数论

常见数列

调和级数

满足调和级数 $\mathcal{O}\left(\frac{N}{1}+\frac{N}{2}+\frac{N}{3}+\cdots+\frac{N}{N}\right)$，可以用 $\approx N\ln N$ 来拟合，但是会略小，误差量级在 10% 左右。本地可以在500ms内完成 10^8 量级的预处理计算。

N的量级	1	2	3	4	5	6	7	8	9
累加和	27	482	7'069	93'668	1'166'750	13'970'034	162'725'364	1'857'511'568	20'877'697'634

下方示例为求解 1 到 N 中各个数字的因数值。

```
const int N = 1E5;
vector<vector<int>> dic(N + 1);
for (int i = 1; i <= N; i++) {
    for (int j = i; j <= N; j += i) {
        dic[j].push_back(i);
    }
}
```

素数密度与分布

N的量级	1	2	3	4	5	6	7	8	9
素数数量	4	25	168	1'229	9'592	78'498	664'579	5'761'455	50'847'534

除此之外，对于任意两个相邻的素数 $p_1,p_2\le 10^9$ ，有 $|p_1-p_2|<300$ 成立，更具体的说，最大的差值为 282 。

因数最多数字与其因数数量

N的量级	1	2	3	4	5	6	7
因数最多数字的因数数量	4	25	32	64	128	240	448
因数最多的数字	-	-	-	7560, 9240	83160, 98280	720720, 831600, 942480, 982800, 997920	-

欧拉筛（线性筛）

时间复杂度为 $\mathcal{O}(N\log\log N)$ 。

```
vector<int> prime; // 这里储存筛出来的全部质数
auto euler_Prime = [&](int n) -> void {
    vector<int> v(n + 1);
    for (int i = 2; i <= n; ++i) {
```

```cpp
        if (!v[i]) {
            v[i] = i;
            prime.push_back(i);
        }
        for (int j = 0; j < prime.size(); ++j) {
            if (prime[j] > v[i] || prime[j] > n / i) break;
            v[i * prime[j]] = prime[j];
        }
    }
};
```

最小质因数

```cpp
std::vector<int> minp, primes;

void sieve(int n) {
    minp.assign(n + 1, 0);
    primes.clear();

    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            minp[i] = i;
            primes.push_back(i);
        }

        for (auto p : primes) {
            if (i * p > n) {
                break;
            }
            minp[i * p] = p;
            if (p == minp[i]) {
                break;
            }
        }
    }
}
```

防爆模乘

借助浮点数实现

以 $\mathcal{O}(1)$ 计算 $a \cdot b \bmod p$，由于不取模，常数比 int128 法小很多。其中 $1 \le n, k, p \le 10^{18}$ 。

```cpp
int mul(int a, int b, int m) {
    int r = a * b - m * (int)(1.L / m * a * b);
    return r - m * (r >= m) + m * (r < 0);
}
```

借助 int128 实现

```
int mul(int a, int b, int m) {
    return (__int128)a * b % m;
}
```

威尔逊定理

1. 当且仅当p为素数时，$(p-1)! \equiv -1 \pmod p$
2. 当且仅当p为素数时，$(p-1)! \equiv p-1 \pmod p$
3. 若p为质数，则p能被$(p-1)!+1$整除
4. 当且仅当p为素数时，$p \mid (p-1)!+1$

裴蜀定理

> $ax+by=c\ (x \in Z^*, y \in Z^*)$ 成立的充要条件是 $gcd(a,b) \mid c$ （Z^* 表示正整数集)。

逆定理

设 a,b 是不全为零的整数，若 $d>0$ 是 a,b 的公因数，且存在整数 x,y, 使得 $ax+by=d$，则 $d=\gcd(a,b)$。

特殊地，设 a,b 是不全为零的整数，若存在整数 x,y, 使得 $ax+by=1$，则 a,b 互质。

多个整数

裴蜀定理可以推广到 n 个整数的情形：设 $a_1,a_2,\ldots,a_n$ 是不全为零的整数，则存在整数 $x_1,x_2,\ldots,x_n$, 使得 $a_1x_1+a_2x_2+\cdots+a_nx_n=\gcd(a_1,a_2,\ldots,a_n)$。其逆定理也成立：设 $a_1,a_2,\ldots,a_n$ 是不全为零的整数，$d>0$ 是 $a_1,a_2,\ldots,a_n$ 的公因数，若存在整数 $x_1,x_2,\ldots,x_n$, 使得 $a_1x_1+a_2x_2+\cdots+a_nx_n=d$，则 $d=\gcd(a_1,a_2,\ldots,a_n)$。

