形式语言与计算复杂性

第 1 章 绪论

黄文超

https://faculty.ustc.edu.cn/huangwenchao

$\longrightarrow$ 教学课程 $\longrightarrow$ 形式语言与计算复杂性

Outline

Outline

课前准备

- 课本: Introduction to the Theory of Computation. 2018

- 加入 QQ 群
- 到课要求（点名）
- 作业要求
  - 布置一周内完成并提交, 作业在网上提交
- 考试要求
  - 考试形式：闭卷（但可带一张带有笔记 A4 纸）
  - 【送分】大部分题跟作业相关–请认真完成课后题
  - 【压轴】有一题会考课程中 PPT 中所讲到的一个证明–请仔细理解每一个证明内容

Outline

Outline

1. Automata, Computability, and Complexity

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?

- Theoretical Beginning: 1930s by mathematical logicians

答: The answers *vary* in each of the three areas

- Automata
- Computability
- Complexity

问: What are the answers?

- 见下页, we introduce these parts in *reverse order*

1. Automata, Computability, and Complexity

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?

- Theoretical Beginning: 1930s by mathematical logicians

答: The answers *vary* in each of the three areas

- Automata
- Computability
- Complexity

问: What are the answers?

- 见下页, we introduce these parts in *reverse order*

1. Automata, Computability, and Complexity

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?

- Theoretical Beginning: 1930s by mathematical logicians

答: The answers *vary* in each of the three areas

- Automata
- Computability
- Complexity

问: What are the answers?

- 见下页, we introduce these parts in *reverse order*

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?

答: Let's study *another problem*: **Complexity, 复杂度**

- What makes some problems computationally hard and others easy?

问: Then, what is the answer?

答: *Classifying* problems according to their computational *difficulty*

问: After *classifying* a problem, how to confront the problem?

答: Several *Options*:

- By understanding which aspect of the problem is at the root of the difficulty, *alter it*.
- Settle for *less than a perfect* solution to the problem
  - Some problems are *hard* only in the *worst case* situation, but *easy most of the time*.

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?
答: Let's study *another problem*: **Complexity, 复杂度**

- What makes some problems computationally hard and others easy?

问: Then, what is the answer?
答: *Classifying* problems according to their computational *difficulty*

问: After *classifying* a problem, how to confront the problem?
答: Several *Options*:

- By understanding which aspect of the problem is at the root of the difficulty, *alter it*.
- Settle for *less than a perfect* solution to the problem
  - Some problems are *hard* only in the *worst case* situation, but *easy most of the time*.

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?
答: Let's study *another problem*: **Complexity, 复杂度**

- What makes some problems computationally hard and others easy?

问: Then, what is the answer?
答: *Classifying* problems according to their computational *difficulty*

问: After *classifying* a problem, how to confront the problem?
答: Several *Options*:

- By understanding which aspect of the problem is at the root of the difficulty, *alter it*.
- Settle for *less than a perfect* solution to the problem
  - Some problems are *hard* only in the *worst case* situation, but *easy most of the time*.

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?
答: Let's study *another problem*: **Complexity, 复杂度**

- What makes some problems computationally hard and others easy?

问: Then, what is the answer?
答: *Classifying* problems according to their computational *difficulty*

问: After *classifying* a problem, how to confront the problem?
答: Several *Options*:

- By understanding which aspect of the problem is at the root of the difficulty, *alter it*.
- Settle for *less than a perfect* solution to the problem
  - Some problems are *hard* only in the *worst case* situation, but *easy most of the time*.

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?
答: Let's study *another problem*: **Complexity, 复杂度**

- What makes some problems computationally hard and others easy?

问: Then, what is the answer?
答: *Classifying* problems according to their computational *difficulty*

问: After *classifying* a problem, how to confront the problem?
答: Several *Options*:

- By understanding which aspect of the problem is at the root of the difficulty, *alter it*.
- Settle for *less than a perfect* solution to the problem
  - Some problems are *hard* only in the *worst case* situation, but *easy most of the time*.

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?
答: Let's study *another problem*: **Complexity, 复杂度**

- What makes some problems computationally hard and others easy?

问: Then, what is the answer?
答: *Classifying* problems according to their computational *difficulty*

问: After *classifying* a problem, how to confront the problem?
答: Several *Options*:

- By understanding which aspect of the problem is at the root of the difficulty, *alter it*.
- Settle for *less than a perfect* solution to the problem
  - Some problems are *hard* only in the *worst case* situation, but *easy most of the time*.

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?

答: Let's study *another problem*: **Computability, 可计算性**

- Are there problems that *cannot* be solved by computers?

问: Then, what is the answer?

答: The problem of *determining* whether a mathematical statement is *true or false*.

- discovered by Kurt Gödel, Alan Turing, and Alonzo Church

问: Are the theories of computability and complexity *related*?

答: Yes, and closely

- Both are to *classify* the problems
- Computability theory introduces several of the *concepts* used in complexity theory.

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?
答: Let's study *another problem*: **Computability, 可计算性**

- Are there problems that *cannot* be solved by computers?

问: Then, what is the answer?
答: The problem of *determining* whether a mathematical statement is *true or false*.

- discovered by Kurt Gödel, Alan Turing, and Alonzo Church

问: Are the theories of computability and complexity *related*?
答: Yes, and closely

- Both are to *classify* the problems
- Computability theory introduces several of the *concepts* used in complexity theory.

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?
答: Let's study *another problem*: **Computability, 可计算性**

- Are there problems that *cannot* be solved by computers?

问: Then, what is the answer?
答: The problem of *determining* whether a mathematical statement is *true or false*.

- discovered by Kurt Gödel, Alan Turing, and Alonzo Church

问: Are the theories of computability and complexity *related*?
答: Yes, and closely

- Both are to *classify* the problems
- Computability theory introduces several of the *concepts* used in complexity theory.

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?

答: Let's study *another problem*: **Automata, 自动机**

- *Definitions* and *Properties* of mathematical models of computation?

问: Then, what is the answer?

答: Automata theory

- finite automaton
- context-free grammar
- ...

问: Relations to Computability and Complexity?

答: an excellent place to *begin* the study of the theory of computation.

So..., we firstly study *Automata theory*.

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?

答: Let's study *another problem*: **Automata, 自动机**

- *Definitions* and *Properties* of mathematical models of computation?

问: Then, what is the answer?

答: Automata theory

- finite automaton
- context-free grammar
- ...

问: Relations to Computability and Complexity?

答: an excellent place to *begin* the study of the theory of computation.

So..., we firstly study *Automata theory*.

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?

答: Let's study *another problem*: **Automata, 自动机**

- *Definitions* and *Properties* of mathematical models of computation?

问: Then, what is the answer?

答: Automata theory

- finite automaton
- context-free grammar
- ...

问: Relations to Computability and Complexity?

答: an excellent place to *begin* the study of the theory of computation.

So..., we firstly study *Automata theory*.

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?
答: Let's study *another problem*: **Automata, 自动机**

- *Definitions* and *Properties* of mathematical models of computation?

问: Then, what is the answer?
答: Automata theory

- finite automaton
- context-free grammar
- ...

