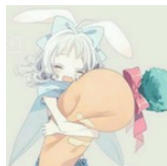


数论及其应用

hzwer,miskcoo

北京大学, 清华大学

2016 年 9 月 24 日



① 数论基础

整除的概念

最大公因数及欧几里得算法

线性方程

同余

素数及其判定

Eratosthenes 篩法

线性筛法

② 数学基础算法

③ 组合数学基础

整除的概念

整除的概念

最大公因数及欧几里得算法

线性方程

同余

素数及其判定

Eratosthenes 篩法

线性筛法

Downloaded from <http://ajph.org/> at University of California, San Diego on June 11, 2015

整除的概念

首先是整除的概念，假设 m 和 n 是整数，并且 $m \neq 0$ 。那么 m **整除** n 指的是 n 是 m 的倍数。即存在整数 k 满足 $n=mk$ 。

如果 m 整除 n ，那么记为 $m \mid n$ 。如果 m 不整除 n ，可以记为 $m \nmid n$ 。

整除的概念

首先是整除的概念，假设 m 和 n 是整数，并且 $m \neq 0$ 。那么 m **整除** n 指的是 n 是 m 的倍数。即存在整数 k 满足 $n=mk$ 。

如果 m 整除 n ，那么记为 $m \mid n$ 。如果 m 不整除 n ，可以记为 $m \nmid n$ 。

例如，由于 3 整除 18，那么我们可以写成 $3 \mid 18$ ，又由于 3 不整除 17，那么我们可以写成 $3 \nmid 17$ 。

① 数论基础

整除的概念

最大公因数及欧几里得算法

线性方程

同余

素数及其判定

Eratosthenes 篩法

线性筛法

② 数学基础算法

③ 组合数学基础

欧几里得算法

首先我们来证明一个式子：

$$\gcd(m, n) = \gcd(n, m \bmod n)$$

欧几里得算法

首先我们来证明一个式子：

$$\gcd(m, n) = \gcd(n, m \bmod n)$$

设 $g = \gcd(m, n)$ ， $r = m \bmod n$ ，我们可以知道 $m = kn + r$ ，再根据 g 的最大公因数的性质（也就是同时整除 m 和 n ）就可以知道 g 必然也整除 r 。同样的，我们也可以证明， g 是同时整除 r 和 n 的最大的整数，这样，这个等式就成立了。

欧几里得算法

首先我们来证明一个式子：

$$\gcd(m, n) = \gcd(n, m \bmod n)$$

设 $g = \gcd(m, n)$, $r = m \bmod n$, 我们可以知道 $m = kn + r$, 再根据 g 的最大公因数的性质 (也就是同时整除 m 和 n) 就可以知道 g 必然也整除 r 。同样的, 我们也可以证明, g 是同时整除 r 和 n 的最大的整数, 这样, 这个等式就成立了。

欧几里得算法就是不断地递归应用这个等式, 因为每次计算后 $m \bmod n$ 一定比 n 要小, 也就是两个数中较大的那一个在不断减小, 这样最后当其中一个先变成 0 时, 另一个数就是答案。

它的复杂度也很低, 是 $\mathcal{O}(\log m)$

代码实现

```
miskcoo@miskcoo:~  
1 int gcd(int a, int b)  
2 {  
3     if(b == 0) return a;  
4     else return gcd(b, a % b);  
5 }  
  
1,1 All
```

更相减损术

第一步：任意给定两个正整数；判断它们是否都是偶数。若是，则用 2 约简；若不是则执行第二步。

第二步：以较大的数减较小的数，接着把所得的差与较小的数比较，并以大数减小数。继续这个操作，直到所得的减数和差相等为止。

1 数论基础

整除的概念

最大公因数及欧几里得算法

线性方程

同余

素数及其判定

Eratosthenes 筛法

线性筛法

2 数学基础算法

3 组合数学基础

4 例题

裴蜀定理

Theorem (裴蜀定理)

若 a 和 b 是整数，方程 $ax+by=d$ 有整数解当且仅当 $\gcd(a, b) \mid d$ 。

例如，方程 $3x+6y=2$ 就不存在整数解，方程 $3x+6y=3$ 存在（无数多个）整数解，其中一个是 $x=1, y=0$ 。

裴蜀定理

Theorem (裴蜀定理)

若 a 和 b 是整数，方程 $ax+by=d$ 有整数解当且仅当 $\gcd(a, b) \mid d$ 。

例如，方程 $3x+6y=2$ 就不存在整数解，方程 $3x+6y=3$ 存在（无数多个）整数解，其中一个是 $x=1, y=0$ 。

这个定理给我们了一个判定形如 $ax+by=d$ 的方程是否有整数解的方法，但是它并没有告诉我们如何求解。求解这样的方程是**扩展欧几里得算法**的内容。

扩展欧几里得算法

现在我们要来求方程 $ax + by = \gcd(a, b)$ 的整数解，首先根据裴蜀定理我们知道它肯定存在解。此外，按照先前计算最大公因数时用到的方程， $bx' + (a \bmod b)y' = \gcd(a, b)$ 同样存在整数解。

扩展欧几里得算法

现在我们要来求方程 $ax + by = \gcd(a, b)$ 的整数解，首先根据裴蜀定理我们知道它肯定存在解。此外，按照先前计算最大公因数时用到的方程， $bx' + (a \bmod b)y' = \gcd(a, b)$ 同样存在整数解。

