离散数学

目录

第1章 逻辑

1.1 命题

命题（Proposition）

逻辑（logic）规则给出数学语句的准确含义，这些规则用来区分有效和无效的数学论证。逻辑不仅对理解数学推理十分重要，而且在计算机科学中有许多应用，逻辑可用于电路设计、程序构造、程序正确性证明等方面。

命题是逻辑的基本成分，一个命题是一个具有真值（truth value）的语句，命题可以为真也可以为假，但不能既为真又为假。

命题	**非命题**
I have a dog.	What day is today?
1 + 2 = 3	Shut the door!
Today is Wednesday.	1 + 2
It is snowing today.	x + 1 = 2

命题习惯上用字母p, q, r, s等来表示，如果一个命题是真命题，它的真值为真，用`T`表示；如果一个命题是假命题，它的真值为假，用`F`表示。

非运算符（NOT, Negation Operator）

非运算符¬只作用于一个命题，其作用是反转命题的真值。

真值表（truth table）可以给出命题真值之间的关系，在确定由简单命题组成的命题的真值时，真值表特别有用。

p	$\neg p$
T	F
F	T

【范例】$\neg p$

p：It snowed last night.

$q : 2 + 3 = 6$

$\neg p$：It didn;t snow last night.

$\neg q : 2+3 \neq 6$

合取运算符（AND, Conjunction Operator）

命题$p \wedge q$表示“p并且q”，当p和q都为真时命题为真，否则为假。

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

【范例】$p \wedge q$

p：今天是星期五。

q：今天会下雨。

$p \wedge q$：今天是星期五并且会下雨。

析取运算符（OR, Disjunction Operator）

命题$p \vee q$表示“p或q”，当p和q都为假时命题为假，否则为真。

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

【范例】$p \vee q$

p：开关坏了。

q：灯泡坏了。

$p \vee q$：开关坏了或者灯泡坏了。

异或运算符（XOR, Exclusive Or）

命题$p \oplus q$表示“p和q的异或”，当p和q中恰有一个为真时命题为真，否则为假。

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

【范例】$p \oplus q$

p：他现在在上海。

q：他现在在北京。

$p \oplus q$：他现在在上海或北京。

【代码】某地发生了一件谋杀案，警察通过排查确定杀人凶手必为4个嫌疑犯的一个，根据以下信息确定凶手。

> A说：不是我。
>
> B说：是C。
>
> C说：是D。
>
> D说：C在胡说。
>
> 已知3个人说了真话，1个人说的是假话。

```python
def main():
    for killer in ['A', 'B', 'C', 'D']:
        if (killer != 'A') + (killer == 'C') \
            + (killer == 'D') + (killer != 'D') == 3:
            print(killer)

if __name__ == "__main__":
    main()
```

> 运行结果

```
C
```

1.2 复合命题

复合命题（Compound Proposition）

使用非运算符和已定义的各联结词可以构造复合命题。小括号用于规定复合命题中多个逻辑运算符的操作顺序，为了减少所需的小括号数量，规定了各联结词的优先级。

优先级	运算符
1	$\neg$
2	$\wedge/\vee$
3	$\rightarrow$ / $\leftrightarrow$

蕴含运算符（Implication Operator）

命题$p \rightarrow q$表示“p蕴含q”，在p为真而q为假时命题为假，否则为真。其中p称为前提，q称为结论。

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

表示$p \rightarrow q$的术语有很多种，常见的有：

- If p, then q.
- p only if q.
- q is necessary for p.

【范例】$p \rightarrow q$

p：我去看电影。

q：我买奶茶。

$p \rightarrow q$：如果我去看电影，那么我会买奶茶。

由$p \rightarrow q$可以构造出几个相关的蕴含：

1. $q \rightarrow p$称为$p \rightarrow q$的逆命题（converse）。
2. $\neg q \rightarrow \neg p$称为$p \rightarrow q$的逆否命题（contrapositive）。

【范例】逆命题与逆否命题

p：今天是星期四。

q：我今天有考试。

$p \rightarrow q$：如果今天是星期四，那么我今天有考试。

$q \rightarrow p$：如果我今天有考试，那么果今天是星期四。

$\neg q \rightarrow \neg p$：如果我今天没有考试，那么今天不是星期四。

双向蕴含（Biconditional Operation）

命题$p \leftrightarrow q$表示“p双向蕴含q”，在p和q有相同的真值时命题为真，否则为假。

$p \leftrightarrow q$等价于$(p \rightarrow q) \wedge (q \rightarrow p)$。

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

双蕴含的真值表与异或的真值表正好相反，因此$p \leftrightarrow q$与$\neg(p \oplus q)$等价。

1.3 逻辑等价

逻辑等价（Logical Equivalence）

两个不同的复合命题可能拥有完全相同的真值，则称这两个命题在逻辑上是等价的。如果无论复合命题中各个命题的真值是什么，复合命题的真值总是为真，这样的复合命题称为永真式（tautology）。如果真值永远为假的复合命题称为矛盾（contradiction）。

p	$\neg p$	$p \vee \neg p$	$p \wedge \neg p$
T	F	T	F
F	T	T	F

如果复合命题s和是r逻辑等价的，可表示为$s \equiv r$。只有当$s \leftrightarrow r$是永真式时，s和r才是逻辑等价的。

【范例】使用真值表证明$p \vee q \equiv \neg(\neg p \wedge \neg q)$

p	q	$p \vee q$	$\neg p$	$\neg q$	$\neg p \wedge \neg q$	$\neg(\neg p \wedge \neg q)$
T	T	T	F	F	F	T
T	F	T	F	T	F	T
F	T	T	T	F	F	T
F	F	F	T	T	T	F

逻辑等价定理

定理	等价关系
幂等律（Idempotent Laws）	$p \wedge p \equiv p$ $p \vee p \equiv p$
恒等律（Identity Laws）	$p \wedge T \equiv p$ $p \vee F \equiv p$
支配律（Domination Laws）	$p \vee T \equiv T$ $p \wedge F \equiv F$
双非律（Double Negation Law）	$\neg(\neg p) == \equiv p$
交换律（Commutative Laws）	$p \wedge q \equiv q \wedge p$ $p \vee q \equiv q \vee p$
结合律（Associative Laws）	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$ $(p \vee q) \vee r \equiv p \vee (q \vee r)$
分配率（Distributive Laws）	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$ $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
德摩根律（De Morgan's Laws）	$\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\neg(p \vee q) \equiv \neg p \wedge \neg q$
吸收律（Absorption Laws）	$p \wedge (p \vee q) \equiv p$ $p \vee (p \wedge q) \equiv p$
条件恒等	$p \rightarrow q \equiv \neg p \vee q$ $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

【范例】证明$(p \vee q) \rightarrow p$永真

$$
\begin{aligned}
&(p \vee q) \rightarrow p \\
&\equiv \neg(p \wedge q) \vee p \\
&\equiv (\neg p \vee \neg q) \vee p \\
&\equiv (\neg q \vee \neg p) \vee p \\
&\equiv \neg q \vee (\neg p \vee p) \\
&\equiv \neg q \vee T \\
&\equiv T
\end{aligned}
$$

1.4 谓词和量词

谓词（Predicate）

命题逻辑并不能表达数学语言和自然语言中所有语句的确切含义。在数学表达式和计算机程序中经常可以看到含有变量的语句，例如“$x > 3$”、“$x = y + 3$”、“程序x正在运行”等。当变量值未指定时，这些语句既不为真也不为假。

利用$P(x)$可以表示语句，其中x是变量，语句$P(x)$可以说是命题函数P在x的值。一旦给变量x赋一个值，语句$P(x)$就称为命题并具有真值。

通常使用大写字母P, Q, R等表示谓词，小写字母x, y, z等表示变量。

【范例】谓词

谓词	真值
$P(x): x + 3 = 6$	$P(3)$为True
$Q(x, y): x = y + 2$	$Q(4, 1)$为False

量词（Quantifier）

量词用量化表示在何种程度上谓词对于一定范围的个体成立。

量词有全称量词（universal quantifier）和存在量词（existential quantifer）：

1. 全称量词$\forall$表示”all“。$\forall_x P(x)$是一个命题，当范围内所有的x都能使语句$P(x)$为真时，命题为真。

$$\forall_x P(x) = P(a_1) \wedge P(a_2) \wedge \cdots \wedge P(a_k)$$

【范例】全称量词

假设x表示“全班所有学生”，$P(x)$表示“x完成了作业”。

$\forall_x P(x)$：全班所有学生都完成了作业。

---

2. 存在量词$\exists$表示“exists”。$\exists_x P(x)$是一个命题，当范围内存在至少一个x能够语句$P(x)$为真时，命题为真。

$$\exists_x P(x) = P(a_1) \vee P(a_2) \vee \cdots \vee P(a_k)$$

【范例】存在量词

假设x表示“全班所有学生”，$P(x)$表示“x完成了作业”。

$\exists_x P(x)$：班里存在有一个学生完成了作业。

【范例】嵌套量词

假设x表示“某个人”，$P(x)$表示“x有父母”。

$\forall_x P(x)$：所有人都有父母。

$\exists_x \neg P(x)$：存在至少有一个人没有父母。

$\exists_x \exists_y (P(x) \wedge P(y))$：至少存在一个人$x$和一个人$y$有父母。

【范例】$P(x)$：x是偶数，$Q(x)$：x能被3整除，$x \in \mathbb{Z}^+$

语句	真值
$\exists_x (P(x) \wedge Q(x))$	True
$\forall_x (P(x) \rightarrow \neg Q(x))$	False

全称量词的否定

否定运算符可以使用在全称量词上。

$$\neg \forall_x P(x) \equiv \exists_x \neg P(x)$$
$$\neg \exists_x P(x) \equiv \forall_x \neg P(x)$$

【范例】全称量词的否定

$P(x)$: x will pass the course (x is a student).

