

N 诺考研系列

# 计算机考研机试攻略

高分篇



N 诺课程教研团队

2019. 12. 01 更新

## 写在前面的话

相信各位同学都知道，N 诺是一个大佬云集的平台。N 诺每周都会定期举办比赛，让同学们在日复一日的枯燥生活中找到一种乐趣。

本书是 N 诺专为计算机考研的同学精心准备的一本**神书**，为什么说它是一本神书，它到底神在什么地方？

- 1、这本书是 N 诺邀请众多 **CSP 大佬**、**ACM 大佬**、**BAT 专业大佬**以及往年**机试高分大佬**共同修订而成。
- 2、这本书与市面上的书都不一样，这是一本专门针对**计算机考研机试**而精心编写而成的书籍。
- 3、这本书上的**例题讲解**以及**习题**都可以在 N 诺上找到进行练习，并且每道题都有大佬们发布的**题解**可以学习。
- 4、N 诺有自己的官方群，群里大佬遍地走，遇到问题在群里可以随时提问。群里**神仙**很多，不怕没有问题，就怕问题太简单。

相信同学们一定听过各种各样的 OJ 平台，但是那些平台不是为了计算机考研而准备的，而 N 诺是**唯一**一个纯粹为计算机考研而准备的学习平台。

相信每一个同学在学习本书之前，都觉得机试是一个很头疼的问题，机试有可能**速成**吗？我可以一周就**变得很强**吗？我们可以在短短两三周之内的学习就能拿到**机试高分**吗？

别人可不可以我不知道，但是在 N 诺这里，你就可以。

虽然考研机试题目千千万，但是你可以“一招鲜，吃遍天”。

管它乱七八糟的排序，我就一个 sort 你能拿我怎么办？

管它查来查去的问题，我会 map 我怕谁？

管它飘来飘去的规律，我有 OEIS 神器坐着看你秀！

管它眼观缭乱的算法，我有 N 诺万能算法模板怕过谁？

本书虽然不能让你变成一个算法高手，但是本书可以让你快速变成一个机试高手，这也是本书最重要的特点，以解决同学们的实际需求为目的。你可以不会算法，你可以不必弄懂算法的原理，你只要学会本书教给你的各种技巧，就足以应对 99.9% 的情况。

N 诺的课程教研团队全是由大佬组成，每个人都做过几千道编程题目，做编程题的经验十分丰富，我们分析近百所院校的历年机试真题，从中发现了各个学校的机试真题都有相同的规律，万变不离其中，于是本书也就诞生了。

计算机考研机试攻略交流群请扫描下方二维码或直接搜索群号：960036920



群名称：N诺 - 计算机机试交流群  
群号：960036920

本书配套视频：<https://www.bilibili.com/video/av76476205>

## 关于 N 诺

N 诺是全国最大的计算机考研在线学习平台。N 诺致力于为同学们提供一个良好的网络学习环境，一个与大佬们随时随地交流的舒适空间。如果你不知道 N 诺，那么你已经输在起跑线上了，因为 N 诺是 - 计算机学习考研必备神器。

N 诺收集并整理了**计算机考研的前世今生**，帮助你了解考研的点点滴滴。

<http://www.noobdream.com/post/585/>

在 N 诺，你可以查询各个院校的**考研信息**，包括分数线、录取人数、考试大纲、导师信息、经验交流等信息，应有尽有。

<http://www.noobdream.com/schoollist/>

在 N 诺，你可以尽情的刷各个科目的**题库**，还可以将题目加入**错题本**方便以后复习，也可以写下自己的**学习笔记**，记录考研过程中跌宕起伏。

<http://www.noobdream.com/Practice/index/>

在 N 诺，你可以在**讨论区**里发表你的问题或感想，与全国几百万考研 er 分享你的喜怒哀乐。

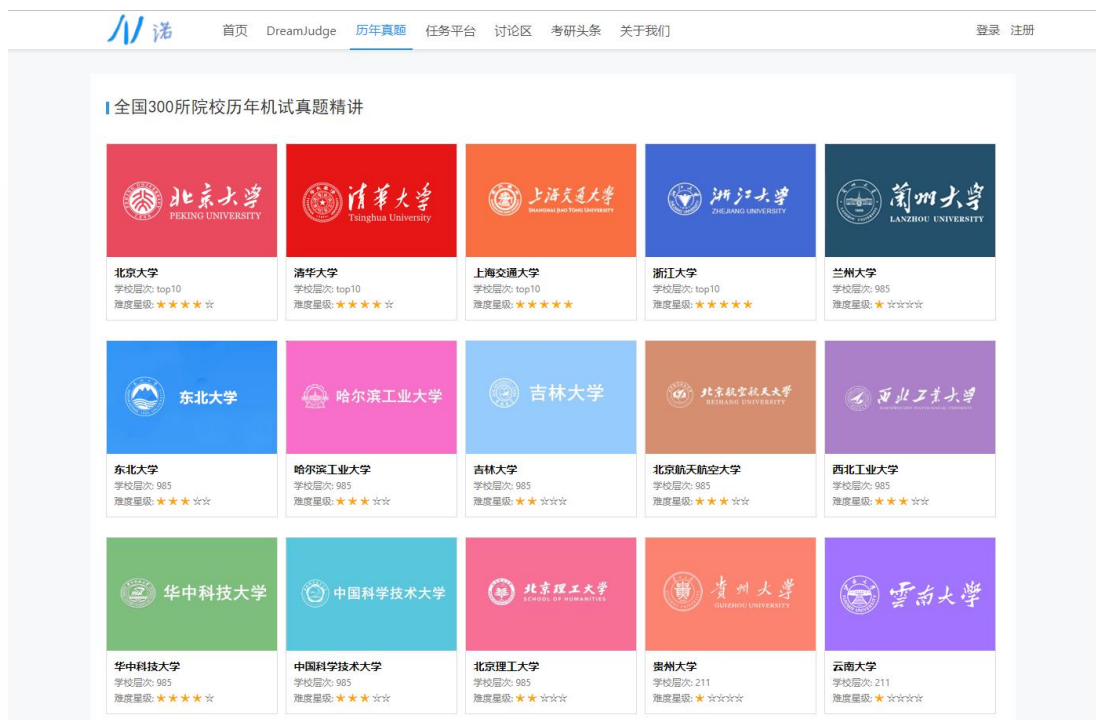
<http://www.noobdream.com/forum/0/>

在 N 诺，你可以将你不用**的书籍或资料**放到**二手交易市场**，既能帮助他人，还能为自己省下一笔当初买书买资料的费用。

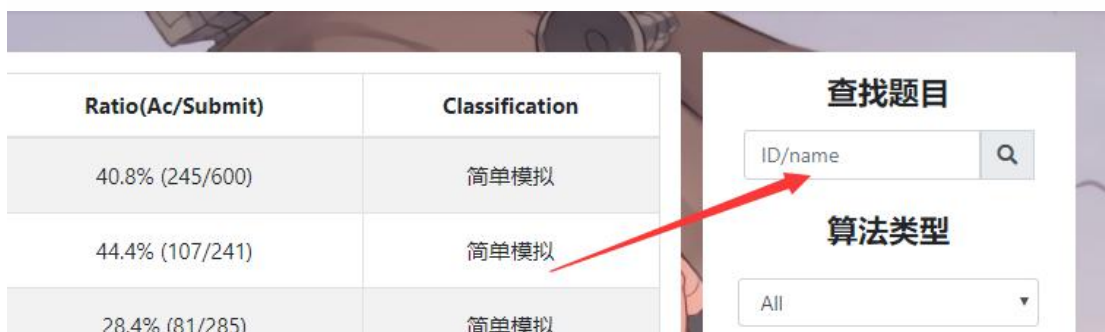
<http://www.noobdream.com/Task/tasklist/>

## 如何使用本书？

访问 N 诺平台（[www.noobdream.com](http://www.noobdream.com)）的历年真题，即可查询到全国各个学校的历年机试真题。



如下图输入课后习题的编号即可搜索到题目



小技巧：N 诺是可以随意更换皮肤的哟，在右上角头像旁边。

# 第一章 从零开始

## 1.1 机试分析

首先我们来看一下机试是怎样的一种考核模式

全国所有院校的机试都大同小异，大部分有自己 OJ (Online Judge 也就是在线代码测评平台) 的学校都会采用 OJ 上做题的方式来进行考核。这种考核方式的好处是公开透明，机器进行判题并给分，就像 N 诺的 DreamJudge 一样。没有 OJ 的学校只能人工进行判题，人工判题的话，一方面是主观性比较强，可能还会对其他方面进行考量，这个就需要自己去了解了。总的来说，不论是 OJ 判题还是人工判题，代码都要能通过测试用例才能得到分数。