同时， $a \bmod b = a - b\lfloor \frac{a}{b} \rfloor$ ，再将上面两个方程联立可以得到

$$ax + by = bx' + (a - b\lfloor \frac{a}{b} \rfloor)y'$$

扩展欧几里得算法

现在我们要来求方程 $ax + by = \gcd(a, b)$ 的整数解，首先根据裴蜀定理我们知道它肯定存在解。此外，按照先前计算最大公因数时用到的方程， $bx' + (a \bmod b)y' = \gcd(a, b)$ 同样存在整数解。

同时， $a \bmod b = a - b \lfloor \frac{a}{b} \rfloor$ ，再将上面两个方程联立可以得到

$$ax + by = bx' + (a - b \lfloor \frac{a}{b} \rfloor)y'$$

按照 a 和 b 进行合并之后

$$ax + by = ay' + b(x' - \lfloor \frac{a}{b} \rfloor y')$$

扩展欧几里得算法

如果已经知道了 (x', y') ，那么在满足下式的情况下方程成立。

$$\begin{cases} x = y' \\ y = x' - \lfloor \frac{a}{b} \rfloor y' \end{cases}$$

扩展欧几里得算法

如果已经知道了 (x', y') ，那么在满足下式的情况下方程成立。

$$\begin{cases} x = y' \\ y = x' - \lfloor \frac{a}{b} \rfloor y' \end{cases}$$

这样就可以利用方程 $bx' + (a \bmod b)y' = \gcd(a, b)$ 的整数解 (x', y') 来计算出方程 $ax + by = \gcd(a, b)$ 的整数解了。

扩展欧几里得算法

如果已经知道了 (x', y') ，那么在满足下式的情况下方程成立。

$$\begin{cases} x = y' \\ y = x' - \lfloor \frac{a}{b} \rfloor y' \end{cases}$$

这样就可以利用方程 $bx' + (a \bmod b)y' = \gcd(a, b)$ 的整数解 (x', y') 来计算出方程 $ax + by = \gcd(a, b)$ 的整数解了。

最后只要按照前面计算最大用因数的方法，在递归的过程中加入 x 和 y 的计算就可以了。当递归到末尾的时候 b 变为 0，这是方程有整数解 $x = 1, y = 0$ 。

代码实现

```
miskcoo@miskcoo:~  
1 int exgcd(int a, int b, int& x, int& y)  
2 {  
3     if(b == 0)  
4     {  
5         x = 1, y = 0;  
6         return a;  
7     }  
8  
9     int g = exgcd(b, a % b, x, y);  
10    int t = x;  
11    x = y;  
12    y = t - a / b * y;  
13    return g;  
14 }
```

1,1 All

1 数论基础

整除的概念

最大公因数及欧几里得算法

线性方程

同余

素数及其判定

Eratosthenes 筛法

线性筛法

2 数学基础算法

3 组合数学基础

4 例题

同余

接下来是关于同余的概念，它提供了一种描述整除性质的简便方法。

如果 m 整除 $a - b$ ，我们就说 a 与 b 模 m **同余**并记为

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

简单来说，就是它们模 m 后的余数相同就可以记成这样。

同余的性质

如果

$$a_1 \equiv b_1 \pmod{m}$$

$$a_2 \equiv b_2 \pmod{m}$$

那么

$$a_1 \pm a_2 \equiv b_1 \pm b_2 \pmod{m}$$

$$a_1 a_2 \equiv b_1 b_2 \pmod{m}$$

NOIP2012. 同余方程

求关于 x 的同余方程 $ax \equiv 1 \pmod{b}$ 的最小正整数解。

其中 $0 \leq a, b \leq 2 \times 10^9$ ，并且保证该方程有解。

NOIP2012. 同余方程

求关于 x 的同余方程 $ax \equiv 1 \pmod{b}$ 的最小正整数解。

其中 $0 \leq a, b \leq 2 \times 10^9$ ，并且保证该方程有解。

如果上述同余方程被满足的话，一定存在整数 y 使得 $ax = 1 + by$ ，这样我们可以直接利用扩展欧几里得算法得出一个解。至于最小正整数解也是可以很容易就计算得出，因为在 $1 \leq x \leq b$ 中这个方程有唯一解。

