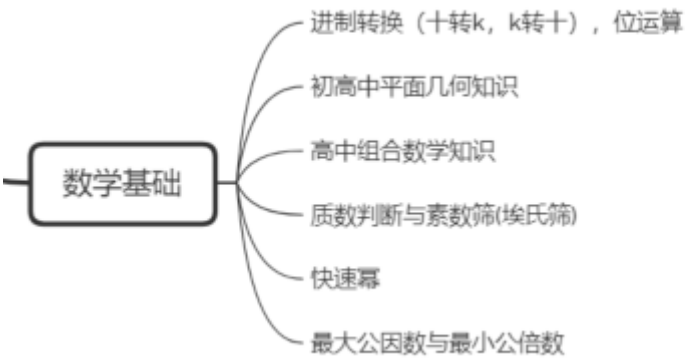
蓝桥杯十天冲刺省一


数学基础
进制转换（十转k，k转十），位运算
初高中平面几何知识
高中组合数学知识
质数判断与素数筛(埃氏筛)
快速幂
最大公因数与最小公倍数

Day-4 数学基础

进制转换

十进制转K进制

十进制

- 十进制是 `Decimal`,简写为 `D`
- 都是由 0 到 9 这十个数字组成

二进制

- 二进制是 `Binary`，简写为 `B`
- 是由 0 和 1 两个数字组成

八进制

- 八进制是 `Octal`，简写为 `O`
- 由数字 0 到 7 这八个数字组成，为了区分与其它进制的数字区别，都是以数字 0 开始

十六进制

- 十六进制是 `Hexadecimal`，简写为 `H`
- 表示方式为 `0x` 开头

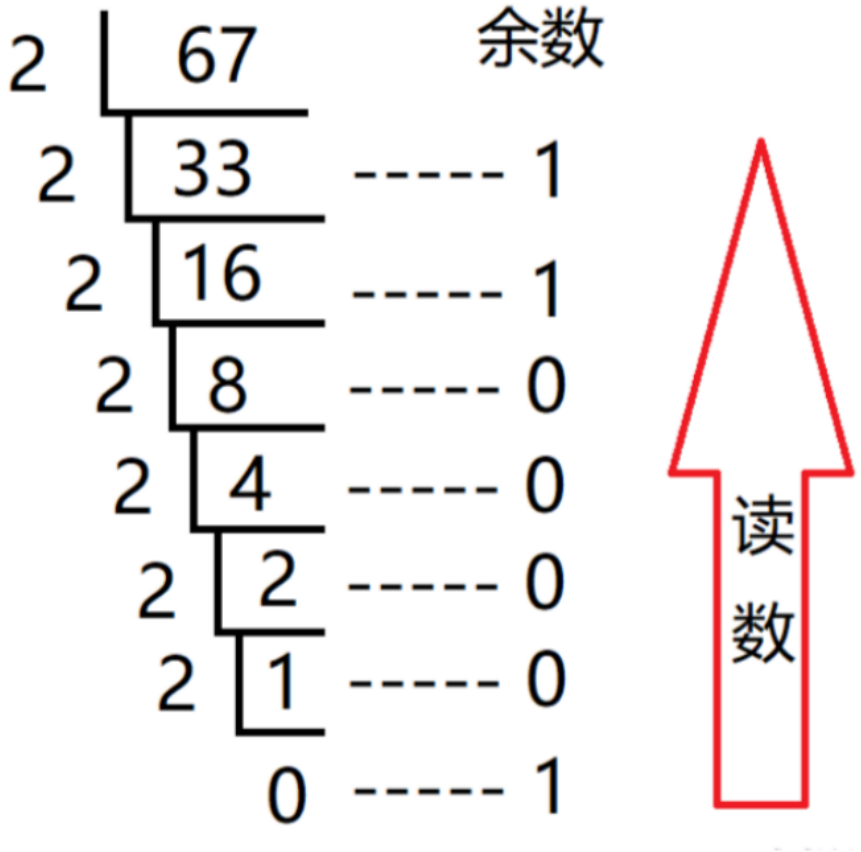
计数到 `F` 之后，在增加 1 个，就进位
由 0 − 9 和 `A-F` 组成，因为字母 `A,B,C,D,E,F` 分别表示数字 10 到 15

十进制转二进制

十进制数除以 2，余数为权位上的数，得到商继续除以 2，直到商为 0 停止，然后反向取余数

例如:将十进制数 67 转换为二进制数

将 67 除以 2 得商 33 ，余数为 1。将商 33 作为第二次的被除数，以此类推，直到商为 0。



所以我们思考一下不难知道，十进制转 K 进制的过程也和十进制转二进制一样

十进制转K进制

```cpp
#include <bits/stdc++.h> // 包含标准头文件
using namespace std;

long long s, base; // 定义两个长整型变量，s 为要转换的数，base 为进制
string p = "0123456789ABCDEF"; // 定义字符串 p，存储 16 进制数的所有可能字符
```

```
string ans; // 定义字符串 ans，用于存储转换后的结果

int main() {
    cin >> s >> base; // 输入要转换的数 s 和进制 base
    while (s) { // 当 s 不为 0 时循环
        ans.push_back(p[s % base]); // 将 s 对 base 取模的结果作为索引，将对应的字符添加到 ans 末尾
        s /= base; // 将 s 除以 base，更新 s 的值
    }
    reverse(ans.begin(), ans.end()); // 将 ans 反转，因为从末尾开始计算的结果需要反转才能得到正确的结果
    cout << ans; // 输出转换后的结果
    return 0; // 返回0，表示程序正常结束
}
```

