ACM-ICPC 数论模板整理

HUT_Gunpowder

2017 年 8 月 27 日

目录

1 定理

1.1 费马小定理

p 是质数，且 $gcd(a,\ p)=1$, 则有 $a^{p-1}\equiv 1\ (mod\ p)$.

1.2 欧拉定理

$gcd(a,\ n)=1$, 则有 $a^{\phi(n)}\equiv 1\ (mod\ n)$.

1.3 威尔逊定理

当且仅当 p 是质数，$(p-1)!\equiv -1\ (mod\ p)$.

1.4 裴蜀定理

$d=gcd(a,b)$, 方程 $ax+by=m$ 有解当且仅当 $d|m$.

1.5 指数循环节

$A^B\ mod\ C=A^{B\ mod\ \phi(C)+\phi(C)}\ mod\ C\quad B>=\phi(C)$

1.6 其他

$a>1\ n,m>0$, 那么有 $gcd(a^m-1,a^n-1)=a^{gcd(n,m)}-1$
$a>b\ \ gcd(a,b)=1$ 那么有 $gcd(a^m-b^m,a^n-b^n)=a^{gcd(n,m)-b^{gcd(n,m)}}$
设 $G=gcd(C_n^1,C_n^2,...,C_n^{n-1})$ 若 n 有唯一质因子则 G 为其质因子，否则为 1
$gcd(Fib_n,Fib_m)=Fib_{gcd(n,m)}$
若 A B 互质，他们最大不能组成的数为 $AB-A-B$ 个数为 $\frac{(A-1)(B-1)}{2}$
如果 p 是素数，那么 $C_p^1,C_p^2...C_p^{p-1}$ 均能被 p 整除
如果 p 是素数，$(x+y...+w)^p\equiv x^p+y^p...+w^p(mod\ p)$
$\sum_{i=1}Ngcd(i,N)=\sum_{d|N}d\phi(\frac{N}{d})$

2 基础

2.1 最大公约数

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

2.2 扩展欧几里德

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

2.3 快速幂

```
inline ll qm(ll a, ll n, const ll & p) {
    ll ans = 1;
    ll tmp = a;
    while (n != 0) {
        if (n & 1)
            ans = ans * tmp % p;
        n = n >> 1;
        tmp = tmp * tmp % p;
    }
    return ans;
}
```

2.4 快速乘

```
LL PowMod(LL a, LL b, LL base) {
    LL ans = 0;
    a %= base;
    b %= base;
    LL now = b;
    while (a != 0) {
        ans += now * (a % 2);
        ans %= base;
        now *= 2;
        now %= base;
        a = a / 2;
    }
    return ans;
```

```
}
```

2.5 原根

```
vector<ll> c;
inline bool pan_g(ll g, ll p) {
    for (int i = 0; i < c.size(); ++i)
        if (qm(g, c[i], p) == 1)
            return 0;
    return 1;
}
inline ll findg(ll p) {
    c.clear();
    ll tmp = p - 1;
    ll k = 2;
    while (k * k <= tmp) {
        if (tmp % k == 0) {
            c.push_back(k);
            while (tmp % k == 0)
                tmp /= k;
        }
        ++k;
    }
    if (tmp != 1)
        c.push_back(tmp);
    for (int i = 0; i < c.size(); ++i)
        c[i] = (p-1) / c[i];
    ll g = 1;
    while (true) {
        if (pan_g(g, p)) {
            return g;
        }
        ++g;
    }
    return 0;
}
```

2.6 欧拉函数

$[1, n]$ 中与 n 互质的数的个数。

$n = p_1^{a_1} p_2^{a_2} ... p_k^{a_k}$

$\phi(n) = n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2})...(1 - \frac{1}{p_k})$

```
int phi = p, m = p, k = 2;
while (k * k <=m) {
    if (m % k == 0) {
        phi /= k;
        phi *= (k-1);
        while (m % k == 0)
            m /= k;
    }
    ++k;
```

```
}
if (m != 1) {
    phi /= m;
    phi *= (m-1);
}
```

