

# 基本数学算法 part 1<sup>1</sup>

赖金霖<sup>2</sup>

清华大学 计算机科学与技术系

Aug 11, 2019

---

<sup>1</sup>本文件可在这里找到: [https://github.com/lll6924/public\\_slides](https://github.com/lll6924/public_slides)

<sup>2</sup>邮箱: [laijl16@mails.tsinghua.edu.cn](mailto:laijl16@mails.tsinghua.edu.cn)

# 赖金霖

- ▶ 高中就读于赣州市第三中学
- ▶ NOI 金牌 国家集训队
- ▶ 现就读于清华大学计算机科学与技术系
- ▶ NOIP2012~2015 一等奖
- ▶ NOIP2014 满分
- ▶ NOI2014 铜牌
- ▶ WC2015 二等奖
- ▶ APIO2015 金牌
- ▶ NOI2015 金牌
- ▶ 常用 id: ll6924



常用头像

数学中的一些美丽定理具有这样的特性：它们极易从事实中归纳出来，但证明却隐藏的极深。数学是科学之王。

——高斯

为什么要学习数学算法？

- ▶ 在脑海里构建计算机科学中的数学图景
- ▶ 拓展解决问题的思路
- ▶ 提高逻辑推理能力

# 算数基本定理

## 算数基本定理（唯一分解定理）

任意一个大于 1 的自然数  $N$ ，都可以唯一分解成有限个质数的乘积  $N = p_1^{a_1} p_2^{a_2} \dots p_n^{a_n}$ ，其中  $p_1 < p_2 < \dots < p_n$  为质数，指数  $a_i$  为正整数。

- ▶ 如果分解目标里可以有合数，分解式就不唯一了
- ▶  $12 = 3 \times 4 = 6 \times 2 = 2^2 \times 3$
- ▶ 算数基本定理为整数问题提供了一个思路，通过考虑一个数字的质因子，来考虑有关这个数字的问题
- ▶ 由于数学工具有限，我们略去证明

# 质因数分解

---

- 很自然地，我们想知道，对任意正整数  $N$ ，如何分解质因数？

# 质因数分解

- ▶ 很自然地，我们想知道，对任意正整数  $N$ ，如何分解质因数？
- ▶ 最基本的做法是，首先把  $1 \sim N$  的质数全部求出，对每个质数，不断除  $N$  直到不能整除，除的次数就是这个质数的指数
- ▶ 为什么这么做是对的？

# 质因数分解

---

- ▶ 很自然地，我们想知道，对任意正整数  $N$ ，如何分解质因数？
- ▶ 最基本的做法是，首先把  $1 \sim N$  的质数全部求出，对每个质数，不断除  $N$  直到不能整除，除的次数就是这个质数的指数
- ▶ 为什么这么做是对的？
- ▶ 根据算数基本定理，把  $N$  分解成  $N = p_1^{a_1} p_2^{a_2} \dots p_n^{a_n}$ ，那么在  
各质因子处算的指数一定是正确的
- ▶ 除法次数是  $O(\log N)$  的
- ▶ 使用线性筛法，总时间复杂度  $O(N)$

# 质因数分解

- ▶ 使用筛法算出所有质数是必要的吗？
- ▶ 令  $T=N$ ，从 1 循环到  $N$ ，遇到的第一个能整除  $T$  的数，满足什么性质？



# 质因数分解

- ▶ 使用筛法算出所有质数是必要的吗？
- ▶ 令  $T=N$ ，从 1 循环到  $N$ ，遇到的第一个能整除  $T$  的数，满足什么性质？
- ▶ 这个数一定是  $p_1$ ！我们可以用  $p_1$  不断除  $T$ ，除出它的指数。
- ▶ 除完之后， $T = p_2^{a_2} p_3^{a_3} \dots p_n^{a_n}$
- ▶ 遇到第二个能整除  $T$  的数，一定是  $p_2$

# 质因数分解

- ▶ 使用筛法算出所有质数是必要的吗？
- ▶ 令  $T=N$ ，从 1 循环到  $N$ ，遇到的第一个能整除  $T$  的数，满足什么性质？
- ▶ 这个数一定是  $p_1$ ！我们可以用  $p_1$  不断除  $T$ ，除出它的指数。
- ▶ 除完之后， $T = p_2^{a_2} p_3^{a_3} \dots p_n^{a_n}$
- ▶ 遇到第二个能整除  $T$  的数，一定是  $p_2$
- ▶ 我们可以用一次循环把所有质因子都找出，顺便算出次数
- ▶ 综上，我们得到的仍然是时间复杂度为  $O(N)$  的算法，但是不依赖筛法

# 质因数分解

- ▶ 回忆质数判定算法，我们只需要循环到  $\sqrt{N}$ ，就能判断出一个数是否是质数
- ▶ 这是因为合数最小的质因子一定小于等于  $\sqrt{N}$
- ▶ 质因数分解是循环到最大的质因子，最大的质因子有什么性质吗？

# 质因数分解

- ▶ 回忆质数判定算法，我们只需要循环到  $\sqrt{N}$ ，就能判断出一个数是否是质数
- ▶ 这是因为合数最小的质因子一定小于等于  $\sqrt{N}$
- ▶ 质因数分解是循环到最大的质因子，最大的质因子有什么性质吗？
- ▶ 最大的质因子可能是  $1 \sim N$  的任意质数
- ▶ 但是大于  $\sqrt{N}$  的质因子最多有一个
- ▶ 当我们循环到  $i = \sqrt{N}$  时，如果  $T=1$ ，质因数分解已经完成；如果  $T \neq 1$ ,  $T$  一定是质数，且是  $N$  的最大质因子
- ▶  $i = \sqrt{N} \geq \sqrt{T}$ ， $2 \sim i$  都不是  $T$  的因子，说明  $T$  是质数，在应用中，只需要循环到  $\sqrt{T}$

