

一、一维数组

(一) 考点分析

类型	重点考点	考察频率	难度	核心知识点
数组操作	反转	25%	★★	首尾交换实现反转
	分区	20%	★★★★	双指针将数组划分为满足不同条件的区域
	删除	15%	★★	快慢指针覆盖目标元素
算法应用	前缀和	30%	★★★★	$\text{prefix}[i] = a[0] + a[1] + \dots + a[i-1]$
	二分查找	15%	★★★★	有序数组的 $O(\log n)$ 搜索
	Josephus 问题	10%	★★★★★	环形数组模拟淘汰过程
易错点	越界访问	40%	★★★★	访问 $a[-1]$ 或 $a[n]$
	浅拷贝陷阱	20%	★★	数组赋值 $b=a$ 导致共享内存
	未初始化	30%	★★	局部数组元素值随机
特殊技巧	计数数组清零	高频	★	全局数组自动初始化为 0, 局部数组需手动初始化
	环形数组遍历	高频	★★	取模实现循环下标
	位标记优化	低频	★★★★★	用 bool 数组代替 int 数组节省空间

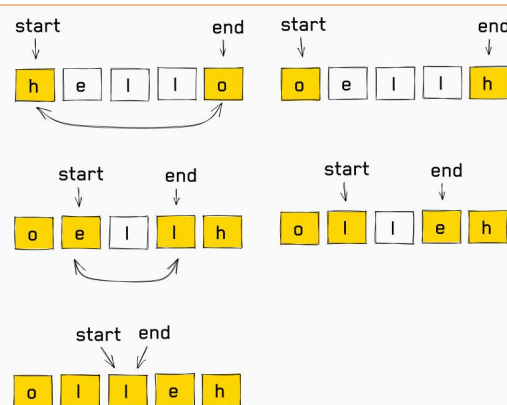
(二) 核心知识点

1. 数组操作

(1) 反转

- 核心知识点：通过**双指针交换**首尾元素实现反转。
- 黄金法则：指针移动条件为 **left < right**，每次交换后左指针右移，右指针左移。
- 易错点：
 - 循环条件写成 `left <= right` 会导致中间元素被多余交换
 - 忘记移动指针造成死循环
- 程序模板：

```
void reverseArray(int arr[], int n) {
    int left = 0, right = n - 1;
    while (left < right) {
        swap(arr[left], arr[right]);
        left++;
        right--;
    }
}
```



(2) 分区

- 核心知识点：将数组按条件（如正负数）划分为独立区域。
- 黄金法则：使用双指针扫描，左指针找不满足条件元素，右指针找满足条件元素后交换。
- 易错点：
 - 未检查 `i < j` 导致越界
 - 交换后忘记移动指针
- 程序模板：

```

void partition(int arr[], int n) {
    int i = 0, j = n - 1;
    while (i < j) {
        while (i < j && arr[i] > 0) i++; // 正数跳过
        while (i < j && arr[j] < 0) j--; // 负数跳过
        if (i < j) swap(arr[i++], arr[j--]);
    }
}

```

i	0	1	2	3	4	5	6	7	8	9
输入	5	4	-6	-11	6	-59	22	-6	1	10
输出	5	4	10	1	6	22	-59	-6	-11	-6

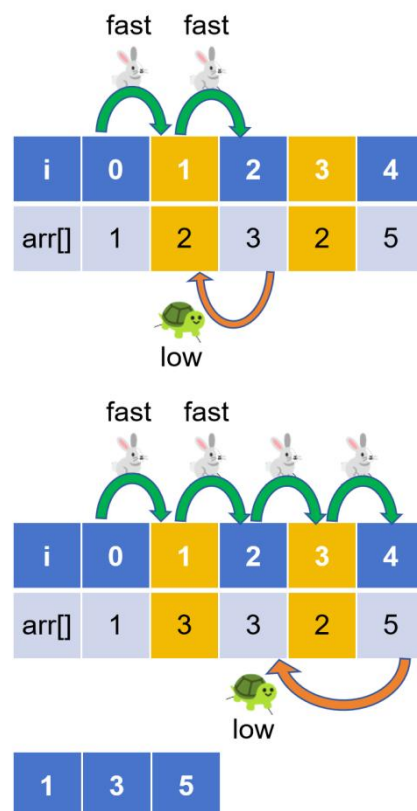
(3) 删除

- 核心知识点：使用快慢指针原地删除元素。
- 黄金法则：慢指针指向有效位置，快指针扫描保留元素。
- 易错点：
 - 未更新数组长度导致访问无效元素
 - 未处理剩余元素内容
- 程序模板：输入数据为 [1, 2, 3, 2, 5]，目标移除所有 2。

```

int removeElements(int arr[], int n, int target) {
    int slow = 0;
    for (int fast = 0; fast < n; fast++) {
        if (arr[fast] != target)
            arr[slow++] = arr[fast];
    }
    return slow; // 新长度
}

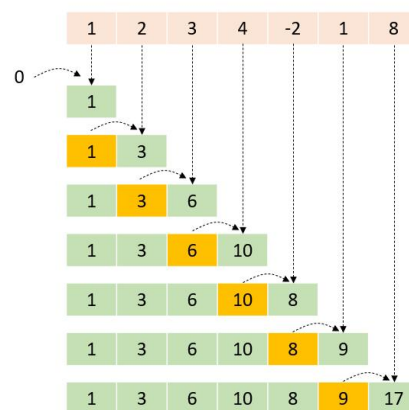
```



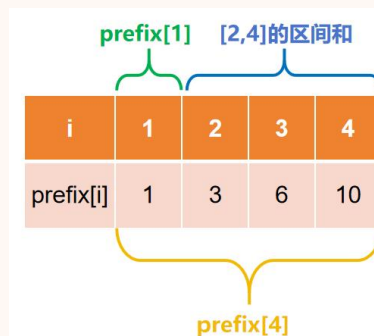
2. 算法应用

(1) 前缀和

- 核心知识点：预处理数组，实现区间和 $O(1)$ 查询。
- 黄金法则： $\text{prefix}[i] = \text{prefix}[i-1] + \text{arr}[i-1]$ 。
- 易错点：
 - 前缀和数组长度应为 $n+1$
 - 区间和计算错位： $\text{sum}[l..r] = \text{prefix}[r+1] - \text{prefix}[l]$
- 程序模板：



```
int main() {
    int arr[] = {1,2,3,4}, n = 4;
    int prefix[n+1] = {0};
    for (int i = 1; i <= n; i++)
        prefix[i] = prefix[i-1] + arr[i-1];
    cout << prefix[4] - prefix[1]; // 输出 9 (2+3+4)
}
```



(2) 二分查找

- 核心知识点：有序数组的折半搜索。
- 黄金法则：维护 $[\text{left}, \text{right}]$ 区间，循环条件 $\text{left} \leq \text{right}$ 。
- 程序模板：

```
int binarySearch(int arr[], int n, int target) {
    int left = 0, right = n - 1;
    while (left <= right) {
        int mid = left + (right - left)/2;
        if (arr[mid] == target) return mid;
        if (arr[mid] < target) left = mid + 1;
    }
}
```



```

else right = mid - 1;

}

return -1; }