问: Relations to Computability and Complexity?
答: an excellent place to *begin* the study of the theory of computation.

So..., we firstly study *Automata theory*.

问: What are the **fundamental** *capabilities* and *limitations* of *computers*?

答: Let's study *another problem*: **Automata, 自动机**

- *Definitions* and *Properties* of mathematical models of computation?

问: Then, what is the answer?

答: Automata theory

- finite automaton
- context-free grammar
- ...

问: Relations to Computability and Complexity?

答: an excellent place to *begin* the study of the theory of computation.

So..., we firstly study *Automata theory*.

Outline

Outline

定义: Set, $\in, \notin, \subseteq$

A *set* is a group of objects represented as a unit.

- The objects in a set are called its *elements* or *members*.
- The symbols $\in$ and $\notin$ denote set *membership* and *nonmembership*.
- A is a *subset* of B, written $A \subseteq B$, if every member of A also is a member of B.

例: Set

The set S contains the elements 7, 21, and 57:

$$S = \{7, 21, 57\}$$

$$7 \in \{7, 21, 57\}, \quad 8 \notin \{7, 21, 57\}, \quad \{7, 21\} \subseteq \{7, 21, 57\}$$

定义: Set, $\in, \notin, \subseteq$

A *set* is a group of objects represented as a unit.

- The objects in a set are called its *elements* or *members*.
- The symbols $\in$ and $\notin$ denote set *membership* and *nonmembership*.
- A is a *subset* of B, written $A \subseteq B$, if every member of A also is a member of B.

例: Set

The set S contains the elements 7, 21, and 57:

$$S = \{7, 21, 57\}$$

$$7 \in \{7, 21, 57\}, \quad 8 \notin \{7, 21, 57\}, \quad \{7, 21\} \subseteq \{7, 21, 57\}$$

问: Given $S = \{7, 21, 57\}$, is $S' = \{57, 7, 7, 7, 21\}$ the *same* set with S?
答: Yes

问: How to differentiate between S and S'?
答: Define *multiset*, i.e., S and S' are different as multisets.

定义: Infinite set, $\mathcal{N}, \mathcal{Z}, \emptyset$

An *infinite set* contains infinitely many elements, e.g.,

- natural numbers: $\mathcal{N} = \{1, 2, 3, \dots\}$
- integers: $\mathcal{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$

The set with zero members is called the *empty set* and is written $\emptyset$.

问: Given $S = \{7, 21, 57\}$, is $S' = \{57, 7, 7, 7, 21\}$ the *same* set with S?
答: Yes

问: How to differentiate between S and S'?
答: Define *multiset*, i.e., S and S' are different as multisets.

定义: Infinite set, $\mathcal{N}, \mathcal{Z}, \emptyset$

An *infinite set* contains infinitely many elements, e.g.,

- natural numbers: $\mathcal{N} = \{1, 2, 3, \dots\}$
- integers: $\mathcal{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$

The set with zero members is called the *empty set* and is written $\emptyset$.

问: Given $S = \{7, 21, 57\}$, is $S' = \{57, 7, 7, 7, 21\}$ the *same* set with S?
答: Yes

问: How to differentiate between S and S'?
答: Define *multiset*, i.e., S and S' are different as multisets.

定义: Infinite set, $\mathcal{N}, \mathcal{Z}, \emptyset$

An *infinite set* contains infinitely many elements, e.g.,

- natural numbers: $\mathcal{N} = \{1, 2, 3, \dots\}$
- integers: $\mathcal{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$

The set with zero members is called the *empty set* and is written $\emptyset$.

定义: $\{n \mid \text{rule about } n\}$

A set containing elements according to some rule, e.g.,

- $\{n \mid n = m^2 \text{ for some } m \in N\}$ means the set of perfect squares

定义: union $\cup$, intersection $\cap$, complement

- $A \cup B$: the set we get by combining all the elements in A and B into a single set
- $A \cap B$: the set of elements that are in both A and B
- complement of A, written $\bar{A}$: the set of all elements under consideration that are *not* in A

定义: $\{n \mid \text{rule about } n\}$

A set containing elements according to some rule, e.g.,

- $\{n \mid n = m^2 \text{ for some } m \in N\}$ means the set of perfect squares

定义: union $\cup$, intersection $\cap$, complement

- $A \cup B$: the set we get by combining all the elements in A and B into a single set
- $A \cap B$: the set of elements that are in both A and B
- complement of A, written $\bar{A}$: the set of all elements under consideration that are *not* in A

Outline

定义: Sequence

A *sequence* of objects is a list of these objects in some order.

- For example, the sequence 7, 21, 57 would be written: $(7, 21, 57)$
- $(7, 21, 57)$ is not the same as $(57, 7, 21)$

定义: Tuple, k-tuple

Finite sequences often are called *tuples*.

- A sequence with k elements is a k-tuple.
- A 2-tuple is also called an ordered pair, e.g., (2,3)

定义: Power Set

The *power set* of A is the set of all subsets of A. If A is the set $\{0, 1\}$,

- the power set of A is the set $\{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$.

定义: Sequence

A *sequence* of objects is a list of these objects in some order.

- For example, the sequence 7, 21, 57 would be written: $(7, 21, 57)$
- $(7, 21, 57)$ is not the same as $(57, 7, 21)$

定义: Tuple, k-tuple

Finite sequences often are called *tuples*.

- A sequence with k elements is a k-tuple.
- A 2-tuple is also called an ordered pair, e.g., (2,3)

定义: Power Set

The *power set* of A is the set of all subsets of A. If A is the set $\{0, 1\}$,

- the power set of A is the set $\{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$.

定义: Sequence

A *sequence* of objects is a list of these objects in some order.

- For example, the sequence 7, 21, 57 would be written: $(7, 21, 57)$
- $(7, 21, 57)$ is not the same as $(57, 7, 21)$

定义: Tuple, k-tuple

Finite sequences often are called *tuples*.

- A sequence with k elements is a k-tuple.
- A 2-tuple is also called an ordered pair, e.g., (2,3)

定义: Power Set

The *power set* of A is the set of all subsets of A. If A is the set $\{0, 1\}$,

- the power set of A is the set $\{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$.