$\neg \forall_x P(x)$: Not all students will pass the course.

$\forall_x \neg P(x)$: No student will pass the course.

$\neg \exists_x P(x)$: There does not exist a student that will pass the course.

$\exists_x \neg P(x)$: There exists a student that will not pass the course.

1.5 证明

证明（Proof）

证明方法非常重要，不仅因为它们可用于证明数学定理，而且在计算机科学中也有许多应用，包括验证程序正确性、建立安全的操作系统、人工智能领域做推论等。证明就是建立定理真实性的有效论证。

证明定理有很多方法：

1. 直接证明法（direct proof）

【范例】证明定理：如果n是奇数，那么n^2也是奇数，$n \in \mathbb{Z}$。

$$\forall_x (P(n) \to Q(n))$$

$$\begin{aligned} n^2 &= (2k+1)^2 \\ &= 4k^2 + 4k + 1 \\ &= 2(2k^2 + 2k) + 1 \end{aligned}$$

---

2. 反证法（proof by contrapositive）：由于$p \to q \equiv \neg q \to \neg p$，因此可以通过证明原命题的逆否命题来反证原命题。

【范例】证明定理：如果xy是偶数，那么x是偶数或y是偶数，$x, y \in \mathbb{Z}$。

逆否命题：如果x是奇数并且y是奇数，那么xy是奇数，$x, y \in \mathbb{Z}$。

$$\begin{aligned} xy &= (2m+1)(2n+1) \\ &= 4mn + 2m + 2n + 1 \\ &= 2(2mn + m + n) + 1 \end{aligned}$$

1.6 布尔代数

布尔代数（Boolean Algebra）

计算机和其它电子设备中的电路都有输入和输出，输入是`0`或`1`，输出也是`0`或`1`。电路可以用任何具有两个不同状态的基本元件来构造，例如开关和光学装置就是这样的原件，开关可位于开或关的位置，光学装置可能是点亮或未点亮。18世纪，乔治·布尔（George Boole）给出了逻辑的基本规则。

电路的操作可以用布尔函数来定义，这样的布尔函数对任意一组输入都能指出其输出的值。

布尔代数提供的是集合$\{0,1\}$上的运算和规则，布尔代数的规则类似于命题逻辑的规则。`1`相当于逻辑中的`真`，`0`相当于逻辑中的`假`。

布尔代码运算主要有三种：

1. 补（complement）

x	$\overline{x}$
1	0
0	1

2. 布尔积（boolean multiplication）

x	y	$x \cdot y$
1	1	1
1	0	0
0	1	0
0	0	0

3. 布尔和（boolean addition）

x	y	$x+y$
1	1	1
1	0	1
0	1	1
0	0	0

【范例】布尔表达式

当$x=y=1$，$w=z=0$时，

$x\cdot y+(w+z)=1\cdot 1+(0+0)=1+0=1$

$x\cdot\overline{y}+z\cdot\overline{(w+z)}=1\cdot\overline{1}+0\cdot\overline{(0+0)}=0+0=0$

$x\cdot\overline{y}+\overline{(\overline{x}+y+\overline{yz})}=1\cdot\overline{1}+\overline{(\overline{1}+1+\overline{1\cdot 0})}=0+\overline{1}=0$

布尔代数定理

定理	等价关系
幂等律（Idempotent Laws）	$x \cdot x = 0$ $x + x = x$
恒等律（Identity Laws）	$x \cdot 1 = x$ $x + 0 = x$
支配律（Domination Laws）	$x \cdot 0 = 0$ $x + 1 = 1$
双非律（Double Negation Law）	$\overline{\overline{x}} = x$
交换律（Commutative Laws）	$x \cdot y = y \cdot x$ $x + y = y + x$
结合律（Associative Laws）	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$ $(x + y) + z = x + (y + z)$
分配率（Distributive Laws）	$x \cdot (y + z) = x \cdot y + x \cdot z$ $x + (y \cdot z) = (x + y) \cdot (x + z)$
德摩根律（De Morgan's Laws）	$\overline{x \cdot y} = \overline{x} + \overline{y}$ $\overline{x + y} = \overline{x} \cdot \overline{y}$
吸收律（Absorption Laws）	$x \cdot (x + y) = x$ $x + (x \cdot y) = x$

【范例】证明$xy + x\overline{y} = x$

$$\begin{aligned}
& xy + x\overline{y} \\
& = x \cdot (y + \overline{y}) \\
& = x \cdot 1 \\
& = x
\end{aligned}$$

布尔函数（Boolean Function）

含有n个变量的布尔函数能够构造出2^n行的输入输出表。

【范例】计算$F(x, y, z) = xy + \overline{z}$

x	y	z	$F(x,y,z)$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

NAND和AND运算符

`NAND`运算符用↑表示`Not And`，$x \uparrow y = \overline{x \cdot y}$。

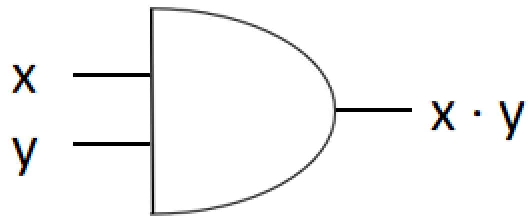
`NOR`运算符用↓表示`Not Or`，$x \downarrow y = \overline{x + y}$。

1.7 逻辑门电路

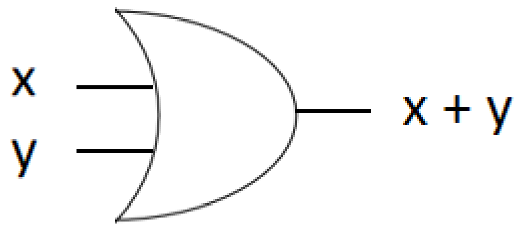
逻辑门电路（Logical Gate Circuit）

基础的逻辑门主要有三种：

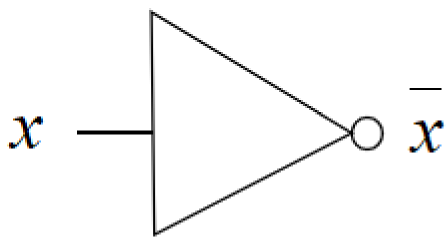
1. 与门（AND gate）：



2. 或门（OR gate）：

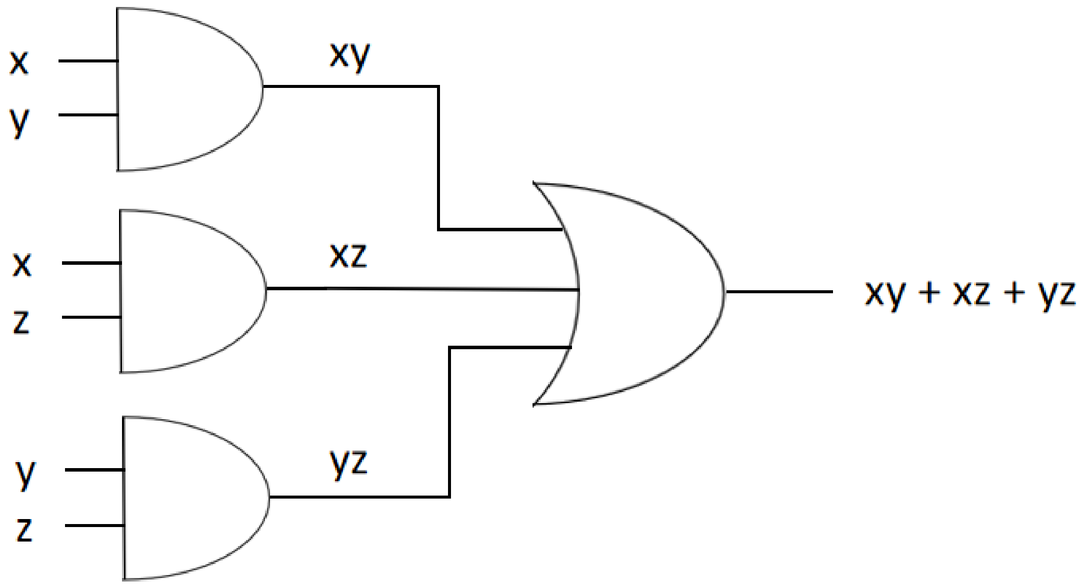


3. 非门（NOT gate）：



【范例】设计一个投票表决电路，三个人中有两人赞成即通过。赞成票为1，否决表为0。

$$F(x, y, z) = xy + xz + yz$$


x
y
xy
x
z
xz
y
z
yz
xy + xz + yz

第2章 集合

2.1 集合

集合（Set）

集合是对象的唯一的、无序的聚集，通常一个集合中的对象都具有相似的性质。对象也称为集合的元素（element）或成员（member）。

通常用大写字母表示集合，小写字母表示元素。$a \in A$表示是a集合A中的元素，$a \notin A$表示a不是集合A中的元素。

使用花名册方法（roster method）列出集合中的元素，可以用于描述集合。

【范例】花名册方法

小写元音字母集合$V = \{a, e, i, o, u\}$

小于10的正奇数集合$O = \{1, 3, 5, 7, 9\}$

小于100的非负整数集合$A = \{0, 1, 2, 3, \ldots, 99\}$

集合构造器（set builder）通过描述元素具有的形式来描述集合。

【范例】集合构造器

小于10的正整数$A = \{x | x < 10\}$

有一些常用的特殊符号可用于描述指定的集合：

符号	含义
$\mathbb{N}$	自然数集$\{0, 1, 2, 3, \ldots\}$
$\mathbb{Z}$	整数集$\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
$\mathbb{Z}^+$	正整数集$\{1, 2, 3, \ldots\}$
$\mathbb{Q}$	有理数集$\{p/q \mid p \in \mathbb{Z},\ q \in Z\ (q \neq 0)\}$
$\mathbb{Q}^+$	正有理数集
$\mathbb{R}$	实数集
$\mathbb{R}^+$	正实数集
$\mathbb{C}$	复数集
$\emptyset$	空集$\{\}$