K进制转十进制

方法:把 K 进制数按权展开、相加即得十进制

数例:用 1001 分别以二进制、八进制、十六进制转成十进制

二进制 1001 转十进制

$1*2^0+0*2^1+0*2^2+1*2^3$

八进制 1001 转十进制

$1*8^0+0*8^1+0*8^2+1*8^3$

十六进制 1001 转十进制

$1*16^0+0*16^1+0*16^2+1*16^3$

K进制转十进制

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

using namespace std;

int main() {
    string s;
```

```
    long long base = 0, ans = 0, k = 0; // 初始化所有变量

    // 输入字符串s和进制base
    cin >> s >> base;

    // 从字符串末尾开始遍历
    for(int i = s.size() - 1; i >= 0; i--) {
        // 如果字符为大写字母
        if(s[i] >= 'A') {
            ans += (s[i] - 'A' + 10) * pow(base, k++); // 更新
结果
        } else {
            ans += (s[i] - '0') * pow(base, k++); // 更新结果
        }
    }

    // 输出结果
    cout << ans;

    return 0;
}
```

练习题

Base62（ICPC2019银川）

快速幂

概念

快速幂算法，是一种快速求解幂值 a^b 的方法，快速幂算法一般不会独立使用，通常配合取余运算法则来使用，让你求 $a^b\%p$。

取余运算

- $(a+b)\%p=(a\%p+b\%p)\%p$
- $(a-b)\%p=(a\%p-b\%p)\%p$
- $(a*b)\%p=(a\%p*b\%p)\%p$

举例

让你求 $a\%p$ 的值，其中 a,b,p 是整数，且 $0<a,p<10^9$，$0<b<10^{18}$

暴力解法：考虑用循环直接计算 a^b 的值，最后将其对 p 取模

```
#define ll long long
ll ans = 1;
for(ll i = 1;i <= b;i++)
{
    ans *= a;
}
cout<< ans % p;
```

缺陷

1. 时间复杂度 $O(b)=O(10^{18})$
2. `ans` 的值会溢出

快速幂原理

- 对于线性求解的问题，如果需要优化，很多时候可以考虑用树形结构或分治思想，可将时间复杂度从 $O(n)$ 降低到 $O(logn)$
  我们考虑手算 3^{10} 如何实现
  $3^{10}=(3^2)^5=9^5=9*9^4=9*81^2=9*(6561)^1$
- 底数平方当指数为偶数时，我们在不改变该数大小的前提下可以直接将指数除以 2，当指数为奇数时，我们在不改变该数大小的前提下,可以先将底数拿一个出来，然后指数变成了偶数，转换成了指数为偶数进行处理。
- 分析时间复杂度
  每次将指数 b 除以 2，即 $2^n=b$，所以时间复杂度为 $O(logb)$ 当 $b=10^{18}$ 时，时间复杂度为 $log_2 10^{18}=59.79$

```
#define ll long long

// 求 a 的 b 次方对 p 取模的值
ll qpow(ll a, ll b, ll p) {
    ll res = 1 % p; // res 的初值为 1 % p，为什么这么写，见后面注释
    while (b) { // 当指数不为 0
        if (b % 2 == 1) { // 如果指数为奇数，就把结果乘上 a 的当前次方的值
            res = (res % p) * (a % p) % p;
        }
        a = (a % p) * (a % p) % p; // 底数平方
        b /= 2; // 指数除 2
    }
    return res; // 返回答案
}
```

解释

```
ll res = 1 % p
```

因为当 b 等于 0 时，后面的 `while` 循环进不去，这时候如果 $p=1$，加上 $\%p$的话，res 等于 0，不加上 $\%p$ 的话，res 等于 1。而实际情况也应该是 0，所以要加上 $\%p$ 更加保险

快速幂

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

typedef long long ll;

ll qmi(ll a, ll k, ll p)
{
    ll res = 1; //初始为1
    while(k)  // 对k进行二进制化，从低位到高位
    {
        if(k & 1) res = res * a % p; //如果k的二进制末位为1，则乘上当前的a
```

```
        k >>= 1;
        a = a * a % p; //更新a。a依次为a^{2^0},a^{2^1},a^{2^2},...,a^{2^logb}
    }
    return res;
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    ll n, a, k, p;
    cin >> n;
    while(n--)
    {
        cin >> a >> k >> p;
        cout << qmi(a, k, p) << "\n";
    }
    return 0;
}
```