针对机试我们应该怎么去训练提升自己呢

首先，一定要做题，在 N 诺上做题，不要自己埋头看书，在不断做题的过程中才能提升自己。如果非要用一个量化的标准来衡量的话，至少要在 N 诺上做够 100 题（也就是达到砖石 II 以上段位），才能保证你机试达到自己满意的成绩。题量是很关键的，看懂的题再多，都不如自己实际敲代码去解决问题更稳妥。

解题速度也是很重要的

其实解题速度和题量是正相关的，相信题量足够的同学，解题的速度都不会太慢。机试一般 2-3 个小时左右，要解决 5-8 道题。平均下来一道题最多半个小时，从读题、分析题意、思考解法、敲代码、调试、测试数据到最后提交这整个流程下来，如果你平时训练的少，读题慢、理解题意慢、思考解法慢、敲代码慢、调试慢，这样一算下来，一道简单的题都可能要一个小时才能写出来，说不定题目还有坑点，再慢慢调试，基本上就凉了。

## 多打比赛也是很重要的

很多同学平时做的题很多，解题速度也挺快，但是一到比赛或者考试的时候就会卡题，在压力的情况下发挥失常比比皆是，所以平时就要锻炼自己的抗压能力。

N 诺上除了每个周定期举办的小白赛，还特别为大家准备了**考研机试冲刺八套卷**。

在本书的最后，你可以看到关于考研机试冲刺八套卷的详细信息。通过这八套卷的练习，相信会让你的水平产生一个脱胎换骨的变化。

## 准备好模板是至关重要的

一般来说，机试都可以带书和纸质资料进入考场。所以提前把那些函数的用法和算法的模板准备好是很重要的，一方面是增加自己的信心，万一没记住还可以翻开来看一下。另外说不定考到原题或者类似的题，就可以直接秒杀了。

**特别提醒：**本书默认读者是会 C 语言的基本语法的，比如 if 语句、for 语句等等。

noobdream.com

C 语言基本语法还未掌握的同学建议学习 N 诺的 C 语言快速入门课程(3 天学会 C 语言不是梦)

地址：<http://www.noobdream.com/Major/majorinfo/1/>



## 1.2 IDE 的选择与评测结果

建议选择 **CodeBlocks** 作为平时敲代码练习的 IDE

下载地址: <http://www.noobdream.com/Major/article/1/>

### 常见做题结果反馈

**Accepted:** 答案正确, 恭喜你正确通过了这道题目。

**Wrong Answer:** 答案错误, 出现这个错误的原因一般是你的程序实现或思路出现了问题, 或者数据范围边界没有考虑到。

**Runtime Error:** 运行时错误, 出现这个错误的原因一般是数组越界或者递归过深导致栈溢出。

**Presentation Error:** 输出格式错误, 出现这个错误的原因一般是末尾多了或少了空格, 多了或少了换行

**Time Limit Exceeded:** 程序运行超时, 出现这个错误的原因一般是你的算法不够优秀, 导致程序运行时间过长。

**Memory Limit Exceeded:** 运行内存超限, 出现这个错误的原因一般是你的程序申请太大了空间, 超过了题目规定的空间大小。

**Compile Error:** 编译错误, 这个不用说了吧, 就是你的代码存在语法错误, 检查一下是不是选择错误的语言提交了。

**Output Limit Exceeded:** 输出超限, 程序输出过多的内容, 一般是循环出了问题导致多次输出或者是调试信息忘记删除了。

**Submitting:** 提交中, 请等待题目结果的返回, 由于判题机有性能差异, 所以返回结果的速度也不一样, N 诺上做题一般瞬间就能出结果。

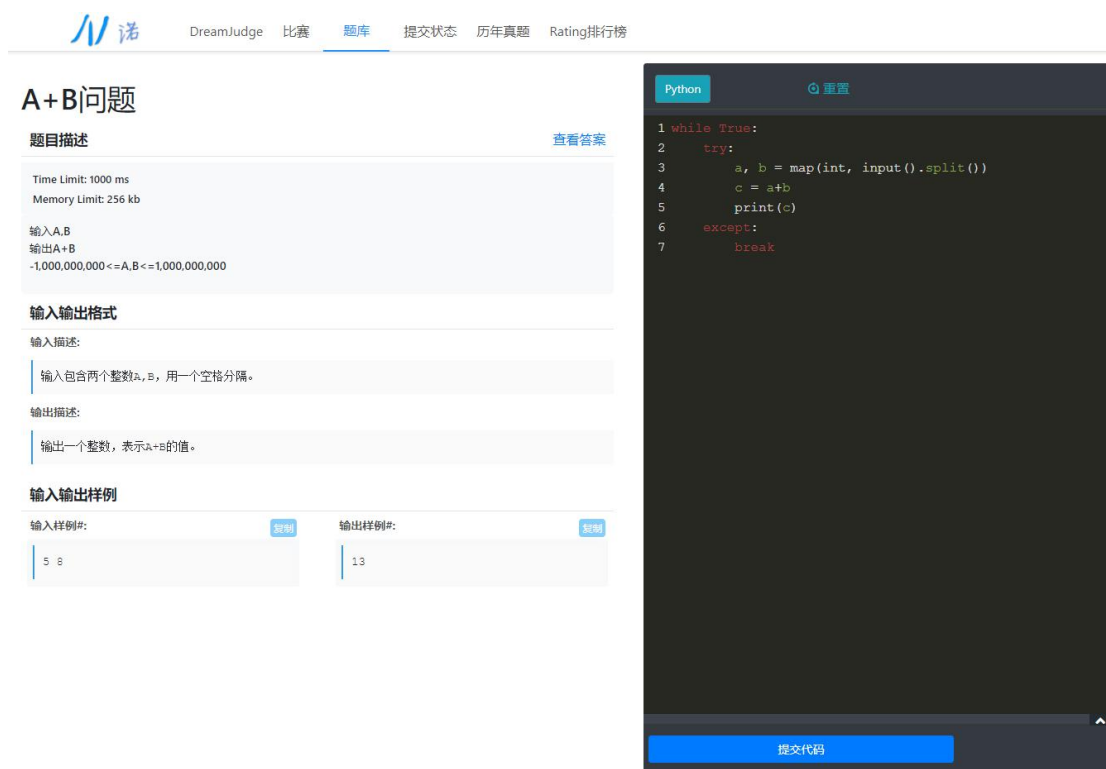
以上几种结果就是评判系统可能会返回的几种常见结果。若返回 Accept, 那么你就可以拿到该题所有分数, 如果返回其他结果, 则要看你报考学校的考试规则, 是根据通过测试点的百分比给分还是只要不是 AC 就得 0 分。



## 1.3 DreamJudge 的使用

DreamJudge 是一个在线代码测评的平台，可以很方便的检验自身的学习情况。使用 DreamJudge 的方法很简单，通过百度搜索 N 诺或者在浏览器中输入网址 [www.noobdream.com](http://www.noobdream.com) 进入 N 诺然后点击网站导航上方的 DreamJudge 就可以进去啦。

做题页面如下：



The screenshot shows the DreamJudge website interface. The top navigation bar includes links for DreamJudge, 比赛 (Contests), 题库 (Problem Set), 提交状态 (Submission Status), 历年真题 (Past Exams), and Rating排行榜 (Rating Ranking). The main content area is titled 'A+B问题' (A+B Problem). Under '题目描述' (Problem Description), it specifies a Time Limit of 1000 ms and a Memory Limit of 256 kb. The input is two integers A and B, and the output is their sum A+B. The constraints are  $-1,000,000,000 \leq A, B \leq 1,000,000,000$ . The '输入输出格式' (Input/Output Format) section states that the input consists of two integers A and B separated by a space, and the output is a single integer representing A+B. The '输入输出样例' (Input/Output Examples) section shows an input of '5 8' and an output of '13'. On the right side, there is a code editor with a Python tab selected, containing a while loop that reads input, calculates the sum, and prints it. A '提交代码' (Submit Code) button is at the bottom of the editor.

