18 杭电计算机考研经验分享

目录

一、自我介绍

本人 18 届考生，性别男，报考杭州电子科技大学计算机科学与技术专业，本科是电子信息工程专业。来这里是给学弟学妹分享一下我的考研经验，仅作为参考。🐶

二、数学一备考经验

1.说明

数学我用的全部是张宇的。跟着宇哥走，没错的。宇哥会给我们规划的。视频分为基础班、强化班、冲刺班、点题班。我没看多少视频，我直接做题的，这个因人而异，但最好还是多花一点时间来刷题。我每天的数学时间固定在 18：00 到 22：00。刚开始做的时候，感觉像天书，什么都不会。我就给自己制定了一个规则，就是每道题都会规定一个时间，一般在 10 分钟以内，时间一到，无论做没做完都立刻结束这题，等待下一次刷第二遍，一遍不会

就两遍，两遍不会就三遍，三遍还不会就别考了🙃。哈哈，开个玩笑，总之就是多刷几遍，加深记忆。那具体是怎么操作的呢，有兴趣的可以 ob 一下。这里举个例子，高数 18 讲，以 3 章作为一个单位，首先前 6 章刷完第一遍，然后前 3 章完成 2 刷，再刷 7-9 章一遍，接下来 4-6 完成 2 刷，然后 1-3 完成 3 刷。这种方式是不是很有趣啊。好吧，具体怎么操作还是因人而异，大家怎么习惯怎么来吧。

前面闭关训练好了之后，就可以开始真题了，一定要给足 180 分钟，一般做题花 130 分钟，检查花 50 分钟（平时能做到，考试一样能做到，相信我）。这真题训练一定要到位，因为你平时几分，考试就几分，不要想着考试的时候超常发挥。真题一般是最简单的，除了前期的适应阶段分不高，后期平均 135 应该没问题的。接下来就是 8 套卷了，这时候就应该心态爆炸了（如果真题就把你心态做爆炸了，不用考了，洗洗睡吧🙃。不好意思，我又开了个玩笑，但是出现这个情况一定要反思，之前的训练是否充分，做的题量是否充足），心态爆炸不要紧，因为 8 套卷我有一套还是不及格的，还是水平不够，这是必须承认的事实。期待未来的大佬来个突破。你以为刷完 8 套卷心态就会好起来？不存在的，宇哥还有劝退 4 套卷呢，我当时 8 套卷做完 4 套后就不想做了，想拿起 4 套卷涨点信心，结果怎样我就不想多说了。所以说，到了考场才理解，真题是真的简单。

还有一些问题，数学要不要记笔记，我个人是没有笔记本的，把重要的全部记在了 36 讲上了。我非常赞成记笔记这个习惯，最好有个笔记本。那错题本呢，按照我的做法就不用错题本了，但是错题本还是有好处的，具体是什么好处呢，我也不知道。🐕

2.理想

我理想中的规划是这样的：

5 月份之前，做（张宇带你学+教材）一遍，基础篇，基础过一遍，了解有哪些概念，题型，解题方法等等。

7 月份之前，做（张宇 36 讲）三遍，提高篇

9 月份之前，做（张宇 1000 题）三遍，强化篇

9 月 20 号之前，做（闭关训练 180 题）三遍，修炼篇

11 月 10 号之前，做（真题大全解）三遍，实战篇，留年份最近的三套先不做。

12 月之前，做（张宇 8 套卷）三遍，劝退篇😶，不要以为真题都会做了就膨胀了，做做 8 套卷，安分点吧。

12 月份开始，每 3 天做一套，先做完（张宇 4 套卷），再做完留下来的三套真题，保持手感，至关重要。

（这里再说明一下，我说的做三遍的意思是：先做一遍，第二遍把第一遍做错的题目再做一遍，第三边把第二遍做错的题目再做一遍😬）

3.现实

接下来说一下我实际做到了什么，（教材）只做了高数和线代，概率论没做，（张宇带你学）只做了高数和线代，（张宇 36 讲）三遍，（张宇 1000 题）b 组三遍，（闭关训练 180 题）没做，（真题大全解）两遍，（张宇 8 套卷）只做了前 4 套，（张宇 4 套卷）只做完 3 套。

我一直认为实践是检验真理的唯一标准，做题>看视频，要相信量变会到质变的，我印象里有两次质变，第一次是真题刷到 135 分，第二次是 4 套卷刷到 135 分。但是发生的时间都比较晚，就是感觉前期的量还是不够，如果按照理想中的规划走，估计会提早质变。

三、其它初试科目备考

1.英语一

英语我是很早就开始了，那时候应该是基础班，可以看看视频，跟着老师的进度学。后期每天时间固定在 13：00-15：00，即每天两小时英语。

英语花时间最多的是阅读，阅读我做了很多，阅读真题刷了两遍，除了真题外我做了差不多有 80 篇课外阅读，其中有朱伟的。真题推荐黄皮书就可以了，解释的很全。，差不多 1 个小时刷一篇阅读，因为对答案要好长时间。刚开始练的时候，会错很多，当时非常难受，感觉阅读怎么跟天书一样，但是这次考试错了 4 道题，平均每篇错一道。

英语一的完形填空基本不用管，我就刷了 1 遍，太难了，平均能拿 6 分就可以了，这次考试我就拿了 6 分，已经很满意了。

英语一的翻译也不用管，真题我就刷了 1 遍，我反正目标拿 2 分就行了。这次考试我翻译只写了一半，估计只得了 1 分。

英语一的任务型阅读，个人感觉不难，有规律可循，反正这次考试我都做对了。真题刷两遍就行了。

剩下的就是作文了，作文我就是一个字，背。考前报个押题班也可以，我就是这样子的，当时我大作文只用了 15 分钟就默写完了，完完全全套上了，太爽了。两篇作文 30 分，我拿了 24 分左右，作文性价比真的高，这分拿的非常舒服。

2.政治

政治我是 9 月份开始的，看了一会儿视频，没看书本，直接做题的，政治我花时间最多的是选择题，肖秀荣的 1000 题我刷了 3 遍，另外刷了肖 8 和肖 4，然而主观题没背多少，考场上我抄了两个小时的题目，难受。这里提醒以下，个人感觉政治需要记住的东西还是挺多的，所以最好别小看它。