2.7 线性筛

p 为是否为素数 $prime$ 为第几个素数 phi 为欧拉函数 mu 为莫比乌斯函数 $div\ num$ 为因子数 nxt 为最大的质因子的编号 e 为最大的质因子的幂次

```
memset(p, 0, sizeof(p));
phi[1] = 1;
mu[1] = 1;
div_num[1] = 1;
top = 0;
for (i = 2; i <= N; ++i) {
    if (!p[i]) {
        prime[++top] = i;
        mu[i] = -1;
        phi[i] = i-1;
        e[i] = 1;
        div_num[i] = 2;
        nxt[i] = top;
    }
    for (j = 1; j <= top; ++j) {
        if (i * prime[j] > N)
            break;
        p[i * prime[j]] = 1;
        nxt[i * prime[j]] = j;
        if (i % prime[j] == 0) {
            phi[i * prime[j]] = phi[i] * prime[j];
            div_num[i*prime[j]]=div_num[i]/(e[i]+1)*(e[i]+2);
            e[i*prime[j]]=e[i]+1;
            break;
        }
        phi[i * prime[j]] = phi[i] * phi[prime[j]];
        mu[i * prime[j]] = -mu[i];
        div_num[i*prime[j]]=div_num[i]*div_num[prime[j]];
        e[i*prime[j]] = 1;
    }
}
```

2.8 卢卡斯定理

$C(n,\ m)\ \equiv\ C(n/p,\ m/p)\ *\ C(n\%p,\ m\%p)\ \%\ p$

```
LL Lucas(LL n, LL m,LL base) {
    LL ans = 1;
    while (n && m) {
        LL a = n % base;
        LL b = m % base;
        if (a < b)
```

```
            return 0;
        LL ret = Frac[b] * Frac[a - b] % base;
        ret = Frac[a] * Inv(ret, base) % base;
        ans = ans * ret % base;
        n /= base;
        m /= base;
    }
    return ans;
}
```

2.9 递推求逆元

$$Inv[n] \;=\; (p \;-\; p \;/\; n) \;*\; Inv[p \;\%\; n] \;\%\; p$$

$$Inv[1] = 1$$

```
LL Lucas(LL n, LL m,LL base) {
    LL ans = 1;
    while (n && m) {
        LL a = n % base;
        LL b = m % base;
        if (a < b)
            return 0;
        LL ret = Frac[b] * Frac[a - b] % base;
        ret = Frac[a] * Inv(ret, base) % base;
        ans = ans * ret % base;
        n /= base;
        m /= base;
    }
    return ans;
}
```

3 Miller-Rabin 素数测试

如果 p 是素数，且 $0 < x < p$，则方程的解 $x^2 \equiv 1(mod\ p)$ 为 1 或 $p-1$。

```
const int Times = 10;
bool Miller_Rabin(LL n) {
    if(n == 2) return true;
    if(n < 2 || !(n & 1)) return false;
    LL m = n - 1;
    int k = 0;
    while((m & 1) == 0) {
        k++;
        m >>= 1;
    }
    for(int i=0; i<Times; i++) {
        LL a = rand() % (n - 1) + 1;
        LL x = quick_mod(a, m, n);
        LL y = 0;
        for(int j=0; j<k; j++){
            y = multi(x, x, n);
            if(y == 1 && x != 1 && x != n - 1) return false;
            x = y;
        }
        if(y != 1) return false;
    }
    return true;
}
```

