数学

数论

线性递推求逆元

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

求区间因数个数 $O(\sqrt{n})$（数论分块）

```
// 求 [1,n] 之间每个数因数和，数论分块
void solve(){
    int n;
    cin >> n;
    ll ans = 0;
    for(int i = 1; i <= n; ++i){
        int k = n / i;
        int r = min(n / k, n);
        ans += k * (r - i + 1);
        i = r;
    }
    cout << ans << endl;
```

```
    return;
}
```

扩展欧几里得

```
// 求解 ax + by = gcd(a, b)
void exgcd(ll a, ll b, ll &x, ll &y){
    if(b == 0){
        x = 1;
        y = 0;
        return;
    }
    exgcd(b, a % b, y, x);
    y = y - a / b * x;
    return;
}
```

```
// OI-wiki 非递归写法
int exgcd(int a, int b, int &x, int &y) {
  int x1 = 1, x2 = 0, x3 = 0, x4 = 1;
  while (b != 0) {
    int c = a / b;
    std::tie(x1, x2, x3, x4, a, b) =
        std::make_tuple(x3, x4, x1 - x3 * c, x2 - x4 * c, b, a - b * c);
  }
  x = x1, y = x2;
  return a;
}
```

中国剩余定理

```
// x % r = a
LL CRT(int k, LL* a, LL* r) {
  LL n = 1, ans = 0;
  for (int i = 1; i <= k; i++) n = n * r[i];
  for (int i = 1; i <= k; i++) {
    LL m = n / r[i], b, y;
    exgcd(m, r[i], b, y);  // b * m mod r[i] = 1
    ans = (ans + a[i] * m * b % n) % n;
  }
  return (ans % n + n) % n;
}
```

扩展中国剩余定理

```
const int MAXN = 1000010;
ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}
ll lcm(ll a, ll b){
    return a / gcd(a, b) * b;
}
struct node{
    ll r, a;
    node(ll r = 0, ll a = 0) : r(r), a(a){
    }
};
ll mul(ll a,ll b,ll p){
    b = (b % p + p) % p;
    ll ans=0,x=a;
    while(b!=0)
    {
        if(b&1)
        {
            ans=(ans+x)%p;
        }
        x=(x+x)%p;
        b>>=1;
    }
    return (ans%p+p)%p;
}
void exgcd(ll a, ll b, ll &x, ll &y){
    if(b == 0){
        x = 1;
        y = 0;
        return;
    }
    exgcd(b, a % b, x, y);
    ll tx = x, ty = y;
    x = ty;
    y = tx - (a / b) * ty;
    return;
}
node merge(node a, node b){
    ll g = gcd(a.a, b.a);
    ll l = lcm(a.a, b.a);
    ll x, y;
    exgcd(a.a / g, b.a / g, x, y);
    ll k1 = (a.r +  mul(mul(x % l, ((b.r - a.r) / g) % l, l), a.a, l) + l) % l;
    return node(k1, l);
}
int main(){
    int n;
    cin >> n;
    stack<node> s;
    while(n--){
        ll r, a;
        cin >> a >> r;
```

```
        s.push(node(r, a));
    }
    while(s.size() >= 2){
        node a, b;
        a = s.top();
        s.pop();
        b = s.top();
        s.pop();
        s.push(merge(a, b));
    }
    cout << s.top().r;
    return 0;
}
/*input
3
11 6
25 9
33 17
*/
/*output
809
*/
```

BSGS

```
// 求解 a^x = b (mod p)
// 无解返回 -1
// 2 <= a, b < p < 2 ^ 32
ll BSGS(ll a, ll b, ll p){
    ll k = sqrt(p) + 1;
    unordered_map<ll, ll> s;
    ll now = b;
    rep(i, 0, k){
        s[now] = i;
        now = now * a % p;
    }
    ll ak = 1;
    rep(i, 1, k) ak = ak * a % p;
    now = ak;
    rep(i, 1, k){
        if(s.count(now)){
            return i * k - s[now];
        }
        now = now * ak % p;
    }
    return -1;
}
```