NOIP2014. 解方程

已知多项式方程：

$$a_0 + a_1x + a_2x^2 + \cdots + a_nx^n = 0$$

求这个方程在 $[1, m]$ 内的整数解。

其中 n 和 m 均为正整数。

NOIP2014. 解方程

已知多项式方程：

$$a_0 + a_1x + a_2x^2 + \cdots + a_nx^n = 0$$

求这个方程在 $[1, m]$ 内的整数解。

其中 n 和 m 均为正整数。

$1 \leq n \leq 100, |a_i| \leq 10^{10000}, a_n \neq 0, 1 \leq m \leq 10^6$ 。

NOIP2014. 解方程

暴力验证 $[1, m]$ 内的整数？

NOIP2014. 解方程

暴力验证 $[1, m]$ 内的整数？

如何验证？系数太大了……

NOIP2014. 解方程

暴力验证 $[1, m]$ 内的整数？

如何验证？系数太大了……

取模！

NOIP2014. 解方程

暴力验证 $[1, m]$ 内的整数？

如何验证？系数太大了……

取模！

如果 $x = x_0$ 的时候这个方程成立，那么将其两边对任意数 p 取模后仍然成立。但是反之则不一定。

NOIP2014. 解方程

然而……一个个验证要验证的个数太多了……

NOIP2014. 解方程

然而……一个个验证要验证的个数太多了……

但是方程的解不会超过 n 个！

NOIP2014. 解方程

然而……一个个验证要验证的个数太多了……

但是方程的解不会超过 n 个！

如果 $x = x_0$ 的时候，在对 p 取模后方程不成立，那不原方程必然不成立！

NOIP2014. 解方程

然而……一个个验证要验证的个数太多了……

但是方程的解不会超过 n 个！

如果 $x = x_0$ 的时候，在对 p 取模后方程不成立，那不原方程必然不成立！

用比较小的质数验证几个，先排除大部分！

分解质因数

给定一个数 n ，求其的质因数分解，有 T 组数据。

分解质因数

给定一个数 n ，求其的质因数分解，有 T 组数据。

对于 20% 的数据， $n \leq 1000000$, $T \leq 100000$

分解质因数

给定一个数 n ，求其的质因数分解，有 T 组数据。

对于 20% 的数据， $n \leq 1000000$, $T \leq 100000$

对于 50% 的数据， $n \leq 10000000$, $T \leq 100000$

分解质因数

给定一个数 n ，求其的质因数分解，有 T 组数据。

对于 20% 的数据， $n \leq 1000000$, $T \leq 100000$

对于 50% 的数据， $n \leq 10000000$, $T \leq 100000$

对于 100% 的数据， $n \leq 30000000$, $T \leq 1000000$

分解质因数

给定一个数 n ，求其的质因数分解，有 T 组数据。

对于 20% 的数据， $n \leq 1000000, T \leq 100000$

对于 50% 的数据， $n \leq 10000000, T \leq 100000$

对于 100% 的数据， $n \leq 30000000, T \leq 1000000$

乘法逆元

在模 p 意义下， x 的**乘法逆元** x^{-1} 满足如下同余方程：

$$xx^{-1} \equiv 1 \pmod{p}$$

它的存在性及求法分别可以根据裴蜀定理和扩展欧几里得算法得出，我们现在来介绍一下它的用途。

乘法逆元

在模 p 意义下， x 的**乘法逆元** x^{-1} 满足如下同余方程：

$$xx^{-1} \equiv 1 \pmod{p}$$

它的存在性及求法分别可以根据裴蜀定理和扩展欧几里得算法得出，我们现在来介绍一下它的用途。

在同余意义下，加减法和乘法都和普通的运算没什么区别，但是唯独“除法”有一些区别：当没有逆元时无法进行“除法”！

乘法逆元

在模 p 意义下， x 的**乘法逆元** x^{-1} 满足如下同余方程：

$$xx^{-1} \equiv 1 \pmod{p}$$

它的存在性及求法分别可以根据裴蜀定理和扩展欧几里得算法得出，我们现在来介绍一下它的用途。

在同余意义下，加减法和乘法都和普通的运算没什么区别，但是唯独“除法”有一些区别：当没有逆元时无法进行“除法”！

因为在这里如果你想要在方程两边除以 x ，你实际上需要做的事是在方程两边乘以 x^{-1} 。如果没有逆元就想进行除法计算是会出现问题的：

$$15 \times 2 \equiv 20 \times 2 \pmod{10}$$

在模 10 意义下，2 是没有乘法逆元的，上面这个方程成立，但如果你想要两边同时消去 2，我们得到的就是 15 和 20 模 10 同余，这显然是不可能的！

1 数论基础

整除的概念

最大公因数及欧几里得算法

线性方程

同余

素数及其判定

Eratosthenes 筛法

线性筛法

2 数学基础算法

3 组合数学基础

4 例题

素数及基本知识

素数是只含有 1 及其本身两个正因子的数，也称为**质数**。如果还有其它正因子的话，那么这个数就被称为**合数**。注意 1 并非素数，亦非合数。

我在这里介绍一个关于素数的定理，它们在算法复杂度分析中或许会用到：

素数及基本知识

素数是只含有 1 及其本身两个正因子的数，也称为**质数**。如果还有其它正因子的话，那么这个数就被称为**合数**。注意 1 并非素数，亦非合数。

我在这里介绍一个关于素数的定理，它们在算法复杂度分析中或许会用到：

Theorem (素数定理)

当 x 很大时，小于 x 的素数个数近似等于 $\frac{x}{\ln x}$ 。

素数的判定

至于如何判定一个数 m 是不是素数，我们可以直接从定义出发，从 2 开始到 $m-1$ 为止，检测是否有一个数整除 m ，如果没有，那么这个数就是素数。这样做的复杂度是 $\mathcal{O}(m)$ 的。

素数的判定

至于如何判定一个数 m 是不是素数，我们可以直接从定义出发，从 2 开始到 $m-1$ 为止，检测是否有一个数整除 m ，如果没有，那么这个数就是素数。这样做的复杂度是 $\mathcal{O}(m)$ 的。

那么，是否有更好的方法呢？

素数的判定

至于如何判定一个数 m 是不是素数，我们可以直接从定义出发，从 2 开始到 $m-1$ 为止，检测是否有一个数整除 m ，如果没有，那么这个数就是素数。这样做的复杂度是 $\mathcal{O}(m)$ 的。

