

数论总结

本来是想按照代教 PPT 的顺序总结的，但是发现我做题的顺序好像从开始除了第一道题之外写的都是跟质数筛，gcd，lcm 相关的题目，再加上蓝书的顺序也是如此，所以就从这里开始总结吧。

• 1. 质数

蓝书上质数章节，分为**质数判定**，**质数筛选**，**质因数分解**三个板块。但是由于指数判定太简单，况且到了这个程度了是人都会吧这里只总结筛法和质因数分解。

1.1. 筛法

问题原型：给定一个整数 N ，求出 $1-N$ 之间所有的质数。

就不写暴力筛子吧

埃氏筛

埃氏筛，又称 Eratosthenes 筛法。显然是 Eratosthenes 发明的。

- 基本思想：从 2 开始，从小到大枚举每个数 x ，把它的倍数 $2x, 3x, \dots$ 标记为合数。当扫到一个数时，若它未被标记，则这个数就是质数。
- 优化：由基本思想可知，我们在标记合数时，会经常把一个合数标记两次及以上，我们能不能通过优化来减少标记次数从而提高时间效率呢？完全可以。举个例子，2 和 3 两个质数都会把 6 标记为合数。所以其实所有小于 x^2 的合数在 x 之前已经有更小的数把它标记过了。（像上面的 3×2 和 2×3 ），所以我们可以对埃氏筛进行优化：对于每一个 x 标记合数时，只需要从 x^2 开始扫到 $\left\lfloor \frac{n}{x} \right\rfloor \times x$ 即可。
- 时间复杂度： $O(N \log \log N)$

代码实现：

```
memset(vis,0,sizeof(vis));
for(int i=2;i<=n;i++)
{
    if(vis[i])continue;//若被标记，则说明不是质数。
    cout<<i<<endl;//输出质数
    for(int j=i;j*i<=n;j++)vis[i*j]++;//将被数标记为合数。
}
```

线性筛(欧拉筛)

- 我们会发现，埃氏筛仍然会有一些地方，重复标记合数。比如，12 同时被 2 和 3 标记。其根本原因就是我们没有确定出唯一的产生 12 的方式。
- 那么我们是否能对每一个合数，通过某种顺序，确定产生这种合数的唯一方式呢？
- 想想我们学过的跟“**唯一**”“**确定**”有关的定理——算术基本定理（唯一分解定理），呃呃现在没学无所谓下面很快会提到，我们可以想办法把一个合数通过类似“分解质因数”的办法把它们“拆”成**唯一的分解形式**。比如，我们让 12 只能通过 $3 \times 2 \times 2$ 得到，而不通过其他方式得到。而这个唯一分解形式的关键就在于：**质因子**!!! 那么我们就可以创造一种类似于**逆向分解质因数**的办法，来使得每个合数产生的方式是唯一的。
- 基本思想：从小到大枚举 i ，如果 i 没被标记过，就标记为质数。每次枚举不大于 i 的质数 x ，将 $x \times i$ 标记为合数，然后如果 x 不是 $x \times i$ 最小的质数，那么break即可。
- 两种实现方法：
 - 第一种：用 vis[] 表示是否被标记为质数。

判断是否为最小质因子的方法：if(!(i%p[j]))break;原理：感觉模板题题解的一部分地方说的挺好的。这里截过来：

```
for(int i=2;i<=n;i++)
{
    if(vis[i]==0)p[++cnt]=i;
    for(int j=1;j<=cnt&& i*p[j]<=n;j++)
    {
        vis[p[j]*i]++;
        if(!(i%p[j]))break;
    }
}
```

- 第二种：用 `vis[]` 存储最小质因子。对于一个数 `x` 的 `vis`，若 `vis[x] < p[j]` 说明 `i` 有比 `p[j]` 更小的质因子，直接 `break`；

```
for(int i=2;i<=n;i++)
{
    if(vis[i]==0){vis[i]=i;p[++cnt]=i;}
    for(int j=1;j<=cnt&& i*p[j]<=n;j++)
    {
        if(p[j]>v[i])break;
        vis[i*p[j]]=p[j];
    }
}
```

- 时间复杂度：由于每个质数只会被它的最小质因子筛一次，所以时间复杂度 $O(n)$ 。

1.1. 质因数分解

唯一分解定理（算数基本定理）

- 任何一个大于 1 的正整数都可以**唯一分解为有限个质数**的乘积，可写作：
- $$N = p_1^{c_1} p_2^{c_2} \dots p_m^{c_m}$$
- 其中 c_i 都是正整数， p_i 都是质数，且满足 $p_1 < p_2 < \dots < p_m$ 。

