2023暑期集训数学模块题解

A:

题意:

给定x (x 可以为负数), 求满足$x = b^p$ 中p 的最大值

思路:

由唯一分解定理, x 可以被分解为$p_1^{\alpha_1}p_2^{\alpha_2}p_3^{\alpha_3}..p_k^{\alpha_k}$, 因为我们要求的是$p$ 的最大值, 那么可以得到$p = gcd(\alpha_1, \alpha_2, ...\alpha_k)$, 因为$x$ 可以小于0, 所以注意判断p 奇偶性即可, 因为如果$x < 0$, 那么p 就不能为偶数, 所以我们可以对p 一直除2到奇数就是答案

code

```
#include <bits/stdc++.h>

using namespace std;

typedef long long LL;
const int maxn = 1e6 + 10;

int f[maxn], p[maxn], cnt = 0;

void prime() {    //筛质数
    LL i, j;
    f[1] = 1;
    for (i = 2; i < maxn; i++) {
        if (f[i])
            continue;
        p[cnt++] = i;
        for (j = i * i; j < maxn; j += i) {
            f[j] = 1;
        }
    }
}

int gcd(int a, int b) { return b ? gcd(b, a % b) : a; } //欧几里得算法求gcd

int main() {
    int T, i, j;
    LL n;
    prime();
    scanf("%d", &T);
    int cas = 1;
    while (T--) {
        scanf("%lld", &n);
        int flag = (n < 0) ? 1 : 0; //提前判断给定的n是正数还是负数
        n = abs(n);
        i = 0;
        int g = 0;
        while (p[i] * p[i] <= n && i < cnt) {    //求唯一分解定理中的指数
            int t = 0;
```

```
            while (n % p[i] == 0) {
                n /= p[i];
                t++;
            }
            i++;
            if (t == 0) continue;
            if (flag == 1) {    //如果为负数就一直除2到奇数为止
                while (t % 2 == 0) t /= 2;
            }
            g = gcd(g, t);  //对每个指数求gcd
        }
        if (n > 1) g = 1;
        printf("Case %d: %d\n", cas++, g);
    }

    return 0;
}
```

B

题意:

10^4 次询问, 每一次对长度为n 的调和级数求和, 范围$n \leq 10^8$

思路:

如果每次询问暴力求, 会因为时间复杂度过高而超时, 所以我们可以采用预处理的技巧, 也就是提前处理好答案, 类似前缀和的方式, 然后每一次可以做到$O(1)$ 询问, 但是现在我们如果想做前缀和, 就要开长度为10^8 的double数组, 会因为内存占用过大而超限, 这个时候我们可以考虑分块的方法, 我们将10^8 分为10^6 块, 然后我们前缀和数组$pre[i]$存储的是 前i 块的和, 这样的化我们每次询问只需要知道当前询问的n 处在哪个块中, 然后在块中暴力循环求和即可, 时间复杂度$O(N + T * 100)$

code

```
#include <bits/stdc++.h>
using namespace std;
const double r=0.57721566490153286060651209;      //欧拉常数
const int maxn=1e4;
double sum[maxn];
void init(){
    sum[1]=1.0;
    for(int i=2;i<=maxn;i++)
        sum[i]=sum[i-1]+1.0/i;
}
int main(){
    int T,n;
    scanf("%d",&T);
    init();
    for(int i=1;i<=T;i++){
        scanf("%d",&n);
        if(n>maxn){
            double ans=log(n)+r+1.0/(2*n);
            printf("Case %d: %.10lf\n",i,ans);
        }
        else printf("Case %d: %.10lf\n",i,sum[n]);
    }
}
```

C:

题意:

10^4 次询问, 每次给定一个$Q(Q \leq 10^8)$, 找到一个最小的n, 使得$n!$ 后面有Q 个0

思路:

我们知道0的产生是因为2 * 5 = 10, 这个时候就会产生一个后缀0, 而$n! = 1 * 2 * 3 * . . * n$ 这$1 - n$ 个数中2 的幂次的个数肯定比5 的幂次的个数多, 这样的化我们可以推断出0的个数只和5这个因子的幂次有关, 所以我们只需要知道$n!$ 的唯一分解中5的幂次就可以确定其中有多少个0, 我们知道当n 增大的时候, 5的幂次也会逐渐增大, 这满足单调性, 我们可以尝试二分来获得答案.