首先要登录我们的 N 诺账号，然后开始做题。如果没有账号，右上角点击注册，然后注册一个账号就可以了。

然后将代码粘贴到右边的输入框里，在上面选择使用哪种语言提交，C/C++/Java/Python，建议选择 C++提交，因为 C++可以编译 C 语言代码。我们一般写代码为了方便，都会使用一点 C++的特性来帮助我们快速解决一道题目。如果代码里含有 C++的特性却选择了 C 语言提交的话，会返回编译错误的提示信息。

## 1.4 输入输出技巧

输入 int 型变量 `scanf("%d", &x);`

输入 double 型变量 `scanf("%lf", &x);` 不用 float 直接 double

输入 char 类型变量 `scanf("%c", &x);`

输入字符串数组 `scanf("%s", s);`

输出与输入表示方式一致

`printf("%s\n", s);`

### scanf 输入解析

输入日期 2019-10-21

1. `int year, month, day;`
2. `scanf("%d-%d-%d", &year, &month, &day);`
3. `printf("%d %d %d\n", year, month, day);`

这样可以解析出来

输入时间 18:21:30

1. `int hour, minute, second;`
2. `scanf("%d:%d:%d", &hour, &minute, &second);`
3. `printf("%d %d %d\n", hour, minute, second);`

### scanf 和 gets

输入一行字符串带空格的话, 使用 `gets`, `scanf` 遇到空格会自动结束

1. `char s[105];`
2. `gets(s);` //例如输入 `how are you?`
3. `printf("%s\n", s);`

### getchar 和 putchar

读入单个字符和输出单个字符, 一般在 `scanf` 和 `gets` 中间使用 `getchar` 用于消除回车'\n'的影响

## 输出进制转换

1. `int a = 10;`
2. `printf("%x\n", a);`//小写十六进制输出 答案 a
3. `printf("%X\n", a);`//大写十六进制输出 答案 A
4. `printf("%o\n", a);`//八进制输出 答案 12

## 输出增加前置 0

1. `int a = 5;`
2. `printf("%02d\n", a);`//其中 2 代表宽度 不足的地方用 0 补充
3. 输出结果 05
4. `printf("%04d\n", a);`
5. 输出结果 0005

## 输出保留小数

1. `double a = 3.6;`
2. `printf("%.2lf\n", a);`//2 表示保留两位小数

输出结果 3.60

有小数输出小数, 没小数输出整数

`%g`

**特别注意:** 中文符号和英文符号要对应一致, 一般情况下都用英文符号 (如中文逗号, 和英文逗号,)

## long long 的使用

很多情况下的计算会超出 `int`, 比如求  $N!$ ,  $N$  比较大的时候 `int` 就存不下了, 这时候我们就要用 `long long`。那么我们去记 `int` 和 `long long` 的范围呢, 有一个简单的记法, `int` 范围  $-1e9$  到  $1e9$ , `long long` 范围  $-1e18$  到  $1e18$ , 这样就容易记了。

1. `long long x;`
2. `scanf("%lld", &x);`
3. `printf("%lld\n", x);`

## 字符的 ASCII 码

不要硬记，直接输出来看

```
printf("%d\n", 'a');
```

输出结果 97

```
printf("%d\n", 'A');
```

输出结果 65

**特别注意：**如果遇到需要 ASCII 码的题目时记住 char 字符和 int 值是可以相互转化的。

## cin 和 cout

很多时候使用 C++ 的输入输出写起来更简单，在应对一些**输入输出量不是很大**的题目时，我们会采用 cin 和 cout 来提高我们的解题速度。

比如求两个数的和：

```
1. #include <iostream> // 输入输出函数的头文件
2.
3. int main() {
4.     int a, b;
5.     cin >> a >> b;
6.     cout << a + b; // 输出两个数之和
7. }
```

可以发现，C++ 的输入输出敲起来更快，这是我们会使用它来进行混合编程的原因之一。

另外，C++ 的 string 类对于字符串操作很方便，但是输入输出只能用 cin、cout。

**特别注意：**大家一定平时练习的时候不要排斥混合编程，即 C 与 C++ 语法混用，然后用 C++ 提交。这样可以极大的帮助你以更快的速度解决一道你用纯 C 写半天才能解决题目，留下充裕的时间去解决更多的题目。

**友情提示：**当输入或输出格式有特殊要求的时候，cin 和 cout 不方便解决，那么我们还是使用 scanf 和 printf 来解决问题。要注意的是 **printf 尽量不要和 cout 同时使用**，会发生一些不可控的意外。

## 1.5 头文件技巧

这里推荐一个万能头文件给大家

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

不过要看考试的评测机支不支持，绝大部分都是支持的。当然，我们还可以留一手，准备一个完整的头文件，在考试开始前敲上去就行。

```
1. #include <stdio.h>
2. #include <string.h>
3. #include <math.h>
4. #include <stdlib.h>
5. #include <time.h>
6. #include <algorithm>
7. #include <iostream>
8. #include <queue>
9. #include <stack>
10. #include <vector>
11. #include <string>
12. using namespace std;
13.
14. int main() {
15.
16.     return 0;
17. }
```

**特别注意：**头文件可以多，但是不能少，但是有一些头文件是不允许的，大部分 OJ 为了系统安全性考虑限制了一些特殊的 API，导致一些头文件不能使用，比如 windows.h。当然不同的 OJ 的安全策略也不尽相同，一般不涉及到系统函数的头文件一般都是可以使用的。

## 1.6 数组使用技巧

数组除了可以存储数据以外，还可以用来进行标记。

例题：

输入 N ( $N \leq 100$ ) 个数，每个数的范围  $> 0$  并且  $\leq 100$ ，请将每个不同的数从小到大输出并且输出它对应的个数。

样例输入

```
8
3 2 2 1 1 4 5 5
```

样例输出

```
1 2
2 2
3 1
4 1
5 2
```

代码如下

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. int f[105]={0}; //注意，尽量将数组开在全局
5. int main() {
6.     int n,x;
7.     scanf("%d", &n);
8.     for (int i = 0; i < n; i++) {
9.         scanf("%d", &x);
10.        f[x]++;
11.    }
12.    for (int i = 0; i <= 100; i++) {
13.        if (f[i] > 0) printf("%d %d\n", i, f[i]);
14.    }
15.    return 0;
16. }
```



在这个程序中, 我们使用一个数组 `f` 记录每个值得个数, `f[i]` 的值表示 `i` 这个数有多少个, 初始的时候每个值的个数都是 0。

数组的使用不一定从 0 开始, 可以从任意下标开始, 只要我们使用的时候对应上就行。

例如我们**存储地图**的时候

####

#. ##

##@#

####

假设一个地图是这样的, 我们要用二维字符数组来存储, 我们可以像下面这样做。

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. char mpt[10][10];
5. int main() {
6.     for (int i = 1; i <= 4; i++) {
7.         scanf("%s", mpt[i] + 1);
8.         /* 不要用下面这种输入方式, 否则会出问题, 因为回车也算一个 char 字符
9.         for (int j = 1; j <= 4; j++) {
10.             scanf("%c", &mpt[i][j]);
11.         }
12.         */
13.     }
14.     for (int i = 1; i <= 4; i++) {
15.         for (int j = 1; j <= 4; j++) {
16.             printf("%c", mpt[i][j]);
17.         }
18.         printf("\n");
19.     }
20.     return 0;
21. }
```

## 数组还可以嵌套使用

我们将上面那题改进一下

### 例题：

输入 N ( $N \leq 100$ ) 个数，每个数的范围  $> 0$  并且  $\leq 100$ ，请将每个不同的数从小到大输出并且输出它对应的个数。如果多个值有相同的个数，输出值最大的那个。

### 样例输入

```
8
3 2 2 1 1 4 5 5
```

### 样例输出

```
4 1
5 2
```

### 代码如下

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. int f[105] = {0};
5. int p[105] = {0}; //p[i]表示有 i 个这样的数的最大值是多少
6. int main() {
7.     int n, x;
8.     scanf("%d", &n);
9.     for (int i = 0; i < n; i++) {
10.         scanf("%d", &x);
11.         f[x]++;
12.     }
13.     for (int i = 0; i <= 100; i++) p[f[i]] = i;
14.     for (int i = 1; i <= 100; i++) {
15.         if (p[i] > 0) printf("%d %d\n", p[i], i);
16.     }
17.     return 0;
18. }
```

## 1.7 审时度势 — 复杂度与是否可做

在做题之前，我们要先判断这道题是否可做，对于简单的模拟题，大家肯定都知道，我能写出来就是可做，写不出来就是不可做。但是对于循环嵌套和算法题，我们就需要去判断思考自己设计的算法是否可以通过这道题。

不懂复杂度计算的同学去看一下数据结构课程的第一章，很简单的。

例如：我们写一个冒泡排序，它是两个 for 循环，时间复杂度是  $O(N^2)$ ，那么在 1S 内我们最多可以对多少个数进行冒泡排序呢，N 在 1000 - 3000 之间。一般情况下我们可以默认评测机一秒内可以运行  $1e7$  条语句，当然这只是一个大概的估计，实际上每个服务器的性能不同，这个值都不同，但是一般都相差不大，差一个常数是正常的。因此，我们可以这样做一个对应，下面是时限 1S 的情况

$O(N)$	N 最大在 500W 左右
$O(N\log N)$	N 最大在 20W 左右
$O(N^2)$	N 最大在 2000 左右
$O(N^2\log N)$	N 最大 700 在左右
$O(N^3)$	N 最大在 200 左右
$O(N^4)$	N 最大在 50 左右
$O(2^N)$	N 最大在 24 左右
$O(N!)$	N 最大在 10 左右

如果是 2S、3S 对应的乘以 2 和 3 就可以。

**特殊技巧：**如果发现自己设计的算法不能在题目要求的时限内解决问题，不要着急，可以先把这道题留一下，继续做其他题，然后看一下排行榜，有多少人过了这道题，如果过的人多，那么说明这道题可能数据比较水，直接暴力做，不要怕复杂度的问题，因为出题人可能偷懒或者失误了导致数据很水。**考研机试的题目数据大部分情况都比较水，所以不要被复杂度吓唬住了，后面的章节会教大家面对不会更好的算法那来解决题目的时候，如何用优雅的技巧水过去。**

## 举个简单的例子

题目要求你对 10W 个数进行排序

假设你只会冒泡排序, 但是冒泡排序很明显复杂度太高了, 但是有可能出题人偷懒, 他构造的测试数据最多只有 100 个, 根本没有 10W 个, 那么你就可以用冒泡排序通过这道题。

但是这种情况比较少见, 一般至少都会有一组极限数据, 所以可以先把这道题放着去做其他题, 然后再看看其他人能不能通过, 如果很多人都过了, 那么你就可以暴力试一下。

**特别注意:** 空间复杂度一般不会限制, 如果遇到了再想办法优化空间。



## 1.8 C++ STL 的使用

C++的算法头文件里有很多很实用的函数，我们可以直接拿来用。

