



计算机相关专业复试指导
2020第一版 皮皮灰



计算机/软工相关专业 考研复试指导

2020 第一版

- 1.机试百题（取自 34 所高校近年真题）
- 2.面试指南（专业面&英语面一本解决）
- 3.冲！

主编：灰灰



前言

首先皮皮灰恭喜大家顺利通过初试的独木桥，进入复试！

复试的成绩的好坏直接决定了你能否考上研究生以及学业奖学金的等级（第一年的学业奖学金通常由考研总成绩决定）。现在离成功只有一步之遥，我们切不可掉以轻心：

（1）复试占比区间一般在 50%-20%之间。

（2）复录比一般为 1.2（12 个人进复试刷掉 2 人），但热门学校复录比皮皮灰看到最高的可以达到 4: 1，另国家规定必须差额复试，这就意味着总会有人被刷掉。

（3）非全日制大行其道，导致分数线一降再降。学校变了法子的让尽可能多的人参加复试，推荐分数不够的人调剂非全日制，甚至分数低的组面试一进去就问你愿不愿意调剂非全日制。

（4）100%的学校都有着复试成绩不合格不予录取的规定。

复试流程通常由体检（查血、胸透）、政治审查（不计入总成绩）、心理测试（不计入总成绩）、机试（部分学校没有该项）、笔试、面试组成，只要有一项不合格均算不合格。

本书针对复试的各个环节进行全方法的复习指导规划，意在给各位小皮皮提供一个精准有方向的复试准备。

（1）建议复习开始准备时间：1 月初

（2）参加复试前需要准备的物品：

外省的同学建议提前预定宾馆，规划好交通路线。

应届生	准考证 考完试别丢了	有效身份证件	复试志愿申请表
	学生证	本科成绩单 需要有学校的章	政审表 辅导员能解决
	1 寸照片若干 体检用	现金 200+ 复试要交现金的	
往届生	准考证 考完试别丢了	有效身份证件	复试志愿申请表
	毕业证和学位证书 原件，复印件	本科成绩单 需要有学校的章	政审表 档案所在地盖章
	1 寸照片若干 体检用	现金 200+ 复试要交现金的	

(3) 体检相关

体检主要查血、胸透和血压。

查血：查转氨酶，肝好不好，一查便知。

如果转氨酶超出正常范围过多，需要在学校指定的学院复查。

对自己肝不放心的同学，可以事先吃药降酶（小葵花护肝片）。

关于乙肝的问题：国家规定，学校不得以学生有乙肝为依据拒绝入学。

胸透：主要查有没有阴影。

血压：高血压什么自己控制点==。

对于有重大疾病影响入学的同学，可以申请休学，学校保留一年入学资格。

(4) 心理测试

通常是在微信上回答几百个选择题，别乱选就好。

(5) 机试

建议使用语言：C++

如果评分的是老师，一般会给你几个测试用例，看你程序输出的结果。时间复杂度与空间复杂度通常不在考虑的范围之内，只考虑算法的正确性。

编程完成后不要急着叫老师评分，花点时间对每个变量与关键点写注释，评分过程中老师很可能会看你的源程序，这有可能成为扣分的点。如果实在想不出来，**切记不要交空的东西**，最后给老师看一份写满注释的程序也是会有辛苦分的。

如果评分的是 OJ 系统，请尽可能给出最优解提交。

本书前篇为**机试指导**：由上机题要点、C++在上机中的应用，机试精选 100 题（题目精选自 34 所 985 高校近年试题）组成。

(6) 面试

综合面试在 20 分钟左右（部分学校会严格计时）

面试的老师人数为 5 人以上，通常包含专业面试老师、英语面试老师、面试记录人员等。

通常的顺序为英语面试->专业面试

按教育部要求，所有考生面试将进行全程录像录音。

参加面试时，考生可提供反映能力与水平的获奖证书、各类证明等相关材料。

几乎所有的学校都存在着：面试不及格不予录取。

本书后篇为**面试指导**：包含家常扯淡篇，英语面试篇，专业面试常考题目汇总。

只要不打老师基本都是稳的

(7) 政治审查表

XX 大学 2020 年研究生复试录取政审表

姓 名	皮皮灰	性 别	皮	
民 族	汉	出生日期	1997/12/06	
政治面貌	群众	入党时间	无	
宗教信仰	无	联系电话	/	
报考院系	计算机学院	Email	/	
本科毕业时间、院校及专业	2020 年 XX 大学软件工程专业			
硕士毕业时间、院校及专业	无			
是否拥护党的路线方针政策	<input checked="" type="checkbox"/> 是 <input type="checkbox"/> 否			
是否参加过非法组织或活动	<input type="checkbox"/> 是 <input checked="" type="checkbox"/> 否			
是否受过违法违纪处分	<input type="checkbox"/> 是 <input checked="" type="checkbox"/> 否			
<p>思想政治表现自述：</p> <p>我在大学本科学历学习期间用心上进，遵纪守法，无任何违法违纪行为，品学兼优，多次获得各类奖项，诚实守信，没有任何不良行为；担任过 XXX 等职务，对社会工作尽职尽责，并于 XX 年加入了光荣的中国共产党。我期望 XX 大学这个人才荟萃的大熔炉里，能够继续创造佳绩，早出成绩、早日成才，早日为国家和社会贡献自己的才智和力量。</p> <p style="text-align: right;">签 名：皮皮灰 2020 年 3 月 3 日</p>				
以下由单位填写（暂无工作学习单位的，由户籍所在地村委、居委会党组织或档案保管单位填写）				
<p>思想政治品德表现情况：</p> <p>该生在各方面表现突出，思想上用心进取，认真学习马克思列宁主义、毛泽东思想、邓小平理论，政治思想觉悟高，学习刻苦发奋，成绩优秀。曾获二等奖学金、“三好学生”称号，用心参加各项活动，全面发展，用心参加社会实践活动，群众基础较好。</p>				
以上政审情况属实。				
负责人签名：XXX		党组织公章：		2020 年 3 月 3 日

备 注： 请用黑色水笔填写政审表内容，以便今后存档，而且所有栏目不能为空，没有则填“无”；

目录

1. 机试篇	1
1.1 推荐使用 C++	1
1.1.1 基础篇	1
1.1.2 常见算法问题	3
1.2 机试 100 题	7
1.2.1 中南大学上机题	7
1.2.2 北京理工大学上机题	13
1.2.3 东华大学上机题	18
1.2.4 福州大学复试上机题	23
1.2.5 杭州电子科技大学上机题	29
1.2.6 华东师范大学上机题	35
1.2.7 兰州大学上机题	39
1.2.8 西北工业大学上机题	42
1.2.9 中国科学技术大学上机题	46
1.2.10 重庆大学上机题	53
1.2.11 安徽大学上机题	56
1.2.12 北京师范大学上机题	60
1.2.13 厦门大学上机题	63
1.2.14 西北大学上机题	66
1.2.15 哈尔滨工业大学上机题	79
1.2.16 西安电子科技大学上机题	85
1.2.17 中国海洋大学上机题	91
1.2.18 华中科技大学上机题	99
1.2.19 武汉大学上机题	108
1.2.20 上海交通大学上机题	114
1.2.21 北京航空航天大学机试题	121
1.2.22 复旦大学机试题	128
1.2.23 吉林大学上机题	133
1.2.24 南京理工大学上机题	138
1.2.25 清华大学上机题	142

1.2.26 中国人民大学上机题	146
1.2.27 西安电子科技大学上机题	149
1.2.28 西安交通大学上机题	161
1.2.29 西北农林科技大学上机题	170
1.2.30 浙江大学上机题	174
2. 综合面试	181
2.1.家常扯淡篇	182
2.2 专业面试常考问题汇总	191
2.2.1 C 语言	192
2.2.2 数据结构	196
2.2.3 操作系统	204
2.2.4 计算机网络	214
2.2.5 计算机组成原理	228
2.2.6 数据库	235
2.2.7 软件工程	246
2.2.8 软件测试	253
2.2.9 面向对象	259
2.2.10 UML	262
2.2.11 离散数学	267
2.2.12 编译原理	272
2.2.13 机器学习概论	283
2.2.14 其他问题	287

1. 机试篇

1.1 推荐使用 C++

为什么选择 C++? 相比于 c 或者 Java 有什么好处?