例题：给定一个序列 a，找到一个序列 x，使得 $\sum_{i=1}^n a_ix_i$ 最小。

```
LL n, a, ans;
LL gcd(LL a, LL b){
    return b ? gcd(b, a % b) : a;
}
int main(){
    cin >> n;
    for (int i = 0; i < n; i ++ ){
        cin >> a;
        if (a < 0) a = -a;
        ans = gcd(ans, a);
    }
    cout << ans << "\n";
    return 0;
}
```

逆元

费马小定理解（借助快速幂）

若 p 为素数，$\gcd(a,p)=1$，则 $a^{p-1}\equiv 1 \pmod p$。

另一个形式：对于任意整数 a，有 $a^p\equiv a \pmod p$。

单次计算的复杂度即为快速幂的复杂度 $\mathcal{O}(\log X)$ 。限制：MOD 必须是质数，且需要满足 x 与 MOD 互质。

```
LL inv(LL x) { return mypow(x, mod - 2, mod);}
```

扩展欧几里得解

此方法的 MOD 没有限制，复杂度为 $\mathcal{O}(\log X)$ ，但是比快速幂法常数大一些。

```
int x, y;
int exgcd(int a, int b, int &x, int &y) { //扩展欧几里得算法
    if (b == 0) {
        x = 1, y = 0;
        return a; //到达递归边界开始向上一层返回
    }
    int r = exgcd(b, a % b, x, y);
    int temp = y; //把x y变成上一层的
    y = x - (a / b) * y;
    x = temp;
    return r; //得到a b的最大公因数
}
LL getInv(int a, int mod) { //求a在mod下的逆元，不存在逆元返回-1
    LL x, y, d = exgcd(a, mod, x, y);
    return d == 1 ? (x % mod + mod) % mod : -1;
}
```

离线求解：线性递推解

以 $\mathcal{O}(N)$ 的复杂度完成 $1-N$ 中全部逆元的计算。

```
inv[1] = 1;
for (int i = 2; i <= n; i ++ )
    inv[i] = (p - p / i) * inv[p % i] % p;
```

扩展欧几里得 exgcd

求解形如 $a\cdot x+b\cdot y=\gcd(a,b)$ 的不定方程的任意一组解。

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

例题：求解二元一次不定方程 $A \cdot x + B \cdot y = C$ 。

```cpp
auto clac = [&](int a, int b, int c) {
    int u = 1, v = 1;
    if (a < 0) { // 负数特判，但是没用经过例题测试
        a = -a;
        u = -1;
    }
    if (b < 0) {
        b = -b;
        v = -1;
    }

    int x, y, d = exgcd(a, b, x, y), ans;
    if (c % d != 0) { // 无整数解
        cout << -1 << "\n";
        return;
    }
    a /= d, b /= d, c /= d;
    x *= c, y *= c; // 得到可行解

    ans = (x % b + b - 1) % b + 1;
    auto [A, B] = pair{u * ans, v * (c - ans * a) / b}; // x最小正整数 特解

    ans = (y % a + a - 1) % a + 1;
    auto [C, D] = pair{u * (c - ans * b) / a, v * ans}; // y最小正整数 特解

    int num = (C - A) / b + 1; // xy均为正整数 的 解的组数
};
```

