (198 - 3 \cdot 54) = 4 \cdot 54 - 1 \cdot 198.$$

Some Useful Results

Theorem 3.43

If a and b are positive integers, then there exist integers s and t such that $\gcd(a,b) = sa + tb$.

Example 3.44

Express $\gcd(252, 198) = 18$ as a linear combinations of 252 and 198.

Solution 3.44

To show that $\gcd(252, 198) = 18$, *the Euclidean algorithm uses these divisions:*

$$252 = 1 \cdot 198 + 54 \qquad 198 = 3 \cdot 54 + 36$$

$$54 = 1 \cdot 36 + 18 \qquad 36 = 2 \cdot 18$$

The first division tells us that

$$54 = 252 - 1 \cdot 198,$$

we conclude that

$$18 = 4 \cdot (252 - 1 \cdot 198) - 1 \cdot 198 = 4 \cdot 252 - 5 \cdot 198,$$

completing the solution.

Some Useful Results

Lemma 3.45

If a, b, and c are positive integers such that $\gcd(a, b) = 1$ and $a \mid bc$, then $a \mid c$.

Proof.

By Theorem 3.43 there are integers s and t such that

$$sa + tb = 1.$$

Multiplying both sides by c, we obtain

$$sac + tbc = c.$$

We conclude that a divides $sac + tbc$, hence $a \mid c$. □

Some Useful Results

Lemma 3.45

If a, b, and c are positive integers such that $\gcd(a, b) = 1$ and $a \mid bc$, then $a \mid c$.

Proof.

By Theorem 3.43 there are integers s and t such that

$$sa + tb = 1.$$

Multiplying both sides by c, we obtain

$$sac + tbc = c.$$

We conclude that a divides $sac + tbc$, hence $a \mid c$. □

Some Useful Results

Lemma 3.46

If p is a prime and $p \mid a_1 a_2 \cdots a_n$, where each a_i is an integer, then $p \mid a_i$ for some i.

Theorem 3.47

Let m be a positive integer and let a, b, and c be integers. If $ac \equiv bc \pmod{m}$ and $\gcd(c, m) = 1$, then $a \equiv b \pmod{m}$.

Some Useful Results

Lemma 3.46

If p is a prime and $p \mid a_1 a_2 \cdots a_n$, where each a_i is an integer, then $p \mid a_i$ for some i.

Theorem 3.47

Let m be a positive integer and let a, b, and c be integers.
If $ac \equiv bc \pmod{m}$ and $\gcd(c, m) = 1$,
then $a \equiv b \pmod{m}$.

Some Useful Results

Lemma 3.46

If p is a prime and $p \mid a_1 a_2 \cdots a_n$, where each a_i is an integer, then $p \mid a_i$ for some i.

Theorem 3.47

Let m be a positive integer and let a, b, and c be integers.
If $ac \equiv bc \pmod{m}$ and $\gcd(c, m) = 1$,
then $a \equiv b \pmod{m}$.

Proof.

Because $ac \equiv bc \pmod{m}, m \mid ac - bc = c(a - b)$.
By Lemma 3.45, because $\gcd(c, m) = 1$, it follows that $m \mid (a - b)$.
We conclude that $a \equiv b \pmod{m}$. □

Some Useful Results

Lemma 3.46

If p is a prime and $p \mid a_1 a_2 \cdots a_n$, where each a_i is an integer, then $p \mid a_i$ for some i.

Theorem 3.47

Let m be a positive integer and let a, b, and c be integers.
If $ac \equiv bc \pmod{m}$ and $\gcd(c, m) = 1$,
then $a \equiv b \pmod{m}$.

Linear Congruences

Definition 3.48

Let a and $\overline{a}$ be integers.
If $\overline{a}a \equiv 1 \pmod{m}$, then $\overline{a}$ is said to be an inverse of a modulo m.

Theorem 3.49

If a and m are relatively prime integers and $m > 1$,
then an inverse of a modulo m exists.
Furthermore, this inverse is unique modulo m.

That is, there is a unique positive integer $\overline{a}$ less than m that is an inverse of a modulo m and every other inverse of a modula m is congruent to $\overline{a}$ modulo m.

Linear Congruences

Definition 3.48

Let a and $\overline{a}$ be integers.
If $\overline{a}a \equiv 1 \pmod{m}$, then $\overline{a}$ is said to be an inverse of a modulo m.

Theorem 3.49

If a and m are relatively prime integers and $m > 1$,
then an inverse of a modulo m exists.
Furthermore, this inverse is unique modulo m.

