ACM算法与程序设计
（十二）
数论和基础数学

杜育根

Ygdu@sei.ecnu.edu.cn

12. 数论和基础数学

- 这节课主要讲解一些数论的知识，并介绍一些矩阵的应用，例如矩阵二分快速幂求解递推问题。

12.1.整除和取余

- 整除
- 定义 ：设 $a,b \in Z$，$a \neq 0$，如果存在 $q \in Z$ 使得 $b=aq$，那么就说 b可被 a 整除， 记做 $a|b$，且称 b是 a 的 倍数，a 是 b 的 约数（也称为除数、因数）。
- 带余数除法
- 设 a,b 是 2个正整数，且 $b \neq 0$，则存在唯一整数 q和r，使 $a=qb+r, 0 \leq r < |b|$。这个式子叫做带余数除法，并记余数 $r=a \bmod b$。例如 $13 \bmod 5=3$，$10 \bmod 2=0$。当 $r=0$的时候，就出现了整除，b 是 a的约数。
- 如果 n被 2除的余数为 0，称 n 为偶数，如果 n 被 2除的余数为 1，则称 n 为奇数。

关于整除有下面的一些性质

1. 若 $a|b$ 且 $a|c$，则 $\forall x,y$，有$a|xb+yc$.
2. 若 $a|b$ 且 $b|c$，则 $a|c$.
3. 设 $m\neq 0$，则 $a|b$，当且仅当 $ma|mb$.
4. 若 $a|b$ 且 $b|a$，则 $a=\pm b$.
5. 若 $a|b$ 且 $b\neq 0$，则 $|a|\leq|b|$.

- **关于余数的一些性质**

1. $(a+b) \bmod p=(a \bmod p+b \bmod p)$
2. $(a\times b) \bmod p=(a \bmod p\times b \bmod p) \bmod p$

12.2.最大公约数

1. 设 a和 b 是 2个不为 0 整数，如果 d|a 且 d|b，则称 d是 a与 b 的公约数。而 a,b 的所有公约数中最大的被称为 a,b的最大公约数，记作 gcd(a,b).

2. 设 a和 b 是 2个不为 000 整数，如果 a|d 且 b|d，则称 d是 a与 b的公倍数。而 a,b的所有公倍数中最小的被称为a,b的最小公倍数，记作 lcm(a,b).

- gcd和 lcm 的性质

1. 若 a|m，b|m，则 gcd(a,b) |m

2. 若 d|a，d|b，则 d|lcm(a,b)

3. lcm(a,b)=ab/gcd(a,b)

4. 设 m,a,b是正整数，则 lcm(ma,mb)=m×lcm(a,b)

5. 若 m是非零整数 a1,a2,a3,..., an 的公倍数，则 lcm(a1,a2,...,an)|m

欧几里得算法

- 欧几里得算法又称为辗转相除法。该算法用来快速计算2个整数的最大公约数。
- 欧几里得算法的原理就是一个公式：gcd(a,b)=gcd(b,a mod b)
- 证明：设 a=qb+r，其中 a,b,q,r 都是整数，我们只需证明gcd(a,b)=gcd(b,r)。设 d是 a与 b的公约数，即 d|a 且 d|b。注意到r=a−qb，根据整除的性质，d|a−qb，即 d|r。所以对于任意 (a,b) 的公约数，都是（b,r）的公约数，故（a,b）的最大公约数等于（b,r)最大公约数，即gcd(a,b)=gcd(b,r)。
- 借助递归来实现欧几里得算法，算法效率很高，
- 时间复杂度可以认为是 O(lgN)，可以
- 证明其递归层数不会超过4.7851lgN+1.6723，
- 其中 N=max(a,b)。

```
int gcd(int a, int b) {
    if (b == 0) {
        return a;
    }
    return gcd(b, a%b);
}
```