```

(3) Josephus 问题

- 核心逻辑:

- 有 n 个人围成一圈，编号为 0 到 $n-1$ 。
- 从某个起点开始报数 (p 表示当前报数状态，初始为 0)。
- 每当报数到 1 时，当前人会被淘汰 ($eliminated[i] = true$)。
- 报数在 0 和 1 之间切换 ($p \wedge 1$ 实现翻转)。
- 循环直到 $c == n-1$ (即只剩 1 人未被淘汰)。

- 核心知识点：环形数组模拟淘汰过程。

- 黄金法则：用取模运算实现环形遍历。

- 高频公式:

```

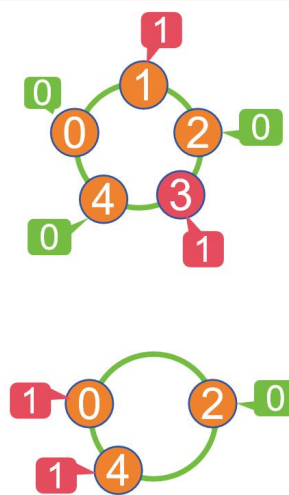
int p = 0; // 当前报数状态

for (int i = (i+1)%n; c < n-1; i = (i+1)%n) {
    if (!eliminated[i] && p == 1) {
        eliminated[i] = true;

        c++;
    }

    if (!eliminated[i])
        p ^= 1; // 翻转报数
}

```



- 程序解释:

①初始化

- p 表示当前的报数值，初始为 0，后续会在 0 和 1 之间切换（模拟“报数”）。

②循环

- 循环变量初始化 $i = (i+1)\%n$
 - 从下一个位置开始（ $\%n$ 确保循环遍历，即 i 在 0 到 $n-1$ 之间循环）。
- 循环变量终止条件 $c < n-1$
 - c 记录已淘汰的人数，当 $c == n-1$ 时停止（只剩 1 人存活）。
- 步长 $i = (i+1)\%n$
 - 每次循环 i 向后移动 1，并取模 n 保证不越界。

③淘汰逻辑

- 如果当前人未被淘汰（!eliminated[i]）且当前报数 p 为 1，则淘汰此人。
- 标记 $eliminated[i] = true$ ，并增加淘汰计数 c 。

④报数翻转

- 条件 !eliminated[i]
 - 只有当前人未被淘汰时，才切换报数 p 。
- $p \wedge = 1$ （异或操作）
 - 如果 $p=0$ ，则变为 1；如果 $p=1$ ，则变为 0（即 0 和 1 交替报数）。
- 程序模板：

```
int josephus(int n) {
    bool elim[n] = {false};
    int i = 0, p = 0, c = 0;
    while (c < n - 1) {
        if (!elim[i] && p) {
            elim[i] = true;
            c++;
        }
        if (!elim[i]) p ^= 1;
    }
}
```

```

        i = (i + 1) % n;
    }
    for (i = 0; i < n; i++)
        if (!elim[i]) return i;
}

```

3. 易错点

(1) 越界访问

- 核心问题：访问**无效索引**导致未定义行为。
- 黄金法则：始终检查索引范围 $[0, n-1]$ 。
- 高频防御：

```

// 访问前检查

if (index >= 0 && index < n) use(arr[index]);

```

- 典型错误：
 - 循环条件错误：for (int i=0; i<=n; i++)
 - 负数索引：arr[-1]
- 案例：
 - 2012 年真题中若未检查 $x[j] < x[i]$ 会导致越界。

(2) 浅拷贝陷阱

- 核心问题：直接赋值 `arr1 = arr2` **仅复制指针**。
- 黄金法则：数组必须深拷贝。
- 高频公式：

```

// 深拷贝

int newArr[n];

memcpy(newArr, arr, n*sizeof(int));

```

- 典型错误：
 - 修改拷贝数组影响原数组
 - 局部数组返回导致悬垂指针

(3) 未初始化

- 核心问题：局部数组默认值随机。
- 黄金法则：显式初始化所有元素。
- 高频公式：

```
int arr[n] = {0}; // 全部初始化为 0  
bool flags[n]={false}; // C++11 初始化
```

- 典型错误：
 - 使用未初始化元素进行运算
 - 逻辑依赖错误初始值

4. 特殊技巧

(1) 计数数组清零

- 核心场景：多次使用计数数组时高效重置。
- 黄金法则：用 `memset` 或循环清零。
- 高频公式：

```
int count[100]{};  
memset(count, 0, sizeof(count)); // 快速清零
```

- 优化点：
 - 避免重复声明数组
 - 比循环赋值更快

(2) 环形数组遍历

- 核心场景：循环队列、约瑟夫环。

- 黄金法则：索引取模实现环形访问。
- 高频公式：

```
for (int i = 0; i < 2*n; i++) {
    int idx = i % n; // 环形访问 }
```

- 典型应用：
 - 数组循环移位
 - 环形缓冲区

(3) 位标记优化 (了解)

- 核心场景：状态压缩（如八皇后）。
- 黄金法则：用位运算代替布尔数组。
- 高频公式：

```
int state = 0;
state |= (1 << i); // 标记第 i 位
bool used = state & (1 << i); // 检查标记
```

- 优势：
 - 空间效率：int 替代 bool[32]
 - 时间效率：位运算 $O(1)$

关键总结：

- 数组操作先定边界，双指针是核心技巧
- 算法应用注意预处理（前缀和）和边界条件（二分）
- 易错点防御：索引检查 + 初始化 + 深拷贝
- 特殊技巧提升效率：位运算 > 计数数组 > 暴力遍历

(三) 做题方法总结

1、理解题意与提取关键信息

首先，仔细阅读题目，提取题目中与数组相关的键信息，如：

- 数组的大小 n ；
- 数组的元素范围；
- 要求输出的值：比如最小值、最大值、和、平均值等；
- 特定的条件（比如某个元素是否满足某种条件）；
- 是否需要进行操作（如排序、查找、过滤等）。

理解题目意图后，再决定如何处理数组。

2. 手工模拟数组的变化

对于阅读程序题或完善程序题，手工模拟数组的变化是一种非常有效的方法。在纸上或心里模拟每个数组元素如何变化，尤其是在有循环或条件判断的情况下。

示例：

假设题目给出数组 $a = \{1, 2, 3, 4, 5\}$ ，要求我们计算数组中所有偶数的和。

做法：

- 开始时数组是 $\{1, 2, 3, 4, 5\}$ 。
- 手工遍历数组，每次检查元素是否为偶数。
 - $a[0] = 1$ ，不是偶数，不加入到和里。
 - $a[1] = 2$ ，是偶数，和加 2。
 - $a[2] = 3$ ，不是偶数，不加。
 - $a[3] = 4$ ，是偶数，和加 4。
 - $a[4] = 5$ ，不是偶数，不加。
- 最终和为 $2 + 4 = 6$ 。