离散对数 bsgs 与 exbsgs

以 $\mathcal{O}(\sqrt{P})$ 的复杂度求解 $a^x \equiv b(\mod P)$ 。其中标准 BSGS 算法不能计算 a 与 MOD 互质的情况，而 exbsgs 则可以。

```cpp
namespace BSGS {
LL a, b, p;
map<LL, LL> f;
inline LL gcd(LL a, LL b) { return b > 0 ? gcd(b, a % b) : a; }
inline LL ps(LL n, LL k, int p) {
    LL r = 1;
```

```
    for (; k; k >>= 1) {
        if (k & 1) r = r * n % p;
        n = n * n % p;
    }
    return r;
}
void exgcd(LL a, LL b, LL &x, LL &y) {
    if (!b)
        x = 1, y = 0;
    } else {
        exgcd(b, a % b, x, y);
        LL t = x;
        x = y;
        y = t - a / b * y;
    }
}
LL inv(LL a, LL b) {
    LL x, y;
    exgcd(a, b, x, y);
    return (x % b + b) % b;
}
LL bsgs(LL a, LL b, LL p) {
    f.clear();
    int m = ceil(sqrt(p));
    b %= p;
    for (int i = 1; i <= m; i++) {
        b = b * a % p;
        f[b] = i;
    }
    LL tmp = ps(a, m, p);
    b = 1;
    for (int i = 1; i <= m; i++) {
        b = b * tmp % p;
        if (f.count(b) > 0) return (i * m - f[b] + p) % p;
    }
    return -1;
}
LL exbsgs(LL a, LL b, LL p) {
    if (b == 1 || p == 1) return 0;
    LL g = gcd(a, p), k = 0, na = 1;
    while (g > 1) {
        if (b % g != 0) return -1;
        k++;
        b /= g;
        p /= g;
        na = na * (a / g) % p;
        if (na == b) return k;
        g = gcd(a, p);
    }
    LL f = bsgs(a, b * inv(na, p) % p, p);
    if (f == -1) return -1;
    return f + k;
}
```

```cpp
} // namespace BSGS

using namespace BSGS;

int main() {
    IOS;
    cin >> p >> a >> b;
    a %= p, b %= p;
    LL ans = exbsgs(a, b, p);
    if (ans == -1) cout << "no solution\n";
    else cout << ans << "\n";
    return 0;
}
```

欧拉函数

直接求解单个数的欧拉函数

1 到 N 中与 N 互质数的个数称为欧拉函数，记作 $\varphi(N)$。求解欧拉函数的过程即为分解质因数的过程，复杂度 $\mathcal{O}(\sqrt{n})$。

```cpp
int phi(int n) { //求解 phi(n)
    int ans = n;
    for(int i = 2; i <= n / i; i ++) { //注意，这里要写 n / i ，以防止 int 型溢出风险和 sqrt 超时风险
        if(n % i == 0) {
            ans = ans / i * (i - 1);
            while(n % i == 0) n /= i;
        }
    }
    if(n > 1) ans = ans / n * (n - 1); //特判 n 为质数的情况
    return ans;
}
```

求解 1 到 N 所有数的欧拉函数

利用上述性质，我们可以快速递推出 $2-N$ 中每个数的欧拉函数，复杂度 $\mathcal{O}(N)$ ，而该算法**即是线性筛的算法**。

$$\varphi(n) = (1-1/p_1)(1-1/p_2)(1-1/p_3)(1-1/p_4)\cdots(1-1/p_n);$$

```cpp
const int N = 1e5 + 7;
int v[N], prime[N], phi[N];
void euler(int n) {
    ms(v, 0); //最小质因子
    int m = 0; //质数数量
    for (int i = 2; i <= n; ++ i) {
        if (v[i] == 0) { // i 是质数
            v[i] = i, prime[++ m] = i;
            phi[i] = i - 1;
        }
         //为当前的数 i 乘上一个质因子
        for (int j = 1; j <= m; ++ j) {
```

```cpp
                //如 i 有比 prime[j] 更小的质因子，或超出 n ，停止
                if(prime[j] > v[i] || prime[j] > n / i) break;
                // prime[j] 是合数 i * prime[j] 的最小质因子
                v[i * prime[j]] = prime[j];
                phi[i * prime[j]] = phi[i] * (i % prime[j] ? prime[j] - 1 : prime[j]);
            }
        }
}
int main() {
    int n; cin >> n; euler(n);
    for (int i = 1; i <= n; ++ i) cout << phi[i] << endl;
    return 0;
}
```

使用莫比乌斯反演求解欧拉函数

```cpp
int phi[N];
vector<int> fac[N];
void get_eulers() {
    for (int i = 1; i <= N - 10; i++) {
        for (int j = i; j <= N - 10; j += i) {
            fac[j].push_back(i);
        }
    }
    phi[1] = 1;
    for (int i = 2; i <= N - 10; i++) {
        phi[i] = i;
        for (auto j : fac[i]) {
            if (j == i) continue;
            phi[i] -= phi[j];
        }
    }
}
```