定义: Cartesian product, or Cross product, $\times$

If A and B are two sets, the *cartesian product* or *cross product* of A and B, written $A \times B$, is the *set* of *all ordered pairs* wherein the first element is a member of A and the second element is a member of B

例:

If $A = \{1, 2\}$ and $B = \{x, y, z\}$,

$$A \times B = \{(1, x), (1, y), (1, z), (2, x), (2, y), (2, z)\}$$

$$\begin{aligned} A \times B \times A = \{&(1, x, 1), (1, x, 2), (1, y, 1), (1, y, 2), (1, z, 1), (1, z, 2), \\ &(2, x, 1), (2, x, 2), (2, y, 1), (2, y, 2), (2, z, 1), (2, z, 2)\} \end{aligned}$$

定义: Cartesian product, or Cross product, $\times$

If A and B are two sets, the *cartesian product* or *cross product* of A and B, written $A \times B$, is the *set* of *all ordered pairs* wherein the first element is a member of A and the second element is a member of B

例:

If $A = \{1, 2\}$ and $B = \{x, y, z\}$,

$$A \times B = \{(1, x), (1, y), (1, z), (2, x), (2, y), (2, z)\}$$

$$\begin{aligned} A \times B \times A = \{&(1, x, 1), (1, x, 2), (1, y, 1), (1, y, 2), (1, z, 1), (1, z, 2), \\ &(2, x, 1), (2, x, 2), (2, y, 1), (2, y, 2), (2, z, 1), (2, z, 2)\} \end{aligned}$$

定义: Cartesian product, or Cross product, $\times$

If A and B are two sets, the *cartesian product* or *cross product* of A and B, written $A \times B$, is the *set* of *all ordered pairs* wherein the first element is a member of A and the second element is a member of B

例:

If $A = \{1, 2\}$ and $B = \{x, y, z\}$,

$$A \times B = \{(1, x), (1, y), (1, z), (2, x), (2, y), (2, z)\}$$

$$\begin{aligned} A \times B \times A = &\{(1, x, 1), (1, x, 2), (1, y, 1), (1, y, 2), (1, z, 1), (1, z, 2), \\ &(2, x, 1), (2, x, 2), (2, y, 1), (2, y, 2), (2, z, 1), (2, z, 2)\} \end{aligned}$$

定义: A^k

$$\underbrace{A \times A \times \cdots \times A}_{k} = A^k$$

例:

The set $\mathcal{N}^2$ equals $\mathcal{N} \times \mathcal{N}$. It consists of all ordered pairs of natural numbers.

- We also may write it as $\{(i, j) \mid i, j \geq 1\}$.

定义: A^k

$$\underbrace{A \times A \times \cdots \times A}_{k} = A^k$$

例:

The set $\mathcal{N}^2$ equals $\mathcal{N} \times \mathcal{N}$. It consists of all ordered pairs of natural numbers.

- We also may write it as $\{(i, j) \mid i, j \geq 1\}$.

Outline

定义: Function, mapping

A *function* is an object that sets up an input–output relationship.

- If f is a function whose output value is b when the input value is a:

$$f(a) = b$$

- A function also is called a *mapping*, and, if $f(a) = b$, we say that f *maps* a to b.

定义: Domain, Range, $f : D \to R$

- The *set* of possible *inputs* to the function is called its *domain*.
- The *outputs* of a function come from a set called its *range*.
- Notation: f is a function with domain D and range R is

$$f : D \to R$$

e.g., addition function for integers: $add : \mathcal{Z} \times \mathcal{Z} \to \mathcal{Z}$

定义: Function, mapping

A *function* is an object that sets up an input–output relationship.

- If f is a function whose output value is b when the input value is a:

$$f(a) = b$$

- A function also is called a *mapping*, and, if $f(a) = b$, we say that f *maps* a to b.

定义: Domain, Range, $f : D \to R$

- The *set* of possible *inputs* to the function is called its *domain*.
- The *outputs* of a function come from a set called its *range*.
- Notation: f is a function with domain D and range R is

$$f : D \to R$$

e.g., addition function for integers: $add : \mathcal{Z} \times \mathcal{Z} \to \mathcal{Z}$

定义: k-ary function

When the domain of a function f is $A_1 \times \cdots \times A_k$ for some sets $A_1, \ldots, A_k$, the input to f is a k-tuple $(a_1, a_2, \ldots, a_k)$

- We call the a_i the *arguments* to f.
- A function with k arguments is called a *k-ary* function
- k is called the *arity* of the function. f is called a
  - *unary function*: if k is 1
  - *binary function*: if k is 2
- For binary function, e.g., $add(a, b)$, there are two notations:
  - *infix (中缀) notation*: $a\ add\ b$
  - *prefix (前缀) notation*: $add(a, b)$

定义: Predicate, or Property

A *predicate* or *property* is a function whose *range* is $\{\text{TRUE}, \text{FALSE}\}$.

- e.g., let even be a property that is TRUE if its input is an even number and FALSE if its input is an odd number.
  - Thus, $even(4) = \text{TRUE}$ and $even(5) = \text{FALSE}$

定义: relation, k-ary relation

A *property* whose *domain* is a set of k-tuples $A \times \cdots \times A$ is called a *relation*, a *k-ary relation*, or a k-ary relation on A.

- *binary relation*: 2-ary relation
- If R is a binary relation, the statement aRb means that $aRb = \text{TRUE}$
- If R is a k-ary relation, the statement $R(a_1, \ldots, a_k)$ means that $R(a_1, \ldots, a_k) = \text{TRUE}$.

定义: Predicate, or Property

A *predicate* or *property* is a function whose *range* is $\{\text{TRUE}, \text{FALSE}\}$.

- e.g., let even be a property that is TRUE if its input is an even number and FALSE if its input is an odd number.
  - Thus, $even(4) = \text{TRUE}$ and $even(5) = \text{FALSE}$

定义: relation, k-ary relation

A *property* whose *domain* is a set of k-tuples $A \times \cdots \times A$ is called a *relation*, a *k-ary relation*, or a k-ary relation on A.

- *binary relation*: 2-ary relation
- If R is a binary relation, the statement aRb means that $aRb = \text{TRUE}$
- If R is a k-ary relation, the statement $R(a_1, \ldots, a_k)$ means that $R(a_1, \ldots, a_k) = \text{TRUE}$.

定义: Equivalence relation

A binary relation R is an *equivalence relation* if R satisfies three conditions:

1. R is *reflexive* if for every x, xRx;
2. R is *symmetric* if for every x and y, xRy implies yRx; and
3. R is *transitive* if for every x, y, and z, xRy and yRz implies xRz.

例: $\equiv_7$

Define an equivalence relation on the natural numbers, written $\equiv_7$. For $i, j \in N$, say that $i \equiv_7 j$, if $i - j$ is a multiple of 7.

定义: Equivalence relation

A binary relation R is an *equivalence relation* if R satisfies three conditions:

1. R is *reflexive* if for every x, xRx;
2. R is *symmetric* if for every x and y, xRy implies yRx; and
3. R is *transitive* if for every x, y, and z, xRy and yRz implies xRz.

例: $\equiv_7$

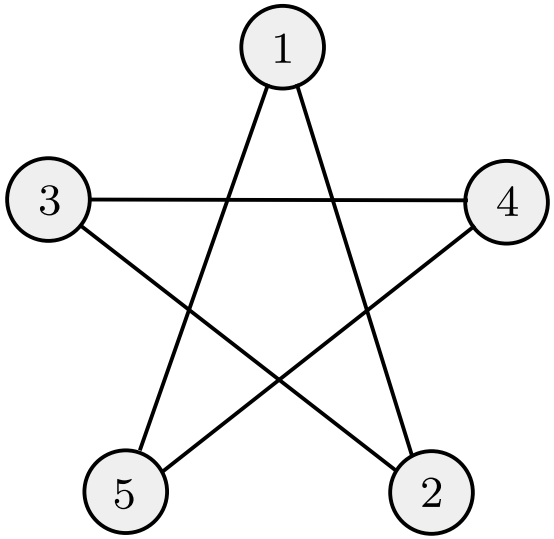
Define an equivalence relation on the natural numbers, written $\equiv_7$. For $i, j \in N$, say that $i \equiv_7 j$, if $i - j$ is a multiple of 7.