That is, there is a unique positive integer $\overline{a}$ less than m that is an inverse of a modulo m and every other inverse of a modula m is congruent to $\overline{a}$ modulo m.

Linear Congruences

Proof.

By Theorem 3.43, because $\gcd(a, m) = 1$, there are integers s and t such that

$$sa + tm = 1.$$

This implies that

$$sa + tm \equiv 1 \pmod{m}.$$

Because $tm \equiv 0 \pmod{m}$, it follows that

$$sa \equiv 1 \pmod{m}.$$

Consequently, s is an inverse of a modulo m. □

Linear Congruences

Proof.

Suppose that b and c are both inverses of a modulo m.
Then $ba \equiv 1 \pmod{m}$ and $ca \equiv 1 \pmod{m}$.
Hence, $ba \equiv ca \pmod{m}$.
Because $\gcd(a, m) = 1$ it follows by Theorem 3.47 that

$$b \equiv c \pmod{m}.$$

Linear Congruences

The proof of Theorem 3.49 describes a method for finding the inverse of a modulo m when a and m are relatively prime: find a linear combination of a and m that equals 1, which can be done by working backward through the steps of the Euclidean algorithm, the coefficient of a in this linear combinations is an inverse of a modulo m.

When we have an inverse $\overline{a}$ of a modulo m, we can easily solve the congruence

$$ax \equiv b \pmod{m}$$

by multiplying both sides of the linear congruence by $\overline{a}$, and obtain

$$x \equiv \overline{a}b \pmod{m}.$$

Linear Congruences

Example 3.50

What are the solutions of the linear congruences $3x \equiv 4 \pmod 7$?

Solution 3.50

From

$$7 = 2 \cdot 3 + 1$$

we have

$$-2 \cdot 3 + 1 \cdot 7 = 1.$$

This shows that -2 is an inverse of 3 modulo 7.

Multiplying both sides of the congruence by -2, we have

$$x \equiv -2 \cdot 4 \pmod 7 \equiv 6 \pmod 7.$$

Linear Congruences

Example 3.50

What are the solutions of the linear congruences $3x \equiv 4 \pmod 7$?

Solution 3.50

From

$$7 = 2 \cdot 3 + 1$$

we have

$$-2 \cdot 3 + 1 \cdot 7 = 1.$$

This shows that -2 is an inverse of 3 modulo 7.

Multiplying both sides of the congruence by -2, we have

$$x \equiv -2 \cdot 4 \pmod 7 \equiv 6 \pmod 7.$$

The Chinese Remainder Theorem

Theorem 3.51

Let $m_1, m_2, \ldots, m_n$ be pairwise relatively prime positive integers and $a_1, a_2, \ldots, a_n$ arbitrary integers. Then the system

$$\begin{aligned} x &\equiv a_1 \pmod{m_1}, \\ x &\equiv a_2 \pmod{m_2}, \\ &\vdots \\ x &\equiv a_n \pmod{m_n} \end{aligned}$$

has a unique solution modulo $m = m_1 m_2 \cdots m_n$.

The Chinese Remainder Theorem

Proof.

We need to show that a solution exists and that it is unique modulo m.

□

The Chinese Remainder Theorem

Proof.

We need to show that a solution exists and that it is unique modulo m.

Let

$$M_k = m/m_k, \quad k = 1, 2, \ldots, n.$$

Because $\gcd(m_i, m_k) = 1$ whenever $i \neq k$,
it follows that $\gcd(m_k, M_k) = 1$.

Consequently, by Theorem 3.49, there is an integer y_k, such that

$$M_k y_k \equiv 1 \pmod{m_k}.$$

Form the sum

$$x = \sum_{i=1}^{n} a_i M_i y_i.$$

The Chinese Remainder Theorem

Proof.

We need to show that a solution exists and that it is unique modulo m.

Note that

$$\begin{aligned} M_j &\equiv 0 \pmod{m_k}, \quad j \neq k; \\ M_j y_j &\equiv 1 \pmod{m_k}, \quad j = k. \end{aligned}$$

We see that

$$x \equiv a_k M_k y_k \equiv a_k \pmod{m_k}, \quad k = 1, 2, \ldots, n.$$

We have shown that x is a simultaneous solution to the n congruences. □

The Chinese Remainder Theorem

Proof.

We need to show that a solution exists and that it is unique modulo m.