3.专业课

专业课就直接王道上的题目刷 3 遍，然后真题 3 遍就行了，毕竟专业课要想考高分，真的难。因为还有错题。真题不知道答案对不对的话，就找个学长问一下或者找个志同道合的人一起讨论就行。组成原理可能会用到推荐的课本，这个可以考虑下，数据结构直接用王道应该没问题。这次考试组成原理是直接考大题的，之前刷过的选择题和背诵题都没用到。

三、复试笔试备考

说明

相信大家看到这里的话，已经都过完初试了，我先祝大家初试顺利，接下来就要好好准备复试了。本人自知代码能力不如计算机专业的，就刷了一些 oj，具体如下：杭电 ACM 刷了 100 多的题目，刷了两遍，浙大 pat 乙级题目刷了 3 遍，浙大 pat 甲级前 100 题都刷了，前 50 题刷了两遍。现在给大家详解杭电复试笔试题目。你以为刷了这些你就无敌了吗，显然不是，这次考试我就低估了题目的难度，感觉自己白刷了，气死我了😣。这几天我整理了一下我写过的代码，顺便我给学弟学妹留下点东西，证明我是还是有那么一点点水平的。我把题目稍加修改（因为谁都不知道原题长什么样子），变得严谨一些。我所有的代码都是通过 Visual Studio 2017 调试的，这个软件在微软官网有的，免费的。

2009 年第 3 题

完数的定义：如果一个大于 1 的正整数的所有因子之和等于它的本身，则称这个数是完数，比如 6，28 都是完数：6=1+2+3；28=1+2+4+7+14。输入一个整数，判断它是否是完数。

输入：一个整数 n(1<n<10000)　　输入 1：6　　输入 2：28　　输入 3：2　　输入 4：9999　　输入 5：8128

输出：如果 n 是完数，输出 Yes，否则输出 No　　输出 1：Yes　　输出 2：Yes　　输出 3：No　　输出 4：No　　输出 5：Yes

什么是整数的因子，因子就是所有可以整除这个数的数,不包括这个数自身。这里举个例子：28=1*28=2*14=4*7，所以 1、2、4、7 是 28 的整数因子。知道了这个概念之后，这道题目就很容易了。参考代码如下：

```
#include<iostream>
using namespace std;
int main() {
    int n, sum = 0;        //sum用来存储所有因子之和
    scanf_s("%d", &n);     //输入
    for (int i = 1; i <= n / 2; i++)    //从1开始，到n/2结束，这是因为不存在大于n/2的整数因子
        sum += n % i ? 0 : i;           //如果是整数因子，就累加起来
    sum == n ? printf("Yes") : printf("No");     //最后判断所有因子之和是否和这个数本身相等，并输出
    return 0;
}
```

其实 10000 以内的完数只有 4 个，它们是 6、28、496、8128，这道题可以参考杭电 ACM1406

2010 年第 3 题

输入一组学生的信息，包括名字、 学号、英语成绩、语文成绩、数学成绩、科学成绩。然后按照总成绩从高到低进行排序并输出学生信息。
输入：第一行输入一个整数 n(0<n<10000)，接下来 n 行，每行给出一位学生的信息，包括：学号、名字、英语成绩、语文成绩、数学成绩、科学成绩，其中学号为 6 位整数，名字（不包含空格、长度不超过 8 个英文字母），各科成绩的区间为[0,100]，每个输入之间用空格隔开。
输出：输出有 n 行，每行输出一个学生的学号、名字、总成绩。按照成绩递减输出，如果有两个人的总成绩相同，那么按照学号升序输出。
输入 1：
5
000000 Tom 50 51 52 53
000001 Mike 60 40 55 45
000002 Eva 90 80 70 60
000003 Tim 80 75 70 65
000004 Joe 20 40 60 80
输出 1：
000002 Eva 300
000003 Tim 290
000000 Tom 206
000001 Mike 200
000004 Joe 200

这道题只需要使用结构体排序就可以了，字符串用 string 类型。参考代码如下：

```
#include<iostream>
#include<string>      //每当用到string最好添加这个头文件
#include<algorithm>   //sort需要用到这个头文件
using namespace std;
```

```
struct STUDENT {  //使用结构体存储学生信息
    string id, name;  //id代表学号，学号用字符串表示，比较方便
    int English, Chinese, math, science, total; //各科成绩以及总成绩
}Stu[10000];
bool cmp(STUDENT A, STUDENT B) {    //按照成绩递减排序，如果成绩相同，则按照学号递增排序
    return A.total == B.total ? A.id < B.id : A.total > B.total;
}
int main() {
    int n;
    cin >> n;     //输入学生数量
    for (int i = 0; i < n; i++) {  //输入

        cin >> Stu[i].id >> Stu[i].name >> Stu[i].English >> Stu[i].Chinese >> Stu[i].math >> Stu[i].science;
        Stu[i].total = Stu[i].English + Stu[i].Chinese + Stu[i].science + Stu[i].math;  //计算总成绩
    }
    sort(Stu, Stu + n, cmp);        //按照成绩递减排序，如果成绩相同，则按照学号递增排序
    for (int i = 0; i < n; i++)     //输出
        cout << Stu[i].id << ' ' << Stu[i].name << ' ' << Stu[i].total << endl;
    return 0;
}
```

这道题目可以参考浙大 pat 乙级 1004

2011 年第 3 题

丑数是这样定义的：如果一个正整数的素因子只包含 2、3、5、7 四种，则它被称为丑数。以下数列 1，2，3，4，5，6，7，8，9，10，12，14，15，

16，18，20，21，24，25，27.....就显示了前 20 个丑数。给出一个正整数 N，判断这个数是否为丑数。

输入：输入一个整数 n(0<n<10000)　　输入 1：1　　输入 2：11　　输入 3：9800　　输入 4：9999

输出：如果是丑数，输出 Yes，否则输出 No　　输出 1：Yes　　输出 2：No　　输出 3：Yes　　输出 4：No

丑数的判断方法：首先除 2，直到不能整除为止，然后除 3 到不能整除为止，然后除 5 直到不能整除为止，然后除 7 直到不能整除为止。最终判断剩余的数字是否为 1，如果是 1 则为丑数，否则不是丑数。参考代码如下：

```
#include<iostream>
using namespace std;
int main() {
    int N[4] = { 2,3,5,7 }, n; //用数组存储4个因子
    scanf_s("%d", &n);    //输入
    for (int i = 0; i < 4; i++)    //依次除以（2、3、5、7）4个因子，直到不能整除为止
        while (n%N[i] == 0)
            n /= N[i];
    n == 1 ? printf("Yes") : printf("No");  //如果剩1，则为丑数，并且输出
    return 0;
}
```