练习题：两仪剑法

- 两仪剑法是武当派武功的高级功夫，且必须 2个人配合使用威力才大。同时该剑法招数变化太快、太多。设武当弟子甲招数变化周期为 n，武当弟子乙招数变化周期为 m，两弟子同时使用该剑法，当 2人恰好同时达到招数变化周期结束时，威力最大，此时能将邪教妖人置于死地。请你计算威力最大时，每人用了多少招?
- 输入格式: 首先输入一个 t(t<100000) 表示测试组数。
- 接下来 t组输入，每组输入 2个数n,m(1≤n,m≤1000000000)。
- 输出格式: 对于每组输出，输出用了多少招数。
- 样例输入
- 3
- 2 3
- 8 9
- 4 8
- 样例输出
- 6
- 72
- 8

练习题：取石子游戏

12.3. 质数筛选

- 质数又称素数。质数定义为大于1的自然数中，除了1和它本身以外不再有其他约数的数称为质数。除了1 和质数以外的自然数称为合数。1既不是质数也不是合数。所以质数又被称为不可约数。
- 质数和合数性质：
- 1. $a>1$ 是合数，当且仅当 $a=bc$，其中 $1<b<a$，$1<c<a$
- 2. 合数必有质数因子
- 3. 如果 $d>1$, p是质数，且 $d|p$，则 $d=p$
- 4. 设 p是质数且 $p|ab$，则必有 $p|a$ 或者 $p|b$
- 5. 存在无穷多个质数

判断一个数是否是质数

- 关于怎么判断一个数 n是否是质数，最简单的方法是枚举 2到 $\sqrt{n}$，判断是否是n的约数

```
int is_prime(int n) {
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            return 0; // 不是质数
        }
    }
    return 1; // 是质数
}
```

素数筛选算法

- 更多的时候，需要预处理出一段区间上的质数，如果按照之前的方法一个一个判断，时间上肯定会承受不了。于是提出了一种预处理1 到 N上质数的算法，称为 Eratosthenes 筛选。
- 素数筛选算法的基本思想是我们先假设 2 到 N上所有数都是素数。我们从 2 开始扫描，对于一个数 i，可以得到 2i,3i…ki 不然都不是素数，因为至少这些数都有 i 这个因子，表示这些数不是素数，一直枚举到 N。对于每一个合数，它至少会被它的一个因子枚举到，所以可能证明这个算法的正确性。接下来分析时间复杂度，对于每个i，枚举的次数为n/i，所以总得时间复杂度为 N/2+N/3+…+N/N=O(NlgN)。再优化下：第一是基于每个合数必然有一个质因子，所以我们可以只用质数来筛选，第二是 j 的初始条件可以写成j=i*i，因为比如j=i*k(k<i)，那么 j 肯定被k筛选掉了。第三是可以只用$\sqrt{n}$之前的质数去筛选。优化之后的时间复杂度比 O(NlgN) 还要低得多。

优化前代码

```
for (int i = 2; i <= n; ++i) {
    is_prime[i] = 1;
}
for (int i = 2; i <= n; ++i) {
    for (int j = i * 2; j <= n; j += i) {
        is_prime[j] = 0;
    }
}
```

优化后代码

```
for (int i = 2; i <= n; ++i) {
    is_prime[i] = 1;
}
for (int i = 2; i * i <= n; ++i) {
    if (is_prime[i]) {
        for (int j = i * i; j <= n; j +=i) {
            is_prime[j] = 0;
        }
    }
}
```

练习题：哥德巴赫猜想

- 任意一个大于 2的偶数好像总能写成 2个质数的和。这个猜想就被称为哥德巴赫猜想。目前还没有证明这个猜想的正确性。告诉你一个整数 n ，让你用这个数去验证。注意 1 不是质数。
- 输入格式
- 输入一个偶数 n(2<n≤8000000)
- 输出格式
- 输出一个整数表示有多少对 (x,y) 满足 x+y=n(x≤y) 且 x,y 均为质数。
- 样例输入1
- 6
- 样例输出1
- 1
- 样例输入2
- 10
- 样例输出2
- 2