(1) 在一般的情况下, C++ 一般比 Java 的运行速度快上许多, 若某些学校的程序有 Limit time 的限制, C++ 是一个不错的选择;

(2) C++ 简单而功能强大, 有些东西用 C++ 实现起来更为方便, 如指针的操作。

(3) 现在对 C++ 提前了解会对以后的工作有好处, 以后去公司上手也会比较快。

(4) C/C++ 的编辑器很多, 且支持性较好, 若程序中存在 bug, 可以借助编辑器强大的 DeBug 功能来快速解决问题。

(5) 学会 C/C++ 以后的工作选择的可能性更多, 可以去搞硬件, 也可以去取搞软件。

C++ 相对于 C, 可以调用更多的函数, 比如常见的 sort 函数, 调用可以直接对所想要的数组进行排序, 而不用自己去实现底层; 同时 C++ 引入了类和对象的概念, 采用封装、继承、多态, 程序的可用性, 安全系更好。

1.1.1 基础篇

1. C++ 版 Hello World!

```
#include <iostream>
using namespace std;
int main(void){
    cout << "Hello World!" << endl;
    return 0;
}
```

2. 基本的输入输出

1) 常用的输入输出函数

C++ 中输入输出首先要加上头文件 `#include <iostream>`, 至于 `using namespace std;` 就不用管它的意思, 当成固定格式写上就行了。

注: 在 C++ 中也可以使用 C 语言中的库函数, 但是写法略有不同。

例如: 在 C 语言中调用 `math.h`, `#include <math.h>`, 而在 C++ 中要调用时的写法是 `#include <cmath>`。基本上就是去掉 `.h` 后缀, 在前面加上 `c` 即可。

C++ 中的标准输出函数:

例: `cout << x1 << x2 << endl;` // `x1`、`x2` 是变量名, `endl` 表示一行结束, 会进行换行

通过 `cout` 可以输出变量、常量等, 而不用像 C 语言中 `printf()` 一样写一长串。

C++ 中的输入函数:

例: `cin >> x1 >> x2;` // 输入变量 `x1`、`x2`

注: 当进行字符串读入时, 若要连空格一起读入要采用以下方式:

a) `string str;`

`getline(cin, str);`

b) `char str2[1024];`

`cin.getline(str2, 1024);` // 第二个参数 1024 为要读取的字符串的长度

2) 带格式控制符的输出函数

举例介绍:

```
#include <iostream>
#include <iomanip>
using namespace std;
void main() {
    float f1 = 2.99;
    float f2 = 8.90909;
    int i = 10;
    cout << setfill('*');
    cout << setw(10) << f1 << endl;
    cout << setw(10) << f2 << endl;
    cout << setw(10) << i << endl;
}
```

其中 `setw` 是设置域宽，就是输出数据所占几列，如果在默认的情况下，输出的数据不能达到所规定的域宽就用空格填充，当然也可以用 `setfill` 来设置填充物，如上例中设置的就是用 “*” 来填充空的列。在使用操纵符时，需要包含头文件 `iomanip`。

需要注意的是，所有的操纵符除了 `setw` 之外，对流的影响都是永久的，除非用户再次进行设置。而 `setw` 默认是 `setw(0)`，每次使用完 `setw` 之后，域宽又被重新置为 0，这就需要在每一次使用时都进行设置。

如上例中，3 次输出就设置了 3 次，而 `setfill` 却只设置了一次。`setw` 还有一点需要说明，如果 `setw` 设置的域宽小于实际数据的域宽时，仍然按照数据的实际宽度来显示

补充:

`setprecision` 用来设置浮点数的精度，默认精度为 6 位。

`left` 和 `right` 用来控制左对齐和右对齐。

`showpoint` 用来强制显示小数点及全部尾部的 0。

`fixed` 强制浮点数以定点格式输出。

这部分读者可以自己根据编辑器尝试得到结果。

说明：如果对于 C++ 的带格式输出函数不熟悉的话，那么也可以直接采用自己熟悉的 C 的输出函数，如 `printf(“%5d”, u)` 等照样能够得到想要的答案，皮皮灰的观点就是不惜一切的得分。

3. 结构体

结构体在 C 语言中就有的，但是前面 C 语言笔试部分中没有进行介绍。

结构体定义方式:

```
struct Student{
    int id;
    char name[20];
};
```

定义结构体变量:

```
struct Student s;
```

更常见的一种方式是使用 typedef 给结构体起一个别名，使操作更方便：

```
typedef struct Student{
    int id;
    char name[20];
}Student;
```

定义的时候就可以更简洁了：

```
Student s;
```

4. 关于字符串

在 C 语言中，字符串的结束标志是 '\0'，我们可以通过下标来逐一访问字符串中的字符：

```
int i = 0;
while(str[i] != '\0'){
    cout << str[i] << endl;
}
```

5. 关于数字和字符之间的转换

字符都是以 ASCII 码进行存储的，'0'到'9'的 ASCII 码是逐个递增的，将数字字符和'0'作差就可以得到对应的数字。同理，将'0'加上对应的数字，就可以得到该数字对应的字符。

例如，将字符 '7' 转换成数字 7：

```
char c = '7';
int num = c - '0';
```

例如，将数字 5 转换成字符 '5'：

```
int num = 5;
char c = '0' + 5;
```

1.1.2 常见算法问题

1. 排序问题

排序的算法有多种，复习数据结构的时候大家肯定也掌握了很多种，因为在大多数机试中并没有时间的要求，只要能运行出正确的结果即可，所以这里就以最简单的冒泡排序为主。

冒泡排序：(以非递减为例)

```
for(int i = 0; i < n; i++){    //第 i 趟排序找出第(i+1)大的元素
    for(int j = 0; j < n-i-1; j++){
        if(num[j] > num[j+1]){
            //交换第 j 和第 j+1 个元素
        }
    }
}
```

2. 进制转换

将十进制转换成二进制是最常见的了，这里以十进制转换成 8 进制为例：

```
char result[100]; //结果通过字符数组来存储，存储是逆序的
int length = 0; //记录结果的长度
while(num/8 != 0){
    result[length] = num%8;
    length++;
    num /= 8;
}
result[length] = num;
length++;
```

3. 质数的判定

1) 直观判断法

最直观的方法，根据定义，因为质数除了 1 和本身之外没有其他约数，所以判断 n 是否为质数，根据定义直接判断从 2 到 $n-1$ 是否存在 n 的约数即可。C++代码如下：

```
bool isPrime_1( int num ){
    int tmp =num- 1;
    for(int i= 2;i <=tmp; i++)
        if(num %i== 0)
            return 0 ;
    return 1 ;
}
```

2) 直观判断法改进

上述判断方法，明显存在效率极低的问题。对于每个数 n ，其实并不需要从 2 判断到 $n-1$ ，我们知道，一个数若可以进行因数分解，那么分解时得到的两个数一定是一个小于等于 \sqrt{n} ，一个大于等于 \sqrt{n} ，据此，上述代码中并不需要遍历到 $n-1$ ，遍历到 \sqrt{n} 即可，因为若 \sqrt{n} 左侧找不到约数，那么右侧也一定找不到约数。C++代码如下：

```
bool isPrime_2( int num ){
    int tmp =sqrt( num);
    for(int i= 2;i <=tmp; i++)
        if(num %i== 0)
            return 0 ;
    return 1 ;
}
```

4. 常见的字符串函数介绍

函数名: strcpy

功 能: 将参数 src 字符串拷贝至参数 dest 所指的地址

用 法: char *strcpy(char *dest, const char *src);

返回值: 返回参数 dest 的字符串起始地址

说 明: 如果参数 dest 所指的内存空间不够大, 可能会造成缓冲溢出的错误情况, 在编写程序时需特别留意, 或者用 strncpy() 来取代;

实例:

```
#include <stdio.h>
#include <string.h>
int main(void) {
    char string[10];
    char *str1 = "abcdefghi";

    strcpy(string, str1);
    printf("%s\n", string); // 输出: abcdefghi
    return 0;
}
```

函数名: strcat

功 能: 字符串拼接函数

用 法: char *strcat(char *dest, const char *src);

返回值: 返回 dest 字符串起始地址

说 明: strcat() 会将参数 src 字符串复制到参数 dest 所指的字符串尾部;

dest 最后的结束字符'\0' 会被覆盖掉, 并在连接后的字符串的尾部再增加一个'\0';

dest 与 src 所指的内存空间不能重叠, 且 dest 要有足够的空间来容纳要复制的字符串;

实例:

```
#include <string.h>
#include <stdio.h>
int main(void) {
    char destination[25];
    char *blank = " ", *c = "C++", *Borland = "Borland";
    strcpy(destination, Borland);
    strcat(destination, blank);
    strcat(destination, c);
    printf("%s\n", destination); // 输出: Borland C++
    return 0;
}
```

函数名: strcmp

功 能: 字符串比较

用 法: `int strcmp(const char *s1, const char *s2);`

返回值: 根据 ASCII 码比较, 若参数 s1 和 s2 字符串相同则返回 0, s1 若大于 s2 则返回大于 0 的值, s1 若小于 s2 则返回小于 0 的值

说 明: 它是区分大小写比较的, 如果希望不区分大小写进行字符串比较, 可以使用 `stricmp` 函数

实例:

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

int main(void)
{
    char *a = "aBcDeF";
    char *b = "AbCdEf";
    char *c = "aacdef";
    char *d = "aBcDeF";
    printf("strcmp(a, b) : %d\n", strcmp(a, b)); // 输出: 1
    printf("strcmp(a, c) : %d\n", strcmp(a, c)); // 输出: -1
    printf("strcmp(a, d) : %d\n", strcmp(a, d)); // 输出: 0
    return 0;
}
```

函数名: strlen

功 能: 计算指定的字符串 s 的长度, 不包括结束字符 '\0'

用 法: `size_t strlen(const char *s);`

返回值: 返回字符串 s 的字符数

说 明: `strlen()` 函数计算的是字符串的实际长度, 遇到第一个 '\0' 结束;

如果你只定义没有给它赋初值, 这个结果是不定的, 它会从首地址一直找下去, 直到遇到 '\0' 停止;

`sizeof` 返回的是变量声明后所占的内存数, 不是实际长度, 此外 `sizeof` 不是函数, 仅仅是一个操作符, `strlen()` 是函数;

实例:

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

int main() {
    char str[5] = "abcd";
    printf("strlen(str)=%d, sizeof(str)=%d\n", strlen(str), sizeof(str));
    // 输出: strlen(str)=4, sizeof(str)=5
    return 0;
}
```