关于丑数问题，参考杭电 ACM1058、1492

2012 年第 2 题

Worm is an old computer game. There are many versions, but all involve maneuvering a "worm" around the screen, trying to avoid running the worm into itself or an obstacle.

We'll simulate a very simplified version here. The game will be played on a 50 x 50 board, numbered so that the square at the upper left is numbered (1, 1). The worm is initially a string of 20 connected squares. Connected squares are adjacent horizontally or vertically. The worm starts stretched out

horizontally in positions (25, 11) through (25, 30), with the head of the worm at (25, 30). The worm can move either East (E), West (W), North (N) or South (S), but will never move back on itself. So, in the initial position, a W move is not possible. Thus, the only two squares occupied by the worm that change in any move are its head and tail. Note that the head of the worm can move to the square just vacated by the worm's tail.
You will be given a series of moves and will simulate the moves until either the worm runs into itself, the worm runs off the board, or the worm successfully negotiates its list of moves. In the first two cases you should ignore the remaining moves in the list.

Input
There will be multiple problems instances. The input for each problem instance will be on two lines. The first line is an integer n (<100) indicating the number of moves to follow. (A value of n = 0 indicates end of input.) The next line contains n characters (either E, W, N or S), with no spaces separating the letters, indicating the sequence of moves.

Output
Generate one line of output for each problem instance. The output line should be one of the follow three:
The worm ran into itself on move m.
The worm ran off the board on move m.
The worm successfully made all m moves.
Where m is for you to determine and the first move is move 1.

Sample Input
18
NWWWWWWWWWWWSESSSWS
20
SSSWWNENNNNNNWWWWSSSS
30
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
13

SWWWWWWWWWNEE
0

Sample Output
The worm successfully made all 18 moves.
The worm ran into itself on move 9.
The worm ran off the board on move 21.
The worm successfully made all 13 moves.

题目大意：整个游戏棋盘是 50*50 大小的，左上角在(1,1)，贪吃蛇由 20 个节点组成，头部位置在（25，30），水平延展到（25，11），可以有四个运动方向：东，西，南，北。题目就是给你一个运动序列，判断最终结果是下面 3 种情况的哪一种：①正常。②头撞到自己身体。③出界。
需要注意的是贪吃蛇每走一步，在游戏盘上改变的只有头尾两个点，相当于尾巴减少一点，头部增加一点。参考代码如下：

```
#include<iostream>
#include<vector>
using namespace std;
int main() {      //下面的数组D表示四个方向，E东，W西，N北，S南。数组M用来将字母转化为对应D的方向
    int n, c, D[4][2] = { { 0,1 },{ 0,-1 },{ -1,0 },{ 1,0 } }, M[128];//c用来遍历下面的运动序列s，c+1记录的是实际走的步数
    M['E'] = 0, M['W'] = 1, M['N'] = 2, M['S'] = 3;      //E表示向东走一步，即y轴+1；W表示向西走一步，即y轴-1
    while (scanf("%d", &n) && n) {      //当输入n为0的时候停止
        vector<vector<bool> >Arr(51, vector<bool>(51)); //二维数组，表示游戏棋盘，用来存储蛇的20个位置
        vector<pair<int, int> >V(20);  //用来存储蛇的20个点的坐标
        for (int i = 0; i < 20; i++)   //刚开始，头部位置在（25，30），水平延展到（25，11）
            V[i].first = 25, V[i].second = 30 - i, Arr[25][30 - i] = 1;
        char s[100];      //用来存储运动序列
        scanf("%s", s);   //输入运动序列
        for (c = 0; c < n; c++) { //从头开始遍历运动序列
```

```
            Arr[V[19].first][V[19].second] = 0;//每走一步，尾巴的位置便清零，即尾部减少一点
            for (int i = 19; i > 0; i--)        //除了头部外，每个点都向前移动一格
                V[i] = V[i - 1];
            V[0].first += D[M[s[c]]][0], V[0].second += D[M[s[c]]][1];        //根据不同的指令，头部进行不同方向的移动，即头部增加一点
            if (V[0].second < 1 || V[0].second > 50 || V[0].first < 1 || V[0].first>50) {   //判断是否出界
                printf("The worm ran off the board on move %d.\n", c + 1);    //c+1记录的是实际走的步数
                break;
            }
            if (Arr[V[0].first][V[0].second]) {//判断是否撞到自己
                printf("The worm ran into itself on move %d.\n", c + 1);
                break;
            }
            Arr[V[0].first][V[0].second] = 1;  //程序运行到这里就表明贪吃蛇的头部可以移动到该单位，置1
        }
        if (c == n)       //如果整个运动序列都能执行完成，则顺利完成移动
            printf("The worm successfully made all %d moves.\n", n);
    }
    return 0;
}
```

这道题目就是 zoj1056，代码是提交成功了的。关于矩阵，参考浙大 pat 乙级 1050

Run ID	Submit Time	Judge Status	Problem ID	Language	Run Time(ms)	Run Memory(KB)
4182046	2018-04-05 15:49:46	Accepted	1056	C++11	0	288

2013 年第 2 题

一个活动有 N 个人参加，一个主持人和 N-1 个普通参加者，其中所有的人都认识主持人，主持人也认识所有的人，主持人要求 N-1 个参加者说出他们在参加者中所认识的人数，如果 A 认识 B，则 B 认识 A，所以最少是会认识一个人，就是主持人，他们说出了自己所认识的人数后，需要判断他们中有没有人说谎。

