A 公益活动

考点	难度
一维数组、if 语句	1

题目分析

本题考察目标希望用一维数组求取所有包裹衣服总重量的最大值和次大值，但是实际上可以有更为简便的编程思路，该思路无需引入数组，可以降低空间复杂度。

示例代码

使用数组的解法：

```
#include<stdio.h>
long long max_first,max_second,x,y;
long long a[100005];
int i,s;
int main()
{
    while(scanf("%lld%lld",&x,&y)!=EOF)
    {
        a[s]=x*y;
        s++;
    }
    for(i=0;i<s;i++)
    {
        if (a[i] >= max_first)
        {
            max_second = max_first;
            max_first=a[i];
        }
        else if (a[i] >= max_second)
           max_second =a[i];
    }
    printf("%lld %lld",max_first,max_second);
    return 0;
}
```

不使用数组的优化解法：

```
#include<stdio.h>

long long max_first,max_second, x, y;

int main()
{
    while (scanf("%lld%lld", &x, &y) != EOF) {
        x = x * y;
        if (x >= max_first)
```

```
        {
            max_second = max_first;
            max_first = x;
        }
        else if (x >= max_second)
            max_second = x;
    }
    printf("%lld %lld\n", max_first, max_second);
    return 0;
}
```

B #查询投票状态

考点	难度
一维数组使用	1

题目分析

如题所述，使用数组储存每个委员的投票情况并输出即可。

注意数组长度的大小以及下标的对应关系。

示例代码

```
#include <stdio.h>
int main(void) {
    int k, n, T;
    // vote数组统计每个委员的投票情况
    int vote[1005] = {0};
    scanf("%d%d%d", &k, &n, &T);
    for (int i = 1; i <= n; i++) {
        //每统计一个委员的投票情况
        scanf("%d", &vote[i]);
    }
    while (T--) {
        int tmp;
        scanf("%d", &tmp);
        printf("%d ", vote[tmp]);
    }
    return 0;
}
```

C Ori and the Entrance Puzzle

题目分析

简单的排序，可以使用冒泡排序或其他方法。排序完成后顺序、倒序输出即可

示例代码

```c
#include <stdio.h>
int num[1000];
int main(){
    int n;
    scanf("%d", &n);
    for(int i = 0; i < n; i++)
        scanf("%d", &num[i]);
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n - i - 1; j++){ //冒泡排序，仿照ppt方法
            if(num[j] > num[j + 1]){ //升序排序
                int temp = num[j];
                num[j] = num[j + 1];
                num[j + 1] = temp;
            }
        }
    for(int i = 0; i < n; i++) //升序输出
        printf("%d ", num[i]);
    printf("\n");
    for(int i = n - 1; i >= 0; i--) //降序输出
        printf("%d ", num[i]);
    return 0;
}
```

D 阿求的岁月史书

考点	难度
字符串，线性查找	2

题目分析

由题意可知，需要在字符串中查找给定的字符出现的位置。

朴素方法可以直接从头开始遍历整个字符串，直到到达字符串结尾，在过程中一一比对字符是否为所需字符，是则输出位置。

可以使用当前字符是否为终止符 `\0` 来判断是否结束——一个正常的字符串的结尾一定为一个终止符；或者可以使用 `string.h` 库中的 `strlen` 函数计算字符串的长度，但是：

！！注意，字符串的储存是从下标 0 开始的，也就是说 $i+1$ 才是其所在位置.长度为 $length$ 的字符串的最后一个可打印字符储存在 s[length-1] 中，而 s[length] 为终止符 `\0` ！！

此外，还需要一个标记变量 $flag$，用于记录遍历过程中是否查找到了字符，如果找到了，则输出行末回车，若未找到，输出给定的字符串。

阿求希望你们善用复制——包括题干中的代码块，样例输入输出，都是可以复制的。一个字符一个字符打多累啊，还可能会打错。

或者调用 `string.h` 库中的 `strchr` 函数，其可以查找字符串中某个字符的首个出现位置并返回指向这个位置的指针，不存在则返回空指针——指针相关内容会在后续学习中习得。