```
#include <algorithm>
```

### 排序

sort() 函数: 依次传入三个参数，要排序区间的起点，要排序区间的终点+1，比较函数。比较函数可以不填，则默认为从小到大排序。

### 使用示例

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. int a[105];
5. int main() {
6.     int n;
7.     scanf("%d", &n);
8.     for (int i = 0; i < n; i++) {
9.         scanf("%d", &a[i]);
10.    }
11.    sort(a, a+n);
12.    for (int i = 0; i < n; i++)
13.        printf("%d ", a[i]);
14.    return 0;
15. }
```

### 查找

lower\_bound() 函数

upper\_bound() 函数

lower\_bound( ) 和 upper\_bound( ) 都是利用二分查找的方法在一个排好序的数组中进行查找的。

在从小到大的排序数组中,

`lower_bound( begin, end, num)`: 从数组的 `begin` 位置到 `end-1` 位置二分查找第一个大于或等于 `num` 的数字, 找到返回该数字的地址, 不存在则返回 `end`。通过返回的地址减去起始地址 `begin`, 得到找到数字在数组中的下标。

`upper_bound( begin, end, num)`: 从数组的 `begin` 位置到 `end-1` 位置二分查找第一个大于 `num` 的数字, 找到返回该数字的地址, 不存在则返回 `end`。通过返回的地址减去起始地址 `begin`, 得到找到数字在数组中的下标。

在从大到小的排序数组中, 重载 `lower_bound()` 和 `upper_bound()`

`lower_bound( begin, end, num, greater<type>() )`: 从数组的 `begin` 位置到 `end-1` 位置二分查找第一个小于或等于 `num` 的数字, 找到返回该数字的地址, 不存在则返回 `end`。通过返回的地址减去起始地址 `begin`, 得到找到数字在数组中的下标。

`upper_bound( begin, end, num, greater<type>() )`: 从数组的 `begin` 位置到 `end-1` 位置二分查找第一个小于 `num` 的数字, 找到返回该数字的地址, 不存在则返回 `end`。通过返回的地址减去起始地址 `begin`, 得到找到数字在数组中的下标。

## 使用示例

```
1. #include<bits/stdc++.h>
2. using namespace std;
3.
4. int cmp(int a,int b){
5.     return a>b;
6. }
7. int main(){
8.     int num[6]={1,2,4,7,15,34};
9.     sort(num,num+6); //按从小到大排序
10.    int pos1=lower_bound(num,num+6,7)-num;
11.    //返回数组中第一个大于或等于被查数的值
12.    int pos2=upper_bound(num,num+6,7)-num;
13.    //返回数组中第一个大于被查数的值
14.    cout<<pos1<<" "<<num[pos1]<<endl;
15.    cout<<pos2<<" "<<num[pos2]<<endl;
16.    sort(num,num+6,cmp); //按从大到小排序
17.    int pos3=lower_bound(num,num+6,7,greater<int>())-num;
```



```
18. //返回数组中第一个小于或等于被查数的值
19. int pos4=upper_bound(num,num+6,7,greater<int>())-num;
20. //返回数组中第一个小于被查数的值
21. cout<<pos3<<" "<<num[pos3]<<endl;
22. cout<<pos4<<" "<<num[pos4]<<endl;
23. return 0;
24. }
```

## 优先队列

通过 `priority_queue<int> q` 来定义一个储存整数的空的 `priority_queue`。当然 `priority_queue` 可以存任何类型的数据, 比如 `priority_queue<string> q` 等等。

## 示例代码

```
1. #include <iostream>
2. #include <queue>
3. using namespace std;
4. int main() {
5.     priority_queue<int> q;//定义一个优先队列
6.     q.push(1);//入队
7.     q.push(2);
8.     q.push(3);
9.     while (!q.empty()) { //判读队列不为空
10.         cout << q.top() << endl; //队首元素
11.         q.pop(); //出队
12.     }
13.     return 0;
14. }
```

C++的 STL（标准模板库）是一个非常重要的东西, 可以极大的帮助你更快速的解决题目。

## vector

通过 `vector<int> v` 来定义一个储存整数的空的 `vector`。当然 `vector` 可以存任何类型的数据, 比如 `vector<string> v` 等等。

## 示例代码

```
1. #include <iostream>
2. #include <vector>
3. using namespace std;
4. int main() {
5.     vector<int> v; //定义一个空的 vector
6.     for (int i = 1; i <= 10; ++i) {
7.         v.push_back(i * i); //加入到 vector 中
8.     }
9.     for (int i = 0; i < v.size(); ++i) {
10.        cout << v[i] << " "; //访问 vector 的元素
11.    }
12.    cout << endl;
13.    return 0;
14. }
```

## queue

通过 `queue<int> q` 来定义一个储存整数的空的 queue。当然 queue 可以存任何类型的数据, 比如 `queue<string> q` 等等。

## 示例代码

```
1. #include <iostream>
2. #include <queue>
3. using namespace std;
4. int main() {
5.     queue<int> q; //定义一个队列
6.     q.push(1); //入队
7.     q.push(2);
8.     q.push(3);
9.     while (!q.empty()) { //当队列不为空
10.        cout << q.front() << endl; //取出队首元素
11.        q.pop(); //出队
12.    }
13.    return 0;
14. }
```

## stack

通过 `stack<int> S` 来定义一个全局栈来储存整数的空的 `stack`。当然 `stack` 可以存任何类型的数据, 比如 `stack<string> S` 等等。

### 示例代码

```
1. #include <iostream>
2. #include <stack>
3. using namespace std;
4. stack<int> S; //定义一个栈
5. int main() {
6.     S.push(1); //入栈
7.     S.push(10);
8.     S.push(7);
9.     while (!S.empty()) { //当栈不为空
10.         cout << S.top() << endl; //输出栈顶元素
11.         S.pop(); //出栈
12.     }
13.     return 0;
14. }
```

## map

通过 `map<string, int> dict` 来定义一个 `key:value` 映射关系的空的 `map`。当然 `map` 可以存任何类型的数据, 比如 `map<int, int> m` 等等。

### 示例代码

```
1. #include <iostream>
2. #include <string>
3. #include <map>
4. using namespace std;
5. int main() {
6.     map<string, int> dict; //定义一个 map
7.     dict["Tom"] = 1; //定义映射关系
8.     dict["Jones"] = 2;
9.     dict["Mary"] = 1;
10.    if (dict.count("Mary")) { //查找 map
11.        cout << "Mary is in class " << dict["Mary"];
12.    }
```

```

13. //使用迭代器遍历 map 的 key 和 value
14. for (map<string, int>::iterator it = dict.begin(); it != dict.end(); ++it) {
15.     cout << it->first << " is in class " << it->second << endl;
16. }
17. dict.clear();//清空 map
18. return 0;
19. }

```

## set

通过 `set<string> country` 来定义一个储存字符串的空的 set。当然 set 可以存任何类型的数据, 比如 `set<int> s` 等等。