输入：第一行输入一个整数 n(1<n<100)，表示参加活动的总人数。第二行输入 n-1 个正整数 m(1<m<100)。

输出：如果他们说的情况可能成立，输出 Maybe truth，否则输出 Lie absolutely

输入 1：	输入 2：	输入 3：	输入 4：
7	9	5	2
1 2 4 5 5 3	3 7 7 7 7 5 6 6	2 2 5 2	1
输出 1：	输出 2：	输出 3：	输出 4：
Lie absolutely	Maybe truth	Lie absolutely	Maybe truth

我以输入 1、2 为例，说一下我的做法。首先定义一个有 n 个元素的数组，每个元素初始值为 n-1，这是因为主持人认识 n-1 个人。

例 1：6 6 6 6 6 6 6　　例 2：8 8 8 8 8 8 8 8

第二步，输入 n-1 个数，从数组第 2 个元素开始

例 1：6 1 2 3 5 5 3　　例 2：8 3 7 7 7 7 5 6 6

第三步，将数组从大到小排序

例 1：6 5 5 3 3 2 1　　例 2：8 7 7 7 7 6 6 5 3

第四步，将数组第一个元素取出，假设值为 a，如果 a 大于数组的长度，则直接输出 Lie absolutely，否则将数组的前 a 个元素减 1

例 1：4 4 2 2 1 0　　例 2：6 6 6 6 5 5 4 2

重复第三、四步，如果出现某个元素值为负数，输出 Lie absolutely，如果数组里边没有了元素，则输出 Maybe truth

例 1：3 1 1 0 0　　例 2：5 5 5 4 4 3 2

例 1：0 0 -1 0　　例 2：4 4 3 3 2 2 -> 3 2 2 1 2 -> 3 2 2 2 1 -> 1 1 1 1 -> 0 1 1 -> 1 1 0 -> 0 0 -> 0

输出 Lie absolutely　　输出 Maybe truth

这里需要注意的是输入 3，一共才 5 人，如果一个人说他认识 5 个人，肯定在说谎，因为最多认识 4 个人。那个回忆版的做法不是完全正确的。
下面给出参考代码：

```
#include<iostream>
#include<vector>
#include<algorithm>        //sort
#include<functional>       //greater<int>()
using namespace std;
int main() {
    int n, f = 1;              //f只是标记，1表示Maybe truth，0代表Lie absolutely
    scanf_s("%d", &n);         //输入
    vector<int>V(n, n - 1);    //第一步，定义一个数组，n个元素，值为n-1
    for (int i = 1; i < n; i++)         //第二步，输入n-1个数
        scanf_s("%d", &V[i]);
    while (V.size() && f) {             //循环结束条件有两个，一个是标志f为0时，表明Lie，一个是数组长度为0，表明truth
        sort(V.begin(), V.end(), greater<int>());    //第三步，将数组从大到小排序
        f = V[0] > V.size() - 1 ? 0 : f;             //数组第一个数如果大于数组剩余元素的个数，将f置0
        for (int i = 1; i <= V[0] && f; i++)         //将数组前a个元素减1，如出现负数，将f置0
            V[i]--, f = V[i] < 0 ? 0 : f;
        V.erase(V.begin());                          //去掉数组第一个元素
    }
    f ? printf("Maybe truth") : printf("Lie absolutely");//输出
    return 0;
}
```

这道题目和可图性判定非常相似，参考杭电 ACM2454
可图性判定，两个概念，一个定理。
度序列：若把图 G 所有顶点的度数排成一个序列 S，则称 S 为图 G 的度序列。

序列是可图的：一个非负整数组成的有限序列如果是某个无向图的度序列，则称该序列是可图的。
Havel-Hakimi 定理：由非负整数组成的非增序列 S：d_1、$d_2 \cdots d_n$ (n≥2, d_1≥1)是可图的，当且仅当序列 S1：d_2-1、$d_3-1 \cdots d_{d_1+1}-1$、$d_{d_1+2} \cdots d_n$是可图的。其中，序列 S1 中有 n-1 个非负整数，S 序列中d_1后的前d_1个度数(即d_2~d_{d_1+1})减 1 后构成 S1 中的前d_1个数。

2014 年第 2 题

Problem Description
Given a string containing only 'A' - 'Z', we could encode it using the following method:
1. Each sub-string containing k same characters should be encoded to "kX" where "X" is the only character in this sub-string.
2. If the length of the sub-string is 1, '1' should be ignored.

Input
The first line contains an integer N (1 <= N <= 100) which indicates the number of test cases. The next N lines contain N strings. Each string consists of only 'A' - 'Z' and the length is less than 10000.

Output
For each test case, output the encoded string in a line.

Sample Input	Sample Output
2	ABC
ABC	A2B3C
ABBCCC	

这道题目直接遍历字符串就行了，如果和前一个字母相等则计数，不想等则输出。参考代码如下：

```
#include<iostream>
```

```
#include<string>  //每当用到string最好添加这个头文件
using namespace std;
int main() {
    int n, c;    //c用来计数的
    cin >> n;    //输入
    while (n--) {
        string s;                    //用来存储字符串
        cin >> s, c = 1, s += '#';    //输入字符串，字符串后面添加一个‘#’方便处理
        for (int i = 1; i < s.size(); i++) {    //从第二个字符开始遍历
            if (s[i] == s[i - 1])               //如果和前面的字符相同，则计数
                c++;
            else if (c > 1)                     //如果不同，则输出，注意1需要省略
                cout << c << s[i - 1], c = 1;
            else cout << s[i - 1];
        }
        cout << endl;    //每个输出用例后面接个换行符
    }
    return 0;
}
```

这道题目是杭电 ACM1020，代码是提交通过的。字符串的处理，可以参考浙大 pat 乙级 1040、1042

Run ID	Submit Time	Judge Status	Pro.ID	Exe.Time	Exe.Memory	Code Len.	Language
24331154	2018-04-06 12:33:08	Accepted	1020	0MS	1940K	561 B	C++