4 线性同余方程组（中国剩余定理）

方程组 $X \equiv x_i\ (mod\ m_i)$ 有整数解。并且在模 $M = m_1m_2...m_n$ 下的解是唯一的，解为 $x \equiv (x_1M_1M_1^{-1} + x_2M_2M_2^{-1}..x_nM_nM_n^{-1})(mod\ M)$. 其中 $M_i = M/m_i$ 而 M_i^{-1} 是 M_i 模 m_i 的逆元。

```
#include<cstdio>
#include<cstring>
#include<cstdlib>
#include<algorithm>
using namespace std;

long long a[110], m[110];
int n, i;

void extend_Euclid(long long a, long long b, long long &x, long long &y) {
    if(b == 0) {
        x = 1;
        y = 0;
        return;
    }
    extend_Euclid(b, a % b, x, y);
    long long tmp = x;
    x = y;
    y = tmp - (a / b) * y;
```

```
}

long long CRT(long long n) {
    long long M = 1;
    long long ans = 0;
    for(long long i=1; i<=n; i++)
        M *= m[i];
    for(long long i=1; i<=n; i++) {
        long long x, y;
        long long Mi = M / m[i];
        extend_Euclid(Mi, m[i], x, y);
        ans = (ans + Mi * x * a[i]) % M;
    }
    if(ans < 0) ans += M;
    return ans;
}
int main() {
    scanf("%d", &n);
    for (i = 1; i <= n; ++i) {
        scanf("%lld%lld", &m[i], &a[i]);
    }
    printf("%lld\n", CRT(n));
    return 0;
}
```

5 线性同余方程组（扩展欧几里得合并）

不保证互质的方程组，所以只能采用两两合并的方式。对于方程组 $\begin{cases} X = a_1x + r_1 \\ X = a_2y + r_2 \end{cases}$. 即有 $a_1x + r_1 = a_2y + r_2$, 即有 $a_1x - a_2y = r_2 - r_1$ 用扩展欧几里得可以求出最小正整数解 x, 即最小正整数解 $X = a_1x + r_1$. 我们便可以构造一个新的方程。$X = Ax + R$, 其中 $R = a_1x + r_1$ $A = lcm(a_1, a_2)$. 不断的两两合并便可以求得最终解。

```
int n;
ll ans, a1, a2, a3, r1, r2, r3, x, y, tmp;
int main() {
    while (~scanf("%d", &n)) {
        --n;
        scanf("%lld%lld", &a1, &r1);
        ans = (r1 % a1 + a1) % a1;
        while (n--) {
            scanf("%lld%lld", &a2, &r2);
            if (ans == -1)
                continue;
            tmp = exgcd(a1, a2, x, y);
            if ((r1 - r2) % tmp != 0) {
                ans = -1;
                continue;
            }
            x = x * (r2 - r1) / tmp;
            x = (x % a2 + a2) % a2;
            r1 = r1 + a1 * x;
            a1 = a1 * a2 / tmp;
```

```
            ans = (r1 % a1 + a1) % a1;
        }
        printf("%lld\n", ans);
    }
    return 0;
}
```

6 离散对数（BSGS 算法）

首先判断是否有解，即 a, p 是否互质。不互质即无解。不妨令 $x = im - j$, 其中 $m = \lceil\sqrt{q}\rceil$, 这样问题变为求得一组 i j 使得条件满足。此时原式变为 $a^{im-j} \equiv b\ (Mod\ p)$, 移项化简得 $(a^m)^i \equiv ba^j\ (Mod\ p)$。这个时候我们只需穷举 i, j 使得式子成立即可。先从让 j 从 $[0, m]$ 中穷举，并用 hash 记录下 ba^j 对应的 j 值。相同的 ba^j 记录较大的 j. 接着让 i 从 $[1, m]$ 中穷举，如果 $(a^m)^i$ 在 hash 表中有对应的 j 存在，则对应的 $im - j$ 是一组解。其中第一次出现的为最小的解。

6.1 map 版本

```
map<ll, int> hash;
ll i, j;
ll bsgs(ll a, ll b, ll p) {
    ll xx, yy;
    if (exgcd(a, p, xx, yy) != 1)
        return -1;
    int i;
    a %= p;
    ll m = ceil(sqrt(p));
    hash.clear();
    ll tmp, ans = b % p;
    for (i = 0; i <= m; ++i) {
        hash[ans] = i;
        ans = ans * a % p;
    }
    tmp = f(a, m, p);
    ans = 1;
    for (i = 1; i <= m; ++i) {
        ans = ans * tmp % p;
        if (hash[ans] != 0)
            return i * m - hash[ans];
    }
    return -1;
}
```

