算法竞赛中的数论

2. 模 m 剩余系的加法与乘法结构及其应用

陈亮舟
PinkRabbit

清华大学

FOI 2022 算法夏令营提高班 day 5

目录

说明

此版本课件为 FOI 2022 算法夏令营提高班 day 5 特供版。
由于赶工严重以及课程时间较短，在内容上进行了精简（删减），只保留关键知识点（?）和例题，可能有缺少结论证明、逻辑链不完整、缺少例子、较难建立完整框架等问题，与最终版本将有较大差异。
本系列课件的后续更新将在 算法竞赛中的数论 – 系列课件 – GitHub/GitPinkRabbit 中上传。

引入

本课件中我们聚焦于给定一个固定的模数 m 并且一切运算都对 m 取模时可能遇到的数论问题。此处的 m 可以为任意正整数。

未特别说明时，本课件中的一切同余式都将省略模 m 的提示性后缀，即将 $\mathcal{A} \equiv \mathcal{B} \pmod{m}$ 简写作 $\mathcal{A} \equiv \mathcal{B}$。

引入

本课件中我们聚焦于给定一个固定的模数 m 并且一切运算都对 m 取模时可能遇到的数论问题。此处的 m 可以为任意正整数。
未特别说明时，本课件中的一切同余式都将省略模 m 的提示性后缀，即将 $\mathcal{A} \equiv \mathcal{B} \pmod{m}$ 简写作 $\mathcal{A} \equiv \mathcal{B}$。
在内容编排上，每一节的内容为：

1. 介绍剩余类与剩余系的概念，并介绍在剩余系内进行运算时的注意事项与快速幂算法。
2. 结合课件一中的知识讨论剩余系的加法结构。
3. 初步讨论剩余系的乘法结构并介绍 Euler 函数、Fermat 小定理、Euler 定理以及它们的应用等等。
4. 进一步讨论剩余系的乘法结构，介绍阶、原根、指标等概念与相关的一些应用，介绍离散对数问题与大步小步算法。

1

模 m 剩余类与剩余系

当前进度

同余等价

回顾课件一中对同余的定义：对于整数 a, b，称 $a \equiv b$，当且仅当 $m \mid (a-b)$。也就是存在整数 k 使得 $km = a - b$，即 $a = b + km$。容易证明同余是一种等价关系，即满足自反性（$a \equiv a$）、对称性（如果 $a \equiv b$ 则 $b \equiv a$）、与传递性（如果 $a \equiv b$ 且 $b \equiv c$ 则 $a \equiv c$）。

同余等价

回顾课件一中对同余的定义：对于整数 a, b，称 $a \equiv b$，当且仅当 $m \mid (a - b)$。也就是存在整数 k 使得 $km = a - b$，即 $a = b + km$。

容易证明同余是一种等价关系，即满足自反性（$a \equiv a$）、对称性（如果 $a \equiv b$ 则 $b \equiv a$）、与传递性（如果 $a \equiv b$ 且 $b \equiv c$ 则 $a \equiv c$）。

同余是一种等价关系说明了可以将全体整数划分为若干类，每个整数恰好属于其中一类，每一类中的整数间两两同余，不同类之间的整数间两两不同余。

同余等价

回顾课件一中对同余的定义：对于整数 a, b，称 $a \equiv b$，当且仅当 $m \mid (a-b)$。也就是存在整数 k 使得 $km = a - b$，即 $a = b + km$。

容易证明同余是一种等价关系，即满足自反性（$a \equiv a$）、对称性（如果 $a \equiv b$ 则 $b \equiv a$）、与传递性（如果 $a \equiv b$ 且 $b \equiv c$ 则 $a \equiv c$）。

同余是一种等价关系说明了可以将全体整数划分为若干类，每个整数恰好属于其中一类，每一类中的整数间两两同余，不同类之间的整数间两两不同余。

例 (等价类)

- 当 $m = 1$ 时，全体整数为一个等价类，因为 1 整除所有整数。
- 当 $m = 6$ 时，$\{\ldots, -10, -4, 2, 8, 14, \ldots\}$ 为一个等价类，容易验证其中任意两数同余，并且其中任意数与其他整数不同余。

剩余类与剩余系

由于 a 与所有 $a+km$（k 为整数）必然在同一个等价类中，而其他整数都与 a 不在同一个等价类中，所以每个等价类都形如 a 加上若干（正数、零、或负数）倍的 m。

定义 (剩余类)

对于整数 a，称所有同余于 a 模 m 的整数构成的集合为一个**模 m 剩余类（residue class modulo m）**，显然 a 在它本身定义的剩余类中，这个剩余类形如 $\{\ldots, a-2m, a-m, a, a+m, a+2m, \ldots\}$。

剩余类与剩余系

定义 (剩余系)

恰好有 m 个剩余类，它们分别可以由 $0, 1, \ldots, m-1$ 定义得到。
将 m 个剩余类看作元素，它们构成的集合称为**模 m 剩余系 (residue system modulo m)**。
对于任意整数 i，将 i 定义得到的剩余类直接记作 i，后文中不再使用记号区分整数与剩余类。可以发现同余的两数 a, b 定义得到的剩余类相同，在此意义上可以写作 $a = b$。

剩余类与剩余系

定义 (剩余系)

恰好有 m 个剩余类，它们分别可以由 $0, 1, \ldots, m-1$ 定义得到。
将 m 个剩余类看作元素，它们构成的集合称为**模 m 剩余系 (residue system modulo m)**。
对于任意整数 i，将 i 定义得到的剩余类直接记作 i，后文中不再使用记号区分整数与剩余类。可以发现同余的两数 a, b 定义得到的剩余类相同，在此意义上可以写作 $a = b$。

例 (剩余系)

当 $m = 6$ 时，$\{0, 1, 2, 3, 4, 5\}$ 为剩余系，而 $\{-2, -1, 3, 6, 7, 15\}$ 也为剩余系，因为它们均表示 m 个不同的剩余类，只是记号不同。

剩余类与剩余系 – 小结

尽管此后整数与剩余类使用同一记号，但请注意模意义下剩余系仍然是一个抽象的概念。
例如在剩余系中 $\underbrace{1+1+\cdots+1}_{m\text{ 个}}=0$，这是通常逻辑无法想象的。
再例如线性同余方程 $ax \equiv b \pmod m$ 的解集本可以表示为 $x \equiv x_0 \pmod{\frac{m}{\gcd(a,m)}}$，但在剩余类的语言中可写作 $x = x_0$（模 $\frac{m}{\gcd(a,m)}$），仅有唯一解。
通过此例，可以发现，也需要注意在谈论剩余类与剩余系时明确模数，否则在一些情况下可能会造成混乱。

当前进度

1 模 m 剩余类与剩余系
- 剩余类与剩余系
- 剩余类的运算
  - 剩余类的运算
  - 计算机中的表示
  - 快速幂

2 加法结构

3 乘法结构（上）

4 乘法结构（中）

5 附录

剩余类的运算

我们还未在剩余类间定义运算。好在，同余的性质保证了剩余类间的运算可以如整数运算一般自然地定义：对于两个剩余类 x, y，它们的和、差、积只需定义为分别从其中取出任意一个整数后，两个整数间的对应运算结果所在的剩余类。

