第二章 数论目录

1、排列组合

1）一般组合

```
int n, m;       //  从 n 个数中选出 m 个构成组合
int rcd[MAX_N]; //  记录每个位置填的数
int num[MAX_N]; //  存放输入的 n 个数
void select_combination(int l, int p)
{
    int i;
    if (l == m){        //  若选出了 m 个数, 则打印
        for (i = 0; i < m; i++){
            printf("%d", rcd[i]);
            if (i < m - 1)
            printf(" ");
        }
        printf("\n");
        return ;
    }
    for (i = p; i < n; i++) //  上个位置填的是 num[p-1],本次从 num[p]开始试探
    {     rcd[l] = num[i];   //  在 l 位置放上该数
         select_combination(l + 1, i + 1);  } //  填下一个位置
}
int read_data()
{
    int i;
    if (scanf("%d%d", &n, &m) == EOF)        return 0;
    for (i = 0; i < n; i++)               scanf("%d", &num[i]);
    return 1;
}
```

2）不重复排列

```
int n, m;          //  共有 n 个数,其中互不相同的有 m 个
int rcd[MAX_N];     //  记录每个位置填的数
int used[MAX_N];    //  标记 m 个数可以使用的次数
int num[MAX_N];     //  存放输入中互不相同的 m 个数
void unrepeat_permutation(int l)
{
    int i;
    if (l == n)     //  填完了 n 个数,则输出
    {
        for (i = 0; i < n; i++)
        {
            printf("%d", rcd[i]);
            if (i < n - 1)     printf(" ");
        }
```

```
        printf("\n");
        return ;
    }
    for (i = 0; i < m; i++)             //  枚举 m 个本质不同的数
    {
        if (used[i] > 0)             //  若数 num[i]还没被用完,则可使用次数减
        {
            used[i]--;
            rcd[l] = num[i];            //  在 l 位置放上该数
            unrepeat_permutation(l+1);  //  填下一个位置
            used[i]++;                 //  可使用次数恢复
        }
    }
}
int read_data()
{
    int i, j, val;
    if (scanf("%d", &n) == EOF)        return 0;
    m = 0;
    for (i = 0; i < n; i++)
    {
        scanf("%d", &val);
        for (j = 0; j < m; j++)
        {
            if (num[j] == val)     used[j]++; break;
        }
        if (j == m)
        {
            num[m] = val;
            used[m++] = 1;
        }
    }
    return 1;
}
```

3）不重复组合

```
int n, m;           //  输入 n 个数,其中本质不同的有 m 个
int rcd[MAX_N];      //  记录每个位置填的数
int used[MAX_N];     //  标记 m 个数可以使用的次数
int num[MAX_N];      //  存放输入中本质不同的 m 个数
void unrepeat_combination(int l, int p)
{
    int i;
    for (i = 0; i < l; i++)      //  每次都输出
```

```
    {
        printf("%d", rcd[i]);
        if (i < l - 1)       printf(" ");
    }
    printf("\n");
    for (i = p; i < m; i++)     //  循环依旧从 p 开始,枚举剩下的本质不同的数
    {
        if (used[i] > 0)       //  若还可以用, 则可用次数减
        {
            used[i]--;
            rcd[l] = num[i];    //  在 l 位置放上该
            unrepeat_combination(l+1, i);   //  填下一个位置
            used[i]++;         //可用次数恢复
        }
    }
}
int read_data()
{
    int i, j, val;
    if (scanf("%d", &n) == EOF)     return 0;
    m = 0;
    for (i = 0; i < n; i++)   {
        scanf("%d", &val);
        for (j = 0; j < m; j++)    {
            if (num[j] == val)     {
                used[j]++;
                break;
            }
        }
        if (j == m)     {
            num[m] = val;
            used[m++] = 1;
        }
    }
    return 1;
}
```

4）全排列枚举

```
int n;                      //  共 n 个数
int rcd[MAX_N];             //  记录每个位置填的数
int used[MAX_N];            //  标记数是否用过
int num[MAX_N];             //  存放输入的 n 个数
void full_permutation(int l)
{
```

```
    int i;
    if (l == n)   {
        for (i = 0; i < n; i++)      {
            printf("%d", rcd[i]);
            if (i < n-1)         printf(" ");
        }
        printf("\n");
        return ;
    }
    for (i = 0; i < n; i++)         //  枚举所有的数(n 个),循环从开始
        if (!used[i])
        {                            //  若 num[i]没有使用过, 则标记为已使用
            used[i] = 1;
            rcd[l] = num[i];        //  在 l 位置放上该数
            full_permutation(l+1);  //  填下一个位置
            used[i] = 0;            //  清标记
        }
}
int read_data()
{
    int i;
    if (scanf("%d", &n) == EOF)      return 0;
    for (i = 0; i < n; i++)     scanf("%d", &num[i]);
    for (i = 0; i < n; i++)     used[i] = 0;
    return 1;
}
```

5）全组合

```
int n;          //  共 n 个数
int rcd[MAX_N]; //  记录每个位置填的数
int num[MAX_N]; //  存放输入的 n 个数
void full_combination(int l, int p)
{
    int i;
    for (i = 0; i < l; i++)     {          //  每次进入递归函数都输出
        printf("%d", rcd[i]);
        if (i < l-1)  printf(" ");
    }
    printf("\n");
    for (i = p; i < n; i++)     {          //  循环同样从 p 开始,但结束条件变为 i>=n
        rcd[l] = num[i];                //  在 l 位置放上该数
        full_combination(l + 1, i + 1); //  填下一个位置
    }
}
```

```
int read_data()
{
    int i;
    if (scanf("%d", &n) == EOF)     return 0;
    for (i = 0; i < n; i++)    scanf("%d", &num[i]);
    return 1;
}
```

6）类循环排列

```
int n, m;                      //  相当于 n 重循环,每重循环长度为 m
int rcd[MAX_N];                //  记录每个位置填的数
void loop_permutation(int l){
    int i;
    if (l == n)                //  相当于进入了 n 重循环的最内层
    {
        for (i = 0; i < n; i++)
        {
            cout << rcd[i];
            if (i < n-1)    cout << " ";
        }
        cout << "\n";
        return ;
    }
    for (i = 0; i < m; i++)    {     //  每重循环长度为 m
        rcd[l] = i;            //  在 l 位置放 i
        loop_permutation(l + 1);  //  填下一个位置
    }
}
```