1.2 机试 100 题

1.2.1 中南大学上机题

1. Problem Description

大家都关心考试的难易程度。K 老师出题有一个规律，在出题之前，他会随机写下一个字符串，只要在这个字符串中能按顺序找到 E, A, S, Y 四个字母，他出题就会比较简单。你拿到了字符串，请你告诉别人题目难不难吧。

Input

输入的数据有多组，每组占一行，由一个字符串组成（字符串的长度不超过 1000）。

Output

对于每组输入数据，输出一行，对应一个要求的答案（题目简单就输出 easy，难就输出 difficult）

Sample Input

```
eAsy
SEoAtSNY
```

Sample Output

```
difficult
easy
```

算法思想：由于要比较的内容已经确定为EASY，那么可以另外设计一个辅助字符串，里面存储的即是这四个字母，设计两个指针i, j，指针i指向给定的字符串，指针j指向辅助的字符串，当匹配时，指针i, j均向后移动一位，则

（1）若i指向了给定了字符串的尾部，而j未指向尾部，那么输出difficult；

（2）若j指向了辅助字符串的尾部，则输出easy。

在本题中，我们加入一个输入的字符串的个数来统计随机的个数

Code:

```
#include <iostream>
#include <string>
using namespace std;
bool match(char a[]) {
    int i = 0, j = 0;
    char tmp[] = { 'E', 'A', 'S', 'Y' };
    while (i < 1000 && j < 3) {
        if (a[i] == tmp[j]) {
            i++;
            j++;
        }
        else
            i++;
    }
    if (i >= 1000)
        return false;
```

```
    else
        return true;
}

int main() {
    cout << "请输入字符串的组数" << endl;
    int n = 0;
    cin >> n;
    //输入字符串数组
    cout << "输入目标字符串" << endl;
    char str[][1000] = {'0'};
    for (int i = 0; i < n; i++)
        cin >> str[i];
    for (int i = 0; i < n; i++) {
        if (match(str[i]))
            cout << "easy" << endl;
        else
            cout << "difficult" << endl;
    }
    system("pause");
}
```

运行结果：

```
请输入字符串的组数
2
输入目标字符串
eAsy
SEoASNY
difficult
easy
请按任意键继续
```



2. 在太平洋的一个小岛上，岛民想要建立一个环岛的堤坝，我们可以将小岛简化为一个二维平面，你需要使用 K 条边（这些边要么是水平或者垂直长度为 1 的边，要么是倾斜 45 度的长度为 $\sqrt{2}$ 的边）围城一个多边形，多边形的顶点必须位于整点，然后要让围成的多边形面积最大，你需要求出最大面积是多少。

Input

输入包含多个测试实例，每组实例给出一个数 K ($3 \leq k \leq 2,000,000,000$)

Output

每一组对应一个要求的答案（保留一位小数）

Sample Input

3

4

5

6

Sample Output

0.5

2.0

2.5

4.0

算法思想：不要死死的抓住小岛的面积不放，看清题意怎么围成的。怎样保留一位小数？

点睛 1：抓清题意，看看边的条数与边长的要求；

点睛 2：看看多边形顶点放置的要求。

本题的关键：通过画图的方式，找到边数与所画的最小的三角形的个数之间的关系，是主要考察分析能力，代码较为简单。

$(k-3)/2=a \dots b$

若 $b=0$, 则 $s = (1+4a) * 0.5$

若 $b \neq 0$, 则 $s = (1+4a+3) * 0.5$

Code:

```
#include<iostream>
#include<algorithm>
#include<math.h>
using namespace std;
int main() {
    int k, x, y;
    double s = 0;
    while (scanf_s("%lld", &k) != EOF) {
        if (k < 3) {
            cout << "边数输入错误" << endl;
            break;
        }
        x = (k-3)/ 2;
        y = (k-3)% 2;
        if (y == 0) {
            s = (1 + 4 * x)*0.5;
            cout << "面积最大值为: " ;
```



```

        printf("%.1lf\n", s);
    }
    else {
        s = (1 + 4 * x+3)*0.5;
        cout << "面积最大值为: ";
        printf("%.1lf\n", s);
    }
}
return 0;
}

```

运行结果:

```

3
面积最大值为: 0.5
4
面积最大值为: 2.0
5
面积最大值为: 2.5
6
面积最大值为: 4.0

```

3. Problem Description

鲁大师和他的朋友们经常去一家奇怪的餐厅,为什么说奇怪呢,一是餐厅提供的菜品比较奇怪,二是餐厅的付费规则比较奇怪,每个人有不同的折扣率和折扣上限(单人从总价里折算的最高金额),超过折扣上限的原价付费。这次鲁大师和蔚然晨风以及朋友们一共N个人去这家餐厅吃饭,他们点的菜品总价是T,现在告诉你每个人的折扣率z和折扣上限H请告诉他们最少需要支付多少钱。

Input

输入数据有多组,每组占 N+1 行,第一行是 N 和 T,接下来 N 行,每行两个数字 z 和 H(输入数据保证最后结果为 int 型, $0 < N < 100$)

Output

对于每组输入数据,输出一行,对应一个要求的答案。

Sample Input

```

2 100
0.7 70
0.6 50
3 500
0.6 100
0.8 200
0.7 100
1 100
0.6 100

```

Sample output

```

65
390
60

```

算法思想：设计一个二维数组 a ，第一列 $a[0]$ 存储折扣率，第二列 $a[1]$ 存储折扣上限。将折扣率按从小到大进行排序；下面进行分类讨论：

- (1) 若 $T <$ 最小折扣率的上限，那么最后的价格直接是 $T \times$ 折扣率；
- (2) 若不够，则选择第二小的折扣率来计算，看其折扣上限与第一个相加是否满足 $>T$ ，若满足，则将剩余的以第二小的折扣率付钱，
- (3) 若最后算了所有人的折扣上限加起来小于消费金额 T ，那么重复 (1) (2)，剩余的部分按原价付钱。

Code:

```
#include <iostream>
#include<algorithm>
using namespace std;
int compute(double a[][2],int n, int T){
    //若所有人折扣的金额加起来小于 T
    int sum1 = 0, sum=0, sum2=0;
    for (int i = 0; i < n; i++)
        sum1 += (int)a[i][1];
    if (sum1 > T) {
        for (int j = 0; j < n-1; j++) {
            T -= (int)a[j][1];
            if(T>0)
                sum += (int)(a[j][0] * a[j][1]);
            int i = j + 1;
            if(i<n&&T<a[i][1])
                sum += (int)(T * a[i][0]);
        }
    }
    else {
        for (int k = 0; k < n; k++)
            sum2 += (int)(a[k][1]*a[k][0]);
        T = T - sum1;
        sum = sum2 + T;
    }
    return sum;
}

int main() {
    cout << "输入总人数" << endl;
    int n = 0;
    scanf_s("%d", &n);
    double a[100][2]; //初始化
    cout << "输入每个人的折扣率和折扣上限" << endl;
    for (int i = 0; i < n; i++) {
        cin >> a[i][0];
        cin >> a[i][1];
    }
}
```

```

    int sum = 0;
//按照折扣率大小进行排序, 在这里选择冒泡排序进行按折扣率大小进行排序,
//即第 1 列进行排序
    for(int i=0;i<n-1;i++)
        for (int j = n - 1; j > i; j--)
            if (a[j - 1][0] > a[j][0]) { //若为逆序则交换
                double tmp = a[j - 1][0];
                a[j - 1][0] = a[j][0];
                a[j][0] = tmp; //交换第一列的值
                double tmp1 = a[j - 1][1];
                a[j - 1][1] = a[j][1];
                a[j][1] = tmp1; //交换第二列的值
            }
    for (int i = 0; i < n; i++) {
        cout << a[i][0] << ", ";
        cout << a[i][1] << endl;
    }
    cout << "请输入消费的总金额 T: ";
    int T = 0;
    cin >> T;
    sum = compute(a, n, T);
    cout << sum << endl;
    system("pause");
    return 0;
}

```

运行结果:

```

输入总人数和消费金额:
2 100
输入每个人的折扣率和折扣上限
0.7 70
0.6 50
65
请按任意键继续. . .