2015 年第 2 题

给定一个数字矩阵，如果上下左右数值相同，则表示是一个连通的区域，求矩阵中连通块为 1 的数字有多少。
输入：第一行输入两个整数 n 和 m，分别表示矩阵的行数和列数。接下来 n 行，每行输入 m 个整数，表示数字矩阵。所有输入的数的区间为[1,100]
输出：矩阵中连通块为 1 的数字的数量

输入 1：
5 5
1 1 2 1 1
2 1 1 1 2
3 3 2 3 3
4 3 2 3 4
5 5 2 5 5
输出 1：
1
说明 1：
数字 1，1 个连通块
数字 2，4 个连通块
数字 3，2 个连通块
数字 4，2 个连通块
数字 5，2 个连通块

输入 2：
5 6
4 4 4 4 4 4
4 2 3 3 1 4
4 2 2 3 1 4
4 2 3 3 1 4
4 4 4 4 4 4
输出 2：
4
说明 2：
数字 1，1 个连通块
数字 2，1 个连通块
数字 3，1 个连通块
数字 4，1 个连通块

输入 3：
1 1
1
输出 3：
1

采用深度优先遍历连通区域，记录每个数字的连通块个数，最后数一下连通块数目为 1 的数字有多少就可以了

```
#include<iostream>
#include<algorithm>        //count需要用到该头文件
using namespace std;       //A储存数字矩阵，D表示上下左右四个方向，M记录数字的连通块个数，Visited标记是否访问过
```

```
int A[101][101], n, m, c, D[4][2] = { { 0,1 },{ 0,-1 },{ 1,0 },{ -1,0 } }, M[101], Visited[101][101];
void DFS(int a, int b, int x) {                  //a、b代表坐标，x代表需要深度遍历的数字
    Visited[a][b] = 1;    //先标记为1，表示访问了
    for (int i = 0; i < 4; i++) {                //分别向四个方向遍历
        int c = a + D[i][0], d = b + D[i][1];
        if (Visited[c][d] == 0 && A[c][d] == x) //如果数字相同并且未被访问，则继续DFS
            DFS(c, d, x);
    }
}
int main() {
    scanf_s("%d%d", &n, &m);            //输入
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            scanf_s("%d", &A[i][j]);   //输入
    for (int i = 1; i <= n; i++)        //从(1,1)开始遍历
        for (int j = 1; j <= m; j++)
            if (Visited[i][j] == 0)    //如果没访问过，就深度优先遍历
                DFS(i, j, A[i][j]), M[A[i][j]]++;   //M[3]=2表示数字3当前有2个连通块
    printf("%d", count(M, M + 101, 1));//输出连通块为1的数字的个数
    return 0;
}
```

类似的可以参考浙大pat甲级1013

2016 年第 4 题

有一个由数字组成的二维矩阵，大小为 N*M；还有一个大小为 n*m 小二维矩阵，想象将小二维矩阵放在大矩阵的不同位置（小矩阵左上角位置和大矩阵某个位置对应放齐），这两个二维矩阵对应位置的数字绝对值之差和一般是不同的，求这个最小绝对值之差的和，并求出对应的大矩阵位置。
输入：第一行输入两个正整数 N 和 M，表示大矩阵的大小，大小区间为[1,100]，接下来 N 行，每行输入 M 个整数，大小区间为[-100,100]。接下来一行，输入两个正整数 n 和 m(n<=N,m<=M)，表示小矩阵的大小，大小区间为[1,100]，接下来 n 行，每行输入 m 个整数，大小区间为[-100,100]。
输出：最小的绝对值之差之和，对应于大矩阵的起始坐标，中间用逗号隔开

输入 1：	输入 2：	输入 3：
4 4	4 6	5 8
1 2 3 4	1 2 3 4 5 6	1 2 3 4 5 6 7 10
4 5 6 8	4 5 6 7 8 9	0 0 4 5 6 7 8 12
1 2 3 4	4 5 6 1 2 3	0 0 5 6 7 8 0 -1
5 6 7 8	7 8 9 4 5 6	3 0 6 7 8 9 0 -2
2 2	2 2	5 6 7 8 9 0 1 -3
1 2	1 2	3 2
4 5	4 5	0 -1
输出 1：	输出 2：	0 -2
0,(1,1)	0,(1,1),(3,4)	3 -1
		输出 3：
		4,(2,1),(3,7)

这道题目直接用暴力求解便可，参考代码如下：

```
#include<iostream>
#include<vector>
#include<algorithm>        //min,min_element需要用到该头文件
```

```
using namespace std;
int N, M, n, m, c, mins = 0x7fffffff;   //c用于Temp数组，mins表示最小的绝对值之差之和，初始值设为int类型的最大整数
void Input(vector<vector<int>>&A) {     //矩阵的输入
    for (unsigned int i = 1; i < A.size(); i++)
        for (unsigned int j = 1; j < A[i].size(); j++)
            scanf_s("%d", &A[i][j]);
}
int main() {
    scanf_s("%d%d", &N, &M);   //输入
    vector<vector<int>>Big(N + 1, vector<int>(M + 1));   //二维数组(N+1)*(M+1)，大矩阵
    Input(Big), scanf_s("%d%d", &n, &m);
    vector<vector<int>>Small(n + 1, vector<int>(m + 1)); //二维数组(n+1)*(m+1)，小矩阵
    Input(Small);              //输入
    vector<int>Temp((N - n + 1)*(M - m + 1));            //临时矩阵，存放绝对值之差之和
    for (int i = 1; i <= N - n + 1; i++)                 //从(1,1)开始遍历大矩阵
        for (int j = 1; j <= M - m + 1; j++) {
            int ans = 0;       //用来计算绝对值之差之和
            for (int c1 = 1; c1 <= n; c1++) {            //从(1,1)开始遍历小矩阵
                for (int c2 = 1; c2 <= m; c2++)          //计算绝对值之差之和
                    ans += abs(Big[i + c1 - 1][j + c2 - 1] - Small[c1][c2]);
            }
            Temp[c++] = ans, mins = min(mins, ans);      //将每个绝对值只差之和存入Temp中，同时用mins存储最小的
        }
    printf("%d", mins);            //输出最小的绝对值之差之和
    auto p = min_element(Temp.begin(), Temp.end()), q = p;    //p和q是指针，指向第一个最小的元素
    while (p != Temp.end()) {      //循环结束条件，就是p指向Temp末尾
```

```
            printf(",(%d,%d)", (p - Temp.begin()) / (M - m + 1) + 1, (p - Temp.begin()) % (M - m + 1) + 1);  //输出坐标
            q = p, p = find(q + 1, Temp.end(), mins);              //使得p指向下一个最小的元素
        }
        return 0;
}
```