另外，字符串也是一个数组，因此和数组一样需要留出足够大的空间，否则会访问到错误的内存。

示例代码1——直接遍历

```
#include <stdio.h>
#include <string.h>
int main(void) {
    char s[101], ch;
    int T, length, i;
    scanf("%d%s", &T, s);
    // strlen函数得到字符串的长度作为循环结束条件
    length = strlen(s);
    while (T--) {
        int flag = 0;
        scanf(" %c", &ch);
        for (i = 0; i < length; i++) {
            //字符串在数组中是从下标0开始保存的，s[length]是终止符'\0'
            if (s[i] == ch) {
                printf("%d ", i + 1);
                flag = 1; //查找到了对应的字符，进行标记
            }
        }
        if (flag == 0) { //遍历整个字符串都没有标记，说明字符不存在
            printf("I'll add that paragraph... after my next reincarnation\n");
        } else { ////标记存在，说明找到过字符
            printf("\n");
        }
    }
    return 0;
}
```

示例代码2——使用 `string.h` 库中的函数 `strchr`

```
#include <stdio.h>
#include <string.h>
int main(void) {
    char s[101], ch;
    int T;
    scanf("%d%s", &T, s);
    while (T--) {
        scanf(" %c", &ch);
        char *p = strchr(s, ch);
        if (p == NULL) { //指针为空，说明未能找到字符
            printf("I'll add that paragraph... after my next reincarnation\n");
        } else { //指针不为空，说明查找到了字符
            //不断进行查找，每次从上次查找的位置之后进行查找
            while (p != NULL) {
                printf("%d ", p - s + 1);
```

```
                //指针运算会在之后的学习中介绍
                p = strchr(p + 1, ch);
            }
            printf("\n");
        }
    }
    return 0;
}
```

E 性能指标

题目分析

本题为双关键字排序，我们需要开两个数组 `id[], w[]` 分别保存性能指标所属的产品类型和对该产品质量的影响权重大小。 排序采用选择排序、冒泡排序、插入排序等均可，注意排序要求以及交换位置时 `id` 和 `weight` 需要同时交换。

最后输出排名时，注意性能指标相等的情况，可以使用一个变量 t 记录当前名次。当指标 i 与前一指标 $i-1$ 相同时，t 即为指标 i 的排名；否则用 $i+1$ 更新 t 。

注：示例代码采用选择排序，选择排序的核心思想为第 i 趟排序从序列的前 $n-i+1$ 个元素中选择值最小（大）的元素，将其置于第 $n-i$ 个。

示例代码

```
#include<stdio.h>
int n, id[1005], w[1005];
int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%d%d", &id[i], &w[i]);
    for (int i = 1; i < n; i++) { //选择排序
        int k = 0;
        for (int j = 1; j <= n - i; j ++)
            if (id[j] > id[k] || (id[j] == id[k] && w[j] < w[k]))
                k = j;
        int temp; // 交换 id 和 weight
        temp = id[k];
        id[k] = id[n - i];
        id[n-i] = temp;
        temp = w[k];
        w[k] = w[n - i];
        w[n-i] = temp;
    }
    int t = 1;
    printf("%d %d %d\n", t, id[0], w[0]); //注意需要先输出第0个指标
    for (int i = 1; i < n; i++) //输出排名
        if (id[i] == id[i-1] && w[i] == w[i-1])
            printf("%d %d %d\n", t, id[i], w[i]);
        else {
            t = i + 1;
            printf("%d %d %d\n", t, id[i], w[i]);
```

```
        }
    return 0;
}
```

F 进行一个数据的查

题目分析

本题总共需要实现三种操作，即插入数据、查询数据，删除数据。

首先考虑如何存储每个数据。我们可以使用一个二维数组，如 `num[1010][2]` ，`num[i][0]` 代表第 `i` 个数据的 id ， `num[i][1]` 代表第 `i` 个数据的值。

其次我们考虑查询数据的方式。通过观察数据范围，$1 \leq n \leq 1000$ 这个数据范围还是相当宽松的。每次对数组进行一次遍历也不会 TLE ，因此我们直接使用遍历进行查询数据即可。