剩余类的运算

我们还未在剩余类间定义运算。好在，同余的性质保证了剩余类间的运算可以如整数运算一般自然地定义：对于两个剩余类 x, y，它们的和、差、积只需定义为分别从其中取出任意一个整数后，两个整数间的对应运算结果所在的剩余类。可以证明无论怎样选取这两个整数，运算结果所在的剩余类均不会改变。

剩余类的运算

我们还未在剩余类间定义运算。好在，同余的性质保证了剩余类间的运算可以如整数运算一般自然地定义：对于两个剩余类 x, y，它们的和、差、积只需定义为分别从其中取出任意一个整数后，两个整数间的对应运算结果所在的剩余类。可以证明无论怎样选取这两个整数，运算结果所在的剩余类均不会改变。

例 (剩余类的运算)

当 $m = 6$ 时（总是将剩余类记作 $[0, m-1]$ 中的整数）：

剩余类的运算

我们还未在剩余类间定义运算。好在，同余的性质保证了剩余类间的运算可以如整数运算一般自然地定义：对于两个剩余类 x, y，它们的和、差、积只需定义为分别从其中取出任意一个整数后，两个整数间的对应运算结果所在的剩余类。可以证明无论怎样选取这两个整数，运算结果所在的剩余类均不会改变。

例 (剩余类的运算)

当 $m = 6$ 时（总是将剩余类记作 $[0, m-1]$ 中的整数）：

- 剩余类 $4, 5$ 的和可以为 $4 + 5 = 9$ 所在的剩余类，即 3。

剩余类的运算

我们还未在剩余类间定义运算。好在，同余的性质保证了剩余类间的运算可以如整数运算一般自然地定义：对于两个剩余类 x, y，它们的和、差、积只需定义为分别从其中取出任意一个整数后，两个整数间的对应运算结果所在的剩余类。可以证明无论怎样选取这两个整数，运算结果所在的剩余类均不会改变。

例 (剩余类的运算)

当 $m = 6$ 时（总是将剩余类记作 $[0, m-1]$ 中的整数）：

- 剩余类 $4, 5$ 的和可以为 $4 + 5 = 9$ 所在的剩余类，即 3。
- 剩余类 $4, 5$ 的差可以为 $4 - (-1) = 5$ 所在的剩余类，即 5。

剩余类的运算

我们还未在剩余类间定义运算。好在，同余的性质保证了剩余类间的运算可以如整数运算一般自然地定义：对于两个剩余类 x, y，它们的和、差、积只需定义为分别从其中取出任意一个整数后，两个整数间的对应运算结果所在的剩余类。可以证明无论怎样选取这两个整数，运算结果所在的剩余类均不会改变。

例 (剩余类的运算)

当 $m = 6$ 时（总是将剩余类记作 $[0, m-1]$ 中的整数）：

- 剩余类 $4, 5$ 的和可以为 $4 + 5 = 9$ 所在的剩余类，即 3。
- 剩余类 $4, 5$ 的差可以为 $4 - (-1) = 5$ 所在的剩余类，即 5。
- 剩余类 $4, 5$ 的积可以为 $(-2) \cdot (-1) = 2$ 所在的剩余类，即 2。

剩余类的运算

我们还未在剩余类间定义运算。好在，同余的性质保证了剩余类间的运算可以如整数运算一般自然地定义：对于两个剩余类 x, y，它们的和、差、积只需定义为分别从其中取出任意一个整数后，两个整数间的对应运算结果所在的剩余类。可以证明无论怎样选取这两个整数，运算结果所在的剩余类均不会改变。

例 (剩余类的运算)

当 $m = 6$ 时（总是将剩余类记作 $[0, m-1]$ 中的整数）：

- 剩余类 $4, 5$ 的和可以为 $4 + 5 = 9$ 所在的剩余类，即 3。
- 剩余类 $4, 5$ 的差可以为 $4 - (-1) = 5$ 所在的剩余类，即 5。
- 剩余类 $4, 5$ 的积可以为 $(-2) \cdot (-1) = 2$ 所在的剩余类，即 2。

即，表示剩余类时，$4 + 5 = 3$、$4 - 5 = 5$、$4 \cdot 5 = 2$。

剩余类运算与整数同余式的关系

在上例中，我们看到当 $m = 6$ 时有 $4 + 5 = 3$、$4 - 5 = 5$、$4 \cdot 5 = 2$。

剩余类运算与整数同余式的关系

在上例中，我们看到当 $m = 6$ 时有 $4 + 5 = 3$、$4 - 5 = 5$、$4 \cdot 5 = 2$。然而这与同余式如出一辙：同样有 $4 + 5 \equiv 3$、$4 - 5 \equiv 5$、$4 \cdot 5 \equiv 2$，这里将数字看作整数而非剩余类。

剩余类运算与整数同余式的关系

在上例中，我们看到当 $m = 6$ 时有 $4 + 5 = 3$、$4 - 5 = 5$、$4 \cdot 5 = 2$。然而这与同余式如出一辙：同样有 $4 + 5 \equiv 3$、$4 - 5 \equiv 5$、$4 \cdot 5 \equiv 2$，这里将数字看作整数而非剩余类。定义了剩余类之间的加法、减法、与乘法后，任何由剩余类、或与整数混合构成的多项式的运算结果都可以被合理定义，并且运算结果与同余式的运算结果将精确对应。

剩余类运算与整数同余式的关系

在上例中，我们看到当 $m=6$ 时有 $4+5=3$、$4-5=5$、$4\cdot 5=2$。然而这与同余式如出一辙：同样有 $4+5\equiv 3$、$4-5\equiv 5$、$4\cdot 5\equiv 2$，这里将数字看作整数而非剩余类。定义了剩余类之间的加法、减法、与乘法后，任何由剩余类、或与整数混合构成的多项式的运算结果都可以被合理定义，并且运算结果与同余式的运算结果将精确对应。

那么，剩余类，抛开与同余式的对应关系，的额外意义是什么呢?