(i

c

```
#include <bits/stdc++.h>

using namespace std;
typedef long long LL;

const int N = 1e6 + 10, M = 1e8 + 5;


LL cal(LL mid) {    //计算n!中5的幂次
    LL res = 0;
    while (mid) res += mid / 5, mid /= 5;
    return res;
}

int main() {
    int T;
    cin >> T;
    for (int _ = 1; _ <= T; _++) {
        LL n;
        cin >> n;
        LL l = 1, r = 1e12, ans = -1;
        while (l <= r) {    //二分答案
            LL mid = (l + r) / 2;
            LL num = cal(mid);
            if (num < n) l = mid + 1;
            else if (num == n) ans = mid, r = mid - 1;
```

```
            else r = mid - 1;
        }
        if (ans == -1) printf("Case %d: impossible\n", _);
        else printf("Case %d: %d\n", _, ans);
    }
    return 0;
}
```

D:

题意:

10^4 次询问, 每次给定两个数A, B, 求两个数之间三的倍数的个数

思路:

其实很简单, 注意到每三个数之间就会出现一个三的倍数, 那么$[A, B]$ 之间的三的倍数就可以直接根据区间长度来计算, 但是要注意一些边界情况

其实处理区间问题有一个很常用的思路, 我们想要知道$[A, B]$中三的倍数的个数, 我们设为$f(A, B)$, 那么我们可以转化为$f(1, B) - f(1, A - 1)$, 从1 到 B 的答案减去从1 到 $A - 1$ 的答案即可

code

```
#include <bits/stdc++.h>

using namespace std;
typedef long long LL;

const int N = 1e6 + 10, M = 1e8 + 5;

int main() {
    int T;
    cin >> T;
    for (int _ = 1; _ <= T; _++) {

        LL l, r;
        scanf("%lld%lld", &l, &r);
        l--;    // 转化为f(1, r) - f(1, l - 1);

        LL la = 0, ra = 0;
        if (l % 3 == 2) la++;
        if (r % 3 == 2) ra++;

        la += l / 3 * 2, ra += r / 3 * 2;

        printf("Case %d: %lld\n", _, ra - la);
    }
    return 0;
}
```

E:

题意:

T 次询问, 每次给定一个偶数n , 询问有多少个数对(a, b) , 使得$a + b = n$ (a, b 均为质数, $a \leq b$)

思路:

看到要对质数对计数, 我们首先将所有的质数筛出来, 然后我们知道$a + b = n$, 那么
$a = n - b \leq n - a \leq n/2$ 所以我们可以枚举a, 然后利用筛质数所获得的st 数组$O(1)$ 判断$n - a$ 是否为质数, 这样就可以完成计数

code

```
#include <bits/stdc++.h>

using namespace std;
typedef long long LL;

const int N = 1e7 + 10, mod = 1e8 + 7;

bool st[N];
int primes[N], cnt;

void get_primes(int n)  // 线性筛质数
{
    for (int i = 2; i <= n; i++) {
        if (!st[i]) primes[cnt++] = i;
        for (int j = 0; primes[j] <= n / i; j++) {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}


int main() {
    int T;
    cin >> T;
    get_primes(N);
    for (int _ = 1; _ <= T; _++) {
        int n;
        cin >> n;
        int ans = 0;
        for (int i = 0; primes[i] <= n / 2; i++) {
            if (!st[n - primes[i]]) ans++;
        }

        printf("Case %d: %lld\n", _, ans);
    }
    return 0;
}
```

F:

题意:

T 次询问, 每次询问n^k 的前三位和后三位

思路:

我们考虑后三位怎么求, 可以直接使用快速幂计算n^k % 1000 的值, 就是后三位, 时间复杂度$O(nlogk)$

接下来我们考虑对数方法, 我们对n^k 取对数, $log(n^k) = k * log(n)$ 所以$10^{klog(n)} = n^k$ (log 以10为底)

我们可以把整数部分去掉，那部分是小数部分要乘的 10 的个数。然后取小数 部分的前三位就行了

(将n^k 写成$0.a_1a_2..*10^l$ 的形式, 我们去掉的整数部分就是l)

code

```
#include <bits/stdc++.h>

using namespace std;
typedef long long LL;

const int N = 1e7 + 10, mod = 1e8 + 7;

int qmi(int a, int k, int p)  // 求a^k mod p
{
    int res = 1 % p;
    while (k) {
        if (k & 1) res = (LL) res * a % p;
        a = (LL) a * a % p;
        k >>= 1;
    }
    return res;
}


int main() {
    int T;
    cin >> T;
    for (int _ = 1; _ <= T; _++) {
        double n;
        int m;
        cin >> n >> m;
        double p = m * log10(n);
        double ans = pow(10, p - (int) p) * 100;
        printf("Case %d: %03d %03d\n", _, (int) ans, qmi(n, m, 1000));

    }
    return 0;
}
```

G:

题意:

问有多少种不同的面积为 a 的矩形，同时满足最短边不能小于 b

思路:

我们考虑抛去最短边的限制, 计算总的方案数, 然后减去最短边小于b的方案数就是答案

总的方案数⇔ a 的因子个数, 我们套用因子个数公式, 假设$a = p_1^{\alpha_1} p_2^{\alpha_2} p_3^{\alpha_3} .. p_k^{\alpha_k}$, 那么因子个数为 $\prod(1+\alpha_k)$