Outline

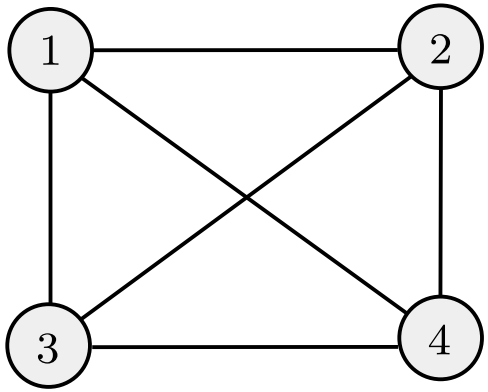
定义: Graph, Nodes, Edges

An *undirected graph*, or simply a *graph*, is a set of points with lines connecting some of the points.

- The points are called *nodes* or *vertices*
- The lines are called *edges*



(a)

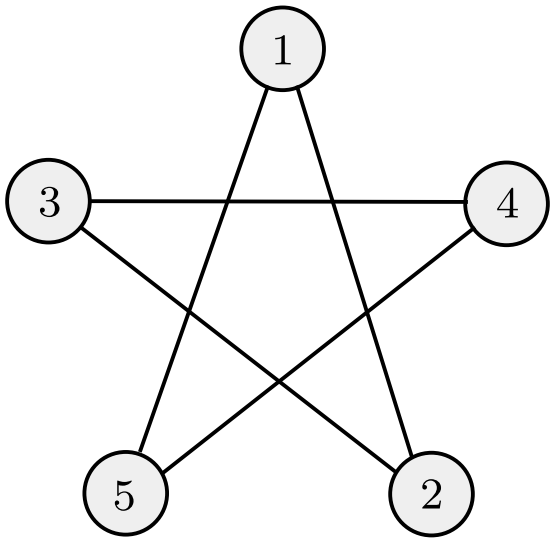


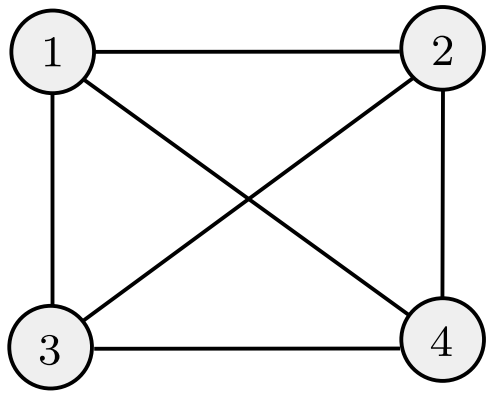
(b)

定义: Degree

The number of edges at a particular node is the *degree* of that node.

- In (a), all the nodes have degree 2
- In (b), all the nodes have degree 3
- *No more than one edge* is allowed between any two nodes.
- We may allow an edge from a node to itself, called a *self-loop*

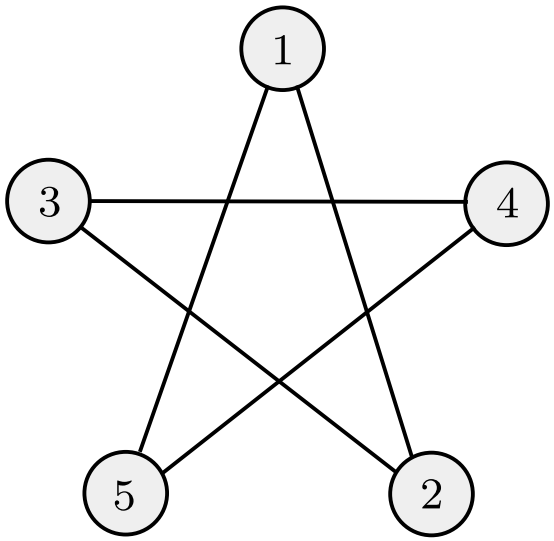




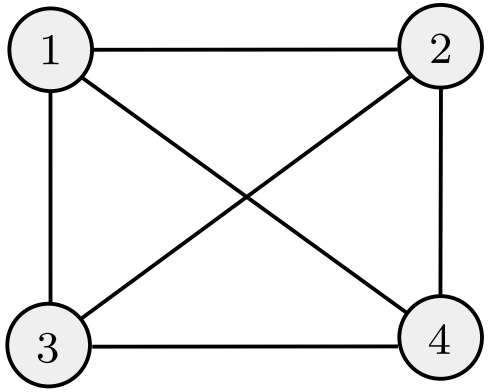
定义: $G = (V, E)$

In a graph G, if V is the set of nodes of G and E is the set of edges, we say $G = (V, E)$

- In (a): $G = (\{1, 2, 3, 4, 5\}, \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 1)\})$
- In (b): $G = (\{1, 2, 3, 4\}, \{(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)\})$



(a)

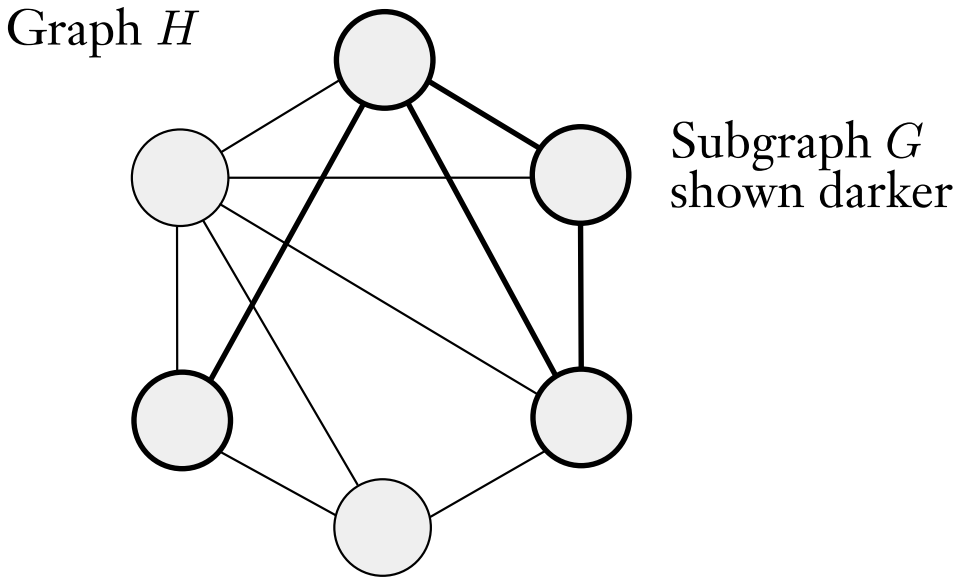


(b)

定义: Subgraph

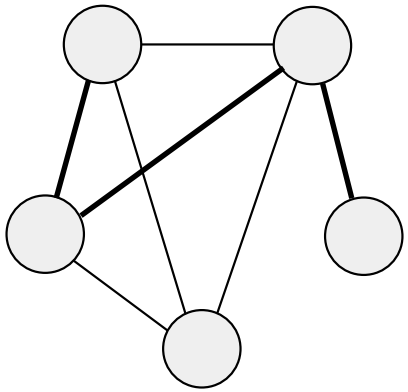
We say that graph G is a *subgraph* of graph H if

- the nodes of G are a subset of the nodes of H
- the edges of G are the edges of H on the corresponding nodes



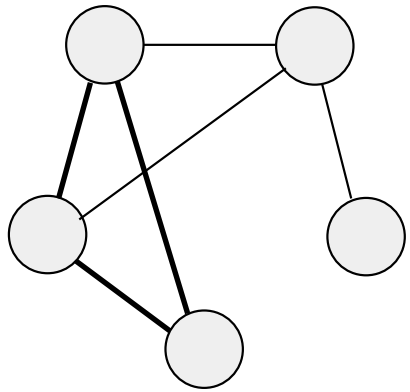
定义:Path, Simple Path, Connected

- A *path* in a graph is a sequence of nodes connected by edges.
- A *simple path* is a path that doesn't repeat any nodes.
- A graph is *connected* if every two nodes have a path between them.