试除法

这个很简单，首先根据质数的判定是从 $1 - \sqrt{n}$ 来扫，那么我们也只需要试除 $1 - \sqrt{n}$ 的质数，即可完成对 `n` 的质因数分解。

如何试除从 $1 - \sqrt{n}$ 的质数呢？我们只需要从 1 枚举到 \sqrt{n} ，每次遇到一个能被 `n` 整除的数就不断对 `n/=i`，直到 `n%i` 不为 0 即可。

为什么这么做是对的呢？因为一个合数的影子一定在扫描到这个合数之前就从 `n` 中被除去了，所以上述过程中能整除 `n` 的一定是质数，多次 `n/=i` 不仅是为了排除之后 `i` 的倍数对 `n` 的干扰，还能够起到累计计算 `i` 的个数的作用。

时间复杂度：显而易见的 $O(\sqrt{n})$

代码实现：

```

for(int i=2;i*i<=n;i++)
{
    if(n%i==0)
    {
        p[++cnt]=i;c[cnt]=0;
        while(n%i==0)n/=i,c[cnt]++;
    }
}

if(n>1) p[++cnt]=n,c[cnt]=1;//注意一下这里，如果这时候 n 还没被除到1，则说明 n 是质数。

```

2. 约数

2.1. 定义

若整数 n 除以整数 d 的余数为 0，即 d 能整除 n ，则称 d 是 n 的约数， n 是 d 的倍数，记为 $d|n$ 。

算术基本定理推论：

在算术基本定理中，若正整数 N 被唯一分解为 $N = p_1^{c_1} p_2^{c_2} \dots p_m^{c_m}$ ，其中 c_i 都是正整数， p_i 都是质数，且满足 $p_1 < p_2 < \dots < p_m$ ，则 N 的正约数集合可写作：

$\{p_1^{b_1} p_2^{b_2} \dots p_m^{b_m}\}$ ，其中 $0 \leq b_i \leq c_i$ 。

N 的正约数个数(\prod 表示连乘符号，与 \sum 类似)：

$$(c_1 + 1) \times (c_2 + 1) \times \dots \times (c_m + 1) = \prod_{i=1}^m (c_i + 1)$$

说明：枚举每个 $0 - c_i$ 选或者不选，然后再套用组合数学公式。

N 的所有正约数之和：

$$(1 + p_1 + p_1^2 + \dots + p_1^{c_1}) \times \dots (1 + p_m + p_m^2 + \dots + p_m^{c_m}) = \prod_{i=1}^m (\sum_{j=0}^{c_i} (p_i)^j)$$

2.2. 求解方法

试除法

试除法用来求正整数 N 的约数集合。

试除法的思想有点类似于朴素算法分解质因数：若 $d \geq \sqrt{N}$ 是 N 的约数，则 $N/d \leq \sqrt{N}$ 也是 N 的约数。换言之，**约数总是成对出现的（完全平方数除外）**。

因此，只需要扫描 $d = 1 - \sqrt{N}$ ，尝试 d 能否整除 N ，若能整除，则 N/d 也是 N 的约数。

实现代码：

```
int f[maxn], cnt=0; // f数组用来存储约数，cnt表示约数个数。
for(int i=1; i*i<=n; i++)
{
    if(!(n%i)){f[++cnt]=i; if(i!=n/i) f[++cnt]=n/i;} // 如果不是根号n，那就把n/i也存储进去。
}
for(int i=1; i<=m; i++) cout<<f[i]<<endl;
```

时间复杂度： $O(\sqrt{N})$

推论：一个整数 N 的约数个数上界是 $2\sqrt{N}$ 。

倍数法

与试除法不同，倍数法用来求 $1 - N$ 每个数的正约数集合。

如果用试除法去求 $1 - N$ 每个数的正约数集合，时间复杂度是 $O(\sqrt{N})$ 。如果 N 很大，一定会 T。

联想到我们之前学过的筛法。筛法的思想不同于普通的试除法，而是利用**逆向思维**。我们这里也把它借鉴过来。

对于 $1 - N$ (**注意不是** \sqrt{N}) 的每一个数 d ，扫描 $1 - N$ 范围内的每个 d 的倍数 $d, 2d, 3d, \dots, \left\lfloor \frac{n}{d} \right\rfloor \times d$ ，并把它们标记一下，表示 d 是这些数的约数。