剩余类运算与整数同余式的关系

这里或需提到几点：

- 剩余类与整数的对应关系，在某种意义上并不精确。当 $m=4$ 时，尽管作为剩余类有 $0=0\cdot 2$，但这并不意味着整数 4 能被两个分别在剩余类 $0,2$ 中的整数相乘得到。若表示成集合内元素相乘，应有 $0\cdot 2=\{\dots,-16,-8,0,8,16,\dots\}$。
- 同余式两侧的整数运算结果是整数，保留了运算的一切信息，而剩余类只保留了余数信息。
- 在运算中，绝不是所有对象都可以转换为剩余类进行，例如幂运算中的指数仍应看作非负整数而非剩余类。

剩余类运算与整数同余式的关系

这里或需提到几点：

- 剩余类与整数的对应关系，在某种意义上并不精确。当 $m = 4$ 时，尽管作为剩余类有 $0 = 0 \cdot 2$，但这并不意味着整数 4 能被两个分别在剩余类 $0, 2$ 中的整数相乘得到。若表示成集合内元素相乘，应有 $0 \cdot 2 = \{\ldots, -16, -8, 0, 8, 16, \ldots\}$。
- 同余式两侧的整数运算结果是整数，保留了运算的一切信息，而剩余类只保留了余数信息。
- 在运算中，绝不是所有对象都可以转换为剩余类进行，例如幂运算中的指数仍应看作非负整数而非剩余类。

算法竞赛实践中，使用剩余类时，既有可能指剩余类对象本身，又有可能指这一剩余类中的特定整数，还有可能指这一剩余类中的全体整数，需要留心辨别剩余类概念在上下文中的含义。

当前进度

1 模 m 剩余类与剩余系
- 剩余类与剩余系
- 剩余类的运算
  - 剩余类的运算
  - 计算机中的表示
  - 快速幂

2 加法结构

3 乘法结构（上）

4 乘法结构（中）

5 附录

剩余类在计算机中的表示

一共有 m 个剩余类，即 $\{0, 1, \ldots, m-1\}$。在计算机中，它们可以直接使用整数类型进行表示：每个剩余类恰好自然地对应一个在 $[0, m-1]$ 内的整数，即这个剩余类中唯一在 $[0, m-1]$ 内的整数。

剩余类在计算机中的表示

一共有 m 个剩余类，即 $\{0, 1, \ldots, m-1\}$。在计算机中，它们可以直接使用整数类型进行表示：每个剩余类恰好自然地对应一个在 $[0, m-1]$ 内的整数，即这个剩余类中唯一在 $[0, m-1]$ 内的整数。

例 (剩余类在 $[0, m-1]$ 内的表示)

当 $m = 6$ 时，

- 剩余类 $3 = \{\ldots, -9, -3, 3, 9, 15, \ldots\}$ 对应整数 3，
- 剩余类 $17 = \{\ldots, 5, 11, 17, 23, 29, \ldots\}$ 对应整数 5。

剩余类在计算机中的表示

一共有 m 个剩余类，即 $\{0, 1, \ldots, m-1\}$。在计算机中，它们可以直接使用整数类型进行表示：每个剩余类恰好自然地对应一个在 $[0, m-1]$ 内的整数，即这个剩余类中唯一在 $[0, m-1]$ 内的整数。

例 (剩余类在 $[0, m-1]$ 内的表示)

当 $m = 6$ 时，

- 剩余类 $3 = \{\ldots, -9, -3, 3, 9, 15, \ldots\}$ 对应整数 3，
- 剩余类 $17 = \{\ldots, 5, 11, 17, 23, 29, \ldots\}$ 对应整数 5。

剩余类 $-1 = \{\ldots, -1-2m, -1-m, -1, m-1, 2m-1\}$ 总是对应整数 $m-1$。

剩余类在计算机中的表示

一共有 m 个剩余类，即 $\{0, 1, \ldots, m-1\}$。在计算机中，它们可以直接使用整数类型进行表示：每个剩余类恰好自然地对应一个在 $[0, m-1]$ 内的整数，即这个剩余类中唯一在 $[0, m-1]$ 内的整数。

例 (剩余类在 $[0, m-1]$ 内的表示)

当 $m = 6$ 时，

- 剩余类 $3 = \{\ldots, -9, -3, 3, 9, 15, \ldots\}$ 对应整数 3，
- 剩余类 $17 = \{\ldots, 5, 11, 17, 23, 29, \ldots\}$ 对应整数 5。

剩余类 $-1 = \{\ldots, -1-2m, -1-m, -1, m-1, 2m-1\}$ 总是对应整数 $m-1$。

可以发现，即使给定整数，它定义的剩余类的表示并不一定是这个整数本身，但总与这个整数同余。

剩余类的运算在计算机中的表示

若严格遵循使用 $[0, m-1]$ 内的整数表示剩余类，计算机执行算数运算时，可能导致运算结果偏离这个范围，尽管得到的结果仍然与正确表示同余，此时需要进行额外操作将结果拉回范围内。

- 将整数 x 所在的剩余类转换为整数表示：$x \bmod m$。
- 将整数表示为 x, y 的剩余类之和拉回范围内：$(x+y) \bmod m$。
- 将整数表示为 x, y 的剩余类之差拉回范围内：$(x-y) \bmod m$。
- 将整数表示为 x, y 的剩余类之积拉回范围内：$(x \cdot y) \bmod m$。

剩余类的运算在计算机中的表示

若严格遵循使用 $[0, m-1]$ 内的整数表示剩余类，计算机执行算数运算时，可能导致运算结果偏离这个范围，尽管得到的结果仍然与正确表示同余，此时需要进行额外操作将结果拉回范围内。

- 将整数 x 所在的剩余类转换为整数表示：$x \bmod m$。
- 将整数表示为 x, y 的剩余类之和拉回范围内：$(x+y) \bmod m$。
- 将整数表示为 x, y 的剩余类之差拉回范围内：$(x-y) \bmod m$。
- 将整数表示为 x, y 的剩余类之积拉回范围内：$(x \cdot y) \bmod m$。

一般来说，必须假设剩余类和整数取模进行加法、减法、乘法运算时是 $\mathcal{O}(1)$ 的，后续课件中将探讨实践中遇到的算法常数问题。

剩余类的运算在计算机中的表示 – 注意事项

然而，在 C++ 中，由于除法运算符向零舍入，使用 % 表示取模时，对于中间结果可能为负的需要进行修正，例如减法应写为

$$(x - y + m) \ \% \ m。$$

剩余类的运算在计算机中的表示 – 注意事项

然而，在 C++ 中，由于除法运算符向零舍入，使用 `%` 表示取模时，对于中间结果可能为负的需要进行修正，例如减法应写为

$$\texttt{(}x\ \texttt{-}\ y\ \texttt{+}\ m\texttt{)}\ \texttt{\%}\ m。$$

同时，需要考虑整型溢出的问题，这在乘法中尤为常见，因为常见的模数 m 一般是在 2^{30} 范围内的大数，而两个整数表示相乘得到的结果范围是 $[0,(m-1)^2]$，只有当 $m \le 46341$ 时，在单次乘法中才没有可能出现整型溢出的情况。当 m 可能 > 46341 时，乘法应写为

$$\texttt{((long long)}x\ \texttt{*}\ y\texttt{)}\ \texttt{\%}\ m。$$

剩余类的运算在计算机中的表示 – 注意事项

然而，在 C++ 中，由于除法运算符向零舍入，使用 % 表示取模时，对于中间结果可能为负的需要进行修正，例如减法应写为

$$\texttt{(}x\texttt{ - }y\texttt{ + }m\texttt{) \% }m\text{。}$$

同时，需要考虑整型溢出的问题，这在乘法中尤为常见，因为常见的模数 m 一般是在 2^{30} 范围内的大数，而两个整数表示相乘得到的结果范围是 $[0,(m-1)^2]$，只有当 $m \le 46341$ 时，在单次乘法中才没有可能出现整型溢出的情况。当 m 可能 > 46341 时，乘法应写为

$$\texttt{((long long)}x\texttt{ * }y\texttt{) \% }m\text{。}$$

有时模数过大或中间结果运算较复杂，均有可能出现即使没有乘法运算仍然整型溢出的现象。实践中，大量代码错误由未正确使用整数表示或出现整型溢出导致。

当前进度

求幂问题

不难理解为何到现在才将“求幂”视为一个问题：在整数中，如果不需要取模，若不使用更高精度的整数，除开 0^n 与 1^n 等平凡情况，即使是计算 2^n，n 也不能超过 64 或 128，否则结果将难以使用 C++ 原生类型表示，于是不需要担心 $\mathcal{O}(n)$ 算法的时间复杂度；而在浮点数中，有 `std::pow` 函数帮助完成更广泛的情况。