## 示例代码

```

1. #include <iostream>
2. #include <set>
3. using namespace std;
4. int main() {
5.     set<string> country;//定义一个存放 string 的集合
6.     country.insert("China");//插入操作
7.     country.insert("America");
8.     country.insert("France");
9.     set<string>::iterator it;
10.    //使用迭代器遍历集合元素
11.    for (it = country.begin(); it != country.end(); ++it) {
12.        cout << * it << " ";
13.    }
14.    cout << endl;
15.    country.erase("American");//删除集合内的元素
16.    country.erase("England");
17.    if (country.count("China")) { //统计元素个数
18.        cout << "China in country." << endl;
19.    }
20.    country.clear();//清空集合
21.    return 0;
22. }

```

## 1.9 多组输入的问题

对有的题目来说，可能需要多组输入。

多组输入是什么意思呢？一般的题目我们输入一组数据，然后直接输出程序就结束了，但是多组输入的话要求我们可以循环输入输出结果。

例题：

输入两个数，输出两个数的和，要求多组输入。

样例输入

```
1 2
3 7
10 24
```

样例输出

```
3
10
34
```

C 循环读入代码如下

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. int main() {
5.     int a, b;
6.     while (scanf("%d%d", &a, &b) != EOF) {
7.         printf("%d\n", a+b);
8.     }
9.     return 0;
10. }
```

**特别注意：**不能使用 while(1) 这样死循环，!=EOF 的意思一直读取到文件末尾（End of file）另外，多组输入一定要注意初始化问题，数组和变量的初始化要放在 while 循环内，否则上一次的运算的结果会影响当前的结果。

C++循环读入代码如下

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. int main() {
5.     int a, b;
6.     while (cin >> a >> b) {
7.         cout << a + b << endl;
8.     }
9.     return 0;
10. }
```

Java 循环读入代码如下

```
1. Scanner stdin = new Scanner(System.in);
2. while (stdin.hasNext()) {
3.     String s = stdin.next();
4.     int n = stdin.nextInt();
5.     double b = stdin.nextDouble();
6. }
```

Python 循环读入代码如下

```
1. while True:
2.     try:
3.         a, b = map(int, input().split())
4.         c = a+b
5.         print(c)
6.     except: #读到文件末尾抛出异常结束循环
7.         break
```



## 第二章 入门经典

我们根据全国上百所院校的历年真题进行分析,总结了其中常见的题型,最常考的知识点。

学会这一章,在机试难度较低的学校基本上可以拿到 60-80 分左右的成绩,在机试难度中等的学校,也可拿到 40-60 分左右的成绩,在机试难度高的学校亦可将签到题做出来,拿到 20-40 分的成绩。

所以,认真看完这一章的内容对你的帮助会很大,加油, fighting!



## 2.1 简单模拟

在考研机试中，有一类很常见的题型叫做简单模拟。顾名思义，就是不需要去考虑什么算法，直接按照题目的意思进行模拟计算就行。

### 促销计算

#### 题目描述：

某百货公司为了促销，采用购物打折的优惠方法，每位顾客一次购物：在 1000 元以上者，按 9.5 折优惠；在 2000 以上者，按 9 折优惠；在 3000 以上者，按 8.5 折优惠；在 5000 以上者，按 8 折优惠；编写程序，购物款数，计算并输出优惠价。

#### 输入样例#：

```
850
1230
5000
3560
```

#### 输出样例#：

```
discount=1, pay=850
discount=0.95, pay=1168.5
discount=0.8, pay=4000
discount=0.85, pay=3026
```

#### 题目来源：

DreamJudge 1091

**解题分析：**根据题目的意思，我们知道就是按照题意去进行打折优惠的计算，只需要判断输入的数值在哪个区间该用什么优惠去计算就好了。

#### 参考代码

```
1. #include <bits/stdc++.h> //万能头文件
2. using namespace std;
3.
```

```

4.  int main() {
5.      double a;
6.      scanf("%lf", &a);
7.      //使用%g 可以自动去掉小数点后多余的0 如果是整数则显示整数
8.      if (a < 1000) printf("discount=1,pay=%g\n", a);
9.      if (a >= 1000 && a < 2000) printf("discount=0.95,pay=%g\n", a*0.95);
10.     if (a >= 2000 && a < 3000) printf("discount=0.9,pay=%g\n", a*0.9);
11.     if (a >= 3000 && a < 5000) printf("discount=0.85,pay=%g\n", a*0.85);
12.     if (a >= 5000) printf("discount=0.8,pay=%g\n", a*0.8);
13.     return 0;
14. }
```

## 题型总结

简单模拟这类题目在考试中很常见, 属于送分签到的题目。所有的考生, 注意了, 这类题必须会做。

对于简单模拟这一类的题目, 怎么去练习提高呢?

很简单, 在 DreamJudge 上**多做题**就行了。那么要达到什么样的标准呢?

如果你想拿高分甚至满分, 平时训练的时候, 这类题尽量要在 8 分钟内解决。

如果你只是想拿个还不错的成绩, 这类题 AC 的时间尽量不要超过 15 分钟, 一定要记住, 最坏情况不能超过 20 分钟, 如果超过了, 说明你平时做的题还是太少了。

在考试的过程中, 大多数考生都会紧张, 有些考生甚至会手抖, 导致敲多某个字母, 然后又调试半天, 找半天错, 会导致比平时解决同样难度的问题时长多一倍甚至更多, 所以平时就要注意, 做题千万不能太慢了, 不然没有足够的时间来解决其他的题目哦。

## 练习题目

DreamJudge 1133 求 1 到 n 的和

DreamJudge 1043 计算  $S_n$

DreamJudge 1040 利润提成

## 2.2 进制转换类问题

进制转换类的题目在绝大多数学校都是必考题目之一，这类题目的既基础又灵活，能看出学生的编程功底，所以这类题目一定要掌握。

总的来说，跟进制相关的题目可以分为以下几种题型

1、**反序数**：输入一个整数如 123，将其转换为反序之后的整数 321

2、**10 进制转 2 进制**：将一个 10 进制整数转化为一个 2 进制的整数

例如：7 转换为 111

3、**10 进制转 16 进制**：将一个 10 进制整数转化为一个 16 进制的整数

例如：10 转换为 A

4、**10 进制转 x 进制**：将一个 10 进制整数转化为一个 x 进制的整数

解析：这是前面两种的一种通解，如果会前面两种那么这个自然也触类旁通。

5、**x 进制转 10 进制**：将一个 x 进制整数转化为一个 10 进制的整数

解析：这是上一种情况的反例，看代码之后相信也能容易理解。

6、**x 进制转 y 进制**：将一个 x 进制整数转化为一个 y 进制的整数

解析：遇到这种情况，可以拆解为 x 先转为 10 进制，然后再将 10 进制转为 y 进制。

7、**字符串转浮点数**

例如：有一串字符串 31.25 将其转换为一个浮点数，可以先转整数部分，再转小数部分，最后相加即可。

8、**浮点数转字符串**

例如：有一个浮点数 23.45 将其转换为一个字符串进行存储，可以将整数和小数拆开再合并成一个字符串。

9、**字符串转整型和整形转字符串**

解析：直接用 atoi 函数和 itoa 函数即可。

## 反序数代码

```
1. #include <stdio.h>
2.
3. int main() {
4.     int n;
5.     scanf("%d", &n);
6.     int ans = 0; //将反序之后的答案存在这里
7.     while (n > 0) { //将 n 逐位分解
8.         ans *= 10;
9.         ans += (n % 10);
10.        n /= 10;
11.    }
12.    printf("%d\n", ans);
13.    return 0;
14. }
```

## 10 进制转 x 进制代码 (x 小于 10 的情况)

```
1. #include <stdio.h>
2.
3. int main() {
4.     int n, x;
5.     int s[105];
6.     //输入 10 进制 n 和 要转换的进制 x
7.     scanf("%d%d", &n, &x);
8.     int cnt = 0; //数组下标
9.     while (n > 0) { //将 n 逐位分解
10.        int w = (n % x);
11.        s[cnt++] = w;
12.        n /= x;
13.    }
14.    //反序输出
15.    for (int i = cnt - 1; i >= 0; i--) {
16.        printf("%d", s[i]);
17.    }
18.    printf("\n");
19.    return 0;
20. }
```

## 10 进制转 x 进制代码 (通用版)

```

1. #include <stdio.h>
2.
3. int main() {
4.     int n, x;
5.     char s[105]; //十进制以上有字符, 所以用 char 存储
6.     //输入 10 进制 n 和 要转换的进制 x
7.     scanf("%d%d", &n, &x);
8.     int cnt = 0; //数组下标
9.     while (n > 0) { //将 n 逐位分解
10.        int w = (n % x);
11.        if (w < 10) s[cnt++] = w + '0'; //变成字符需要加 '0'
12.        else s[cnt++] = (w - 10) + 'A'; //如果转换为小写则加 'a'
13.        //如果大于 10 则从 A 字符开始
14.        n /= x;
15.    }
16.    //反序输出
17.    for (int i = cnt - 1; i >= 0; i--) {
18.        printf("%c", s[i]);
19.    }
20.    printf("\n");
21.    return 0;
22.}