基数（Cardinality）

基数表示有限集合中元素的个数，集合A的基数记为$|A|$。

【范例】基数

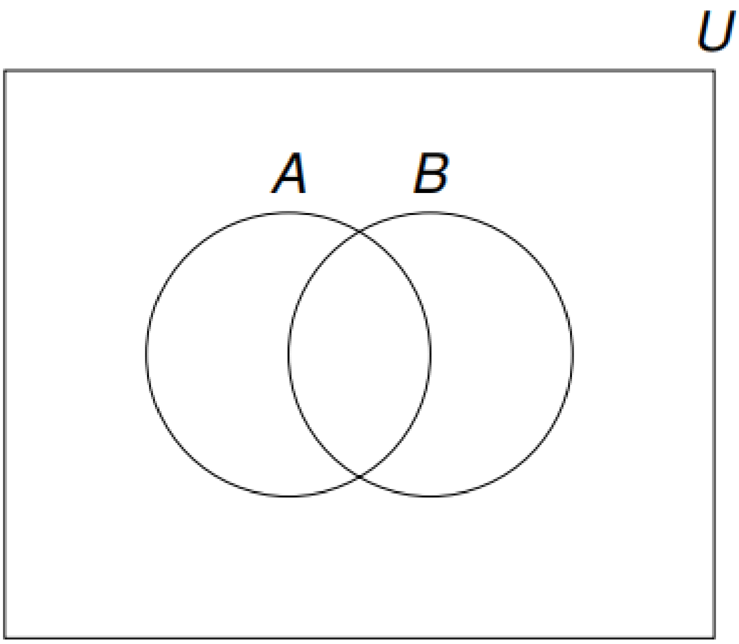
英语字母集合A，$|A| = 26$

空集$\emptyset$，$|\emptyset| = 0$

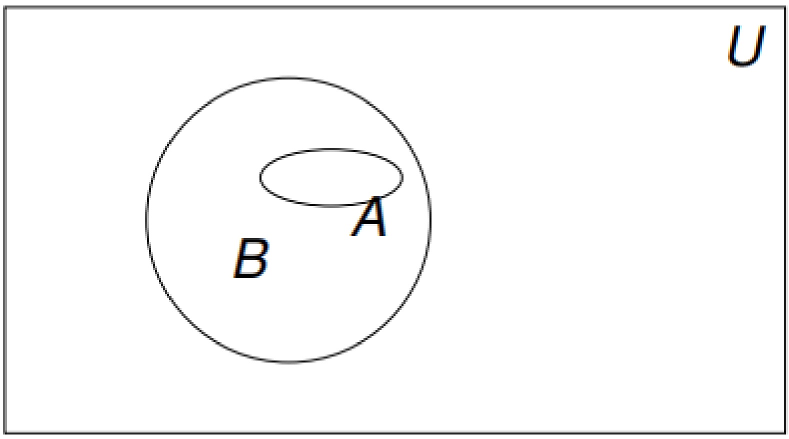
韦恩图（Venn Diagram）

集合还可以使用韦恩图来表示。

全集（universal set）包含所研究问题中所有的元素，用符号$\bigcup$表示。



假设有两个集合A和B，如果A中的所有元素都在B中，那么A就是B的子集，表示为$A \subseteq B$。如果A中有一个元素不在B中，那么A就不是B的子集，表示为$A \not\subseteq B$
。



只有当两个集合互相为对方的子集时，那么这两个集合相等，即：

$$A = B \text{ iff } A \subseteq B \ and \ B \subseteq A$$

如果$A \subseteq B$，并且B中有一个元素不是A的元素，那么称A是B的真子集（proper subset），表示为$A \subset B$。

幂集（Power Set）

一个集合中是可以包含另一个集合的，如$\{\{1\}.\{1,2\},\{1,2,3\}\}$。需要注意，
$1 \neq \{1\} \neq \{\{1\}\}$。

幂集用于表示一个集合所有子集的集合，集合A的幂集表示为$P(A)$。

【范例】计算$A=\{1,2,3\}$的幂集

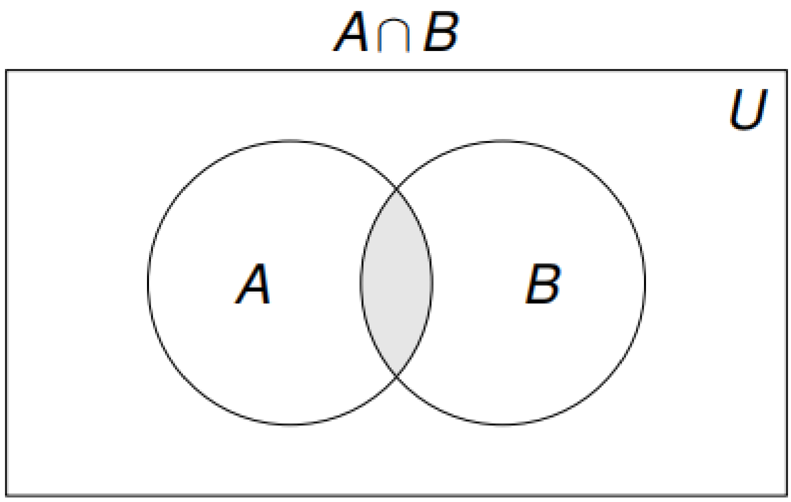
$$P(A)=\{\emptyset,\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$$

如果集合A的基数为n，那么A的幂集的基数为2^n，即$|P(A)|=2^n$。

2.2 集合运算

交集（Intersection）

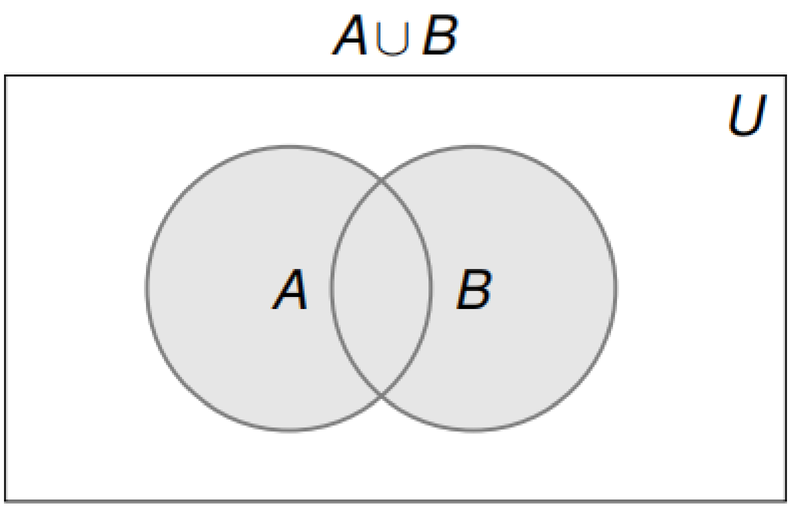
假设A和B是两个集合，由所有属于A并且属于B的元素所组成的集合，称为A与B的交集，表示为$A \cap B$。