那么，是否有更好的方法呢？

如果 d 整除 m ，那么我们可以肯定 $\frac{m}{d}$ 也整除 m ，并且 d 和 $\frac{m}{d}$ 一定有一个是不超过 \sqrt{m} 的。

这样就可以得到一个新的算法，我们只需要从 2 开始一直检测到 $\lceil \sqrt{m} \rceil$ 就可以结束，因为如果到这里为止还没有出现因子的话之后也不会出现。这个算法的复杂度是 $\mathcal{O}(\sqrt{m})$ 。已经是一个不错的算法了。

1 数论基础

整除的概念

最大公因数及欧几里得算法

线性方程

同余

素数及其判定

Eratosthenes 筛法

线性筛法

2 数学基础算法

3 组合数学基础

4 例题

Eratosthenes 筛法

Eratosthenes **筛法**是一种用来求素数的方法，它的思路比较简单。

由于每个合数都可以被分解成几个素数的乘积，如果我们将所有素数的倍数都删去，那么剩下的就是素数了。

Eratosthenes 筛法

Eratosthenes **筛法**是一种用来求素数的方法，它的思路比较简单。

由于每个合数都可以被分解成几个素数的乘积，如果我们将所有素数的倍数都删去，那么剩下的就是素数了。

因此，我们可以从 2 开始，先将 2 的所有倍数都删去。然后往下找到第一个没有被删去的数，这个数一定是素数，再将这个数的所有倍数都删去，不断进行这个操作。

Eratosthenes 筛法

Eratosthenes **筛法**是一种用来求素数的方法，它的思路比较简单。

由于每个合数都可以被分解成几个素数的乘积，如果我们将所有素数的倍数都删去，那么剩下的就是素数了。

因此，我们可以从 2 开始，先将 2 的所有倍数都删去。然后往下找到第一个没有被删去的数，这个数一定是素数，再将这个数的所有倍数都删去，不断进行这个操作。

我们来分析一下它的复杂度，假设需要求 n 以内的素数，对于每个素数 p ，都会删去 $\frac{n}{p}$ 个数，那么一共就会删去大约

$$n\left(\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \cdots + \frac{1}{p_k}\right)$$

个数，在这里 p_k 是小于 n 的最大素数。

括号内的和大约是 $\log \log n$ ，因此，复杂度大约是 $\mathcal{O}(n \log \log n)$ 。

1 数论基础

整除的概念

最大公因数及欧几里得算法

线性方程

同余

素数及其判定

Eratosthenes 筛法

线性筛法

2 数学基础算法

3 组合数学基础

4 例题

线性筛法

线性筛法，又称**欧拉筛法**，是一种可以在 $\mathcal{O}(n)$ 时间内求出素数的方法。

尽管 Eratosthenes 筛法已经比较快了，但是它任然做了很多冗余的运算，比如说 6 就会重复被 2 和 3 剔除两次。而线性筛法则避免了这个问题，它保证了每一个合数只会被它最小的那个因子剔除。

线性筛法

这个算法从 2 开始，不断往下处理，并且记录下当前找到的所有素数。

然后每处理到一个数，将当前找到的素数与这个数的乘积剔除。直到这个数被某个素数整除为止。

```
miskcoo@miskcoo:~  
1 int prime[10000], tot;  
2 bool not_prime[10000];  
3  
4 void sieve(int n)  
5 {  
6     for(int i = 2; i <= n; ++i)  
7     {  
8         if(!not_prime[i])  
9             prime[tot++] = i;  
10  
11         for(int j = 0; j < tot && i * prime[j] <= n; ++j)  
12         {  
13             not_prime[i * prime[j]] = 1;  
14             if(i % prime[j] == 0) break;  
15         }  
16     }  
17 }
```

1,1 All

线性筛法

上面的第 14 行就保证了一个数不会被重复剔除。

由于当 $p_j | i$ 的时候就跳出了第二层循环，并且 p_j 是 i 的最小的素因子，这样就保证了上面被剔除的数是被它最小的素因子剔除的。

在这个过程中我们还可以顺便记录下一个数的最小素因子，这样之后就可以非常快速地计算出每个数的素因子分解。

② 数学基础算法

线性方程组和高斯消元

快速幂

快速幂是用来快速求解 a^b 的方法，它可以在 $O(\log b)$ 的时间内计算出答案。

这个算法是一个非常基本但是也非常重要的算法，现在我就来介绍一下这个算法。

快速幂的基本思想就是减少重复的计算，比如说

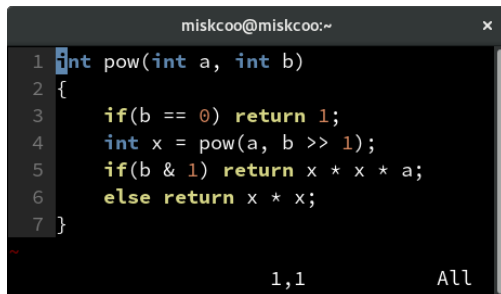
$$a^9 = a^4 a^4 a^1$$

算法介绍

快速幂的基本思想就是减少重复的计算，比如说

$$a^9 = a^4 a^4 a^1$$

我们可以利用递归的方法来完成这个：



```
miskcoo@miskcoo:~  
1 int pow(int a, int b)  
2 {  
3     if(b == 0) return 1;  
4     int x = pow(a, b >> 1);  
5     if(b & 1) return x * x * a;  
6     else return x * x;  
7 }  
  
1,1 All
```