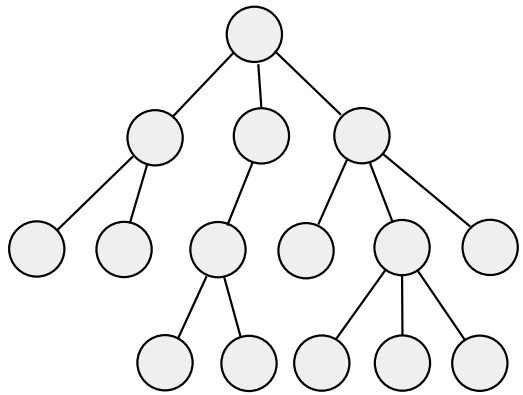
定义: Cycle, Simple Cycle

- A path is a *cycle* if it starts and ends in the same node.
- A *simple cycle* is one that contains at least three nodes and repeats only the first and last nodes.



定义: Tree, Root, Leaf

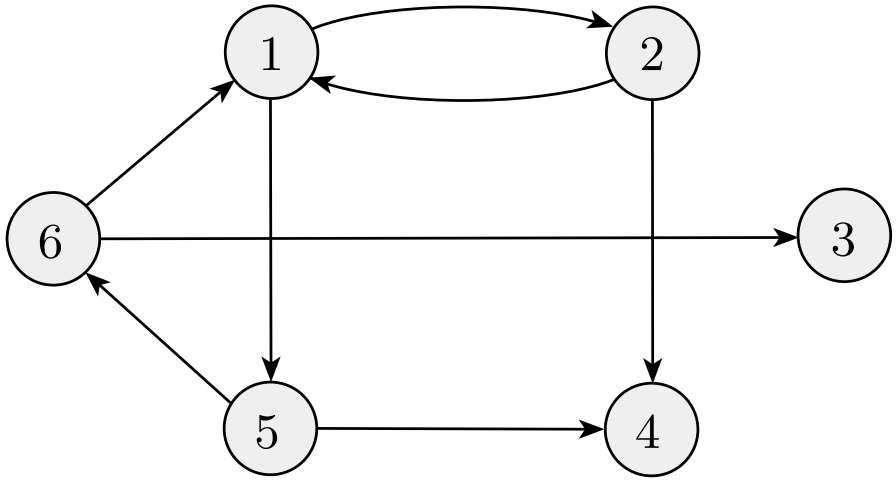
- A graph is a *tree* if it is connected and has no simple cycles
- A tree may contain a specially designated node called the *root*
- The nodes of degree 1 in a tree, other than the root, are called the *leaves* of the tree



定义: Directed Graph

A *directed graph* has arrows instead of lines

- The number of arrows pointing *from* a particular node is the *outdegree* of that node
- The number of arrows pointing *to* a particular node is the *indegree*.

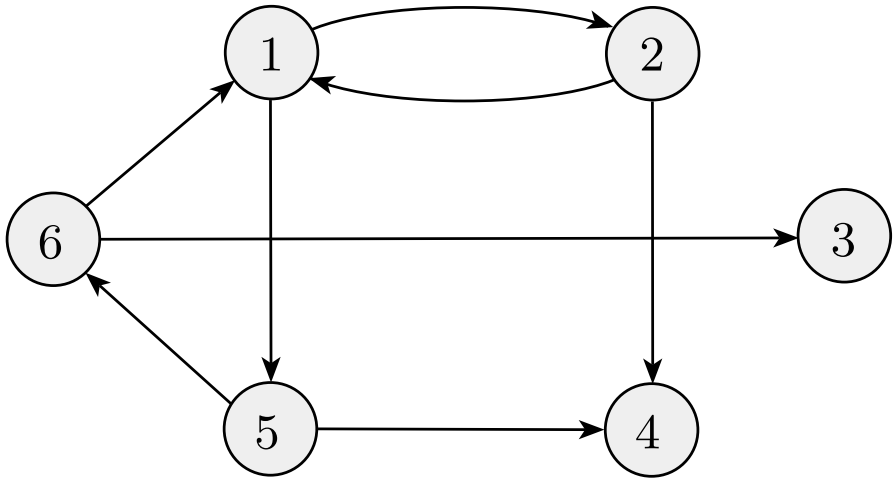


The formal description of the graph is:

$$(\{1,2,3,4,5,6\},\{(1,2),(1,5),(2,1),(2,4),(5,4),(5,6),(6,1),(6,3)\})$$

定义: Directed path, Strongly connected

- A path in which all the arrows point in the same direction as its steps is called a *directed path*
- A directed graph is *strongly connected* if a directed path connects every two nodes



Outline

定义: Alphabet, Symbols, Σ, Γ

- *Alphabet*: *any* nonempty *finite* set
- *Symbols*: The members of the alphabet are the *symbols* of the alphabet
- Σ, Γ: *alphabets* and a *typewriter font* for symbols from an alphabet

例: Σ, Γ

- $\Sigma_1 = \{0, 1\}$
- $\Sigma_2 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$
- $\Gamma = \{0, 1, x, y, z\}$

定义: Alphabet, Symbols, Σ, Γ

- *Alphabet*: *any* nonempty *finite* set
- *Symbols*: The members of the alphabet are the *symbols* of the alphabet
- Σ, Γ: *alphabets* and a *typewriter font* for symbols from an alphabet

例: Σ, Γ

- $\Sigma_1 = \{0, 1\}$
- $\Sigma_2 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$
- $\Gamma = \{0, 1, x, y, z\}$

定义: String, Length w, Empty String ε, Reverse w_R, Substring

- A *string over an alphabet* is a finite sequence of symbols from that alphabet, usually written next to one another and not separated by commas, e.g.,
  - If $\Gamma_1 = \{0, 1\}$, then 01001 is a string over Γ_1
  - If $\Gamma_2 = \{a, b, c, \ldots, z\}$, then $abracadabra$ is a string over Γ_2
- If w is a string over Σ, the *length* of w, written $|w|$, is the number of symbols that it contains.
- The string of length zero is called the *empty string* and is written ε.
- If w has length n, we can write $w = w_1 w_2 \cdots w_n$ where each $w_i \in \Sigma$. The *reverse* of w, written w_R, is the string obtained by writing w in the opposite order (i.e., $w_n w_{n-1} \cdots w_1$).
- String z is a *substring* of w if z appears consecutively within w.
  - For example, cad is a substring of $abracadabra$.