3. 与数学知识结合，寻找规律

一维数组的题目往往涉及到一些常见的数学操作，如求和、平均值、最大最小值、元素间的关系等。这时，结合数学知识可以帮助你更快地理解题目，并找到解题的规律。

例 1：求数组元素的和

假设题目要求计算数组的和：

例子：给定数组 $a = \{1, 2, 3, 4, 5\}$ ，要求计算和。

做法：

- 手工求和： $1 + 2 + 3 + 4 + 5 = 15$ 。
- 数学技巧：如果题目有规律的数列，可以使用数学公式。例如，等差数列的和可以用公式 $S = n * (a_1 + a_n) / 2$ ，其中 n 为元素个数， a_1 为第一个元素， a_n 为最后一个元素。

例 2：查找数组中的最大最小值

例如，题目要求找出数组中的最大和最小值。

做法：

- 遍历数组时，每次与当前最大值和最小值进行比较，更新它们。
- 数学规律：利用最大值和最小值的定义：最大值是数组中最大的元素，最小值是数组中最小的元素。

示例：

给定数组 $a = \{5, 8, 3, 1, 9\}$ ，要求找最大值和最小值：

- 最大值： $\max(5, 8) = 8$ ， $\max(8, 3) = 8$ ， $\max(8, 1) = 8$ ， $\max(8, 9) = 9$ ，最终最大值为 9。
- 最小值： $\min(5, 8) = 5$ ， $\min(5, 3) = 3$ ， $\min(3, 1) = 1$ ， $\min(1, 9) = 1$ ，最终最小值为 1。

4. 注意循环与条件的影响

对于包含循环和条件判断的题目，可以通过逐步模拟每个步骤来推算结果。对于数组的遍历，常见的做法是通过条件判断来筛选、修改或者输出满足条件的元素。

示例：

给定数组 $a = \{1, 2, 3, 4, 5\}$ ，要求输出数组中大于 3 的元素。

做法：

模拟循环：检查每个元素：

- $a[0] = 1$ ，不大于 3，跳过。
- $a[1] = 2$ ，不大于 3，跳过。
- $a[2] = 3$ ，不大于 3，跳过。

- $a[3] = 4$, 大于 3, 输出 4。
- $a[4] = 5$, 大于 3, 输出 5。

输出为: 4 5。

5. 数组与数学数列的结合

对于数组题目, 特别是涉及到数列性质时, 可以利用数学数列的性质简化计算, 如等差数列、等比数列、斐波那契数列等。

示例: 等差数列

如果题目给出一个等差数列, 比如 $a = \{1, 3, 5, 7, 9\}$, 要求计算前 n 项的和。

做法:

- 数列的和可以用公式 $S = n * (a_1 + a_n) / 2$ 来计算, 其中 a_1 为首项, a_n 为末项, n 为项数。
- 这里, $a_1 = 1$, $a_n = 9$, $n = 5$, 因此 $S = 5 * (1 + 9) / 2 = 25$ 。

6. 对于不规则数据, 注意检查数组边界

在一些题目中, 可能需要特别注意数组的边界问题, 尤其是输入与输出数组时。常见的错误包括数组越界、未初始化数组元素等。

示例:

给定数组 a , 要求计算每个元素与下一个元素的差值:

做法:

- 通过 $a[i+1] - a[i]$ 来计算差值, 但需要确保数组索引不越界。
- 如 $a[i+1]$ 可能会访问越界元素, 所以应确保 $i < n - 1$ 。

(四) 课后练习

第 1 题. 2008 年第 25 题 (一维数组、循环)

阅读程序写结果:

```
#include <iostream>

using namespace std;

void func(int ary[], int n )
```

```

{
    int i = 0, j, x;
    j = n - 1;
    while (i < j)
    {
        while (i < j && ary[i] > 0) i++;
        while (i < j && ary[j] < 0) j--;
        if (i < j)
        {
            x = ary[i];
            ary[i++] = ary[j];
            ary[j--] = x;
        }
    }
}

```

i	0	1	2	3	4	5	6	7	8	9
输入	5	4	-6	-11	6	-59	22	-6	1	10

i	0	1	2	3	4	5	6	7	8	9
输出	5	4	10	1	6	22	-59	-6	-11	-6

```

int main()
{
    int a[20], i, m;
    m = 10;
    for(i = 0; i < m; i++)
    {
        cin >> a[i];
    }
    func(a, m);
    for (i = 0; i < m; i++)
        cout << a[i] << " ";
}

```

```

    cout << endl;

    return 0;

}

```

i	0	1	2	3	4	5	6	7	8	9
输入	5	4	-6	-11	6	-59	22	-6	1	10
i	0	1	2	3	4	5	6	7	8	9
输出	5	4	10	1	6	22	-59	-6	-11	-6

输入:

5 4 -6 -11 6 -59 22 -6 1 10

正确答案:

5 4 10 1 6 22 -59 -6 -11 -6

```

5 4 -6 -11 6 -59 22 -6 1 10
5 4 10 1 6 22 -59 -6 -11 -6

```

解析:

程序通过双指针实现数组分区:

- 指针初始化: i 指向数组起始位置 (0), j 指向数组末尾 (9)。
- 主循环: 当 $i < j$ 时执行:
 - 左指针移动: i 向右移动, 跳过正数, 停在负数或 0 上 (条件: $\text{ary}[i] > 0$)。
 - 右指针移动: j 向左移动, 跳过负数, 停在正数或 0 上 (条件: $\text{ary}[j] < 0$)。
 - 交换元素: 若 $i < j$, 交换 $\text{ary}[i]$ 和 $\text{ary}[j]$, 然后 $i++$, $j--$ 。
- 输入处理:
 - 初始数组: [5, 4, -6, -11, 6, -59, 22, -6, 1, 10]。
 - 逐步执行:
 - ◆ $i=0$ (正数 5、4 跳过) $\rightarrow i=2$ (-6); $j=9$ (正数 10 跳过) $\rightarrow j=9$ 。交换 $a[2](-6)$ 和 $a[9](10) \rightarrow$ 数组变为 [5,4,10,-11,6,-59,22,-6,1,-6], $i=3$, $j=8$ 。
 - ◆ $i=3$ (-11); $j=8$ (正数 1 跳过) $\rightarrow j=8$ 。交换 $a[3](-11)$ 和 $a[8](1) \rightarrow$ 数组变为 [5,4,10,1,6,-59,22,-6,-11,-6], $i=4$, $j=7$ 。
 - ◆ $i=4$ (正数 6 跳过) $\rightarrow i=5$ (-59); $j=7$ (-6 跳过) $\rightarrow j=6$ (22)。交换 $a[5](-59)$ 和 $a[6](22) \rightarrow$ 数组变为 [5,4,10,1,6,22,-59,-6,-11,-6], $i=6$, $j=5$ (循环结束)。
- 输出: 正数在前 (5,4,10,1,6,22), 负数在后 (-59,-6,-11,-6)。

第 2 题. 2011 年第 25 题 (一维数组, for 循环)