求幂问题

不难理解为何到现在才将“求幂”视为一个问题：在整数中，如果不需要取模，若不使用更高精度的整数，除开 0^n 与 1^n 等平凡情况，即使是计算 2^n，n 也不能超过 64 或 128，否则结果将难以使用 C++ 原生类型表示，于是不需要担心 $\mathcal{O}(n)$ 算法的时间复杂度；而在浮点数中，有 `std::pow` 函数帮助完成更广泛的情况。

在模意义下，求 $a^n \bmod m$ 是一个需要考虑如何快速计算的问题，其中 a 可以是任意整数，而 n 是非负整数。
（当 $n = 0$ 时，认为 $a^0 = 1$ 对所有整数 a 成立，包括 $0^0 = 1$。）

求幂问题

不难理解为何到现在才将“求幂”视为一个问题：在整数中，如果不需要取模，若不使用更高精度的整数，除开 0^n 与 1^n 等平凡情况，即使是计算 2^n，n 也不能超过 64 或 128，否则结果将难以使用 C++ 原生类型表示，于是不需要担心 $\mathcal{O}(n)$ 算法的时间复杂度；而在浮点数中，有 `std::pow` 函数帮助完成更广泛的情况。

在模意义下，求 $a^n \bmod m$ 是一个需要考虑如何快速计算的问题，其中 a 可以是任意整数，而 n 是非负整数。
(当 $n = 0$ 时，认为 $a^0 = 1$ 对所有整数 a 成立，包括 $0^0 = 1$。)

后文中，将看到 $\{\langle n, a^n \rangle\}_{n=0}^{\infty}$ 可能含有相对复杂的结构，但对于求幂问题我们有简单方便的解决方案。

快速幂 – 二进制表示

考虑 n 的二进制表示 $n = \overline{n_k n_{k-1} \cdots n_2 n_1 n_0}_{(2)} = \sum_{i=0}^{k} n_i \cdot 2^i$。

快速幂 – 二进制表示

考虑 n 的二进制表示 $n = \overline{n_k n_{k-1} \cdots n_2 n_1 n_0}_{(2)} = \sum_{i=0}^{k} n_i \cdot 2^i$。
根据公式 $a^{s+t} = a^s \cdot a^t$，又由于 n_i 为 0 或 1，我们有

$$
\begin{aligned}
a^n &= a^{\sum_{i=0}^{k} n_i \cdot 2^i} \\
&= a^{n_0} \cdot a^{n_1 \cdot 2} \cdot a^{n_2 \cdot 4} \cdot \cdots \cdot a^{n_k \cdot 2^k} \\
&= \prod_{n_i = 1} a^{2^i}。
\end{aligned}
$$

快速幂 – 二进制表示

考虑 n 的二进制表示 $n = \overline{n_k n_{k-1} \cdots n_2 n_1 n_0}_{(2)} = \sum_{i=0}^{k} n_i \cdot 2^i$。
根据公式 $a^{s+t} = a^s \cdot a^t$，又由于 n_i 为 0 或 1，我们有

$$\begin{aligned} a^n &= a^{\sum_{i=0}^{k} n_i \cdot 2^i} \\ &= a^{n_0} \cdot a^{n_1 \cdot 2} \cdot a^{n_2 \cdot 4} \cdot \cdots \cdot a^{n_k \cdot 2^k} \\ &= \prod_{n_i = 1} a^{2^i}。 \end{aligned}$$

例

由于 $13 = \overline{1101}_{(2)}$，有 $a^{13} = a \cdot a^4 \cdot a^8$。
由于 $33289 = \overline{1000001000001001}_{(2)}$，有 $a^{33289} = a \cdot a^8 \cdot a^{512} \cdot a^{32768}$。

快速幂 – 反复平方

据此，结果可拆分成不超过 $\lceil \log_2 n \rceil$ 个形如 a^{2^i} 的中间结果的乘积，而 a^{2^i} 中的 i 不超过 $\lfloor \log_2 n \rfloor$。

快速幂 – 反复平方

据此，结果可拆分成不超过 $\lceil \log_2 n \rceil$ 个形如 a^{2^i} 的中间结果的乘积，而 a^{2^i} 中的 i 不超过 $\lfloor \log_2 n \rfloor$。

对于 a^{2^i}，可以考虑反复平方：

- a 平方后得到 $a^2 = a^{2^1}$。
- a^2 平方后得到 $a^4 = a^{2^2}$。
- a^4 平方后得到 $a^8 = a^{2^3}$。
- 以此类推，a 反复平方 i 次后恰好得到 a^{2^i}。

据此得到每个 a^{2^i} 后，再使用 n 的二进制表示，即可通过中间结果得到 $a^n \bmod m$。需要注意中间结果与最终计算时每一步都要取模。

快速幂 – 反复平方

据此，结果可拆分成不超过 $\lceil \log_2 n \rceil$ 个形如 a^{2^i} 的中间结果的乘积，而 a^{2^i} 中的 i 不超过 $\lfloor \log_2 n \rfloor$。

对于 a^{2^i}，可以考虑反复平方：

- a 平方后得到 $a^2 = a^{2^1}$。
- a^2 平方后得到 $a^4 = a^{2^2}$。
- a^4 平方后得到 $a^8 = a^{2^3}$。
- 以此类推，a 反复平方 i 次后恰好得到 a^{2^i}。

据此得到每个 a^{2^i} 后，再使用 n 的二进制表示，即可通过中间结果得到 $a^n \bmod m$。需要注意中间结果与最终计算时每一步都要取模。假设求 n 的二进制表示只需要 $\mathcal{O}(\log n)$ 的时间复杂度，容易看出快速幂的时间复杂度为 $\mathcal{O}(\log n)$。

快速幂 – C++ 代码实现

快速幂有很简便的代码实现方式。此处给出一种一边反复平方、一边计算结果的实现方式，此种方式的空间复杂度为 $\mathcal{O}(1)$。

```
int quick_pow(int a, int n) {
  int b = 1;
  for (; n; n >>= 1, a = (long long)a * a % M)
    if (n & 1)
      b = (long long)b * a % M;
  return b;
}
```