6.2 二分查找版本

rec 为查找的结构体。

```
struct re{
    ll x;
    int id;
```

```
    bool operator < (const re & b) const {
        if (x == b.x)
            return id > b.id;
        return x < b.x;
    }
    bool operator == (const re & b) const {
        return x == b.x;
    }
} rec[100100];
ll bsgs(ll a, ll b, ll p) {
    int i;
    a %= p;
    ll m = ceil(sqrt(p));
    ll tmp, ans = b % p;
    for (i = 0; i <= m; ++i) {
        rec[i].id = i;
        rec[i].x = ans;
        ans = ans * a % p;
    }
    sort(rec, rec+1+m);
    int top = -1;
    for (i = 0; i <= 1+m; ++i)
        if (i == 0 || !(rec[i] == rec[i-1])) {
            rec[++top] = rec[i];
        } else {
            rec[top].id = max(rec[top].id, rec[i].id);
        }
    tmp = qm(a, m, p);
    ans = 1;
    int j;
    re tmp1;
    for (i = 1; i <= m; ++i) {
        ans = ans * tmp % p;
        tmp1.id = m+2;
        tmp1.x = ans;
        j = lower_bound(rec, rec+top, tmp1) - rec;
        if (rec[j].x == ans)
            return i * m - rec[j].id;
    }
    return -1;
}
```

6.3 hash 版本

LS 是记录计算次数，用于多次 hash，这样不用清空 hash 数组。

```
const int M = 65535, A = 2939;
ll rx[M + 10];
int ls[M + 10], LS = 0;
int ry[M + 10];
inline void ins(int x, int i) {
    int now = x & M;
    while (ls[now] == LS && rx[now] != x)
        now = now + A & M;
    ls[now] = LS;
```

```
    rx[now] = x;
    ry[now] = i;
    return;
}
inline int find(int x) {
    int now = x & M;
    while (ls[now] == LS) {
        if (rx[now] == x)
            return ry[now];
        now = now + A & M;
    }
    return -1;
}
ll bsgs(ll a, ll b, ll p) {
    LS++;
    register int i;
    a %= p;
    ll m = ceil(sqrt(p));
    ll tmp, ans = b % p;
    for (i = 0; i <= m; ++i) {
        ins(ans, i);
        ans = ans * a % p;
    }
    tmp = qm(a, m, p);
    ans = 1;
    for (i = 1; i <= m; ++i) {
        ans = ans * tmp % p;
        int j = find(ans);
        if (j != -1)
            return i * m - j;
    }
    return -1;
}
```

7 莫比乌斯反演

$F(n) = \sum_{d|n} f(d)$ $f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d})$

8 Pollard-Rho 分解

8.1 C++ 版本

```
const int Times = 10;
const int N = 5500;
LL ct, cnt;
LL fac[N], num[N];
LL pollard_rho(LL n, LL c) {
    LL i = 1, k = 2;
    LL x = rand() % (n - 1) + 1;
    LL y = x;
    while(true) {
        i++;
```

```
        x = (multi(x, x, n) + c) % n;
        LL d = gcd((y - x + n) % n, n);
        if(1 < d && d < n) return d;
        if(y == x) return n;
        if(i == k) {
            y = x;
            k <<= 1;
        }
    }
}
void find(LL n, int c) {
    if(n == 1) return;
    if(Miller_Rabin(n)) {
        fac[ct++] = n;
        return ;
    }
    LL p = n;
    LL k = c;
    while(p >= n) p = pollard_rho(p, c--);
    find(p, k);
    find(n / p, k);
}
int main() {
    LL n;
    while(cin>>n) {
        ct = 0;
        find(n, 120);
        sort(fac, fac + ct);
        num[0] = 1;
        int k = 1;
        for(int i=1; i<ct; i++){
            if(fac[i] == fac[i-1])
                ++num[k-1];
            else {
                num[k] = 1;
                fac[k++] = fac[i];
            }
        }
        cnt = k;
        for(int i=0; i<cnt; i++)
            cout<<fac[i]<<”^”<<num[i]<<” ”;
        cout<<endl;
    }
    return 0;
}
```