# 质因数分解

- ▶ 回忆质数判定算法，我们只需要循环到  $\sqrt{N}$ ，就能判断出一个数是否是质数
- ▶ 这是因为合数最小的质因子一定小于等于  $\sqrt{N}$
- ▶ 质因数分解是循环到最大的质因子，最大的质因子有什么性质吗？
- ▶ 最大的质因子可能是  $1 \sim N$  的任意质数
- ▶ 但是大于  $\sqrt{N}$  的质因子最多有一个
- ▶ 当我们循环到  $i = \sqrt{N}$  时，如果  $T=1$ ，质因数分解已经完成；如果  $T \neq 1$ ， $T$  一定是质数，且是  $N$  的最大质因子
- ▶  $i = \sqrt{N} \geq \sqrt{T}$ ， $2 \sim i$  都不是  $T$  的因子，说明  $T$  是质数，在应用中，只需要循环到  $\sqrt{T}$
- ▶ 这个做法时间复杂度为  $O(\sqrt{N})$ ，如果已经预处理了质数表，时间复杂度是  $O(\frac{\sqrt{N}}{\log N})$

```
int T=N,M=0;
for (int i=2; i*i<=T; i++)
    if (T%i==0){
        p[M]=i;
        q[M]=0;
        while (T%i==0){
            q[M]++;
            T/=i;
        }
        M++;
    }
if (T>1){
    p[M]=T;
    q[M++]=1;
}
```

# 因数枚举

在实际应用中，我们还需要枚举一个正整数  $N$  的所有因数，可以怎么做

- ▶ 最朴素的方法当然是从 1 循环到  $N$ ，判断每个数是否是  $N$  的因数

# 因数枚举

在实际应用中，我们还需要枚举一个正整数  $N$  的所有因数，可以怎么做

- ▶ 最朴素的方法当然是从 1 循环到  $N$ ，判断每个数是否是  $N$  的因数
- ▶ 注意到，如果  $i$  是  $N$  的因数，那么  $\frac{N}{i}$  也是  $N$  的因数
- ▶  $N$  的因数成对出现，其中一个一定小于等于  $\sqrt{N}$
- ▶ 于是我们可以从 1 循环到  $\sqrt{N}$ ，把每对因数中更小的求出，再算出更大的因数。这样枚举的因数一定不漏，但当  $N$  是完全平方数时， $\sqrt{N}$  处会重复计算，需要特殊处理



# 因数枚举

- ▶ 根据算数基本定理，我们还可以通过唯一分解式给出正整数  $N$ ，此时该如何枚举因数呢？

# 因数枚举

- ▶ 根据算数基本定理，我们还可以通过唯一分解式给出正整数  $N$ ，此时该如何枚举因数呢？
- ▶  $N$  的因数的分解式中，每个质数的指数一定不超过  $N$  分解式中对应的指数
- ▶ 一个自然的想法是，我们可以枚举分解式中每个质数的指数
- ▶ 设  $N = p_1^{a_1} p_2^{a_2} \dots p_n^{a_n}$ ，我们枚举  $b_1, b_2, \dots, b_n$ ，满足  $0 \leq b_i \leq a_i$ ，就能枚举  $N$  的所有因数了

# 因数枚举

- ▶ 根据算数基本定理，我们还可以通过唯一分解式给出正整数  $N$ ，此时该如何枚举因数呢？
- ▶  $N$  的因数的分解式中，每个质数的指数一定不超过  $N$  分解式中对应的指数
- ▶ 一个自然的想法是，我们可以枚举分解式中每个质数的指数
- ▶ 设  $N = p_1^{a_1} p_2^{a_2} \dots p_n^{a_n}$ ，我们枚举  $b_1, b_2, \dots, b_n$ ，满足  $0 \leq b_i \leq a_i$ ，就能枚举  $N$  的所有因数了
- ▶ 我们可以使用递归来枚举这样的  $b_i$ ，时间复杂度为  $O(\prod_{i=1}^n a_i)$

---

```
void find(int k, int x){
    if (k==M){
        factor[f++] = x;
        return;
    }
    for (int i=0; i<=q[k]; i++){
        find(k+1, x);
        x = x * p[k];
    }
}

...
find(0, 1);
...
```

# 几个基本计数问题

- ▶ 对于给定正整数  $N$ ，它的因数个数、因数之和分别是多少？
- ▶ 若  $N = p_1^{a_1} p_2^{a_2} \dots p_n^{a_n}$ ，因数个数为  $\prod_{i=1}^n (a_i + 1)$ ，因数之和为  $\prod_{i=1}^n (\sum_{j=0}^{a_i} p_i^j)$ ，计算复杂度  $O(\sqrt{N})$

# 几个基本计数问题

- ▶ 对于给定正整数  $N$ ，它的因数个数、因数之和分别是多少？
- ▶ 若  $N = p_1^{a_1} p_2^{a_2} \dots p_n^{a_n}$ ，因数个数为  $\prod_{i=1}^n (a_i + 1)$ ，因数之和为  $\prod_{i=1}^n (\sum_{j=0}^{a_i} p_i^j)$ ，计算复杂度  $O(\sqrt{N})$
- ▶ 对于给定正整数  $N$ ，在  $1 \sim N$  中与  $N$  互质的正整数个数，记做  $\varphi(N)$ ，这个函数叫做欧拉函数，对于任意的  $N$ ，如何求解  $\varphi(N)$ ？

# 几个基本计数问题

- ▶ 对于给定正整数  $N$ ，它的因数个数、因数之和分别是多少？
- ▶ 若  $N = p_1^{a_1} p_2^{a_2} \dots p_n^{a_n}$ ，因数个数为  $\prod_{i=1}^n (a_i + 1)$ ，因数之和为  $\prod_{i=1}^n (\sum_{j=0}^{a_i} p_i^j)$ ，计算复杂度  $O(\sqrt{N})$
- ▶ 对于给定正整数  $N$ ，在  $1 \sim N$  中与  $N$  互质的正整数个数，记做  $\varphi(N)$ ，这个函数叫做欧拉函数，对于任意的  $N$ ，如何求解  $\varphi(N)$ ？
- ▶ 我们不加证明地给出， $\varphi(N) = N \prod_{i=1}^n \frac{p_i - 1}{p_i}$
- ▶ 注意到，我们只需要求出  $N$  的所有质因子就可以计算了，计算复杂度  $O(\sqrt{N})$

# 几个基本计数问题

---

定义莫比乌斯函数  $\mu(x)$

$$\mu(x) = \begin{cases} 0 & x \text{ is a multiple of a square number} \\ (-1)^k & x = p_1 p_2 \dots p_k \end{cases}$$

- ▶ 对任意正整数  $N$ ，如何计算  $\mu(N)$ ?