定义: String, Length w, Empty String ε, Reverse w_R, Substring

- A *string over an alphabet* is a finite sequence of symbols from that alphabet, usually written next to one another and not separated by commas, e.g.,
  - If $\Gamma_1 = \{0, 1\}$, then 01001 is a string over Γ_1
  - If $\Gamma_2 = \{a, b, c, \ldots, z\}$, then $abracadabra$ is a string over Γ_2
- If w is a string over Σ, the *length* of w, written $|w|$, is the number of symbols that it contains.
- The string of length zero is called the *empty string* and is written ε.
- If w has length n, we can write $w = w_1 w_2 \cdots w_n$ where each $w_i \in \Sigma$. The *reverse* of w, written w_R, is the string obtained by writing w in the opposite order (i.e., $w_n w_{n-1} \cdots w_1$).
- String z is a *substring* of w if z appears consecutively within w.
  - For example, cad is a substring of $abracadabra$.

定义: String, Length w, Empty String ε, Reverse w_R, Substring

- A *string over an alphabet* is a finite sequence of symbols from that alphabet, usually written next to one another and not separated by commas, e.g.,
  - If $\Gamma_1 = \{0, 1\}$, then 01001 is a string over Γ_1
  - If $\Gamma_2 = \{a, b, c, \ldots, z\}$, then $abracadabra$ is a string over Γ_2
- If w is a string over Σ, the *length* of w, written $|w|$, is the number of symbols that it contains.
- The string of length zero is called the *empty string* and is written ε.
- If w has length n, we can write $w = w_1 w_2 \cdots w_n$ where each $w_i \in \Sigma$. The *reverse* of w, written w_R, is the string obtained by writing w in the opposite order (i.e., $w_n w_{n-1} \cdots w_1$).
- String z is a *substring* of w if z appears consecutively within w.
  - For example, cad is a substring of $abracadabra$.

定义: String, Length w, Empty String ε, Reverse w_R, Substring

- A *string over an alphabet* is a finite sequence of symbols from that alphabet, usually written next to one another and not separated by commas, e.g.,
  - If $\Gamma_1 = \{0, 1\}$, then 01001 is a string over Γ_1
  - If $\Gamma_2 = \{a, b, c, \ldots, z\}$, then $abracadabra$ is a string over Γ_2
- If w is a string over Σ, the *length* of w, written $|w|$, is the number of symbols that it contains.
- The string of length zero is called the *empty string* and is written ε.
- If w has length n, we can write $w = w_1 w_2 \cdots w_n$ where each $w_i \in \Sigma$. The *reverse* of w, written w_R, is the string obtained by writing w in the opposite order (i.e., $w_n w_{n-1} \cdots w_1$).
- String z is a *substring* of w if z appears consecutively within w.
  - For example, cad is a substring of $abracadabra$.

定义: String, Length w, Empty String ε, Reverse w_R, Substring

- A *string over an alphabet* is a finite sequence of symbols from that alphabet, usually written next to one another and not separated by commas, e.g.,
  - If $\Gamma_1 = \{0, 1\}$, then 01001 is a string over Γ_1
  - If $\Gamma_2 = \{a, b, c, \ldots, z\}$, then $abracadabra$ is a string over Γ_2
- If w is a string over Σ, the *length* of w, written $|w|$, is the number of symbols that it contains.
- The string of length zero is called the *empty string* and is written ε.
- If w has length n, we can write $w = w_1 w_2 \cdots w_n$ where each $w_i \in \Sigma$. The *reverse* of w, written w_R, is the string obtained by writing w in the opposite order (i.e., $w_n w_{n-1} \cdots w_1$).
- String z is a *substring* of w if z appears consecutively within w.
  - For example, cad is a substring of $abracadabra$.

定义: Concatenation xy, x^k

If we have string x of length m and string y of length n, the *concatenation* of x and y, written xy, is the string obtained by appending y to the end of x, as in $x_1 \cdots x_m y_1 \cdots y_n$

- To concatenate a string with itself many times, we use the superscript notation x^k to mean

$$\underbrace{xx \ldots x}_{k}$$

定义: Lexicographic order, String order

- The *lexicographic order* of strings is the same as the familiar dictionary order.
- *Shortlex order* or *string order*: identical to lexicographic order, *except* that *shorter* strings *precede longer* strings.
  - Thus the *string ordering* of all strings over the alphabet $\{0, 1\}$ is

$$(\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots)$$

定义: Language, Prefix-Free

- A ***language*** is a set of strings.
- String x is a *prefix* of string y if a string z exists where $xz = y$
  - x is a *proper prefix* of y if in addition $x \neq y$.
- A language is *prefix-free* if no member is a *proper prefix* of another member

定义: Lexicographic order, String order

- The *lexicographic order* of strings is the same as the familiar dictionary order.
- *Shortlex order* or *string order*: identical to lexicographic order, *except* that *shorter* strings *precede longer* strings.
  - Thus the *string ordering* of all strings over the alphabet $\{0, 1\}$ is

$$(\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots)$$

定义: Language, Prefix-Free

- A ***language*** is a set of strings.
- String x is a *prefix* of string y if a string z exists where $xz = y$
  - x is a *proper prefix* of y if in addition $x \neq y$.
- A language is *prefix-free* if no member is a *proper prefix* of another member

Outline

定义: Boolean logic, Boolean value, Boolean operations, Operands

- *Boolean logic*: a mathematical system built around the two values
  - **TRUE** and **FALSE**: The values TRUE and FALSE are called the *Boolean values* and are often represented by the values *1 and 0*.
- *Boolean operations*: which manipulate Boolean values, i.e.,
  - *negation* $\neg$: $\neg 0 = 1$ and $\neg 1 = 0$
  - *conjunction* or *AND* $\wedge$: The conjunction of two Boolean values is 1 if both of those values are 1.
  - *disjunction* or *OR* $\vee$: The disjunction of two Boolean values is 1 if either of those values is 1.
- *Operands*: Inputs of the operations

$$0 \wedge 0 = 0 \qquad 0 \vee 0 = 0 \qquad \neg 0 = 1$$
$$0 \wedge 1 = 0 \qquad 0 \vee 1 = 1 \qquad \neg 1 = 0$$
$$1 \wedge 0 = 0 \qquad 1 \vee 0 = 1$$
$$1 \wedge 1 = 1 \qquad 1 \vee 1 = 1$$

定义: Boolean logic, Boolean value, Boolean operations, Operands

- *Boolean logic*: a mathematical system built around the two values
  - **TRUE** and **FALSE**: The values TRUE and FALSE are called the *Boolean values* and are often represented by the values *1 and 0*.
- *Boolean operations*: which manipulate Boolean values, i.e.,
  - *negation* $\neg$: $\neg 0 = 1$ and $\neg 1 = 0$
  - *conjunction* or *AND* $\wedge$: The conjunction of two Boolean values is 1 if both of those values are 1.
  - *disjunction* or *OR* $\vee$: The disjunction of two Boolean values is 1 if either of those values is 1.
- *Operands*: Inputs of the operations

$$0 \wedge 0 = 0 \qquad 0 \vee 0 = 0 \qquad \neg 0 = 1$$
$$0 \wedge 1 = 0 \qquad 0 \vee 1 = 1 \qquad \neg 1 = 0$$
$$1 \wedge 0 = 0 \qquad 1 \vee 0 = 1$$
$$1 \wedge 1 = 1 \qquad 1 \vee 1 = 1$$