最短边小于b 的方案数: 因为最短边不能超过$\sqrt{a}$ 所以b 如果超过了$\sqrt{a}$ 那么无解

然后我们直接暴力枚举最短边多长即可, 时间复杂度为$O(T * \sqrt{a})$

这样仍然会导致超时, 所以我们分解质因数的时候可以考虑先筛质数, 然后用质数分解质因数, 可以少一个 log, 这样就可以通过本题

code

```
#include <bits/stdc++.h>

using namespace std;
typedef long long LL;

const int N = 1e6 + 10, mod = 1e8 + 7;

LL prime[N];
bool st[N];
LL m = 0;

void primes() { //筛质数
    for (LL i = 2; i <= N; i++) {
        if (!st[i]) {
            prime[++m] = i;
        }
        for (LL j = 1; j <= m && prime[j] * i <= N; j++) {
            if (i % prime[j] == 0)
                break;
            st[i * prime[j]] = true;
        }
    }
}

LL solve(LL n) {
    LL x = 1;
    for (LL i = 1; i <= m && prime[i] <= sqrt(n); i++) {
        if (n % prime[i] == 0) {
            LL y = 0;
            while (n % prime[i] == 0)
                n = n / prime[i], y++;
            x = x * (y + 1);
        }
    }
    if (n > 1)
        x = x * 2;
```

```
    return x;

}

int main() {
    LL t, a, b;
    primes();
    scanf("%lld", &t);
    for (LL j = 1; j <= t; j++) {
        scanf("%lld%lld", &a, &b);
        if (b * b > a) {
            printf("Case %lld: 0\n", j);
            continue;
        }
        LL p = solve(a);    //获得总的方案数
        p = p / 2;  // 注意(a, b), (b, a)会重复计数, 所以除以二
        for (LL i = 1; i < b; i++)
            if (a % i == 0)
                p--;
        printf("Case %lld: %lld\n", j, p);
    }
    return 0;
}
```

H:

题意:

T 组数据, 给定n, 求$1-n$ 中约数和为偶数的个数

思路:

我们首先有约数和公式:

$$o(n) = (1 + p_1 + p_1^2 + .. + p_1^{\alpha_1}) * ... * (1 + p_k + p_k^2 + .. + p_k^{\alpha_k})$$

但是就算有了这个公式也不能直接进行暴力求解, 我们分析一下性质

要使 $o(n)$ 为奇数，每一个因式都要为奇数，若 p_i 为奇因子，则 α_i 一 定为偶数，若 p_i 为偶因子，α_i 既能为偶数又能为奇数，我们分两种情况考虑，若 α_i 为偶数，则 n 的所有因子的幂均为偶数，即 n 为完全平方数，若 α_i 为奇数，则 $\frac{n}{2}$ 的所有因子的幂均为偶数，即 $\frac{n}{2}$ 为完全平方数。综上所述, 满足题意的 n 的形式为 k^2 , $2k^2$，个数为 $\sqrt{n} + \sqrt{n/2}$。

code

```
#include <bits/stdc++.h>

using namespace std;
typedef long long LL;

const int N = 1e7 + 10, mod = 1e8 + 7;

int main() {
    int T;
```

```
    cin >> T;
    for (int _ = 1; _ <= T; _++) {
        LL n;
        scanf("%lld", &n);

        printf("Case %d: %lld\n", _, n - (int) sqrt(n) - (int) sqrt(n / 2));
    }
    return 0;
}
```

I:

题意:

给定 n 个数 a_1，a_2 .....a_n，求欧拉函数 $\phi(b_i)$ 大于等于 a_i 的 b_i 的最小和

思路:

题意意思就是我们要对每个a_i 求 最小的b_i 然后加起来, 我们知道欧拉函数$\phi(n)$ 为 $1-n$ 中与n 互质的数的和, 我们知道当n 为素数时, $\phi(n)=n-1$

带入题目中即可得到$n-1\geq a_i$ 即 $n\geq a_i+1$ 于是只要找到大于等于a_i 的第一个素数即可, 这是因为任意两个相邻质数p_i,p_{i+1}之间的合数y 我们有$\phi(y)<\phi(p_i)<\phi(p_{i+1})$

(有兴趣的同学可以自行证明)

code

```
#include <bits/stdc++.h>

using namespace std;
typedef long long LL;

const int N = 1e7 + 10, mod = 1e8 + 7;

int primes[N], cnt;
bool st[N];

void get_primes(int n)  // 线性筛质数
{
    for (int i = 2; i <= n; i++) {
        if (!st[i]) primes[cnt++] = i;
        for (int j = 0; primes[j] <= n / i; j++) {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}


int main() {
    int T;
    cin >> T;
    get_primes(N);
    for (int _ = 1; _ <= T; _++) {
        int n, m;
        scanf("%d", &n);
```

```
        LL ans = 0;
        for (int i = 0; i < n; i++) {
            scanf("%d", &m);
            for (int j = m + 1; j < N; j++) {
                if (!st[j]) {
                    ans += j;
                    break;
                }
            }
        }

        printf("Case %d: %lld Xukha\n", _, ans);
    }
    return 0;
}
```

J:

题意:

给出 2 个数 a，b 的 Gcd（最大公约数 n）和 Lcm（最小公倍数 m），求所有符合条件的 a，b 中，$a+b$ 的最小值

思路:

我们设$p=\frac{m}{n}$ 那么我们就相当于在找 $A+B$ 的最小值, 其中$A=a/n, B=b/n, A*B=p$

我们由均值不等式可知A, B 相差越小, 和越小, 所以我们从$\sqrt{p}$ 向前枚举, 第一个满足的就是答案

code

```
#include <bits/stdc++.h>

using namespace std;
typedef long long LL;

const int N = 1e7 + 10, mod = 1e8 + 7;

int primes[N], cnt;
bool st[N];

void get_primes(int n)  // 线性筛质数
{
    for (int i = 2; i <= n; i++) {
        if (!st[i]) primes[cnt++] = i;
        for (int j = 0; primes[j] <= n / i; j++) {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}

int gcd(int a, int b)  // 欧几里得算法
{
```

```
    return b ? gcd(b, a % b) : a;
}


int main() {
    int T;
    cin >> T;
    for (int _ = 1; _ <= T; _++) {
        LL n, m;
        cin >> n >> m;
        LL c = m / n;
        LL ans = 1e18;
        for (int i = 1; i <= c / i; i++) {
            if (c % i == 0) {
                if (gcd(i, c / i) == 1) ans = min(ans, (i + c / i) * n);
            }
        }
        cout << ans << endl;
    }
    return 0;
}
```

K:

给出 n 和 k，求 $1^k + 2^k + 3^k + \ldots + n^k$ 的和，由于结果太大，输出 mod 10007 的结果。

思路:

我们发现因为要mod 10007, 所以我们n 在增大的过程中每到10007的倍数都会开始重复, 所以我们只需要算出到10007的前缀和即可

code

```
#include <bits/stdc++.h>

using namespace std;
typedef long long LL;

LL gcd(LL a, LL b) { return b ? gcd(b, a % b) : a; }

LL lcm(LL a, LL b) { return a / gcd(a, b) * b; }

void cinO() {
    ios::sync_with_stdio(0);
    cin.tie(0);
}


const int N = 10008, M = 5010, mod = 10007, INF = 0x3f3f3f3f;
const double eps = 1e-6;

int a[N];

int qmi(int a, int k, int p)  // 求a^k mod p
```

```
{
    int res = 1 % p;
    while (k) {
        if (k & 1) res = (LL) res * a % p;
        a = (LL) a * a % p;
        k >>= 1;
    }
    return res;
}

signed main() {
    cin0();
    int T;
    cin >> T;

    while (T--) {
        LL n, k;
        cin >> n >> k;

        for (int i = 1; i <= 10007; i++) {
            a[i] = (a[i - 1] + qmi(i, k, mod)) % mod;
        }

        cout << (n / 10007 * a[10007] % mod + a[n % 10007] % mod) % mod << endl;
    }


    return 0;
}
```

L:

题意:

给定两个整数 L 和 U，你需要在闭区间$[L, U]$内找到距离最接近的两个相邻质数 C_1和 C_2（即 $C_2 - C_1$ 是最小的），如果存在相同距离的其他相邻质数对，则输出第一对。

同时，你还需要找到距离最远的两个相邻质数 D_1和 D_2(即 $D1 - D2$是最大的)，如果存在相同距离的其他相邻质数对，则输出第一对。

思路:

数据范围在 10^8之上，因此不能直接使用筛素数的方法。观察到区间长度在10^6

所以考虑对于任意一个合数x, 他必定存在一个最小的质因子 d
，且这个最小的质因子 $d \leq \sqrt{x}$ 那么我们去筛区间 $[L, R]$ 的素数时候，可以把该区间所有的合数用他们的最小质因子筛掉。对于该范围内任意一个合数的质因子x, 有 $0 \leq x \leq 10^9$ --- $\sqrt{x} \leq 10^5$ 因此我们的思路就是:

1. 先把 $[1, 10^5]$ 内所有的素数筛出
2. 对于任意 $d \in [1, 10^5]$，我们把 $[L, R]$ 内所有以 d 作为最小质因子的合数用 d 筛掉, 如此一来，对于任意 $n \in [L, R]$，如果 n 是合数，则他必定会被 n
   的最小 质因子 d 筛掉

code

```cpp
#include <bits/stdc++.h>

using namespace std;

const int N = 1e6 + 10;

typedef long long LL;

int cnt, primes[N];
bool st[N];
int l, r;

void get_primes(int n)  // 线性筛质数
{
    memset(st, 0, sizeof st);
    cnt = 0;
    for (int i = 2; i <= n; i ++ ) {
        if (!st[i]) primes[cnt ++ ] = i;
        for (int j = 0; primes[j] <= n / i; j ++ ) {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}

int main() {
    while(cin >> l >> r) {
        get_primes(50000);

        memset(st, 0, sizeof st);

        //把[l,r]区间内所有的合数用他们的最小质因子筛掉
        for(int i = 0; i < cnt; i ++ ) {
            LL p = primes[i];
            for(LL j = max(2 * p, (l + p - 1) / p * p); j <= r; j += p )
                st[j - l] = 1;
        }

        //剩下的所有的都是素数了
        cnt = 0;
        for(int i = 0; i <= r - l; i ++ ) {
            if(!st[i] && i + l >= 2)
                primes[cnt ++] = i + l;
        }

        if(cnt < 2) puts("There are no adjacent primes.");
        else {
            int a = 0, b = 0;
            //计算间隔
            for(int i = 0; i < cnt - 1; i ++ ) {
                int d1 = primes[i + 1] - primes[i];
                if(d1 < primes[a + 1]- primes[a]) a = i;
                if(d1 > primes[b + 1]- primes[b]) b = i;
            }
            printf("%d,%d are closest, %d,%d are most distant.\n",
                primes[a], primes[a + 1],
                primes[b], primes[b + 1]);
```

```
        }
    }
}
```