```



1.2.2 北京理工大学上机题

1. 身份证号的校验身份证号码共 18 位，最后一位是校验位 A[18] : aaaaaabbbbbbbcccd 校验的规则是如下

前十七位的权值分别是：W[17]:7 9 10 5 8 4 2 1 6 3 7 9 10 5 8 4 2

$x = (A[0]*W[0] + A[1]*W[1] + A[2]*W[2] + \dots + A[16]*W[16]) \bmod 11$

x 和校验位 y 的对应规则对应如下：

x:0 1 2 3 4 5 6 7 8 9 10

y:1 0 x 9 8 7 6 5 4 3 2

若 y 等于 d 则身份证号码正确

输出格式：aaaaaabbbbbbbcccd 正确

若 y 不等于 d 则身份证号码不正确

输出格式：应为:aaaaaabbbbbbbcccy

测试用例：拿自己身份证实验一下就知道了

例如：52242619811105565x

例如：533325199108247066

需要解决的问题：

问题 1：验证输入的身份证的位数，是否是 18 位

问题 2：身份证校验位的确定

Code:

```
#include <iostream>
using namespace std;
int main() {
    //输入权值数组
    int W[17] = { 7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2 };
    //校验位对照表
    char check[] = { '1', '0', 'x', '9', '8', '7', '6', '5', '4', '3', '2' };
    cout << "请输入您的 18 位身份证号：" << endl;
    char A[17] = { '0' };
    cin >> A;
    //验证输入的身份证号是否为 18 位
    int num = strlen(A);
    //cout << num << endl;
    if (num != 18)
        cout << "身份证号输入不完全，请再次输入。";
    //创建辅助数组，将前 17 位存到辅助数组中
    int B[17] = { 0 };
    for (int i = 0; i < 17; i++)
        B[i] = (int)A[i] - 48;

    //计算前 17 位的权值之和 MOD11
    int sum = 0;
    for (int j = 0; j < 17; j++)
        sum += W[j] * B[j];
```

```
int x = sum % 11;
int y = check[x] - 48;
if (check[x] == A[17]) {
    cout << A;
    cout << " " << "正确"<<endl;
    system("pause");
    return 0;
}
else {
    for (int i = 0; i < 17; i++)
        cout << B[i];
    if (check[x] == 'x')
        cout << 'x' << endl;
    else
        cout << y << endl;
    system("pause");
    return 0;
}
return 0;
}
```

运行结果:

```
请输入您的18位身份证号:
52242619811105565x
52242619811105565x 正确
请按任意键继续. . .
```



2. 二分查找{-36 -25 0 12 14 29 35 47 76 100}, 对上述十个数进行二分查找

测试用例:

请输入要查找的数据: 14 14 是第 5 个数, 查找次数为 1

请输入要查找的数据: -25 -25 是第 2 个数, 查找次数为 2

请输入要查找的数据: 121 查找失败

编者说: 写上机题的时候一定要分析学校的题型是怎么样, 一定要相信, 基础算法也重要, 快速的写出此类题型才能更好的去写难题。

Code:

```
#include <iostream>
using namespace std;
//直接是二分查找的应用
int count_num = 0; //记录查找的次数
int binarySearch(int a[], int n, int key) {
    int low = 0, high = n - 1;
    while (low <= high) {
        count_num++;
        int mid = (low + high) / 2;
        if (a[mid] == key)
            return mid;
        if (a[mid] > key)
            high = mid - 1;
        else
            low = mid + 1;
    }
    return -1;
}

int main() {
    int a[10] = {-36, -25, 0, 12, 14, 29, 35, 47, 76, 100 };
    cout << "请输入需要查找的数: ";
    int num = 0;
    cin >> num;
    int n = binarySearch(a, 10, num);
    if (n >= 0)
        cout << num << "是第" << (n+1) << "个数, 查找的次数为" << count_num << endl;
    else
        cout << "查找失败" << endl;
    system("pause");
    return 0;
}
```

运行结果:

```
请输入需要查找的数: 14
14是第5个数, 查找的次数为1
请输入需要查找的数: 121
查找失败
请按任意键继续. . .
```

3. 建立一个学生信息系统，输入学生信息，输出有挂科同学的信息，再按照平均成绩从高到低排序输出

输入：

5

zhaoyi 70 80 90 240

qianer 65 32 77 174

sunsan 100 55 68 223

lisi 86 77 90 253

wangwu 100 59 66 225

输出：

*[qianer] 65 32 77

*[sunsan] 100 55 68

*[wangwu] 100 59 66

lisi 86 77 90

zhaoyi 70 80 90

wangwu 100 59 66

sunsan 100 55 68

qianer 65 32 77

Code:

```
#include<iostream>
using namespace std;
typedef struct {
    char name[20];
    float score1;
    float score2;
    float score3;
    int total;
}Student;
void swap1(Student s[], int i, int j){
    Student temp = s[i];
    s[i] = s[j];
    s[j] = temp;
}
void find_fail(Student s[],int n) {
    //输出不及格的人数
    for (int i = 0; i < n; i++) {
        if (s[i].score1 < 60 || s[i].score2 < 60 || s[i].score3 < 60)
            cout << "[ " << s[i].name << " ] " << s[i].score1 << " " << s[i].score2
            << " " << s[i].score3 << endl;
    }
    cout << endl;
}
```

```

//按照平均成绩的高低输出
void printByAverage(Student s[],int n) {
    //按照基本的冒泡排序法按照成绩排序
    for (int i = 0; i < n - 1; i++)
        for (int j = n - 1; j > i; j--)
            if (s[j-1].total>s[j].total)//逆序
                swap1(s, j-1, j);
    for (int k = n-1; k >=0; k--) {
        cout << s[k].name << " ";
        cout << s[k].score1 << " ";
        cout << s[k].score2 << " ";
        cout << s[k].score3 << endl;
    }
}

int main() {
    cout << "输入学生的人数: ";
    int n = 0;
    cin >> n;
    Student s[100];//假定最多输入 100 个学生的成绩
    cout << "请依次输入学生的姓名, 第一门课的成绩, 第二门课的成绩, 第三门课的成绩以及总分: " << endl;
    for (int i = 0; i < n; i++) {
        cin >> s[i].name;
        cin >> s[i].score1;
        cin >> s[i].score2;
        cin >> s[i].score3;
        cin >> s[i].total;
    }
    find_fail(s, n);
    printByAverage(s, n);
    system("pause");
    return 0;
}

```

运行结果:

```

输入学生的人数: 5
请依次输入学生的姓名, 第一门课的成绩, 第二门课的成绩, 第三门课的成绩以及总分:
zhaoyi 70 80 90 240
qianer 65 32 77 174
sunsan 100 55 68 223
lisi 86 77 90 253
wangwu 100 59 66 225
*[qianer] 65 32 77
*[sunsan] 100 55 68
*[wangwu] 100 59 66

lisi 86 77 90
zhaoyi 70 80 90
wangwu 100 59 66
sunsan 100 55 68
qianer 65 32 77
请按任意键继续. . .

```


1.2.3 东华大学上机题

1. 写一个程序,该程序的功能是输出 100 到 999 之间的所有水仙花数。水仙花数的特点是:它的每个位上的数字的三次幂之和等于它本身。

例如: $371=3^3+7^3+1^3$,因此 371 是水仙花数。

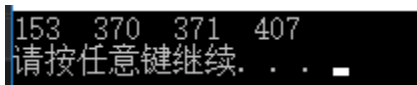
本题为简单题,了解水仙花数的定义即可,然后根据取模、取余运算即可得到结果。

Code:

```
#include <iostream>
using namespace std;
void findWaterFlowerNum() {
    int ge = 0, shi = 0, bai = 0;
    //100~999 的数, 包含头部不包含尾部
    for (int i = 100; i < 999; i++) {
        ge = i % 10;
        int n = i / 10;
        shi = n % 10;
        bai = n / 10;
        if (ge*ge*ge + shi * shi*shi + bai * bai*bai == i)
            cout << i << " ";
    }
    cout << endl;
}

int main() {
    findWaterFlowerNum();
    system("pause");
    return 0;
}
```

运行结果:



```
153 370 371 407
请按任意键继续...
```

2. 在一个递增有序的数组中,有数值相同的元素存在,程序的功能是去掉数值相同的元素,使数组中不再有重复的元素。例如 (7,10,10,21,30,42,42,42,51) 变成 (7,10,21,30,42,51)

算法要求: 尽量优化算法的时间复杂度与空间复杂度。

算法思想: 确定该数组为递增有序。那么只是需要比较相邻两个元素的值即可。在这里我们可以换一种思考方式,可以采用直接插入排序的思想,若要插入的一个元素在数组中存在,则不插入。另一种思考方式是设计一个辅助数组(较常用),设计两个指针 i, j 分别指向原数组和辅助数组,比较原数组中的值和辅助数组中的值,若相同,则 $i++$, 否则将数据插入到辅助数组中,两个指针同时后移,直到遍历完原数组。

两种排序方式比较: 采用直接排序的思想,其时间复杂度为 $O(n^2)$, 空间复杂度为 $O(1)$ 采用增加一个辅助数组的方法,时间复杂度为 $O(n)$, 空间复杂度为 $O(n)$

Code:

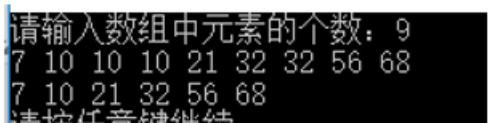
```

#include <iostream>
using namespace std;
//按照第一种方式-直接插入排序的方式来去除同样的元素
void deleteSame1(int a[], int n) {
    int i, j;
    for (i = 0, j = 1; j < n; j++)
        if (a[i] != a[j])
            a[++i] = a[j];
    for (int k = 0; k < (i + 1); k++)
        cout << a[k] << " ";
    cout << endl;
}
//按照方式二来去除数组中的相同的元素，需要借助额外的 n 个空间
void deleteSame2(int a[], int n) {
    int *b = new int[n];
    int i = 0, k = 0;
    //数组 b 中存储的数据为 a 中按从小到大均不相同的数据
    b[0] = a[0];
    while (i < n) {
        if (a[i] == b[k]) {
            i++;
            continue;
        }
        else {
            b[++k] = a[i++];
        }
    }
    for (int j = 0; j <= k; j++)
        cout << b[j] << " ";
    cout << endl;
}

int main() {
    int a[100] = { 0 }; //假设数组中元素的最大值为 100
    int n = 0;
    cout << "请输入数组中元素的个数: ";
    cin >> n;
    //为 n 个元素赋值
    for (int i = 0; i < n; i++)
        cin >> a[i];
    deleteSame1(a, n);
    //deleteSame2(a, n);
    system("pause");
    return 0;}

```

运行结果:



```

请输入数组中元素的个数: 9
7 10 10 10 21 32 32 56 68
7 10 21 32 56 68
请按任意键继续

```

3. 从数据结构中树的定义可知，除根节点外，树中的每个节点都有唯一的一个双亲结点。根据这一特性，可用一组连续的存储空间（一维数组）存储树中的各结点。树中的结点除保存本身结点的信息外，还要保存其双亲结点在数组中的位置（即在数组中的下标。双亲的信息为-1 则表示该结点为根结点），树的这种表示方法称为双亲表示法。

树中的每个结点的数据类型定义如下：

```
typedef struct {
    char data;    //结点数据域
    int parent;    //结点双亲在数组中的位置
} PTNode;
```

树的数据类型定义如下：

```
#define MAX_TREE_SIZE 100
```

```
typedef struct {
    PTNode nodes[MAX_TREE_SIZE];    //存储树中所有的结点
    int n;    //树中的结点数，n 不超过 100
} PTree;
```

则下图所示的树中，按照双亲表示法存储结构，存储为图 b 所示的形式（n 为 10）。

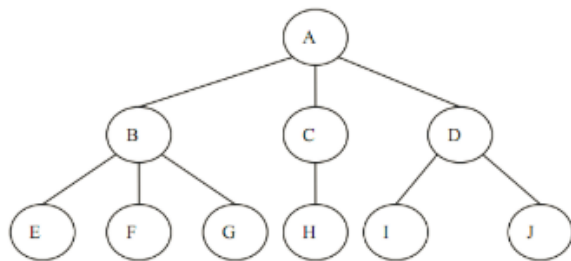


图 a 树的示意图

序号 data parent

0	A	-1
1	B	0
2	C	0
3	D	0
4	E	1
5	F	1
6	G	1
7	H	2
8	I	3
9	J	3

图 b 双亲表示法存储

已知一棵树以存储以上形式，请编写函数 GetLeavesCount，计算叶子结点数目。

GetLeavesCount 的函数原型为：int GetLeavesCount (PTree T)

其中，形参 T 中保存了树中的结点数目以及图 b 所示的结点数组，函数返回叶子结点的数目。

比如：对图 b 的树调用函数 GetLeavesCount (T)，返回结果为 6

输入的第一个数 n 表示树中的结点数，此后有 n 行输入，每行表示一个节点的信息，第一个信息为结点的数据，第二个信息为结点的双亲结点在数组中的位置。

如输入：

```
10
a -1
b 0
c 0
d 0
e 1
f 1
```

g 1
h 2
i 3
h 3

则创建图 b 所对应的树。

对此树调用函数 `GetLeavesCount (T)` ,返回结果为 6

如输入 “

8
a -1
b 0
e 1
h 2
c 0
d 0
f 5
g 5

对此树调用函数 `GetLeavesCount (T)` ,返回结果为 4

Code:

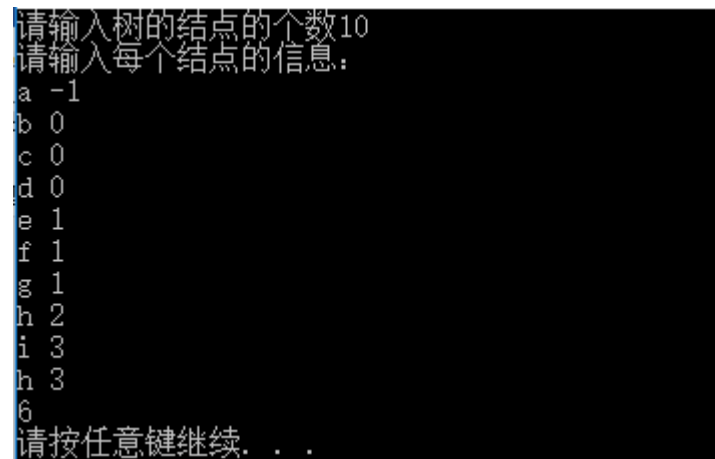
```
#include<iostream>
using namespace std;
#define MAX_TREE_SIZE 100
typedef struct {
    char data;        //结点数据域
    int parent;       //结点双亲在数组中的位置
} PTNode;
typedef struct {
    PTNode nodes[MAX_TREE_SIZE];    //存储树中所有的结点
    int n;        //树中的结点数, n 不超过 100
} PTree;
/*
算法思想: 遍历结点数组, 当该结点的序号有另外一个节点指向时, 那么该结点为非叶节点,
当遍历完结点时, 没有一个结点的指针指向该序号的话, 那么该结点即是叶结点。
*/

//返回树中叶结点的个数
int GetLeavesCount (PTree T) {
    int count = 0; //统计非叶结点的个数, 转换一下思考方式, 否则很难直接统计叶结点个数
    for(int i=0; i<T.n; i++)
        for (int j = 0; j < T.n; j++) {
            if (i == j) //自己指向自己不算
                continue;
            else if (T.nodes[j].parent == i) {
```

```
        count++;
        break;
    }
}
return T.n-count;
}

int main() {
    cout << "请输入树的结点的个数";
    int n = 0;
    cin >> n;
    PTree pt;
    pt.n = n;
    cout << "请输入每个结点的信息: " << endl;
    for (int i = 0; i < n; i++) { // 输入结点的数据域和结点双亲在数组中的位置
        cin >> pt.nodes[i].data;
        cin >> pt.nodes[i].parent;
    }
    int leaves = GetLeavesCount(pt);
    cout << leaves << endl;
    system("pause");
    return 0;
}
```

运行结果:



请输入树的结点的个数10
请输入每个结点的信息:
a -1
b 0
c 0
d 0
e 1
f 1
g 1
h 2
i 3
h 3
6
请按任意键继续. . .

1.2.4 福州大学复试上机题

1. 输入一个数组，然后求出任意给定的区间内数值的和。

Code:

```
#include<iostream>
using namespace std;
int main() {
    cout << "输入数组元素的个数: ";
    int n = 0;
    cin >> n;
    cout << "输入数组的元素: " << endl;
    int *a = new int[n];
    for (int i = 0; i < n; i++)
        cin >> a[i];
    cout << "输入要计算的区间[p, q]" << endl;
    int p = 0, q = 0;
    cin >> p >> q;
    int sum = 0;
    for (int i = p; i < q+1; i++)
        sum += a[i];
    cout << sum << endl;
    system("pause");
    return 0;
}
```

运行结果:

```
输入数组元素的个数: 5
输入数组的元素:
1 2 3 4 5
输入要计算的区间[p, q]
1 3
9
请按任意键继续. . .
```



2. 3. 螺旋矩阵

例如 $n=4$

1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

要求：打印出螺旋矩阵，求 i 行 j 列的数字， $0 < n < 10000$ 算法思想：只要找出每一层的第一个数即可，第一个数值为上一层的第一个数 $+4*n+4$ ，循环时 n 每次减 2

```

+-----> X 轴
|  1  2  3  4
| 12 13 14  5
| 11 16 15  6
| 10  9  8  7
|
Y 轴

```

设元素 1 的坐标为(0,0)，元素 13 的坐标为 (1, 1)，.....，任一元素的坐标为 (x,y)
 亦可以采用递归的方式，借助于一个二维数组，从外圈开始，向内圈打印输出。

Code:

```

#include <iostream>
using namespace std;
//打印出螺旋矩阵
void printSpiralMatrix(int **matrix, int x, int y, int start, int n) {
    if (n <= 0)
        return;
    if (n == 1) {
        matrix[x][y] = start;
        return;
    }
    for (int i = x; i < x + n - 1; i++) //上部
        matrix[y][i] = start++;
    for (int j = y; j < y + n - 1; j++) //右边
        matrix[j][x + n - 1] = start++;
    for (int i = x + n - 1; i > x; i--) //底部
        matrix[y + n - 1][i] = start++;
    for (int j = y + n - 1; j > y; j--) //左边
        matrix[j][x] = start++;
    printSpiralMatrix(matrix, x + 1, y + 1, start, n - 2);
}

```

```
//查找 i 行 j 列的值,按照人们的常识,从 1 开始
int searchNum(int **matrix,int i,int j) {
    return matrix[i - 1][j - 1];
}

int main() {
    cout << "输入螺旋矩阵的行数: ";
    int n = 0;
    cin >> n;
    int **matrix = (int **)malloc(n * sizeof(int *));
    for (int i = 0; i < n; i++)
        matrix[i] = (int *)malloc(n * sizeof(int));

    printSpiralMatrix(matrix,0,0,1,n);

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            printf("%5d",matrix[i][j]);
        cout << endl;
    }
    //这里直接查找第 3 行第四列的数据,也可以自己指定
    int num=searchNum(matrix, 3, 4);
    cout << num << endl;

    system("pause");
    return 0;
}
```