练习题：素数距离

12.4. 欧拉函数和积性函数

- 欧拉函数
- 概述
- 欧拉函数 $\phi(n)$：小于等于 n的所有数中与 n 互质的数的个数。
- 例如 $\phi(10) = 4$，因为 1,3,7,9 均和 10 互质。
- 公式
- $\phi(x) = x(1 - \frac{1}{p1})(1 - \frac{1}{p2}) \cdots (1 - \frac{1}{pn})$，其中 $p1, p2 \cdots pn$是 x的所有质因数。
- 例如：$\phi(7) = 7 \times (1 - 17) = 6, \phi(12) = 12 \times (1 - 12) \times (1 - 13) = 4$

积性函数

- **积性函数：对于正整数n的一个算术函数f(n)，若f(1)=1，且当a,b互质时f(ab)=f(a)f(b)，在数论上就称它为积性函数。完全积性函数指所有对于任何a，b都有性质f(ab)=f(a)f(b)的函数。**
- **欧拉函数 ϕ(n)为积性函数，但不是完全积性函数。**
- $\phi(p)=p-1$
- $\phi(pk)=p^k-p^{k-1}$
- 若 m,n互质，则有 $\phi(mn)=\phi(m)\phi(n)$。

欧拉定理

- 若 a,n互质，则有 $a^{\phi(n)} \equiv 1(mod\ n)$ 。
- **特例：费马小定理**
- $a^{p-1} \equiv 1(mod\ p)$，**其中a不是p的倍数，p为质数。**
- **其它性质**
- $\sum_{d|n} \phi(d) = n$

练习题：互质数个数

- 给定一个整数 n，请问有多少个整数 i满足条件：gcd(i,n)=1，2≤i≤n。
- 输入格式
- 输入一行，输入一个整数 n $(n \le 10^9)$。
- 输出格式
- 输出一行，输出一个整数，表示符合条件的整数个数。
- 样例输入
- 16
- 样例输出
- 8

练习题：质数原根

12.5. 扩展欧几里得

- 扩展欧几里得算法是用来在已知a,b 的情况下求解一组 x,y，使它们满足等式：$ax + by = \gcd(a, b) = d$（gcd 表示最大公约数，该方程的解一定存在）。
- 我们要计算的是 a和 b的最大公约数，并求出 x和 y使得 $ax + by = d$。
- 我们已经计算出了下一个状态：b和（a%b）的最大公约数，并且求出了一组 x_1, y_1 使得： $bx_1 + (a\%b)y_1 = d$， 那么这两个相邻的状态之间是否存在一种关系呢?
- 因为我们知道： $a\%b = a - \left(\frac{a}{b}\right) \times b$，那么：
- $d = b \times x_1 + \left[a - \left(\frac{a}{b}\right) \times b\right] \times y_1$
- $= b \times x_1 + a \times y_1 - \left(\frac{a}{b}\right) \times b \times y_1$
- $= a \times y_1 + b \times (x_1 - \frac{a}{b} \times y_1)$
- 所以： $x = y1, y = x_1 - \frac{a}{b} \times y_1$。

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

- 通过扩展欧几里得求出来的一组特解，假设特解为 $(x_0, \boldsymbol{y}_0)$。那么满足 $ax_0 + b\boldsymbol{y}_0 = d$。那么有

$$a(x_0 + k\frac{b}{d}) + b(\boldsymbol{y}_0 - k\frac{\boldsymbol{a}}{d}) = d$$

- 其中 $k \in Z$。这样我们可以写出方程的通解。

$$x = x_0 + k\frac{b}{d}$$

$$y = \boldsymbol{y}_0 - k\frac{\boldsymbol{a}}{d}$$

- 求解出来了 $ax + by = d$的解，那么对于 $ax + by = c$ （其中 c为任意正整数）的解其实也很简单了。只有当 d|c 的时候才有解。通解形式为