Suppose that a and b are both simultaneous solutions to the n congruences, and that p is a prime factor of $m = m_1 m_2 \dots m_n$.
Because $\gcd(m_i, m_k) = 1, i, k = 1, 2, \dots, n, i \neq k$,
p is a factor of exactly one of the m_is, say m_j.

Because $m_j \mid a - b$, it follows that $a - b$ has the factor p in its prime factorization to a power as large as the power to which it appears in the prime factorization of m_j.

It follows that $m_1 m_2 \dots m_n \mid a - b$, so

$$a \equiv b \pmod{m_1 m_2 \dots m_n}.$$

□

The Chinese Remainder Theorem

Example 3.52

Sun-Tsu Problem:

There are certain things whose number is unknown.
When divided by 3, the remainder is 2;
when divided by 5, the remainder is 3;
when divided by 7, the remainder is 2.
What will be the number of things?

The Chinese Remainder Theorem

Solution 3.52

The puzzle can be translated into the following systems of congruence:

$$x \equiv 2 \pmod 3,$$
$$x \equiv 3 \pmod 5,$$
$$x \equiv 2 \pmod 7.$$

Let

$$m = 3 \cdot 5 \cdot 7;$$
$$M_1 = m/3 = 35, \qquad y_1 = 2;$$
$$M_2 = m/5 = 21, \qquad y_2 = 1;$$
$$M_3 = m/7 = 15, \qquad y_3 = 1.$$

Hence,

$$\begin{aligned} x &\equiv a_1 M_1 y_1 + a_2 M_2 y_2 + a_3 M_3 y_3 \\ &= 2 \cdot 35 \cdot 2 + 3 \cdot 21 \cdot 1 + 2 \cdot 15 \cdot 1 \\ &\equiv 23 \pmod{105}. \end{aligned}$$

Fermat's Little Theorem

Theorem 3.53

If p is prime and a is an integer not divisible by p, then

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

Furthermore, for every integer a we have

$$a^p \equiv a \pmod{p}.$$

Proof.

In the first $p-1$ positive multiples of a:

$$a, 2a, \ldots, (p-1)a,$$

suppose that $ra \equiv sa \pmod{p}$, then we have $r \equiv s \pmod{p}$.
So the $p-1$ multiples of a above are distinct and nonzero;
that is, they must be congruent to 1, 2, ..., $p-1$ in some order.
Multiply all these together, we have

$$a \cdot 2a \cdot \cdots \cdot (p-1) \cdot a \equiv 1 \cdot 2 \cdot \cdots \cdot (p-1) \pmod{p},$$

that is,

$$a^{p-1}(p-1)! \equiv (p-1)! \pmod{p}.$$

Hence,

$$a^{p-1} \equiv 1 \pmod{p} \text{ and } a^p \equiv a \pmod{p}.$$

□

Fermat's Little Theorem

Theorem 3.53

If p is prime and a is an integer not divisible by p, then

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

Furthermore, for every integer a we have

$$a^p \equiv a \pmod{p}.$$

Proof.

In the first $p-1$ positive multiples of a:

$$a, 2a, \ldots, (p-1)a,$$

suppose that $ra \equiv sa \pmod{p}$, then we have $r \equiv s \pmod{p}$.
So the $p-1$ multiples of a above are distinct and nonzero;
that is, they must be congruent to 1, 2, ..., $p-1$ in some order.
Multiply all these together, we have

$$a \cdot 2a \cdot \cdots \cdot (p-1) \cdot a \equiv 1 \cdot 2 \cdot \cdots \cdot (p-1) \pmod{p},$$

that is,

$$a^{p-1}(p-1)! \equiv (p-1)! \pmod{p}.$$

Hence,

$$a^{p-1} \equiv 1 \pmod{p} \text{ and } a^p \equiv a \pmod{p}.$$

□

Outline

Definitions

Definition 3.54

A *matrix* is a rectangular array of numbers.
A matrix with m rows and n columns is called an $m \times n$ matrix.

A matrix with the same number of rows and columns is called *square*.

Two matrices are *equal* if they have the same number of rows and the same number of columns and the corresponding entries in every position are equal.

Definitions

Definition 3.55

Let

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}.$$

The ith row of $\mathbf{A}$ is the $1 \times n$ matrix

$$[a_{i1}, a_{i2}, \ldots, a_{in}].$$

The jth column of $\mathbf{A}$ is the $m \times 1$ matrix

$$\begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{bmatrix}.$$

The (i,j)th *element* or *entry* of $\mathbf{A}$ is the element a_{ij}.