练习题

小美与数组（2024美团技术岗春招笔试）

初高中平面几何

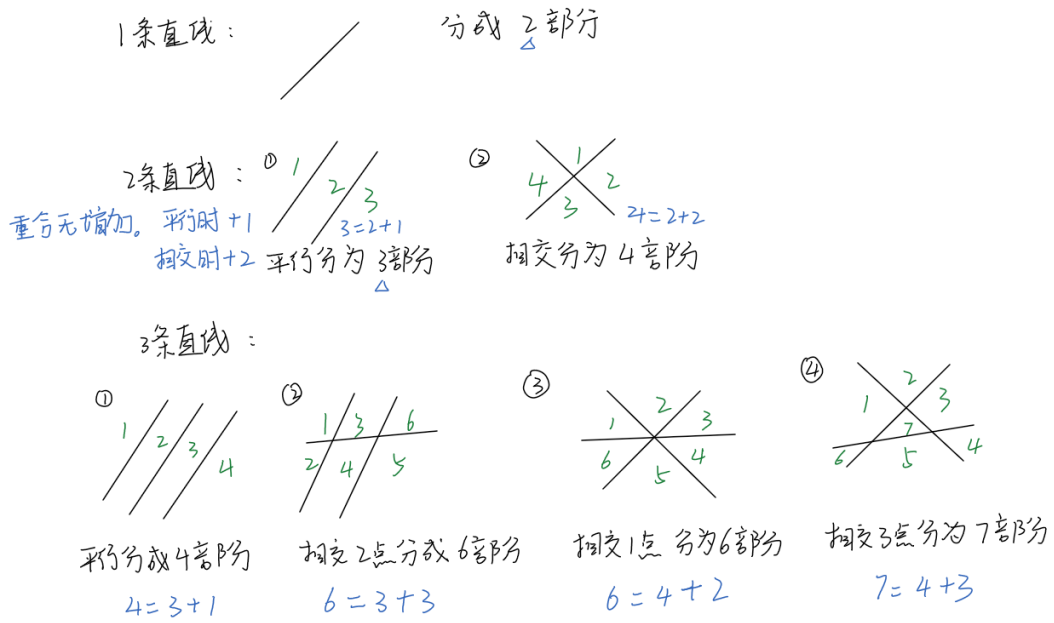
这一部分只是利用代码去解决一些初高中的数学题目，以提供例题为主

螺旋折线（蓝桥杯C/C++2018B组省赛）

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    long long  x, y,t;
    long long dis;
```

```
    cin >> x >> y;
    if (x >= 0 && y >= 0) //第一象限
    {
        t = max(x, y);
        dis = 4 * t * t + x - y;
    }
    else if (x >= 0 && y <= 0) //第四象限
    {
        t = max(x, abs(y));
        dis = 2 * t * (2 * t + 1) + abs(y) - x;
    }
    else if (x <= 0 && y <= 0) //第三象限
    {
        t = max(abs(x), abs(y) + 1);
        dis = (t + t - 1) * (t + t - 1) + abs(x) - abs(y) - 1;
    }
    else // 第二象限
    {
        t = max(abs(x), y);
        dis = 2 * t * (2 * t - 1) - abs(x) + y;
    }