定义: Boolean logic, Boolean value, Boolean operations, Operands

- *Boolean logic*: a mathematical system built around the two values
  - **TRUE** and **FALSE**: The values TRUE and FALSE are called the *Boolean values* and are often represented by the values *1 and 0*.
- *Boolean operations*: which manipulate Boolean values, i.e.,
  - *negation* $\neg$: $\neg 0 = 1$ and $\neg 1 = 0$
  - *conjunction* or *AND* $\wedge$: The conjunction of two Boolean values is 1 if both of those values are 1.
  - *disjunction* or *OR* $\vee$: The disjunction of two Boolean values is 1 if either of those values is 1.
- *Operands*: Inputs of the operations

$$0 \wedge 0 = 0 \qquad 0 \vee 0 = 0 \qquad \neg 0 = 1$$
$$0 \wedge 1 = 0 \qquad 0 \vee 1 = 1 \qquad \neg 1 = 0$$
$$1 \wedge 0 = 0 \qquad 1 \vee 0 = 1$$
$$1 \wedge 1 = 1 \qquad 1 \vee 1 = 1$$

定义: Boolean logic, Boolean value, Boolean operations, Operands

- *Boolean logic*: a mathematical system built around the two values
  - **TRUE** and **FALSE**: The values TRUE and FALSE are called the *Boolean values* and are often represented by the values *1 and 0*.
- *Boolean operations*: which manipulate Boolean values, i.e.,
  - *negation* $\neg$: $\neg 0 = 1$ and $\neg 1 = 0$
  - *conjunction* or *AND* $\wedge$: The conjunction of two Boolean values is 1 if both of those values are 1.
  - *disjunction* or *OR* $\vee$: The disjunction of two Boolean values is 1 if either of those values is 1.
- *Operands*: Inputs of the operations

$$0 \wedge 0 = 0 \qquad 0 \vee 0 = 0 \qquad \neg 0 = 1$$
$$0 \wedge 1 = 0 \qquad 0 \vee 1 = 1 \qquad \neg 1 = 0$$
$$1 \wedge 0 = 0 \qquad 1 \vee 0 = 1$$
$$1 \wedge 1 = 1 \qquad 1 \vee 1 = 1$$

定义: Exclusive, XOR, Equality, Implication

- *Exclusive*, or *XOR* $\oplus$: 1, if either but not both of its two operands is 1
- *Equality* $\leftrightarrow$: 1, if both of its operands have the same value
- *implication* $\rightarrow$:
  - 0, if its first operand is 1 and its second operand is 0
  - 1, otherwise

$$
\begin{array}{ll}
P \vee Q & \neg(\neg P \wedge \neg Q) \\
P \rightarrow Q & \neg P \vee Q \\
P \leftrightarrow Q & (P \rightarrow Q) \wedge (Q \rightarrow P) \\
P \oplus Q & \neg(P \leftrightarrow Q)
\end{array}
$$

定理: Distributive Law (分配律) (a Boolean Version)

- $P \wedge (Q \vee R)$ equals $(P \wedge Q) \vee (P \wedge R)$
- $P \vee (Q \wedge R)$ equals $(P \vee Q) \wedge (P \vee R)$

定义: Exclusive, XOR, Equality, Implication

- *Exclusive*, or *XOR* $\oplus$: 1, if either but not both of its two operands is 1
- *Equality* $\leftrightarrow$: 1, if both of its operands have the same value
- *implication* $\rightarrow$:
  - 0, if its first operand is 1 and its second operand is 0
  - 1, otherwise

$$
\begin{array}{ll}
P \vee Q & \neg(\neg P \wedge \neg Q) \\
P \rightarrow Q & \neg P \vee Q \\
P \leftrightarrow Q & (P \rightarrow Q) \wedge (Q \rightarrow P) \\
P \oplus Q & \neg(P \leftrightarrow Q)
\end{array}
$$

定理: Distributive Law (分配律) (a Boolean Version)

- $P \wedge (Q \vee R)$ equals $(P \wedge Q) \vee (P \wedge R)$
- $P \vee (Q \wedge R)$ equals $(P \vee Q) \wedge (P \vee R)$

定义: Exclusive, XOR, Equality, Implication

- *Exclusive*, or *XOR* $\oplus$: 1, if either but not both of its two operands is 1
- *Equality* $\leftrightarrow$: 1, if both of its operands have the same value
- *implication* $\rightarrow$:
  - 0, if its first operand is 1 and its second operand is 0
  - 1, otherwise

$$\begin{array}{ll} P \vee Q & \neg(\neg P \wedge \neg Q) \\ P \rightarrow Q & \neg P \vee Q \\ P \leftrightarrow Q & (P \rightarrow Q) \wedge (Q \rightarrow P) \\ P \oplus Q & \neg(P \leftrightarrow Q) \end{array}$$

定理: Distributive Law (分配律) (a Boolean Version)

- $P \wedge (Q \vee R)$ equals $(P \wedge Q) \vee (P \wedge R)$
- $P \vee (Q \wedge R)$ equals $(P \vee Q) \wedge (P \vee R)$

Outline

3. Definitions, Theorems, and Proofs

These *three entities* are *central* to *every* mathematical subject

- *Definitions* (定义): describe the objects and notions that we use.
  - A mathematical *statement* about the definitions: expresses that some object has a certain property.
    - The statement *may or may not* be *true*
  - *No ambiguity* for both *definitions* and *statements*
- A *proof* (证明): is a convincing logical argument that a statement is true.
- A *theorem* (定理) is a mathematical statement proved true.
  - *lemmas* (引理): we prove statements, called *lemmas*, that are interesting only because they assist in the proof of another, more significant statement
  - *corollaries* (推论) of a theorem: the theorem or its proof may allow us to conclude easily that other, related statements, called *corollaries* are true

3. Definitions, Theorems, and Proofs

These *three entities* are *central* to *every* mathematical subject

- *Definitions* (定义): describe the objects and notions that we use.
  - A mathematical *statement* about the definitions: expresses that some object has a certain property.
    - The statement *may or may not* be *true*
  - *No ambiguity* for both *definitions* and *statements*
- A *proof* (证明): is a convincing logical argument that a statement is true.
- A *theorem* (定理) is a mathematical statement proved true.
  - *lemmas* (引理): we prove statements, called *lemmas*, that are interesting only because they assist in the proof of another, more significant statement
  - *corollaries* (推论) of a theorem: the theorem or its proof may allow us to conclude easily that other, related statements, called *corollaries* are true

3. Definitions, Theorems, and Proofs

These *three entities* are *central* to *every* mathematical subject

- *Definitions* (定义): describe the objects and notions that we use.
  - A mathematical *statement* about the definitions: expresses that some object has a certain property.
    - The statement *may or may not* be *true*
  - *No ambiguity* for both *definitions* and *statements*
- A *proof* (证明): is a convincing logical argument that a statement is true.
- A *theorem* (定理) is a mathematical statement proved true.
  - *lemmas* (引理): we prove statements, called *lemmas*, that are interesting only because they assist in the proof of another, more significant statement
  - *corollaries* (推论) of a theorem: the theorem or its proof may allow us to conclude easily that other, related statements, called *corollaries* are true

3. Definitions, Theorems, and Proofs

How to find proofs?