在第 i 次进入循环体时，a 的值恰为 $a^{2^{i-1}}$，而 n & 1 恰为 n_{i-1}。

快速幂 – 其他应用

快速幂算法流程本身与数论并无太大关系，可以发现，只要是有结合律的运算由同一个对象连续进行大量次数，都可以使用快速幂在 $\mathcal{O}(\log n)$ 次运算内计算。
快速幂的其他常见应用有：矩阵快速幂（取模或浮点数）、多项式快速幂（取模）等等。

快速幂 – 其他应用

快速幂算法流程本身与数论并无太大关系，可以发现，只要是有结合律的运算由同一个对象连续进行大量次数，都可以使用快速幂在 $\mathcal{O}(\log n)$ 次运算内计算。
快速幂的其他常见应用有：矩阵快速幂（取模或浮点数）、多项式快速幂（取模）等等。

需要进行的运算次数关于 n 的函数，是衡量快速幂算法效率的一个重要指标，在如矩阵快速幂等运算复杂度较高的场景中，运算次数需要尽可能少。可以发现，前文给出的快速幂算法使用不超过 $2\log_2 n$ 次运算。针对特定的 n，找出运算次数尽可能少的算法，被称为最短加法链问题。容易发现，再短的加法链也至少需要 $\log_2 n$ 次运算，所以前文给出的快速幂算法在常数上最多劣一倍。

2

加法结构

未完成

此部分未完成，后续更新将在 算法竞赛中的数论 – 系列课件 – GitHub/GitPinkRabbit 中上传。

3

乘法结构（上）

关于内容编排的说明

相比于剩余系的加法结构，剩余系的乘法结构则要难以把握得多。本系列课件分三节讨论剩余系的乘法结构，本课件中含有其中前两节，而第三节将包含于课件三《中国剩余定理》中。

当前进度

简化剩余系

为了讨论剩余系的乘法结构，首先引入简化剩余系的概念。

定义 (简化剩余系)

在 m 个剩余类中，拥有乘法逆元的剩余类的集合称为**模 m 简化剩余系（缩系，reduced residue system modulo m）**。
作为对比，模 m 剩余系常被称为完全剩余系（简称完系）。
记简化剩余系中的剩余类 a 的逆元为 a^{-1}。

简化剩余系

为了讨论剩余系的乘法结构，首先引入简化剩余系的概念。

定义 (简化剩余系)

在 m 个剩余类中，拥有乘法逆元的剩余类的集合称为**模 m 简化剩余系（缩系，reduced residue system modulo m）**。
作为对比，模 m 剩余系常被称为完全剩余系（简称完系）。
记简化剩余系中的剩余类 a 的逆元为 a^{-1}。

根据课件一中的结论：只有与 m 互素的整数才可以定义逆元 (Bézout 定理)，可以给出简化剩余系的另一个等价定义：

- $[0, m-1]$ 中与 m 互素的整数所在的剩余类组成了简化剩余系。

简化剩余系 – 例子

例 (简化剩余系)

- 当 $m=4$ 时，缩系为 $\{1,3\}$，这是因为 $1\cdot1=3\cdot3=1$。
- 当 $m=5$ 时，缩系为 $\{1,2,3,4\}$，这是因为 $1\cdot1=2\cdot3=3\cdot2=4\cdot4=1$，其中 $2,3$ 互为逆元。
- 当 $m=6$ 时，缩系为 $\{1,5\}$，其他剩余类 $0,2,3,4$ 均没有乘法逆元。
- 当 $m=1$ 时，缩系为 $\{0\}$。这是唯一一个简化剩余系等于完全剩余系的例子。

可以看出这些简化剩余系中的任意整数都与 m 互素。

简化剩余系 – 性质

由于逆元的存在性，简化剩余系有着很好的性质。

简化剩余系 – 性质

由于逆元的存在性，简化剩余系有着很好的性质。

- 缩系对乘法封闭，即如果 a, b 在缩系中，则 ab 也在。这是因为 $(ab)(a^{-1}b^{-1}) = 1$。

简化剩余系 – 性质

由于逆元的存在性，简化剩余系有着很好的性质。

- 缩系对乘法封闭，即如果 a, b 在缩系中，则 ab 也在。这是因为 $(ab)(a^{-1}b^{-1}) = 1$。
- 缩系对逆元封闭。a 的逆元的逆元即为 a 自身，这是由于 $a \cdot a^{-1} = a^{-1} \cdot a = 1$。

简化剩余系 – 性质

由于逆元的存在性，简化剩余系有着很好的性质。

- 缩系对乘法封闭，即如果 a, b 在缩系中，则 ab 也在。这是因为 $(ab)(a^{-1}b^{-1}) = 1$。
- 缩系对逆元封闭。a 的逆元的逆元即为 a 自身，这是由于 $a \cdot a^{-1} = a^{-1} \cdot a = 1$。
- 上一条说明缩系中的剩余类要么两两配对，要么逆元为自身。例如，当 $m = 8$ 时，缩系 $\{1, 3, 5, 7\}$ 中每一个的逆元均为自身。

简化剩余系 – 性质

由于逆元的存在性，简化剩余系有着很好的性质。

- 缩系对乘法封闭，即如果 a, b 在缩系中，则 ab 也在。这是因为 $(ab)(a^{-1}b^{-1}) = 1$。
- 缩系对逆元封闭。a 的逆元的逆元即为 a 自身，这是由于 $a \cdot a^{-1} = a^{-1} \cdot a = 1$。
- 上一条说明缩系中的剩余类要么两两配对，要么逆元为自身。例如，当 $m = 8$ 时，缩系 $\{1, 3, 5, 7\}$ 中每一个的逆元均为自身。
- 在缩系中可以定义除法，a/b 可被定义为 $a \cdot b^{-1}$。

简化剩余系 – 性质

由于逆元的存在性，简化剩余系有着很好的性质。

- 缩系对乘法封闭，即如果 a, b 在缩系中，则 ab 也在。这是因为 $(ab)(a^{-1}b^{-1}) = 1$。
- 缩系对逆元封闭。a 的逆元的逆元即为 a 自身，这是由于 $a \cdot a^{-1} = a^{-1} \cdot a = 1$。
- 上一条说明缩系中的剩余类要么两两配对，要么逆元为自身。例如，当 $m = 8$ 时，缩系 $\{1, 3, 5, 7\}$ 中每一个的逆元均为自身。
- 在缩系中可以定义除法，a/b 可被定义为 $a \cdot b^{-1}$。
- 但是，缩系不对加减法封闭。例如，当 $m = 9$ 时，$1 + 2 = 3$ 不在缩系中；除了 $m = 1$ 外，0 均不在缩系中，而显然同一个数相减即可得到 0。