7）计数方法

```
C(n,k)+C(n,k+1) = c(n+1,k+1)
C(n,k+1)=C(n,k)*(n-k)/(k+1)
```

2、exgcd&&求逆元

```
/*
 *  扩展欧几里得法（求 ax + by = gcd）
 */
//  返回 d = gcd(a, b);和对应于等式 ax + by = d 中的 x、y
long long extendGcd(long long a, long long b, long long &x, long long &y)
{
    if (a == 0 && b == 0)    return -1;  //  无最大公约数
    if (b == 0)
    {
        x = 1;
```

```
        y = 0;
        return a;
    }
    long long d = extendGcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
//  求逆元 ax = 1(mod n)
long long modReverse(long long a, long long n)
{
    long long x, y;
    long long d = extendGcd(a, n, x, y);
    if (d == 1)      return (x % n + n) % n;
    else   return -1;  //  无逆元
}
/*
 *  简洁写法 I
 *  只能求 a < m 的情况，且 a 与 m 互质
 *  求 ax = 1(mod m)的 x 值，即逆元(0 < a < m)
 */
long long inv(long long a, long long m)
{
    if (a == 1)      return 1;
    return inv(m % a, m) * (m - m / a) % m;
}
```

3、gcd && exgcd

```
int gcd(int x, int y)
{
    if (!x || !y)     return x > y ? x : y;
    for (int t; t = x % y, t; x = y, y = t) ;
    return y;
}
/*
 *  Çóx£¬yÊ¹µÃgcd(a, b) = a * x + b * y;
 */
int extgcd(int a, int b, int &x, int &y){
    if (b == 0)    {
        x = 1;
        y = 0;
        return a;
    }
    int d = extgcd(b, a % b, x, y);
    int t = x;
```

```
     x = y;
     y = t - a / b * y;
     return d;
 }
```

4、判断连续素数

```
/*
 *  素数筛选，查找出小于等于 MAXN 的素数
 *  prime[0]存素数的个数
 */
const int MAXN = 100000;
int prime[MAXN + 1];
void getPrime(){
    memset(prime, 0, sizeof(prime));
    for (int i = 2; i <= MAXN; i++)   {
        if (!prime[i])    prime[++prime[0]] = i;
        for (int j = 1; j <= prime[0] && prime[j] <= MAXN / i; j++)
        {
            prime[prime[j] * i] = 1;
            if (i % prime[j] == 0)     break;
        }}}
```

5、合数的分解

```
/*
 *  合数的分解需要先进行素数的筛选
 *  factor[i][0]存放分解的素数
 *  factor[i][1]存放对应素数出现的次数
 *  fatCnt 存放合数分解出的素数个数(相同的素数只算一次)
 */
const int MAXN = 10000;
int prime[MAXN + 1];
//  获取素数
void getPrime(){
    memset(prime, 0, sizeof(prime));
    for (int i = 2; i <= MAXN; i++)    {
        if (!prime[i])      prime[++prime[0]] = i;
        for (int j = 1; j <= prime[0] && prime[j] <= MAXN / i; j++)
        {
            prime[prime[j] * i] = 1;
            if (i % prime[j] == 0)     break;
        }
    }
    return ;
}
```

```
long long factor[100][2];
int fatCnt;
//  合数分解
int getFactors(long long x)
{
    fatCnt = 0;
    long long tmp = x;
    for (int i = 1; prime[i] <= tmp / prime[i]; i++)    {
        factor[fatCnt][1] = 0;
        if (tmp % prime[i] == 0)    {
            factor[fatCnt][0] = prime[i];
            while (tmp % prime[i] == 0)    {
                factor[fatCnt][1]++;
                tmp /= prime[i];
            }
            fatCnt++;
        }
    }
    if (tmp != 1){
        factor[fatCnt][0] = tmp;
        factor[fatCnt++][1] = 1;
    }
    return fatCnt;
}
int main(){
     ll n ;
     readll(n);
     getPrime(); // 得到质数表
     fatCnt = getFactors(n);
     rep(i , fatCnt){
          cout << factor[i][0] << " ";
     }
```

6、快速幂取模

```
/*
 *  欧拉函数法
 *  a 和 m 互质
 */
//  快速幂取模
long long powM(long long a, long long b, long long m){
    long long tmp = 1;
    if (b == 0)     return 1;
    if (b == 1)   return a % m;
    tmp = powM(a, b >> 1, m);
```

```
    tmp = tmp * tmp % m;
    if (b & 1)     tmp = tmp * a % m;
    return tmp;
}
long long inv(long long a, long long m)   { return powM(a, m - 2, m);}
```

7、最大 1 矩阵(01 组成的全是 1 的最大子矩阵)

```
const int N = 1000;
bool a[N][N];
int Run(const int &m, const int &n) {     // a[1...m][1...n]  // O(m*n)
    int i, j, k, l, r, max=0;
    int col[N];
    for (j = 1; j <= n; j++)   {
        if (a[1][j] == 0 )    col[j] = 0;
        else{
            for (k = 2; k <= m && a[k][j] == 1; k++);
            col[j] = k - 1;
        }
    }
    for (i = 1; i <= m; i++) {
        if (i > 1)
        {
            for (j = 1; j <= n; j++)
            {
                if (a[i][j] == 0)    col[j] = 0;
                else{
                    if (a[i - 1][j] == 0)
                    {
                        for (k = i + 1; k <= m && a[k][j] == 1; k++);
                        col[j] = k-1;
                    }}}}
        for (j = 1; j <= n; j++) {
            if (col[j] >= i) {
                for (l = j - 1; l > 0 && col[l] >= col[j]; --l);
                l++;
                for (r = j + 1; r <= n && col[r] >= col[j]; ++r);
                r--;
                int res = (r - l + 1) * (col[j] - i + 1);
                if( res > max )     max = res;
            }}}
    return max;
}
```