运行结果:

```
输入螺旋矩阵的行数: 4
 1   2   3   4
12  13  14  5
11  16  15  6
10   9   8   7
```



3. 某省调查城镇交通状况，得到现有城镇道路统计表，表中列出了每条道路直接连通的城镇。省政府“畅通工程”的目标是使全省任何两个城镇间都可以实现交通（但不一定有直接的道路相连，只要互相间接通过道路可达即可）。问最少还需要建设多少条道路？

Input

测试输入包含若干测试用例。每个测试用例的第 1 行给出两个正整数，分别是城镇数目 N (< 1000) 和道路数目 M ；随后的 M 行对应 M 条道路，每行给出一对正整数，分别是该条道路直接连通的两个城镇的编号。为简单起见，城镇从 1 到 N 编号。

注意：两个城市之间可以有 multiple 道路相通（允许存在平行边），也就是说

3 3

1 2

1 2

2 1

这种输入也是合法的

当 N 为 0 时，输入结束，该用例不被处理。

Output

对每个测试用例，在 1 行里输出最少还需要建设的道路数目。

Sample Input

测试用例 1:

4 2

1 3

4 3

测试用例 2:

3 3

1 2

1 3

2 3

测试用例 3:

5 2

1 2

3 5

测试用例 4:

999 0

0

Sample Output

结果 1:

1

结果 2:

0

结果 3:

2

结果 4:

998

算法介绍：并查集的应用

并查集是树的简单应用，它支持一下三种操作：

1. $\text{Union}(S, \text{Root1}, \text{Root2})$: 把集合 S 的子集合 Root2 并入到子集合 Root1 中，要求 Root1 和 Root2 互不相交，否则不执行合并。
2. $\text{Find}(S, x)$: 查找集合 S 中元素 x 所在的子集合，并返回该集合的名字。
3. $\text{Initial}(S)$: 将集合 S 中的每一个元素都初始化为只要一个单元元素的子集合

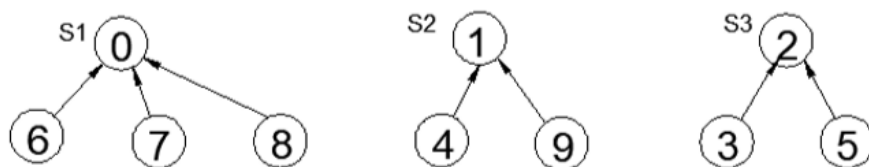
通常用树（森林）的双亲表示法作为并查集的存储结构，每个子集合用一棵树表示。

设存在一个集合 $S = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ ，执行 initial 方法后其存储结构变为：

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

经过计算后，集合变为： $S_1 = \{0, 6, 7, 8\}$ ； $S_2 = \{1, 4, 9\}$ ； $S_3 = \{2, 3, 5\}$ ；

此时并查集的树形表示和存储结构为：



存储结构表示									
0	1	2	3	4	5	6	7	8	9
-1	-1	-1	2	1	2	0	0	0	1

为了得到两个子集合的并，只要将其中一个子集合根节点的双亲指向另外一个集合的根节点即可。此就是 Union 算法的思想。

并查集的模板 code

1. 结构体定义

```
#define SIZE 100
int UFSet[SIZE];
```

2. 并查集的初始化操作

```
void initial(int s[]){
    for(int i=0;i<size;i++){
        s[i]=i;
    }
}
```

3. $\text{find}(x)$ 操作：查找包含元素 x 的树的根

```
int find(int s[],int x){
    while(s[x] != x)
        x=s[x];
    return x;
}
```

4. Union 操作--求两个不相交的集合的并集

```
void union(int s[],int root1,int root2){
    s[root2]=root1;
}
```

Code:

```

#include<iostream>
using namespace std;
int find(int s[],int x) {
    while (s[x]!=x)
        x = s[x];
    return x;
}
void initial(int * a,int n){
    for (int i = 1; i <= n; i++)    a[i] = i;
}
void union1(int s[], int x, int y) {
    int fx = find(s, x), t fy = find(s, y);
    if (fx != fy)    s[fx] = fy;
}
//确定连通分量的个数
int find_ans(int s[], int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++)
        if (s[i] == i) ++sum;
    return sum;
}
int main() {
    cout << "输入城镇的数目和道路数目: ";
    int m = 0, n = 0;
    cin >> n>>m;
    //创建一个双亲表示法的数组
    int *arr = new int[n+1];
    //执行 initial 方法
    initial(arr,n);
    //将集合分类—每个节点都归到自己的根节点上
    int a = 0, b = 0;
    while (n != EOF) {
        if(!n) break;
        initial(arr, n);
        for (int i = 0; i < m; i++) {
            cin >> a >> b;
            union1(arr, a, b);
        }
        printf("%d\n", find_ans(arr, n));
    }
    system("pause");
    return 0;
}

```

运行结果:

```

输入城镇的数目和道路数目: 4 2
1 3
4 3
2

```

1.2.5 杭州电子科技大学上机题

1.关羽过关斩三将，输入四个人的武力值（大于 0 小于 50），若超过界限 需要重新输入，关羽的武力值 x ，将士武力值为 y ，满足 $(x-y)^2+(x-y)+41$ 若为素数则关羽获胜，若关羽三次获胜输出 WIN，若失败则输出 失败的将领序号（第几关）。

本题主要解决两个问题：

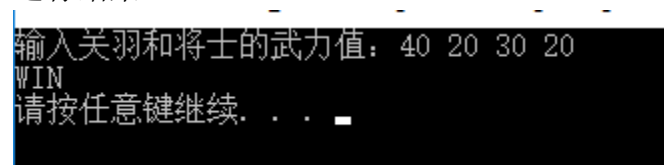
- a.素数的定义：在大于 1 的自然数中，除了 1 和它本身以外不再有其他[因数](#)。
- b.判断武力值是否满足条件

Code:

```
#include<iostream>
using namespace std;
int countNum = 1;//记录关羽过的关数
//判断武力值是否合乎标准
bool judge(int force) {
    if (force > 0 && force < 50)
        return true;
    return false;
}
bool isPrime(int n) {
    if (n == 1)
        return false;
    if (n == 2)
        return true;
    // 所有偶数都不是素数
    if (n%2 == 0)
        return false;
    // 只需要检查奇数
    for (int i = 3; i < n; i += 2)
        if (n%i == 0)
            return false;
    return true;
}
//判断是否通过当前关卡
bool passNum(int x,int y) {
    int sum = (x - y)*(x - y) + (x - y) + 41;
    if(isPrime(sum))
        return true;
    return false;
}
int main() {
    cout << "输入关羽和将士的武力值：";
    int x = 0, y1 = 0,y2=0,y3=0;
    cin >> x >> y1>>y2>>y3;
    //有一个不合乎标准则需要重新输入
```

```
while (!judge(x) || !judge(y1) || !judge(y2) || !judge(y3)) {  
    cout << "重新输入关羽和将士的武力值：";  
    cin >> x >> y1 >> y2 >> y3;  
}  
if (!passNum(x, y1)) { //未通过,必须一关一关的过  
    cout << countNum << endl;  
    system("pause");  
    return 0;  
}  
countNum++;  
if (!passNum(x, y2)) {  
    cout << countNum << endl;  
    system("pause");  
    return 0;  
}  
countNum++;  
if (!passNum(x, y3)) {  
    cout << countNum << endl;  
    system("pause");  
    return 0;  
}  
else  
    cout << "WIN" << endl;  
system("pause");  
return 0;  
}
```

运行结果：



输入关羽和将士的武力值：40 20 30 20
WIN
请按任意键继续. . .