Miller Rabin

```
bool millerRabin(int n) {
  if (n < 3 || n % 2 == 0) return n == 2;
  if (n % 3 == 0) return n == 3;
  int u = n - 1, t = 0;
  while (u % 2 == 0) u /= 2, ++t;
  // test_time 为测试次数，建议设为不小于 8
  // 的整数以保证正确率，但也不宜过大，否则会影响效率
  for (int i = 0; i < test_time; ++i) {
    // 0, 1, n-1 可以直接通过测试，a 取值范围 [2, n-2]
    int a = rand() % (n - 3) + 2, v = quickPow(a, u, n);
    if (v == 1) continue;
    int s;
    for (s = 0; s < t; ++s) {
      if (v == n - 1) break;  // 得到平凡平方根 n-1，通过此轮测试
      v = (long long)v * v % n;
    }
    // 如果找到了非平凡平方根，则会由于无法提前 break; 而运行到 s == t
    // 如果 Fermat 素性测试无法通过，则一直运行到 s == t 前 v 都不会等于 -1
    if (s == t) return 0;
  }
  return 1;
}
```

Pollard-Rho 算法

```
/*对于一个数 n，用 Miller Rabin 算法 判断是否为素数，如果是就可以直接返回了，否则用
Pollard-Rho 算法找一个因子 p，将 n 除去因子 p。再递归分解 n 和 p，用 Miller Rabin 判断
是否出现质因子，并用 max_factor 更新就可以求出最大质因子了。由于这个题目的数据过于庞大，
用 Floyd 判环的方法是不够的，这里采用倍增优化的方法。*/
#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <iostream>

using namespace std;
using ll = long long;
using ull = unsigned long long;

int t;
ll max_factor, n;

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

ll bmul(ll a, ll b, ll m) {  // 快速乘
  ull c = (ull)a * (ull)b - (ull)((long double)a / m * b + 0.5L) * (ull)m;
  if (c < (ull)m) return c;
  return c + m;
```

```
}

ll qpow(ll x, ll p, ll mod) {  // 快速幂
  ll ans = 1;
  while (p) {
    if (p & 1) ans = bmul(ans, x, mod);
    x = bmul(x, x, mod);
    p >>= 1;
  }
  return ans;
}

bool Miller_Rabin(ll p) {  // 判断素数
  if (p < 2) return false;
  if (p == 2) return true;
  if (p == 3) return true;
  ll d = p - 1, r = 0;
  while (!(d & 1)) ++r, d >>= 1;  // 将d处理为奇数
  for (ll k = 0; k < 10; ++k) {
    ll a = rand() % (p - 2) + 2;
    ll x = qpow(a, d, p);
    if (x == 1 || x == p - 1) continue;
    for (int i = 0; i < r - 1; ++i) {
      x = bmul(x, x, p);
      if (x == p - 1) break;
    }
    if (x != p - 1) return false;
  }
  return true;
}

ll Pollard_Rho(ll x) {
  ll s = 0, t = 0;
  ll c = (ll)rand() % (x - 1) + 1;
  int step = 0, goal = 1;
  ll val = 1;
  for (goal = 1;; goal *= 2, s = t, val = 1) {  // 倍增优化
    for (step = 1; step <= goal; ++step) {
      t = (bmul(t, t, x) + c) % x;
      val = bmul(val, abs(t - s), x);
      if ((step % 127) == 0) {
        ll d = gcd(val, x);
        if (d > 1) return d;
      }
    }
    ll d = gcd(val, x);
    if (d > 1) return d;
  }
}

void fac(ll x) {
  if (x <= max_factor || x < 2) return;
  if (Miller_Rabin(x)) {              // 如果x为质数
    max_factor = max(max_factor, x);  // 更新答案
```

```
    return;
  }
  ll p = x;
  while (p >= x) p = Pollard_Rho(x);  // 使用该算法
  while ((x % p) == 0) x /= p;
  fac(x), fac(p);  // 继续向下分解x和p
}

int main() {
  cin >> t;
  while (t--) {
    srand((unsigned)time(NULL));
    max_factor = 0;
    cin >> n;
    fac(n);
    if (max_factor == n)  // 最大的质因数即自己
      cout << "Prime\n";
    else
      cout << max_factor << '\n';
  }
  return 0;
}
/*input
6
2
13
134
8897
1234567654321
1000000000000
*/
/*output
Prime
Prime
67
41
4649
5
*/
```

欧拉函数

```
constexpr int N = 1E7;
constexpr int P = 1000003;

bool isprime[N + 1];
int phi[N + 1];
std::vector<int> primes;

std::fill(isprime + 2, isprime + N + 1, true);
phi[1] = 1;
```

```
for (int i = 2; i <= N; i++) {
    if (isprime[i]) {
        primes.push_back(i);
        phi[i] = i - 1;
    }
    for (auto p : primes) {
        if (i * p > N) {
            break;
        }
        isprime[i * p] = false;
        if (i % p == 0) {
            phi[i * p] = phi[i] * p;
            break;
        }
        phi[i * p] = phi[i] * (p - 1);
    }
}
```