扩展欧拉定理

若正整数 a 与 m 互质，则

$$a^{\varphi(m)} \equiv 1 (\bmod m)$$

推论：

$$a^b \equiv a^{b \bmod \varphi(m)} (\bmod m)$$

当 a, m 不互质时，扩展 Euler 定理表述如下：

$$a^b \equiv a^{b \bmod \varphi(m) + \varphi(m)} (\bmod m)$$

式子仅在 $\varphi(m) \leq b$ 时成立

```cpp
#include <bits/stdc++.h>
using namespace std;
bool large_enough = false; // 判断是否有b >= phi(m)
```

```cpp
inline int read(int MOD = 1e9 + 7) // 快速读入稍加修改即可以边读入边取模，不取模时直接模一个大于数据范围的数
{
    int ans = 0;
    char c = getchar();
    while (!isdigit(c))
        c = getchar();
    while (isdigit(c))
    {
        ans = ans * 10 + c - '0';
        if (ans >= MOD)
        {
            ans %= MOD;
            large_enough = true;
        }
        c = getchar();
    }
    return ans;
}
int phi(int n) // 求欧拉函数
{
    int res = n;
    for (int i = 2; i * i <= n; i++)
    {
        if (n % i == 0)
            res = res / i * (i - 1);
        while (n % i == 0)
            n /= i;
    }
    if (n > 1)
        res = res / n * (n - 1);
    return res;
}
int qpow(int a, int n, int MOD) // 快速幂
{
    int ans = 1;
    while (n)
    {
        if (n & 1)
            ans = 1LL * ans * a % MOD; // 注意防止溢出
        n >>= 1;
        a = 1LL * a * a % MOD;
    }
    return ans;
}
int main()
{
    int a = read(), m = read(), phiM = phi(m), b = read(phiM);
    cout << qpow(a, b + (large_enough ? phiM : 0), m);
    return 0;
}
```

求解连续数字的正约数集合——倍数法

使用规律递推优化，时间复杂度为 $\mathcal{O}(N\log N)$ ，如果不需要详细的输出集合，则直接将 `vector` 换为普通数组即可（时间更快） 。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 7;
vector<int> f[N];

void divide(int n) {
    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 (auto it : f[i]) cout << it << " ";
        cout << endl;
    }
}
int main() {
    int x; cin >> x; divide(x);
    return 0;
}
```

试除法判是否是质数

标准解

$\mathcal{O}(\sqrt{N})$ 。

```
bool is_prime(int n) {
    if (n < 2) return false;
    for (int i = 2; i <= x / i; i++) {
        if (n % i == 0) return false;
    }
    return true;
}
```

常数优化法

常数优化，达到 $\mathcal{O}(\frac{\sqrt{N}}{3})$ 。

```cpp
bool is_prime(int n) {
    if (n < 2) return false;
    if (n == 2 || n == 3) return true;
    if (n % 6 != 1 && n % 6 != 5) return false;
    for (int i = 5, j = n / i; i <= j; i += 6) {
        if (n % i == 0 || n % (i + 2) == 0) {
            return false;
        }
    }
    return true;
}
```

同余方程组、拓展中国剩余定理 excrt

公式：$x \equiv b_i(\bmod\ a_i)$，即 $(x-b_i) \mid a_i$。

```cpp
int n; LL ai[maxn], bi[maxn];
inline int mypow(int n, int k, int p) {
    int r = 1;
    for (; k; k >>= 1, n = n * n % p)
        if (k & 1) r = r * n % p;
    return r;
}
LL exgcd(LL a, LL b, LL &x, LL &y) {
    if (b == 0) { x = 1, y = 0; return a; }
    LL gcd = exgcd(b, a % b, x, y), tp = x;
    x = y, y = tp - a / b * y;
    return gcd;
}
LL excrt() {
    LL x, y, k;
    LL M = bi[1], ans = ai[1];
    for (int i = 2; i <= n; ++ i) {
        LL a = M, b = bi[i], c = (ai[i] - ans % b + b) % b;
        LL gcd = exgcd(a, b, x, y), bg = b / gcd;
        if (c % gcd != 0) return -1;
        x = mul(x, c / gcd, bg);
        ans += x * M;
        M *= bg;
        ans = (ans % M + M) % M;
    }
    return (ans % M + M) % M;
}
int main() {
    cin >> n;
    for (int i = 1; i <= n; ++ i) cin >> bi[i] >> ai[i];
    cout << excrt() << endl;
    return 0;
}
```

求解连续按位异或

以 $\mathcal{O}(1)$ 复杂度计算 $0 \oplus 1 \oplus \cdots \oplus n$ 。