如果两个集合没有公共元素，那么它们的交集为空集。

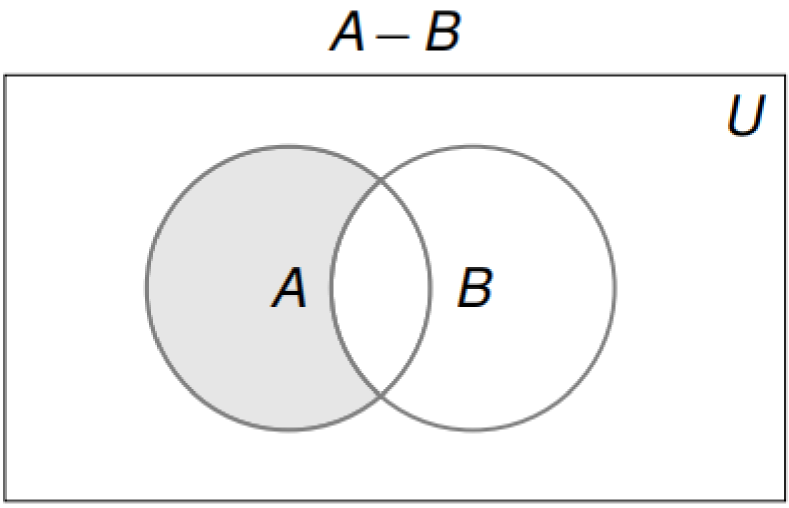
并集（Union）

假设A和B是两个集合，由它们所有元素合并在一起组成的集合，称为A与B的并集，表示为$A \cup B$。



差集（Difference）

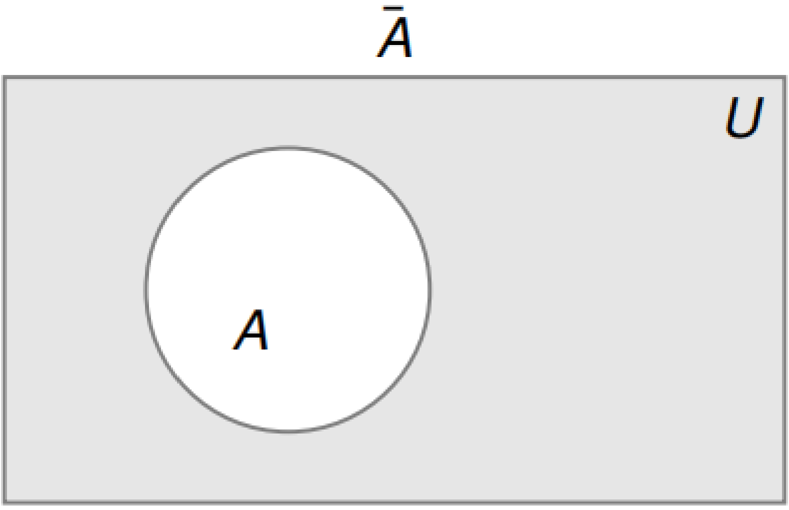
假设A和B是两个集合，由属于A而不属于B的元素组成的集合，称为A与B的差集，表示为$A-B$。



差集运算不满足交换律，即$A-B\neq B-A$。

补集（Complement）

假设A是一个集合，由全集∪中所有不属于A的元素组成的集合，称为A的补集，表示为$\overline{A}$。



2.3 集合恒等式

集合恒等式

集合恒等式可以直接由对应的逻辑等价式证明。

定理	等价关系
幂等律（Idempotent Laws）	$A \cap A = A$ $A \cup A = A$
恒等律（Identity Laws）	$A \cap U = A$ $A \cup \emptyset = A$
支配律（Domination Laws）	$A \cap \emptyset = \emptyset$ $A \cup U = U$
双非律（Double Negation Law）	$\overline{\overline{A}} = A$
交换律（Commutative Laws）	$A \cap B = B \cap A$ $A \cup B = B \cup A$
结合律（Associative Laws）	$(A \cap B) \cap C = A \cap (B \cap C)$ $(A \cup B) \cup C = A \cup (B \cup C)$
分配率（Distributive Laws）	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
德摩根律（De Morgan's Laws）	$\overline{A \cap B} = \overline{A} \cup \overline{B}$
吸收律（Absorption Laws）	$A \cap (A \cup B) = A$ $A \cup (A \cap B) = A$

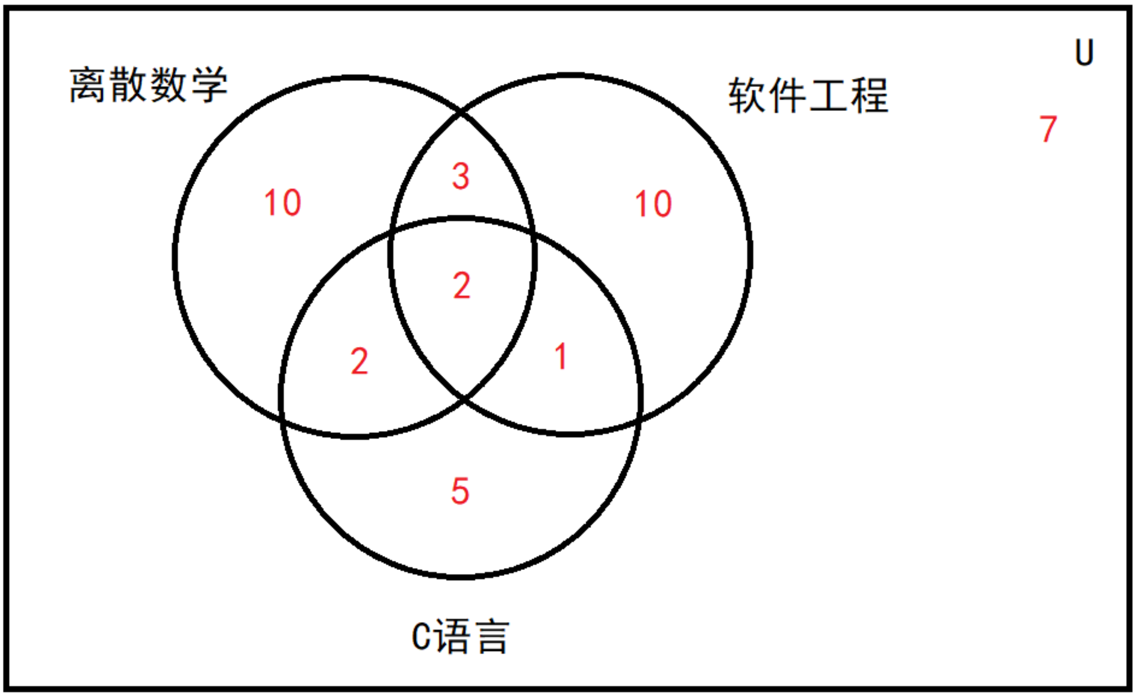
【范例】证明$\overline{A \cup (B \cap C)} = (\overline{C} \cup \overline{B}) \cap \overline{A}$

$$
\begin{aligned}
&\overline{A \cup (B \cap C)} \\
&= \overline{A} \cap \overline{B \cap C} \\
&= \overline{A} \cap (\overline{B} \cup \overline{C}) \\
&= (\overline{B} \cup \overline{C}) \cap \overline{A} \\
&= (\overline{C} \cup \overline{B}) \cap \overline{A}
\end{aligned}
$$

【范例】韦恩图

> 一共有40个学生，有3门课程可供学生选择（C语言、离散数学、软件工程）。

> 7人没有选任何课程；
>
> 16人选软件工程；
>
> 10人选C语言；
>
> 5人同时选离散数学和软件工程；
>
> 4人同时选离散数学和C语言；
>
> 3人同时选软件工程和C语言；
>
> 2人同时选离散数学、软件工程和C语言。



2.4 笛卡尔积

元组（Tuple）

有时候元素聚集中次序是很重要的，由于集合是无序的，所以就需要一种不同的结构表示有序的聚集，这就是有序n元组（ordered-n-tuple）。

有序n元组$(a_1, a_2, \ldots, a_n)$是以a_1为第1个元素，a_2为第2个元素，a_n为第n个元素的有序聚集。

只有两个有序n元组的每一对对应的元素都相等，那么这两个有序n元组是相等的，即：

$$(a_1, a_2, \ldots, a_n) = (b_1, b_2, \ldots, b_n) \text{ iff } i = 1, 2, \ldots, n$$

需要注意，(a, b)与(b, a)不相等，除非$a = b$。

笛卡尔积（Cartesian Product）

假设有两个集合A和B，A和B的笛卡尔积用$A \times B$表示，笛卡尔积是所有序偶(a, b)的集合，其中$a \in A$且$b \in B$。

$$A \times B = \{(a, b) | a \in A \land b \in B\}$$

笛卡尔积$A \times B$和$B \times A$是不相等的，除非$A = \emptyset$或$B = \emptyset$或$A = B$。

【范例】笛卡尔积

学生集合$S = \{s1, s2\}$

课程集合$C = \{c1, c2, c3\}$

$S \times C = \{(s1, c1), (s1, c2), (s1, c3), (s2, c1), (s2, c2), (s2, c3)\}$

笛卡尔积$S \times C$表示学生选课的所有可能情况

【范例】笛卡尔积$A \times B \times C$

$A = \{0, 1\}$

$B = \{1, 2\}$

$C = \{0, 1, 2\}$

$$\begin{aligned} A \times B \times C = \{&(0,1,0), (0,1,1), (0,1,2), (0,2,0), (0,2,1), (0,2,2), \\ &(1,1,0), (1,1,1), (1,1,2), (1,2,0), (1,2,1), (1,2,2)\} \end{aligned}$$

一个集合与自身的笛卡尔积，如$A \times A$可表示为A^2。

【范例】笛卡尔积A^2

$A = \{1, 2\}$

$A^2 = \{(1,1), (1,2), (2,1), (2,2)\}$

第3章 函数

3.1 函数

函数（Function）

函数在数学和计算机科学中的概念非常重要，在离散数学中函数用于定义像序列和字符串这样的离散结构。

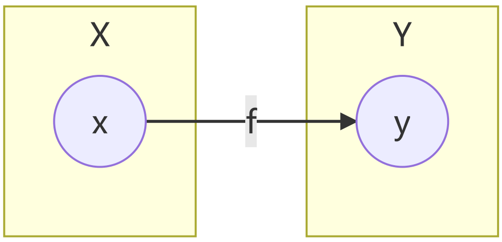
利用一个函数f，可以将一个值$x \in \mathbb{R}$映射（mapping）到一个特定的值$y = f(x)$上（$y \in \mathbb{R}$)。