莫比乌斯反演

```
void getMu() {
  mu[1] = 1;
  for (int i = 2; i <= n; ++i) {
    if (!flg[i]) p[++tot] = i, mu[i] = -1;
    for (int j = 1; j <= tot && i * p[j] <= n; ++j) {
      flg[i * p[j]] = 1;
      if (i % p[j] == 0) {
        mu[i * p[j]] = 0;
        break;
      }
      mu[i * p[j]] = -mu[i];
    }
  }
}
```

排列组合

球与盒子

球	箱	任意方式	至多一个球	至少一个球
可区分	可区分	1	2	3
不可区分	可区分	4	5	6
可区分	不可区分	7	8	9
不可区分	不可区分	10	11	12

```
// 注意模数
const int MOD = 1000000007;
const int MAXN = 1000010;
ll jc[MAXN];
ll inv[MAXN];
ll S[2000][2000];
ll p[2000][2000];
ll ksm(ll a, ll k){
    if(k < 0) return 0;
    ll ans = 1, now = a;
    while(k){
        if(k & 1) ans = (ans * now) % MOD;
        now = (now * now) % MOD;
        k >>= 1;
    }
    return ans;
}
ll C(int n, int m){
    return jc[n] * inv[n - m] % MOD * inv[m] % MOD;
}
ll f1(ll n, ll k){
    return ksm(k, n);
}
ll f2(ll n, ll k){
    if(n > k) return 0;
    return C(k, n) * jc[n] % MOD;
}
ll f3(ll n, ll k){
    return jc[k] * S[n][k] % MOD;;
}
ll f4(ll n, ll k){
    return C(n + k - 1, k - 1);
}
ll f5(ll n, ll k){
    if(n > k) return 0;
    return C(k, n);
}
ll f6(ll n, ll k){
    return C(n - 1, k - 1);
}
ll f7(ll n, ll k){
    ll ans = 0;
    rep(i, 0, k){
        ans += S[n][k - i];
    }
    return ans % MOD;
}
ll f8(ll n, ll k){
    if(n > k) return 0;
    return 1;
}
ll f9(ll n, ll k){
    return S[n][k];
```

```
}
ll f10(ll n, ll k){
    ll ans = 0;
    rep(i, 0, k){
        ans += p[n][k - i];
    }
    return ans % MOD;
}
ll f11(ll n, ll k){
    if(n > k) return 0;
    return 1;
}
ll f12(ll n, ll k){
    return p[n][k];
}

int main(){
    int N = 100000;
    jc[0] = 1;
    rep(i, 1, N) jc[i] = (jc[i - 1] * i) % MOD;
    inv[N] = ksm(jc[N], MOD - 2);
    drep(i, N - 1, 0) inv[i] = inv[i + 1] * (i + 1) % MOD;
    int n = 1000;
    S[0][0] = 1;
    rep(i, 1, n){
        S[i][i] = 1;
        S[i][0] = 0;
        rep(j, 1, i - 1){
            S[i][j] = (S[i - 1][j - 1] + j * S[i - 1][j]) % MOD;
        }
    }
    p[0][0] = 1;
    rep(i, 1, n){
        p[i][i] = 1;
        p[i][0] = 0;
        rep(j, 1, i - 1){
            p[i][j] = p[i - 1][j - 1];
            if(i >= j) p[i][j] = (p[i][j] + p[i - j][j]) % MOD;
        }
    }
    int t;
    cin >> t;
    while(t--){
        int op, n, k;
        cin >> op >> n >> k;
        ll ans;
        switch(op){
            case 1: ans = f1(n, k); break;
            case 2: ans = f2(n, k); break;
            case 3: ans = f3(n, k); break;
            case 4: ans = f4(n, k); break;
            case 5: ans = f5(n, k); break;
            case 6: ans = f6(n, k); break;
            case 7: ans = f7(n, k); break;
```

```
            case 8: ans = f8(n, k); break;
            case 9: ans = f9(n, k); break;
            case 10: ans = f10(n, k); break;
            case 11: ans = f11(n, k); break;
            case 12: ans = f12(n, k); break;
        }
        cout << ans << endl;
    }
    return 0;
}
```

卡特兰数

```
int main() {
  f[0] = 1;
  for (int i = 1; i <= n; i++) f[i] = f[i - 1] * (4 * i - 2) / (i + 1);
  // 这里用的是常见公式2
  cout << f[n] << endl;
  return 0;
}
```