阅读程序写结果：

```
#include<iostream>

#include<cstring>

using namespace std;

const int SIZE = 100;

int main()
{
    int n,i,sum,x,a[SIZE];

    cin>>n;

    memset(a,0,sizeof(a));

    for(i=1;i<=n;i++){

        cin>>x;

        a[x]++;

    }

    i=0;

    sum=0;

    while(sum<(n/2+1)){

        i++;

        sum+=a[i];

    }

    cout<<i<<endl;

    return 0;

}
```

输入：

11

4 5 6 6 4 3 3 2 3 2 1

正确答案：

3

```
11
4 5 6 6 4 3 3 2 3 2 1
3
```

解析：

程序功能：寻找最小整数 i ，使得数组中所有 小于等于 i 的元素个数之和 $\geq n/2 + 1$ （即超过半数）。

● 输入处理：

■ $n = 11$ ，数组元素：[4,5,6,6,4,3,3,2,3,2,1]。

■ 统计频次：

◆ $a[1]=1$

◆ $a[2]=2$

◆ $a[3]=3$

◆ $a[4]=2$

◆ $a[5]=1$

◆ $a[6]=2$

● 累加计算：

■ 初始化 $i=0$ ， $sum=0$ 。

■ 循环条件： $sum < (11/2 + 1) = 6$ （整数除法）。

◆ $i=1$ ： $sum = a[1] = 1$ ($1 \leq 6$ ，继续)。

◆ $i=2$ ： $sum = 1 + a[2] = 3$ ($3 \leq 6$ ，继续)。

◆ $i=3$ ： $sum = 3 + a[3] = 6$ ($6 \geq 6$ ，退出循环)。

● 输出： $i=3$ （最小满足条件的整数）。

第 3 题. 2012 年第 27 题（一维数组，最值）

完善程序（坐标统计）：

输入 n 个整点在平面上的坐标。对于每个点，可以控制所有位于它左下方的点（即 x,y 坐标都比它小），它可以控制的点的数目称为“战斗力”。依次输出每个点的战斗力，最后输出战斗力最高的点的编号（如果战斗力并列最高，输出最大编号）。

```
#include <iostream>

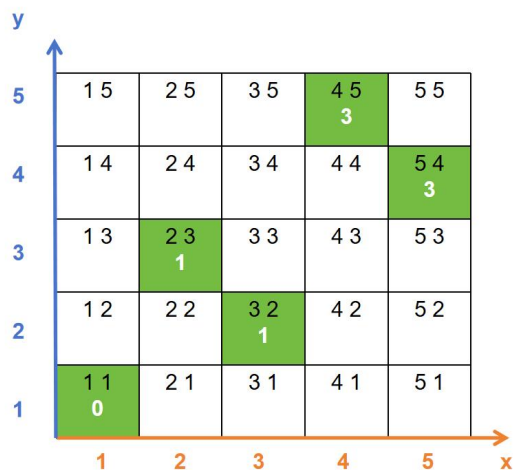
using namespace std;

const int SIZE = 100;

int x[SIZE], y[SIZE], f[SIZE];

int n, i, j, max_f, ans;

int main()
{
```



```
    cin >> n;

    for (i = 1; i <= n; i++) cin >> x[i] >> y[i];

    max_f = 0;

    for (i = 1; i <= n; i++)
    {
```

编号	1	2	3	4	5
坐标	1 1	2 3	3 2	4 5	5 4
战力	0	1	1	3	3

```
        f[i] = __①__; //

        for (j = 1; j <= n; j++)
        {
            if (x[j] < x[i] && __②__)
                __③__; //
        }

        if __④__ //
        {
            max_f = f[i];
            __⑤__ i; //
        }
    }

    for (i = 1; i <= n; i++) cout << f[i] << endl;

    cout << ans << endl;
```

```

    return 0;

}

```

正确答案：

- ① 0
- ② $y[j] < y[i]$
- ③ $f[i]++$ (或 $f[i] = f[i] + 1$)
- ④ $f[i] \geq \max_f$
- ⑤ $\text{ans} = i$

解析：

程序计算每个点的战斗力（左下方点的数量）：

- ① 初始化战斗力： $f[i] = 0$ （每个点初始战斗力为 0）。
- ② 判断纵坐标：需满足 $y[j] < y[i]$ （横坐标已通过 $x[j] < x[i]$ 判断）。
- ③ 增加战斗力：满足左下条件时， $f[i]++$ 。
- ④ 更新最大战斗力： $f[i] \geq \max_f$ （用 \geq 保证战斗力并列时取最大编号）。
- ⑤ 记录编号： $\text{ans} = i$ （更新当前最大战斗力对应的点编号）。

关键点：循环顺序（ i 从 1 到 n ）确保战斗力并列时， ans 记录的是最后出现的最大编号（即编号最大的点）。

完整版程序：

```

#include <iostream>

using namespace std;

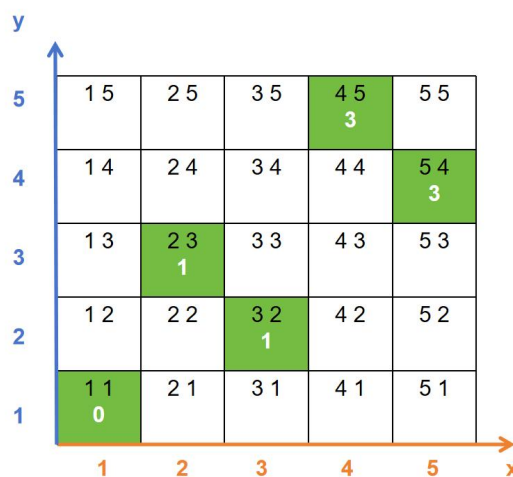
const int SIZE = 100;

int x[SIZE], y[SIZE], f[SIZE];

int n, i, j, max_f, ans;

int main() {

```



```

cin >> n;
for (i = 1; i <= n; i++) cin >> x[i] >> y[i];
max_f = 0;
for (i = 1; i <= n; i++) {
    f[i] = 0; // ① 初始化战斗力为 0
    for (j = 1; j <= n; j++) {
        if (x[j] < x[i] && y[j] < y[i]) // ② 判断点 j 是否在点 i 的
左下方
            f[i]++; // ③ 如果满足条件，战斗力+1
    }
    if (f[i] >= max_f) { // ④ 如果当前点的战斗力≥当前最大值
        max_f = f[i]; // 更新最大值
        ans = i; // ⑤ 记录当前点的编号
    }
}

for (i = 1; i <= n; i++) cout << f[i] << endl;

cout << ans << endl;

return 0;
}

```

编号	1	2	3	4	5
坐标	1 1	2 3	3 2	4 5	5 4
战力	0	1	1	3	3

程序解释

● 输入部分：

- 首先输入 n ，表示平面上有 n 个点。
- 然后依次输入每个点的 (x, y) 坐标，存储在数组 $x[]$ 和 $y[]$ 中。

● 计算每个点的战斗力：

- 初始化 $f[i] = 0$ (①)，表示点 i 的初始战斗力为 0。

- 遍历所有点 j ，检查是否满足 $x[j] < x[i] \ \&\& \ y[j] < y[i]$ (②)，即点 j 是否严格位于点 i 的左下方。
- 如果满足条件，则 $f[i]++$ (③)，表示点 i 可以控制点 j ，战斗力 $+1$ 。