8.2 Java 版本

```
import java.math.BigInteger;
import java.security.SecureRandom;
class PollardRho  {
    private final static BigInteger ZERO = new BigInteger(”0”);
    private final static BigInteger ONE  = new BigInteger(”1”);
    private final static BigInteger TWO  = new BigInteger(”2”);
    private final static SecureRandom random = new SecureRandom();
```

```java

    public static BigInteger rho(BigInteger N) {
        BigInteger divisor;
        BigInteger c  = new BigInteger(N.bitLength(), random);
        BigInteger x  = new BigInteger(N.bitLength(), random);
        BigInteger xx = x;
        if (N.mod(TWO).compareTo(ZERO) == 0) return TWO;
        do {
            x  =  x.multiply(x).mod(N).add(c).mod(N);
            xx = xx.multiply(xx).mod(N).add(c).mod(N);
            xx = xx.multiply(xx).mod(N).add(c).mod(N);
            divisor = x.subtract(xx).gcd(N);
        } while((divisor.compareTo(ONE)) == 0);
        return divisor;
    }

    public static void factor(BigInteger N) {
        if (N.compareTo(ONE) == 0) return;
        if (N.isProbablePrime(20)) {
            System.out.println(N);
            return;
        }
        BigInteger divisor = rho(N);
        factor(divisor);
        factor(N.divide(divisor));
    }

    public static void main(String[] args){
        BigInteger N = BigInteger.valueOf(120);
        factor(N);
    }
}
```

9 高斯消元（数论）

```cpp
const int MAXN = 1e2+5;
int equ, var;///个方程equ 个变量var
int a[MAXN][MAXN];//增广矩阵/
int x[MAXN];//解的数目/
bool free_x[MAXN];//判断是不是自由变元/
int free_num;//自由变元的个数/
int Gauss(){
    int Max_r;//当前列绝对值最大的存在的行/
    ///: 处理当前的列col
    int row = 0;
    int free_x_num;
    int free_index;
    for(int col=0; row<equ&&col<var; row++,col++){
        Max_r = row;
        for(int i=row+1; i<equ; i++)
            if(abs(a[i][col]) > abs(a[Max_r][col]))
                Max_r = i;

        if(Max_r != row)
```

```
            for(int i=0; i<var+1; i++)
                swap(a[row][i], a[Max_r][i]);

        if(a[row][col] == 0){
            row--;
            continue;
        }
        for(int i=row+1; i<equ; i++){
            if(a[i][col]){
                int lcm = LCM(abs(a[i][col]), abs(a[row][col]));
                int tp1=lcm/abs(a[i][col]), tp2=lcm/abs(a[row][col]);
                if(a[row][col]*a[i][col] < 0)
                    tp2 = -tp2;
                for(int j=col; j<var+1; j++)
                    a[i][j] = tp1*a[i][j]-tp2*a[row][j];
            }
        }
    }
    for(int i=row; i<equ; i++)
        if(a[i][var])
            return -1;//无解/
    if(row < var) {
        for(int i=row-1; i>=0; i--){
            free_x_num = 0;
            for(int j=0; j<var; j++)
                if(a[i][j] && free_x[j]) {
                    free_x_num++;
                    free_index = j;
                }

            if(free_x_num > 1)
                continue;
            int tmp = a[i][var];
            for(int j=0; j<var; j++)
                if(a[i][j] && j!=free_index)
                    tmp -= a[i][j]*x[j];
            x[free_index] = tmp/a[i][free_index];/// 求出该变元.
            free_x[free_index] = 0; /// 该变元是确定的.
        }
        return var - row;//自由变元的个数/
    }
    for(int i=var-1; i>=0; i--) {
        int tmp = a[i][var];
        for(int j=i+1; j<var; j++)
            if (a[i][j])
                tmp -= a[i][j]*x[j];
        if (tmp%a[i][i])
            return -2; /// 说明有浮点数解，但无整数解.
        x[i] = tmp/a[i][i];
    }
    return 0;//唯一解/
}
int main(){
    while(cin>>equ>>var){
        for(int i=0; i<equ; i++){
            for(int j=0; j<var+1; j++)
                cin>>a[i][j];
```

```
        }
        cout<<Gauss()<<endl;
    }
    return 0;
}
```