```

#### x 进制转 10 进制 (x 为 2 时)

```

1. #include <stdio.h>
2. #include <string.h>
3.
4. int main() {
5.     char s[105];
6.     //输入二进制字符串
7.     scanf("%s", &s);
8.     int ans = 0; //
9.     int len = strlen(s);
10.    for (int i = 0; i < len; i++) {
11.        if (s[i] == '0') {
12.            ans = ans * 2;
13.        }
14.        else {
15.            ans = ans * 2 + 1;
16.        }
17.    }

```



```
18.     printf("%d\n", ans);
19.     return 0;
20. }
```

### x 进制转 10 进制（通用版）

```
1.  #include <stdio.h>
2.  #include <string.h>
3.
4.  int main() {
5.      char s[105];
6.      int x;
7.      //输入 x 进制字符串 和 代表的进制 x
8.      scanf("%s%d", &s, &x);
9.      int ans = 0; //
10.     int len = strlen(s);
11.     for (int i = 0; i < len; i++) {
12.         ans = ans * x;
13.         if (s[i] >= '0' && s[i] <= '9') ans += (s[i] - '0');
14.         else ans += (s[i] - 'A') + 10;
15.     }
16.     printf("%d\n", ans);
17.     return 0;
18. }
```

### x 进制转 y 进制（通用版）

```
1.  #include <stdio.h>
2.  #include <string.h>
3.
4.  int main() {
5.      char s[105];
6.      int x, y;
7.      //输入二进制字符串 和 代表的进制 x 以及要转换的进制 y
8.      scanf("%s%d%d", &s, &x, &y);
9.      int ans = 0;
10.     int len = strlen(s);
11.     for (int i = 0; i < len; i++) {
12.         ans = ans * x;
13.         if (s[i] >= '0' && s[i] <= '9') ans += (s[i] - '0');
```

```
14.     else ans += (s[i] - 'A') + 10;
15.     }
16.     char out[105];
17.     int cnt = 0;
18.     while (ans > 0) {
19.         int w = (ans % y);
20.         if (w < 10) out[cnt++] = w + '0';
21.         else out[cnt++] = (w-10) + 'A';
22.         ans /= y;
23.     }
24.     for (int i = cnt - 1; i >= 0; i--) {
25.         printf("%c", out[i]);
26.     }
27.     printf("\n");
28.     return 0;
29. }
```

## 题型总结

这类题目任他千变万化，本质上都是不变的。就是数位的拆解与合并，拆解很明显就是两步，先取模然后除取整，合并就是先乘后加。只要掌握了以上的几类变化，不管题目如何变化，你都立于不败之地。

## 题目练习

DreamJudge 1454 反序数

DreamJudge 1178 进制转换

DreamJudge 1259 进制转换 2

DreamJudge 1176 十进制和二进制

DreamJudge 1380 二进制数

DreamJudge 1417 八进制

DreamJudge 1422 进制转换 3

DreamJudge 1097 负二进制

## 2.3 排版类问题

排版类问题也是机试中经常出现的题目，这类题目主要考验考生对代码的掌控程度。表面上看起来很简单，但是对于大部分没有认真研究过的同学来学，这些题可能会搞半天才能搞出来。

总的来说，排版类的题目可以以下几种题型为代表。

### 1、输出字符菱形 DreamJudge 1473

这类题目的变形可以是输出长方形、三角形、梯形等形状。

### 2、旋转数字输出

### 3、矩阵顺/逆指针旋转

### 4、矩阵翻转

这类题目的变形可以是轴对称翻转、中心对称翻转等。

### 5、杨辉三角形

### 6、2048 问题

以上，我们选择其中输出**字符菱形**和**杨辉三角形**进行详细讲解，其他题型我们给出解题思路以及题目编号，大家可以在本节后面的练习题目里找到并完成。如果自己无法理解并完成题目，请加入我们的机试交流群进行提问交流。

### 字符菱形

#### 题目描述：

输入一个整数  $n$  表示菱形的对角半长度，请你用\*把这个菱形画出来。

输入：3

输出：

```
*  
***  
*****
```

\*\*\*

\*

输入样例#:

1

输出样例#:

\*

题目来源:

DreamJudge 1473

**解题分析:** 对于这类题目, 我们可以将它进行分解。从中间切开, 上面一个三角形, 下面一个三角形。那么问题就转化为了如何输出三角形, 我们可以利用两个 for 循环控制来输出三角形。

参考代码

```
1. #include <stdio.h>
2.
3. int main() {
4.     int n;
5.     scanf("%d", &n);
6.     //上三角
7.     for (int i = 1; i <= n; i++) {
8.         for (int j = 1; j <= n - i; j++) {
9.             printf(" ");
10.        }
11.        for (int j = n - i + 1; j <= n; j++) {
12.            printf("*");
13.        }
14.        printf("\n");
15.    }
16.    //下三角 下三角只需要将上三角反过来输出就行
17.    for (int i = n - 1; i >= 1; i--) {
18.        for (int j = 1; j <= n - i; j++) {
19.            printf(" ");
20.        }
21.        for (int j = n - i + 1; j <= n; j++) {
22.            printf("*");
23.        }
```

```
24.     printf("\n");
25.     }
26.     return 0;
27. }
```

## 杨辉三角形

### 题目描述:

提到杨辉三角形, 大家应该都很熟悉. 这是我国宋朝数学家杨辉在公元 1261 年著书《详解九章算法》提出的。 1 1 1 1 2 1 1 3 3 1 1 4 6 4 1 1 5 10 10 5 1 1 6 15 20 15 6 1 我们不难其规律: S1: 这些数排列的形状像等腰三角形, 两腰上的数都是 1 S2: 从右往左斜着看, 第一列是 1, 1, 1, 1, 1, 1, 1; 第二列是, 1, 2, 3, 4, 5, 6; 第三列是 1, 3, 6, 10, 15; 第四列是 1, 4, 10, 20; 第五列是 1, 5, 15; 第六列是 1, 6……。 从左往右斜着看, 第一列是 1, 1, 1, 1, 1, 1, 1; 第二列是 1, 2, 3, 4, 5, 6……和前面的看法一样。我发现这个数列是左右对称的。 S3: 上面两个数之和就是下面的一行的数。 S4: 这行数是第几行, 就是第二个数加一。…… 现在要求输入你想输出杨辉三角形的行数 n; 输出杨辉三角形的前 n 行。

### 输入描述:

输入你想输出杨辉三角形的行数 n( $n \leq 20$ ); 当输入 0 时程序结束。

### 输出描述:

对于每一个输入的数, 输出其要求的三角形. 每两个输出数中间有一个空格. 每输完一个三角形换行。

### 输入样例#:

5

### 输出样例#:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

### 题目来源:

DreamJudge 1062

**解题分析：**这是一道特别经典的题目，我们只需要按照题意用二维数组去计算即可。对于任意一个数  $a[i][j]$ ，都有  $a[i][j] = a[i-1][j] + a[i-1][j-1]$ ；

### 参考代码

```
1. #include <stdio.h>
2. int main() {
3.     int a[21][21] = {0}; //数组里的所有值初始化为0
4.     int n;
5.     while (scanf("%d", &n) != EOF) {
6.         if (n == 0) break;
7.         a[1][1] = 1;
8.         for (int i = 2; i <= n; i++) {
9.             for (int j = 1; j <= i; j++) {
10.                a[i][j] = a[i-1][j] + a[i-1][j-1];
11.            }
12.        }
13.        for (int i = 1; i <= n; i++) {
14.            for (int j = 1; j <= i; j++) {
15.                printf("%d ", a[i][j]);
16.            }
17.            printf("\n");
18.        }
19.    }
20.    return 0;
21. }
```

**题型总结：**这类题目尽量在平时练习，解法主要就是把一个大问题进行分解，一部分一部分的实现。在考试的时候遇到，千万不要急，将问题进行分解，找到其中的规律，然后再写出来。当然，如果平时就有练习，那就不用担心了。

### 练习题目

DreamJudge 1392 杨辉三角形 - 西北工业大学

DreamJudge 1377 旋转矩 - 北航

DreamJudge 1216 旋转方阵

DreamJudge 1221 旋转矩阵

DreamJudge 1472 2048 游戏



## 2.4 日期类问题

日期类的题目也是常考的题目，这类题目一般都为以下几种考法。

1、判断某年是否为闰年

2、某年某月某日是星期几

变形问法：某日期到某日期之间有多少天

3、某天之后  $x$  天是几月几日

4、10:15 分之后  $x$  分钟是几点几分

变形问法：某点到某点之间有多少分或多少秒

注意输入时候一般用 scanf 解析输入值

如：2019-11-8 2019-11-08 2019/11/8 10:10

```
1. int year, month, day;
2. scanf("%d-%d-%d", &year, &month, &day);
3. scanf("%d/%d/%d", &year, &month, &day);
4. int hour, minute;
5. scanf("%d:%d", &hour, &minute);
```

noobdream.com

### 日期

#### 题目描述：

定义一个结构体变量（包括年、月、日），编程序，要求输入年月日，计算并输出该日在本年中第几天。

#### 输入描述：

输输入三个整数(并且三个整数是合理的, 既比如当输入月份的时候应该在 1 至 12 之间, 不应该超过这个范围) 否则输出 Input error!