代码实现：

```
vector<int> f[maxn]; // 由于直接开数组空间不够，所以开了动态数组vector
for(int i=1; i<=n; i++)
    for(int j=1; j<=n/i; j++) f[i*j].push_back(i);
for(int i=1; i<=n; i++)
    for(int j:f[i]) cout<<f[i][j]<<'\\n';
```

时间复杂度: $O(N + N/2 + N/3 + \dots + N/N) = O(N \log N)$ 。

推论: $1 - N$ 每个数的约数个数的总和大约为 $N \log N$

• 3. 同余

这块有好多%意义下的运算，由于我太菜，这块很迷惑，所以先把这块总结一下吧。

3.3. 定义

若整数 a 和整数 b 除以正整数 m 的余数相同，则称 a, b 模 m **同余**，记为 $a \equiv b \pmod{m}$ 。

同余系与剩余系

同余类: 对于 $\forall a \in [0, m-1]$, 集合 $\{a + km\} (k \in \mathbb{Z})$ 的所有数模 m 同余, 余数都是 a , 该集合称为一个模 m 的**同余类**，简记为 \bar{a} 。

完全剩余系: m 个模 m 的同余类构成 m 的**完全剩余系**，分别是 $\bar{0}, \bar{1}, \bar{2}, \dots, \overline{m-1}$ 。

简化剩余系: $1 - m$ 中与 m 互质的数代表的同余类共同构成 m 的简化同余系，共有 m 个。
e.g. 模 10 的简化剩余系为 $\{\bar{1}, \bar{3}, \bar{7}, \bar{9}\}$ 。

一个性质

首先我们要说一个概念：

运算封闭: 若从某个**非空集合**中**任选**两个元素（**同一元素可重复选出**），选出的这两个元素通过某种（或几种）运算后的得数仍然是该数集中的元素，那么就说该集合对于这种（或几种）运算是封闭的。

简化剩余系关于模 m 算法完全封闭。理由：若 $a, b (1 \leq a, b \leq m)$ 与 m 互质，则 $a \times b$ 也不可能与 m 含有相同的质因子，即 $a \times b$ 也与 m 互质。而我们知道，如果 a 与 m 互质，那么 $a \bmod b$ 也一定与 m 互质，所以 $a \times b \bmod m$ 也与 m 互质，即 $a \times b \bmod m$ 也属于 m 的简化剩余系。

3.4. 费马小定理

若 p 是质数，则对于任意整数 a ，有 $a^p \equiv a \pmod{p}$ 。

3.5. 欧拉定理

若正整数 a, n 互质, 则 $a^{\varphi(n)} \equiv 1 \pmod{n}$, 其中 $\varphi(n)$ 为欧拉函数。

说明:

1. $\varphi(n)$ 表示的是小于等于 n 并且和 n 互质的数。易知: 当 n 为质数, $\varphi(n) = n - 1$
2. $a^{\varphi(n)} \equiv 1 \pmod{n}$ 表示的是 $a^{\varphi(n)} \bmod n = 1$ 。也就是, $a^{\varphi(n)} = nk + 1 (n \in \mathbb{Z})$ 。

欧拉定理推论

若正整数 a, n 互质, 则对于任意正整数 b , 有 $a^b \equiv a^{b \bmod \varphi(n)} \pmod{n}$ 。

//感性理解一下 (不是正文) a 的 b 次方和 a 的 $b \bmod \varphi(n)$ 次方 $\bmod n$ 的值是一样的。也就是说, $a^b \bmod n = a^{(b \bmod \varphi(n))}$

许多计数类的题目要求我们把答案对一个质数 p 取模后输出。面对 $a + b, a - b, a * b$ 这样的算式, 可以先把 a, b 对 p 取模。面对乘方算式, 根据欧拉定理的推论, 可以先把底数对 p 取模, 指数对 $\varphi(n)$ 取模, 再计算乘方。

——《算法竞赛进阶指南》

3.6. 扩展欧几里得算法(exgcd)

扩展欧几里得算法(exgcd), 常用于求解 $ax + by = \gcd(a, b)$ 一组合法解问题。

求解:

设 $ax_1 + by_1 = \gcd(a, b), bx_2 + a \bmod b y_2 = \gcd(b, a \bmod b)$ 。

根据**欧几里得定理**, $\gcd(a, b) = \gcd(b, a \bmod b)$ 。

$$\therefore ax_1 + by_1 = bx_2 + a \bmod b y_2$$

$$\because a \bmod b = a - b \times \left\lfloor \frac{a}{b} \right\rfloor$$

$$\therefore ax_1 + by_1 = bx_2 + (a - b \times \left\lfloor \frac{a}{b} \right\rfloor) y_2 = bx_2 + ay_2 - b \left\lfloor \frac{a}{b} \right\rfloor y_2 = ay_2 + b(x_2 - \left\lfloor \frac{a}{b} \right\rfloor y_2)$$

$$\because a = a, b = b$$

$$\therefore x_1 = y_2, y_1 = x_2 - \left\lfloor \frac{a}{b} \right\rfloor y_2$$

所以只需要将 x_2, y_2 不断代入递归求解直至 gcd 为 0 递归 $x = 1, y = 0$ 求解。

代码实现：

```
int exgcd(int a, int b, int &x, int &y)
{
    if(b==0){x=1;y=0;return a;}
    int d=exgcd(b, a%b, x, y);
    int t=x;x=y;y=t-y*(a/b);
    return d;
}
```

说明：

1. 简述一下上述代码的过程。定义变量 d, x_0, y_0 ，调用 $d = \text{exgcd}(a, b, x_0, y_0)$ 。其中 x_0, y_0 是通过引用的方式传递的。每次递归 $d = \text{exgcd}(b, a \% b, x, y)$ ，再通过引用的方式每次改变 x, y 的值。上述程序中求出方程 $ax + by = \text{gcd}(a, b)$ 的一组特解 x_0, y_0 ，并返回 $d = \text{gcd}(a, b)$ ；
2. **更一般地**，对于方程 $ax + by = c$ ，它有解的充要条件是当且仅当 $d|c$ 。

证明：

- **充分性**：首先根据 exgcd 可知，存在 (x_0, y_0) 使得 $ax_0 + by_0 = d$ ，又 $d|c$ ，所以 $c = dk = (ax_0 + by_0)k = a(kx_0) + b(ky_0)$ ，所以方程有整数解 (kx_0, ky_0)
- **必要性**：因为 $ax_0 + by_0 = c$ ， d 是 a, b 的最大公约数，所以 $d|a, d|b$ ，所以 $d|ax_0 + by_0$ ，即 $d|c$ 。

所以，对于该方程我们可以先求出方程 $ax + by = d$ 一组特解 (x_0, y_0) ，然后给 x_0, y_0 同时乘上 c/d ，即可得到 $ax + by = c$ 的一组特解 $(c/d)x_0, (c/d)y_0$ 。（这里不会的可以手推一下）事实上，将 x, y 代入原方程中就会发现：方程 $ax + by = c$ 的通解可以表示为：
 $x = (c/d)x_0 + k(b/d), y = (c/d)y_0 - k(a/d) (k \in \mathbb{Z})$ 。

3.7. 乘法逆元

如果一个线性同余方程 $ax \equiv 1 \pmod{b}$ 则 x 称为 $a \bmod b$ 的逆元，记作 a^{-1} 。

求解方法：

1. exgcd:

转化题意可知要求 $ax = bk + 1 (k \in \mathbb{Z})$ 中的 x , 也就是 $ax - bk = 1$ 。

令 $-k = y$, 则方程 $ax + by = c$ 与方程 $ax \equiv c \pmod{b}$ 等价, 所以直接套用上述求 exgcd 的特解的做法即可。

注意: 这里只需要求出 x 即可, 不需要求解 y 。

实现代码:

```
void exgcd(int a, int b, int &x, int &y) // 注意这里是 void 而不是 int, 因为我们最后要求的不是 exgcd, 而是里面的 x。
{
    if (b == 0) { x = 1; y = 0; return a; }
    int d = exgcd(b, a % b, y, x); // 由于要求的直接是 x, 就可以在这里直接把 x 和 y swap 一下即可。
    y = y - (a / b) * y;
}
```

2. 快速幂法:

3.8. 线性同余方程

给定整数 a, b, m , 求一个整数 x 满足 $ax \equiv b \pmod{m}$, 或者给出无解, 因为未知数的指数为 1, 所以我们称之为**一次同余方程**, 也称**线性同余方程**。

同样可以转化题意: $ax \equiv b \pmod{m}$ 可以转化为 $ax - bk = m$, 思路和上述 exgcd 法求解乘法逆元一样, 这里不再赘述。

根据 exgcd, 上述方程有解的充要条件是 $\gcd(a, m) | b$ 。与一般的线性方程相同, 这个方程的**特解**是 $x = x_0 \times b / \gcd(a, m)$ 。通解: 所有模 $m / \gcd(a, m)$ 与 x 同余的整数。