```
unsigned xor_n(unsigned n) {
    unsigned t = n & 3;
    if (t & 1) return t / 2u ^ 1;
    return t / 2u ^ n;
}
```

```
i64 xor_n(i64 n) {
    if (n % 4 == 1) return 1;
    else if (n % 4 == 2) return n + 1;
    else if (n % 4 == 3) return 0;
    else return n;
}
```

高斯消元求解线性方程组

题目大意：输入一个包含 N 个方程 N 个未知数的线性方程组，系数与常数均为实数（两位小数）。求解这个方程组。如果存在唯一解，则输出所有 N 个未知数的解，结果保留两位小数。如果无数解，则输出 `X` ，如果无解，则输出 `N` 。

```
const int N = 110;
const double eps = 1e-8;
LL n;
double a[N][N];
LL gauss(){
    LL c, r;
    for (c = 0, r = 0; c < n; c ++ ){
        LL t = r;
        for (int i = r; i < n; i ++ )     //找到绝对值最大的行
            if (fabs(a[i][c]) > fabs(a[t][c]))
                t = i;
        if (fabs(a[t][c]) < eps) continue;
        for (int j = c; j < n + 1; j ++ ) swap(a[t][j], a[r][j]);     //将绝对值最大的一行换到最顶端
        for (int j = n; j >= c; j -- ) a[r][j] /= a[r][c];     //将当前行首位变成 1
        for (int i = r + 1; i < n; i ++ )     //将下面列消成 0
            if (fabs(a[i][c]) > eps)
                for (int j = n; j >= c; j -- )
                    a[i][j] -= a[r][j] * a[i][c];
        r ++ ;
    }
    if (r < n){
        for (int i = r; i < n; i ++ )
            if (fabs(a[i][n]) > eps)
                return 2;
        return 1;
    }
    for (int i = n - 1; i >= 0; i -- )
```

```
        for (int j = i + 1; j < n; j ++ )
            a[i][n] -= a[i][j] * a[j][n];
    return 0;
}
int main(){
    cin >> n;
    for (int i = 0; i < n; i ++ )
        for (int j = 0; j < n + 1; j ++ )
            cin >> a[i][j];
    LL t = gauss();
    if (t == 0){
        for (int i = 0; i < n; i ++ ){
            if (fabs(a[i][n]) < eps) a[i][n] = abs(a[i][n]);
            printf("%.2lf\n", a[i][n]);
        }
    }
    else if (t == 1) cout << "Infinite group solutions\n";
    else cout << "No solution\n";
    return 0;
}

```

Min25 筛

求解 $1-N$ 的质数和，其中 $N \leq 10^{10}$ 。

```
namespace min25{
    const int N = 1000000 + 10;
    int prime[N], id1[N], id2[N], flag[N], ncnt, m;
    LL g[N], sum[N], a[N], T;
    LL n;
    LL mod;
    inline LL ps(LL n,LL k) {LL r=1;for(;k;k>>=1){if(k&1)r=r*n%mod;n=n*n%mod;}return r;}
    void finit(){ // 最开始清0
        memset(g, 0, sizeof(g));
        memset(a, 0, sizeof(a));
        memset(sum, 0, sizeof(sum));
        memset(prime, 0, sizeof(prime));
        memset(id1, 0, sizeof(id1));
        memset(id2, 0, sizeof(id2));
        memset(flag, 0, sizeof(flag));
        ncnt = m = 0;
    }
    int ID(LL x) {
        return x <= T ? id1[x] : id2[n / x];
    }