$$x = \frac{\boldsymbol{c}}{d}x_0 + k\frac{b}{d}$$

$$y = \frac{\boldsymbol{c}}{d}\boldsymbol{y}_0 - k\frac{\boldsymbol{a}}{d}$$

乘法逆元

- 给定两个整数 a和 p。假设存在一个 x使得 $ax \equiv 1(mod\, p)$。那么我们称 x为 a 关于p的乘法逆元。对于逆元的求法，可以借助扩展欧几里得，把上面的式子变形一下，变成 $ax + kp = 1$。就可以用扩展欧几里得求出一个 x。并且，我们也可以发现，a关于 p 的逆元存在的充要条件是 $\gcd(a, p) = 1$，也就是 a 和 p 必须互质。
- 乘法逆元的主要是用来求解除法取模的问题的。一个经典的问题是，假设 $C = \frac{A}{B}$，求 $C\%p$ 的值。但是由于 A很大，我们只知道 $A\%p$ 的值和 B 的值。假设 x 为 B 关于 p的逆元。通过下面的变换可以求解

$$\frac{A}{B}\%p = (\frac{A}{B} \cdot Bx)\%p = Ax\%p = A\%p \cdot x\%p$$

- 如果p是质数的时候，求a关于p的逆元有另外一种方法。$x = a^{p-2}\%p$。可以证明 $x * a\%p = a^{p-1}\%p = 1$（根据费马小定理），然后用二分快速幂来加速。

线性预处理逆元

- 有些情况，需要用到一段区间上的逆元，如果一个一个单独的求，可能比较慢。有一个可以在线性时间内预处理出来 1-n 的逆元的方法。

$$inv_i = (p - \frac{p}{i})\ inv_{p\%i} \% p$$

```
// p 必须为质数，p / i 为整除。
inv[1] = 1;
for (int i = 2; i <= n; ++i) {
    inv[i] = (p - p / i) * inv[p % i] % p;
}
```

证明：

$$i \cdot inv_i \% p = i * (p - \frac{p}{i}) inv_{p\%i} \% p = i * (p - \frac{p - p\%i}{i}) inv_{p\%i} \% p = p\%i * inv_{p\%i} \% p = 1$$

应用：

求组合数取模的时候可以用到。另 $fact_i = 1 * 2 * 3 \cdots i\%p$,

$invs_i = inv_1 * inv_2 \cdots inv_i \% p$

$$C_n^m \% p = \frac{n!}{m!(n-m)!} \% p = fact_n \cdot invs_m \cdot invs_{n-m} \% p$$

练习题：同余方程

- 已知整数 a和 b，求关于 x的同余方程 $ax \equiv 1 (mod\ b)$ 的最小正整数解。
- 输入格式
- 输入一行，输入两个整数 a, b （$2 \le a, b \le 2 \times 10^9$）。
- 输出格式
- 输出一行，输出一个整数，即同余方程的最小正整数解。输入数据保证一定有解。
- 样例输入
- 3 8
- 样例输出
- 3

练习题：机器人的相遇问题

12.6. 中国剩余定理（模线性方程组）

- 《孙子算经》中这样提到：有物不知其数，三三数之剩二，五五数之剩三，七七数之剩二。问物几何?
- 这句话的意思就是，一个整数除以三余二，除以五余三，除以七余二，问这个整数的值是多少。
- 因为《孙子算经》中首次提出了这种问题并给出了具体的解法，所以中国剩余定理也叫孙子定理。
- 中国剩余定理的一元模线性方程组如下：

$$x \equiv a1(mod\ m1)$$
$$x \equiv a2(mod\ m2)$$
$$\dots$$
$$x \equiv an(mod\ mn)$$

中国剩余定理

假设整数 $m_1, m_2, \cdots, m_n$ 两两互质，则方程组有解，通解可以构造如下：

设 $M = m_1 \times m_2 \times \cdots \times m_n = \prod m_i$，并设 $M_i = M/m_i, t_i = M_i^{-1}$，表示 M_i 模 m_i 意义下的倒数，即 $M_i t_i \equiv 1\% m_i$。