● 记录战斗力最高的点：

- 在计算完每个点的战斗力后，检查 $f[i]$ 是否 \geq 当前最大值 \max_f (④)。
- 如果满足条件，则更新 $\max_f = f[i]$ ，并记录当前点的编号 $\text{ans} = i$ (⑤)。
- 注意：如果多个点的战斗力相同，由于遍历顺序是从小到大，最终 ans 会存储编号最大的那个点（题目要求）。

● 输出结果：

- 首先输出每个点的战斗力 $f[i]$ 。
- 最后输出战斗力最高的点的编号 ans 。

● 样例运行

输入：

5

1 1

2 3

3 2

4 5

5 4

输出：

0 （点 1 的战斗力，没有点在它左下方）

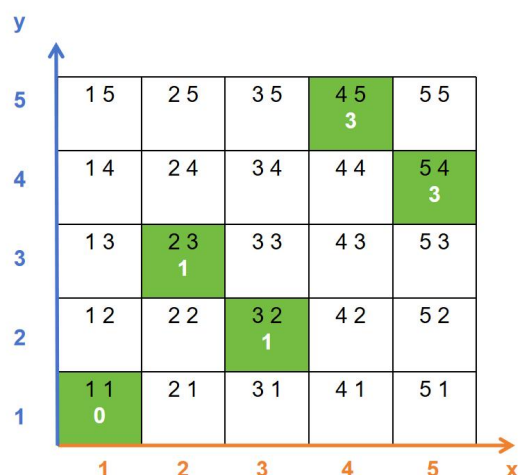
1 （点 2 的战斗力，点 1 在它左下方）

1 （点 3 的战斗力，点 1 在它左下方）

3 （点 4 的战斗力，点 1、2、3 在它左下方）

3 （点 5 的战斗力，点 1、2、3 在它左下方）

5 （战斗力最高的点是 5，因为点 4 和点 5 战斗力相同，取编号更大的 5）



编号	1	2	3	4	5
坐标	1 1	2 3	3 2	4 5	5 4
战力	0	1	1	3	3

总结

该程序的核心思想是暴力枚举，通过双重循环计算每个点的战斗力，并在遍历过程中动态更新最大值。虽然效率不是最优($O(n^2)$)，但对于题目给定的数据规模($n \leq 100$)是完全可行的。

第 4 题

以下程序输出的值是？

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

using namespace std;

int main() {

    int a[5] = {1,2,3};

    cout << a[4];

    return 0;

}
```

- A) -1
- B) 3
- C) 随机值
- D) 编译错误

答案： C

解析：

数组 `a` 在函数内定义(局部数组)，初始化部分为 `{1,2,3}`，剩余元素 `a[3]` 和 `a[4]` 未显式初始化。在 C++ 中，局部数组未初始化元素的值是不确定的（未定义行为），可能为随机值。因此输出 `a[4]` 是随机值，选项 C 正确。全局数组会初始化为 0，但这里是局部，不一定会是 0；选项 B 是 `a[2]` 的值；选项 D 不成立，程序编译正常。

第 5 题. 2007 年第 23 题

阅读程序写结果：

```
#include<stdio.h>

int main() {
```

```

int i, p[5], a, b, c, x, y = 20;

for (i = 0; i <= 4; i++) scanf("%d", &p[i]);

a = (p[0] + p[1]) + (p[2] + p[3] + p[4]) / 7;

b = p[0] + p[1] / ((p[2] + p[3]) / p[4]);

c = p[0] * p[1] / p[2];

x = a + b - p[(p[3] + 3) % 4];

if (x > 10)

    y += (b * 100 - a) / (p[p[4] % 3] * 5);

else

    y += 20 + (b * 100 - c) / (p[p[4] % 3] * 5);

printf("%d,%d\n", x, y);

return 0;

}

```

输入：6 6 5 5 3

答案：15,46

6 6 5 5 3
15,46

解析：

读入数组 p 的值为 6 6 5 5 3。

计算 a 为： $a = (6 + 6) + (5 + 5 + 3) / 7 = 12 + 13 / 7 = 12 + 1 = 13$

计算 b 为： $b = 6 + 6 / ((5 + 5) / 3) = 6 + 6 / 10 / 3 = 6 + 1 = 7$

计算 c 为： $c = 6 * 6 / 5 = 36 / 5 = 7$

计算 x 为： $x = 13 + 7 - p[(5 + 3) \% 4] = 13 + 7 - p[4 \% 4] = 13 + 7 - p[0] = 13 + 7 - 6 = 14$

$x > 10$ ，因此： $y += (7 * 100 - 13) / (p[3 \% 3] * 5) = 20 + (700 - 13) / (6 * 5) = 20 + 687 / 30 = 46$

输出 15,46。

第 6 题. 2008 年第 23 题

阅读程序写结果：

```
#include<iostream>

using namespace std;

int main() {

    int i, a, b, c, d, f[4];

    for(i = 0; i < 4; i++) cin >> f[i];

    a = f[0] + f[1] + f[2] + f[3];

    a = a / f[0];

    b = f[0] + f[2] + f[3];

    b = b / a;

    c = (b * f[1] + a) / f[2];

    d = f[(b / c) % 4];

    if(f[(a + b + c + d) % 4] > f[2])

        cout << a + b << endl;

    else

        cout << c + d << endl;

    return 0;

}
```

输入：9 19 29 39

答案：23

```
9 19 29 39
23
```

解析：

数组初始化：f[0]=9, f[1]=19, f[2]=29, f[3]=39

计算 a：a = (9 + 19 + 29 + 39) / 9 = 96 / 9 = 10

计算 b：b = (9 + 29 + 39) / 10 = 77 / 10 = 7

计算 c：c = (7 * 19 + 10) / 29 = 143 / 29 = 4

计算 d : $d = f[(7 / 4) \% 4] = f[1] = 19$

判断条件: $(a + b + c + d) \% 4 = (10 + 7 + 4 + 19) \% 4 = 40 \% 4 = 0$

检查 $f[0] > f[2] \rightarrow 9 > 29$ (不成立) \rightarrow 执行 else 分支

输出: $c + d = 4 + 19 = 23$

第 7 题. 2013 年第 26 题

阅读程序写结果:

```
#include <iostream>

using namespace std;

int main() {
    const int SIZE = 100;
    int height[SIZE], num[SIZE], n, ans;

    cin >> n;

    for (int i = 0; i < n; i++) {
        cin >> height[i];
        num[i] = 1;
        for (int j = 0; j < i; j++) {
            if ((height[j] < height[i]) && (num[j] >= num[i]))
                num[i] = num[j] + 1;
        }
    }

    ans = 0;
    for (int i = 0; i < n; i++) {
        if (num[i] > ans)
            ans = num[i];
    }

    cout << ans << endl;
```

i	0	1	2	3	4	5
height[]	2	5	3	11	12	4

i	0	1	2	3	4	5
num[]	1	2	2	3	4	3

i	0	1
height[]	2	5

i	0	1	2
height[]	2	5	3

i	0	1	2	3
height[]	2	5	3	11

i	0	1	2	3	4
height[]	2	5	3	11	12

i	0	1	2	3	4	5
height[]	2	5	3	11	12	4