错排

错位排列数列的前几项为 0,1,2,9,44,265（OEIS A000166）。

$D_n = nD_{n-1} + (-1)^n$

$D_n = (n-1)(D_{n-1} + D_{n-2})$

多项式

全家桶

```
const int mod = 998244353, gen = 3;

inline int add(int x, int y) {
    return x + y >= mod ? x + y - mod : x + y;
}
inline int sub(int x, int y) {
    return x - y >= 0 ? x - y : x - y + mod;
}
inline int power(int x, int y) {
    int res = 1;
    for (; y; y >>= 1, x = 1ll * x * x % mod) {
        if (y & 1) { res = 1ll * res * x % mod; }
    }
    return res;
}
```

```
namespace Combin {
    std::vector<int> inv, fac, invf;

    void getCombin(int n) {
        if (inv.empty()) { inv = fac = invf = std::vector<int>(2, 1); }
        int m = inv.size(); n++;
        if (m < n) {
            inv.resize(n); fac.resize(n); invf.resize(n);
            for (int i = m; i < n; i++) {
                inv[i] = 1ll * (mod - mod / i) * inv[mod % i] % mod;
                fac[i] = 1ll * fac[i - 1] * i % mod;
                invf[i] = 1ll * invf[i - 1] * inv[i] % mod;
            }
        }
    }
    inline int binom(int n, int m) {
        if (n < m || m < 0) { return 0; }
        getCombin(n);
        return 1ll * fac[n] * invf[m] % mod * invf[n - m] % mod;
    }
}

using namespace Combin;

namespace Polynom {
    std::vector<int> rev, rt;

    void getRevRoot(int n) {
        int m = std::__lg(n);
        rev.resize(n);
        for (int i = 1; i < n; i++) {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << m - 1;
        }
        static int len = 1;
        if (len < n) {
            rt.resize(n);
            for (; len < n; len *= 2) {
                int uni = power(gen, (mod - 1) / (len * 2));
                rt[len] = 1;
                for (int i = 1; i < len; i++) {
                    rt[i + len] = 1ll * rt[i + len - 1] * uni % mod;
                }
            }
        }
    }
    void ntt(std::vector<int> &f, int n) {
        f.resize(n);
        for (int i = 0; i < n; i++) {
            if (i < rev[i]) { std::swap(f[i], f[rev[i]]); }
        }
        for (int len = 1; len < n; len *= 2) {
            for (int i = 0; i < n; i += len * 2) {
                for (int j = 0; j < len; j++) {
                    int x = f[i + j], y = 1ll * f[i + j + len] * rt[j + len] %
```

```
mod;
                    f[i + j] = add(x, y); f[i + j + len] = sub(x, y);
                }
            }
        }
    }
    std::vector<int> operator *(std::vector<int> f, std::vector<int> g) {
        int n = 1, m = f.size() + g.size(); m--;
        while (n < m) { n *= 2; }
        int invn = power(n, mod - 2);
        getRevRoot(n); ntt(f, n); ntt(g, n);
        for (int i = 0; i < n; i++) { f[i] = 1ll * f[i] * g[i] % mod; }
        std::reverse(f.begin() + 1, f.end()); ntt(f, n); f.resize(m);
        for (int i = 0; i < m; i++) { f[i] = 1ll * f[i] * invn % mod; }
        return f;
    }

    std::vector<int> polyInv(std::vector<int> f, int n) {
        if (n == 1) { return std::vector<int>(1, power(f[0], mod - 2)); }
        f.resize(n);
        std::vector<int> g = polyInv(f, n / 2), h(n);
        g.resize(n);
        for (int i = 0; i < n / 2; i++) { h[i] = g[i]; }
        int invn = power(n, mod - 2);
        getRevRoot(n); ntt(f, n); ntt(g, n);
        for (int i = 0; i < n; i++) { f[i] = 1ll * f[i] * g[i] % mod; }
        std::reverse(f.begin() + 1, f.end()); ntt(f, n);
        for (int i = 1; i < n / 2; i++) { f[i] = 0; }
        for (int i = n / 2; i < n; i++) { f[i] = 1ll * f[i] * invn % mod; }
        f[0] = 1; ntt(f, n);
        for (int i = 0; i < n; i++) { f[i] = 1ll * f[i] * g[i] % mod; }
        std::reverse(f.begin() + 1, f.end()); ntt(f, n);
        for (int i = n / 2; i < n; i++) { h[i] = sub(0, 1ll * f[i] * invn % mod);
}
        return h;
    }