方程式的通解：

$$x = a_1 t_1 M_1 + a_2 t_2 M_2 + \cdots + a_n t_n M_n + kM = \sum_{i=1}^{n} a_i t_i M_i + kM$$

代码

```
int CRT(int a[], int m[], int n) {
    int M = 1;
    int ans = 0;
    for(int i = 1; i <= n; i++) {
        M *= m[i];
    }
    for(int i = 1; i <= n; i++) {
        int x, y;
        int Mi = M / m[i];
        exgcd(Mi, m[i], x, y);
        ans = (ans + Mi * x * a[i]) % M;
    }
    if(ans < 0) ans += M;
    return ans;
}
```

更一般的模线性方程组

- 若 mi 不能保证两两互质，那么我们可以对模方程两两合并，然后做 n−1 次扩展欧几里得即可。

练习题：整数复原

- 假设整数 m，分别对 $a1, a2, \cdots, ak$ 这 k个整数取余，可以得到余数 $r1, r2, \cdots, rk$。
- 现在已知余数 $r1, r2, \cdots, rk$ ，求最小的正整数解 m。
- 输入格式
- 第一行输入一个整数 k。
- 接下来输入 k行，一行输入两个整数 ai, ri。
- 输出格式
- 输出一行，输出一个整数，代表最小的正整数解 m，如果无解则输出 –1。
- 样例输入
- 2
  2 1
  4 3
- 样例输出
- 3

12.7. 二分快速幂

- 这节课我们学习一个快速求幂的算法。幂就是我们常说的次方，a^b 被称为 a 的 b次幂。C/C++在<math>头文件里有一个 pow(double a, double b) 函数能求 a^b，在 Java 中，java.lang.Math.pow(double a, double b) 用来求幂。上面 2个函数的实现都是 $O(b)$ 的。有些时候，我们遇到的 a,b 都是整数，并且 b 很大，比如 $b > 10^8$ 的时候，我们就需要一些优化方法。
- 二分快速幂的思想很简单，很容易理解。如果 b 是偶数，有 $a^b = (a^{\frac{b}{2}})^2$，如果b是奇数，有$a^b = (a^{\frac{b-1}{2}})^2 * a$。下面看代码，由于 b 很大，一般情况下都会伴随有取模。这样我们便得到了 $O(lgb)$ 的时间复杂度的算法计算 a^b次方。

```
int Pow_mod(int a, int b, int mod) {
    if (b == 0) {
        return 1%mod;
    }
    int temp = Pow_mod(a, b/2, mod);
    temp = temp * temp % mod;
    if (b%2 == 1) {
        temp = temp * a % mod;
    }
    return temp;
}
```

- 上面的写法用 dfs 写的，也可以用循环来写，假设

$$b = 2^{x_1} + 2^{x_2} + 2^{x_3} + \cdots + 2^{x_n},$$

- 于是 $a^b = a^{2^{x_1}+2^{x_2}+2^{x_3}+\cdots+2^{x_n}} = a^{2^{x_1}} * a^{2^{x_2}} * a^{2^{x_3}} * \cdots * a^{2^{x_n}}$。
- 那么我们实际上只需要加上 b 的 2 进制表示中 1的位置对应的权值。