算法介绍

还有一种非递归的写法：

首先对于指数 b ，它可以按照二进制来表示，我来用一个具体的例子来说明。比如现在要计算 a^5 。

把 5 写成二进制之后就是 101，换一种方法写出来就是 $2^2 + 2^0$ 。

首先对于指数 b ，它可以按照二进制来表示，我来用一个具体的例子来用。比如现在要计算 a^5 。

那么我们要计算的就是 $a^{2^2+2^0}$ ，如果把它展开一下，可以得到 $\times a^{2^0}$ 。

不管指数是什么，在分解成二进制像上面这样表示出来之后，我们会发现最后的答案是像 a^{2^k} 这样的数的乘积。

算法介绍

对于像 a^{2^k} 这样的数，我们可以从 $k = 0$ 开始不断递推。

算法介绍

当 $k = 0$ 时, $a^{2^0} = a_0$.

算法介绍

对于像 a^{2^k} 这样的数，我们可以从 $k = 0$ 开始不断递推。

当 $k = 0$ 时, $a^{2^0} = a$ 。

为了计算 $k = 1$ 时候的值，可以利用前面的结果，也就是 $a^{2^1} = a^{2^0} \times a^{2^0}$ 。

算法介绍

对于像 a^{2^k} 这样的数，我们可以从 $k = 0$ 开始不断递推。

当 $k = 0$ 时, $a^{2^0} = a$.

为了计算 $k = 1$ 时候的值，可以利用前面的结果，也就是 $a^{2^1} = a^{2^0} \times a^{2^0}$ 。

然后以此类推，每次将前面一个数平方后就可以得到下一个结果了。

再利用刚刚对指数的分解，再这一位是 1 的时候就在答案上乘上这个数。

代码实现

```
miskcoo@miskcoo:/tmp
1 int pow(int a, int b)
2 {
3     int v = 1;
4     while(b)
5     {
6         if(b & 1) v = v * a;
7         a = a * a;
8         b >>= 1;
9     }
10    return v;
11 }
12
```

1,1 Top

如果要求在模某个数的情况下计算，只需要在每次乘法计算完后取模即可。

线性方程组

在许多情况下，我们会需要解这样的方程组：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m = b_2 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_m = b_n \end{cases}$$

这样的方程组称为**线性方程组**。

线性方程组

通常在求解的时候我们会将这个线性方程组表示成矩阵的形式：

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1m} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2m} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} & b_n \end{array} \right)$$

这样的矩阵称为**增广矩阵**。

线性方程组

例如，如果我们有这样一个线性方程组：

$$\begin{cases} x_1 - 2x_2 + x_3 = 0 \\ 2x_2 - 8x_3 = 8 \\ -4x_1 + 5x_2 + 9x_3 = -9 \end{cases}$$

它的增广矩阵就可以表示成：

$$\left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 0 & 2 & -8 & 8 \\ -4 & 5 & 9 & -9 \end{array} \right)$$

增广矩阵的阶梯形式

$$\left(\begin{array}{cccc|c} * & ? & ? & ? & ? \\ 0 & * & ? & ? & ? \\ 0 & 0 & 0 & * & ? \end{array} \right)$$

增广矩阵的这种形式被称为**阶梯形式**。上面的“*”表示非零数，“?”表示任意数。

增广矩阵的阶梯形式

$$\left(\begin{array}{cccc|c} * & ? & ? & ? & ? \\ 0 & * & ? & ? & ? \\ 0 & 0 & 0 & * & ? \end{array} \right)$$

增广矩阵的这种形式被称为**阶梯形式**。上面的“*”表示非零数，“?”表示任意数。

这种形式的特征就是每一行第一个非零元的位置必须在前一行的右侧，并且每一行第一个非零元之后行的对应位置都应该是 0。

增广矩阵的阶梯形式

$$\left(\begin{array}{cccc|c} * & ? & ? & ? & ? \\ 0 & * & ? & ? & ? \\ 0 & 0 & 0 & * & ? \end{array} \right)$$