M

题意: 略

思路:

设经过时间 t 后，两只青蛙跳到同一个点上，所以$(x + m \times t) \equiv (y + n \times t) \bmod L$，所以 $t \times (n - m) + k \times L \equiv x - y$，这个时候我们有$t, k$ 两个变量, 所以可以用exgcd求解

code

```
#include <bits/stdc++.h>

#define int long long

using namespace std;
typedef long long LL;

const int N = 1e3 + 10, mod = 10007;

int exgcd(int a, int b, int &x, int &y)  // 扩展欧几里得算法，求x，y，使得ax + by = gcd(a, b)
{
    if (!b) {
        x = 1;
        y = 0;
        return a;
    }
    int d = exgcd(b, a % b, y, x);
    y -= (a / b) * x;
    return d;
}


signed main() {
    ios::sync_with_stdio(0);
    cin.tie(0);


    int x, y, m, n, l;
    cin >> x >> y >> m >> n >> l;

    int a = 0, b = 0, M, s;
    M = exgcd(n - m, l, a, b);
    if ((x - y) % M != 0 || n == m) cout << "Impossible" << endl;
    else {
        s = l / M;
        a = a * ((x - y) / M);
        a = (a % s + s) % s;
```

```
        cout << a << endl;
    }
    return 0;
}
```

N:

题意: 给定一些同余方程, 解出x

思路:

中国剩余定理板子题

利用中国剩余定理, 直接求解即可

code

```
#include<iostream>
#include<algorithm>

using namespace std;
struct pig {
    int a1, b1;
} a[11];//定义结构体存储a与b

int gcd(int x, int y) {//公约数函数，用来求每一次循环累加的值
    if (x % y == 0) return y;
    else return gcd(y, x % y);
}

int main() {
    long long n, ans, sum = 1;//数据较大用longlong
    cin >> n;
    for (int i = 1; i <= n; ++i) {
        cin >> a[i].a1 >> a[i].b1;
    }
    ans = a[1].b1;//初始化最小值
    for (int i = 1; i < n; ++i) {//开始循环
        sum = sum * a[i].a1 / gcd(sum, a[i].a1);//sum为累加的值
        while (ans % a[i + 1].a1 != a[i + 1].b1) {//循环条件
            ans += sum;
        }
    }
    cout << ans;
    return 0;
}
```

O:

题意: 求解扩展中国剩余定理

思路:

课上例题, 根据推导过程实现即可, 注意可能会爆long long, 可以使用龟速乘

code

```
#include<cstdio>

using namespace std;
typedef long long lt;

lt read() { //快速读入
    lt f = 1, x = 0;
    char ss = getchar();
    while (ss < '0' || ss > '9') {
        if (ss == '-')f = -1;
        ss = getchar();
    }
    while (ss >= '0' && ss <= '9') {
        x = x * 10 + ss - '0';
        ss = getchar();
    }
    return f * x;
}

const int maxn = 100010;
int n;
lt ai[maxn], bi[maxn];

lt mul(lt a, lt b, lt mod) {    //龟速乘
    lt res = 0;
    while (b > 0) {
        if (b & 1) res = (res + a) % mod;
        a = (a + a) % mod;
        b >>= 1;
    }
    return res;
}

lt exgcd(lt a, lt b, lt &x, lt &y) {    //exgcd
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    lt gcd = exgcd(b, a % b, x, y);
    lt tp = x;
    x = y;
    y = tp - a / b * y;
    return gcd;
}

lt excrt() {
    lt x, y, k;
    lt M = bi[1], ans = ai[1];//第一个方程的解特判
    for (int i = 2; i <= n; i++) {
        lt a = M, b = bi[i], c = (ai[i] - ans % b + b) % b;//ax≡c(mod b)
        lt gcd = exgcd(a, b, x, y), bg = b / gcd;
        if (c % gcd != 0) return -1; //判断是否无解，然而这题其实不用
```

```
        x = mul(x, c / gcd, bg);
        ans += x * M;//更新前k个方程组的答案
        M *= bg;//M为前k个m的lcm
        ans = (ans % M + M) % M;
    }
    return (ans % M + M) % M;
}

int main() {
    n = read();
    for (int i = 1; i <= n; ++i)
        bi[i] = read(), ai[i] = read();
    printf("%lld", excrt());
    return 0;
}
```