在删除数据时，其实我们也要先进行一次数据查询的操作。在查询到编号为 id 的数据后，再考虑如何将其删除。删除的方式很简单：因为我们约定了 id 是非负整数，因此我们直接将 id 置为 -1 即可实现删除操作。

标准程序

```
#include <stdio.h>

int data[1010][2];
int main(){
    int n;
    int len = 0;    //存储过去总共读入数据的总个数
    scanf("%d", &n);
    while(n--){
        int op;
        int id;
        int value;
        scanf("%d", &op);
        ;if(op == 1){
            scanf("%d%d", &id, &value);
            data[len][0] = id;
            data[len][1] = value;
            len++;
        }
        else if(op == 2){
            scanf("%d", &id);
            int flag = 0;                //标记变量，标记是否成功查询到数据
            for(int i = 0; i < len; i++){
                if(id == data[i][0]){    //查询到了数据
                    flag = 1;            //将标记变量置0
                    printf("%d\n", data[i][1]);
                    break;
                }
            }
            if(flag == 0)
                printf("Not Found!\n");
        }
```

```
        else if(op == 3){
            scanf("%d", &id);
            int flag = 0;
            for(int i = 0; i < len; i++){
                if(id == data[i][0]){
                    flag = 1;
                    data[i][0] = -1;     //当要删除的数据存在于列表中，将其id置为-1即可成功删除
                    break;
                }
            }
            if(flag == 0)
                printf("Not Found!\n");
        }
    }

    return 0;
}
```

G 扫雷大手子 void

题目分析

本题是二维数组基础题，但是要注意如下问题

1. 数组的越界问题，如果数组是从 [0,0] 开始，而要求统计的九宫格中心点又在 [0,0]，那么就有可能会访问 [0,0] 的左上角，也就是 [-1,-1] 的位置，从而RE。要处理这个问题有两种方法，一种是在统计之前判断是否在范围内，另外一种就是把二维数组开成全局数组，然后从 [1,1] 开始计算，这样超出边界不会统计到错误的值也不会出现数组越界
2. 每一次进入统计循环时是否清零变量，或者可以使用函数来统计九宫格
3. 题目要求输出的字符串不是 WIN 和 LOSE，这是 void 故意的，这种要求输出某字符串的题最保险的就是从题目中复制，而代码块（红字）也便于复制，切忌凭感觉打个 WIN 和 LOSE 上去
4. void 想要介绍一种二维数组中围绕中心点统计的通用方法：方向数组。这一次统计九宫格比较简单，直接二重循环扫一下就行，但是如果遇到复杂情况，比如四角+中心格子，或者马走日形格子，那么就不方便统计，挨个列出来会使得代码冗长并且容易出错。那么在这个情况下，我们可以将中心点附近两个方向上的偏移量存到数组中，这样就可以用一个循环统计，对于马走日等形状也比较方便

代码示例

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<ctype.h>
#include<string.h>
#include<time.h>
int health,k;
int n,m,x,y;
int a[510][510];
int dx[9]={0,1,-1,0,0,1,-1,1,-1};//x方向的偏移量
int dy[9]={0,0,0,1,-1,1,1,-1,-1};//y方向的偏移量，注意要与x方向对应上
int T;
```

```
int main(){
    scanf("%d%d",&health,&k);
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
            scanf("%d",&a[i][j]);//二重循环读入二维数组
    scanf("%d",&T);
    for(int cnt=1;cnt<=T;cnt++){
        scanf("%d%d",&x,&y);
        int tot=0;
        for(int i=0;i<=8;i++){//枚举偏移量
            int tx=x+dx[i],ty=y+dy[i];//计算偏移之后的点，如果要判断是否越界可以在下面
加：
            //if((tx<=0)||(tx>=n)||(ty<=0)||(ty>=m))continue;
            tot+=a[tx][ty];//统计
            a[tx][ty]=0;//这个九宫格一定能清除
        }
        if(tot>k)
            health-=(1<<tot);//受伤
        if(health<=0){//判断失败
            printf("LOSE\n");
            printf("%d",cnt);
            return 0;
        }
    }
    printf("WlN\n");//成功
    printf("%d",health);
}
```

H I Appreciate Your Creativity.

题目分析

比较输入输出中的可见字符，直接使用 `scanf("%s", string)` 即可解决所有问题。

示例代码