2. 输入 N 个员工，每个员工输出 ID 号，上班时间，下班时间，
 第一行输出 最早去的员工的 ID 和上班时间
 第二行输出 最迟走的员工的 ID 和下班时间
 第三行输出 工作最久的员工的 ID 和上班时间

sample input:

ID100001 07:00:00 17:00:00
 ID100002 08:00:00 18:00:00
 ID100003 09:00:00 21:00:00

sample out:

OPEN:ID100001, 07:00:00
 CLOSE:ID100003, 21:00:00
 LONGEST WORK TIME ID:ID100003, 09:00:00

算法思想：本题主要是基于比较，将结构体定义在一起，主要是其中将所有的时间化作秒来计算比较方便

Code:

```
#include <iostream>    #include <algorithm>    using namespace std;
//定义时间结构体
typedef struct {
    int hour;int min;int sec;
}time;
//定义员工结构体
typedef struct {
    char id[100];
    time t1, t2;        //定义上班和下班的时间
    int workTime;        //员工的工作时间
}employee;
//比较员工上班时间最早的
bool cmp1(employee a, employee b) {
    if (a.t1.hour != b.t1.hour)
        return a.t1.hour < b.t1.hour;
    else if (a.t1.min != b.t1.min)
        return a.t1.min < b.t1.min;
    else
        return a.t1.sec < b.t1.sec;
}
//比较员工下班时间最晚的
bool cmp2(employee a, employee b) {
    if (a.t2.hour != b.t2.hour)
        return a.t2.hour > b.t2.hour;
    else if (a.t2.min != b.t2.min)
        return a.t2.min > b.t2.min;
    else
        return a.t2.sec > b.t2.sec;
}
```

```

//比较员工留在单位的时间最长的
bool cmp3(employee a, employee b) {
    return a.workTime > b.workTime;
}

int main() {
    //判断输入时间的合法性本示例程序省略
    cout << "输入员工的数目: ";
    int n = 0;
    cin >> n;
    employee em[100]; //假设最多存在 100 个员工
    cout << "输入每个员工的 id, 以及上下班时间:" << endl;
    for (int i=0; i<n; i++) {
        cin >> em[i].id;
        scanf_s("%2d:%2d:%2d", &em[i].t1.hour, &em[i].t1.min, &em[i].t1.sec);
        scanf_s("%2d:%2d:%2d", &em[i].t2.hour, &em[i].t2.min, &em[i].t2.sec);
        em[i].workTime = (em[i].t2.hour * 3600 + em[i].t2.min * 60 +
em[i].t2.sec)
        - (em[i].t1.hour * 3600 + em[i].t1.min * 60 + em[i].t1.sec);
    }
    //比较上班的时间来的最早的, 每进行一次比较, 就输出一排序后的结果
    sort(em, em + n, cmp1);
    cout << "OPEN:";
    cout << em[0].id << " ";
    printf("%.2d: %.2d: %.2d\n", em[0].t1.hour, em[0].t1.min, em[0].t1.sec);
    sort(em, em + n, cmp2);
    cout << "CLOSE:";
    cout << em[0].id << " ";
    printf("%.2d: %.2d: %.2d\n", em[0].t2.hour, em[0].t2.min, em[0].t2.sec);
    sort(em, em + n, cmp3);
    cout << "LONGEST WORK TIME ID:";
    cout << em[0].id << " ";
    printf("%.2d: %.2d: %.2d\n", em[0].t1.hour, em[0].t1.min, em[0].t1.sec);
    system("pause");
    return 0;
}

```

运行结果:

```

输入员工的数目: 3
输入每个员工的id, 以及上下班时间:
ID100001 07:00:00 17:00:00
ID100002 08:00:00 18:00:00
ID100003 09:00:00 21:00:00
OPEN:ID100001 07:00:00
CLOSE:ID100003 21:00:00
LONGEST WORK TIME ID:ID100003 09:00:00
请按任意键继续. . .

```

3.有一个 $M \times N$ 的材料和一个 $s \times t$ 的模板，从材料中切除模板，求最大能切出来的模板的数量。

输入：

第一行 M N

下面的 M 行 N 列：材料的具体内容

第 $M+2$ 行： s t

下面的 s 行 t 列：模板材料的具体内容

Sample input:

```
3 4
a b c d
c d a b
a c c d
2 2
a b
c d
```

Sample out :

```
2
```

算法思想：

1. 定义两个字符数组将这两幅图存储起来；
2. 使用一个判断模块，当进行搜索的时候，判断该位置是否合法(也就是以该位置为起点的 $s \times t$ 图形是否能和输入的 $s \times t$ 图形恰好匹配)；
3. 构造一个初始化模块，当位置合法的时候，可以选择使用或者不使用这种位置；
4. 采用深度搜索方式，逐行进行扫描。

(1) 如果到某一位置，判断时合法。我们采取两种行动（要么将大图中匹配的位置标记为 0，然后继续深度搜索，要么就当什么事都没有发生，继续逐行扫描）采取这两种操作的原因是，我们不知道哪种操作是更好的结果，所以我们就都试试；

(2) 当该行扫描完之后，进入下一行继续进行扫描；

(3) 程序出口是扫描到了最后一行；

(4) 因为递归过程中遍历了所有的可能，如果遇到更优解，我们需要进行答案的更新。

Code:

```
#include<string.h>
#include<iostream>
using namespace std;
#define MAX 100
char map[MAX][MAX], mapb[MAX][MAX];
char temp[MAX][MAX]; //记录中间图的变换，方便在递归中的回溯进行使用
int M, N, s, t; //分别为这两个图形的长和宽
int Max, ans; //分别记录每次遍历的最大块儿数和最终答案
bool judge(char map[][MAX], int i, int j) {
    int x = i;
    int y = j; //x 来表示行数，y 来表示列数
    for (i = 0; i < s; i++) //判断两个图是否匹配
        for (j = 0; j < t; j++)
```



```

        if (map[x + i][y + j] != mapb[i][j])
            return false;
        return true;
    }
    //将已经遍历过的位置标记
    void init_temp(int x, int y) {
        for (int i = 0; i < M; i++)
            for (int j = 0; j < N; j++)
                temp[i][j] = map[i][j];
        for (int i = 0; i < s; i++)
            for (int j = 0; j < t; j++)
                temp[x + i][y + j] = '0';
    }
    void dfs(char map[][MAX], int x, int y, int Max) {
        if (ans < Max)        //ans 来记录块的数量
            ans = Max;
        if (x >= M) {        //记录递归出口,走到最后一行就返回
            return;
        }
        for (int i = y; i < N; i++) { //对 x 行的每一列都进行搜索
            if (judge(map, x, i)) { //满足判断进行下一步递归
                init_temp(x, i);
                dfs(temp, x, i + t, Max + 1); //将遍历过的位置标记,然后继续遍历
            }
        }
        dfs(map, x + 1, 0, Max);
    }
    int main() {
        //输入 M*N 材料的信息
        cin >> M >> N;
        for (int i = 0; i < M; i++)
            for (int j = 0; j < N; j++)
                cin >> map[i][j];
        //输入 s*t 材料的信息
        cin >> s >> t;
        for (int i = 0; i < s; i++)
            for (int j = 0; j < t; j++)
                cin >> mapb[i][j];
        ans = 0; Max = 0;
        dfs(map, 0, 0, 0);
        cout << ans << endl;
        system("pause");
        return 0;
    }

```

运行结果:

```

3 4
a b c d
c d a b
a c c d
2 2
a b
c d
2
请按任意键继续. . .

```

1.2.6 华东师范大学上机题

1. 给一个小学生都会算的 1 位数与 1 位数运算的代数式，请你求出这个表达式的值。
表达式仅含+*/四种运算，题目保证 0 不为除数。

Sample Input 1:

1+1

SampleOutput 1:

2

Sample Input 2:

3*4

Sample OutPut2:

12

选取本题的主要想法：主要是考察表达式求值，除法怎么求
另外读者可以尝试着写一下表达式求值的内容，和本题类似

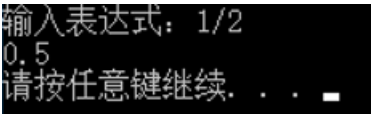
Code:

```
#include<iostream>
using namespace std;
float calculate(char a[]) {
    char n1 = a[0];
    char op = a[1];
    char n2 = a[2];
    if (op == '+')
        return n1 + n2-2*48;
    else if (op == '-')
        return n1 - n2;
    else if (op == '*')
        return (n1-'0') * (n2-'0');
    else
        return (n1-'0'-0.0) / (n2-'0');//保证除法的正确性
}

int main() {
    //题目中只是给出了一位数与一位数的运算
    cout << "输入表达式: ";
    char str[3];
    cin >> str;
    float result = calculate(str);
    cout << result << endl;

    system("pause");
    return 0;
}
```

运行结果:



```
输入表达式: 1/2
0.5
请按任意键继续. . .
```

2. 有一个研究团队，团队分成许多研究小组，每个小组的一部分成员可能再分成小组。小组的成员只知道自己的组长是谁，而在同一个组长领导下的成员之间却相互不认识。现在这个团队希望有一个程序能统计一下各组长带领小组的规模，即对每一个成员想知道自己及自己带领下的小组有多少人。

输入： 2 行，第 1 行有 1 个数字 N ($0 < N < 2 \times 10^5$) ($0 < N < 2 \times 10^5$)，代表小组的人数

第 2 行有 N 个数 $a_1, a_2, \dots, a_i, \dots, a_N$ ，表示第 i 个人的领导是 a_i 。团队的领导用 0 表示，说明没有人做他的组长。数据保证没有环路。单独的一个成员视为 1 个人的小组。

输出： 1 行， N 个数字，表示第 i 名成员的团队的规模

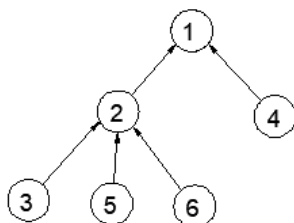
Sample Input:

0 1 2 1 2 2

Sample Output:

6 4 1 1 1 1

算法思想：画出下图所示的双亲表示法的树：



根据其数组给出的方式，当求以 1 为根节点的所包含的结点的个数时， $3 \rightarrow 2 \rightarrow 1$, $5 \rightarrow 2 \rightarrow 1$, $6 \rightarrow 2 \rightarrow 1$, $2 \rightarrow 1$, $4 \rightarrow 1$ 共有 $5+1=6$ 个，同理可得 2 是 $3+1=4$ 个。可以定义一个数组，遍历这个双亲表示法的数组，看每个节点被扫描的次数即是以其为根节点所包含的结点的数目。

Code:

```

#include<iostream>
using namespace std;
void manageNum(int * arr, int n){
    //创建一个计数的数组
    int *b = new int[n + 1];
    //数组初始化
    for (int i = 1; i <= n + 1; i++)
        b[i] = 1;
    for (int i = 1; i < n + 1; i++) {
        if(arr[i]==0)
            continue;
        else{
            int k = i;
            while (arr[k] != 0) {
                int j = arr[k];
                b[j]++;
                k = j;
            }
        }
    }
}
  
```

```

    }
}
for (int i = 1; i < n + 1; i++)
    cout << b[i] << " ";
}
int main() {
    cout << "输入团队的总人数: ";
    int n = 0;
    cin >> n;
    int *arr = new int[n+1];
    //双亲表示法数组的输入
    for (int i = 1; i < n + 1; i++)
        cin >> arr[i];
    manageNum(arr, n);

    system("pause");
    return 0;
}

```

运行结果:

```

输入团队的总人数: 6
0 1 2 1 2 2
6 4 1 1 1 1 请按任意键继续. . .