简化剩余系 – 性质

由于逆元的存在性，简化剩余系有着很好的性质。

- 如果 a 在缩系中，则 $-a$ 也在缩系中，因为 $(-a)\cdot(-a^{-1})=1$。

简化剩余系 – 性质

由于逆元的存在性，简化剩余系有着很好的性质。

- 如果 a 在缩系中，则 $-a$ 也在缩系中，因为 $(-a) \cdot (-a^{-1}) = 1$。
- 缩系中任意剩余类 a 均可以通过一次与缩系中的剩余类 c 的乘法得到任意缩系中的剩余类 b，即 $ax = b$ 总是在缩系中有解，取 $x = b/a$ 即可。

简化剩余系 – 性质

由于逆元的存在性，简化剩余系有着很好的性质。

- 如果 a 在缩系中，则 $-a$ 也在缩系中，因为 $(-a)\cdot(-a^{-1})=1$。
- 缩系中任意剩余类 a 均可以通过一次与缩系中的剩余类 c 的乘法得到任意缩系中的剩余类 b，即 $ax=b$ 总是在缩系中有解，取 $x=b/a$ 即可。
- 对于任意缩系中的剩余类 c，不存在两个不同的（完系中的）剩余类 a,b 满足 $ac=bc$，即 $\{0,c,2c,\ldots,(m-1)c\}$ 两两不同。
  $ac=bc \implies (a-b)c=0 \implies a-b=0\cdot c^{-1} \implies a=b$。

简化剩余系 – 性质

由于逆元的存在性，简化剩余系有着很好的性质。

- 如果 a 在缩系中，则 $-a$ 也在缩系中，因为 $(-a)\cdot(-a^{-1})=1$。
- 缩系中任意剩余类 a 均可以通过一次与缩系中的剩余类 c 的乘法得到任意缩系中的剩余类 b，即 $ax=b$ 总是在缩系中有解，取 $x=b/a$ 即可。
- 对于任意缩系中的剩余类 c，不存在两个不同的（完系中的）剩余类 a,b 满足 $ac=bc$，即 $\{0,c,2c,\ldots,(m-1)c\}$ 两两不同。
  $ac=bc \implies (a-b)c=0 \implies a-b=0\cdot c^{-1} \implies a=b$。
- 上一条说明取定剩余类 c 后，映射 $a\mapsto ac$ 形成完系到自身的一个双射，即置换。换句话说，$\{0,c,2c,\ldots,(m-1)c\}$ 仍然构成完系。同时，由于缩系对乘法封闭，也形成缩系的一个置换。

当前进度

1 模 m 剩余类与剩余系

2 加法结构

3 乘法结构（上）
- 简化剩余系与 Euler 函数
  - 简化剩余系
  - Euler 函数
- Euler 定理与 Fermat 小定理

4 乘法结构（中）

5 附录

Euler 函数 – 定义

定义 (Euler 函数)

模 m 简化剩余系的大小称为 m **的 Euler 函数**，记作 $\varphi(m)$。

Euler 函数 – 定义

定义 (Euler 函数)

模 m 简化剩余系的大小称为 **m 的 Euler 函数**，记作 $\varphi(m)$。

一个等价的定义是，$\varphi(m)$ 为 $[0, m-1]$ 中与 m 互素的整数的个数。

Euler 函数 – 定义

定义 (Euler 函数)

模 m 简化剩余系的大小称为 **m 的 Euler 函数**，记作 $\varphi(m)$。

一个等价的定义是，$\varphi(m)$ 为 $[0, m-1]$ 中与 m 互素的整数的个数。

例 (Euler 函数)

- $\varphi(1) = 1$
- $\varphi(2) = 1$
- $\varphi(3) = 2$
- $\varphi(4) = 2$
- $\varphi(5) = 4$
- $\varphi(6) = 2$
- $\varphi(7) = 6$
- $\varphi(90) = 24$
- $\varphi(315) = 144$

可以看出，除了 $m = 1, 2$ 外，$\varphi(m)$ 均为偶数，因为剩余类恰好通过 a 与 $-a$ 两两配对（当 m 为 ≥ 4 的偶数时，$m/2$ 不在缩系中)。

Euler 函数 – 性质

- 当 p 为素数时，$\varphi(p) = p - 1$：
  除了 0 不与 p 互素外，$[1, p-1]$ 中的整数均与 p 互素。

Euler 函数 – 性质

- 当 p 为素数时，$\varphi(p) = p - 1$：
  除了 0 不与 p 互素外，$[1, p-1]$ 中的整数均与 p 互素。
- 当 p 为素数、α 为正整数时，$\varphi(p^\alpha) = (p-1) \cdot p^{\alpha-1}$：
  与 p^α 互素当且仅当不为 p 的倍数，显然 $[0, p^\alpha - 1]$ 中恰有 $p^{\alpha-1}$ 个 p 的倍数，扣除这些数即可。

Euler 函数 – 性质

- 当 p 为素数时，$\varphi(p) = p - 1$：
  除了 0 不与 p 互素外，$[1, p - 1]$ 中的整数均与 p 互素。
- 当 p 为素数、α 为正整数时，$\varphi(p^\alpha) = (p - 1) \cdot p^{\alpha-1}$：
  与 p^α 互素当且仅当不为 p 的倍数，显然 $[0, p^\alpha - 1]$ 中恰有 $p^{\alpha-1}$ 个 p 的倍数，扣除这些数即可。
- 当整数 a, b 互素（$a \perp b$）时，$\varphi(ab) = \varphi(a) \cdot \varphi(b)$，这个性质称为积性，即 Euler 函数是一个**积性函数**。
  此性质将在课件三中证明。

Euler 函数 – 计算

由前三个性质，可以推导出更多能够帮助我们进行计算的性质：

Euler 函数 – 计算

由前三个性质，可以推导出更多能够帮助我们进行计算的性质：

- 设 n 的标准分解式为 $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$，则

$$\varphi(n) = \prod_{i=1}^{k} (p_i - 1) \cdot p_i^{\alpha_i - 1} = n \cdot \prod_{i=1}^{k} \frac{p_i - 1}{p_i};$$

由 Euler 函数的积性，有 $\varphi(n) = \prod_{i=1}^{k} \varphi(p_i^{\alpha_i})$，再由性质展开 $\varphi(p_i^{\alpha_i})$ 即可。

Euler 函数 – 计算

由前三个性质，可以推导出更多能够帮助我们进行计算的性质：

- 设 n 的标准分解式为 $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$，则

$$\varphi(n) = \prod_{i=1}^{k} (p_i - 1) \cdot p_i^{\alpha_i - 1} = n \cdot \prod_{i=1}^{k} \frac{p_i - 1}{p_i};$$

  由 Euler 函数的积性，有 $\varphi(n) = \prod_{i=1}^{k} \varphi(p_i^{\alpha_i})$，再由性质展开 $\varphi(p_i^{\alpha_i})$ 即可。

- 当 p 为素数、i 为正整数，$\varphi(p \cdot i) = \begin{cases} (p-1) \cdot \varphi(i) & , p \nmid i \\ p \cdot \varphi(i) & , p \mid i \end{cases}$