```
#include <stdio.h>
#include <string.h>

char read[5010][5010];
char tmp[5010];

int main() {
    int state = 0, flag = 1, n = 0, m = 0;

    while (scanf("%s", tmp) > 0) {
        if (strcmp(tmp, "STD:") == 0)
            state = 1;
        else if (strcmp(tmp, "ANS:") == 0)
            state = 2;
        else if (state == 1) {
```

```
            n++;
            strcpy(read[n], tmp);
        }
        else if (state == 2) {
            m++;
            if (strcmp(read[m], tmp))
                flag = 0;
        }
    }
    if(n!=m)flag=0;

    printf("Expected: %d, Received: %d\n", n, m);
    if (flag)
        printf("Good Job! ");
    else
        printf("I Appreciate Your Creativity");
    return 0;
}
```

I 贤者之石

难度	考点
2	前缀和

问题分析

由题意，我们需要找到一个连续子串，使得其中 `R` 和 `B` 的数量相同。

考虑枚举每一个字符，再枚举以该字符为头的字符串，时间复杂度为 $O(n^2)$。注意到数据范围 $|\mathtt{s}| \leq 2 \times 10^5$，无法通过本题。

记 $\mathtt{s}_{ij}$ 为 s 从第 i 个字符到第 j 个字符的连续子串。为了简化状态的表示，我们用 1 和 -1 来表示 `R` 和 `B`，并记第 i 个字符的数值为 a_i。当子串 $\mathtt{s}_{ij}$ 满足题意时，有：

$$\sum_{k=i}^{j} a_k = 0$$

记 S_n 为 $\{a_n\}$ 的前 n 项和，则有：

$$S_j - S_i = 0$$

因此我们可以计算出 $\{S_n\}$，求该数列中相同项的项数之差最大值。考虑用 pos_i 表示 S_n 的值为 i 出现的最早位置。由于数组下标为非负数，我们将 $\{S_n\}$ 整体加上 $|\mathtt{s}|_{max}$，即 2×10^5。因为一次遍历即可同时更新 S_i 和 pos_i 的值，因此不需要为 $\{S_n\}$ 申请一个数组，用 sum 表示即可。

参考代码

```
#include <stdio.h>
#include <string.h>
#define N 200000
#define MAX(a,b) (((a)<(b))?(b):(a))
```

```
int sum = N, i, pos[2 * N + 5], len, ans;
char s[N + 5];

int main()
{
    scanf("%s", s);
    len = strlen(s);
    for (i = 1; i <= len; i++)
    {
        sum += s[i - 1] == 'R' ? 1 : -1;
        if (sum != N && !pos[sum])//为什么?
            pos[sum] = i;
        else
            ans = MAX(ans, i - pos[sum]);
    }
    printf("%d", ans);
    return 0;
}
```

当 $pos_i = 0$ 时，表示数值 i 还未出现过，因此可以更新。但有一个例外，不能更新 pos_N 的值，因为 $pos_N = 0$，$sum = N$ 表示 $\mathtt{s}_{1i}$ 是满足题意的子串。

J 冯 · 诺依曼的自然数理论 Plus

题目描述

冯 · 诺依曼给出了自然数的另一个定义，即序数和基数构造性定义，其可以简略地概括为：

$$
\begin{gathered}
0 = \{\} \\
1 = \{0\} = \{\{\}\} \\
2 = \{0, 1\} = \{\{\}, \{\{\}\}\} \\
\cdots \\
n = \{0, 1, \cdots, n-1\} \\
\cdots
\end{gathered}
$$

其中，将自然数用集合形式（即仅包含左大括号、右大括号、逗号这三种字符）表示，我们称其为该自然数的**本原表示**。

给定一个自然数 n，输出它的**本原表示**（其中元素顺序必须按照上述规则，即不能交换元素顺序）。

注意：仅包含上述三种字符，不包含空格！

我们不难发现，一个自然数 n **本原表示**的字符串长度关于 n 呈指数级增长，当 n 较大时，已经无法在很短的时间内打印出这些数字。

设自然数 n 本原表示的字符串 $idt(n)$。

现在， 对于一个给定的自然数 n，你需要求出：

- $|idt(n)|$，即自然数 n 本原表示的字符串长度，由于该数字很大，你需要对 `0x3ffffffe` 取模。
- $idt(n)_{l \ldots r}$，即自然数 n 本原表示的字符串从第 l 位至第 r 位的子串。

解题思路

这道题可以分解为两个子问题：求本原表示字符串长度、求该字符串指定一段区间的子串

求本原表示字符串长度

首先，同上次的题目，我们不难得到长度 $len[n]$ 的递推公式（大括号、逗号和子问题）：

$$len[0] = 2$$

$$len[n] = n + 1 + \Sigma_{i=0}^{n-1} len[i]$$

我们需要预处理来记录长度，以使得每次询问都能够在 $O(1)$ 的时间复杂度解决。于是我们只用关心预处理的时间复杂度。

但是我们注意到，$0 \leq n \leq 10^5$，也就是不能双重循环来预处理。我们记录前缀和来将复杂度降低为 $O(n)$。