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

平面切分（蓝桥杯C/C++2020B组省赛第二场）

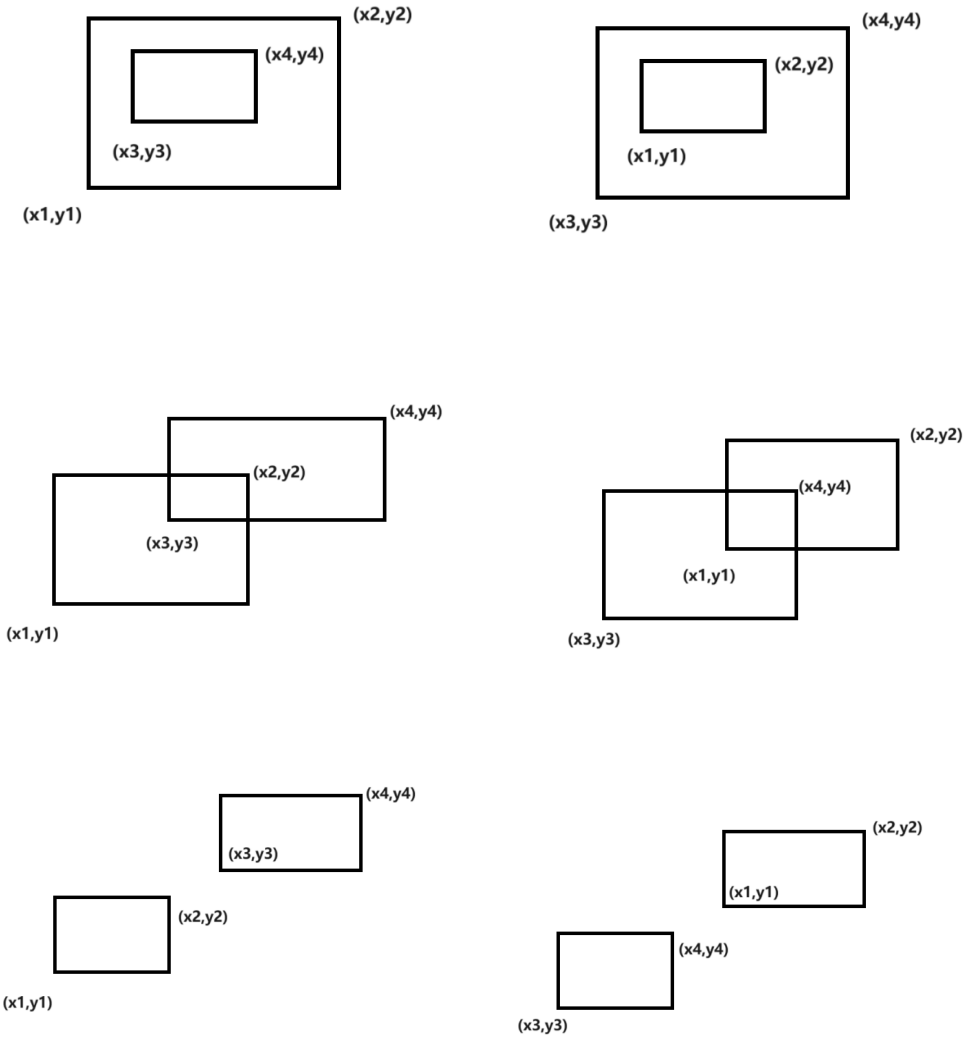


（上图出自acwing题解，作者saye）

```cpp
#include <algorithm>
#include <iostream>
#include <set>
#include <vector>
#define mp make_pair
using namespace std;
using ll = long long;
using pii = pair<double, double>;

const int N = 1e6 + 7;
const int INF = 0x3f3f3f3f;
const ll MOD = 1e9 + 7;

int n;
set<pii> s1;
vector<pii> v1;

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
```

```
    cin >> n;
    for (int i = 1; i <= n; i++) {
        double a, b;
        cin >> a >> b;
        s1.insert({a, b});
    }
    for (const auto i : s1) {
        v1.push_back(i);
    }

    // 一条线能分出两个部分
    ll ans = 2;
    for (int i = 1; i < v1.size(); i++) {
        set<pii> s2;
        for (int j = i - 1; j >= 0; j--) {
            double k1 = v1[i].first;
            double k2 = v1[j].first;
            double b1 = v1[i].second;
            double b2 = v1[j].second;
            if (k1 == k2) {
                // 平行，无交点
                continue;
            }
            // 求交点
            double x = (b2 - b1) / (k1 - k2);
            double y = k2 * x + b2;
            s2.insert({x, y});
        }
        // 每多n个交点，就多n+1个部分
        ans += s2.size() + 1;
    }
    cout << ans << "\n";
    return 0;
}
```

矩形总面积（蓝桥杯Java2023B组省赛）



（以上图片出自acwing题解，作者L-China）

我的视频题解

矩形总面积-2023蓝桥杯JavaB组省赛*哔哩哔哩*bilibili

代码

```
import java.util.Scanner;

public class Main {
```

```
    // 计算矩形面积的方法
    static long getSum(long x1, long y1, long x2, long y2) {
        return (x2 - x1) * (y2 - y1); // 返回矩形面积
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // 读取坐标值
        long x1 = scanner.nextLong(), y1 = scanner.nextLong();
        long x2 = scanner.nextLong(), y2 = scanner.nextLong();
        long x3 = scanner.nextLong(), y3 = scanner.nextLong();
        long x4 = scanner.nextLong(), y4 = scanner.nextLong();

        long sum1 = getSum(x1, y1, x2, y2) + getSum(x3, y3, x4,
y4); // 计算两个矩形总面积
        long sum2 = 0; // 初始化重叠部分面积
        long x_1, y_1, x_2, y_2; // 定义重叠矩形的坐标

        // 判断是否有重叠部分
        if (x3 < x2 && x4 > x1 && y3 < y2 && y4 > y1) {
            x_1 = Math.max(x1, x3); // 计算重叠部分左下角横坐标
            y_1 = Math.max(y1, y3); // 计算重叠部分左下角纵坐标
            x_2 = Math.min(x2, x4); // 计算重叠部分右上角横坐标
            y_2 = Math.min(y2, y4); // 计算重叠部分右上角纵坐标
            sum2 = getSum(x_1, y_1, x_2, y_2); // 计算重叠部分面
积
        }

        System.out.print(sum1 - sum2); // 输出两个矩形总面积减去
重叠部分的面积
    }
}
```

数论模数数学知识

只要遇到＋ － ×这三个运算符组成的算式，便可以在运算途中进行取模，以防止运算中途数字超过 `int` 或者 `long long` 的范围。

证明（选看）

(1)同余式可以逐项相加。 [2]

若 $a_1 \equiv b_1 \,(mod\ m)$, $a_2 \equiv b_2 \,(mod\ m)$, $\ldots$, $a_n \equiv b_n \,(mod\ m)$ ，则

$$a_1 + a_2 + \ldots + a_n \equiv (b_1 + b_2 + \ldots + b_n) \,(mod\ m) .$$

(2)同余式一边的数可以移到另一边，只要改变符号就可以了。

若 $a + b \equiv c \,(mod\ m)$ ，则 $a \equiv c - b \,(mod\ m)$ 。

(3)同余式的每一边都可以增加或减去模的任意倍数。

若 $a \equiv b \,(mod\ m)$ ，则 $a \mp km \equiv b \,(mod\ m)$ 。

(4)同余式可以逐项相乘。

若 $a_1 \equiv b_1 \,(mod\ m)$, $a_2 \equiv b_2 \,(mod\ m)$, $\ldots$, $a_n \equiv b_n \,(mod\ m)$ ，则 $a_1 a_2 \ldots a_n \equiv b_1 b_2 \ldots b_n \,(mod\ m)$ 。

(5)我们可以将性质(1)(4)推广成以下的情形。

①若 $A \equiv B \,(mod\ m)$, $x_1 \equiv y_1 \,(mod\ m)$, $x_2 \equiv y_2 \,(mod\ m)$, $\ldots$, $x_k \equiv y_k \,(mod\ m)$, 则

$$\sum A x_1^{a_1} x_2^{a_2} \ldots x_k^{a_k} \equiv \sum B y_1^{a_1} y_2^{a_2} \ldots y_k^{a_k} \,(mod\ m) .$$

②若 $a_0 \equiv b_0 \,(mod\ m)$, $a_1 \equiv b_1 \,(mod\ m)$, $\ldots$, $a_n \equiv b_n \,(mod\ m)$ ，则

$$a_0 x^n + a_1 x^{n-1} + \ldots + a_n \equiv \left(b_0 x^n + b_1 x^{n-1} + \ldots + b_n\right) (mod\ m) .$$