10 自然数幂和（伯努利数）

$B_n = -\frac{1}{n+1}(C_{n+1}^0 B_0 + C_{n+1}^1 B_1 + \ldots + C_{n+1}^{n-1} B_{n-1})$

$\sum_{i=1}^{n} i^k = \frac{1}{k+1}\sum_{i=1}^{k+1} C_{k+1}^i B_{k+1-i}(n+1)^i$

其中 $Inv[N]$ 是逆元数组 $c[N][N]$ 是组合数 $b[N]$ 是伯努利数 $m[N]$ 是幂数组。

```
LL solve() {
    LL ans = 0;
    for (int i = 1;i <= k+1; ++i) {
        ans += c[k+1][i] * b[k+1 -i] % MOD * m[i] % MOD;
        ans %= MOD;
    }
    ans = ans * Inv[k+1] % MOD;
    return ans;
}
int main() {
    //首先 递推求逆元求出 Inv 数组。
    //其次 初始化组合数
    //初始化伯努利数
    memset(b, 0, sizeof(b));
    b[0] = 1;
    for (i = 1;i < N;++i) {
        for (j = 0;j < i; ++j) {
            b[i] += c[i+1][j] * b[j];
            b[i] %= MOD;
        }
        b[i] *= -1 * Inv[i + 1];
        b[i] %= MOD;
        b[i] = (b[i] + MOD) % MOD;
    }
    scanf("%d",&t);
    while (t--) {
        scanf("%lld %d",&n,&k);
        n %= MOD;
        //初始化幂数组
        m[0] = 1;
        for (i = 1;i <= N;++i)
            m[i] = m[i-1] * (n+1) % MOD;
        ans = solve();
        printf("%lld\n", ans);
    }
    return 0;
}
```

11 数论变换

```
const int N = 404005;
const int G = 3;
const ll mod = 1004535809;
ll w[2][N];
ll d;
ll la, lb, i, t, m, inv;
char s[400000];
ll a[400000];
ll b[400000];
ll pow(ll x, ll y, ll p) {
    static ll res;
    res = 1;
    while (y) {
        if (y & 1) res = res * x % p;
        x = x * x % p, y >>= 1;
    }
    return res;
}
inline int get_inv(ll x, ll mod) {
    return pow(x, mod - 2, mod);
}

inline void pre() {
    int i, t;
    w[0][0] = w[0][d] = 1;
    t = pow(G, (mod - 1) / d, mod);
    for (i = 1; i < d; ++i) w[0][i] = 1ll * w[0][i - 1] * t % mod;
    for (i = 0; i <= d; ++i) w[1][i] = w[0][d - i];
}
void NTT(ll *x, ll k, ll v) {
    int i, j, l, tmp;
    for (i = j = 0; i < k; ++i) {
        if (i > j) swap(x[i], x[j]);
        for (l = k >> 1; (j ^= l) < l; l >>= 1);
    }
    for (i = 2; i <= k; i <<= 1)
        for (j = 0; j < k; j += i)
            for (l = 0; l < i >> 1; ++l) {
                tmp = 1ll * x[j + l + (i >> 1)] * w[v][k / i * l] % mod;
                x[j + l + (i >> 1)] = (1ll * x[j + l] - tmp + mod) % mod;
                x[j + l] = (1ll * x[j + l] + tmp) % mod;
            }
}
int main() {
    scanf("%s", s); la = strlen(s); memset(a, 0, sizeof(a));
    for (i = 0; i < la; ++i) a[i] = s[la - 1 - i] - 48;
    scanf("%s", s); lb = strlen(s); memset(b, 0, sizeof(b));
    for (i = 0; i < lb; ++i) b[i] = s[lb - 1 - i] - 48;
    m = max(la, lb);
    for (d = 1; d <= m << 1; d <<= 1);
    pre();
    NTT(a, d, 0);
    NTT(b, d, 0);
    for (i = 0; i < d; ++i) a[i] = 1ll * a[i] * b[i] % mod;
    NTT(a, d, 1);
    for (inv = get_inv(d, mod), i = 0; i < d; ++i)
```

```
        a[i] = 1ll * a[i] * inv % mod;
    t = 0;
    for (i = 0; i < d; ++i) {
        a[i+1] += a[i] / 10;
        a[i] %= 10;
        if (a[i] != 0) t = i;
    }
    for (i = t; i >= 0; --i) printf("%lld", a[i]);
    printf("\n");
    return 0;
}
```