也可以做做 pat 乙级 1068

2017 年第 3 题

有一个 N*M 的大字符矩阵和一个 n*m 的小字符矩阵，从大矩阵中取出小矩阵，求最多能取出多少个小矩阵

输入：第一行输入两个正整数 N 和 M，代表大矩阵，区间[1,100]，接下来 N 行，每行输入 M 个字符(仅 26 个英文字母的大小写)，中间用空格隔开，表示大矩阵。接下来一行输入两个正整数 n(n<=N)和 m(m<=M)，代表小矩阵，然后 n 行，每行输入 m 个字符(仅 26 个英文字母的大小写)，中间用空格隔开，表示小矩阵

输出：能从大矩阵中取出的小矩阵的最大数目

输入 1：
3 4
a b c d
c d a b
a c c d
2 2
a b
c d

输出 1：
2

输入 2：
5 6
a b a b a b
a b a b a b
a b a a b a
a b a a b a
a b a a b a
2 3

输出 2：
a b a
a b a

3

这道题目和上一道题目一样，暴力即可。参考代码如下：

```
#include<iostream>
#include<vector>
using namespace std;
vector<string>A(1001, string(1001, '#')), B(1001, string(1001, '#'));  //A表示大矩阵，B表示小矩阵
int N, M, n, m, ans, Visited[1001][1001];        //ans用来记录从大矩阵中取出小矩阵的数量，Visited标记是否被取出
void Input(vector<string>&V, int x, int y) {     //矩阵输入函数
    for (int i = 1; i <= x; i++)
        for (int j = 1; j <= y; j++)
            scanf_s(" %c", &V[i][j], 1);         //相当于scanf(" %c",&V[i][j])，但是VS2017需要用scanf_s才能通过编译
}
bool F(int x, int y) {          //该函数的功能：判断能否取出，能返回1
    bool f = 1;                 //标记，0代表不能取出
    for (int i = 1; i <= n && f; i++)
        for (int j = 1; j <= m && f; j++)        //如果大矩阵与小矩阵不匹配，或者被取出过，则f置0。表示不能取出
            f = (A[x + i - 1][y + j - 1] == B[i][j] && Visited[x + i - 1][y + j - 1] == 0) ? f : 0;
    return f;
}
void F1(int x, int y) {         //该函数的功能是取出小矩阵，将相应位置的Visited置为1
    for (int i = x; i <= x + n - 1; i++)
        for (int j = y; j <= y + m - 1; j++)
            Visited[i][j] = 1;
}
int main() {
```

```
    scanf_s("%d%d", &N, &M), Input(A, N, M), scanf_s("%d%d", &n, &m), Input(B, n, m);   //输入
    for (int i = 1; i <= N - n + 1; i++)          //从大矩阵的(1,1)开始遍历
        for (int j = 1; j <= M - m + 1; j++)
            if (F(i, j))      //如果能取出，就取出小矩阵，并累加数量
                F1(i, j), ans++;
    printf("%d", ans);          //输出最多取出的数量
    return 0;
}
```

字符串，可以做做浙大 pat 乙级 1076

2018 年第 3 题

题目描述

瓜农王大爷去年种西瓜赚了不少钱。看到收入不错，今年他又重新开辟了 n 个西瓜地。

为了能给他的 n 个西瓜地顺利的浇上水，对于每个西瓜地他可以选择在本地打井，也可以修管道从另一个瓜地（这个瓜地可能打了井；也可能没打井，它的水也是从其它瓜地引来的）将水引过来。

当然打井和修管道的费用有差别。已知在第 i 个西瓜地打井需要耗费w_i元，在第 i、j 个西瓜地之间修管道需要耗费$p_{i,j}$元。

现在的问题是：王大爷要想使所有瓜地都被浇上水，至少需要花费多少钱（打井与修管道的费用和）？

由于瓜地较多，王大爷无法选择在哪些（个）瓜地打井，哪些西瓜地之间修管道。

请你编程帮王大爷做出决策求出最小费用。

输入格式

第 1 行，一个正整数 n，代表西瓜地的数量。

第 2 行，依次给出整数w_1、w_2…w_n（每块西瓜地的打井费用），两个数之间有一个空格隔开

紧接着是一个 n*n 的整数矩阵，矩阵的第 i 行第 j 列的数代表$p_{i,j}$（两块西瓜地之间建立管道的费用），每行的两个数之间有一个空格隔开

输出格式

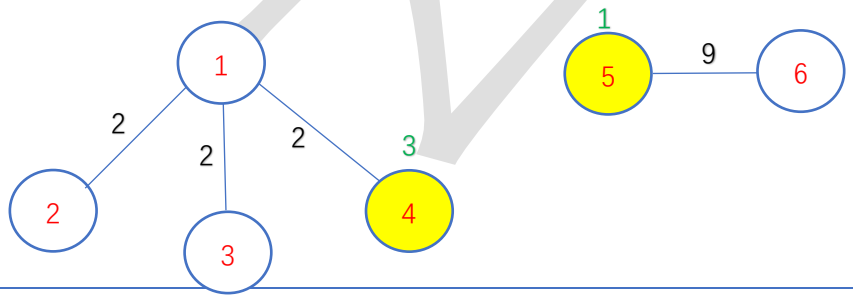
一个正整数，所求的最小花费

数据规模和约定

对于所有数据，1<=N<=300，1<=w_i<=100000，$p_{i,i}$=0，1<=$p_{i,j}$<=100000

输入 1：	输出 1：	输入 2：	输出 2：	输入 3：	输出 3：
6	19	6	18	1	1
5 4 4 3 1 20		10 4 5 3 1 20		1	
0 2 2 2 9 9		0 2 2 2 9 8		0	
2 0 3 3 9 9		2 0 3 3 9 9			
2 3 0 4 9 9		2 3 0 4 9 9			
2 3 4 0 9 9		2 3 4 0 9 9			
9 9 9 9 0 9		9 9 9 9 0 9			
9 9 9 9 9 0		8 9 9 9 9 0			