八次求和（蓝桥杯JavaB组2020年省赛第一场）

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // 读取输入的整数n
        int n = sc.nextInt();
        // 用于存储总和的变量，初始化为0
        long sum = 0L;
        // 循环计算每个i的值
        for (int i = 1; i <= n; i++) {
            // 将i转换为长整型，防止整型溢出
            long s1 = i * 1L;
            // 对s1进行三次幂运算，每次取余以防止数值过大
```

```
            for (int j = 1; j <= 3; j++) {
                s1 = (s1 * s1) % 123456789L;
            }
            // 将s1累加到总和中，并取余
            sum += s1;
            sum %= 123456789L;
        }
        // 输出总和
        System.out.print(sum);
    }
}
```

> ⚠ **Warning**
>
> 运算过程中如果出现负数的取模，最终答案需要经过如下的取模变换
>
> ```
> ans = (ans % MOD + MOD) % MOD
> ```

高中组合数学知识

排列问题

排列： 从 n 个不同元素中取出 $m(m \leq n)$ 个元素，按照一定的顺序排成一列，叫做从 n 个不同元素中取出 m 个元素的一个排列。

排列数：从 n 个不同元素中取出 $m(m \leq n)$ 个元素的所有不同排列的个数，叫做从 n 个不同元素中取出 m 个元素的排列数，记作 A_n^m。

计算公式：

$$A_n^m = \frac{n!}{(n-m)!}$$
$$= n(n-1)(n-2)\cdots(n-m+1)$$

证明： 利用乘法原理，我们将完成这件事情分成 m 步，第一步从 n 个元素中选一个元素，有 n 种选法，第二步从剩下 $(n-1)$ 个元素中选一个元素，有 $(n-1)$ 种选法，第三步从 $(n-2)$ 个元素中选一个元素，有 $(n-2)$ 种选法，……，选第 m 个元素时，有 $(n-m+1)$ 种选法。因为在选这些元素时，相互独立，因此，利用乘法原理可知，总方案数为：$n(n-1)(n-2)\cdots(n-m+1)$，我们把这个式子记作 A_n^m。

其他性质：

- $A_n^n = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1 = n!$
- $A_n^0 = 0! = 1$

> *例子：10 个小朋友，要从中挑出 3 个小朋友让他们站成一排，有多少种站法？*
>
> *答案：* $A_{10}^3 = \frac{10!}{7!} = 10 \times 9 \times 8 = 720$

组合问题

组合：从 n 个不同元素中取出 $m(m \leq n)$ 个元素合成一组，叫做从 n 个不同元素中取出 m 个元素的一个组合。

组合数：从 n 个不同元素中取出 $m(m \leq n)$ 个元素的所有不同组合的个数，叫做从 n 个不同元素中取出 m 个元素的组合数，记作 C_n^m。

计算公式：

$$C_n^m = \frac{A_n^m}{A_m^m} = \frac{n!}{(n-m)! \times m!}$$
$$= \frac{n(n-1)(n-2)\cdots(n-m+1)}{m!}$$

证明：对于上面的排列问题，我们从另外一个角度去做问题的拆分：首先，我们不要求顺序地选出 m 个元素出来（组合问题），然后再对这 m 个元素进行排列，那么得到的方案数应该仍旧是 A_n^m。我们把上面“不要求顺序地选出 m 个元素出来”这么一个组合问题的方案数记作 C_n^m，对 m 个元素进行排列的方案数为 $A_m^m = m!$，因此有：

$$A_n^m = C_n^m \times m!$$
$$\Rightarrow C_n^m = \frac{A_n^m}{m!} = \frac{n!}{(n-m)! \times m!}$$

其他性质：

- $C_n^0 = 1$：一个都不选的方案数是 1
- 互补性质：$C_n^m = C_n^{n-m}$：选择一些元素组成一组等价于选择其他的元素组成另外一组

- 杨辉三角：$C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$：由分类计数（加法原理）可知，如果第 n 个元素不选择，那 m 个元素只能从前 $1 \sim n-1$ 中选择，方案数为 C_{n-1}^m，如果选择第 n 个元素，那方案数为 C_{n-1}^{m-1}，相加即可。观察该等式，发现它是杨辉三角（数字三角形）的递推式
- 二项式定理：$(x+y)^n = \sum_{k=0}^n C_n^k x^k y^{n-k}$，从 n 个 x 里选 k 个（同时也相当于 n 个 y 里选 $n-k$ 个）有 C_n^k 种方法。特别的，$2^n = (1+1)^n = \sum_{k=0}^n C_n^k$

计算 $(a+b)^n$ 所有项系数的方法

根据杨辉三角：$C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$ 公式，我们可以在 $O(n^2)$ 的时间内递推计算出所有项的系数。但是如果我们只是要计算杨辉三角某一行的话，就有点浪费时间了，此时我们可以有一个 (O(n)) 计算的方法：

$C_n^k = \frac{n-k+1}{k} C_n^{k-1}$

尽管上式的“实际意义”不是很明显，却很容易用组合数公式来证明，并且给我们提供了一个线性递推计算 C_n^i 的方法。参考代码如下：