12 线性递推函数杜教模板

```
#include <cstring>
#include <cmath>
#include <algorithm>
#include <vector>
#include <string>
#include <map>
#include <set>
#include <cassert>
using namespace std;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define fi first
#define se second
#define SZ(x) ((int)(x).size())
typedef vector<int> VI;
typedef long long ll;
typedef pair<int,int> PII;
const ll mod=1000000007;
ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0); for(;b;b>>=1){if(b&1)res=res*a%mod;a=a*a%mod;}return res;}
// head
int _;
ll n;
namespace linear_seq {
    const int N=10010;
    ll res[N],base[N],_c[N],_md[N];
    vector<ll> Md;
    void mul(ll *a,ll *b,ll k) {
        rep(i,0,k+k) _c[i]=0;
        rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
        for (int i=k+k-1;i>=k;i--) if (_c[i])
            rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[j]]-_c[i]*_md[Md[j]])%mod;
        rep(i,0,k) a[i]=_c[i];
    }
    int solve(ll n,VI a,VI b) {
        ll ans=0,pnt=0;
        ll k=SZ(a);
```

```
        assert(SZ(a)==SZ(b));
        rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
        Md.clear();
        rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
        rep(i,0,k) res[i]=base[i]=0;
        res[0]=1;
        while ((1ll<<pnt)<=n) pnt++;
        for (int p=pnt;p>=0;p--) {
            mul(res,res,k);
            if ((n>>p)&1) {
                for (int i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;
                rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-res[k]*_md[Md[j]])%mod;
            }
        }
        rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
        if (ans<0) ans+=mod;
        return ans;
    }
    VI BM(VI s) {
        VI C(1,1),B(1,1);
        int L=0,m=1,b=1;
        rep(n,0,SZ(s)) {
            ll d=0;
            rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
            if (d==0) ++m;
            else if (2*L<=n) {
                VI T=C;
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                L=n+1-L; B=T; b=d; m=1;
            } else {
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                ++m;
            }
        }
        return C;
    }
    int gao(VI a,ll n) {
        VI c=BM(a);
        c.erase(c.begin());
        rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
        return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
    }
};
int main() {
    for (scanf("%d",&_);_;_--) {
        scanf("%lld",&n);
        printf("%d\n",linear_seq::gao(VI{31, 197, 1255, 7997, 50959, 324725, 2069239, 13185773, 84023455},n-2));
    }
}
```