    std::vector<int> operator ~(std::vector<int> f) {
        if (f.empty()) { return f; }
        int n = 1, m = f.size();
        while (n < m) { n *= 2; }
        f = polyInv(f, n); f.resize(m);
        return f;
    }

    std::vector<int> polyDeri(std::vector<int> f) {
        if (f.empty()) { return f; }
        int m = f.size();
        for (int i = 1; i < m; i++) { f[i - 1] = 1ll * f[i] * i % mod; }
        f.pop_back();
        return f;
    }
    std::vector<int> polyInte(std::vector<int> f) {
        f.push_back(0);
        int m = f.size();
```

```
        getCombin(m);
        for (int i = m - 1; i >= 1; i--) { f[i] = 1ll * f[i - 1] * inv[i] % mod; }
        f[0] = 0;
        return f;
    }

    std::vector<int> polyLn(std::vector<int> f) {
        if (f.empty()) { return f; }
        int m = f.size();
        f = (~f) * polyDeri(f);
        f.resize(m); f = polyInte(f); f.pop_back();
        return f;
    }
    std::vector<int> polyExp(std::vector<int> f, int n) {
        if (n == 1) { return std::vector<int>(1, 1); }
        f.resize(n);
        std::vector<int> g = polyExp(f, n / 2), h(n), g0;
        g.resize(n); g0 = polyLn(g);
        for (int i = 0; i < n / 2; i++) { h[i] = g[i]; }
        for (int i = 0; i < n; i++) { f[i] = sub(g0[i], f[i]); }
        int invn = power(n, mod - 2);
        getRevRoot(n); ntt(f, n); ntt(g, n);
        for (int i = 0; i < n; i++) { f[i] = 1ll * f[i] * g[i] % mod; }
        std::reverse(f.begin() + 1, f.end()); ntt(f, n);
        for (int i = n / 2; i < n; i++) { h[i] = sub(0, 1ll * f[i] * invn % mod);
}
        return h;
    }
    std::vector<int> polyExp(std::vector<int> f) {
        if (f.empty()) { return f; }
        int n = 1, m = f.size();
        while (n < m) { n *= 2; }
        f = polyExp(f, n); f.resize(m);
        return f;
    }
}

using Polynom::operator ~;
using Polynom::operator *;

std::vector<int> f, g;

signed main() {
    ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
    int n = 10;
    f.resize(n);
    for(int i = 0; i < n; ++i) f[i] = n - i;

    g = f * f; // 多项式乘法
    for(auto x : g) cout << x << " "; // output: 100 180 241 284 310 320 315 296
264 220 165 120 84 56 35 20 10 4 1
    cout << endl;

    // 反转数组
```

```
    for(auto x : Polynom::rev) cout << x << " ";// output: 0 16 8 24 4 20 12 28 2
18 10 26 6 22 14 30 1 17 9 25 5 21 13 29 3 19 11 27 7 23 15 31 299473306 529069507
981274199 7799

    g = ~f; // 多项式求逆
    for(auto x : g) cout << x << " "; // output: 299473306 529069507 981274199
779928313 758096709 534433074 787525252 466980036 314029169 45958780
    cout << endl;

    std::vector<int> ans = g * f; //检验
    for(auto x : ans) cout << x << " "; // output: 1 0 0 0 0 0 0 0 0 492697773
286624499 609620732 915646811 5112497 850919245 234670361 405946729 45958780
    cout << endl;

    g = Polynom::polyLn(f); // 多项式对数
    for(auto x : g) cout << x << " "; // output: 0 698771048 284499641 208633070
271946718 638610853 568755876 136957799 418817133 919723866
    cout << endl;

    g = Polynom::polyExp(f); // 多项式指数
    for(auto x : g) cout << x << " "; // output: 1 9 499122225 499122377 623903419
224607130 420100524 572223576 14670202 295773748
    cout << endl;
    return 0;
}
```

拉格朗日插值

```
int main(){
    int n = read(), m = read();
    rep(i, 1, n){
        x[i] = read();
        y[i] = read();
    }
    ll ans = 0;
    rep(i, 1, n){
        ll s1 = 1, s2 = 1;
        rep(j, 1, n){
            if(i == j) continue;
            s1 = s1 * (m - x[j]) % MOD;
            s2 = s2 * (x[i] - x[j]) % MOD;
        }
        ans = (ans + y[i] * s1 % MOD * inv(s2) % MOD + MOD) % MOD;
    }
    printf("%lld", ans);

    return 0;
}
/*input
3 100
1 4
```

```
2 9
3 16
*/
/*output
10201
*/
```