```
C[0] = 1;
for(int i = 1; i <= n; i++) C[i] = C[i - 1] * (n - i + 1) / i;
```

注意，应该先乘后除，因为 $C[i-1]/i$ 可能不是整数。但这样一来增加了溢出的可能性。——即使最后结果在 int 或 long long 范围之内，乘法也可能溢出。如果担心这样的情况出现，可以先约分，不过一般来说是不必要的。

> *例子：10 个小朋友，要从中挑出 3 个小朋友组成一组，有多少种选法？*
>
> *答案：* $C_{10}^3 = \frac{10!}{7!3!} = \frac{10 \times 9 \times 8}{3 \times 2 \times 1} = 120$

数组分割（蓝桥杯Java2023B组省赛）

前置知识

- 二项式定理

$(x+y)^n = \sum_{k=0}^n C_n^k x^k y^{n-k}$

- 取模运算

```
(a + b) % p = (a % p + b % p) % p
(a - b) % p = (a % p - b % p + p) % p
(a * b) % p = (a % p * b % p) % p
a ^ b % p = ((a % p) ^ b) % p
```

题目思路

- n 个数要分成两个集合 S_1 和 S_2 并且两个集合各自的元素之和都为偶数，我们需要先思考如何凑出和为偶数的情况：
  - 偶数个偶数的和为偶数
  - 偶数个奇数的和为偶数

  所以我们在把 n 个数划分为两个集合，里面的元素必须满足上面两种情况之一，我们需要先统计 n 个数中奇数和偶数的数量，如果奇数个数为奇数个一定无法满足两个集合元素之和为偶数的条件。
- 知道集合中的元素必须要满足什么性质之后我们需要计算出满足条件的情况数量，设偶数个数为 a，奇数个数为 b：
  - 偶数个偶数的情况数量(设情况数量为 S_1)：

    $S_1 = C_a^0 + C_a^1 + \ldots + C_a^a = 2^a$

    为什么组和数最后和为 2^a，我们需要借助二项式定理 $(x+y)^n = \sum_{k=0}^{n} C_n^k x^k y^{n-k}$

    我们假设 $a = 1$ 和 $b = 1$ 可得到 $(1+1)^n = C_n^0 + C_n^1 + \ldots + C_n^n$
  - 偶数个奇数的和为偶数(设情况数量为 S_2)

    $S_2 = C_b^0 + C_b^2 + \ldots + C_b^b = 2^{b-1}$ （奇数必须两个两个的取）

    为什么会有这样一个等式，我们还是要借助二项式定理

    假设 $a = 1$ 和 $b = 1$ 可得到 $(1+1)^n = C_n^0 + C_n^1 + \ldots + C_n^n \ (1)$

    假设 $a = 1$，$b = -1$ 并且 n 为偶数可得到 $0 = C_n^0 - C_n^1 + C_n^2 - C_n^3 \ldots + C_n^n \ (2)$

    我们将 $(1) + (2)$ 可得到 $C_n^0 + C_n^2 + \ldots + C_n^2 = 2^{n-1}$
- 最后我们再把子集个数分成以下三种情况：
  - 奇数个奇数，可能的情况个数为 0
  - 奇数个数为 0，可能的情况个数为 2^a
  - 奇数个数为偶数个并且不等于 0，可能的情况个数为 2^{a+b-1}

- 本题 $n \leq 1000$ 且不超过 10 组输入，所以即使不用快速幂也不会超时,最后不要忘了在运算过程中加上取模。

我的视频题解

数组分割-2023蓝桥杯JavaB组省赛*哔哩哔哩*bilibili

代码

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

const int mod = 1e9 + 7;

int main() {
    int t;
    cin >> t;
    while(t--) {
        int n;
        cin >> n;
        int a = 0,b = 0;
        for(int i = 1;i <= n;i++) {
            int t;
            cin >> t;
            if(t % 2 == 0) {
                a++;
            }
            else {
                b++;
            }
        }
        long long res = 1;
        if(b % 2 != 0) {//奇数个奇数
            cout << 0 << endl;
        }
        else {
            if(b == 0) {//奇数个数为0
                for(int i = 1;i <= a;i++) {
                    res = ((res % mod) * (2 % mod)) % mod;
```

```
                }
            }
            else {
                for(int i = 1;i <= a + b - 1;i++) {
                    res = ((res % mod) * (2 % mod)) % mod;
                }
            }
            cout << res << endl;
        }
    }
}
```

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int mod = 1000000007;
        int t = scanner.nextInt();
        while (t-- > 0) {
            int n = scanner.nextInt();
            int a = 0, b = 0;
            for (int i = 1; i <= n; i++) {
                int num = scanner.nextInt();
                if (num % 2 == 0) {
                    a++;
                } else {
                    b++;
                }
            }
            long res = 1;
            if (b % 2 != 0) { // 奇数个奇数
                System.out.println(0);
            } else {
                if (b == 0) { // 奇数个数为0
                    for (int i = 1; i <= a; i++) {
                        res = ((res % mod) * (2 % mod)) % mod;
                    }
                } else {
```

```
                    for (int i = 1; i <= a + b - 1; i++) {
                        res = ((res % mod) * (2 % mod)) % mod;
                    }
                }
                System.out.println(res);
            }
        }
    }
}
```

质数判断与素数筛

质数判断

试除法

我们这里直接给出质数的定义，如果整数 N 满足下面两个条件，我们就可以判断 N 是一个质数：

（1）这个正整数 $N >= 2$；

（2）这个正整数 N 只能被 1 或本身整除。

我们现在已经了解了什么是质数，判断方法其实很简单，我们只需要把这个数字 N 对 1 到 N 的每个数字都进行取余运算，然后看看有几个算式的余数是 0 就可以了，时间复杂度是 $O(n)$ 的。