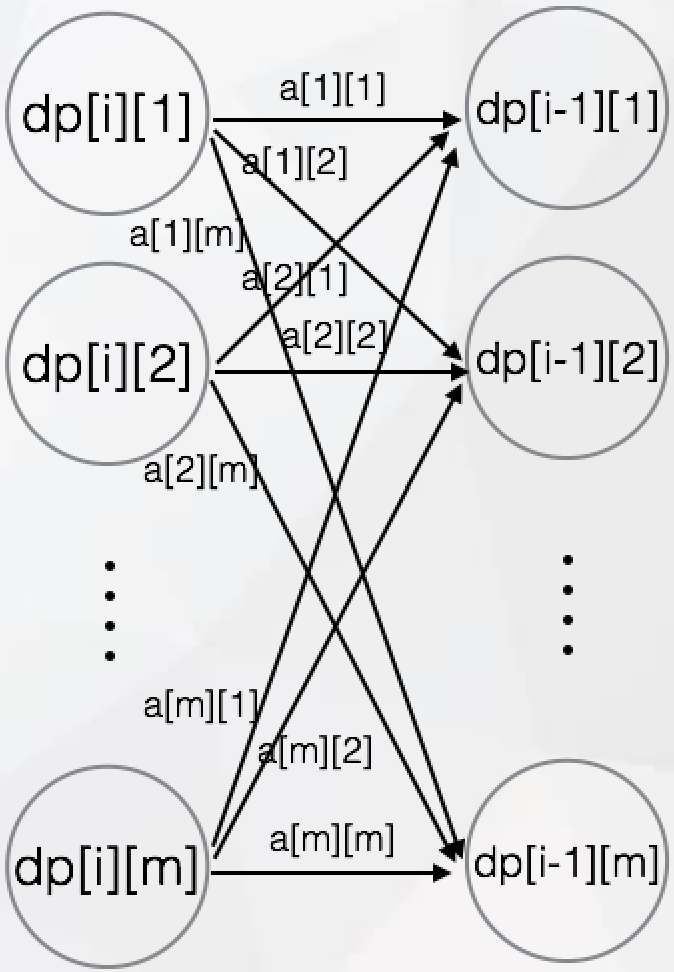
```
int Pow_mod(int a, int b, int mod){
    int res = 1, temp = a;
    for (; b; b /= 2) {
        if (b & 1) {
            res = res * temp % mod; // 2进制上这一位为1，乘上这一位权值
        }
        temp = temp * temp % mod; // 位数加1， 权值平方
    }
    return res;
}
```

练习题：气球消消乐

- 小明有 n只气球，把气球排成一排。初始时，气球都是白色，现在想用 m 种颜色给气球涂色，如果相邻的气球的颜色相同，这 2个气球会发生消消乐，小明希望你求出会发生消消乐的涂色方法有多少种。最后答案对 10^9+7 取模。
- 输入格式
- 输入两个整数$n(1 \le n \le 10^{12}), m(1 \le m \le 10^8)$
- 输出格式
- 输出一行表示答案。
- 样例输入1
- 2 2
- 样例输出1
- 2
- 样例输入2
- 3 4
- 样例输出2
- 28

12.8. 矩阵在竞赛中的应用

矩阵经常被应用到动态规划的转移当中，有一个 2 维的状态 $dp[n][m]$，有如下转移方程 $dp[i][j] = a_{j1}dp[i-1][1] + a_{j2}dp[i-1][2] + \cdots + a_{jm}dp[i-1][m]$。状态转移图就如下图



- 实际上，我们正好可以用一个矩阵乘法来表示这个过程，A矩阵是一个 m*m 的矩阵我们习惯称为状态转移矩阵。

$$
\begin{bmatrix} dp[i][1] \\ dp[i][2] \\ \cdots \\ dp[i][m] \end{bmatrix} = \begin{bmatrix} a[1][1] & a[1][2] & \cdots & a[1][m] \\ a[2][1] & a[2][2] & \cdots & a[2][m] \\ \cdots \\ a[m][1] & a[m][2] & \cdots & a[m][m] \end{bmatrix} \begin{bmatrix} dp[i-1][1] \\ dp[i-1][2] \\ \cdots \\ dp[i-1][m] \end{bmatrix}
$$

- 简单的表示，就写成用 $dp[i] = A \cdot dp[i-1]$，我们处理出来$dp[1]$，然后通过递推，可以得到，$dp[n] = A^{n-1}dp[1]$，最后我们通过矩阵的形式表示了一个 dp 转移。我们可以通过求 A^{n-1}然后算出$dp[n]$。