We express $\mathbf{A}$ as $\mathbf{A} = [a_{ij}]$.

Matrix Arithmetic

Definition 3.56

Let $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$ be $m \times n$ matrices.
The *sum* of $\mathbf{A}$ and $\mathbf{B}$ is

$$\mathbf{A} + \mathbf{B} = [a_{ij} + b_{ij}].$$

Definition 3.57

Let $\mathbf{A}$ be an $m \times k$ matrix and $\mathbf{B}$ be a $k \times n$ matrix.
The *product* of $\mathbf{A}$ and $\mathbf{B}$, denoted by $\mathbf{AB}$,
is the $m \times n$ matrix with its (i,j)th entry equal to
the sum of the products of the corresponding elements
from the ith row of $\mathbf{A}$ and jth column of $\mathbf{B}$.
In other words, if $\mathbf{AB} = [c_{ij}]$, then

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{ik}b_{kj}.$$

Matrix Arithmetic

Definition 3.56

Let $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$ be $m \times n$ matrices.
The *sum* of $\mathbf{A}$ and $\mathbf{B}$ is

$$\mathbf{A} + \mathbf{B} = [a_{ij} + b_{ij}].$$

Definition 3.57

Let $\mathbf{A}$ be an $m \times k$ matrix and $\mathbf{B}$ be a $k \times n$ matrix.
The *product* of $\mathbf{A}$ and $\mathbf{B}$, denoted by $\mathbf{AB}$,
is the $m \times n$ matrix with its (i,j)th entry equal to
the sum of the products of the corresponding elements
from the ith row of $\mathbf{A}$ and jth column of $\mathbf{B}$.
In other words, if $\mathbf{AB} = [c_{ij}]$, then

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{ik}b_{kj}.$$

Algorithms for Matrix Multiplication

Algorithm 3.14: Matrix Multiplication.

procedure *matrixmultiplication*($\mathbf{A}, \mathbf{B}$: matrices)
for $i := 1$ **to** m **do**
  for $j := 1$ **to** n **do**
    $c_{ij} := 0$
    for $q := 1$ **to** k **do**
      $c_{ij} := c_{ij} + a_{iq} b_{qj}$
    end
  end
end
$\{\mathbf{C} = [c_{ij}]$ is the product of $\mathbf{A}_{m \times k}$ and $\mathbf{B}_{k \times n}\}$

Matrix-Chain Multiplication

How should the matrix-chain $\mathbf{A}_1\mathbf{A}_2\cdots\mathbf{A}_n$ be computed using the fewest multiplications of integers, where $\mathbf{A}_1$, $\mathbf{A}_2$, $\ldots$, $\mathbf{A}_n$ are $m_1 \times m_2, m_2 \times m_3, \ldots, m_n \times m_{n+1}$ matrices, respectively, and each has integers as entries?

Example 3.58

In which order should the matrices $\mathbf{A}_1, \mathbf{A}_2$, and $\mathbf{A}_3$—where $\mathbf{A}_1$ is 30×20, $\mathbf{A}_2$ is 20×40, and $\mathbf{A}_3$ is 40×10, all with integer entries—be multiplied to use the least number of multiplications of integers?

Solution 3.58

$$\underbrace{(\overbrace{\mathbf{A}_1\mathbf{A}_2}^{30\times20\times40})\mathbf{A}_3}_{30\times40\times10} \quad 36000 : 14000 \quad \underbrace{\mathbf{A}_1(\overbrace{\mathbf{A}_2\mathbf{A}_3}^{20\times40\times10})}_{30\times20\times10}$$

Matrix-Chain Multiplication

How should the matrix-chain $\mathbf{A}_1\mathbf{A}_2\cdots\mathbf{A}_n$ be computed using the fewest multiplications of integers, where $\mathbf{A}_1$, $\mathbf{A}_2$, $\ldots$, $\mathbf{A}_n$ are $m_1 \times m_2, m_2 \times m_3, \ldots, m_n \times m_{n+1}$ matrices, respectively, and each has integers as entries?

Example 3.58

In which order should the matrices $\mathbf{A}_1, \mathbf{A}_2$, and $\mathbf{A}_3$—where $\mathbf{A}_1$ is 30×20, $\mathbf{A}_2$ is 20×40, and $\mathbf{A}_3$ is 40×10, all with integer entries—be multiplied to use the least number of multiplications of integers?