```
}

```

输入：

6

2 5 3 11 12 4

答案：4

解析：

程序计算最长递增子序列的长度。

输入序列：[2, 5, 3, 11, 12, 4]。

```
6
2 5 3 11 12 4
4

```

动态规划过程：

i=0: num[0]=1 (子序列 [2])

i=1: 5>2 → num[1]=num[0]+1=2 (子序列 [2,5])

i=2: 3>2 但 3<5 → num[2]=num[0]+1=2 (子序列 [2,3])

i=3: 11>2,5,3 → 取最大 num[j] (num[1]=2) → num[3]=3 (子序列 [2,5,11] 或 [2,3,11])

i=4: 12>11 → num[4]=num[3]+1=4 (子序列 [2,5,11,12])

i=5: 4>3 但 4<5,11,12 → num[5]=num[2]+1=3 (子序列 [2,3,4])

最长递增子序列为 [2,5,11,12] 或 [2,3,11,12]，长度 ans=4。

第 8 题. 2013 年第 27 题

完善程序：（序列重排）

全局数组变量 a 定义如下：

```
const int SIZE = 100;

int a[SIZE], n;

```

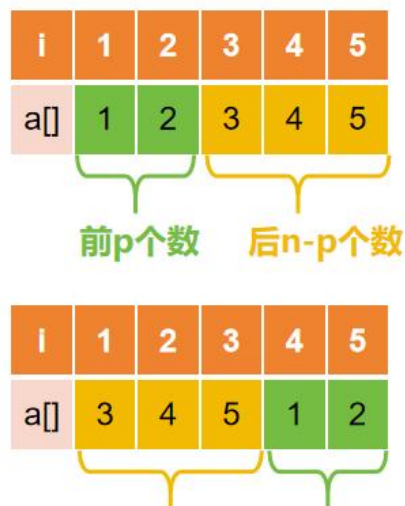
它记录着一个长度为 n 的序列 a₁, a₂, ..., a_n。

现在需要一个函数，以整数 p (1≤p≤n) 为参数，实现如下功能：将序列 a 的前 p 个数与后 n-p 个数对调，且不改变这 p 个数（或 n-p 个数）之间的相对位置。例如，

长度为 5 的序列 1,2,3,4,5, 当 $p=2$ 时重排结果为 3,4,5,1,2。

有一种朴素的算法可以实现这一需求,其时间复杂度为 $O(n)$ 、空间复杂度为 $O(n)$:

```
void swap1(int p) {
    int i, j, b[SIZE];
    for (i = 1; i <= p; i++)
        b[【填空 1】] = a[i]; //
    for (i = p + 1; i <= n; i++)
        b[i - p] = 【填空 2】; //
    for (i = 1; i <= 【填空 3】; i++)
        a[i] = b[i];
}
```



我们也可以用时间换空间, 使用时间复杂度为 $O(n^2)$ 、空间复杂度为 $O(1)$ 的算法:

```
void swap2(int p) {
    int i, j, temp;
    for (i = p + 1; i <= n; i++) {
        temp = a[i];
        for (j = i; j >= 【填空 4】; j--)
            a[j] = a[j - 1];
        【填空 5】 = temp; //
    }
}
```



答案:

$n - p + i$

$a[i]$

n

$i - p + 1$

$a[i - p]$

解析：

swap1 函数（空间复杂度 $O(n)$ ）：

填空 1：前 p 个元素移到数组 b 的末尾位置 $n-p+i$ 。

填空 2：后 $n-p$ 个元素移到数组 b 的开头位置 $i-p$ 。

填空 3：将 b 的所有元素复制回 a ，循环范围 $i=1$ to n 。

swap2 函数（空间复杂度 $O(1)$ ）：

填空 4：将元素 $a[i]$ 插入到前段，需移动 p 个元素，循环条件 $j \geq i-p+1$ 。

填空 5：将 $temp$ 放入正确位置 $a[i-p]$ 。

完整版程序

```
#include <iostream>

using namespace std;

const int SIZE = 100;

int a[SIZE], n;

// 时间复杂度  $O(n)$ , 空间复杂度  $O(n)$  的解法

void swap1(int p) {
    int i, b[SIZE];

    // 将前  $p$  个元素移动到新数组的末尾
    for (i = 1; i <= p; i++)
        b[n - p + i] = a[i]; // 填空 1:  $n - p + i$ 

    // 将后  $n-p$  个元素移动到新数组的开头
    for (i = p + 1; i <= n; i++)
        b[i - p] = a[i]; // 填空 2:  $a[i]$ 

    // 将新数组复制回原数组
```

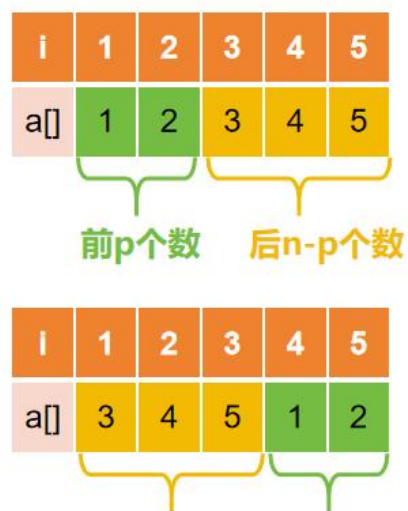


}

}

}

// 输出结果: 应为 3,4,5,1,2



```

cout << "重排结果: ";

for (int i = 1; i <= n; i++)

    cout << a[i] << " ";

cout << endl;

return 0;

}

```

第 9 题. 2015 年第 27 题

完善程序：

(打印日历) 输入月份 $m(1 \leq m \leq 12)$, 按一定格式打印 2015 年第 m 月的月历。
(第三、四空 2.5 分, 其余 3 分)

例如, 2015 年 1 月的月历打印效果如下 (第一列为周日) :

S	M	T	W	T	F	S
						1
						2
						3
						4
						5
						6
						7
						8
						9
						10
						11
						12
						13
						14
						15
						16
						17
						18
						19
						20
						21
						22
						23
						24
						25
						26
						27
						28
						29
						30
						31