  由上一条性质容易得到。

Euler 函数 – 计算

由前三个性质，可以推导出更多能够帮助我们进行计算的性质：

- 设 n 的标准分解式为 $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$，则
  $$\varphi(n) = \prod_{i=1}^{k} (p_i - 1) \cdot p_i^{\alpha_i - 1} = n \cdot \prod_{i=1}^{k} \frac{p_i - 1}{p_i}\text{:}$$
  由 Euler 函数的积性，有 $\varphi(n) = \prod_{i=1}^{k} \varphi(p_i^{\alpha_i})$，再由性质展开 $\varphi(p_i^{\alpha_i})$ 即可。
- 当 p 为素数、i 为正整数，$\varphi(p \cdot i) = \begin{cases} (p-1) \cdot \varphi(i) & , p \nmid i \\ p \cdot \varphi(i) & , p \mid i \end{cases}$

  由上一条性质容易得到。

可以根据这两条性质计算 Euler 函数：第一条指出使用标准分解式即可求值，第二条给出在筛法中求值的简便方法。

Euler 函数 – 使用标准分解式求值 – C++ 代码

若使用标准分解式，只需修改素因数分解的试除法的代码：

```
int Euler_phi(int n) {
  int phi = n;
  for (int d = 2; d * d <= n; ++d)
    if (n % d == 0) {
      int p = d;
      phi = phi / p * (p - 1);
      while (n % p == 0)
        n /= p;
    }
  if (n != 1)
    phi = phi / n * (n - 1);
  return phi;
}
```

时间复杂度与素因数分解相同。

Euler 函数 – 使用筛法求值 – C++ 代码

若使用筛法，则可以一次性求出 $[1, n]$ 内所有数的 Euler 函数值：
（由于代码行数过多，使用 4 空格缩进代替花括号。）

```
bool is_composite[MaxN]; int Euler_phi[MaxN];
void sieve_of_Euler(int n)
    std::vector<int> primes;
    for (int i = 2; i <= n; ++i)
        if (!is_composite[i]) primes.push_back(i);
        for (int p : primes)
            int k = p * i;
            if (k > n) break;
            is_composite[k] = true;
            if (i % p != 0)
                Euler_phi[k] = (p - 1) * Euler_phi[i];
            else
                Euler_phi[k] = p * Euler_phi[i];
                break;
```

时间复杂度为 $\mathcal{O}(n)$。

Euler 函数 – 例题 – HAOI 2012《外星人》

洛谷 P2350

以标准分解式的形式给出 $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$，求出使得 $\varphi^{(x)}(n) = 1$ 成立的最小正整数 x。其中 $\varphi^{(x)}(n)$ 表示 φ 嵌套 x 次，即 $\varphi(\varphi(\cdots \varphi(n) \cdots))$。

- $k \le 2000$，$p_i \le 10^5$，$\alpha_i \le 10^9$。

样例：当 $n = 2^2 \cdot 3 = 12$ 时，$\varphi^{(3)}(12) = \varphi^{(2)}(4) = \varphi(2) = 1$。
注意：n 可能非常大，素因数的范围可以接受，但是幂次非常高。

Euler 函数 – 例题 – HAOI 2012《外星人》

洛谷 P2350

以标准分解式的形式给出 $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$，求出使得 $\varphi^{(x)}(n) = 1$ 成立的最小正整数 x。其中 $\varphi^{(x)}(n)$ 表示 φ 嵌套 x 次，即 $\varphi(\varphi(\cdots\varphi(n)\cdots))$。

- $k \le 2000$，$p_i \le 10^5$，$\alpha_i \le 10^9$。

样例：当 $n = 2^2 \cdot 3 = 12$ 时，$\varphi^{(3)}(12) = \varphi^{(2)}(4) = \varphi(2) = 1$。
注意：n 可能非常大，素因数的范围可以接受，但是幂次非常高。

提示：从标准分解式的角度解释为什么当 $n \ge 3$ 时 $\varphi(n)$ 为偶数。

Euler 函数 – 例题 – HAOI 2012《外星人》– 题解

当 $n \geq 3$ 时 $\varphi(n)$ 为偶数的原因是：每个奇素因数 p 都会向 φ 值中贡献一个 $p-1$，为偶数，当 $n = 2^\alpha$ 没有奇素因数时，也必有 $\alpha \geq 2$ 导致 $\varphi(n) = 2^{\alpha-1}$ 为偶数。

Euler 函数 – 例题 – HAOI 2012《外星人》– 题解

这启发我们观察在迭代过程中，标准分解式中 2 的幂次是如何变化的。只要 n 为偶数，根据 $\frac{\varphi(n)}{n} = \prod \frac{p-1}{p}$，可以看作删去了一个 2，但又通过 $\prod(p-1)$ 添加了若干（也可能没有）2 作为素因数。

Euler 函数 – 例题 – HAOI 2012《外星人》– 题解

只要 n 为偶数，根据 $\frac{\varphi(n)}{n} = \prod \frac{p-1}{p}$，可以看作删去了一个 2，但又通过 $\prod(p-1)$ 添加了若干（也可能没有）2 作为素因数。

由于直到最后一步 $2 \to 1$ 时每一步都恰好如上删去一个 2，可以转而统计在过程中总共获得了多少个 2（也包括初始时已有的 2），获得的个数即为迭代步数。
当 n 为奇数时，由于第一步迭代没有删去 2，故步数多一次。

Euler 函数 – 例题 – HAOI 2012《外星人》– 题解

只要 n 为偶数，根据 $\frac{\varphi(n)}{n} = \prod \frac{p-1}{p}$，可以看作删去了一个 2，但又通过 $\prod(p-1)$ 添加了若干（也可能没有）2 作为素因数。

转而统计在过程中总共获得了多少个 2（也包括初始时已有的 2），获得的个数即为迭代步数。
当 n 为奇数时，由于第一步迭代没有删去 2，故步数多一次。

由于每个素因数最终都必然被拆分为一个个 2，获得的 2 的个数之和关于每个素因数独立。将通过 n 能获得的 2 的个数记作 $f(n)$，则有 $f(2) = 1$、$f(p) = p - 1$、以及 $f(p \cdot i) = f(p) + f(i)$，据此可以 Euler 筛。

Euler 函数 – 例题 – HAOI 2012《外星人》– 题解

只要 n 为偶数，根据 $\frac{\varphi(n)}{n} = \prod \frac{p-1}{p}$，可以看作删去了一个 2，但又通过 $\prod(p-1)$ 添加了若干（也可能没有）2 作为素因数。

转而统计在过程中总共获得了多少个 2（也包括初始时已有的 2），获得的个数即为迭代步数。
当 n 为奇数时，由于第一步迭代没有删去 2，故步数多一次。

将通过 n 能获得的 2 的个数记作 $f(n)$，则有 $f(2) = 1$、$f(p) = p - 1$、以及 $f(p \cdot i) = f(p) + f(i)$，据此可以 Euler 筛。