#### 输出描述：

输出一个整数. 既输入的日期是本月的第几天。

#### 输入样例#：



1985 1 20

2006 3 12

输出样例#:

20

71

题目来源:

DreamJudge 1051

**解题分析:** 这个题目的考点在于两个地方, 一个是每个月的天数都不一样, 另一个是 2 月如果是闰年则多一天, 最后我们还要判断输入的日期是否存在, 如果不存在则输出 Input error!

参考代码

```
1. #include <stdio.h>
2.
3. struct node {
4.     int year, month, day;
5. }p;
6. int f[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
7. int main() {
8.     while (scanf("%d%d%d", &p.year, &p.month, &p.day) != EOF) {
9.         //判断是否闰年
10.        if ((p.year%400== 0) || (p.year%4==0)&&(p.year%100!=0)) {
11.            f[2] = 29;
12.        }
13.        else f[2] = 28;
14.        int flag = 0;
15.        //判断月份输入是否合法
16.        if (p.month < 1 || p.month > 12) flag = 1;
17.        //判断天的输入是否合法
18.        for (int i = 1; i <= 12; i++) {
19.            if (p.day < 0 || p.day > f[i]) {
20.                flag = 1;
21.            }
22.        }
23.        if (flag) {
```

```
24.         printf("Input error!\n");
25.         continue;
26.     }
27.     int ans = p.day;
28.     for (int i = 1; i < p.month; i++) {
29.         ans += f[i];
30.     }
31.     printf("%d\n", ans);
32. }
33. return 0;
34. }
```

## 题型总结

日期类的题目就是要特别注意闰年的判断, 这些题目一般都是考察代码细节的把握, 时间类的题目注意时间的转换, 1 天=24 小时, 1 小时=60 分, 1 分=60 秒。

**特别注意:** 一天之内时针和分针会重合 22 次, 而不是 24 次。

noobdream.com

## 练习题目

DreamJudge 1011 日期  
DreamJudge 1290 日期差值  
DreamJudge 1410 打印日期  
DreamJudge 1437 日期类  
DreamJudge 1446 日期累加  
DreamJudge 1053 偷菜时间表

## 2.5 字符串类问题

字符串类的问题也是各个院校必考的题型之一，基本上有以下这些考点：

- 1、统计字符个数
- 2、单词首字母大写
- 3、统计子串出现次数

解析：考察大家基础的字符串遍历能力。

- 4、文本加密/解密

解析：通过循环往后移动  $x$  位或直接给一个映射表是比较常见的考法。

- 5、文本中的单词反序

解析：灵活使用 `string` 可以秒杀这类题目，当然也可以用字符串一步步解析。

- 6、删除字符串（大小写模糊）

解析：如果大小写不模糊，那么就是直接找到之后删除。大小写模糊的话，只是多一个判断。

### 加密算法

#### 题目描述：

编写加密程序，加密规则为：将所有字母转化为该字母后的第三个字母，即  $A \rightarrow D$ 、 $B \rightarrow E$ 、 $C \rightarrow F$ 、.....、 $Y \rightarrow B$ 、 $Z \rightarrow C$ 。小写字母同上，其他字符不做转化。输入任意字符串，输出加密后的结果。

例如：输入 "I love 007"，输出 "L oryh 007"

#### 输入描述：

输入一行字符串，长度小于 100。

#### 输出描述：

输出加密之后的结果。

#### 输入样例#：

I love 007

#### 输出样例#：

L oryh 007

#### 题目来源：

DreamJudge 1014

**题目解析：**这是一道很常见的加解密考法，往后移动 3 位是这道题的核心，我们只需要按照题意将大写字母、小写字母、和其他分开进行处理就可以，具体看代码。

### 参考代码

```
1. #include <stdio.h>
2. #include <string.h>
3.
4. int main() {
5.     char s[105];
6.     gets(s); //输入一行文本用 gets
7.     int len = strlen(s);
8.     for (int i = 0; i < len; i++) {
9.         if (s[i] >= 'A' && s[i] <= 'Z') {
10.            s[i] += 3;
11.            if (s[i] > 'Z') s[i] -= 26; //溢出循环
12.        }
13.        else if (s[i] >= 'a' && s[i] <= 'z') {
14.            s[i] += 3;
15.            if (s[i] > 'z') s[i] -= 26; //溢出循环
16.        }
17.        else {
18.            continue;
19.        }
20.    }
21.    puts(s);
22.    return 0;
23. }
```

### 练习题目

DreamJudge 1012 字符移动

DreamJudge 1292 字母统计

DreamJudge 1240 首字母大写

DreamJudge 1394 统计单词

DreamJudge 1027 删除字符串 2

## 2.6 排序类问题

排序类的问题基本上是每个学校**必考**的知识点，所以它的重要性不言而喻。

如果你在网上一查，或者看数据结构书，十几种排序算法可以把你吓的魂不守舍。表面上看各种排序都有其各自的特点，那是不是我们需要掌握每一种排序呢？

答案自然是否定的。我们一种排序也不需要掌握，你需要会用一个 `sort` 函数就可以了，正所谓一个 `sort` 走天下。

`sort` 函数本质上是封装了快速排序，但是它做了一些优化，所以你只管用它就行了。

**复杂度为： $n\log n$**

所以 `sort` 可以对最大 30W 个左右的元素进行排序，可以应对考研机试中的 99.9% 的情况。

**sort 函数的用法**

`sort()` 函数: 依次传入三个参数，要排序区间的起点，要排序区间的终点+1，比较函数。比较函数可以不填，则默认为从小到大排序。

**sort 函数有两个常见的应用场景**

- 1、自定义函数排序
- 2、多级排序

### 成绩排序

**题目描述：**

输入任意（用户，成绩）序列，可以获得成绩从高到低或从低到高的排列, 相同成绩都按先录入排列在前的规则处理。

**示例：**

jack	70
peter	96
Tom	70

```
smith      67
从高到低  成绩
peter      96
jack       70
Tom        70
smith      67
从低到高
smith      67
jack       70
Tom        70
peter      96
```

**输入描述:**

输入多行, 先输入要排序的人的个数, 然后输入排序方法 0 (降序) 或者 1 (升序) 再分别输入他们的名字和成绩, 以一个空格隔开

**输出描述:**

按照指定方式输出名字和成绩, 名字和成绩之间以一个空格隔开

**输入样例#:**

```
3
0
fang 90
yang 50
ning 70
```

**输出样例#:**

```
fang 90
ning 70
yang 50
```

**题目来源:**

DreamJudge 1151

**题目解析：**这题唯一的一个考点在于稳定排序，sort 排序是不稳定的，排序之后相对次序有可能发生改变。解决这个问题有两个方法，一个是用 stable\_sort 函数，它的用法和 sort 一样，但是它是稳定的，所以如果我们遇到有稳定的需求的排序时，可以用它。另一个方法是给每一个输入增加一个递增的下标，然后二级排序，当值相同时，下标小的排在前面。

### 参考代码（稳定排序）

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. struct Student {
5.     string name;
6.     int grade;
7. }stu[1005];
8. //从大到小排序
9. bool compareDesc(Student a,Student b) {
10.     return a.grade > b.grade;
11. }
12. //从小到大排序
13. bool compareAsc(Student a,Student b) {
14.     return a.grade < b.grade;
15. }
16. int main() {
17.     int n,order;
18.     while(cin>>n) {
19.         cin>>order;
20.         for(int i=0;i<n;i++) {
21.             cin>>stu[i].name>>stu[i].grade;
22.         }
23.         if(order==0)
24.             stable_sort(stu,stu+n,compareDesc);
25.         else
26.             stable_sort(stu,stu+n,compareAsc);
27.         for(int i=0;i<n;i++) {
28.             cout<<stu[i].name<<" "<<stu[i].grade<<endl;
29.         }
30.     }
31.     return 0;
32. }
```

## 参考代码（标记 id）

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. struct Student {
5.     string name;
6.     int grade, id;
7. }stu[1005];
8. //从大到小排序
9. bool compareDesc(Student a, Student b) {
10.     if (a.grade == b.grade) return a.id < b.id;
11.     return a.grade > b.grade;
12. }
13. //从小到大排序
14. bool compareAsc(Student a, Student b) {
15.     if (a.grade == b.grade) return a.id < b.id;
16.     return a.grade < b.grade;
17. }
18. int main() {
19.     int n, order;
20.     while(cin >> n) {
21.         cin >> order;
22.         for(int i=0; i<n; i++) {
23.             cin >> stu[i].name >> stu[i].grade;
24.             stu[i].id = i; //通过标记 ID 进行判断
25.         }
26.         if(order == 0)
27.             sort(stu, stu+n, compareDesc);
28.         else
29.             sort(stu, stu+n, compareAsc);
30.         for(int i=0; i<n; i++) {
31.             cout << stu[i].name << " " << stu[i].grade << endl;
32.         }
33.     }
34.     return 0;
35. }
```