```

#include <iostream>

#include <string>

using namespace std;

const int dayNum[] = {-1, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

int m, offset, i;

int main() {

    cin >> m;

    cout << "S\tM\tT\tW\tT\tF\tS" << endl;

```

```

offset = 【填空 1】; //
for (i = 1; i < m; i++)
    offset = 【填空 2】; //
for (i = 0; i < offset; i++)
    cout << '\t';

for (i = 1; i <= 【填空 3】; i++) { //
    cout << 【填空 4】; //
    if (i == dayNum[m] || 【填空 5】 == 0) //
        cout << endl;
    else
        cout << '\t';
}

return 0;
}

```

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

2	S	M	T	W	T	F	S
1	2	3	4	5	6	7	
8	9	10	11	12	13	14	
15	16	17	18	19	20	21	
22	23	24	25	26	27	28	

3	S	M	T	W	T	F	S
1	2	3	4	5	6	7	
8	9	10	11	12	13	14	
15	16	17	18	19	20	21	
22	23	24	25	26	27	28	
29	30	31					

4	S	M	T	W	T	F	S
5	6	7	8	9	10	11	
12	13	14	15	16	17	18	
19	20	21	22	23	24	25	
26	27	28	29	30			

答案:

4

$(\text{offset} + \text{dayNum}[i]) \% 7$

$\text{dayNum}[m]$

i

$(\text{offset} + i) \% 7$

解析:

填空 1: 2015 年 1 月 1 日是星期四, 初始偏移量 $\text{offset}=4$ 。

填空 2: 计算第 m 月 1 日的星期偏移量, 公式为 $(\text{上月偏移量} + \text{上月天数}) \% 7$ 。

填空 3: 循环输出当前月的天数 $\text{dayNum}[m]$ 。

填空 4: 直接输出日期 i 。

填空 5: 每 7 天换行 (周日换行), 判断条件 $(\text{offset} + i) \% 7 == 0$ 。

完整版程序

```
#include <iostream>

#include <string>

using namespace std;

const int dayNum[] = {-1, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

int m, offset, i;

int main() {
    cin >> m;

    cout << "S\tM\tT\tW\tT\tF\tS" << endl;

    offset = 4; // 填空 1

    for (i = 1; i < m; i++)
        offset = (offset + dayNum[i]) % 7; // 填空 2

    for (i = 0; i < offset; i++)
        cout << '\t';

    for (i = 1; i <= dayNum[m]; i++) { // 填空 3
        cout << i; // 填空 4

        if (i == dayNum[m] || (offset + i) % 7 == 0) // 填空 5
            cout << endl;
        else
            cout << '\t';
    }

    return 0;
}
```

第 10 题. 2016 年第 25 题

阅读程序写结果：

```
#include <iostream>

using namespace std;

int main() {

    int a[6] = {1, 2, 3, 4, 5, 6};

    int pi = 0;

    int pj = 5;

    int t, i;

    while (pi < pj) {

        t = a[pi];

        a[pi] = a[pj];

        a[pj] = t;

        pi++;

        pj--;

    }

    for (i = 0; i < 6; i++)

        cout << a[i] << " ";

    cout << endl;

    return 0;

}
```

输出：6,5,4,3,2,1,

解析：

pi 从首元素开始，pj 从末元素开始。

while 循环交换 a[pi] 和 a[pj]，并向中间移动 (pi++, pj--)。

最终数组完全反转：[6,5,4,3,2,1]。

第 11 题. 2019 年第 17 题

```

#include<stdio>

using namespace std;

int n, m;

int a[100], b[100];

int main() {

    scanf("%d%d", &n, &m);

    for (int i = 1; i <= n; ++i)

        a[i] = b[i] = 0;

    for (int i = 1; i <= m; ++i) {

        int x, y;

        scanf("%d%d", &x, &y);

        if (a[x] < y && b[y] < x) {

            if (a[x] > 0) b[a[x]] = 0;

            if (b[y] > 0) a[b[y]] = 0;

            a[x] = y;

            b[y] = x;

        }

    }

    int ans = 0;

    for (int i = 1; i <= n; ++i) {

        if (a[i] == 0) ++ans;

        if (b[i] == 0) ++ans;

    }

    printf("%d", ans);

    return 0;

```

4	2
2	3
1	4
4	

i	1	2	3	4
a[]	0	0	0	0
b[]	0	0	0	0

i	1	2	3	4
a[]	0	3	0	0
b[]	0	0	2	0

i	1	2	3	4
a[]	4	3	0	0
b[]	0	0	2	1

2	2
1	2
2	1
0	

i	1	2
a[]	0	0
b[]	0	0

i	1	2
a[]	2	0
b[]	0	1

i	1	2
a[]	2	1
b[]	2	1

```
}

```

判断题：

1、当 $m > 0$ 时，输出的值一定小于 $2n$ 。 ()

答案：A. 正确

解析：每对有效 (x,y) 会减少 2 个未匹配点 $(a[x]$ 和 $b[y])$ ，因此 $ans < 2n$ 。

2、执行完第 27 行的 $++ans$ 时， ans 一定是偶数。 ()

答案：B. 错误

解析：循环内有两个独立 if ， ans 可能增加 0、1 或 2，不一定是偶数。

3、 $a[i]$ 和 $b[i]$ 不可能同时大于 0。 ()

答案：B. 错误

解析：若存在匹配对 (x,y) 且 $x=y=i$ ，则 $a[i] > 0$ 且 $b[i] > 0$ 。

4、程序执行到第 13 行时， x 总是小于 y ，那么第 15 行不会被执行。 ()

答案：B. 错误

解析：第 15 行是 $if(a[x] > 0)$ 的条件，与 $x < y$ 无关。

选择题：

5. 若 m 个 x 两两不同，且 m 个 y 两两不同，则输出的值为 ()

答案：A. $2n - 2m$

解析：每对 (x,y) 消除两个未匹配点 $(x$ 和 $y)$ ，剩余 $ans = 2n - 2m$ 。

6. 若 m 个 x 两两不同，且 m 个 y 都相等，则输出的值为 ()

答案：A. $2n - 2$

解析：所有 y 相同 (设为 k)，仅最后一个 (x,k) 有效 (其他冲突被清除)，因此消除 2 个点， $ans = 2n - 2$ 。

(1) 程序功能详解

这段程序实现了一个**双向匹配管理系统**，主要用于处理两组元素 (1 到 n) 之间的匹配关系，并最终统计**未匹配元素的数量**。

(2) 核心数据结构

- $a[100]$: 存储 元素 x 的匹配对象
 $a[x] = y$ 表示元素 x 匹配了元素 y
 $a[x] = 0$ 表示元素 x 未匹配
- $b[100]$: 存储 元素 y 的匹配对象
 $b[y] = x$ 表示元素 y 匹配了元素 x
 $b[y] = 0$ 表示元素 y 未匹配

关键特性: 匹配是双向绑定的, 即 $a[x]=y$ 时必有 $b[y]=x$

(3) 程序逻辑分步

①初始化:

```
scanf("%d%d", &n, &m); // 输入元素总数 n 和操作次数 m
for (int i = 1; i <= n; ++i)
    a[i] = b[i] = 0; // 初始化所有元素未匹配
```

②处理匹配操作 (循环 m 次) :

```
int x, y;
scanf("%d%d", &x, &y); // 输入尝试匹配的 x 和 y
```

匹配条件检查:

```
if (a[x] < y && b[y] < x)
```

$a[x] < y$: 当前 x 的匹配对象 (如果有) 必须小于新尝试的 y

$b[y] < x$: 当前 y 的匹配对象 (如果有) 必须小于新尝试的 x

✓ 条件保证: 新匹配的 (x,y) 比旧匹配"更优"

③清除旧匹配 (若存在) :