假设有两个非空集合X和Y，从X到Y的函数f是指对于X的每个元素恰好都对应Y的一个元素（$f(x) = y,\ x \in X,\ y \in Y$），那么就写成$f : X \rightarrow Y$。

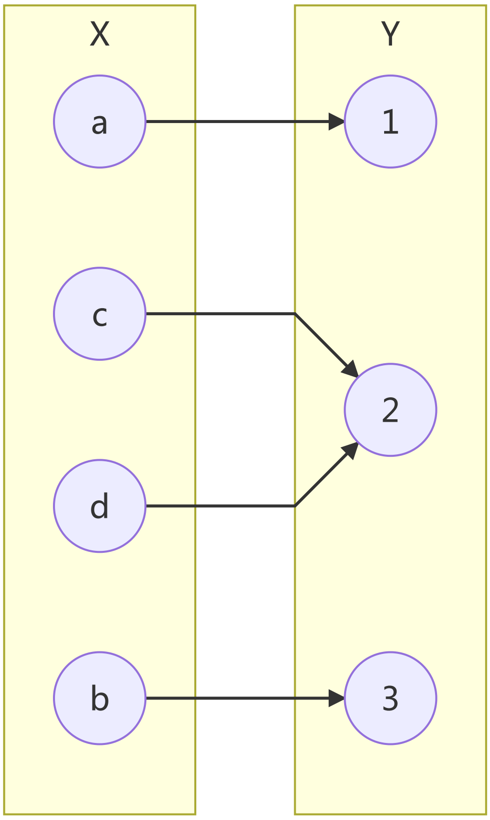
集合X被称为函数f的定义域（domain），集合Y被称为函数f的陪域（co-domain）。

如果$f(x) = y$，那么y是x在函数f下的像（image），x是y在函数f下的原像（pre-image）。函数f的值域（range）是集合X中所有像的集合。

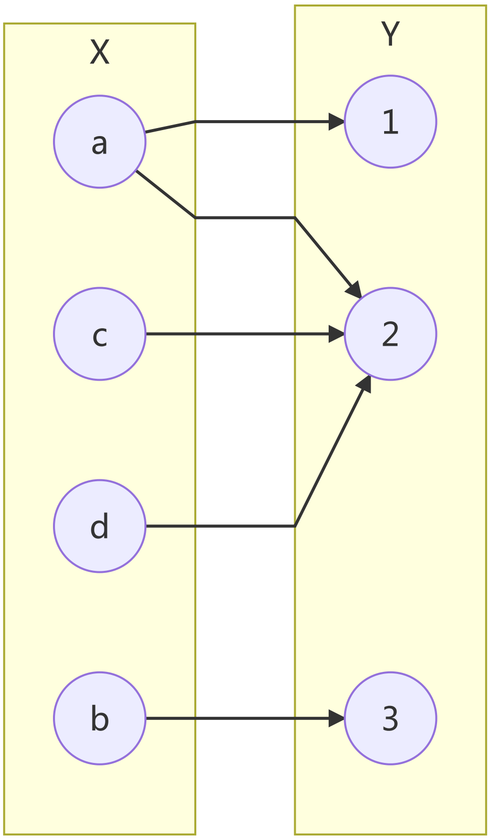
当两个函数f和g有相同的定义域和陪域，并且对于定义域中所有元素x都满足$f(x) = g(x)$，那么函数f和g相等，表示为$f = g$。