8、模线性方程

```
/*模线性方程 a * x = b (% n) */
void modeq(int a, int b, int n){
    int e, i, d, x, y;
    d = extgcd(b, a % b, x, y);
    if (b % d > 0)     cout << "No answer!\n";
    else    {
        e = (x * (b / d)) % n;
        for (i = 0; i < d; i++) cout << i + 1 << "-th ans:" << (e + i * (n / d)) % n << '\n';
    }
    return ;
}
/*   模线性方程组
*  a = B[1](% W[1]); a = B[2](% W[2]); ... a = B[k](% W[k]);
  *  其中 W，B 已知，W[i] > 0 且 W[i]与 W[j]互质，求 a（中国剩余定理） */
int china(int b[], int w[], int k){
    int i, d, x, y, m, a = 0, n = 1;
    for (i = 0; i < k; i++)  n *= w[i];  //  注意不能 overflow
    for (i = 0; i < k; i++) {
        m = n / w[i];
        d = extgcd(w[i], m, x, y);
        a = (a + y * m * b[i]) % n;
    }
    if (a > 0)    return a;
    else  return (a + n);//最小正解
}
    /*w[i]和 w[j]不互质 */
const int MAXN = 11;
int n, m;
int a[MAXN], b[MAXN];
int main(int argc, const char * argv[])
{
    int T;
    cin >> T;
    while (T--){
        cin >> n >> m;
        for (int i = 0; i < m; i++)   cin >> a[i];
        for (int i = 0; i < m; i++)      cin >> b[i];
        ll ax = a[0], bx = b[0], x, y;
        int flag = 0;
        for (int i = 1; i < m; i++)
        {
            ll d = extgcd(ax, a[i], x, y);
            if ((b[i] - bx) % d != 0){
```

```
            flag = 1;   //  无整数解
            break;
        }
        ll tmp = a[i] / d;
        x = x * (b[i] - bx) / d;    //  约分
        x = (x % tmp + tmp) % tmp;
        bx = bx + ax * x;
        ax = ax * tmp;              //  ax = ax * a[i] / d
    }
    if (flag == 1 || n < bx)     puts("0");
    else{
        ll ans = (n - bx) / ax + 1;
        if (bx == 0)     ans--;
        printf("%lld\n", ans);
    }
  }
  return 0;
}
```

9、欧拉函数单独求解

```
int IT_MAX = 1 << 19;
int MOD = 1000000007;
const int INF = 0x3f3f3f3f;
const ll LL_INF = 0x3f3f3f3f3f3f3f3f;
const db PI = acos(-1);
const db ERR = 1e-10;
const int MAX_N = 100005;
bool cmp (int a , int b){ return a>b;}
unsigned euler(unsigned x); //小于等于 x 中的数与 x 互质的个数
int main(){
    int n ;
    read(n);
    cout << euler(n) << endl;
    return 0;
}
/*单独求解的本质是公式的应用 */
unsigned euler(unsigned x)
{
    unsigned i, res = x;    //  unsigned == unsigned int
    for (i = 2; i < (int)sqrt(x * 1.0) + 1; i++) {
        if (!(x % i)){
            res = res / i * (i - 1);
            while (!(x % i))     x /= i;     //  保证 i 一定是素数
        }
```

```
    }
    if (x > 1)    res = res / x * (x - 1);
    return res;
}
```

10、欧拉线性筛

```
/*同时得到欧拉函数和素数表 */
const int MAXN = 10000000;
bool check[MAXN + 10];
int phi[MAXN + 10];
int prime[MAXN + 10]; //tot 个素数
int tot;    //  素数个数
void phi_and_prime_table(int N); // N 以内的素数
int main(){
     int n;
     scanf("%d",&n);
     phi_and_prime_table(n);
     for(int i = 0 ; i < tot ; i ++){
         cout << prime[i] << " ";
     }
     cout << endl << tot << endl;
     return 0;
}
void phi_and_prime_table(int N)
{
    memset(check, false, sizeof(check));
    phi[1] = 1;
    tot = 0;
    for (int i = 2; i <= N; i++){
        if (!check[i]){
            prime[tot++] = i;
            phi[i] = i - 1;
        }
        for (int j = 0; j < tot; j++){
            if (i * prime[j] > N)     break;
            check[i * prime[j]] = true;
            if (i % prime[j] == 0){
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            else phi[i * prime[j]] = phi[i] * (prime[j] - 1);
        }
    }
    return ;     }
```

11、求阶乘逆元

```
const ll MOD = 1e9 + 7;     //  必须为质数才管用
const ll MAXN = 1e5 + 3;
ll fac[MAXN];       //  阶乘
ll inv[MAXN];       //  阶乘的逆元
ll QPow(ll x, ll n)
{
    ll ret = 1;
    ll tmp = x % MOD;
    while (n){
        if (n & 1)      ret = (ret * tmp) % MOD;
        tmp = tmp * tmp % MOD;
        n >>= 1;
    }
    return ret;
}
void init(){
    fac[0] = 1;
    for (int i = 1; i < MAXN; i++)    fac[i] = fac[i - 1] * i % MOD;
    inv[MAXN - 1] = QPow(fac[MAXN - 1], MOD - 2);
    for (int i = MAXN - 2; i >= 0; i--)inv[i] = inv[i + 1] * (i + 1) % MOD;
}
```

12、波利亚计数(不同颜色的珠子组成项链)

```
/*c 种颜色的珠子，组成长为 s 的项链，项链没有方向和起始位置*/
int gcd(int a, int b)   return b ? gcd(b, a % b) : a;
int main(int argc, const char * argv[])
{
      int c, s;
      while (cin >> c >> s){
      int k;
      long long p[64];
      p[0] = 1; // power of c
      for (k = 0; k < s; k++)    p[k + 1] = p[k] * c;
      // reflection part
      long long count = s & 1 ? s * p[s / 2 + 1] : (s / 2) * (p[s / 2] + p[s / 2 + 1]);
      // rotation part
      for (k = 1 ; k <= s ; k++){
          count += p[gcd(k, s)];
          count /= 2 * s;
      }
        cout << count << '\n';
      }
      return 0;   }
```

13、简易素数筛

```
/* 素数筛选，判断小于 MAXN 的数是不是素数
*notprime 是一张表，false 表示是素数，true 表示不是*/
const int MAXN = 1000010;
bool notprime[MAXN];
void init()
{
    memset(notprime, false, sizeof(notprime));
    notprime[0] = notprime[1] = true;
    for (int i = 2; i < MAXN; i++){
        if (!notprime[i]){
            if (i > MAXN / i)    continue;   // 阻止后边 i * i 溢出(或者 i,j 用 long long)
        // 直接从 i * i 开始就可以，小于 i 倍的已经筛选过了
        for (int j = i * i; j < MAXN; j += i)    notprime[j] = true;
        }
    }
}
```

14、约瑟夫环

```
/* n 个人(编号 1...n),先去掉第 m 个数,然后从 m+1 个开始报 1,报到 k 的退出,剩下的人继续从 1 开
始报数.求胜利者的编号.*/
int main(int argc, const char * argv[]){
    int n, k, m;
    while (cin >> n >> k >> m, n || k || m){
        int i, d, s = 0;
        for (i = 2; i <= n; i++) s = (s + k) % i;
        k = k % n;
        if (k == 0)k = n;
        d = (s + 1) + (m - k);
        if (d >= 1 && d <= n)cout << d << '\n';
        else if (d < 1)cout << n + d << '\n';
        else if (d > n)cout << d % n << '\n';
}
return 0;
}
```