P:

题意;

给定多项式$(by+ax)^k$ 计算 $x^n * y^m$ 的系数

思路:

根据二项式定理

$(a+b)^n = \sum_{k=0}^{n} C_n^k a^k b^{n-k}$

所以$(ax+by)^k = \sum_{p=0}^{k} C_k^p (ax)^p (by)^{k-p} = \sum_{p=0}^{k} C_k^p a^p b^{k-p} x^p y^{k-p}$

算a^n, b^m 用到快速幂

然后我们预处理出来组合数(可以使用递推也可以使用逆元)

code

```
#include<iostream>

using namespace std;
const int mo = 10007;
const int maxn = 1007;

//给定一个多项式(by+ax)^k，请求出多项式展开后x^n*y^m 项的系数。
int a, b, k, n, m;
int c[maxn][maxn];

int ksm(int a, int b) {
    int base = a, ans = 1;
    while (b) {
        if (b & 1) ans = (ans * base) % mo;
        base = (base * base) % mo;
        b >>= 1;
    }
    return ans;
}
```

```
int main() {
    cin >> a >> b >> k >> n >> m;
    a %= mo;
    b %= mo;
    a = ksm(a, n);
    b = ksm(b, m);
    for (int i = 1; i <= k; i++) {
        C[i][0] = 1;
        C[i][i] = 1;
    }
    int p = min(n, m);
    for (int i = 2; i <= k; i++)//从1开始就会WA
    {
        for (int j = 1; j <= p; j++) {
            C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % mo;
        }
    }
    int ans = (a * b) % mo;
    ans = (ans * C[k][p]) % mo;
    cout << ans << endl;
    return 0;
}
```

Q:

题意:

多尔的家人正在庆祝多尔的生日派对。他们爱多尔，所以他们计划让他的生日蛋糕变得不可思议的!

蛋糕是一个由$n\times n$ 的等边正方形组成的形状，长度为1。每个方块要么是空的，要么是由一个巧克力组成的。他们买了蛋糕，便开始把巧克力放在蛋糕上。“家庭之门”的幸福值等于蛋糕中同一行或同一列中装有巧克力的一对方块的数量。多尔的家人想知道他们的幸福程度是多少?

思路:

对于每一行以及每一列, 我们求出巧克力的数目为x, 那么这一行\列中一对方块的数量为C_x^2 , 我们分别遍历求和即可

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    int n, t = 0, c = 0; //初始化
    char ch[105][105];
    cin >> n;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            cin >> ch[i][j];   //输入
    for (int i = 1; i <= n; i++) {
        c = 0;
        for (int j = 1; j <= n; j++) {
```

```
            if (ch[i][j] == 'C')
                c++;
        }
        t += (c * (c - 1)) / 2;
    }            //横向遍历
    for (int j = 1; j <= n; j++) {
        c = 0;
        for (int i = 1; i <= n; i++) {
            if (ch[i][j] == 'C')
                c++;
        }
        t += (c * (c - 1)) / 2;
    }          //竖向遍历
    cout << t;
}
```

R:

题意:

给你 n 个数 $a[1...n]$ 问你满足 $a[i]\times a[j]\times a[k]$ 的值最小，且$i<j<k$ 的有序对有几个?

思路:

先考虑如何使得值最小, 我们只需要对a 排序, 然后取前3个数乘起来就是答案

接下来我们考虑重复情况

- 设c 为最小的数的数量, 如果$c\geq 3$ 那么答案为C_c^3
- 如果$c=2$ 我们设次小值的数量为b, 答案为 b
- 如果$c=1,b>1$ 答案为b 中选两个 C_b^2
- $c=1,b=1$ 设第三小的值的个数为d, 答案为d

code

```
#include <iostream>
#include <algorithm>
using namespace std;
#define int long long

int a[100010];

signed main() {
    int n, ans = 0; cin >> n;
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
    }
    sort (a + 1, a + n + 1); // 排序
    for (int i = 1; i <= n; ++i) {
        if (a[i] == a[3]) ans ++;
    } // 找出于 a[3] 相同的数有多少个
```

```
    if (a[1] == a[3]) cout << ans * (ans - 1) * (ans - 2) / 6 << endl; // 第一种情况
    else if (a[2] == a[3]) cout << ans * (ans - 1) / 2 << endl; // 第二种情况
    else cout << ans << endl;
    return 0; // 以三种情况
}
```

S:

n 个 男孩, m 个女孩, 选择t 人，保证其中至少4个男孩和1个女孩。问有多少种选择方式。

思路:

我们可以枚举剩下的人有多少个是男生, 那么其他剩下的人就是女生, 然后我们利用排列组合算出方案即可

i 从0 枚举到 $min(t-5, n-4)$, $ans + C_n^{4+i} * C_m^{t-4-i}$