样例 1 说明：



在第 4 个瓜地打井（费用为 3），然后将第 2、3、4 个瓜地与第 1 个瓜地间修管道（费用分别是 2、2、2），这样水可以经过管道从 4 流向 1，然后经 1 再流向 2、3；

在第 5 个瓜地打井（费用为 1），5 和 6 之间修管道（费用为 9）。

这样一共打了 2 口井，修了 4 条管道，能给所有的 6 个瓜地浇水，费用是：3+2+2+2+1+9=19。

这道题目要明白几点：

①如果一个西瓜地通过修管道的方式得到水，那么所修的管道一定是和它相连的一条耗费最小的。

②如果一个西瓜地的一条修管道费用比打井费用高，那么这条管道一定不修

这道题的解题思路参考 https://blog.csdn.net/dmgy110/article/details/50637401，也就是说这道题目是抄来的。但是我在考场的时候就没做出来，很多人都只写了思路，代码没写完整。我问了很多高分选手，都是没能完全写出来。

现在说一下我的解题思路：

①将西瓜地按照打井费用从低到高排序

②选择打井费用最低的西瓜地，如果该地还未通水，就直接打井

③每当一块地 i 通水后，便要判断是否可以引水出去，需要通过两层判断。第一层，假设可以引水到西瓜地 j，那么修管道费用$p_{i,j}$必须不大于打井费用w_j。满足第一层判断之后，就要进行第二层判断，在与西瓜地相连的修管道费用中，如果$p_{i,j}$是最小的，那么才可以修此管道。

重复步骤 2、3，直到所有地都通水，参考代码如下：

```
#include<iostream>
#include<algorithm>
using namespace std;
struct POINT {        //POINT储存顶点信息，index标记顶点号，weight表示打井费
    int index, weight;
}P[300];
int n, E[301][301], Cost[301], V[301], ans;      //E存储打井费，Cost记录每块地通水的花费，V记录每块地打井的费用
bool cmp(POINT A, POINT B) {   //按照打井费从低到高排序，打井费相同的情况下，按照顶点号从高到低排序
    return A.weight == B.weight ? A.index < B.index : A.weight < B.weight;
```

```
}
void DFS(int p) {     //每当一块地有了水源，就判断是否需要将该地的水通过修管道引到其它地
    for (int i = 1; i <= n; i++)        //从1号地开始遍历
        if (i != p && E[i][p] <= V[i] && Cost[i] == 0) {     //寻找修管道的费用比打井费用低的一块未通水地
            bool f = 1;       //标记符号，记录是否能够修管道
            for (int j = 1; j <= n; j++)        //如果这条管道花费最小，就修该管道
                f = E[i][j] < E[i][p] && i != j ? 0 : f;
            if (f)        //如果能修该管道，则计算总费用，输出，由于i号地有了水，就继续遍历，判断能否引水出去
                ans += Cost[i] = E[i][p], printf("%d号与%d号修管道，花费%d\n", i, p, Cost[i]), DFS(i);
        }
}
int main() {
    scanf_s("%d", &n);              //输入西瓜地的数量
    for (int i = 0; i < n; i++)     //输入顶点信息，方便排序
        P[i].index = i + 1, scanf_s("%d", &P[i].weight), V[i + 1] = P[i].weight;
    sort(P, P + n, cmp);       //按照打井费从低到高排序，打井费相同的情况下，按照顶点号从高到低排序
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            scanf_s("%d", &E[i][j]);   //输入管道费
    for (int i = 0; i < n; i++)             //从打井费最低的地开始遍历
        if (Cost[P[i].index] == 0)      //如果该地没有水源，直接打井，计算总费用，判断能否引水出去
            ans += Cost[P[i].index] = P[i].weight, printf("%d号打井，花费%d\n", P[i].index, P[i].weight), DFS(P[i].index);
    printf("最小花费%d", ans);          //输出结果
    return 0;
}
```

代码运行结果

输出 1：
5 号打井，花费 1
6 号与 5 号修管道，花费 9
4 号打井，花费 3
1 号与 4 号修管道，花费 2
2 号与 1 号修管道，花费 2
3 号与 1 号修管道，花费 2
最小花费 19

输出 2：
5 号打井，花费 1
4 号打井，花费 3
1 号与 4 号修管道，花费 2
2 号与 1 号修管道，花费 2
3 号与 1 号修管道，花费 2
6 号与 1 号修管道，花费 8
最小花费 18

输出 3：
1 号打井，花费 1
最小花费 1

四、复试面试备考

大概流程

面试之前，很多人说要将奖状、成绩单之类的打印几份，但是我觉得不用，因为老师根本就没看，一直在我手上，怎么拿进去就怎么拿出来。只要准备个人陈述表（8 份足够了）和复试登记表即可。这次面试我是被安排在了第一组。先说一下我们这组的面试流程。进去先来一段中文的自我介绍，然后问几个专业课问题。接着就开始问几个英语问题，最后以英语自我介绍结束。

专业课问答

如果本科是计算机专业，那么本科学的专业课都会问。如果是跨考生，老师会问你熟悉的一门，所以只要准备一门课就可以了。我是跨考生，我就准备了数据结构。我准备的问题如下：

1. 顺序存储和链式存储的优缺点比较？
(1)顺序存储

存储密度大，存储空间利用率高
插入或删除元素不方便
适合查找等静态操作
(2)链式存储
存储密度小，存储空间利用率低。
插入或删除元素方便、灵活
适合动态操作

2. 最小生成树两种算法
(1)最小生成树
一个连通图的极小连通子图就是生成树，在这些生成树中边权之和最小的就是最小生成树
(2)Prime 算法
每次在顶点集 U，（V－U）之间找一条权值最小的边，将相应的顶点加入 U。
(3)Kruskal 算法
将所有边按照边权排序，每次找一条权值最小的边，同时保证加入的边不产生环。