    LL calc(LL x) {
        return x * (x + 1) / 2 - 1;
    }

```

```
    LL init(LL x) {
        T = sqrt(x + 0.5);
        for (int i = 2; i <= T; i++) {
            if (!flag[i]) prime[++ncnt] = i, sum[ncnt] = sum[ncnt - 1] + i;
            for (int j = 1; j <= ncnt && i * prime[j] <= T; j++) {
                flag[i * prime[j]] = 1;
                if (i % prime[j] == 0) break;
            }
        }
        for (LL l = 1; l <= x; l = x / (x / l) + 1) {
            a[++m] = x / l;
            if (a[m] <= T) id1[a[m]] = m; else id2[x / a[m]] = m;
            g[m] = calc(a[m]);
        }
        for (int i = 1; i <= ncnt; i++)
            for (int j = 1; j <= m && (LL) prime[i] * prime[i] <= a[j]; j++)
                g[j] = g[j] - (LL) prime[i] * (g[ID(a[j] / prime[i])] - sum[i - 1]);
    }
    LL solve(LL x) {
        if (x <= 1) return x;
        return n = x, init(n), g[ID(n)];
    }
}

using namespace min25;

int main() {
    // while (1) {
    int tt;
    scanf("%d",&tt);
    while(tt--){
        finit();
        scanf("%lld%lld", &n, &mod);
        LL ans = (n + 3) % mod * n % mod  * ps(2 , mod - 2) % mod + solve(n + 1) - 4;
        // cout << solve(n) << endl;
        // ans = (ans + mod) % mod;
        ans = (ans + mod) % mod;
        printf("%lld\n", ans);
    }

    // }
}
```

矩阵四则运算

封装来自。矩阵乘法复杂度 $\mathcal{O}(N^3)$。

```
const int SIZE = 2;
struct Matrix {
    ll M[SIZE + 5][SIZE + 5];
    void clear() { memset(M, 0, sizeof(M)); }
    void reset() { //初始化
```

```
        clear();
        for (int i = 1; i <= SIZE; ++i) M[i][i] = 1;
    }
    Matrix friend operator*(const Matrix &A, const Matrix &B) {
        Matrix Ans;
        Ans.clear();
        for (int i = 1; i <= SIZE; ++i)
            for (int j = 1; j <= SIZE; ++j)
                for (int k = 1; k <= SIZE; ++k)
                    Ans.M[i][j] = (Ans.M[i][j] + A.M[i][k] * B.M[k][j]) % mod;
        return Ans;
    }
    Matrix friend operator+(const Matrix &A, const Matrix &B) {
        Matrix Ans;
        Ans.clear();
        for (int i = 1; i <= SIZE; ++i)
            for (int j = 1; j <= SIZE; ++j)
                Ans.M[i][j] = (A.M[i][j] + B.M[i][j]) % mod;
        return Ans;
    }
};

inline int mypow(LL n, LL k, int p = MOD) {
    LL r = 1;
    for (; k; k >>= 1, n = n * n % p) {
        if (k & 1) r = r * n % p;
    }
    return r;
}
bool ok = 1;
Matrix getinv(Matrix a) { //矩阵求逆
    int n = SIZE, m = SIZE * 2;
    for (int i = 1; i <= n; i++) a.M[i][i + n] = 1;
    for (int i = 1; i <= n; i++) {
        int pos = i;
        for (int j = i + 1; j <= n; j++)
            if (abs(a.M[j][i]) > abs(a.M[pos][i])) pos = j;
        if (i != pos) swap(a.M[i], a.M[pos]);
        if (!a.M[i][i]) {
            puts("No Solution");
            ok = 0;
        }
        ll inv = q_pow(a.M[i][i], mod - 2);
        for (int j = 1; j <= n; j++)
            if (j != i) {
                ll mul = a.M[j][i] * inv % mod;
                for (int k = i; k <= m; k++)
                    a.M[j][k] = ((a.M[j][k] - a.M[i][k] * mul) % mod + mod) % mod;
            }
        for (int j = 1; j <= m; j++) a.M[i][j] = a.M[i][j] * inv % mod;
    }
    Matrix res;
    res.clear();
```

```
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            res.M[i][j] = a.M[i][n + j];
    return res;
}
```

矩阵快速幂

以 $\mathcal{O}(N^3 \log M)$ 的复杂度计算。