【范例】判断是否为函数

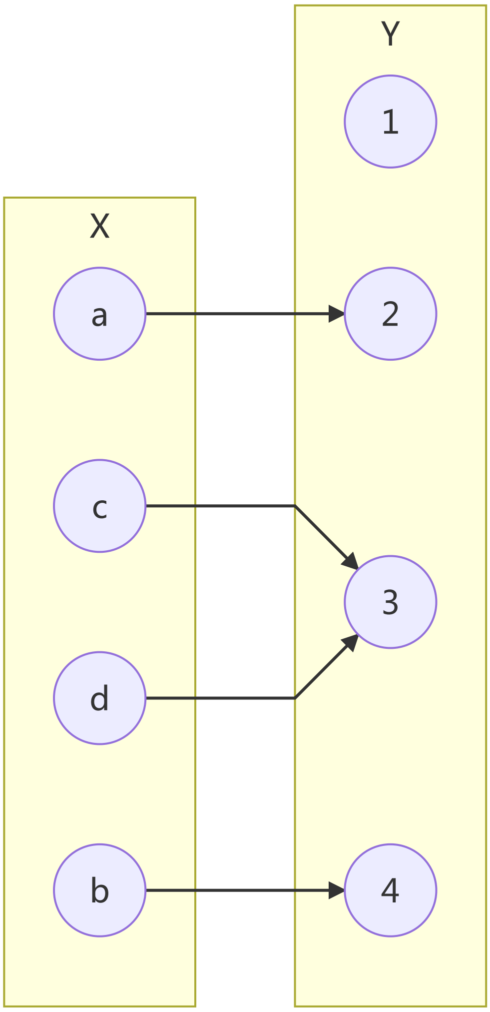


是否为函数：是



是否为函数：否

【范例】函数



是否为函数	定义域（domain）	陪域（co-domain）	值域（range）
是	a, b, c, d	$1, 2, 3, 4$	$2, 3, 4$

3.2 取整函数

上取整函数（Ceiling Function）

取整函数包括上取整和下取整，可以将实数映射到整数（$\mathbb{R} \to \mathbb{Z}$)，它们以不同的方式将实数近似到相邻的整数。

上取整函数将实数x向上取到大于或等于x的最小整数，表示为$\lceil x \rceil$。

【范例】上取整函数

$\lceil 3.2 \rceil = 4$

$\lceil 2.6 \rceil = 3$

$\lceil -0.5 \rceil = 0$

下取整函数（Floor Function）

下取整函数将实数x向下取到小于或等于x的最大整数，表示为$\lfloor x \rfloor$。

【范例】下取整函数

$\lfloor 3.2 \rfloor = 3$

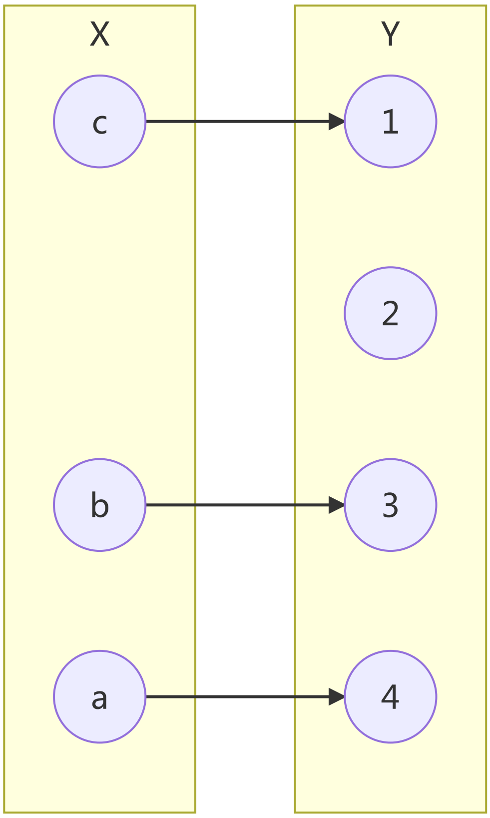
$\lfloor 5.9 \rfloor = 5$

$\lfloor -0.5 \rfloor = -1$

3.3 函数分类

一对一函数（One-to-one）/ 单射函数（Injection）

一对一函数 / 单射函数是指对于函数f的定义域中所有的a和b，如果$a \neq b$，那么$f(a) \neq f(b)$。



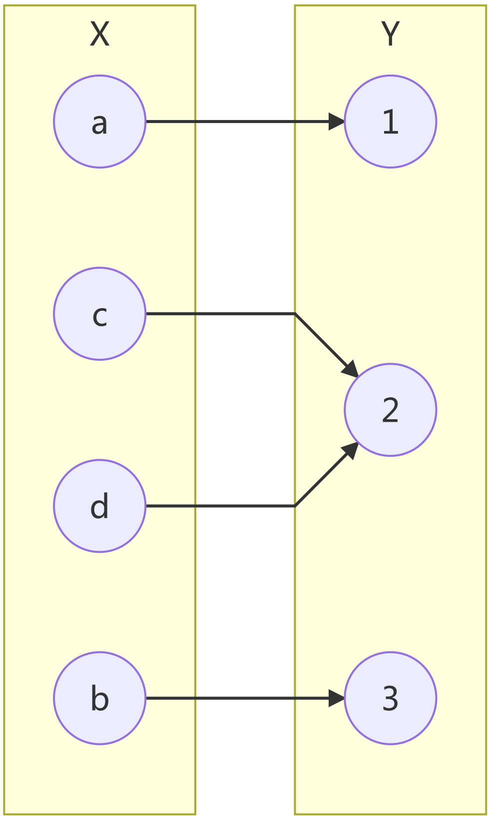
【范例】一对一函数 / 单射函数

$f(x) = x + 1$是一对一函数。

$f(x) = x^2$不是一对一函数，因为$f(1) = f(-1) = 1$。

映上函数（Onto）/ 满射函数（Surjection）

映上函数 / 满射函数是指对于函数$f : A \to B$，每个$b \in B$都有元素$a \in A$使得$f(a) = b$。



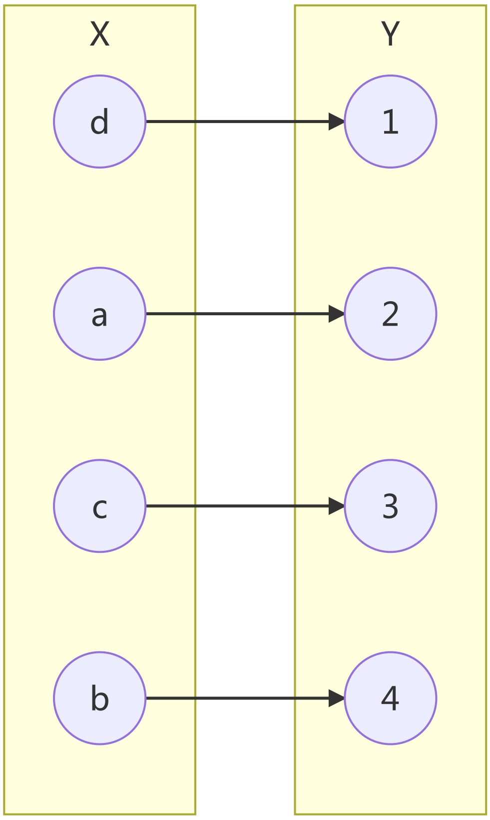
【范例】映上函数 / 满射函数

$f: \mathbb{Z} \to \mathbb{Z},\ f(x) = x + 1$是映上函数。

$f: \mathbb{Z} \to \mathbb{Z},\ f(x) = x^2$不是映上函数，因为没有整数$x$使$x^2 = -1$。

一一对应函数（One-to-One Correspondance）/ 双射函数（Bijection）

如果一个函数既是一对一函数又是映上函数，那么这个函数就被称为一一对应函数 / 双射函数。



【范例】——对应函数 / 双射函数

f是从$\{a, b, c, d\}$到$\{1, 2, 3, 4\}$的函数，定义
$f(a) = 4,\ f(b) = 2,\ f(c) = 1,\ f(d) = 3$。

函数f是单射函数，因为没有两个值映射到相同的函数值。

函数f是满射函数，因为陪域的个数与值域的个数相同。

因此，函数f是双射函数。

3.4 反函数

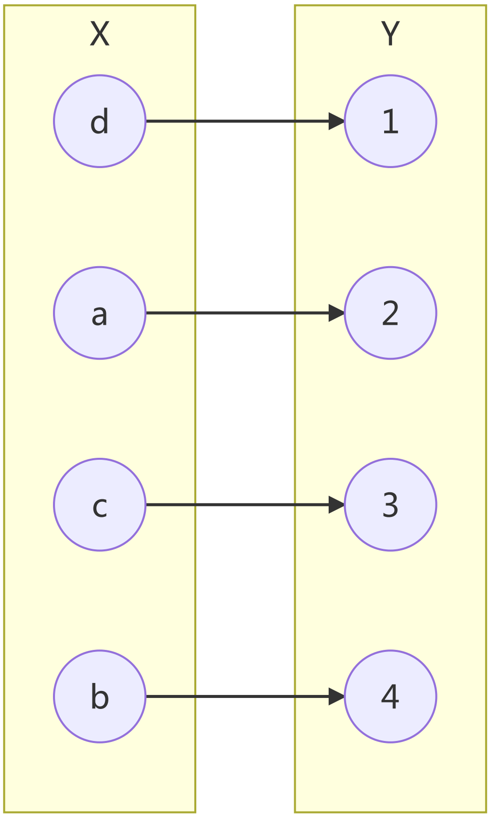
反函数（Inverse Function）

假设有一个从集合A到集合B的双射函数f。由于f是满射函数，所以B中的每个元素都是A中某些元素的像；又由于f还是单射函数，所以B的每个元素都是A中唯一一个元素的像。

于是，通过把f的对应关系颠倒，获得的从B到A的新函数被称为f的反函数，用f^{-1}表示。当$f(a) = b$时，$f^{-1}(b) = a$。

需要注意，不要将f^{-1}与$\frac{1}{f}$混淆。

【范例】反函数



是否有反函数	$f^{-1}(2)$	$f^{-1}(1)$
是	a	d

【范例】计算$f(x) = x + 3$的反函数

$f^{-1}(x) = x - 3$

3.5 合成函数

合成函数（Composition Function）

假设g是从集合A到集合B的函数，f是从集合B到集合C的函数。函数f和g的合成，记作$f \circ g$。

$$(f \circ g)(x) = f(g(x))$$

函数合成的顺序很重要，$f \circ g$与$g \circ f$并不相等。

【范例】合成函数

$f : \mathbb{R}^+ \rightarrow \mathbb{R}^+,\ f(x) = x^3$

$g : \mathbb{R}^+ \rightarrow \mathbb{R}^+,\ g(x) = x + 2$

$(f \circ g)(x) = f(g(x)) = (x + 2)^3$

$(g \circ f)(x) = g(f(x)) = x^3 + 2$

恒等函数（Identity Function）

如果一个从集合A到集合B的函数f有反函数，那么f与f^{-1}的合成函数得到的是恒等函数。

如果$f(a) = b$，那么$f^{-1}(b) = a$。

$$(f \circ f^{-1})(a) = f^{-1}(f(a)) = f^{-1}(b) = a$$

3.6 指数函数与对数函数

指数函数（Exponential Function）

指数函数的定义为$y = a^x \ (a > 0 | a \neq 1)$，其中$a$称为底数（base），$x$称为指数（exponent）。

$$
\begin{aligned}
a^m \cdot a^n &= a^{m+n} \\
\frac{a^m}{a^n} &= a^{m-n} \\
(a^m)^n &= a^{mn} \\
(ab)^m &= a^m \cdot b^m
\end{aligned}
$$

【范例】指数函数

$(6^{2k})^3 = 6^{6k}$

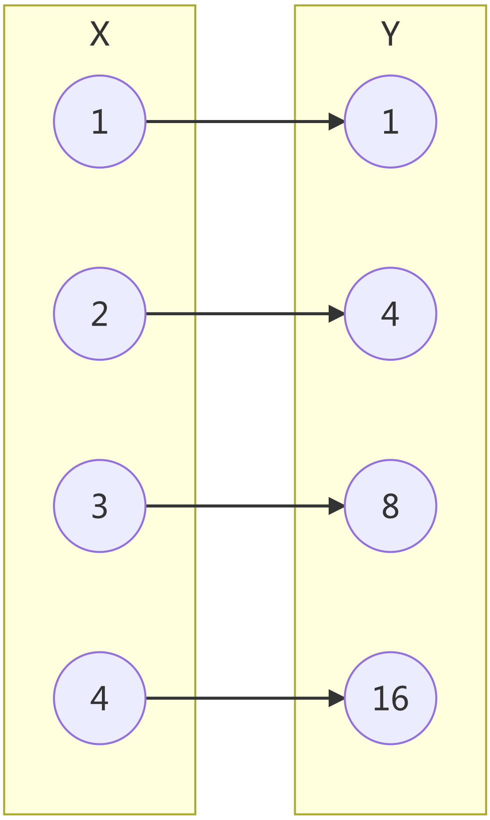
$6^{k^2} \times 6 = 6^{k^2+1}$

$\frac{3^k}{9} = \frac{3^k}{3^2} = 3^{k-2}$

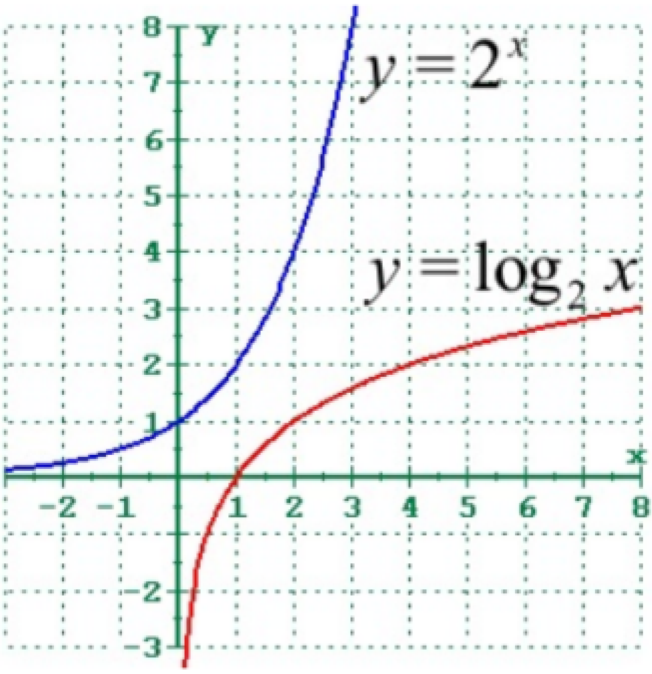
$3^k \times 27 = 3^k \times 3^3 = 3^{k+3}$