15、阶乘位数

```
#define PI 3.1415926
int main(){
int n, a;
    while (~scanf("%d", &n)){
    a = (int)((0.5 * log(2 * PI * n) + n * log(n) - n) / log(10));
    printf("%d\n", a + 1);   }
    return 0;   }
```

16、阶乘最后一个非零位

```
/*阶乘最后非零位 复杂度 O(nlongn)
* 返回改为，n 以字符串方式传入*/
#define MAXN 10000
const int mod[20] = {1, 1, 2, 6, 4, 2, 2, 4, 2, 8, 4, 4, 8, 4, 6, 8, 8, 6, 8, 2};
int lastDigit(char *buf){
    int len = (int)strlen(buf);
    int a[MAXN], i, c, ret = 1;
    if (len == 1)      return mod[buf[0] - '0'];
    for (i = 0; i < len; i++)    a[i] = buf[len - 1 - i] - '0';
    for (; len; len -= !a[len - 1]){
    ret = ret * mod[a[1] % 2 * 10 + a[0]] % 5;
    for (c = 0, i = len - 1; i >= 0; i--){
        c = c * 10 + a[i];
        a[i] = c / 5;
        c %= 5;
    }
}
return ret + ret % 2 * 5;
}
```

17、随机大素数测试+大素数快速分解

```
const int MAXN = 65;
long long x[MAXN];
set<long long>prime;
long long qpow(long long a, long long b, long long p) {
    long long ans = 1;
    while(b) {
    if(b&1LL) ans = ans*a%p;
      a = a*a%p;
      b >>= 1;
    }
    return ans;
}
bool Miller_Rabin(long long n) {
    if(n == 2) return true;
    int s = 20, i, t = 0;
    long long u = n-1;
    while(!(u & 1)) {
    t++;
    u >>= 1;
    }
while(s--) {
    long long a = rand()%(n-2)+2;
```

```
        x[0] = qpow(a, u, n);
        for(i = 1; i <= t; i++) {
        x[i] = x[i-1]*x[i-1]%n ;
        if(x[i] == 1 && x[i-1] != 1 && x[i-1] != n-1) return false;
        }
        if(x[t] != 1) return false;
        }
return true;
}
long long gcd(long long a, long long b)         return b ? gcd(b, a%b) : a;
long long Pollard_Rho(long long n, int c) {
        long long i = 1, k = 2, x = rand()%(n-1)+1, y = x;
        while(true) {
        i++;
        x = (x*x%n + c)%n;
        long long p = gcd((y-x+n)%n, n);
        if(p != 1 && p != n) return p;
        if(y == x) return n;
        if(i == k) {
        y = x;
        k <<= 1;
        }
    }
}
void find(long long n, int c) {
        if(n == 1) return;
        if(Miller_Rabin(n)) {
                prime.insert(-n);
        return;
        }
        long long p = n, k = c;
        while(p >= n) p = Pollard_Rho(p, c--);
        find(p, k);
        find(n/p, k);
}
```

18、高斯消元求 1 类开关问题的解

```
//   高斯消元法求方程组的解
//   适用于格子涂色一类问题
const int MAXN = 300;
//   有 equ 个方程，var 个变元。增广矩阵行数为 equ，列数为 var＋1，分别为 0 到 var
int equ, var;
int a[MAXN][MAXN];   //   增广矩阵
int x[MAXN];               //   解集
```

```
int free_x[MAXN];     //  用来存储自由变元（多解枚举自由变元可以使用）
int free_num;           //  自由变元的个数
//  返回值为－1 表示无解，为 0 是唯一解，否则返回自由变元个数
int Gauss(){
        int max_r, col, k;
        free_num = 0;
            for (k = 0, col = 0; k < equ && col < var; k++, col++){
                    max_r = k;
                for (int i = k + 1; i < equ; i++){
                    if (abs(a[i][col]) > abs(a[max_r][col]))    max_r = i;
                }
                if (a[max_r][col] == 0){
                k--;
                free_x[free_num++] = col;     //  这是自由变元
                    continue;
                    }
            if (max_r != k)    for (int j = col; j < var + 1; j++)        swap(a[k][j], a[max_r][j]);
            for (int i = k + 1; i < equ; i++){
            if (a[i][col] != 0)      for (int j = col; j < var + 1; j++)      a[i][j] ^= a[k][j];
            }
      }
        for (int i = k; i < equ; i++)       if (a[i][col] != 0)    return -1;   //  无解
        if (k < var)       return var - k; //  自由变元个数
            //  唯一解，回代
        for (int i = var - 1; i >= 0; i--) {
        x[i] = a[i][var];
        for (int j = i + 1; j < var; j++)        x[i] ^= (a[i][j] && x[j]);
        }
    return 0;
}
```

19、FFT(多项式处理)

```
const double PI = acos(-1.0);
//  复数结构体
struct Complex
{
    double x, y;      //  实部和虚部  x + yi
    Complex(double _x = 0.0, double _y = 0.0){
        x = _x;
        y = _y;
    }
    Complex operator - (const Complex &b) const{
        return Complex(x - b.x, y - b.y);
    }
```

```
    Complex operator + (const Complex &b) const{
        return Complex(x + b.x, y + b.y);
    }
    Complex operator * (const Complex &b) const{
        return Complex(x * b.x - y * b.y, x * b.y + y * b.x);
    }
};
//  进行 FFT 和 IFFT 前的反转变换
//  位置 i 和（i 二进制反转后的位置）互换
//  len 必须去 2 的幂
void change(Complex y[], int len)
{
    int i, j, k;
    for (i = 1, j = len / 2; i < len - 1; i++){
        if (i < j)   swap(y[i], y[j]);
        //  交换护卫小标反转的元素，i < j 保证交换一次
        //  i 做正常的+1，j 左反转类型的+1，始终保持 i 和 j 是反转的
        k = len / 2;
        while (j >= k){
            j -= k;
            k /= 2;}
        if (j < k)   j += k;
    }
    return ;
}
//  FFT
//  len 必须为 2 ^ k 形式
//  on == 1 时是 DFT，on == -1 时是 IDFT
void fft(Complex y[], int len, int on){
    change(y, len);
    for (int h = 2; h <= len; h <<= 1){
        Complex wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));
        for (int j = 0; j < len; j += h){
            Complex w(1, 0);
            for (int k = j; k < j + h / 2; k++){
                Complex u = y[k];
                Complex t = w * y[k + h / 2];
                y[k] = u + t;
                y[k + h / 2] = u - t;
                w = w * wn;
            }
        }
    }
    if (on == -1)     for (int i = 0; i < len; i++)     y[i].x /= len;     }
```