```
#include <cstdio>
#include <algorithm>
using namespace std;
long long ans;
long long com(int n, int m) {
    long long res = 1;
    for (register int i = m - n + 1; i <= m; ++i)
        res = res * i / (i - m + n);
    return res;
}

int main() {
    int n, m, t;
    scanf("%d%d%d", &n, &m, &t);
    for (register int i = max(1, t - n); i <= min(t - 4, m); ++i)
        ans += com(i, m) * com(t - i, n);
    printf("%I64d\n", ans);
    return 0;
}
```

T:

n 个 1, 和 m 个 0组成字符串, 要求在组成的字符串中在任意前k 个字符中1的个数大于等于0的个数, 求这样的字符串有多少?

思路:

可以考虑把1的个数与0的个数的**和**看成x 坐标，1的个数与0的个数的**差**看成y坐标

向右上走（x 坐标加1，y 坐标加1）就表示这个字符选择1。

向右下走（x坐标加1，y 坐标减1）就表示这个字符选择0。

我们经过这样的转化之后, 可以将题意转化为就表示从$(0, 0)$ 走$n+m$ 步到达$(n+m, n-m)$，这相当于从$n+m$步中选出m步向右下走，也就是$C(n+m, m)$

考虑限制条件, 任意前缀中1的个数大于等于0的个数, 也就是这条路径不能经过直线$y=-1$

通过对称性发现, 从$(0,0)$ 走到 $y=-1$ 上的一个点, 相当于从$(0,-2)$ 走到该点, 也就是说, 路径经过直线$y=-1$ 的方案数就是从(0, -2)走$n+m$ 步到达$(n+m,n-m)$ 这个方案数为$C(n+m,m-1)$, 因为我们是对称下来的, 所以方案有一一对应的关系, 不重不漏

所以最终答案为$C(n+m,m)-C(n+m,m-1)$

(有兴趣的同学可以自行了解卡特兰数)

code

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

inline int read() {
    int res = 0; bool bo = 0; char c;
    while (((c = getchar()) < '0' || c > '9') && c != '-');
    if (c == '-') bo = 1; else res = c - 48;
    while ((c = getchar()) >= '0' && c <= '9')
        res = (res << 3) + (res << 1) + (c - 48);
    return bo ? ~res + 1 : res;
}

const int N = 2e6 + 5, PYZ = 20100403;

int n, m, fac[N], inv[N];

int qpow(int a, int b) {
    int res = 1;
    while (b) {
        if (b & 1) res = 1ll * res * a % PYZ;
        a = 1ll * a * a % PYZ;
        b >>= 1;
    }
    return res;
}

int C(int x, int y) {
    int z = 1ll * fac[x] * inv[y] % PYZ;
    return 1ll * z * inv[x - y] % PYZ;
}

int main() {
    int i; fac[0] = 1; n = read(); m = read();
    for (i = 1; i <= n + m; i++) fac[i] = 1ll * fac[i - 1] * i % PYZ;
    inv[n + m] = qpow(fac[n + m], PYZ - 2);
    for (i = n + m - 1; i >= 0; i--)
        inv[i] = 1ll * inv[i + 1] * (i + 1) % PYZ;
    printf("%d\n", (C(n + m, m) - C(n + m, m - 1) + PYZ) % PYZ);
    return 0;
```

```
}
```

U:

题意:

构造一个左边严格单增，右边严格单减的数组，并且数组中有且仅有一对数相同，问有多少种构造方法。

思路:

题目要求我们最后的数组a 具有单峰性, 又因为其中有一对相同的元素, 所以这一对相同的元素必定在最大值两边, 考虑把其余的数字划分成两部分，把第一部分放到最大数的左边、把第二部分放到右边。因为最大数左右都没有重复的数字，所以能排成一个满足条件的数组。每种分法一一对应着一个满足条件的数组。

现在已经有三个数字确定(最大值和一对相同值), 剩余的数字划分到两边有两种选择, 方案为2^{n-3}

那么如何确定数组中的数字呢?

我们先从1 - m中选择n - 1个数, 然后在从中选取一个非最大值作为重复数字, 方案数为$C_m^{n-1} * (n-2)$

综上所述, 答案为$2^{n-3} * C_m^{n-1} * (n-2)$

注意当$n = 2$ 的时候无解, 需要特判

code

```
#include<bits/stdc++.h>
using namespace std;

#define int long long
#define inv(x) ksm(x,mod - 2)

const int MAXN = 2e5 + 10;
const int INF = 2e9;
const int mod = 998244353;

int n,m;

inline int ksm(int a,int b) {
    if(b < 0) return 0;//判断一下即可
    if(a == 1) return 1;
    int res = 1;
    while(b) {
        if(b & 1) (res *= a) %= mod;
        (a *= a) %= mod;
        b >>= 1;
    }
    return res;
}

inline int C(int a,int b) {
    int res = 1;
    for(int i = a;i >= a - b + 1; --i)
```

```
        (res *= i) %= mod;
    for(int i = 1;i <= b; ++i)
        (res *= inv(i)) %= mod;
    return res;
}

signed main () {
    cin >> n >> m;
    cout << ksm(2,n - 3) % mod * C(m,n - 1) % mod * (n - 2) % mod << endl;
    return 0;//End.
}
```