增广矩阵的这种形式被称为**阶梯形式**。上面的“*”表示非零数，“?”表示任意数。

这种形式的特征就是每一行第一个非零元的位置必须在前一行的右侧，并且每一行第一个非零元之后行的对应位置都应该是 0。

在阶梯形式下，线性方程组的解是比较容易求出的。

初等行变换

对这样的线性方程组的增广矩阵可以进行一些基本操作：

初等行变换

对这样的线性方程组的增广矩阵可以进行一些基本操作：

初等行变换

对这样的线性方程组的增广矩阵可以进行一些基本操作：

① 交换两行

初等行变换

对这样的线性方程组的增广矩阵可以进行一些基本操作：

- ① 交换两行
- ② 将一行的所有元素乘以一个非零常数

初等行变换

对这样的线性方程组的增广矩阵可以进行一些基本操作：

- ① 交换两行
- ② 将一行的所有元素乘以一个非零常数
- ③ 将某一行的元素加上另外一行对应位置的元素

初等行变换

对这样的线性方程组的增广矩阵可以进行一些基本操作：

- ① 交换两行
- ② 将一行的所有元素乘以一个非零常数
- ③ 将某一行的元素加上另外一行对应位置的元素

我们可以验证这些操作是不会改变这个线性方程组的解的。

高斯消元

高斯消元是不断利用初等行变换来解线性方程组的一个算法，它的目的是将一个增广矩阵化为阶梯形式。它的时间复杂度是 $\mathcal{O}(n^3)$ 。

高斯消元

高斯消元是不断利用初等行变换来解线性方程组的一个算法，它的目的是将一个增广矩阵化为阶梯形式。它的时间复杂度是 $\mathcal{O}(n^3)$ 。

首先从选中第一个不全为零的列，并且保证第一行这列的元素非零（必要时交换两行）。

高斯消元

高斯消元是不断利用初等行变换来解线性方程组的一个算法，它的目的是将一个增广矩阵化为阶梯形式。它的时间复杂度是 $\mathcal{O}(n^3)$ 。

首先从选中第一个不全为零的列，并且保证第一行这列的元素非零（必要时交换两行）。

然后利用上述初等行变换的操作 2 和操作 3 将接下来行该列的元素变为零。

高斯消元

高斯消元是不断利用初等行变换来解线性方程组的一个算法，它的目的是将一个增广矩阵化为阶梯形式。它的时间复杂度是 $\mathcal{O}(n^3)$ 。

首先从选中第一个不全为零的列，并且保证第一行这列的元素非零（必要时交换两行）。

然后利用上述初等行变换的操作 2 和操作 3 将接下来行该列的元素变为零。

最后从剩下的子矩阵开始继续进行这个操作，直到整个增广矩阵变成阶梯形式为止

高斯消元

还是拿刚刚那个线性方程组来举例子。

$$\left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 0 & 2 & -8 & 8 \\ -4 & 5 & 9 & -9 \end{array} \right)$$

高斯消元

还是拿刚刚那个线性方程组来举例子。

$$\left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 0 & 2 & -8 & 8 \\ -4 & 5 & 9 & -9 \end{array} \right)$$

首先将第一行乘以 4 再添加到第三行上可以得到

$$\left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 0 & 2 & -8 & 8 \\ 0 & -3 & 13 & -9 \end{array} \right)$$

高斯消元

然后将第二行乘以 $\frac{3}{2}$ 再加入到第三行上可以得到

$$\left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 0 & 2 & -8 & 8 \\ 0 & 0 & 1 & 3 \end{array} \right)$$

高斯消元

然后将第二行乘以 $\frac{3}{2}$ 再加入到第三行上可以得到

$$\left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 0 & 2 & -8 & 8 \\ 0 & 0 & 1 & 3 \end{array} \right)$$

这样的增广矩阵已经是阶梯形式了，它对应的线性方程组是

$$\begin{cases} x_1 - 2x_2 + x_3 = 0 \\ 2x_2 - 8x_3 = 8 \\ x_3 = 3 \end{cases}$$

然后最后只要从最下面一行开始不断回代就可以解出整个方程，在这里解是 $x_1 = 29, x_2 = 16, x_3 = 3$ 。

高斯消元

那么什么时候线性方程组会没有解呢？

高斯消元

那么什么时候线性方程组会没有解呢？

如果你高斯消元后得到的阶梯形式是这样的：

$$\left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 0 & 2 & -8 & 8 \\ 0 & 0 & 0 & 3 \end{array} \right)$$

高斯消元

那么什么时候线性方程组会没有解呢？

如果你高斯消元后得到的阶梯形式是这样的：

$$\left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 0 & 2 & -8 & 8 \\ 0 & 0 & 0 & 3 \end{array} \right)$$

此时，最后一个行表示的是方程 $0 = 3$ ，这显然是不可能的，因此这时候方程就没有解了。

高斯消元

那么什么时候线性方程组会有无穷多个解呢？

高斯消元

那么什么时候线性方程组会有无穷多个解呢？

如果你高斯消元后得到的阶梯形式是这样的：

$$\left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 0 & 0 & -8 & 8 \\ 0 & 0 & 0 & 0 \end{array} \right)$$

高斯消元

那么什么时候线性方程组会有无穷多个解呢？

如果你高斯消元后得到的阶梯形式是这样的：

$$\left(\begin{array}{ccc|c} 1 & -2 & 1 & 0 \\ 0 & 0 & -8 & 8 \\ 0 & 0 & 0 & 0 \end{array} \right)$$

就是变量的个数多于非零行的个数，这时候这个方程组就有无穷多个解了。因为你可以任意确定其中的一些变量，并且使得整个方程仍然有解。

JSOI2008. 球形空间产生器

有一个球形空间产生器能够在 n 维空间中产生一个坚硬的球体。现在，你被困在了这个 n 维球体中，你只知道球面上 $n+1$ 个点的坐标，你需要以最快的速度确定这个 n 维球体的球心坐标，以便于摧毁这个球形空间产生器。

JSOI2008. 球形空间产生器

有一个球形空间产生器能够在 n 维空间中产生一个坚硬的球体。现在，你被困在了这个 n 维球体中，你只知道球面上 $n+1$ 个点的坐标，你需要以最快的速度确定这个 n 维球体的球心坐标，以便于摧毁这个球形空间产生器。