```
// 试除法：判断一个整数 i 是不是素数，是则返回 true，否则返回 false
bool isprime(int n){
    if(n < 2)
        return false;
    for(int i = 2; i <= n; i++)
        if(n % i == 0)
            return false;
    return true;
}
```

根号优化

首先第一个叫做因子成对原理，一个大于一的自然数，它的因子一定会是成对出现的，比如以 100 为例，可以看到它有一个因子是 1，另外一个因子一定可以找到是 100，有个因子 2 一定可以找到另外的一个因子是 50，一个因子是 4 一定可以找到另外一个因子是 25，这个就叫做因子成对原理，它是成对出现的。

推论：因为因子是成对出现的，所以说我们只要找到部分的因子，就能够由这个部分的因子推导出其余的因子。以 100 来说，我们知道因子 1 是 $1, 2, 4, 5, 10$ 这五个因子，就可以推导出他另外的因五个因子，$100, 50, 25, 20, 10$。也就是说，我们的要求这个 100 的所有的因子，我们只要求到他的一部分因子，就可以求到他所有的因子。

因子1	1	2	4	5	10
因子2	100	50	25	20	10

推论：所以我们在质数判断的时候，并不需要从 1 运行到这个数字本身，而只需从 1 运行到它的因子 1 结束的地方就可以了。比如以这 100 来说，我们从只需要从 1 运行到 10 就可以了。这 10 是怎么来的呢？其实这个 10 就是我们所要求的数字的 100 的平方根，也就是$sqrt(100) = 10$。

结论：一个整数 n 的因子对(除了 1 和他本身），一定可以在 2 到 $sqrt(n)$ 的范围内求解完毕。

```
// 试除法：判断一个整数 i 是不是素数，是则返回 true，否则返回 false
bool isprime(int i){
    if(i < 2)
        return false;
    for(int j = 2; j <= sqrt(i); j++)
        if(i % j == 0)
            return false;
    return true;
}
```

但我们更常用的是变开根为乘法的代码写法，代码见题目

判断质数

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

int main()
{
    bool flag = 0;
    long long n;
    cin >> n;
    for(int i = 2; i * i <= n; i++)
    {
        if(n % i == 0)
        {
            flag = 1;
            break;
        }
    }
    if(flag == 0 && n >= 2)//保证n要大于等于2
    {
        cout << "YES";
    }
    else
    {
        if(n == 1) cout << 1;//特判1的情况
        for(int i = 2; i * i <= n; i++)
        {
            if(n % i == 0)//从小往大找因子，第一个因子若是i，那么最大的就是n/i
            {
                cout << n / i;
                break;
            }
        }
    }
}
```

练习题

这个美术社大有问题！

素数筛

埃氏筛

埃氏筛的时间复杂度为 $O(n*log(logn))$，而线性筛的时间复杂度为 $O(n)$，一般情况下来说埃氏筛就已经满足大多竞赛题的需求，所以我们这里为了降低大家复习难度，大家只需要会比较好理解的埃氏筛就行。

算法思想:一个素数的整数倍一定是合数

步骤
1.找素数
2.筛倍数
3.移下标

核心
合数被约数多次筛掉

```
int primes[N], cnt; // primes[] 存储所有素数
bool f[N]; // f[i] 存储 i 是否被筛掉，true 表示被筛掉，false 表示未被筛掉

// 筛选出小于等于 n 的所有素数
void get_primes(int n) {
    for (int i = 2; i <= n; i++) {
        if (f[i])
            continue; // 如果 i 已经被筛掉，则跳过
```

```
            primes[++cnt] = i; // 将当前素数 i 记录到 primes[] 数组中
            for (int j = i + i; j <= n; j += i) {
                f[j] = true; // 将 i 的倍数标记为已筛掉
            }
        }
    }
```

线性筛素数

```
//虽然是埃氏写法，但过不了这道题，数据卡的很死，有兴趣的可以自己学学线性筛
#include<bits/stdc++.h>
using namespace std;

const int N = 1e8 + 5; // 定义最大范围
bool f[N]; // 标记数组，标记是否为合数
int p[6000010], cnt; // p[] 存储素数，cnt 记录素数的个数

// 筛选出小于等于 n 的所有素数
void prime(int n) {
    for (int i = 2; i <= n; i++) {
        if (f[i] == false) {
            p[++cnt] = i; // 将当前素数记录到 p[] 数组中
            for (long long j = (long long)i + i; j <= n; j += i) {
                f[j] = true; // 将 i 的倍数标记为合数
            }
        }
    }
}

int main() {
    ios::sync_with_stdio(0); // 关闭 C++ 输入输出同步
    cin.tie(0); // 解除 cin 与 cout 的绑定，加快输入速度
    cout.tie(0);
    int n, q, k;
    cin >> n >> q; // 输入范围 n 和查询次数 q
    prime(n); // 调用函数筛选素数
    while (q--) { // 循环处理每一个查询
```

```
        cin >> k; // 输入查询的索引
        cout << p[k] << '\n'; // 输出第 k 个素数
    }
    return 0; // 返回0，表示程序正常结束
}
```

线性筛（选学）

将前面埃氏筛的代码弄清楚后，我们可以发现，对于埃氏筛来说，如果要将 30 这个数标记为合数，会存在重复标记的 5 乘 6，而这明显会浪费时间，比如 2 乘 15，3 乘 10。