```
#define ll long long
#define MOD 0x3fffffffll
ll idt_len[200000];
ll prefix_sum[200000]; // prefix_sum[n] = idt_len[0] + ... + idt_len[n-1]
void calc_idt_len() {
    idt_len[0] = 2;
    prefix_sum[1] = 2;
    for (int n = 1;n <= 100001;n++) {
        idt_len[n] = (n + 1 + prefix_sum[n]) % MOD;
        prefix_sum[n + 1] = (idt_len[n] + prefix_sum[n]) % MOD;
    }
}
```

求该字符串指定一段区间的子串

我们肯定不能将整个字符串存下来然后打印对应下标，因为这个字符串长度随 n 指数增长。

于是我们可以定义一个递归函数 `ll print_von_Neumann(int n, int l, int r)` 来打印子串，其中 l 表示左下标，r 表示右下标。详见示例代码。

示例代码

```
#include <stdio.h>
#define ll long long


ll idt_len[200000];
ll prefix_sum[200000]; // prefix_sum[n] = idt_len[0] + ... + idt_len[n-1]
#define MOD 0x3fffffffll
void calc_idt_len() {
    idt_len[0] = 2;
    prefix_sum[1] = 2;
    for (int n = 1;n <= 100001;n++) {
        idt_len[n] = (n + 1 + prefix_sum[n]) % MOD;
        prefix_sum[n + 1] = (idt_len[n] + prefix_sum[n]) % MOD;
    }
    // for (int i = 0;i < 50;i++) printf("idt_len[%d] = %lld\n", i, idt_len[i]);
```

```
}

ll print_von_Neumann(int n, int l, int r) {
    if (r < 0) return MOD; // 既然都到头了，那就直接结束
    if (l < 0) l = 0;
    if (l <= 0) printf("{");
    ll offset = 1; // 扫描的指针，从 0 开始，1 是因为左大括号
    for (int i = 0;i < n;i++) {
        offset += print_von_Neumann(i, l - offset, r - offset);
        if (i != n - 1) {
            if (offset >= l && offset <= r) printf(",");
            offset++;
        }
        if (offset > r) break;
    }
    if (offset >= l && offset <= r) printf("}");
    offset++;
    return offset;
}