我们可以设圆心的坐标是 (x_1, x_2, \dots, x_n) ，半径是 r 。之后可以列出 $n+1$ 个方程直接进行高斯消元。

1 数论基础

2 数学基础算法

3 组合数学基础

排列组合

二项式定理

4 例题

排列与组合

现在来考虑一个问题：你需要在 n 个不同的人里面选出 m 个人排成一行，问有多少种排列方案？

排列与组合

现在来考虑一个问题：你需要在 n 个不同的人里面选出 m 个人排成一行，问有多少种排列方案？

首先我们可以想到如果先选一个人排在第一个位置能有 n 种方案。

排列与组合

现在来考虑一个问题：你需要在 n 个不同的人里面选出 m 个人排成一行，问有多少种排列方案？

首先我们可以想到如果先选一个人排在第一个位置能有 n 种方案。

这时第一个位置已经被排完了，我们可以来选一个人排在第二个位置。因为前面已经用掉一个人了，所以还剩下 $n-1$ 个人，也就是有 $n-1$ 种方案来排第二个位置。

排列与组合

现在来考虑一个问题：你需要在 n 个不同的人里面选出 m 个人排成一行，问有多少种排列方案？

首先我们可以想到如果先选一个人排在第一个位置能有 n 种方案。

这时第一个位置已经被排完了，我们可以来选一个人排在第二个位置。因为前面已经用掉一个人了，所以还剩下 $n-1$ 个人，也就是有 $n-1$ 种方案来排第二个位置。

以此类推，当 m 个位置都排完的时候，方案就是

$$n(n-1)(n-2)\cdots(n-m+1)$$

这个数通常被成为**排列**，记成 A_n^m ，或者 P_n^m 。

排列与组合

这个数通常被成为**排列**，记成 A_n^m ，或者 P_n^m 。

如果我们定义一种名为**阶乘**的运算：

$$n! = 1 \times 2 \times 3 \times \cdots \times n$$

特别地，当 $n = 0$ 时， $0! = 1$ 。

排列与组合

这个数通常被成为**排列**，记成 A_n^m ，或者 P_n^m 。

如果我们定义一种名为**阶乘**的运算：

$$n! = 1 \times 2 \times 3 \times \cdots \times n$$

特别地，当 $n = 0$ 时， $0! = 1$ 。

那么这个数就可以简单地写成：

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

排列与组合

如果把问题改成：你只需要在 n 个不同的人里面选出 m 个人，问有多少种方案？

排列与组合

如果把问题改成：你只需要在 n 个不同的人里面选出 m 个人，问有多少种方案？

可以把这个方案分成两大类：

- 小 A 被选出来了
- 小 A 没有被选出来

排列与组合

排列与组合

我们先设答案是 $C(n, m)$ 。

排列与组合

我们先设答案是 $C(n, m)$ 。

如果小 A 被选出来的话，那么现在就剩下 $n-1$ 个人，并且还需要选出 $m-1$ 个人，这时候方案数是 $C(n-1, m-1)$ 。

排列与组合

我们先设答案是 $C(n, m)$ 。

如果小 A 被选出来的话，那么现在就剩下 $n-1$ 个人，并且还需要选出 $m-1$ 个人，这时候方案数是 $C(n-1, m-1)$ 。

如果小 A 没有被选出来的话，那么现在剩下 $n-1$ 个人（因为小 A 已经确定了没有选），并且还需要选出 m 个人，那么这时候方案也就是 $C(n-1, m)$ 。

排列与组合

我们先设答案是 $C(n, m)$ 。

如果小 A 被选出来的话，那么现在就剩下 $n-1$ 个人，并且还需要选出 $m-1$ 个人，这时候方案数是 $C(n-1, m-1)$ 。

如果小 A 没有被选出来的话，那么现在剩下 $n-1$ 个人（因为小 A 已经确定了没有选），并且还需要选出 m 个人，那么这时候方案也就是 $C(n-1, m)$ 。