# 几个基本计数问题

定义莫比乌斯函数  $\mu(x)$

$$\mu(x) = \begin{cases} 0 & x \text{ is a multiple of a square number} \\ (-1)^k & x = p_1 p_2 \dots p_k \end{cases}$$

- ▶ 对任意正整数  $N$ ，如何计算  $\mu(N)$ ?
- ▶ 我们既要计算  $N$  的质因数个数，又要计算每个质因数的个数是否大于 1
- ▶ 采用分解质因数的算法，时间复杂度  $O(\sqrt{N})$

# 积性函数

- ▶ 上面几个函数，都是积性函数
- ▶ 若定义域为正整数的函数  $f(x)$ ，对任意互质的  $x$ 、 $y$  都有， $f(x*y)=f(x)*f(y)$ ，那么  $f(x)$  是积性函数
- ▶ 为什么因数个数是积性函数？

# 积性函数

- ▶ 上面几个函数，都是积性函数
- ▶ 若定义域为正整数的函数  $f(x)$ ，对任意互质的  $x$ 、 $y$  都有， $f(x*y)=f(x)*f(y)$ ，那么  $f(x)$  是积性函数
- ▶ 为什么因数个数是积性函数？
- ▶ 形象地理解，若  $x$  和  $y$  互质，则  $x$  的质因子和  $y$  的质因子不相交， $x*y$  的质因子要么属于  $x$ ，要么属于  $y$ 。而因数个数是质因子次数 +1 之积， $x*y$  的因数个数等于属于  $x$  的质因子次数 +1 之积乘以属于  $y$  的质因子次数 +1 之积
- ▶ 为什么因数之和、欧拉函数、莫比乌斯函数也是积性函数？

# 最大公因数与最小公倍数

## 最大公因数与最小公倍数

对于任意正整数  $x$ 、 $y$ ， $x$  和  $y$  的最大公因数定义为最大的  $d$ ，使得  $d|x$  且  $d|y$ ； $x$  和  $y$  的最小公倍数定义为最小的  $m$ ，使得  $x|m$  且  $y|m$

- ▶ 我们通常把两个数的最大公因数记做  $\gcd(x,y)$ ，最小公倍数记做  $\text{lcm}(x,y)$
- ▶ 几个简单的性质：
- ▶  $\gcd(x,kx)=x$ ，其中  $k$  是任意正整数
- ▶  $\text{lcm}(x,kx)=kx$ ，其中  $k$  是任意正整数
- ▶  $x*y=\gcd(x,y)*\text{lcm}(x,y)$

## 最大公因数与最小公倍数

- ▶ 根据最后一条性质，求两个数的最小公倍数，等价于求两个数的最大公因数
- ▶ 最大公因数的通用求解算法是辗转相除，辗转相除为什么是对的呢？

## 最大公因数与最小公倍数

- ▶ 根据最后一条性质，求两个数的最小公倍数，等价于求两个数的最大公因数
- ▶ 最大公因数的通用求解算法是辗转相除，辗转相除为什么是对的呢？
- ▶ 在考虑辗转相除之前，我们先考虑辗转相减，每次从大的数里扣去小的数
- ▶ 不妨设  $x > y$ ，且  $\gcd(x, y) = d$
- ▶ 如果  $\gcd(x - y, y) = d$ ，那么辗转相减算法是正确的

# 最大公因数与最小公倍数

- ▶ 根据最后一条性质，求两个数的最小公倍数，等价于求两个数的最大公因数
- ▶ 最大公因数的通用求解算法是辗转相除，辗转相除为什么是对的呢？
- ▶ 在考虑辗转相除之前，我们先考虑辗转相减，每次从大的数里扣去小的数
- ▶ 不妨设  $x > y$ ，且  $\gcd(x, y) = d$
- ▶ 如果  $\gcd(x - y, y) = d$ ，那么辗转相减算法是正确的
- ▶ 因为  $d \mid x$  且  $d \mid y$ ，所以  $d \mid (x - y)$ ，故  $d$  至少是  $x - y$  和  $y$  的公因数
- ▶ 考虑反证法，如果  $d$  不是  $x - y$  和  $y$  的最大公因数，即存在  $c > d$ ，使得  $c \mid (x - y)$  且  $c \mid y$ ，那么  $c \mid x$ ，所以  $c$  会是  $x$  和  $y$  的公因数
- ▶ 但是  $c > d$ ，导出  $d$  不是  $x$  和  $y$  的最大公因数，矛盾，证毕
- ▶ 辗转相除是辗转相减的加速版本

# 用质因数分解看待最大公因数与最小公倍数

- ▶ 给定两个正整数  $x$  和  $y$ ，我们可以计算它们的质因数分解
- ▶  $x = a_1^{b_1} a_2^{b_2} \dots a_m^{b_m} \quad b_i \geq 1$
- ▶  $y = c_1^{d_1} c_2^{d_2} \dots c_n^{d_n} \quad d_i \geq 1$
- ▶ 如果我们规定质因子次数可以是 0，那么  $x$  和  $y$  能在同一组质因子下分解
- ▶  $x = p_1^{q_1} p_2^{q_2} \dots p_t^{q_t} \quad q_i \geq 0$
- ▶  $y = p_1^{r_1} p_2^{r_2} \dots p_t^{r_t} \quad r_i \geq 0$
- ▶  $\gcd(x,y)$  和  $\text{lcm}(x,y)$  的质因数分解是什么样的？



# 用质因数分解看待最大公因数与最小公倍数

- ▶ 给定两个正整数  $x$  和  $y$ ，我们可以计算它们的质因数分解
- ▶  $x = a_1^{b_1} a_2^{b_2} \dots a_m^{b_m} \quad b_i \geq 1$
- ▶  $y = c_1^{d_1} c_2^{d_2} \dots c_n^{d_n} \quad d_i \geq 1$
- ▶ 如果我们规定质因子次数可以是 0，那么  $x$  和  $y$  能在同一组质因子下分解
- ▶  $x = p_1^{q_1} p_2^{q_2} \dots p_t^{q_t} \quad q_i \geq 0$
- ▶  $y = p_1^{r_1} p_2^{r_2} \dots p_t^{r_t} \quad r_i \geq 0$
- ▶  $\gcd(x, y)$  和  $\text{lcm}(x, y)$  的质因数分解是什么样的？
- ▶  $\gcd(x, y) = p_1^{\min(q_1, r_1)} p_2^{\min(q_2, r_2)} \dots p_t^{\min(q_t, r_t)}$
- ▶  $\text{lcm}(x, y) = p_1^{\max(q_1, r_1)} p_2^{\max(q_2, r_2)} \dots p_t^{\max(q_t, r_t)}$
- ▶  $x * y = \gcd(x, y) * \text{lcm}(x, y)$  这条性质自动成立了