13 积性函数求前缀和

13.1 莫比乌斯函数

首先我们习惯性把求区间和问题改成求前缀和，即求 $M(n)=\sum_{i=1}^{a}\mu(i)$. 我们知道 $\sum_{d|i}\mu(d)$ 只有在 $i=1$ 的情况下值为 1, 否则为 0. 所以有 $1=\sum_{i=1}^{n}\sum_{d|i}\mu(d)$. 我们换一个角度去考虑，考虑 $\mu(d)$ 的贡献，因为 $i=kd$ 时 $\mu(i/k)$ 即 $\mu(d)$ 会被计入贡献，所以当 k 值固定时，要被计入一次的函数值为 $\mu(k/k),\mu(2k/k)....\mu(jk/k)$ 其中 $j=\lfloor\frac{n}{k}\rfloor$. 所以 $1=\sum_{i=1}^{n}\sum_{d|i}\mu(d)=\sum_{k=1}^{n}\sum_{i=1}^{\lfloor\frac{n}{k}\rfloor}\mu(i)$ 即 $1=\sum_{k=1}^{n}M(\lfloor\frac{n}{k}\rfloor)$ 将 $k=1$ 的那一项移除即得 $M(n)=1-\sum_{k=2}^{n}M(\lfloor\frac{n}{k}\rfloor)$. 但是此时 n 还是很大，所以我们可以分块处理，对于小于 5000000 的部分用线性筛做，大于的部分递归加上记忆化处理。在递归计算时我们发现，在 $k\in[l,l/(n/l)]$ (整除) 这个区间里，$\lfloor\frac{n}{k}\rfloor$ 的值是相等的，所以这里也是一个可以优化的地方。

13.2 欧拉函数

知 $n=\sum_{d|n}\phi(d)$. 另 $M(n)=\sum_{i=1}^{n}\phi(i)$ 有 $\sum_{i=1}^{n}i=\sum_{i=1}^{n}\sum_{d|i}\phi(d)$ 即 $\frac{n(n+1)}{2}=\sum_{i=1}^{n}\sum_{d|i}\phi(d)$, 考虑 $\phi(d)$ 的贡献，有 $\frac{n(n+1)}{2}=\sum_{i=1}^{n}\sum_{j=1}^{\lfloor\frac{n}{i}\rfloor}\phi(j)$. 即 $\frac{n(n+1)}{2}=\sum_{i=1}^{n}M(\lfloor\frac{n}{i}\rfloor)$, 移项得 $M(n)=\frac{n(n+1)}{2}-\sum_{i=2}^{n}M(\lfloor\frac{n}{i}\rfloor)$. 分块记忆化递归处理即可。

```
const ll mod = 1e9 + 7;
const ll mod2 = 5e8 + 4;
const ll N = 1000000;
int prime[N + 10];
ll phi[N + 10];
bool p[N + 10];
ll i, j, top;
ll ans[2 * N + 10];
ll tot;
ll l, r, a, n, tmp;
map<ll, int> hs;

ll find(ll n) {
    if (n <= N)
        return phi[n];
    if (hs[n] != 0)
        return ans[hs[n]];
    ll an = (n % mod) * ((n + 1) % mod) % mod * mod2 % mod;
    for (ll l = 2, r; l <= n; l = r + 1) {
        r = n / (n / l);
        an -= find(n / l) * (r - l + 1) % mod;
        an = (an % mod + mod) % mod;
    }
    hs[n] = ++tot;
    ans[tot] = an;
    return an;
}

int main() {
    memset(p, 0, sizeof(p));
    tot = 0;
    hs.clear();
```

```
    phi[1] = 1;
    top = 0;
    for (i = 2; i <= N; ++i) {
        if (!p[i]) {
            prime[++top] = i;
            phi[i] = i-1;
        }
        for (j = 1; j <= top; ++j) {
            if (i * prime[j] > N)
                break;
            p[i * prime[j]] = 1;
            if (i % prime[j] == 0) {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            phi[i * prime[j]] = phi[i] * phi[prime[j]];
        }
    }
    phi[0] = 0;
    tot = 0;
    for (i = 2; i <= N; ++i) {
        phi[i] += phi[i-1];
        phi[i] %= mod;
    }
    scanf("%lld", &n);
    a = find(n);
    printf("%lld\n", a);
    return 0;
}
```