- 算数基本定理
  任何一个大于 1 的自然数 n，如果 n 不是素数，那么 n 可以唯一分解成有限个素数的乘积
- 最小质因数

  一个合数分解出的有限个素数乘积中，最小的那个素数。也就是说，每一个合数，都有它对应的一个最小质因子，而我们只通过这个最小质因子将其标记为素数或合数，这样就避免了重复标记

以这个为基础，线性筛诞生了。线性筛是在埃氏筛的基础上，去掉重复标记的部分，使得每一个数只会被它的最小质因子标记一次。

```
int primes[N], cnt; // primes[ ]存储所有素数
bool f[N]; // f[i]存储i是否被筛掉,true表示被筛掉，false表示未被筛掉

void get_primes(int n) {
    for(int i = 2; i <= n; i++) {
        if(!f[i])
            primes[++cnt] = i; // 将素数添加到数组中
        for(int j = 1; primes[j] <= n / i; j++) {
            f[i * primes[j]] = true; // 标记被筛掉的数
            if(i % primes[j] == 0)
                break;
        }
    }
}
```

解释

```
if(i%primes[j]==0)  break;
```

为什么当 i 是 $prime[j]$ 的整数倍时就 $break$ 了呢，我们可以回头想想，我们要改进埃氏筛，只通过这个数的最小质因数将数字标记它，那么我们就要加上一句话来判断:即将被筛掉的数是否通过其最小质因数来筛掉的。

所以上面题目的正解是

线性筛素数正解

```
#include <iostream>
#include <algorithm>
using namespace std;

bool f[(int)1e8 + 10];
int prime[(int)6e6 + 10], cnt; //1亿以内有不到600w个质数

void get_prime(int n) //线性筛
{
    for(int i = 2; i <= n; i++)
    {
        if(!f[i]) prime[++cnt] = i; //存储质数
        for(int j = 1; prime[j] * i <= n; j++)
        {
            f[prime[j] * i] = 1; //标记不是质数
            if(i % prime[j] == 0) break; //优化成O(n)
        }
    }
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0), cout.tie(0);
    int n, q, k;
    cin >> n >> q;
    get_prime(n);
```

```
    while(q--)
    {
        cin >> k;
        cout << prime[k] << "\n";
    }
    return 0;
}
```

最大公因数与最小公倍数

最大公因数

最大公因数，常缩写为 gcd 。

一组整数的公因数，是指同时是这组数中每一个数的因数的数。

一组整数的最大公因数，是指所有公约数里面最大的一个。

欧几里德算法:辗转相除法求最大公因数辗转相除法是一种算法，也称欧几里德算法(Euclid)。如果我们已知两个数 a 和 b，如何求出二者的最大公因数呢?可以用如下等式去递归求解：$gcd(a,b) = gcd(b,a\%b)$。

```
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}
```

当然如果记不住也不要紧，C++有自带的 __gcd(a,b) 注意这里 gcd 前面是两个 _ ，而且变量 a 和 b 数据类型必须相同，不能 a 是 int 型，b 是 long long 型。

最大公因数

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

// 求最大公约数
long long gcd(long long a, long long b) {
    if (b == 0) {
        return a;
```

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

int main() {
    long long a, b;
    // 输入两个数
    cin >> a >> b;
    // 输出它们的最大公约数
    cout << gcd(a, b);
    return 0;
}
```

```
#include<bits/stdc++.h> // 包含标准头文件
using namespace std;

int main() {
    long long int a, b; // 定义两个长整型变量
    cin >> a >> b; // 输入两个数
    cout << __gcd(a, b); // 输出它们的最大公约数
    return 0; // 返回0，表示程序正常结束
}
```

等差数列（蓝桥杯C/C++B组2019年省赛）

```
#include <cstdio>
#include <algorithm>
#include <cstring>

using namespace std;

const int N = 100010;

int a[N];

// 求最大公约数
int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
```

```
}

int main() {
    int n;
    scanf("%d", &n);

    // 读取数组元素
    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }

    // 对数组进行排序
    sort(a, a + n);

    // 求后面各项与第一项所有差值的最大公约数
    int d = 0;
    for (int i = 1; i < n; i++) {
        d = gcd(d, a[i] - a[0]);
    }

    // 如果最大公约数为0，则所有元素相等，输出n
    if (!d)
        printf("%d\n", n);
    else
        // 否则输出（最后一个元素 - 第一个元素）/ 最大公约数 + 1
        printf("%d\n", (a[n - 1] - a[0]) / d + 1);

    return 0;
}
```

练习题

最大比例（蓝桥杯C/C++2016B组省赛）

最小公倍数

最小公倍数(Least Common Multiple, LCM)。
组整数的公倍数，是指同时是这组数中每一个数的倍数的数。0 是任意一组整数的公倍数。
组整数的最小公倍数，是指所有正的公倍数里面，最小的一个数。

只需要记住这两个公式简单来说就是 两个数的最大公约数×最小公倍数=两数相乘： $gcd(a,b)\times lcm(a,b)=a\times b$，$lcm(a,b)=a\times b/gcd(a,b)$。

最小公倍数

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

// 求最大公约数
long long gcd(long long a, long long b) {
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}

int main() {
    long long a, b;
    // 输入两个数
    cin >> a >> b;
    // 计算最小公倍数，并输出结果
    cout << a / gcd(a, b) * b; // 注意这里先乘再除可能会溢出，要先除再乘。
    return 0;
}
```