# 多个正整数的最大公因数与最小公倍数

- 问题：给定  $x_1, x_2, \dots, x_n$  共  $n$  个正整数，求它们的最大公因数
- ▶ 考虑  $n=3$  的情况，不妨设 3 个数分别为  $x$ 、 $y$ 、 $z$ ，我们将他们分解质因数
  - ▶  $x = p_1^{q_1} p_2^{q_2} \dots p_t^{q_t} \quad q_i \geq 0$
  - ▶  $y = p_1^{r_1} p_2^{r_2} \dots p_t^{r_t} \quad r_i \geq 0$
  - ▶  $z = p_1^{s_1} p_2^{s_2} \dots p_t^{s_t} \quad s_i \geq 0$
  - ▶ 显然  $\gcd(x, y, z) = p_1^{\min(q_1, r_1, s_1)} p_2^{\min(q_2, r_2, s_2)} \dots p_t^{\min(q_t, r_t, s_t)}$
  - ▶ 对任意  $n$ ，如果我们分解因数计算，复杂度为  $O(n\sqrt{A})$ ，其中  $[1, A]$  是正整数范围

# 多个正整数的最大公因数与最小公倍数

问题：给定  $x_1, x_2, \dots, x_n$  共  $n$  个正整数，求它们的最大公因数

- ▶ 考虑  $n=3$  的情况，不妨设 3 个数分别为  $x, y, z$ ，我们将他们分解质因数
- ▶  $x = p_1^{q_1} p_2^{q_2} \dots p_t^{q_t} \quad q_i \geq 0$
- ▶  $y = p_1^{r_1} p_2^{r_2} \dots p_t^{r_t} \quad r_i \geq 0$
- ▶  $z = p_1^{s_1} p_2^{s_2} \dots p_t^{s_t} \quad s_i \geq 0$
- ▶ 显然  $\gcd(x, y, z) = p_1^{\min(q_1, r_1, s_1)} p_2^{\min(q_2, r_2, s_2)} \dots p_t^{\min(q_t, r_t, s_t)}$
- ▶ 对任意  $n$ ，如果我们分解因数计算，复杂度为  $O(n\sqrt{A})$ ，其中  $[1, A]$  是正整数范围
- ▶ 你可能已经发现了， $\gcd(x, y, z) = \gcd(\gcd(x, y), z)$ ，于是我们可以一遍循环，维护前  $i$  个数的最大公因数，最后求出  $n$  个数的最大公因数，时间复杂度  $O(n \log A)$

## 多个正整数的最大公因数与最小公倍数

问题：给定  $x_1, x_2, \dots, x_n$  共  $n$  个正整数，求它们的最小公倍数模  $P$  的值

- ▶ 你可能也发现了， $\text{lcm}(x, y, z) = \text{lcm}(\text{lcm}(x, y), z)$ ，于是我们可以一遍循环，维护前  $i$  个数的最小公倍数，最后求出  $n$  个数的最小公倍数，时间复杂度  $O(n \log A)$
- ▶ 等等，真的可以这么做吗？

1151

---

你可能也发现了  $\text{lcm}(x, y, z) = \text{lcm}(\text{lcm}(x, y), z)$  于是我们可

# 例题讲解-NOIP2009 Hankson 的趣味题

题目大意：给定若干组正整数  $a_0, a_1, b_0, b_1$ ，对每组统计  $x$  的个数，满足  $\gcd(x, a_0) = a_1$  且  $\text{lcm}(x, b_0) = b_1$

## 样例输入

```
2
41 1 96 288
95 1 37 1776
```

## 样例输出

```
6
2
```

输入最多  $N=2000$  组，每个数字都在 `int` 范围内

对第一组数据，满足条件的  $x$  有 9,18,36,72,144,288；对第二组数据，满足条件的  $x$  有 48,1776

## 例题讲解-NOIP2009 Hankson 的趣味题

- ▶ 我们当然不能直接枚举，因为数据范围很大， $x$  有什么额外的限制？



# 例题讲解-NOIP2009 Hankson 的趣味题

- ▶ 我们当然不能直接枚举，因为数据范围很大， $x$  有什么额外的限制？
- ▶  $x$  必须满足  $a_1|x$  且  $x|b_1$
- ▶ 于是我们可以以  $O(\sqrt{b_1})$  的代价枚举  $\frac{b_1}{a_1}$  的因数，将因数乘以  $a_1$  即为可能的  $x$ ，再判断原式是否成立即可，复杂度  $O(N\sqrt{A}\log A)$ ，其中  $A$  为数据范围
- ▶ 这个做法虽然能通过这题，但看上去没有利用题目的所有条件

## 例题讲解-NOIP2009 Hankson 的趣味题

- ▶ 我们当然不能直接枚举，因为数据范围很大， $x$  有什么额外的限制？
- ▶  $x$  必须满足  $a_1|x$  且  $x|b_1$
- ▶ 于是我们可以以  $O(\sqrt{b_1})$  的代价枚举  $\frac{b_1}{a_1}$  的因数，将因数乘以  $a_1$  即为可能的  $x$ ，再判断原式是否成立即可，复杂度  $O(N\sqrt{A}\log A)$ ，其中  $A$  为数据范围
- ▶ 这个做法虽然能通过这题，但看上去没有利用题目的所有条件
- ▶ 不妨设  $a_0$  是  $a_1$  的倍数，与  $a_0$  的最大公因数为  $a_1$  的  $x$  的分解式是什么样的？若
- ▶  $a_0 = p_1^{q_1} p_2^{q_2} \dots p_t^{q_t} \quad q_i \geq 0$
- ▶  $a_1 = p_1^{r_1} p_2^{r_2} \dots p_t^{r_t} \quad r_i \geq 0$
- ▶ 则  $x = p_1^{s_1} p_2^{s_2} \dots p_t^{s_t} * (\text{other primes}) \quad \text{s.t.} \quad \min(s_i, q_i) = r_i$

## 例题讲解-NOIP2009 Hankson 的趣味题

- ▶ 若  $q_i = r_i$ ，则只需满足  $s_i \geq r_i$
- ▶ 若  $q_i > r_i$ ，则一定  $s_i = r_i$
- ▶ 根据  $\text{lcm}(x, b_0) = b_1$ ，我们也能算出一些质因子次数的范围
- ▶ 综合这些质因子次数的范围，可以用类似因数计数的方法算出  $x$  的数量
- ▶ 时间复杂度  $O(n\sqrt{A})$ ，如果提前打素数表，复杂度可进一步降为  $O(n\frac{\sqrt{A}}{\log A})$

# 同余问题

在平时的解题里，我们会使用如下几个公式

- ▶  $(c + d) \% P = ((c \% P) + (d \% P)) \% P$
- ▶  $(c - d) \% P = ((c \% P) - (d \% P)) \% P$
- ▶  $(c * d) \% P = ((c \% P) * (d \% P)) \% P$