```
if (a[x] > 0) b[a[x]] = 0; // 解除 x 原匹配对象的关系
```

```
if (b[y] > 0) a[b[y]] = 0; // 解除 y 原匹配对象的关系
```

④建立新匹配:

```
a[x] = y; // 建立  $x \rightarrow y$  的匹配
```

```
b[y] = x; // 建立  $y \rightarrow x$  的匹配
```

⑤统计未匹配数量:

```
int ans = 0;

for (int i = 1; i <= n; ++i) {

    if (a[i] == 0) ++ans; // 统计未匹配的 x

    if (b[i] == 0) ++ans; // 统计未匹配的 y

}

printf("%d", ans); // 输出总未匹配数
```

(4) 关键特性说明

匹配优先级:

- 系统总是保留更大的匹配对 (通过 $a[x] < y$ 和 $b[y] < x$ 实现)
- 示例: 若已有匹配 (2,3), 新匹配 (3,4) 会覆盖旧匹配

冲突解决:

- 当建立新匹配 (x,y) 时:
 - 自动解除 x 原有的匹配 (若有)
 - 自动解除 y 原有的匹配 (若有)
- 示例: 若 $x=1$ 原匹配 $y=2, y=3$ 原匹配 $x=4$, 新匹配 (1,3) 会同时解除 (1,2) 和 (4,3)

未匹配统计:

- 最终统计的是 所有元素的未匹配状态总和
- 每个元素在 a[] 和 b[] 中各被统计一次
- 最大未匹配数 = $2n$ (完全无匹配)
- 最小未匹配数 = $2n - 2m$ (所有匹配无冲突)

(5)示例演示 1

初始化: $n=4, m=2$

初始状态: $a = [0,0,0,0], b = [0,0,0,0]$

操作 1: ($x=2, y=3$)

$a[2] < 3$ & $b[3] < 2$, 条件满足 → 建立匹配:

$a[2] = 3$

$b[3] = 2$

当前状态:

$a = [0, 3, 0, 0]$

$b = [0, 0, 2, 0]$

操作 2: ($x=1, y=4$)

$a[1] < 4$ & $b[4] < 1$, 条件满足 → 建立匹配:

$a[1] = 4$

$b[4] = 1$

最终状态:

$a = [4, 3, 0, 0]$ // 元素 3、4 未匹配

$b = [0, 0, 2, 1]$ // 元素 1、2 未匹配

统计未匹配数量:

a 中未匹配: 位置 3、4 → +2

b 中未匹配: 位置 1、2 → +2

总计: $2+2=4$, 输出 4

(5) 示例演示 2

初始化: $n=2, m=2$

初始状态: $a = [0, 0], b = [0, 0]$

操作 1: ($x=1, y=2$)

$a[1] < 2$ & $b[2] < 1$, 条件满足 → 建立匹配:

$a[1] = 2$

$b[2] = 1$

当前状态:

$a = [2, 0]$

i	1	2	3	4
a[]	0	0	0	0
b[]	0	0	0	0

i	1	2	3	4
a[]	0	3	0	0
b[]	0	0	2	0

i	1	2	3	4
a[]	4	3	0	0
b[]	0	0	2	1

i	1	2
a[]	0	0
b[]	0	0

i	1	2
a[]	2	0
b[]	0	1

i	1	2
a[]	2	1
b[]	2	1

b = [0,1]

操作 2: (x=2, y=1)

a[2]<1&& b[1]<2, 条件满足 → 建立匹配:

a[2] = 1

b[1] = 2

最终状态:

a = [2,1]

b = [2,1]

第 12 题. 2021 年第 19 题 (Josephus 问题)

完善程序:

有 n 个人围成一个圈，依次标号 0 至 n-1。从 0 号开始，依次 0,1,0,1,... 交替报数，报到 1 的人会离开，直至圈中只剩一个人。求最后剩下人的编号。

```
#include <iostream>

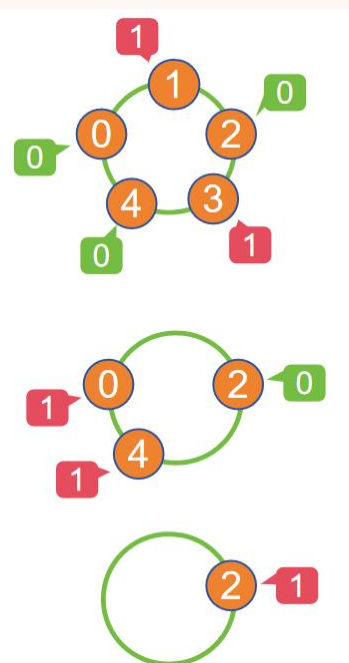
using namespace std;

const int MAXN = 1000000;

int F[MAXN];

int main() {
    int n;
    cin >> n;
    int i = 0, p = 0, c = 0;
    while (【①】) {
        if (F[i] == 0) {
            if (【②】) {
                F[i] = 1;
                【③】;
            }
        }
    }
```

5
2



```

        【④】;
    }

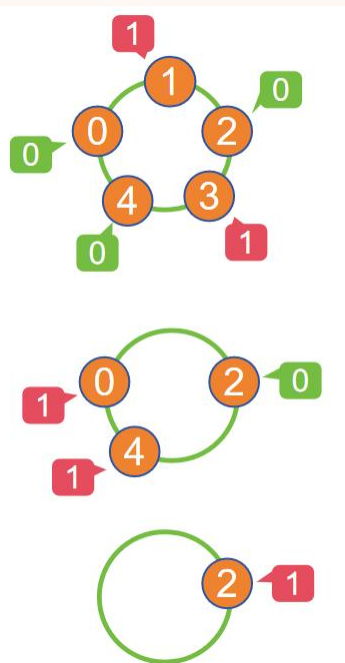
    【⑤】;
}

int ans = -1;
for (i = 0; i < n; i++)
    if (F[i] == 0) ans = i;

cout << ans << endl;

return 0;
}

```



答案:

① $c < n - 1$

② p

③ $c++$

④ $p \wedge = 1$

⑤ $i = (i + 1) \% n$

解析:

填空①: 需淘汰 $n-1$ 人, 循环条件 $c < n-1$ (c 为淘汰人数)。

填空②: p 控制报数 (0 或 1), $p=1$ 时淘汰当前人。

填空③: 淘汰后计数器 $c++$ 。

填空④: 每报一次数, p 翻转 ($p \wedge = 1$)。

填空⑤: 循环遍历数组, $i = (i+1) \% n$ 实现环形访问。

完整版程序

```

#include <iostream>

using namespace std;

```

```
const int MAXN = 1000000;

int F[MAXN];

int main() {

    int n;

    cin >> n;

    int i = 0, p = 0, c = 0;

    while ( c < n - 1 ) {

        if (F[i] == 0) {

            if (p) {

                F[i] = 1;

                c++;

            }

            p ^= 1;

        }

        i = (i + 1) % n;

    }

    int ans = -1;

    for (i = 0; i < n; i++)

        if (F[i] == 0) ans = i;

    cout << ans << endl;

    return 0;

}
```