```
const int N = 110, mod = 1e9 + 7;
LL n, k, a[N][N], b[N][N], t[N][N];
void matrixQp(LL y){
    while (y){
        if (y & 1){
            memset(t, 0, sizeof t);
            for (int i = 1; i <= n; i ++ )
                for (int j = 1; j <= n; j ++ )
                    for (int k = 1; k <= n; k ++ )
                        t[i][j] = ( t[i][j] + (a[i][k] * b[k][j]) % mod ) % mod;
            memcpy(b, t, sizeof t);
        }
        y >>= 1;
        memset(t, 0, sizeof t);
        for (int i = 1; i <= n; i ++ )
            for (int j = 1; j <= n; j ++ )
                for (int k = 1; k <= n; k ++ )
                    t[i][j] = ( t[i][j] + (a[i][k] * a[k][j]) % mod ) % mod;
        memcpy(a, t, sizeof t);
    }
}
int main(){
    cin >> n >> k;
    for (int i = 1; i <= n; i ++ )
        for (int j = 1; j <= n; j ++ ){
            cin >> b[i][j];
            a[i][j] = b[i][j];
        }
    matrixQp(k - 1);
    for (int i = 1; i <= n; i ++ )
        for (int j = 1; j <= n; j ++ )
            cout << b[i][j] << " \n"[j == n];
    return 0;
}
```

矩阵加速

```
const int mod = 1e9 + 7;
LL T, n, t[5][5], a[5][5], b[5][5];
void matrixQp(LL y){
    while (y){
```

```
        if (y & 1){
            memset(t, 0, sizeof t);
            for (int i = 1; i <= 3; i ++ )
                for (int j = 1; j <= 1; j ++ )
                    for (int k = 1; k <= 3; k ++ )
                        t[i][j] = ( t[i][j] + (a[i][k] * b[k][j]) % mod ) % mod;
            memcpy(b, t, sizeof t);
        }
        y >>= 1;
        memset(t, 0, sizeof t);
        for (int i = 1; i <= 3; i ++ )
            for (int j = 1; j <= 3; j ++ )
                for (int k = 1; k <= 3; k ++ )
                    t[i][j] = ( t[i][j] + (a[i][k] * a[k][j]) % mod ) % mod;
        memcpy(a, t, sizeof t);
    }
}
void init(){
    b[1][1] = b[2][1] = b[3][1] = 1;
    memset(a, 0, sizeof a);
    a[1][1] = a[2][1] = a[1][3] = a[3][2] = 1;
}
void solve(){
    cin >> n;
    if (n <= 3) cout << "1\n";
    else{
        init();
        matrixQp(n - 3);
        cout << b[1][1] << "\n";
    }
}
int main(){
    cin >> T;
    while ( T -- )
        solve();
    return 0;
}

```

莫比乌斯函数/反演

莫比乌斯函数定义：$\mu(n) = \begin{cases} 1 & n = 1 \\ (-1)^k & n = \prod_{i=1}^{k} p_i \text{ 且 } p_i \text{ 互质} \\ 0 & else \end{cases}$ 。

> 莫比乌斯函数性质：对于任意正整数 n 满足 $\sum\limits_{d|n} \mu(d) = \begin{cases} 1 & n = 1 \\ 0 & n \neq 1 \end{cases}$；$\sum\limits_{d|n} \frac{\mu(d)}{d} = \frac{\varphi(n)}{n}$ 。

莫比乌斯反演定义：定义：$F(n)$ 和 $f(n)$ 是定义在非负整数集合上的两个函数，并且满足 $F(n) = \sum\limits_{d|n} f(d)$ ，可得 $f(n) = \sum\limits_{d|n} \mu(d) F(\left\lfloor \frac{n}{d} \right\rfloor)$ 。