20、FFT 优化大数乘法 1

```
//	FFT
/*HDU 1402  求高精度乘法	A * B Problem Plus */
const double PI = acos(-1.0);
//	复数结构体
struct Complex{
	double x, y;	//	实部和虚部  x + yi
	Complex(double _x = 0.0, double _y = 0.0){
		x = _x;
		y = _y;
	}
	Complex operator - (const Complex &b) const{
		return Complex(x - b.x, y - b.y);
	}
	Complex operator + (const Complex &b) const{
		return Complex(x + b.x, y + b.y);
	}
	Complex operator * (const Complex &b) const{
		return Complex(x * b.x - y * b.y, x * b.y + y * b.x);
	}
};
//	进行 FFT 和 IFFT 前的反转变换
//	位置 i 和（i 二进制反转后的位置）互换
//	len 必须去 2 的幂
void change(Complex y[], int len){
	int i, j, k;
	for (i = 1, j = len / 2; i < len - 1; i++){
		if (i < j)	swap(y[i], y[j]);
		//	交换护卫小标反转的元素，i < j 保证交换一次
		//	i 做正常的+1，j 左反转类型的+1，始终保持 i 和 j 是反转的
		k = len / 2;
		while (j >= k){
			j -= k;
			k /= 2;
		}
		if (j < k)	j += k;
	}
	return ;
}
//	FFT
//	len 必须为 2 ^ k 形式
//	on == 1 时是 DFT，on == -1 时是 IDFT
void fft(Complex y[], int len, int on){
	change(y, len);
```

```
    for (int h = 2; h <= len; h <<= 1){
        Complex wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));
        for (int j = 0; j < len; j += h){
            Complex w(1, 0);
            for (int k = j; k < j + h / 2; k++){
                Complex u = y[k];
                Complex t = w * y[k + h / 2];
                y[k] = u + t;
                y[k + h / 2] = u - t;
                w = w * wn;
            }
        }
    }
    if (on == -1){
        for (int i = 0; i < len; i++)       y[i].x /= len;
    }
}
const int MAXN = 200010;
Complex x1[MAXN], x2[MAXN];
char str1[MAXN / 2], str2[MAXN];
int sum[MAXN];
int main(int argc, const char * argv[])
{
    while (cin >> str1 >> str2)
    {
        int len1 = (int)strlen(str1);
        int len2 = (int)strlen(str2);
        int len = 1;
        while (len < len1 * 2 || len < len2 * 2)    len <<= 1;
        for (int i = 0; i < len1; i++){
            x1[i] = Complex(str1[len1 - 1 - i] - '0', 0);
        }
        for (int i = len1; i < len; i++){
            x1[i] = Complex(0, 0);
        }
        for (int i = 0; i < len2; i++){
            x2[i] = Complex(str2[len2 - 1 - i] - '0', 0);
        }
        for (int i = len2; i < len; i++){
            x2[i] = Complex(0, 0);
        }
        //  求 DFT
        fft(x1, len, 1);
        fft(x2, len, 1);
```

```
        for (int i = 0; i < len; i++){
            x1[i] = x1[i] * x2[i];
        }
        fft(x1, len, -1);
        for (int i = 0; i < len; i++){
            sum[i] = (int)(x1[i].x + 0.5);
        }
        for (int i = 0; i < len; i++){
            sum[i + 1] += sum[i] / 10;
            sum[i] %= 10;
        }
        len = len1 + len2 - 1;
        while (sum[len] <= 0 && len > 0)    len--;
        for (int i = len; i >= 0; i--)    printf("%c", sum[i] + '0');
        putchar('\n');
    }
    return 0;
}
```

21、FFT 优化高精度乘法 2

```
template <class T>
void read(T &x){
    char c;
    bool op = 0;
    while(c = getchar(), c < '0' || c > '9')
    if(c == '-') op = 1;
        x = c - '0';
    while(c = getchar(), c >= '0' && c <= '9')
        x = x * 10 + c - '0';
    if(op) x = -x;
}
template <class T>
void write(T x){
    if(x < 0) putchar('-'), x = -x;
    if(x >= 10) write(x / 10);
    putchar('0' + x % 10);
}
const int N = 1000005;
const double PI = acos(-1);
typedef complex <double> cp;
char sa[N], sb[N];
int n = 1, lena, lenb, res[N];
cp a[N], b[N], omg[N], inv[N];
void init(){
```

```
	for(int i = 0; i < n; i++){
		omg[i] = cp(cos(2 * PI * i / n), sin(2 * PI * i / n));
		inv[i] = conj(omg[i]);
	}
}
void fft(cp *a, cp *omg){
	int lim = 0;
	while((1 << lim) < n) lim++;
	for(int i = 0; i < n; i++){
		int t = 0;
		for(int j = 0; j < lim; j++)
			if((i >> j) & 1) t |= (1 << (lim - j - 1));
if(i < t) swap(a[i], a[t]); // i < t  的限制使得每对点只被交换一次（否则交换两次相当于没交换）
	}
	for(int l = 2; l <= n; l *= 2){
		int m = l / 2;
	for(cp *p = a; p != a + n; p += l)
		for(int i = 0; i < m; i++){
			cp t = omg[n / l * i] * p[i + m];
			p[i + m] = p[i] - t;
			p[i] += t;
		}
	}
}
int main(){
	scanf("%s%s", sa, sb);
	lena = strlen(sa), lenb = strlen(sb);
	while(n < lena + lenb) n *= 2;
	for(int i = 0; i < lena; i++)
		a[i].real(sa[lena - 1 - i] - '0');
	for(int i = 0; i < lenb; i++)
		b[i].real(sb[lenb - 1 - i] - '0');
	init();
	fft(a, omg);
	fft(b, omg);
	for(int i = 0; i < n; i++)
		a[i] *= b[i];
	fft(a, inv);
	for(int i = 0; i < n; i++){
		res[i] += floor(a[i].real() / n + 0.5);
		res[i + 1] += res[i] / 10;
		res[i] %= 10;
	}
	for(int i = res[lena + lenb - 1] ? lena + lenb - 1: lena + lenb - 2; i >= 0; i--)
```

```
        putchar('0' + res[i]);
    enter;
    return 0;
}
```