我们需要考虑，1. 为什么这几个公式是正确的;2. 除法式取余要怎么做

为了方便，若  $a \% P = b \% P$ ，则记为  $a \equiv b(mod\ P)$

# 同余问题

- ▶ 对于第一个问题，我们只需要把模  $P$  看做不断加减  $P$ ，直到落入  $[0, P-1]$  内，就可以理解了
- ▶ 我们将模  $P$  的数都称作“ $i$  类”数，那么两个数是同一类数等价于两个数相差  $P$  的倍数。 $c$  和  $c \% P$  是同一类数， $d$  和  $d \% P$  是同一类数。容易看出  $c+d$  和  $c \% P + d \% P$  也是同一类数，所以  $c + d \equiv c \% P + d \% P \pmod{P}$
- ▶ 对减法的分析是类似的，为什么乘法的公式是正确的呢？

# 同余问题

- ▶ 对于第一个问题，我们只需要把模  $P$  看做不断加减  $P$ ，直到落入  $[0, P-1]$  内，就可以理解了
- ▶ 我们将模  $P$  的数都称作“ $i$  类”数，那么两个数是同一类数等价于两个数相差  $P$  的倍数。 $c$  和  $c \% P$  是同一类数， $d$  和  $d \% P$  是同一类数。容易看出  $c+d$  和  $c \% P + d \% P$  也是同一类数，所以  $c + d \equiv c \% P + d \% P \pmod{P}$
- ▶ 对减法的分析是类似的，为什么乘法的公式是正确的呢？
- ▶  $c * d$  和  $(c \% P) * (d \% P)$  也是同一类数
- ▶  $(c \% P) * (d \% P) = (c - k_1 P) * (d - k_2 P) = cd + (k_1 k_2 P - ck_2 - dk_1)P$
- ▶ 除法也有类似的公式吗？

- 假如我们要计算  $\frac{24}{6} \% 7$ ，我们会先计算  $24 \% 7 = 3$ ，但是  $\frac{3}{6} \% 7$  要怎么计算，我们还没有定义

对整数  $x$ ，如果存在一个整数  $y$ ，使得  $x * y \equiv 1(mod P)$ ，则称  $y$  是  $x$  模  $P$  下的逆元

在模  $P$  的意义下，除以  $x$  可以转化为乘以  $\text{inv}(x)$

$$((24\%7) * inv(6))\%7 = ((4\%7) * (6\%7) * inv(6))\%7 = 4$$

- ▶ 如果  $y$  是  $x$  模  $P$  下的逆元, 那么  $y$  是  $x+kP$  模  $P$  下的逆元
- ▶ 如果  $y$  是  $x$  模  $P$  下的逆元, 那么  $y+kP$  是  $x$  模  $P$  下的逆元

25 / 52



## 26 / 52

# 快速幂

- ▶ 如果我们要计算  $2^{4096} \% P$ ，会怎么做？
- ▶ 我们会进行转换  

$$2^{4096} = (2^{2048})^2 = ((2^{1024})^2)^2 = \dots = (\dots((2^2)^2)\dots)^2$$
- ▶ 只要将 2 自平方 12 次，共计算 12 次乘法，每次乘法结果都模 P，就能算出答案了
- ▶ 对于任意指数，可以怎么计算？

# 快速幂

- ▶ 如果我们要计算  $2^{4096} \% P$ ，会怎么做？
- ▶ 我们会进行转换  

$$2^{4096} = (2^{2048})^2 = ((2^{1024})^2)^2 = \dots = (\dots((2^2)^2)\dots)^2$$
- ▶ 只要将 2 自平方 12 次，共计算 12 次乘法，每次乘法结果都模 P，就能算出答案了
- ▶ 对于任意指数，可以怎么计算？
- ▶ 
$$a^q = \begin{cases} a(a^{\lfloor \frac{q}{2} \rfloor})^2 & q \text{ is odd} \\ (a^{\frac{q}{2}})^2 & q \text{ is even} \end{cases}$$
- ▶ 我们可以递归计算！

# 快速幂

```
int ksm(int a, int q){  
    if(q==0) return 1;  
    if(q==1) return a%P;  
    int ret=ksm(a, q/2);  
    ret=ret*ret%P;  
    if(q%2==1) ret=ret*a%P;  
    return ret;  
}
```

还有别的做法吗？

# 快速幂

- ▶ 递归算法通常有对应的非递归算法
- ▶ 我们将  $q$  转为二进制，考虑值为 1 的位
- ▶ 设值为 1 的位的位置分别是  $b_1, b_2, \dots, b_t$
- ▶ 那么  $q = 2^{b_1} + 2^{b_2} + \dots + 2^{b_t}$ ，其中  
 $0 \leq b_1 < b_2 < \dots < b_t \leq \log P$
- ▶ 所以  $a^q = a^{2^{b_1}} a^{2^{b_2}} \dots a^{2^{b_t}}$
- ▶ 我们以  $O(\log P)$  的复杂度把  $a^{2^0}, a^{2^1}, \dots, a^{2^{\lfloor \log P \rfloor}}$  模  $P$  的值都算出，然后直接乘出答案
- ▶ 在具体实现时，一遍循环就能把答案算出

若  $a$ 、 $p$  互质, 则  $a^{\varphi(p)} \equiv 1 \pmod{p}$

- ▶ 欧拉定理是费马小定理的扩展
- ▶ 当  $\gcd(a,p)=1$  是,  $a$  在模  $p$  意义下逆元为  $a^{\varphi(p)-1}$
- ▶ 若  $\gcd(a,p)>1$ ,  $a$  在模  $p$  意义下有逆元吗?