12.9. 矩阵二分快速幂优化dp

- 前一节我们讲到了利用矩阵来表示动态规划状态的转移，形如 $dp[n] = A^{n-1}dp[1]$，矩阵 A 称为转移矩阵。如果我们按照正常的动态规划的写法，这个解法的时间复杂度应该是 $O(nm^2)$，n是转移的次数， $O(m^2)$ 是一次转移的复杂度。朴素的动态规划的写法就相当于用 dp[1] 求出 dp[2]，然后用 dp[2] 求出 dp[3]，迭代下去，直到求出 dp[n]。
- 假设我们现在想一次性求出 dp[n]，可以先求出A^{n-1}。在求 A^{n-2} 的时候，因为矩阵运算和数值运算均满足结合律，我们可以利用二分快速幂的思想来优化，这里直接给出求矩阵快速幂的代码。同样地，由于幂的次数很大，求解矩阵快速幂一般也会伴有取模运算。

代码

```
int n; // 所有矩阵都是 n * n 的矩阵
struct matrix {
    int a[100][100];
};
matrix matrix_mul(matrix A, matrix B, int mod) {
    // 2 个矩阵相乘
    matrix C;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            C.a[i][j] = 0;
            for (int k = 0; k < n; ++k) {
                C.a[i][j] += A.a[i][k] * B.a[k][j] % mod;
                C.a[i][j] %= mod;
            }
        }
    }
    return C;
}
matrix unit() {    // 返回一个单位矩阵
    matrix res;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (i == j) {
                res.a[i][j] = 1;
            } else {
                res.a[i][j] = 0;
            }
        }
    }
    return res;
}
matrix matrix_pow(matrix A, int n, int mod) {
    // 快速求矩阵 A 的 n 次方
    matrix res = unit(), temp = A;
    for (; n; n /= 2) {
        if (n & 1) {
            res = matrix_mul(res, temp, mod);
        }
        temp = matrix_mul(temp, temp, mod);
    }
    return res;
}
```

- 矩阵二分快速幂的一个经典应用就是斐波那契数列的求解。我们都知道斐波那契数列数列的第 n项的递推公式是 f[n]=f[n−1]+f[n−2](n>2)，我们巧妙的构建一个转移矩阵

$$A=\begin{bmatrix}1&1\\1&0\end{bmatrix}$$

$$\begin{bmatrix}f[i]\\f[i-1]\end{bmatrix}=A\cdot\begin{bmatrix}f[i-1]\\f[i-2]\end{bmatrix}=\begin{bmatrix}1&1\\1&0\end{bmatrix}\begin{bmatrix}f[i-1]\\f[i-2]\end{bmatrix}$$

- 这个转移翻译过来的意思就是 $f[i]=f[i-1]+f[i-2], f[i-1]=f[i-1]$。
- 于是进而可以得到

$$\begin{bmatrix}f[i]\\f[i-1]\end{bmatrix}=A^{i-2}\cdot\begin{bmatrix}f[2]\\f[1]\end{bmatrix}$$

- 这样就把斐波拉契数列求解的时间复杂度优化成 $O(2^3lgn)$。到这里如果你都能理解以后，后面的过程就已经很简单了。

练习题：Fib数列问题之二

- 用 fib(n) 表示斐波那契数列的第 n项，现在要求你求 fib(n) mod m。fib(1)=1,fib(2)=1。
- 输入格式
- 输入 2个整数 $n(1 \le n \le 10^{18})$，$m(2 \le m \le 100000000)$。
- 输出格式
- 输出 fib(n) 对 m 取模的值。
- 样例输入1
- 4 10
- 样例输出1
- 3
- 样例输入2
- 100000000 100000000
- 样例输出2
- 60546875

练习题：小明倒水

练习题：垒骰子