22、五边形定理，整数划分

```
const int MAXN = 1e5 + 10;
const int MOD = 1e9 + 7;
int n, ans[MAXN];
int main(){
    scanf("%d", &n);//读入 n 为求把 n 划分成至多 n 个数 // 任意划分
    ans[0] = 1;
    for (int i = 1; i <= n; ++i){
        for (int j = 1; f(j) <= i; ++j){
            if (j & 1)    ans[i] = (ans[i] + ans[i - f(j)]) % MOD;
            else      ans[i] = (ans[i] - ans[i - f(j)] + MOD) % MOD;
        }
        for (int j = 1; g(j) <= i; ++j){
            if (j & 1)        ans[i] = (ans[i] + ans[i - g(j)]) % MOD;
            else        ans[i] = (ans[i] - ans[i - g(j)] + MOD) % MOD;
        }
    }
    printf("%d\n", ans[n]);
    return 0;}
```

23、整数划分,不能有 k 重复

```
//  问一个数 n 能被拆分成多少种情况
//  且要求拆分元素重复次数不能≥k
const int MOD = 1e9 + 7;
const int MAXN = 1e5 + 10;
int ans[MAXN];
//  此函数求 ans[]效率比上一个代码段中求 ans[]效率高很多
void init(){
    memset(ans, 0, sizeof(ans));
    ans[0] = 1;
    for (int i = 1; i < MAXN; ++i){
        ans[i] = 0;
        for (int j = 1; ; j++){
            int tmp = (3 * j - 1) * j / 2;
            if (tmp > i)      break;
            int tmp_ = ans[i - tmp];
            if (tmp + j <= i)        tmp_ = (tmp_ + ans[i - tmp - j]) % MOD;
            if (j & 1)      ans[i] = (ans[i] + tmp_) % MOD;
            else        ans[i] = (ans[i] - tmp_ + MOD) % MOD;
```

```
		}
	}
	return ;
}
int solve(int n, int k){
	int res = ans[n];
	for (int i = 1; ; i++){
		int tmp = k * i * (3 * i - 1) / 2;
		if (tmp > n) break;
		int tmp_ = ans[n - tmp];
		if (tmp + i * k <= n)	tmp_ = (tmp_ + ans[n - tmp - i * k]) % MOD;
		if (i & 1)	res = (res - tmp_ + MOD) % MOD;
		else	res = (res + tmp_) % MOD;
	}
	return res;
}
int main(int argc, const char * argv[])
{
	init();
	int T, n, k;
	cin >> T;
	while (T--)
	{
		cin >> n >> k;
		cout << solve(n, k) << '\n';
	}
	return 0;
}
```

24、约数之和

```
/*	求 A^B 的约数之和对 MOD 取模
 *	需要素数筛选和合数分解的算法，需要先调用 getPrime();
 *	参考《合数相关》
   1+p+p^2+p^3+...+p^n */
const int MOD = 1000000;
int prime[MAXN + 1];
//	获取素数
void getPrime(){
	memset(prime, 0, sizeof(prime));
	for (int i = 2; i <= MAXN; i++){
		if (!prime[i])	prime[++prime[0]] = i;
		for (int j = 1; j <= prime[0] && prime[j] <= MAXN / i; j++){
			prime[prime[j] * i] = 1;
			if (i % prime[j] == 0)	break;
```

```
        }
    }
    return ;
}
long long factor[100][2];
int fatCnt;
//  合数分解
int getFactors(long long x){
    fatCnt = 0;
    long long tmp = x;
    for (int i = 1; prime[i] <= tmp / prime[i]; i++){
        factor[fatCnt][1] = 0;
        if (tmp % prime[i] == 0){
            factor[fatCnt][0] = prime[i];
            while (tmp % prime[i] == 0){
                factor[fatCnt][1]++;
                tmp /= prime[i];
            }  fatCnt++;}
    }
    if (tmp != 1){
        factor[fatCnt][0] = tmp;
        factor[fatCnt++][1] = 1;
    } return fatCnt;
}
long long pow_m(long long a, long long n){
    long long ret = 1;
    long long tmp = a % MOD;
    while(n){
        if (n & 1)    ret = (ret * tmp) % MOD;
        tmp = tmp * tmp % MOD;
        n >>= 1; }
    return ret;}
//  计算 1+p＋p^2+...+p^n
long long sum(long long p, long long n){
    if (p == 0)  return 0;
    if (n == 0)  return 1;
    if (n & 1)    return ((1 + pow_m(p, n / 2 + 1)) % MOD * sum(p, n / 2) % MOD) % MOD;
    else  return ((1 + pow_m(p, n / 2 + 1)) % MOD * sum(p, n / 2 - 1) + pow_m(p, n / 2) %
MOD) % MOD;
}
//  返回 A^B 的约数之和%MOD
long long solve(long long A, long long B){
    getFactors(A);
    long long ans = 1;
```

```
	for (int i = 0; i < fatCnt; i++){
		ans *= sum(factor[i][0], B * factor[i][1]) % MOD;
		ans %= MOD;
	}
	return ans;}
```

25、莫比乌斯单独求解

```
int MOD(int a, int b)	{return a - a / b * b;}
int miu(int n){
	int cnt, k = 0;
	for (int i = 2; i * i <= n; i++){
		if (MOD(n, i))	continue;
		cnt = 0;
		k++;
		while (MOD(n, i) == 0){
			n /= i;
			cnt++;
		}
		if (cnt >= 2)		return 0;
	}
	if (n != 1)	k++;
	return MOD(k, 2) ? -1 : 1;}
int MOD(int a, int b)	{return a - a / b * b;}
int miu(int n){
	int cnt, k = 0;
	for (int i = 2; i * i <= n; i++){
		if (MOD(n, i))	continue;
		cnt = 0;
		k++;
		while (MOD(n, i) == 0){
			n /= i;
			cnt++;
		}
		if (cnt >= 2)		return 0;
	}
	if (n != 1)	k++;
	return MOD(k, 2) ? -1 : 1;}
int MOD(int a, int b)	{return a - a / b * b;}
```

26、莫比乌斯线性筛法

```
/*	莫比乌斯反演公式
	线性筛法求解积性函数（莫比乌斯函数）
	如果一个数包含平方因子，那么 miu(n) = 0。
	例如：miu(4), miu(12), miu(18) = 0。
```

```
    如果一个数不包含平方因子，并且有 k 个不同的质因子，
    那么 miu(n) = (-1)^k。例如：miu(2), miu(3), miu(30)
    = -1,miu(1), miu(6), miu(10) = 1。*/
const int MAXN = 1000000;
bool check[MAXN + 10];
int prime[MAXN + 10];
int mu[MAXN + 10];
void Moblus(){
    memset(check, false, sizeof(check));
    mu[1] = 1;
    int tot = 0;
    for (int i = 2; i <= MAXN; i++){
        if (!check[i]){
            prime[tot++] = i;
            mu[i] = -1;
        }
        for (int j = 0; j < tot; j++){
            if (i * prime[j] > MAXN)        break;
            check[i * prime[j]] = true;
            if (i % prime[j] == 0){
                mu[i * prime[j]] = 0;
                break;
            }
            else    mu[i * prime[j]] = -mu[i];
        }
    }
}
int main(){
    Moblus(); // mu[]存储对应位置的莫比乌斯函数值
    cout << mu[5] << endl;  }
```