# 模为任意正整数的情况

## 欧拉定理

若  $a$ 、 $p$  互质，则  $a^{\varphi(p)} \equiv 1 \pmod{p}$

- ▶ 欧拉定理是费马小定理的扩展
- ▶ 当  $\gcd(a,p)=1$  是， $a$  在模  $p$  意义下逆元为  $a^{\varphi(p)-1}$
- ▶ 若  $\gcd(a,p)>1$ ， $a$  在模  $p$  意义下有逆元吗？
- ▶ 假设  $\gcd(a,p)=d$ ，采用反证法，设模  $p$  意义下  $a$  的逆元是  $b$
- ▶ 那么  $ab \equiv 1 \pmod{p}$
- ▶ 那么  $ab - kp = 1$ ，其中  $k$  是某个整数
- ▶ 因为  $d|a$  且  $d|p$ ，所以  $d|1$ ，但  $d>1$ ，矛盾
- ▶ 所以当  $\gcd(a,p)>1$  时，在模  $p$  意义下  $a$  没有逆元

# 求前 $N$ 个正整数的逆元

问题：给出  $N$ 、 $P$ ，求前  $N$  个正整数在模  $P$  下的逆元



$$\text{inv}(i * x) = \text{inv}(i) * \text{inv}(x)$$

- ▶ 逆元是积性函数，我们只需要计算质数幂的逆元
- ▶ 而质数幂的逆元为质数逆元的幂，我们只需要计算质数的逆元
- ▶ 如果用线性筛以  $O(N)$  的时间复杂度筛出所有质数，再以  $O(\log P)$  的复杂度算出这些质数的逆元，就可以递推出前  $N$  个正整数的逆元
- ▶ 递推方法：筛法进行到  $i$  时，筛去  $i \times x$ ，则计算  $\text{inv}(i \times x) = \text{inv}(i) \times \text{inv}(x)$
- ▶ 这个做法时间复杂度是多少？
- ▶ 时间复杂度是  $O(N \frac{\log P}{\log N})$

# 求前 N 个正整数的逆元

- ▶ 求前 N 个正整数的逆元，有一个通用方法
- ▶ 设我们要求  $i$  在模  $p$  意义下的逆元，且  $0 < i < p$
- ▶ 那么  $p = ki + r$ ，其中  $0 < r < i$
- ▶ 所以  $ki + r \equiv 0(mod\ p)$
- ▶ 所以  $k * inv(r) + inv(i) \equiv 0(mod\ p)$
- ▶ 所以  $inv(i) = (-k * inv(r)) \% p = (-\lfloor \frac{p}{i} \rfloor * inv(p \% i)) \% p$
- ▶ 上面的递归与辗转相除很像，我们可以得到一个以  $O(\log p)$  的复杂度计算某个数的逆元，其中边界条件为  $inv(1)=1$
- ▶ 回到我们的问题，因为  $p \% i < i$ ，所以我们可以从 1 循环到 N，推出每个数的逆元

中国剩余定理是求解以下方程的一个算法

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \dots \\ x \equiv a_n \pmod{m_n} \end{cases}$$

其中  $m_1, m_2, \dots, m_n$  两两互质

- 设  $M = \prod_{i=1}^n m_i$ , 若  $x$  是方程的解, 则  $x+kM(k \in \mathbb{Z})$  也是方程的解, 且  $[x+1, x+M-1]$  内一定没有解, 为什么?

# 中国剩余定理

中国剩余定理是求解以下方程的一个算法

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \dots \\ x \equiv a_n \pmod{m_n} \end{cases}$$

其中  $m_1, m_2, \dots, m_n$  两两互质

- ▶ 设  $M = \prod_{i=1}^n m_i$ ，若  $x$  是方程的解，则  $x+kM(k \in \mathbb{Z})$  也是方程的解，且  $[x+1, x+M-1]$  内一定没有解，为什么？
- ▶ 首先， $M$  是所有模数的倍数，所以  $x+kM$  模所有模数的答案不会变
- ▶ 其次，方程的两个不同的解之差，一定是所有模数的倍数

# 中国剩余定理

- ▶ 基于上面的性质，我们只需要找到方程的一个解，就能得到方程的全部解了
- ▶ 我们看到，方程的形式很简单，只是数量比较多，可以考虑用构造法求解
- ▶ 设  $t_i$  为  $\frac{M}{m_i}$  在模  $m_i$  下的逆元，则我们有
- ▶  $a_i \frac{M}{m_i} t_i \equiv a_i (mod\ m_i)$
- ▶ 那么  $\sum_{i=1}^n a_i \frac{M}{m_i} t_i$  是否是我们要找的  $x$  呢？

# 中国剩余定理

- ▶ 基于上面的性质，我们只需要找到方程的一个解，就能得到方程的全部解了
- ▶ 我们看到，方程的形式很简单，只是数量比较多，可以考虑用构造法求解
- ▶ 设  $t_i$  为  $\frac{M}{m_i}$  在模  $m_i$  下的逆元，则我们有
- ▶  $a_i \frac{M}{m_i} t_i \equiv a_i \pmod{m_i}$
- ▶ 那么  $\sum_{i=1}^n a_i \frac{M}{m_i} t_i$  是否是我们要找的  $x$  呢？
- ▶ 答案是肯定的。我们需要考虑  $a_i \frac{M}{m_i} t_i$  模  $m_j$  之外的模数的值。注意到  $\frac{M}{m_i}$  是任意  $m_j, j \neq i$  的倍数，所以
- ▶  $a_i \frac{M}{m_i} t_i \equiv 0 \pmod{m_j} \quad j \neq i$
- ▶ 所以  $x = \sum_{i=1}^n a_i \frac{M}{m_i} t_i$  是方程组的一个解
- ▶ 若要寻找最小解，则需要将  $x$  模  $M$

# 中国剩余定理的讨论

- ▶ 设  $A = \max_{i=1}^n m_i$
- ▶ 如果我们要在 64 位正整数内实现中国剩余定理，则  $M \cdot A$  不能超过 64 位整数的存储范围
- ▶ 如果我们的  $m_i$  取前若干个质数，则  $n$  最大为 14，此时  $m_n = 43$
- ▶ 如果不考虑存储范围，中国剩余定理的时间复杂度为  $O(n \log A)$
- ▶ 中国剩余定理的条件比较严格， $m_i$  必须互相互质，在下午我们会继续讨论不要求互质的情况



## 例题讲解-AHOI2005 洗牌

题目大意：有一摞纸牌，牌上数字依次为  $1 \sim N$ ， $N$  是偶数。每次洗牌可以把前半摞牌和后半摞牌拿出，后半摞牌一张、前半摞牌一张、后半摞牌一张、... 地放牌，执行完后相当于一次洗牌。如当  $N=6$  时，一开始的牌堆顺序为 1、2、3、4、5、6，洗牌一次之后牌堆顺序为 4、1、5、2、6、3，再洗一次之后顺序为 2、4、6、1、3、5。现想知道，对于一摞有  $N$  张牌的纸牌，洗  $M$  次之后，第  $L$  张牌是多少？