3. hash
(1)hash 函数
把查找表中的关键字映射成该关键字对应的地址。
一定长度的输入，通过散列算法，变成固定长度的输出，也就是说定义域比值域范围更广。
构造方法：
直接定址法：取关键字的某个线性函数值为散列地址
除留余数法：取一个不大于表长但最接近表长的质数 p，然后将关键字对 p 取余
平方取中法：取关键字的平方值的中间几位作为散列地址
数字分析法、折叠法
(2)哈希表遇到冲突时怎么解决

①线性探测法
当遇到冲突时，顺序查看表中下一个元素，直到找出一个空闲单元。
②平方探测法
增量序列为…
③拉链法
把所有同义词储存在一个线性表中

4. 排序算法
(1)直接插入排序
每次将一个元素插入到有序的序列里面，保证插入后有序。直到整个数据都有序
时间复杂度：最好 n，平均n^2，最坏n^2，稳定

(2)冒泡排序
①从列表的第一个元素开始比较，若某个元素大于下个元素，则交换。
②重复 1 号步骤，直至再也不能交换。
时间复杂度：最好 n，平均n^2，最坏n^2，稳定

(3)简单选择排序：
①从第 1 个元素到最后一个元素中选择一个最小的元素，与第 1 个元素交换
②从第 2 个元素到最后一个元素中选择一个最小的元素，与第 2 个元素交换
…
时间复杂度：最好n^2，平均n^2，最坏n^2，不稳定

(4)希尔排序
将列表按照步长 d 分割成子表，分别进行直接插入排序，当整个表基本有序时，再进行一次直接插入排序。

(5)快速排序
首先通过一趟排序将待排序表分成两部分，使得前半部分的元素都比后半部分的元素都要小，而后分别递归地对两个子表重复上述过程，直到每部分内只有一个元素或空为止。
时间复杂度：最好$n\log_2 n$，平均$n\log_2 n$，最坏n^2，不稳定

(6)堆排序
建堆，调整堆，堆排序
时间复杂度：最好$n\log_2 n$，平均$n\log_2 n$，最坏$n\log_2 n$，不稳定

(7)2 路归并排序
首先将 n 个元素看成是 n 个有序的子表，每个子表的长度为 1，每次将两个有序表合成一个新的有序表。
时间复杂度：最好$n\log_2 n$，平均$n\log_2 n$，最坏$n\log_2 n$，稳定

5. 拓扑排序
(1) 选择一个入度为 0 的顶点并输出
(2) 从网中删除此顶点及所有出边
循环结束，如果还有顶点未输出，则有回路。

6. 线索二叉树
二叉链表表示的二叉树中存在大量的空指针，利用这些空链域存放指向其直接前驱或后继的指针，这样就变成了线索二叉树，目的是加快查找结点前驱和后继的速度。

7. 图的储存
(1)邻接矩阵法
用一个一维数组存储图中的顶点信息，用一个二维数组存储图中边的信息。
(2)邻接表法

每个顶点建立一个单链表
(3)十字链表
有向图
(4)邻接多重表
无向图

8. 图的遍历
(1)广度优先遍历
类似于二叉树的层序遍历，首先访问起始顶点 v，然后依次访问与 v 相连的顶点 w1、w2、w3，再依次访问与 w1 相连的顶点。
(2)深度优先遍历
类似于二叉树的先序遍历，首先访问起始顶点 v，然后访问与 v 相连的一个顶点 w，再访问与 w 相连的一个顶点。当不能再向下访问时，退回到最近被访问的顶点。

9. 算法
(1)概念
对特定问题求解步骤的描述
(2)5 个特性
有穷性、确定性、可行性、输入、输出
(3)所用的算法
①贪心算法
只考虑当前最优，而不考虑整体
②排序
…

10. 二叉树和度为 2 的树有什么区别
二叉树可以为空，度为 2 的树至少有一个结点有两棵子树

二叉树是有序树，度为 2 的树是无序的

11. 数据结构与算法与 c 语言的关系
数据结构是相互之间存在一种或多种特定关系的数据元素的集合，算法是对特定问题求解步骤的一种描述，一个算法的设计与实现依赖于所采用的数据结构。C 语言只是一个工具，类似还有 C++、JAVA、go 语言等等，算法和数据结构在不同语言之间是相通的

12. 树和图的区别
树可以是空树，但图不可以是空图
树中的元素存在一对多的关系，图中的元素存在多对多的关系
树没有环，图可以有环
树是连通的，图有不连通的情况
树是一种特殊的图

13. 树的遍历
树：先根遍历，后根遍历
二叉树：先序遍历，中序遍历，后序遍历，层序遍历

14. 图的遍历与树的遍历区别
二叉树的先序、中序、后序遍历都是 DFS，而层序遍历是 BFS
树的遍历不需要设置 VIS 数组来标记是否访问过
树的遍历一般以根节点为起点，而图的遍历可以以任意一点为起点
图有不连通的情况

面试问我的问题是：
1、数据结构在电子信息工程方面的应用
2、数据结构在现实生活中的应用

3、简要描述一下你所做的毕业设计
问题都没有准备到，当时非常慌乱，没答上来。可能这就是差距吧，本事嘛也没有，问题也答不上，复试垫底无疑。😭

英语口语

英语口语就问了一个问题：用英语描述一下你的编程技能。没准备到，也没答上来，简直崩溃。最后背了一下自我介绍就出来了。英语口语可以淘宝一下，几块钱的东西，里边有自我介绍，一些常用问答，应该会有帮助。🤦

五、结束语

历经 1 年多的考研之旅终于结束了，初试靠坚持不懈的努力，相信大家都能取得好的成绩。复试能得多少分还是要靠实力说话，所以千万别小看复试。我个人感觉自己对于这场考试已经是尽力了，无论是初试还是复试，我都是非常努力的。这次考研我没有留下遗憾，希望学弟学妹们也一样，不要给自己留下遗憾，所以抓紧时间刷题吧。我花了整整 1 天时间整理出来的这些东西，希望对大家有所帮助。

2018 年 4 月 6 日