27、莫比乌斯求解区间 gcd(x,y)等于 k 的对数

```
int IT_MAX = 1 << 19;
int MOD = 1000000007;
const int INF = 0x3f3f3f3f;
const ll LL_INF = 0x3f3f3f3f3f3f3f3f;
const db PI = acos(-1);
const db ERR = 1e-10;
const int MAX_N = 100005;
bool cmp (int a , int b){
    return a>b;
}
```

28、BGGS,a^x=b%n

```
/*    baby_step giant _step
    a^x = b(mod n) n 不要求是素数
      求解上式 0 ≤ x < n 的解*/
#define MOD 76543
int hs[MOD];
int head[MOD];
int _next[MOD];
int id[MOD];
int top;
void insert(int x, int y){
        int k = x % MOD;
        hs[top] = x;
        id[top] = y;
        _next[top] = head[k];
        head[k] = top++;
        return ;
}
int find(int x){
        int k = x % MOD;
        for (int i = head[k]; i != -1; i = _next[i]){
                if (hs[i] == x)      return id[i];
        }
        return -1;
}
long long BSGS(int a, int b, int n){
        memset(head, -1, sizeof(head));
        top = 1;
        if (b == 1)        return 0;
        int m = (int)sqrt(n * 1.0), j;
        long long x = 1, p = 1;
        for (int i = 0; i < m; i++, p = p * a % n)      insert(p * b % n, i);
        for (long long i = m; ; i++){
                if ((j = find(x = x * p % n)) != -1)        return i - j;
                if (i > n)        break;
        }
        return -1;
}
```

29、牛顿法多项式求根

```
/* 牛顿法解多项式的根
  *    输入:多项式系数 c[],多项式度数 n,求在[a,b]间的根
  *    输出:根  要求保证[a,b]间有根*/
double fabs(double x){    return (x < 0) ? -x : x;}
```

```
double f(int m, double c[], double x){
    int i;
    double p = c[m];
    for (i = m; i > 0; i--)     p = p * x + c[i - 1];
    return p;
}
int newton(double x0, double *r, double c[], double cp[], int n, double a, double b, double eps){
    int MAX_ITERATION = 1000;
    int i = 1;
    double x1, x2, fp, eps2 = eps / 10.0;
    x1 = x0;
    while (i < MAX_ITERATION){
        x2 = f(n, c, x1);
        fp = f(n - 1, cp, x1);
        if ((fabs(fp) < 0.000000001) && (fabs(x2) > 1.0))    return 0;
        x2 = x1 - x2 / fp;
        if (fabs(x1 - x2) < eps2){
            if (x2 < a || x2 > b)    return 0;
            *r = x2;
            return 1;
        }
        x1 = x2;
        i++;
    }
    return 0;
}
double Polynomial_Root(double c[], int n, double a, double b, double eps){
    double *cp;
    int i;
    double root;
    cp = (double *)calloc(n, sizeof(double));
    for (i = n - 1; i >= 0; i--)     cp[i] = (i + 1) * c[i + 1];
    if (a > b){
        root = a;
        a = b;
        b = root;
    }
    if ((!newton(a, &root, c, cp, n, a, b, eps)) && (!newton(b, &root, c, cp, n, a, b, eps))){
    newton((a + b) * 0.5, &root, c, cp, n, a, b, eps);
    }
    free(cp);
    if (fabs(root) < eps)     return fabs(root);
    else     return root;
}
```

```
int main(){
    Dobule c[2]={2,-1};
    int n = 1, a = -5, b = 5;
    double eps = 1e-8;
    double root = Polynomial_Root(c, n, a, b, eps);
    cout << root << endl;   }
```

30、自适应积分

```
const double eps = 1e-6; // 积分精度
    // 被积函数
double F(double x){
    double ans;
    // 被积函数
    // ans = x * exp(x); // 椭圆为例
    return ans;
}
// 三点 simpson 法，这里要求 F 是一个全局函数
double simpson(double a, double b){
    double c = a + (b - a) / 2;
    return (F(a) + 4 * F(c) + F(b)) * (b - a) / 6;
}
// 自适应 simpson 公式（递归过程），已知整个区间[a, b]上的三点 simpson 指 A
double asr(double a, double b, double eps, double A){
    double c = a + (b - a) / 2;
    double L = simpson(a, c), R = simpson(c, b);
    if (fabs(L + R - A) <= 15 * eps){
        return L + R + (L + R - A) / 15.0;
    }
    return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
}
// 自适应 simpson 公式（主过程）
double asr(double a, double b, double eps){ return asr(a, b, eps, simpson(a, b));   }
int main(int argc, const char * argv[]){
    // std::cout << asr(1, 2, eps) << '\n';
      return 0; }
```

31、容斥 dfs

```
const int MAXN = 1111;
int n;
double ans;
double p[MAXN];
void dfs(int x, int tot, double sum){       //   dfs(1, 0, ?)
    if (x == n + 1){
        if (sum == 0.0)    return ;
```

```
        if (tot & 1)          ans += 1 / sum; //    可替换为任何公式
        else                       ans -= 1 / sum;
        return ;
    }
    dfs(x + 1, tot, sum);
    dfs(x + 1, tot + 1, sum + p[x]);
}
void init(){
    for (int i=1; i<=n; i++) cin>>p[i];
}
int main(){
    while(cin >> n){
        ans = 0;
        init();
        dfs(1,0,0.0);
        printf("%.4f\n",ans);
    }
    return 0;
}
```