样例输入

6 2 3

$$N \leq 10^{10}, M \leq 10^{10}$$

样例输出

6

## 例题讲解-AHOI2005 洗牌

- ▶ 通过观察和验证，可以发现，在一次洗牌时，第  $i$  张牌被洗成了第  $i * 2\%(N + 1)$  张
- ▶ 问题转化为在  $1 \sim N$  内求一个数  $x$ ，满足  $x * 2^M \equiv L(mod\ N + 1)$
- ▶ 无论  $N$  是哪个偶数， $2$  和  $N+1$  都互质，我们可以用欧拉定理求出  $2$  在模  $N+1$  下的逆元  $inv(2)$ ，然后通过  $x = L * inv(2)^M\%(N + 1)$  计算
- ▶ 这样做问题就解决了吗？

## 例题讲解-AHOI2005 洗牌

- ▶ 通过观察和验证，可以发现，在一次洗牌时，第  $i$  张牌被洗成了第  $i * 2 \% (N + 1)$  张
- ▶ 问题转化为在  $1 \sim N$  内求一个数  $x$ ，满足
$$x * 2^M \equiv L \pmod{N + 1}$$
- ▶ 无论  $N$  是哪个偶数， $2$  和  $N + 1$  都互质，我们可以用欧拉定理求出  $2$  在模  $N + 1$  下的逆元  $\text{inv}(2)$ ，然后通过
$$x = L * \text{inv}(2)^{M \% (N + 1)}$$
 计算
- ▶ 这样做问题就解决了吗？
- ▶ 数据范围是  $10^{10}$ ，再算逆元时，乘法会超过存储范围
- ▶ 出题人的恶意？
- ▶ 可以借鉴快速幂算法，设计“快速加”算法。 $a * b$  本质上是  $a$  个  $b$  相加， $a$  的角色和快速幂里幂次的角色相同
- ▶ 采用快速加算法之后，总复杂度为  $O(\log^2 N)$

# 进制转换

问题：对一个 10 进制数  $x$ ，写出它的  $k$  进制形式

- ▶ 将  $x$  写成  $x = a_n k^n + a_{n-1} k^{n-1} + \dots + a_1 k^1 + a_0$ ,  $0 \leq a_i < k$   
后,  $x$  的  $k$  进制形式为  $\overline{a_n a_{n-1} \dots a_1 a_0}$
- ▶ 注意到  $a_0 = x \% k, a_1 = x / k \% k, a_2 = x / k / k \% k, \dots$
- ▶ 在纸上, 我们可以写成短除法的形式
- ▶ 在程序中, 我们可以用一遍循环求出  $a_0 \sim a_n$ , 再倒序输出
- ▶ 由于最后需要倒序输出, 我们可以写成递归的形式, 在回溯时输出

# 进制转换

问题：对一个  $k$  进制数  $\overline{a_n a_{n-1} \dots a_1 a_0}$ ，算出它 10 进制形式

- ▶ 这个  $k$  进制数的值就是
 
$$x = a_n k^n + a_{n-1} k^{n-1} + \dots + a_1 k^1 + a_0$$
- ▶ 在纸上，我们可以直接计算
- ▶ 在程序中，我们一遍循环就能算出  $x$  了，直接用 10 进制输出  $x$  即可
- ▶ 现在有另一个问题，对一个  $k$  进制数  $\overline{a_n a_{n-1} \dots a_1 a_0}$ ，求它的  $l$  进制形式，该如何做？
- ▶ 可以先算出数字的值，然后转成  $l$  进制

# 进制转换

问题：对一个  $k$  进制小数  $\overline{a_n a_{n-1} \dots a_1 a_0 . a_{-1} a_{-2} \dots a_{-m}}$ ，写出它的 10 进制形式

- ▶ 这个  $k$  进制小数的值是  $x =$   

$$a_n k^n + a_{n-1} k^{n-1} + \dots + a_1 k^1 + a_0 + a_{-1} k^{-1} + a_{-2} + \dots + a_{-m} k^{-m}$$
- ▶ 可以用类似的方法算出
- ▶ 如果我们给定 10 进制小数，如何计算它的  $k$  进制形式？

---

- ▶ 这个  $k$  进制小数的值是  $x = a_n k^n + a_{n-1} k^{n-1} + \dots + a_1 k^1 + a_0 + a_{-1} k^{-1} + a_{-2} + \dots + a_{-m} k^{-m}$
- ▶ 可以用类似的方法算出
- ▶ 如果我们给定 10 进制小数，如何计算它的  $k$  进制形式？
- ▶ 我们需要指定  $k$  进制形式中小数点后的位数
- ▶ 若小数点后保留  $m$  位，则我们设  $y = \text{round}(x * k^m)$
- ▶ 求出  $y$  的  $k$  进制形式，再将小数点左移  $m$  位即可

---

- ▶ 我们可以模拟加法的计算过程
- ▶ 设  $k = \max(n, m)$ ,  $c_i = (a_i + b_i + s_i) \% k$
- ▶  $s_{i+1} = (a_i + b_i + s_i) / k$ ,  $0 \leq i \leq k$ ,  $s_0 = 0$ , 其中  $s_i$  为上一位的进位, 每位的结果存储在  $c_i$  中
- ▶ 我们可以用一遍循环, 模拟上面的计算
- ▶ 当  $s_{k+1}$  非零时, 最高位发生进位, 结果增加到  $k+1$  位
- ▶ 高精度减法也是类似的, 在循环时维护借位。注意需要提前比较减数和被减数, 用更大的减去更小的, 最后再考虑符号



- ▶ 我们可以模拟加法的计算过程
- ▶ 设  $k = \max(n, m)$ ,  $c_i = (a_i + b_i + s_i) \% k$
- ▶  $s_{i+1} = (a_i + b_i + s_i) / k$ ,  $0 \leq i \leq k$ ,  $s_0 = 0$ , 其中  $s_i$  为上一位的进位, 每位的结果存储在  $c_i$  中
- ▶ 我们可以用一遍循环, 模拟上面的计算
- ▶ 当  $s_{k+1}$  非零时, 最高位发生进位, 结果增加到  $k+1$  位
- ▶ 高精度减法也是类似的, 在循环时维护借位。注意需要提前比较减数和被减数, 用更大的减去更小的, 最后再考虑符号
- ▶ 对于高精度计算,  $k$  越高, 效率越高, 但  $k$  过高会存不下
- ▶ 高精度的常见坑点是输出, 假设  $k=1000$ , 当前位为 1, 需要在当前位前补 3 个 0

# 高精度计算

---

问题：输入两个  $k$  进制数  $\overline{a_n a_{n-1} \dots a_1 a_0}$  和  $\overline{b_m b_{m-1} \dots b_1 b_0}$ ，求它们的积