```cpp
const int N = 5e4 + 10;
bool st[N];
int mu[N], prime[N], cnt, sum[N];
void getMu() {
    mu[1] = 1;
    for (int i = 2; i <= N - 10; i++) {
        if (!st[i]) {
            prime[++cnt] = i;
            mu[i] = -1;
        }
        for (int j = 1; j <= cnt && i * prime[j] <= N - 10; j++) {
            st[i * prime[j]] = true;
            if (i % prime[j] == 0) {
                mu[i * prime[j]] = 0;
                break;
            }
            mu[i * prime[j]] = -mu[i];
        }
    }
    for (int i = 1; i <= N - 10; i++) {
        sum[i] = sum[i - 1] + mu[i];
    }
}
void solve() {
    int n, m, k; cin >> n >> m >> k;
    n = n / k, m = m / k;
    if (n < m) swap(n, m);
    LL ans = 0;
    for (int i = 1, j = 0; i <= m; i = j + 1) {
        j = min(n / (n / i), m / (m / i));
        ans += (LL)(sum[j] - sum[i - 1]) * (n / i) * (m / i);
    }
    cout << ans << "\n";
}
int main() {
    getMu();
    int T; cin >> T;
    while (T--) solve();
}
```

整除（数论）分块

$\left\lfloor\frac{n}{l}\right\rfloor=\left\lfloor\frac{n}{l+1}\right\rfloor=\ldots=\left\lfloor\frac{n}{r}\right\rfloor \iff \left\lfloor\frac{n}{l}\right\rfloor\leq\frac{n}{r}<\left\lfloor\frac{n}{l}\right\rfloor+1$，根据不等式左侧，得到 $r\leq\left\lfloor\frac{n}{\lfloor\frac{n}{l}\rfloor}\right\rfloor$。

```cpp
void solve() {
    LL n; cin >> n;
    LL ans = 0;
    for (LL i = 1, j; i <= n; i = j + 1) {
        j = n / (n / i);
        ans += (LL)(j - i + 1) * (n / i);
    }
    cout << ans << "\n";
}
int main() {
    int T; cin >> T;
    while (T--) solve();
}
```

Miller - Rabin 素数测试

以平均 $\mathcal{O}(4 \cdot \log^3 X)$ 的复杂度判定数字 X 是否是素数，这里记录的版本复杂度非常优秀，基本可以看作是 $\mathcal{O}(1)$ 。

```cpp
int mul(int a, int b, int m) {
    int r = a * b - m * (int)(1.L / m * a * b);
    return r - m * (r >= m) + m * (r < 0);
}
int mypow(int a, int b, int m) {
    int res = 1 % m;
    for (; b; b >>= 1, a = mul(a, a, m)) {
        if (b & 1) {
            res = mul(res, a, m);
        }
    }
    return res;
}

int B[] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
bool MR(int n) {
    if (n <= 1) return 0;
    for (int p : B) {
        if (n == p) return 1;
        if (n % p == 0) return 0;
    }
    int m = (n - 1) >> __builtin_ctz(n - 1);
    for (int p : B) {
        int t = m, a = mypow(p, m, n);
        while (t != n - 1 && a != 1 && a != n - 1) {
            a = mul(a, a, n);
            t *= 2;
        }
        if (a != n - 1 && t % 2 == 0) return 0;
    }
    return 1;
}
```

Pollard - Rho 因式分解

以单个因子 $\mathcal{O}(\log X)$ 的复杂度输出数字 X 的全部质因数，由于需要结合素数测试，总复杂度会略高一些。如果遇到超时的情况，可能需要考虑进一步优化，例如检查题目是否强制要求枚举全部质因数等等。此外，还有一个较长的模板可供参考，比这里记录的版本常数小约五倍。

```cpp
int PR(int n) {
    for (int p : B) {
        if (n % p == 0) return p;
    }
    auto f = [&](int x) -> int {
        x = mul(x, x, n) + 1;
        return x >= n ? x - n : x;
    };
    int x = 0, y = 0, tot = 0, p = 1, q, g;
    for (int i = 0; (i & 255) || (g = gcd(p, n)) == 1; i++, x = f(x), y = f(f(y))) {
        if (x == y) {
            x = tot++;
            y = f(x);
        }
        q = mul(p, abs(x - y), n);
        if (q) p = q;
    }
    return g;
}
vector<int> fac(int n) {
    #define pb emplace_back
    if (n == 1) return {};
    if (MR(n)) return {n};
    int d = PR(n);
    auto v1 = fac(d), v2 = fac(n / d);
    auto i1 = v1.begin(), i2 = v2.begin();
    vector<int> ans;
    while (i1 != v1.end() || i2 != v2.end()) {
        if (i1 == v1.end()) {
            ans.pb(*i2++);
        } else if (i2 == v2.end()) {
            ans.pb(*i1++);
        } else {
            if (*i1 < *i2) {
                ans.pb(*i1++);
            } else {
                ans.pb(*i2++);
            }
        }
    }
    return ans;
}
```

/END/