- Step 0: finding proofs *isn't always easy*
- Step 1: *carefully* read the statement you want to prove
- Step 2: when you want to prove a statement or part thereof, try to get an *intuitive*, *"gut" feeling* of why it should be true.
- Step 3: when you believe that you have found the proof, you *must* write it up *properly*.

Advices:

- Be patient
- Come back to it
  - think about it a bit, leave it, and then *return* a few minutes or *hours* later
- Be neat
- Be concise

3. Definitions, Theorems, and Proofs

How to find proofs?

- Step 0: finding proofs *isn't always easy*
- Step 1: *carefully* read the statement you want to prove
- Step 2: when you want to prove a statement or part thereof, try to get an *intuitive*, *"gut" feeling* of why it should be true.
- Step 3: when you believe that you have found the proof, you *must* write it up *properly*.

Advices:

- Be patient
- Come back to it
  - think about it a bit, leave it, and then *return* a few minutes or *hours* later
- Be neat
- Be concise

3. Definitions, Theorems, and Proofs

How to find proofs?

- Step 0: finding proofs *isn't always easy*
- Step 1: *carefully* read the statement you want to prove
- Step 2: when you want to prove a statement or part thereof, try to get an *intuitive*, *"gut" feeling* of why it should be true.
- Step 3: when you believe that you have found the proof, you *must* write it up *properly*.

Advices:

- Be patient
- Come back to it
  - think about it a bit, leave it, and then *return* a few minutes or *hours* later
- Be neat
- Be concise

3. Definitions, Theorems, and Proofs

How to find proofs?

- Step 0: finding proofs *isn't always easy*
- Step 1: *carefully* read the statement you want to prove
- Step 2: when you want to prove a statement or part thereof, try to get an *intuitive*, *"gut" feeling* of why it should be true.
- Step 3: when you believe that you have found the proof, you *must* write it up *properly*.

Advices:

- Be patient
- Come back to it
  - think about it a bit, leave it, and then *return* a few minutes or *hours* later
- Be neat
- Be concise

3. Definitions, Theorems, and Proofs

How to find proofs?

- Step 0: finding proofs *isn't always easy*
- Step 1: *carefully* read the statement you want to prove
- Step 2: when you want to prove a statement or part thereof, try to get an *intuitive*, *"gut" feeling* of why it should be true.
- Step 3: when you believe that you have found the proof, you *must* write it up *properly*.

Advices:

- Be patient
- Come back to it
  - think about it a bit, leave it, and then *return* a few minutes or *hours* later
- Be neat
- Be concise

3. Definitions, Theorems, and Proofs

How to find proofs?

- Step 0: finding proofs *isn't always easy*
- Step 1: *carefully* read the statement you want to prove
- Step 2: when you want to prove a statement or part thereof, try to get an *intuitive*, *"gut" feeling* of why it should be true.
- Step 3: when you believe that you have found the proof, you *must* write it up *properly*.

Advices:

- Be patient
- Come back to it
  - think about it a bit, leave it, and then *return* a few minutes or *hours* later
- Be neat
- Be concise

3. Definitions, Theorems, and Proofs

Types of proofs:

- Proof by *Construction*
  - by demonstrating how to construct the object
  - e.g., given $A, A \to B, B \to C$, prove: C
- Proof by *Contradiction*
  - We assume that the theorem is false and then show that this assumption leads to an obviously false consequence, called a *contradiction*.
  - e.g., given $\neg A, B \to A$, prove $\neg B$
- Proof by *Induction*
  - Every proof by induction consists of two parts, the *basis* and the *induction* step.
    - basis step: prove $P(1)$ is true
    - induction step: prove $P(n) \to P(n+1)$
    - $P(i)$ is called *induction hypothesis*
  - e.g., prove $1 + 2 + \cdots + n = \frac{n(n+1)}{2}$

3. Definitions, Theorems, and Proofs

Types of proofs:

- Proof by *Construction*
  - by demonstrating how to construct the object
  - e.g., given $A, A \to B, B \to C$, prove: C
- Proof by *Contradiction*
  - We assume that the theorem is false and then show that this assumption leads to an obviously false consequence, called a *contradiction*.
  - e.g., given $\neg A, B \to A$, prove $\neg B$
- Proof by *Induction*
  - Every proof by induction consists of two parts, the *basis* and the *induction* step.
    - basis step: prove $P(1)$ is true
    - induction step: prove $P(n) \to P(n+1)$
    - $P(i)$ is called *induction hypothesis*
  - e.g., prove $1 + 2 + \cdots + n = \frac{n(n+1)}{2}$

3. Definitions, Theorems, and Proofs

Types of proofs:

- Proof by *Construction*
  - by demonstrating how to construct the object
  - e.g., given $A, A \to B, B \to C$, prove: C
- Proof by *Contradiction*
  - We assume that the theorem is false and then show that this assumption leads to an obviously false consequence, called a *contradiction*.
  - e.g., given $\neg A, B \to A$, prove $\neg B$
- Proof by *Induction*
  - Every proof by induction consists of two parts, the *basis* and the *induction* step.
    - basis step: prove $P(1)$ is true
    - induction step: prove $P(n) \to P(n+1)$
    - $P(i)$ is called *induction hypothesis*
  - e.g., prove $1 + 2 + \cdots + n = \frac{n(n+1)}{2}$

作业

0.2 Write formal descriptions of the following sets.

a. The set containing the numbers 1, 10, and 100
b. The set containing all integers that are greater than 5
c. The set containing all natural numbers that are less than 5
d. The set containing the string `aba`
e. The set containing the empty string
f. The set containing nothing at all

作业

0.3 Let A be the set $\{\mathtt{x}, \mathtt{y}, \mathtt{z}\}$ and B be the set $\{\mathtt{x}, \mathtt{y}\}$.

a. Is A a subset of B?

b. Is B a subset of A?

c. What is $A \cup B$?

d. What is $A \cap B$?

e. What is $A \times B$?

f. What is the power set of B?

作业

0.7 For each part, give a relation that satisfies the condition.

a. Reflexive and symmetric but not transitive

b. Reflexive and transitive but not symmetric

c. Symmetric and transitive but not reflexive

0.8 Consider the undirected graph $G = (V, E)$ where V, the set of nodes, is $\{1, 2, 3, 4\}$ and E, the set of edges, is $\{\{1, 2\}, \{2, 3\}, \{1, 3\}, \{2, 4\}, \{1, 4\}\}$. Draw the graph G. What are the degrees of each node? Indicate a path from node 3 to node 4 on your drawing of G.