Solution 3.58

$$\underbrace{(\overbrace{\mathbf{A}_1\mathbf{A}_2}^{30\times20\times40})\mathbf{A}_3}_{30\times40\times10} \quad 36000 : 14000 \quad \underbrace{\mathbf{A}_1(\overbrace{\mathbf{A}_2\mathbf{A}_3}^{20\times40\times10})}_{30\times20\times10}$$

Matrix-Chain Multiplication

How should the matrix-chain $\mathbf{A}_1\mathbf{A}_2\cdots\mathbf{A}_n$ be computed using the fewest multiplications of integers, where $\mathbf{A}_1$, $\mathbf{A}_2$, $\ldots$, $\mathbf{A}_n$ are $m_1 \times m_2, m_2 \times m_3, \ldots, m_n \times m_{n+1}$ matrices, respectively, and each has integers as entries?

Example 3.58

In which order should the matrices $\mathbf{A}_1, \mathbf{A}_2$, and $\mathbf{A}_3$—where $\mathbf{A}_1$ is 30×20, $\mathbf{A}_2$ is 20×40, and $\mathbf{A}_3$ is 40×10, all with integer entries—be multiplied to use the least number of multiplications of integers?

Solution 3.58

$$\underbrace{(\overbrace{\mathbf{A}_1\mathbf{A}_2}^{30\times20\times40})\mathbf{A}_3}_{30\times40\times10} \quad 36000 : 14000 \quad \underbrace{\mathbf{A}_1(\overbrace{\mathbf{A}_2\mathbf{A}_3}^{20\times40\times10})}_{30\times20\times10}$$

Transposes and Powers of Matrices

Definition 3.59

The *identity matrix of order n* is the $n \times n$ matrix $\mathbf{I}_n = [\delta_{ij}]$, where $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ if $i \neq j$.

When $\mathbf{A}$ is an $m \times n$ matrix, we have

$$\mathbf{A}\mathbf{I}_n = \mathbf{I}_m\mathbf{A} = \mathbf{A}.$$

When $\mathbf{A}$ is an $n \times n$ matrix, we have

$$\mathbf{A^0} = \mathbf{I_n}, \quad \mathbf{A}^r = \underbrace{\mathbf{A}\mathbf{A}\cdots\mathbf{A}}_{r \text{ times}}$$

Definition 3.60

Let $\mathbf{A} = [a_{ij}]$ be an $m \times n$ matrix.
The *transpose* of $\mathbf{A}$, denoted by $\mathbf{A}^t$, is an $n \times m$ matrix, and $\mathbf{A}^t = [a_{ji}]$.

A sqaure matrix $\mathbf{A}$ is called *symmetric* if $\mathbf{A} = \mathbf{A}^t$.

Transposes and Powers of Matrices

Definition 3.59

The *identity matrix of order n* is the $n \times n$ matrix $\mathbf{I}_n = [\delta_{ij}]$, where $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ if $i \neq j$.

When $\mathbf{A}$ is an $m \times n$ matrix, we have

$$\mathbf{A}\mathbf{I}_n = \mathbf{I}_m\mathbf{A} = \mathbf{A}.$$

When $\mathbf{A}$ is an $n \times n$ matrix, we have

$$\mathbf{A^0} = \mathbf{I_n}, \quad \mathbf{A}^r = \underbrace{\mathbf{A}\mathbf{A}\cdots\mathbf{A}}_{r \text{ times}}$$

Definition 3.60

Let $\mathbf{A} = [a_{ij}]$ be an $m \times n$ matrix.
The *transpose* of $\mathbf{A}$, denoted by $\mathbf{A}^t$, is an $n \times m$ matrix, and $\mathbf{A}^t = [a_{ji}]$.

A sqaure matrix $\mathbf{A}$ is called *symmetric* if $\mathbf{A} = \mathbf{A}^t$.

Zero-One Matrices

Definition 3.61

Let $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$ be $m \times n$ zero-one matrices.
The *join* of $\mathbf{A}$ and $\mathbf{B}$ is $\mathbf{A} \vee \mathbf{B} = [a_{ij} \vee b_{ij}]$.
The *meet* of $\mathbf{A}$ and $\mathbf{B}$ is $\mathbf{A} \wedge \mathbf{B} = [a_{ij} \wedge b_{ij}]$.