对数函数（Logarithm Function）

对于函数$f : \{1, 2, 3, 4\} \to \{1, 4, 8, 16\}, \ f(x) = 2^x$，指数函数是双射函数，因此它是有反函数的。



对数函数是指数函数的反函数，对数函数的定义为$y=log_ax\ (a>0|a\neq 1)$，其中a称为底数。



$$
\begin{aligned}
log_axy &= log_ax + log_ay \\
log_a\frac{x}{y} &= log_ax - log_ay \\
log_ax^y &= ylog_ax \\
log_ax &= \frac{log_bx}{log_ba}
\end{aligned}
$$

【范例】对数函数

$$log_5 k + log_5 2 = log_5 2k$$

$$log_2 5^2 = 2 \times log_2 5$$

$$\frac{log_3 k^2}{log_3 25} = \frac{2 \times log_3 k}{log_3 5^2} = \frac{2 \times log_3 k}{2 \times log_3 5} = log_5 k$$

第4章 数论

4.1 进制转换

进制

日常生活中都用十进制（decimal）来表示整数，十进制数由$0, 1, 2, 3, 4, 5, 6, 7, 8, 9$这10个字符组成。一个十进制整数的第$k$位的值可以由$10^{k-1}$计算得到。

【范例】十进制

$$\begin{aligned}256 &= 200 + 50 + 6 \\ &= 2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0\end{aligned}$$

二进制（binary）、八进制（octal）、十六进制（hexadecimal）也是非常常用的表示法，例如计算机通常用二进制来做算术运算，而用八进制或十六进制来表示字符。

十进制	二进制	八进制	十六进制
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

进制转换

一个b进制的正整数n可以唯一地构造展开式：

$$n = a^k \times b^k + a_{k-1} \times b^{k-1} + \cdots + a_1 \times b^1 + a_0 \times b^0$$

【范例】b进制转十进制

$(1011)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = (11)_{10}$

$(21022)_3 = 2 \times 3^4 + 1 \times 3^3 + 0 \times 3^2 + 2 \times 3^1 + 2 \times 3^0 = (197)_{10}$

十进制转b进制还可以使用短除法的方式。

【范例】十进制转b进制

4	637	1
4	159	3
4	39	3
4	9	1
4	2	2

$(21331)_4$

2	637	1
2	318	0
2	159	1
2	79	1
2	39	1
2	19	1
2	9	1
2	4	0
2	2	0
2	1	1

$(1001111101)_2$

8	1000	0
8	125	5
8	15	7
8	1	1

$(1750)_8$

4.2 素数

素数（Prime Numbers）

基于整除性的一个重要概念就是素数，素数是大于1的且不能被1和它自身以外的正整数整除的整数。素数是现代密码学中必不可少的一部分，密码学中的大素数就用在信息加密的某些方法中。

【范例】判断素数

7是素数，因子：1、7。

9是合数，9能被3整除。

每个大于1的整数都可以唯一地写成多个素数的乘积。

【范例】素因子分解

$100 = 2 * 2 * 5 * 5 = 2^2 * 5^2$

$999 = 3 * 3 * 3 * 37 = 3^3 * 37$

$1024 = 2^{10}$

如果n是一个合数，那么n必有一个素因子小于或等于$\sqrt{n}$。

【范例】证明101是素数

不超过$\sqrt{101}$的素数只有$2, 3, 5, 7$，因为101不能被$2, 3, 5, 7$整除，所以101是素数。

【代码】素数

```python
import math

def is_prime(num):
    for i in range(2, int(math.sqrt(num)) + 1):
        if num % i == 0:
            return False
    return True

def main():
    print(is_prime(13))
```

```python
    print(is_prime(18))

if __name__ == "__main__":
    main()
```

埃拉托斯特尼筛法（Sieve of Eratosthenes）可以用来寻找不超过一个给定整数的所有素数。

步骤：

1. 建立包含所有给定整数以内的表格。
2. 从$i = 2$开始。
3. 移除所有整数$n \% i == 0$（除i以外）。
4. $i = i + 1$。
5. 重复第3步和第4步。

	埃	拉	托	斯	特	尼	筛	法	
	2	**3**	~~4~~	**5**	~~6~~	**7**	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	**17**	~~18~~	**19**	~~20~~
~~21~~	~~22~~	**23**	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	**29**	~~30~~

4.3 序列

序列（Sequence）

序列是一种用来表示有序列表的离散结构。例如$1, 2, 3, 5, 8$是一个含有五项的序列，而$1, 3, 9, 27, 81, ...$是一个无穷序列。序列可以用记号$\{a_n\}$表示。

- 递增序列（increasing sequence）：一个序列任意相邻的两项满足$a_k < a_{k+1}$。
- 非递减序列（non-decreasing sequence）：一个序列任意相邻的两项满足$a_k \leq a_{k+1}$。
- 递减序列（decreasing sequence）：一个序列任意相邻的两项满足$a_k > a_{k+1}$。
- 非递增序列（non-increasing sequence）：一个序列任意相邻的两项满足$a_k \geq a_{k+1}$。

【范例】序列

$\{a_n\},\ a_n = \frac{1}{n}:\ 1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, ...$

$\{b_n\},\ b_n = 2^n:\ 1, 2, 4, 8, 16, 32, ...$

算术级数（Arithmetic Sequence）

算术级数也称等差级数，序列形式如下：

$$a, a+d, a+2d, ..., a+nd, ...\ (a, d \in \mathbb{R})$$

【范例】算术级数

$\{a_n\},\ a_n = -1 + 4n:\ -1, 3, 7, 11, ...$

$\{b_n\},\ b_n = 7 - 3n:\ 7, 4, 1, -2, ...$

几何级数（Geometric Sequence）

几何级数也称等比级数，序列形式如下：

$$a, ar, ar^2, ..., ar^n, ...,\ (a, r \in \mathbb{R})$$

【范例】几何级数

$\{a_n\},\ a_n = (-1)^n:\ 1, -1, 1, -1, 1, \ldots$

$\{b_n\},\ b_n = -2 \times 5^n:\ 2, 10, 50, 250, 1250, \ldots$

$\{c_n\},\ c_n = 6 \times (\frac{1}{3})^n:\ 6, 2, \frac{2}{3}, \frac{2}{9}, \frac{2}{27}, \ldots$

4.4 递推关系

递推（Recurrence）

如果数列$\{a_n\}$的第n项与它前一项的关系可以用一个公式来表示，那么这个公式就叫做这个数列的递推方程。

算术级数的递推关系：

$$\begin{aligned} a_0 &= a \\ a_n &= a_{n-1} + d \end{aligned}$$

几何级数的递推关系：

$$\begin{aligned} a_0 &= a \\ a_n &= a_{n-1} \times r \end{aligned}$$

【范例】银行储蓄账户上有10000元，年利率为5.8%，7年后账户中将有多少钱？

$$P_n = P_{n-1} + 0.058P_{n-1} = (1.058)P_{n-1}$$

$$\begin{aligned} &P_0 = 10000 \\ &P_1 = (1.058)P_0 \\ &P_2 = (1.058)P_1 = (1.058)^2 P_0 \\ &\dots \\ &P_7 = (1.058)P_6 = (1.058)^7 P_0 \approx 14838.83 \end{aligned}$$

斐波那契数列（Fibonacci Sequence）

斐波那契数列f_0，f_1，f_2，$\dots$的递推公式为：

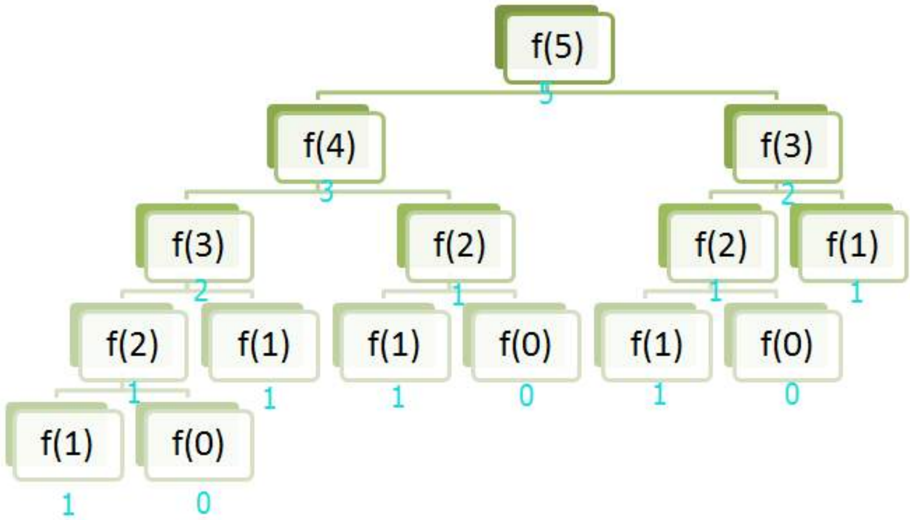
$$f(n) = \begin{cases} 1 & n = 1 \\ 1 & n = 2 \\ f(n-1) + f(n-2) & n > 3 \end{cases}$$

斐波那契数列的通项公式为：

$$f_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^{n+1}$$

【代码】斐波那契数列（递归）

```
int fibonacci(int n) {
    if(n == 1 || n == 2) {
        return 1;
    }
    return fibonacci(n-2) + fibonacci(n-1);
}
```