因此，最后可以得到一个递推关系

$$C(n, m) = C(n-1, m) + C(n-1, m-1)$$

似曾相识？

排列与组合

似曾相识？

杨辉三角！

这个递推关系和杨辉三角那里的规律是相同的！

排列与组合

似曾相识？

杨辉三角！

这个递推关系和杨辉三角那里的规律是相同的！

事实上，这个数被成为**组合数**，通常记为 C_n^m 。

排列与组合

似曾相识？

杨辉三角！

这个递推关系和杨辉三角那里的规律是相同的！

事实上，这个数被成为**组合数**，通常记为 C_n^m 。

那么这个数是否有类似先前 A_n^m 一样的通项呢？

排列与组合

换一种方法来思考，如果我们选出来 m 个人后，再将他们排成一队，那么方案数就是先前的排列数 A_n^m 。

但是这样的计数会出现很多的重复，也就是每次我们都多算了将 m 个人排成一队的方案数，那么这个数是什么呢？

排列与组合

换一种方法来思考，如果我们选出来 m 个人后，再将他们排成一队，那么方案数就是先前的排列数 A_n^m 。

但是这样的计数会出现很多的重复，也就是每次我们都多算了将 m 个人排成一队的方案数，那么这个数是什么呢？

它就是 A_m^m ，或者说 $m!$ 。

排列与组合

换一种方法来思考，如果我们选出来 m 个人后，再将他们排成一队，那么方案数就是先前的排列数 A_n^m 。

但是这样的计数会出现很多的重复，也就是每次我们都多算了将 m 个人排成一队的方案数，那么这个数是什么呢？

它就是 A_m^m ，或者说 $m!$ 。

那么由于每种方案都被重复计算了 $m!$ 次，我们只要将 A_n^m 除以 $m!$ 就可以得到组合数的公式了！

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

组合数的基本性质

首先第一个性质就是先前的递推关系

$$C_n^m = C_{n-1}^m + C_{n-1}^{m-1} \quad (1)$$

此外，直接根据通项可以得到一个对称的性质：

$$C_n^m = C_n^{n-m} \quad (2)$$

二项式定理

现在再来观察一些东西：

$$(x + y)^1 = x + y$$

$$(x + y)^2 = x^2 + 2xy + y^2$$

$$(x + y)^3 = x^3 + 3x^2y + 3xy^2 + y^3$$

$$(x + y)^4 = x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4$$

二项式定理

现在再来观察一些东西：

$$(x + y)^1 = x + y$$

$$(x + y)^2 = x^2 + 2xy + y^2$$

$$(x + y)^3 = x^3 + 3x^2y + 3xy^2 + y^3$$

$$(x + y)^4 = x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4$$

是不是又有一种似曾相识的感觉？

二项式定理

事实上，这个是被称作**二项式定理**的一个定理。它告诉了我们 $(x + y)^n$ 的展开式（这里 n 是正整数）。

我们如果设 $(x + y)^n$ 的 $x^{n-k}y^k$ 项系数是 $f(n, k)$ ，那么可以根据

$$(x + y)^n = (x + y)(x + y)^{n-1}$$

得到递推关系：

$$f(n, k) = f(n - 1, k) + f(n - 1, k - 1)$$

再加上边界条件 $f(n, 0) = f(n, n) = 1$ ，这刚好又是组合数！

二项式定理

因此，我们就得到了这样一个定理：

Theorem (二项式定理)

对于正整数 n ,

$$(x + y)^n = \sum_{k=0}^n C_n^k x^k y^{n-k}$$

求 $C_n^m \bmod 10007$, 有 T 组数据

BZOJ2982.combination

求 $C_n^m \bmod 10007$, 有 T 组数据

对于 20% 的数据, $n, m \leq 1000$

BZOJ2982.combination

求 $C_n^m \bmod 10007$, 有 T 组数据

对于 20% 的数据, $n, m \leq 1000$

对于 40% 的数据, $m \leq 10^5, \tau \leq 1$

BZOJ2982.combination

求 $C_n^m \bmod 10007$, 有 T 组数据

对于 20% 的数据, $n, m \leq 1000$

对于 40% 的数据, $m \leq 10^5, \tau \leq 1$

对于 100% 的数据, $n, m \leq 10^9, \tau \leq 100$

1 数论基础

2 数学基础算法

3 组合数学基础

4 例题

例题. Remainders Game

例题. On Sum of Fractions

例题. Civilization

① 数论基础

② 数学基础算法

③ 组合数学基础

4 例题

例题. Remainders Game

例题. On Sum of Fractions

例题. Civilization

Remainders Game

中国剩余定理

$$k \mid \text{lcm}(c_1, c_2, \dots, c_n)$$

On Sum of Fractions

定义函数

- $v(n)$ 表示小于或等于 n 最大的素数
- $u(n)$ 表示大于 n 最小的素数

计算

$$\sum_{i=2}^n \frac{1}{v(i)u(i)}$$

多组数据，其中 $2 \leq n \leq 10^9$ 。

Source : Codeforces 396B. On Sum of Fractions

On Sum of Fractions

$$\frac{1}{2 \times 3} + \frac{1}{3 \times 5} + \frac{1}{3 \times 5} + \frac{1}{5 \times 7} + \frac{1}{5 \times 7} + \dots$$

On Sum of Fractions

$$\frac{1}{2 \times 3} + \frac{1}{3 \times 5} + \frac{1}{3 \times 5} + \frac{1}{5 \times 7} + \frac{1}{5 \times 7} + \dots$$

$$\frac{1}{2 \times 3} + \frac{5 - 3}{3 \times 5} + \frac{7 - 5}{5 \times 7} + \frac{11 - 7}{7 \times 11} + \dots$$

On Sum of Fractions

$$\frac{1}{2 \times 3} + \frac{1}{3 \times 5} + \frac{1}{3 \times 5} + \frac{1}{5 \times 7} + \frac{1}{5 \times 7} + \dots$$

$$\frac{1}{2 \times 3} + \frac{5-3}{3 \times 5} + \frac{7-5}{5 \times 7} + \frac{11-7}{7 \times 11} + \dots$$

$$\frac{b-a}{ab} = \frac{1}{a} - \frac{1}{b}$$

On Sum of Fractions

$$\frac{1}{2 \times 3} + \frac{1}{3 \times 5} + \frac{1}{3 \times 5} + \frac{1}{5 \times 7} + \frac{1}{5 \times 7} + \dots$$

$$\frac{1}{2 \times 3} + \frac{5-3}{3 \times 5} + \frac{7-5}{5 \times 7} + \frac{11-7}{7 \times 11} + \dots$$

$$\frac{b-a}{ab} = \frac{1}{a} - \frac{1}{b}$$

$$\frac{1}{2} - \frac{1}{3} + \frac{1}{3} - \frac{1}{5} + \frac{1}{5} - \frac{1}{7} + \dots$$

Civilization

给出一个 n 个未知数， n 个方程的线性方程组。求未知数的值。

其中 $n \leq 200$ ，系数在 $[0, 10^9]$ 之间，解在 $[0, 10^{18}]$ 之间。

Civilization

高斯消元

Civilization

高斯消元

中国剩余定理