答案为 $\left(\sum_{i=1}^{k} f(p_i) \cdot \alpha_i\right) + [p_1 \neq 2]$。

Euler 函数 – 例题 – HAOI 2012《外星人》– 回顾

Euler 函数的迭代在后文中还会见到。
关于它的一个重要结论是对于所有偶数 n 有 $\varphi(n) \le \frac{n}{2}$，并且对于所有奇数 n 有 $\varphi(n)$ 为偶数。
故最多第一次变为偶数后，每次迭代数值都减半，然后在 $\log_2 n$ 步内到达 1。换句话说，迭代步数 $< \log_2 n + 1$。

Euler 函数 – 例题 – HAOI 2012《外星人》– 回顾

Euler 函数的迭代在后文中还会见到。
关于它的一个重要结论是对于所有偶数 n 有 $\varphi(n) \le \frac{n}{2}$，并且对于所有奇数 n 有 $\varphi(n)$ 为偶数。
故最多第一次变为偶数后，每次迭代数值都减半，然后在 $\log_2 n$ 步内到达 1。换句话说，迭代步数 $< \log_2 n + 1$。

事实上，我们有 $\log_3(n/2) + 1 \le$ 迭代步数 $< \log_2 n + 1$。
左侧在 $2 \cdot 3^\alpha$ 处取到等号。

Euler 函数 – 例题 – HAOI 2012《外星人》– 回顾

Euler 函数的迭代在后文中还会见到。
关于它的一个重要结论是对于所有偶数 n 有 $\varphi(n) \le \frac{n}{2}$，并且对于所有奇数 n 有 $\varphi(n)$ 为偶数。
故最多第一次变为偶数后，每次迭代数值都减半，然后在 $\log_2 n$ 步内到达 1。换句话说，迭代步数 $< \log_2 n + 1$。

事实上，我们有 $\log_3(n/2) + 1 \le$ 迭代步数 $< \log_2 n + 1$。
左侧在 $2 \cdot 3^\alpha$ 处取到等号。
若不存在更多的 Fermat 素数[1]，右侧可改为
$\le \log_2 n + (9 - \log_2 257) \approx \log_2 n + 0.994375$ 并在 257 处取到等号。

[1] 即 $2^{2^k} + 1$ 型素数

当前进度

Euler 定理

Euler 函数可不只有能表示缩系大小那么简单，接下来介绍**Euler 定理（Euler's theorem）**。

Euler 定理

Euler 函数可不只有能表示缩系大小那么简单，接下来介绍**Euler 定理（Euler's theorem）**。

定理 (Euler 定理)

对于任意正整数 m 和任意与 m 互素的整数 a，均有 $a^{\varphi(m)} \equiv 1$。
使用剩余类的语言来说，对于任意模 m 简化剩余系中的剩余类 a，均有 $a^{\varphi(m)} = 1$。

Euler 定理

Euler 函数可不只有能表示缩系大小那么简单，接下来介绍**Euler 定理（Euler's theorem）**。

定理 (Euler 定理)

对于任意正整数 m 和任意与 m 互素的整数 a，均有 $a^{\varphi(m)} \equiv 1$。
使用剩余类的语言来说，对于任意模 m 简化剩余系中的剩余类 a，均有 $a^{\varphi(m)} = 1$。

例 (Euler 定理)

- $2^6 \equiv 1 \pmod 7$
- $5^{256} \equiv 1 \pmod{512}$
- $3^{40} \equiv 1 \pmod{100}$
- $0^1 \equiv 1 \pmod 1$

Euler 定理 – 证明

Euler 定理.

前文中提到 $b \mapsto ab$ 形成缩系的一个置换，记缩系中全体元素为 $\{b_1, b_2, \ldots, b_{\varphi(m)}\}$，则 $\{a \cdot b_1, a \cdot b_2, \ldots, a \cdot b_{\varphi(m)}\}$ 恰好也形成缩系。

求乘积得到 $\prod_{i=1}^{\varphi(m)} b_i = \prod_{i=1}^{\varphi(m)} ab_i$

$$= a^{\varphi(m)} \prod_{i=1}^{\varphi(m)} b_i。$$

消去两侧的 $\prod_{i=1}^{\varphi(m)} b_i$，即得 $a^{\varphi(m)} = 1$。 □

当前进度

1 模 m 剩余类与剩余系

2 加法结构

3 乘法结构（上）

- 简化剩余系与 Euler 函数
- Euler 定理与 Fermat 小定理
  - Euler 定理
  - Fermat 小定理
  - 应用

4 乘法结构（中）

5 附录

Fermat 小定理

定理 (Fermat 小定理)

对于任意素数 p 和任意不为 p 的倍数的整数 a，均有 $a^{p-1} \equiv 1$。
使用剩余类的语言来说，对于任意非零剩余类 a，均有 $a^{p-1} = 1$。

有时，*Fermat* 小定理也使用“对任意素数 p 和任意整数 a，均有 $p \mid (a^p - a)$”这一形式表述。

Fermat 小定理

定理 (Fermat 小定理)

对于任意素数 p 和任意不为 p 的倍数的整数 a，均有 $a^{p-1} \equiv 1$。
使用剩余类的语言来说，对于任意非零剩余类 a，均有 $a^{p-1} = 1$。

有时，*Fermat* 小定理也使用"对任意素数 p 和任意整数 a，均有 $p \mid (a^p - a)$"这一形式表述。

例 (Fermat 小定理)

- $2^6 \equiv 1 \pmod 7$
- $10^{16} \equiv 1 \pmod{17}$
- $2^{22} \equiv 1 \pmod{23}$
- $1^{196} \equiv 1 \pmod{197}$

Fermat 小定理 – 证明

Fermat 小定理.

对于 $a^{p-1} \equiv 1$ 的形式，Euler 定义取 m 为素数的特例即可。
对于 $p \mid (a^p - a)$ 的形式，只需额外考虑 a 为 p 的倍数的情况。 $\square$

Fermat 小定理 – 证明

Fermat 小定理.

对于 $a^{p-1} \equiv 1$ 的形式，Euler 定义取 m 为素数的特例即可。
对于 $p \mid (a^p - a)$ 的形式，只需额外考虑 a 为 p 的倍数的情况。 $\square$

可以发现，Fermat 小定理完全是 Euler 定理的特例。
不过，Fermat 小定理的优势在于，不需要考虑求 φ 值，但前提是模数为素数。

当前进度

1 模 m 剩余类与剩余系

2 加法结构

3 乘法结构（上）

- 简化剩余系与 Euler 函数
- Euler 定理与 Fermat 小定理
  - Euler 定理
  - Fermat 小定理
  - 应用

4 乘法结构（中）

5 附录

高精度指数快速幂

求逆元

4 乘法结构（中）

未完成

此部分未完成，后续更新将在 算法竞赛中的数论 – 系列课件 – GitHub/GitPinkRabbit 中上传。

5 附录

参考文献与致谢

- OI Wiki，https://oi-wiki.org/
- Wikipedia，https://en.wikipedia.org/
- 初等数论学习笔记 I：同余相关，Alex_Wei，https://www.cnblogs.com/alex-wei/p/Number_Theory.html

- 本课件编写时的哔哩哔哩直播间观众