32、斐波那契单独求解

```
/*求斐波那契数列第 N 项，模 MOD */
#define mod(a, m) ((a) % (m) + (m)) % (m)
const int MOD = 1e9 + 9;
struct MATRIX{
    long long a[2][2];};
MATRIX a;
long long f[2];
void ANS_Cf(MATRIX a)
{
    f[0] = mod(a.a[0][0] + a.a[1][0], MOD);
    f[1] = mod(a.a[0][1] + a.a[1][1], MOD);
    return ;}
MATRIX MATRIX_Cf(MATRIX a, MATRIX b){
    MATRIX ans;
    int k;
    for (int i = 0; i < 2; i++){
        for (int j = 0; j < 2; j++){
            ans.a[i][j] = 0;
            k = 0;
            while (k < 2){
                ans.a[i][j] += a.a[k][i] * b.a[j][k];
                ans.a[i][j] = mod(ans.a[i][j], MOD);
                ++k;
```

```
            }
        }
    }
    return ans;
}
MATRIX MATRIX_Pow(MATRIX a, long long n){
    MATRIX ans;
    ans.a[0][0] = 1;
    ans.a[1][1] = 1;
    ans.a[0][1] = 0;
    ans.a[1][0] = 0;
    while (n){
        if (n & 1){
            ans = MATRIX_Cf(ans, a);
        }
        n = n >> 1;
        a = MATRIX_Cf(a, a);
    }
    return ans;
}
int main(){
    long long n;
    while (cin >> n){
        if (n == 1){
            cout << '1' << '\n';
            continue;}
        a.a[0][0] = a.a[0][1] = a.a[1][0] = 1;
        a.a[1][1] = 0;
        a = MATRIX_Pow(a, n - 2);
        ANS_Cf(a);
        cout << f[0] << '\n';
    }
    return 0;
}
```

33、求循环节长度

```
/*求 1/i 的循环节长度的最大值，i<=n*/
const int MAXN = 1005;
int res[MAXN];    //    循环节长度
int main(){

    int i, temp, j, n;
    for (temp = 1; temp <= 1000; temp++){
        i = temp;
```

```
        while (i % 2 == 0)      i /= 2;
        while (i % 5 == 0)      i /= 5;
        n = 1;
        for (j = 1; j <= i; j++){
            n *= 10;
            n %= i;
            if (n == 1){
                res[temp] = j;
                break;    }
        }
    }
    int max_re;
    while (cin >> n){
        max_re = 1;
        for (i = 1; i <= n; i++){
            if (res[i] > res[max_re])      max_re = i;
        }
        cout << max_re << endl;
    }
    return 0;}
```

34、矩阵

1）nn 矩阵的 x 次幂快速幂

```
/*矩阵快速幂 n*n 矩阵的 x 次幂*/
#define MAXN 111
#define mod(x) ((x) % MOD)
#define MOD 1000000007
#define LL long long
int n;
struct mat{
    int m[MAXN][MAXN];
} unit; //  单元矩阵
//  矩阵乘法
mat operator * (mat a, mat &b){
    mat ret;
    memset(ret.m, 0, sizeof(ret.m));
    for (int k = 0; k < n; k++){
        for (int i = 0; i < n; i++){
            if (a.m[i][k])
            for (int j = 0; j < n; j++)      ret.m[i][j] = mod(ret.m[i][j] + (LL)a.m[i][k] * b.m[k][j]);
        }
    }
    return ret;
}
```

```
void init_unit(){
	for (int i = 0; i < MAXN; i++){
		unit.m[i][i] = 1;
	}
	return ;}
mat pow_mat(mat a, LL n){
	mat ret = unit;
	while (n){
		if (n & 1){
//				n--;
			ret = ret * a;
		}
		n >>= 1;
		a = a * a;
	}
	return ret;
}
int main()
{
	LL x;
	init_unit();
	while (cin >> n >> x){
		mat a;
		for (int i = 0; i < n; i++) {
			for (int j = 0; j < n; j++)	cin >> a.m[i][j];
		}
		a = pow_mat(a, x);	//	a 矩阵的 x 次幂
		//	输出矩阵
		for (int i = 0; i < n; i++)
			for (int j = 0; j < n; j++)
				if (j + 1 == n)		cout << a.m[i][j] << endl;
				else	cout << a.m[i][j] << " ";
	}
	return 0;
}
```

2）矩阵乘法

```
int n;
struct mat{
	int m[MAXN][MAXN];};
mat operator * (mat a, mat &b)
{
	mat ret;
	memset(ret.m, 0, sizeof(ret.m));
```

```
	for (int k = 0; k < n; k++){
		for (int i = 0; i < n; i++){
			if (a.m[i][k])
			for (int j = 0; j < n; j++)	ret.m[i][j] = mod(ret.m[i][j] + (LL)a.m[i][k] * b.m[k][j]);
		}
	}
	return ret;	}
```

3）矩阵乘法结构体实现加判等

```
/*AB == C ??? */
struct Matrix{
	Type mat[MAXN][MAXN];
	int n, m;
	Matrix(){
		n = m = MAXN;
		memset(mat, 0, sizeof(mat));
	}
	Matrix(const Matrix &a){
		set_size(a.n, a.m);
		memcpy(mat, a.mat, sizeof(a.mat));
	}
	Matrix & operator = (const Matrix &a){
		set_size(a.n, a.m);
		memcpy(mat, a.mat, sizeof(a.mat));
		return *this;
	}
	void set_size(int row, int column){
		n = row;
		m = column;
	}
	friend Matrix operator * (const Matrix &a, const Matrix &b){
		Matrix ret;
		ret.set_size(a.n, b.m);
		for (int i = 0; i < a.n; ++i)
			for (int k = 0; k < a.m; ++k)
				if (a.mat[i][k])
					for (int j = 0; j < b.m; ++j)
						if (b.mat[k][j])
							ret.mat[i][j] = ret.mat[i][j] + a.mat[i][k] * b.mat[k][j];
		return ret;
	}
	friend bool operator == (const Matrix &a, const Matrix &b){
		if (a.n != b.n || a.m != b.m)	return false;
		for (int i = 0; i < a.n; ++i)
```

```
            for (int j = 0; j < a.m; ++j)
                if (a.mat[i][j] != b.mat[i][j])    return false;
        return true;
    }
};
```

35、求 N 以内因子最多的数

```
const int MAXP = 16;
const int prime[MAXP] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53};
ll n, res, ans;
void dfs(ll cur, ll num, int key, ll pre) { //    当前值/当前约数数量/当前深度/上一个数
    if (key >= MAXP)      return ;
    else{
        if (num > ans){
            res = cur;
            ans = num;
        }
        else if (num == ans){       //    如果约数数量相同，则取较小的数
            res = min(cur, res);
        }
        ll i;
        for ( i = 1; i <= pre; i++){
            if (cur <= n / prime[key]) { //    cur*prime[key]<=n
                cur *= prime[key];
                dfs(cur, num * (i + 1), key + 1, i);
            }
            else       break;
        }
    }}
void solve(){
    res = 1;
    ans = 1;
    dfs(1, 1, 0, 15);
    cout << res << ' ' << ans << endl;}
int main(int argc, const char * argv[]){
    int T;
    cin >> T;
    while (T--){
        cin >> n;
        solve();
    }
    return 0;    }
```