【代码】斐波那契数列（迭代）

```
int fibonacci(int n) {
    int f[n];
    f[0] = f[1] = 1;
    for(int i = 2; i < n; i++) {
        f[i] = f[i-2] + f[i-1];
    }
    return f[n-1];
}
```

4.5 求和

求和（Summation）

求和符号$\sum$可以用于表示序列中所有项的累加和。

$$\sum_{i=lower}^{upper} a_i$$

> i：求和下标
>
> lower：下限
>
> upper：上限

【范例】求和

$$\sum_{i=1}^{100} i = 1 + 2 + 3 + \cdots + 99 + 100 = 5050$$
$$\sum_{j=1}^{5} j^2 = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 = 55$$
$$\sum_{k=4}^{6} (-1)^k = (-1)^4 + (-1)^5 + (-1)^6 = 1 - 1 + 1 = 1$$

双重求和

很多情况下需要使用双重求和，比如在计算机程序中嵌套循环的分析中。

计算双重求和的方法是先展开内层求和，再继续计算外层求和。

【范例】双重求和

$$
\begin{aligned}
&\sum_{i=1}^{4}\sum_{j=1}^{3} ij \\
&= \sum_{i=1}^{4}(i + 2i + 3i) \\
&= \sum_{i=1}^{4} 6i \\
&= 6 + 12 + 18 + 24 \\
&= 60
\end{aligned}
$$

4.6 数学归纳法

数学归纳法（Mathematical Induction）

数学归纳法是一种数学证明方法，通常被用于证明某个给定命题在一个给定范围内成立。

数学归纳法分为三个步骤：

1. 归纳基础
2. 归纳假设
3. 归纳递推

【范例】证明$\sum_{i=1}^{n} i = \frac{n(n+1)}{2},\ n \in \mathbb{Z}^+$

$$1.\ \text{归纳基础：当} n = 1$$

$$\sum_{i=1}^{1} i = \frac{1(1+1)}{2} = 1$$

$$2.\ \text{归纳假设：假设} n = k$$

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2} \text{成立}$$

$$3.\ \text{归纳递推：证明} n = k+1 \text{时}$$

$$\sum_{i=1}^{k+1} i = \frac{(k+1)(k+2)}{2} \text{成立}$$

$$\begin{aligned} \sum_{i=1}^{k+1} i &= \sum_{i=1}^{k} i + k + 1 \\ &= \frac{k(k+1)}{2} + k + 1 \\ &= \frac{k(k+1) + 2(k+1)}{2} \\ &= \frac{(k+1)(k+2)}{2} \end{aligned}$$

【范例】证明$2^n \geq 3n.\ n \geq 4$

$$1. \text{归纳基础：当} n = 4$$
$$2^4 \geq 3 \times 4$$

$$2. \text{归纳假设：假设} n = k \ (k \geq 4)$$
$$2^k \geq 3k \text{成立}$$

$$3. \text{归纳递推：证明} n = k + 1 \text{时}$$
$$2^{k+1} \geq 3(k+1) \text{成立}$$

$$\begin{aligned} 2^{k+1} &= 2 \times 2^k \\ &\geq 2 \times 3k \\ &= 3k + 3k \\ &\geq 3k + 3 \\ &\geq 3(k+1) \end{aligned}$$

第5章 计数原理

5.1 计数原理

分类加法计数原理

完成一件事有n种不同的方案，其中第1种方案有m_1种不同方法，第2种方案有m_2种不同方法，$...$，第n种方案有m_n种不同方法，那么完成这件事共有$m_1 + m_2 + \cdots + m_n$种不同方法。

【范例】从A地到B地，可以乘火车、汽车、飞机。火车有4班、汽车2班、飞机3班，那么一天中乘坐这些交通工具从A地到B地有多少种不同的走法？

$$4 + 2 + 3 = 9$$

分步乘法计数原理

完成一件事需要n个步骤，其中第1个步骤有m_1种不同方法，第2个步骤有m_2种不同方法，$...$，第n个步骤有m_n种不同方法，那么完成这件事共有$m_1 \times m_2 \times ... m_n$种不同方法。

【范例】一个书架的第1层有4本不同的计算机书，第2层有3本不同的经济书，第3层有2本不同的数学书。从书架的每一层各取一本书，有多少种不同取法？

$$4 \times 3 \times 2 = 24$$

5.2 排列

排列（Permutation）

从n个不同元素中取出m $(m \leq n)$ 个元素，按照一定次序排成一列，称为从n个不同元素中取出m个元素的一个排列。

$$P_n^m = \frac{n!}{(n-m)!}$$

例如一共有8个人，A、B、C、D、E、F、G、H。现在有3个奖杯，分别为金牌、银牌和铜牌。将这3个奖牌颁发给8个人中的3个，问颁发奖牌的不同方式总共有几种？

很明显这是一个排列的问题，因为把金牌先颁给A，再把银牌颁给B，跟把金牌先颁给B，再把银牌颁给A 这是两种不同的颁奖方式。

> 第一步颁发金牌，金牌可以颁发给8个人中的1个，共有8种选择。
>
> 第二步颁发银牌，银牌可以颁发剩下7个人中的1个，共有7种选择。
>
> 第二步颁发铜牌，铜牌可以颁发剩下6个人中的1个，共有6种选择。

那么总共的颁奖方式共有$8 \times 7 \times 6 = 336$种。

【范例】用0-9这10个数字可以组成多少个没有重复数字的三位数？

> 方法1

由于0没有排在百位上，那么百位只能是1-9这9个数字任选1个，有P_9^1种选法。

对于十位和个位，从余下的9个数字种选2个，有P_9^2种选法。

$P_9^1 \times P_9^2 = 9 \times 9 \times 8 = 648$

> 方法2

符合条件的三位数可以分为3大类：

1. 每一位数字都不是0的三位数，也就是从1-9中选3个，有P_9^3种选法。
2. 个位数字为0，那么需要从剩下9个数字中选2个作为十位和百位，有P_9^2种选法。
3. 十位数字为0，那么需要从剩下9个数字中选2个作为个位和百位，有P_9^2种选法。

$P_9^3 + P_9^2 + P_9^2 = 648$

> 方法3

利用容斥法。从0-9这10个数字任取3个数字的排列数为P_{10}^3，其中0在百位上（也就是从1-9中选2个作为十位和个位）的排列数是P_9^2。

$$P_{10}^3 - P_9^2 = 648$$

【代码】打印字符串"math"的全排列

```c
#include <stdio.h>
#include <string.h>

void swap(char *a, char *b) {
    char temp = *a;
    *a = *b;
    *b = temp;
}

void permutation(char *s, int start, int end) {
    if(start >= end) {
        printf("%s\n", s);
    } else {
        for(int i = start; i < end; i++) {
            swap(s + i, s + start);
            permutation(s, start + 1, end);
            swap(s + i, s + start);
        }
    }
}

int main() {
    char s[] = "math";
    int len = strlen(s);
    permutation(s, 0, len);
    return 0;
}
```

5.3 组合

组合（Combination）

从n个不同元素中取出m $(m \leq n)$ 个元素，称为从n个不同元素中取出m个元素的一个组合。

排列与组合的不同点在于，排列与元素的顺序有关，组合与元素的顺序无关。

$$C_n^m = \frac{P_n^m}{P_m^m} = \frac{n!}{(n-m)!m!}$$

【范例】在100件产品中，有98件合格品，2件次品，从这100件产品中任意抽出3件。

（1）有多少种不同的抽法？

$$C_{100}^3 = \frac{100 \times 99 \times 98}{3 \times 2 \times 1} = 161700$$

（2）抽出的3件中恰好有1件事次品的抽法有多少种？

2件次品抽出1件有C_2^1种，再从98件合格品种抽出2件合格品有C_{98}^2种。

$$C_2^1 \times C_{98}^2 = 9506$$

（3）抽出的3件中至少有1件事次品的抽法有多少种？

> 方法1

恰好有1件次品有$C_2^1 \times C_{98}^2$种，恰好有2件次品有$C_2^2 \times C_{98}^1$种。

$$C_2^1 \times C_{98}^2 + C_2^2 \times C_{98}^1 = 9604$$

> 方法2

利用容斥法。先从100件抽出3件有C_{100}^3种，其中3件都是合格品有C_{98}^3种。

$$C_{100}^3 - C_{98}^3 = 9604$$

5.4 古典概型

古典概型

如果一个随机试验所包含的单位事件是有限的，且每个单位事件发生的可能性均相等，则这个随机试验叫做拉普拉斯试验，这种条件下的概率模型就叫古典概型。古典概型是概率论中最直观和最简单的模型，概率的许多运算规则，也首先是在这种模型下得到的。

单位事件的特点是两两互斥的，例如抛一枚质地均匀的硬币时，正面朝上和背面朝上不会同时出现。

在古典概型中，概率的计算公式为：

$$P(A)=\frac{A\text{包含的单位事件个数}m}{\text{单位事件的总数}n}$$

【范例】掷两个质地均匀的骰子

（1）一共有多少种不同的结果？

$6\times 6=36$

（2）点数之和为9的结果有多少种？

一共有4种：$(3,6),(6,3),(4,5),(5,4)$

（3）点数之和为9的概率是多少？

$P(A)=\frac{4}{36}=\frac{1}{9}$