int main() {
    calc_idt_len();
    int t;
    scanf("%d", &t);
    while (t--) {
        int n, l, r;
        scanf("%d%d%d", &n, &l, &r);
        printf("%lld ", idt_len[n]);
        print_von_Neumann(n, l - 1, r - 1);
        printf("\n");
    }
}
```

K 哪吒的 UNO

题目分析

本题使用递归遍历求解。类似 E5-F 的遍历思路，用其实本题每位玩家各自的所有出牌顺序就是 m 的全排列，总方案不超过 $(m!)^n$ 个。可以按照每次的玩家出牌来递归，遍历递归深度不超过 $m*n$ 层。具体思路如下：

全局二维数组 $color$ 和 num 分别记录第 i 个人的第 j 张牌的颜色和数字，为了方便从 0 开始计。递归函数接收三个参数 i,j,k，分别表示这是所有人出的牌中的第 k 次出牌，是第 i 个玩家的第 j 张牌（其实 i 可以由 k 对 n 取模得到，函数也可以写成接收两个参数）。用全局二维数组 map 记录第 i 个人的第 j 张牌是否被出过。

每次出牌（即每次调用递归函数），基本情况的条件应是 $k=n\times m$，即出到最后一张牌。对于一般情况，将 $map[i][j]$ 设为 1 表示此牌已经出过，设下一次出牌是第 ii 个玩家的第 jj 张牌，则 $ii=i+1$，若 $i=n-1$ 则 $ii=0$。从 $jj=0$ 开始遍历到 $jj=n-1$，只有第 ii 个玩家的第 jj 张牌此前没有被出过，且颜色与数字至少有一者与第 i 个玩家的第 j 张牌相同，才能出此牌，递归调用 $f(ii,jj,k+1)$。遍历结束后，记得将 $map[i][j]$ 设为 0，表示第 k 次出牌是此牌的情况已经遍历完，此牌没有出过，函数结束。

在 $main$ 函数中，输入数据后，从 $j=0$ 遍历到 $j=n-1$，调用 $f(0,j,1)$，表示第一次出牌是第 0 个玩家的第 j 张牌。

计数方法可以让递归函数返回值设为当前情况下能够完成目标的所有方案数，每次递归到基本情况对应一种出牌方案，返回 1。一般情况下，用局部变量 sum 记录当前情况下方案数，遍历 jj 时 sum 增加 $f(ii,jj,k+1)$，最后返回 sum，主函数中也用局部变量 sum 记录方案数，遍历 j 时 sum 增加 $f(0,j,1)$，最后输出 sum。 也可以用全局变量 sum 记录总方案数，每次递归到基本情况时 sum 自增 1，随后输出 sum 即可。

示例代码

递归基本相同，计数方式不同。

法一：递归函数返回当前情况下的方案数

```
#include <stdio.h>
int n, m;
int map[5][4], color[5][4], num[5][4];
int f(int i, int j, int k) {
    if(k == n * m) return 1; //基本情况对应一种方案，返回1
    map[i][j] = 1; //标记此牌出过，不能再出
    int ii = (i + 1) % n; //ii=i+1, i=n-1时ii=0
    int sum = 0; //记录当前情况下方案数
    for(int jj = 0; jj < m; ++jj)
        if(!map[ii][jj] && (color[ii][jj] == color[i][j] || num[ii][jj] ==
num[i][j])) //只有此牌此前没有被出过，且颜色与数字至少有一者与上一张牌相同时才能出此牌
            sum += f(ii, jj, k + 1); //递归调用，返回值加到sum上
    map[i][j] = 0; //出此牌的情况遍历完毕，标记此牌未被出过
    return sum;
}
int main() {
    scanf("%d%d", &n, &m);
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < m; ++j)
            scanf("%d%d", &color[i][j], &num[i][j]);
    int sum = 0; //记录总方案数
    for(int j = 0; j < m; ++j)
        sum += f(0, j, 1); //调用递归函数，返回值加到sum上
    printf("%d", sum); //输出结果
    return 0;
}
```

法二：全局变量sum记录总方案数，在递归函数基本情况中++sum

```
#include <stdio.h>
int n, m, sum; //全局变量sum记录总方案数
int map[5][4], color[5][4], num[5][4];
int f(int i, int j, int k) {
    if(k == n * m) {
        ++sum; //基本情况，sum自增1
        return ;
    }
    map[i][j] = 1;
    int ii = (i + 1) % n;
```

```
    for(int jj = 0; jj < m; ++jj)
        if(!map[ii][jj] && (color[ii][jj] == color[i][j] || num[ii][jj] ==
num[i][j]))
            f(ii, jj, k + 1); //递归调用
    map[i][j] = 0;
    return ;
}
int main() {
    scanf("%d%d", &n, &m);
    for(int i = 0; i < n; ++i)
        for(int j = 0; j < m; ++j)
            scanf("%d%d", &color[i][j], &num[i][j]);
    for(int j = 0; j < m; ++j)
        f(0, j, 1); //调用递归函数
    printf("%d", sum); //输出结果
    return 0;
}
```

Author: 哪吒