- ▶ 比起加法，乘法的计算更难模拟，需要设计好写的算法

# 高精度计算

问题：输入两个  $k$  进制数  $\overline{a_n a_{n-1} \dots a_1 a_0}$  和  $\overline{b_m b_{m-1} \dots b_1 b_0}$ ，求它们的积

- ▶ 比起加法，乘法的计算更难模拟，需要设计好写的算法
- ▶ 如果不考虑进位，我们可以计算  $c_i = \sum_{j=\max(0, i-m)}^{\min(i, n)} a_j b_{i-j}$ ，为结果的第  $i$  位
- ▶ 这个计算也可以用二个重循环枚举  $a$  和  $b$  的下标完成
- ▶ 最后，我们可以用一遍循环把进位算出

# 高精度计算

问题：输入两个  $k$  进制数  $\overline{a_n a_{n-1} \dots a_1 a_0}$  和  $\overline{b_m b_{m-1} \dots b_1 b_0}$ ，求它们的积

- ▶ 比起加法，乘法的计算更难模拟，需要设计好写的算法
- ▶ 如果不考虑进位，我们可以计算  $c_i = \sum_{j=\max(0, i-m)}^{\min(i, n)} a_j b_{i-j}$ ，为结果的第  $i$  位
- ▶ 这个计算也可以用二个重循环枚举  $a$  和  $b$  的下标完成
- ▶ 最后，我们可以用一遍循环把进位算出
- ▶ 以计算十进制数  $987 \times 234$  为例
- ▶ 如果不进位，则结果为 18 43 74 53 28
- ▶ 进位后，结果为 2 3 0 9 5 8

```
for (int i=0; i<=n; i++)
    for (int j=0; j<=m; j++)
        c[i+j] += a[i] * b[j]
int s=0, d=n+m;
for (int i=0; i<=n+m; i++){
    int t=s;
    s=(c[i]+s)/k;
    c[i]=(c[i]+t)%k;
}
while (s>0){
    c[++d]=s%k;
    s/=k;
}
```

以上是高精度乘法的代码





# 位运算的性质和技巧

- ▶ 与、或、异或、非都是按位计算，各位的运算相互独立
- ▶ 与运算的结果不超过运算数，或运算的结果不小于运算数
- ▶ 一个数和自己的异或等于 0
- ▶ 仅由异或组成的公式满足交换律、结合律，仅由与组成的公式和仅由或组成的公式也满足交换律、结合律
- ▶ 在 cpu 中，位运算比乘除法更快
- ▶  $a \& (2^i - 1)$  等价于  $a \% 2^i$ ，当  $i=1$  时，可用于判断奇偶
- ▶  $a \gg i$  等价于  $a / 2^i$ ，当  $i=1$  时，可用于除以 2
- ▶ 可用  $x \& (1 \ll i)$  来判断， $x$  的第  $i$  位（个位为第 0 位）是否为 1



# 枚举子集

- ▶ 对于大小为  $n$  的全集  $S$ ，我们通常用  $n$  位二进制来表示  $S$  的子集
- ▶ 二进制的每位都对应全集中的一个元素，当此位为 1 时，表示包含这一元素；当此位为 0 时，表示不包含这一元素
- ▶ 设  $S'$  是  $S$  的一个子集，该如何枚举  $S'$  的子集？

# 枚举子集

- ▶ 对于大小为  $n$  的全集  $S$ ，我们通常用  $n$  位二进制来表示  $S$  的子集
- ▶ 二进制的每位都对应全集中一个元素，当此位为 1 时，表示包含这一元素；当此位为 0 时，表示不包含这一元素
- ▶ 设  $S'$  是  $S$  的一个子集，该如何枚举  $S'$  的子集？
- ▶ 如果  $S'$  的二进制表示为  $T$ ，那么可以用以下的循环枚举  $S'$  的子集
- ▶ `for(int i=T; i!=0; i=(i-1)&T)`
- ▶  $i-1$  让最后一位 1 变为 0，最后一位 1 之后的位变为 1
- ▶ 在  $T$  中是 0 的位始终为 0
- ▶ 拿出在  $T$  中是 1 的位组成新的数，上述循环实际上是不不断让这个数-1



# 格雷码

- ▶ 对于所有  $n$  位二进制数，将其重新排列后，若相邻两个数最多有 1 位不同，首尾两个数最多有 1 位不同，则称排列后的序列是格雷码
- ▶ 当  $n=2$  时，一个合法的格雷码是 00、01、11、10
- ▶ 从  $n=k$  的格雷码，可以生成  $n=k+1$  的格雷码：若  $n=k$  的格雷码为  $g_0, g_1, \dots, g_{2^k-1}$ ，则  $n=k+1$  的格雷码是  $\overline{0g_0}, \overline{0g_1}, \dots, \overline{0g_{2^k-1}}, \overline{1g_{2^k-1}}, \overline{1g_{2^k-2}}, \dots, \overline{1g_0}$
- ▶ 我们也可以直接计算第  $i$  个格雷码 (0 是第 0 个格雷码)，第  $i$  个格雷码是  $i \wedge (i \gg 1)$
- ▶ 其中第  $i$  个格雷码相比第  $i-1$  个格雷码， $i$  的最后一位 1 所在的位被取反

题目大意：给出  $n$  个门，数字通过每个门，需要和门上的数字做门给出的运算，运算包含与、或、异或，求  $0 \sim m$  中的数，依次通过这  $n$  个门后，最大的结果

## 样例输入

3 10  
AND 5  
OR 6  
XOR 7

## 样例输出

1

样例解释：1、3、5、7、9 通过这 3 扇门后变为 0，0、2、4、6、8、10 通过这 3 扇门后变为 1

## 例题讲解-NOI2014 起床困难综合症

- ▶ 注意到，运算时各二进制位是独立的
- ▶ 可以计算出每位分别取 0 或 1 时此位的结果

# 例题讲解-NOI2014 起床困难综合症

- ▶ 注意到，运算时各二进制位是独立的
- ▶ 可以计算出每位分别取 0 或 1 时此位的结果
- ▶ 从高位到低位贪心，在  $\leq m$  的约束下，选取最大的输出
- ▶ 从高位到低位循环，当  $\leq m$  的约束还有时，若  $m$  的当前位为 0，则此位的只能取 0；若  $m$  的当前位为 1，取 0 和 1 中答案更大的一个，如果答案一样大，取 0，并取消  $\leq m$  的约束；当没有  $\leq m$  的约束时，取 0 和 1 中答案更大的一个
- ▶ 时间复杂度  $O(n \log m)$

Thanks for listening

Any Questions?