```

3. 假设在周末舞会上, 男士们和女士们进入舞厅时, 各自排成一队。跳舞开始时, 依次从男队和女队的队头上各出一人配成舞伴。规定每个舞曲能有一对跳舞者。若两队初始人数不相同, 则较长的那一队中未配对者等待下一轮舞曲。现要求写一个程序, 模拟上述舞伴配对问题。

Input

三个整数 m , n , k ($1 \leq m, n \leq 150, 1 \leq k \leq 4000$), 分别表示男士人数、女士人数、几轮舞曲。

Output

输出各轮舞曲的配对方案。

Examples

input

2 4 6

output

1 1

2 2

1 3

2 4

1 1

2 2

Code:

```
#include <iostream>
using namespace std;
int main() {
    //实际情况不需要输入文字，输入文字是为了可读性
    cout << "输入男士人数、女士人数、舞曲轮数：";
    int m = 0, n = 0, k = 0;
    cin >> m >> n >> k;
    int m_temp = 1, n_temp = 1;
    int cycle = 1;
    while (cycle <= k) {
        if (m_temp > m)
            m_temp = 1;
        if (n_temp > n)
            n_temp = 1;

        cout << m_temp << " ";
        cout << n_temp << endl;

        m_temp++;
        n_temp++;
        cycle++;
    }
    system("pause");
    return 0;
}
```

运行结果：

```
输入男士人数、女士人数、舞曲轮数： 2 4 6
1 1
2 2
1 3
2 4
1 1
2 2
请按任意键继续. . .
```



1.2.7 兰州大学上机题

1 输入个字符串，例如 `asdfghj`，然后又给了个字符串，`sdfg` 要求将输入字符串中的 `sdfg` 删除，就是相当于删除子字符串。

算法思想：先找到子字符串，然后再进行删除操作，这里根据所学的数据结构存在两种方法，笔者先采用暴力法解决，后面采用 **KMP 算法解决**（写出了 **KMP 算法**，删除操作自己实现）。

Code:

```
#include<iostream>
#include <string.h>
using namespace std;
//暴力搜索
int BF(char str[], char substr[]) {
    //确定每个字符串的长度
    int m = strlen(str);
    int n = strlen(substr);
    int i = 0, j = 0;
    while (i < m && j < n) {
        if (str[i] == substr[j]) { //若相等，则继续比较后序字符
            i++;
            j++;
        }
        else { //指针回退，重新匹配
            i = i - j + 1;
            j = 0;
        }
    }
    if (j >= n) return i - n;
    else return 0;
}

int length(char arr[], int pos, int pos1) {
    int slength = strlen(arr);
    for (int i = pos1; i < slength; i++)
        arr[pos++] = arr[i];
    return pos;
}

//KMP 算法求解
void getNext(char T[], int next[], int n) {
    int i = 1;
    next[0] = 0; //不存储任何数据
    next[1] = 0;
    int j = 0;
    while (i <= n) { //n 表示字符串 T 的长度
        if (j == 0 || T[i - 1] == T[j - 1]) {
            i++;
            j++;
        }
    }
}
```

```

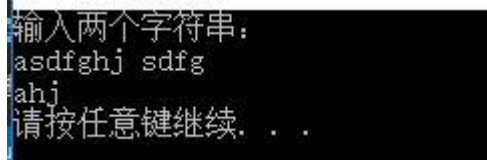
        next[i] = j;
    }
    else
        j = next[j];
}
}

//KMP--求子串在主串中第 pos 个字符之后的位置的 KMP 算法
int KMP(char S[], char T[], int next[], int pos) {
    int i = pos;
    int j = 1;
    while (i <= strlen(S) && j <= strlen(T)) {
        if (j == 0 || S[i - 1] == T[j - 1]) {
            ++i;    ++j;
        }
        else
            j = next[j];
    }
    if (j > strlen(T)) return i - strlen(T) - 1;
    else return 0;
}

int main() {
    cout << "输入两个字符串: " << endl;
    //假设两个字符串的最大长度均为 100
    char str[100], substr[100];
    cin >> str >> substr;
    /*int pos = BF(str, substr);
    int pos1 = pos + strlen(substr);
    int lengthStr = length(str, pos, pos1);
    for (int i = 0; i < lengthStr; i++)
        cout << str[i];*/
    //以下为 KMP 算法代码内容
    int next[100];
    getNext(substr, next, strlen(substr));
    for (int i = 1; i <= strlen(substr); i++)
        cout << next[i] << " ";
    cout << endl;
    int flag=KMP(str, substr, next, 0);
    cout << flag << endl;
    cout << endl;
    system("pause");
    return 0;
}

```

运行结果:



```

输入两个字符串:
asdfghj sdfg
ahj
请按任意键继续. . .

```

2 字符串压缩，比如 xxxxxdddf，输出 x5d3f3

算法思想：设置一个游标，一个计数器，依次将当前的字符和游标字符比较，如果相同，计数器++，游标继续后移，如果不同，将当前的字符加入到待输出的字符串中，并加入计数器的数量

Code:

```
#include <iostream>
#include <string>
using namespace std;
#define MAX_SIZE 100 //假设字符串长度最大为 100
string compress(string iniString) {
    string str;
    int count = 1;
    for (int i = 0; i < iniString.length(); i++){
        if (iniString[i] == iniString[i + 1]){
            count++;
            continue;
        }
        str += iniString[i];
        str += to_string(count);
        count = 1;
    }
    if (str.length() >= iniString.length())
        return iniString;
    return str;
}

int main() {
    char str[MAX_SIZE];
    cin >> str;
    string s = compress(str);
    cout << s << endl;

    system("pause");
    return 0;
}
```

运行结果：

```
xxxxxxdddf
x5d3f3
请按任意键继续. . .
```


1.2.8 西北工业大学上机题

1. 十进制转为二进制

描述：将一个十进制的数转换为二进制数。

输入：输入一个 10000 以内的数转换为二进制数。

输入：转换为二进制后输出。

输入样例

1030

输出样例

10000000110

Code:

```
#include <iostream>
using namespace std;
long long power(int X, int Y){
    int i = 0;
    long long Z = 1;
    for (i; i < Y; i++){
        Z = Z * 10;
    }
    return Z;
}
//采用递归的方式转化
long long recurrence(int X){
    int i = 0;
    long long Z = 0;
    while (X > 0){
        if (X < 2){
            Z = Z + (X % 2)*power(10, i);
            X = 0;
        }
        else{
            Z = Z + (X % 2)*power(10, i);
            X = X / 2;
            i++;
            recurrence(X);
        }
    }
    return Z;
}
//采用循环的方式转换 loop
long long loop(int X){
    int i, j;
    long long Z = 0;
    for (i = 0; X != 0; i++){
```

```

        j = X % 2;
        Z = Z + j * power(10, i);
        X = X / 2;
    }
    return Z;
}

int main() {
    int X=0;
    printf("请输入一个整数: ");
    cin >> X;
    printf("二进制数为: %lld\n", loop(X));
    printf("二进制数为: %lld\n", recurrence(X));
    system("pause");
    return 0;
}

```

运行结果:



```

请输入一个整数: 1030
二进制数为: 100000000110
二进制数为: 100000000110
请按任意键继续. . .

```

2. 输入一组数据的个数 再输入这组数据, 然后排序。

本题采用的排序: 快速排序方法。

写本题的思考: 其实有很多学校实际上考的还是比较基础的, 学完数据结构后对于基本的排序, 查找一定要十分熟悉, 千万不能忘记。在西工大试题集里面, 笔者还看到归并排序的上机题, 所以这些基本技能一定不要忘记。

读者可以直接尝试一下能不能快速的写出快速排序的代码

Code:

```

#include<iostream>
using namespace std;
int partition(int * a, int low, int high) {
    int pivot = a[low];
    while (low < high) {
        if (high > low && a[high] > pivot) --high;
        a[low] = a[high];
        if (high > low && a[low] < pivot) ++low;
        a[high] = a[low];
    }
    a[low] = pivot;
    return low;
}

void quickSort(int * a, int low, int high) {

```