Definition 3.62

Let $\mathbf{A} = [a_{ij}]$ be an $m \times k$ zero-one matrix and $\mathbf{B} = [b_{ij}]$ be a $k \times n$ zero-one matrix. Then the *Boolean product* of $\mathbf{A}$ and $\mathbf{B}$ is the $m \times n$ matrix

$$\begin{aligned} \mathbf{A} \odot \mathbf{B} &= [(a_{i1} \wedge b_{1j}) \vee (a_{i2} \wedge b_{2j}) \vee \cdots \vee (a_{ik} \wedge b_{kj})] \\ &= [\vee_{t=1}^{k} (a_{it} \wedge b_{tj})]. \end{aligned}$$

Definition 3.63

Let $\mathbf{A}$ be a square zero-one matrix and let r be a positive integer. The rth *Boolean power of* $\mathbf{A}$ is the matrix

$$\mathbf{A}^{[r]} = \underbrace{\mathbf{A} \odot \mathbf{A} \odot \cdots \odot \mathbf{A}}_{r \text{ times}}.$$

Zero-One Matrices

Definition 3.61

Let $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$ be $m \times n$ zero-one matrices.
The *join* of $\mathbf{A}$ and $\mathbf{B}$ is $\mathbf{A} \vee \mathbf{B} = [a_{ij} \vee b_{ij}]$.
The *meet* of $\mathbf{A}$ and $\mathbf{B}$ is $\mathbf{A} \wedge \mathbf{B} = [a_{ij} \wedge b_{ij}]$.

Definition 3.62

Let $\mathbf{A} = [a_{ij}]$ be an $m \times k$ zero-one matrix and $\mathbf{B} = [b_{ij}]$ be a $k \times n$ zero-one matrix. Then the *Boolean product* of $\mathbf{A}$ and $\mathbf{B}$ is the $m \times n$ matrix

$$\begin{aligned}\mathbf{A} \odot \mathbf{B} &= [(a_{i1} \wedge b_{1j}) \vee (a_{i2} \wedge b_{2j}) \vee \cdots \vee (a_{ik} \wedge b_{kj})] \\ &= [\vee_{t=1}^{k} (a_{it} \wedge b_{tj})].\end{aligned}$$

Definition 3.63

Let $\mathbf{A}$ be a square zero-one matrix and let r be a positive integer.
The rth *Boolean power of* $\mathbf{A}$ is the matrix

$$\mathbf{A}^{[r]} = \underbrace{\mathbf{A} \odot \mathbf{A} \odot \cdots \odot \mathbf{A}}_{r \text{ times}}.$$

Zero-One Matrices

Definition 3.61

Let $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$ be $m \times n$ zero-one matrices.
The *join* of $\mathbf{A}$ and $\mathbf{B}$ is $\mathbf{A} \vee \mathbf{B} = [a_{ij} \vee b_{ij}]$.
The *meet* of $\mathbf{A}$ and $\mathbf{B}$ is $\mathbf{A} \wedge \mathbf{B} = [a_{ij} \wedge b_{ij}]$.

Definition 3.62

Let $\mathbf{A} = [a_{ij}]$ be an $m \times k$ zero-one matrix and $\mathbf{B} = [b_{ij}]$ be a $k \times n$ zero-one matrix. Then the *Boolean product* of $\mathbf{A}$ and $\mathbf{B}$ is the $m \times n$ matrix

$$\begin{aligned} \mathbf{A} \odot \mathbf{B} &= [(a_{i1} \wedge b_{1j}) \vee (a_{i2} \wedge b_{2j}) \vee \cdots \vee (a_{ik} \wedge b_{kj})] \\ &= [\vee_{t=1}^{k} (a_{it} \wedge b_{tj})]. \end{aligned}$$

Definition 3.63

Let $\mathbf{A}$ be a square zero-one matrix and let r be a positive integer. The rth *Boolean power of* $\mathbf{A}$ is the matrix

$$\mathbf{A}^{[r]} = \underbrace{\mathbf{A} \odot \mathbf{A} \odot \cdots \odot \mathbf{A}}_{r \text{ times}}.$$

Zero-One Matrices

Algorithm 3.15: The Boolean Product.

procedure *Booleanproduct*($\mathbf{A}, \mathbf{B}$: matrices)
for $i := 1$ **to** m **do**
  for $j := 1$ **to** n **do**
    $c_{ij} := 0$
    for $q := 1$ **to** k **do**
      $c_{ij} := c_{ij} \vee (a_{iq} \wedge b_{qj})$
    end
  end
end
$\{\mathbf{C} = [c_{ij}]$ is the Boolean product of $\mathbf{A}$ and $\mathbf{B}\}$