V:

题意：8个小球与盒模型模板题

思路：

7.33 斯特林数

7.33.1 第一类斯特林数

第一类 Stirling 数 $S(p,k)$ 的一个的组合学解释是：将 p 个物体排成 k 个非空循环排列的方法数。

$S(p,k)$ 的递推公式：$S(p,k)=(p-1)S(p-1,k)+S(p-1,k-1), 1\le k\le p-1$

边界条件：$S(p,0)=0, p\ge 1 \quad S(p,p)=1, p\ge 0$

7.33.2 第二类斯特林数

第二类 Stirling 数 $S(p,k)$ 的一个的组合学解释是：将 p 个物体划分成 k 个非空的不可辨别的（可以理解为盒子没有编号）集合的方法数。

$S(p,k)$ 的递推公式：$S(p,k)=kS(p-1,k)+S(p-1,k-1), 1\le k\le p-1$

边界条件：$S(p,0)=0, p\ge 1 \quad S(p,p)=1, p\ge 0$

也有卷积形式：

$$S(n,m)=\frac{1}{m!}\sum_{k=0}^{m}(-1)^k C(m,k)(m-k)^n=\sum_{k=0}^{m}\frac{(-1)^k(m-k)^n}{k!(m-k)!}=\sum_{k=0}^{m}\frac{(-1)^k}{k!}\times\frac{(m-k)^n}{(m-k)!}$$

7.34 各种情况下小球放盒子的方案数

k 个球	m 个盒子	是否允许有空盒子	方案数
各不相同	各不相同	是	m^k
各不相同	各不相同	否	$m!\text{Stirling2}(k,m)$
各不相同	完全相同	是	$\sum_{i=1}^{m}\text{Stirling2}(k,i)$
各不相同	完全相同	否	$\text{Stirling2}(k,m)$
完全相同	各不相同	是	$C(m+k-1,k)$
完全相同	各不相同	否	$C(k-1,m-1)$
完全相同	完全相同	是	$\frac{1}{(1-x)(1-x^2)\ldots(1-x^m)}$ 的 x^k 项的系数
完全相同	完全相同	否	$\frac{x^m}{(1-x)(1-x^2)\ldots(1-x^m)}$ 的 x^k 项的系数

code

```cpp
#include <bits/stdc++.h>
#define mod 1000000007
#define int long long
using namespace std;
const int N=2000;
const int M=1005;
int c[2010][1010];//组合数
int f[1010][1010];
int kk[1010];//k的阶乘
int S[1010][1010];//第二类斯特林数

void init()//预处理组合数，第二类斯特林数，k的阶乘
{
    //组合数
    c[0][0]=1;
    for(int i=1;i<=N;i++){
        c[i][0]=1;
        for(int j=1;j<=min(M,i);j++)
            c[i][j]=(c[i-1][j]+c[i-1][j-1])%mod;

    }
    //同球同盒递推式处理
    for(int i=0;i<=M;i++){
        for(int j=1;j<=M;j++){
            if(j==1||i==0) f[i][j]=1;
            else if(i<j) f[i][j]=f[i][i];
            else if(i>=j) f[i][j]=(f[i-j][j]+f[i][j-1])%mod;
        }
    }
    //第二类斯特林数
    S[0][0]=1;
    for(int i=1;i<=M;i++){
        for(int j=1;j<=M;j++){
            S[i][j]=S[i-1][j-1]+j*S[i-1][j];
            S[i][j]%=mod;
        }
    }
    //k的阶乘
    kk[1]=1;
    for(int i=2;i<=M;i++){
        kk[i]=(kk[i-1]*i)%mod;
    }
}
```

```
int ksm(int n,int k)//快速幂
{
    int ans=1;
    while(k)
    {
        if(k&1) ans=(ans*n)%mod;
        n=(n*n)%mod;
        k=k>>1;
    }
    return ans;
}

int cond1(int n,int k)
{
    return ksm(k,n);
}

int cond2(int n,int k)
{
    if(k<n) return 0;
    int sum=1;
    for(int i=k;i>=k-n+1;i--){
        sum=(sum*i)%mod;
    }
    return sum;
}

int cond3(int n,int k)
{
    return (kk[k]*S[n][k])%mod;
}

int cond4(int n,int k)
{
    return c[n+k-1][k-1];
}

int cond5(int n,int k)
{
    if(k<n) return 0;
    return c[k][n];
}
```

```
int cond7(int n,int k)
{
    int sum=0;
    for(int i=1;i<=k;i++){
        sum=(sum+S[n][i])%mod;
    }
    return sum;
}

int cond8(int n,int k)
{
    if(k<n) return 0;
    return 1;
}

int cond9(int n,int k)
{
    return S[n][k];
}

int cond10(int n,int k)
{
    return f[n][k];
}
int cond11(int n,int k)
{
    if(k<n) return 0;
    return 1;
}

int cond12(int n,int k)
{
    if(n<k) return 0;
    return cond10(n-k,k);

}
```