## 排序

### 题目描述:

输入  $n$  个数进行排序, 要求先按奇偶后按从小到大的顺序排序。

### 输入描述:

第一行输入一个整数  $n$ , 表示总共有多少个数,  $n \leq 1000$ 。

第二行输入  $n$  个整数, 用空格隔开。

### 输出描述:

输出排序之后的结果。

### 输入样例#:

```
8
1 2 3 4 5 6 7 8
```

### 输出样例#:

```
1 3 5 7 2 4 6 8
```

### 题目来源:

DreamJudge 1010

**题目解析:** 题目要求我们按照奇数在前偶数在后的排序方法, 同为奇数或同为偶数再从小到大排序。我们有两种简便的方法可以解决这个问题, 其一是我们将奇数和偶数分离开来, 然后分别排好序, 再合并在一起。其二是使用 `sort` 进行二级排序, 这里我们采用第二种方法进行演示。

### 参考代码

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. bool cmp(int a,int b){
5.     if(a % 2 == b % 2)//如果同奇同偶
6.         return a < b;//直接从小到大排序
7.     else//如果奇偶性不同
8.         return (a%2) > (b%2); //奇数在偶数前
```

```
9. }
10. int main() {
11.     int n;
12.     int a[1005] = {0};
13.     cin >> n;
14.     for (int i = 0; i < n; i++) {
15.         cin >> a[i];
16.     }
17.     sort(a, a+n, cmp);
18.     for(int i = 0; i < n; i++) {
19.         cout << a[i] << " ";
20.     }
21.     cout << endl;
22.     return 0;
23. }
```

小结：由上面可以看出，只要我们掌握好 sort 的用法，不管什么样的花里胡哨的排序，我们都可以一力破之。

### 一些特殊的排序题

1、如果题目给的数据量很大，上百万的数据要排序，但是值的区间范围很小，比如值最大只有 10 万，或者值的范围在 1000W 到 1010W 之间，对于这种情况，我们可以采用空间换时间的计数排序。

2、字符串的字典序排序是一个常见的问题，需要掌握，也是用 sort。

下面两种情况了解即可，追求满分的同学需要掌握

3、如果题目给你一个数的序列，要你求逆序数对有多少，这是一个经典的问题，解法是在归并排序合并是进行统计，复杂度可以达到  $n\log n$ 。如果数据量小，直接冒泡排序即可。

4、如果题目让你求 top10，即最大或最小的 10 个数，如果数据量很大，建议使用选择排序，也就是一个一个找，这样复杂度比全部元素排序要低。

5、如果题目给的数据量有几百万，让你从中找出第 K 大的元素，这时候 sort 是会超时的。解法是利用快速排序的划分的性质，进入到其中一个分支继续寻找，

以上都是一些数据很特殊且数据量非常大的情况下的解决方案。

## 练习题目

DreamJudge 1106 排序 2  
DreamJudge 1159 成绩排序 2.0  
DreamJudge 1217 国名排序  
DreamJudge 1227 日志排序  
DreamJudge 1248 整数奇偶排序  
DreamJudge 1254 字符串排序  
DreamJudge 1255 字符串排序 2  
DreamJudge 1261 字符串排序 3  
DreamJudge 1294 后缀子串排序  
DreamJudge 1310 奥运排序问题  
DreamJudge 1338 EXCEL 排序  
DreamJudge 1360 字符串内排序  
DreamJudge 1399 排序 - 华科  
DreamJudge 1400 特殊排序  
DreamJudge 1404 成绩排序 - 华科  
DreamJudge 1412 大整数排序

## 2.7 查找类问题

查找是一类我们必须掌握的算法，它不仅会在题目中直接考察，同时也可能是其他算法中的重要组成部分。本章中介绍的查找类问题都是单独的基础查找问题，对于这类基础查找的问题，我们应该将它完全掌握。

查找类题目一般有以下几种考点

- 1、**数字查找**：给你一堆数字，让你在其中查找  $x$  是否存在  
题目变形：如果  $x$  存在，请输出有几个。
- 2、**字符串查找**：给你很多个字符串，让你在其中查找字符串  $s$  是否存在

顺序查找就不说了，这个大家会。

什么时候不能用顺序查找呢？

很明显，当满足下面这种情况的时候

- 1、数据量特别大的时候，比如有 10W 个元素。
- 2、查询次数很多的时候，比如要查询 10W 次。

遇到这类题大多数人的想法是先 sort 排序，然后二分查找，这是一个很常规的解决这类问题的方法。

但是，我们不推荐你这么做，我们有更简单易用且快速的方法。我们推荐你了解并使用 map 容器。

前面介绍过 map，它是 STL 的一种关联式容器，它的底层是红黑树实现的，也就意味着它的插入和查找操作都是  $\log$  级别的。

相信每一个用过 map 的同学，都会情不自禁的说一句，map 真香！

## 查找学生信息 2

### 题目描述:

输入 N 个学生的信息, 然后进行查询。

### 输入描述:

输入的第一行为 N, 即学生的个数 ( $N \leq 1000$ )

接下来的 N 行包括 N 个学生的信息, 信息格式如下:

01 李江 男 21

02 刘唐 男 23

03 张军 男 19

04 王娜 女 19

然后输入一个 M ( $M \leq 10000$ ), 接下来会有 M 行, 代表 M 次查询, 每行输入一个学号, 格式如下:

02

03

01

04

### 输出描述:

输出 M 行, 每行包括一个对应于查询的学生的信息。

如果没有对应的学生信息, 则输出 “No Answer!”

### 输入样例#:

4

01 李江 男 21

02 刘唐 男 23

03 张军 男 19

04 王娜 女 19

5

02

03

01

04

03

输出样例#:

02 刘唐 男 23

03 张军 男 19

01 李江 男 21

04 王娜 女 19

03 张军 男 19

题目来源:

DreamJudge 1476

**题目解析:** 对于这类查询量大的题目, 我们有两种方法来解决这个问题。第一是将学号先排好序, 然后使用二分查找, 但是很多同学写二分的时候容易出现问題, 而且代码量也比较大, 我们不推荐这种做法。推荐大家使用 map 来解决这类问题, 基本上 map 可以通过 99.9% 的这类题目。

参考代码

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. struct node{
5.     string num;
6.     string name;
7.     string sex;
8.     int age;
9. };
10. int main(){
11.     int n,q;
12.     map<string, node> M;//定义一个 map 映射
13.     while(scanf("%d", &n)!=EOF){
14.         for(int i=0;i<n;i++){
15.             node tmp;
16.             cin>>tmp.num>>tmp.name>>tmp.sex>>tmp.age;
17.             M[tmp.num] = tmp;//将学号指向对应的结构体
18.         }
19.         scanf("%d", &q);
```

```

20.     for(int i=0;i<q;i++){
21.         string num;
22.         cin>>num;
23.         if((M.find(num))!=M.end())//find 查找 如果找不到则返回末尾
24.             cout<<M[num].num<<" "<<M[num].name<<" "<<M[num].sex<<" "<<M[num].age<<endl;

25.     else
26.         cout<<"No Answer!"<<endl;
27.     }
28. }
29. return 0;
30. }
    
```

可以发现, 用 map 解决这个题目的时候, 不用去考虑字符串排序的问题, 也不用想二分查找会不会写出问题, 直接用 map, 所有的烦恼都没有了, 而且它的复杂度和二分查找是一个量级的。

上面讲的是一类静态查找的问题, 实际中为了增加思维难度或代码难度, 会经过一定的改变变成动态查找问题。

### 动态查找问题

#### 题目描述:

有  $n$  个整数的集合, 想让你从中找出  $x$  是否存在。

#### 输入描述:

第一行输入一个正整数  $n$  ( $n < 100000$ )

第二行输入  $n$  个正整数, 用空格隔开。

第三行输入一个正整数  $q$  ( $q < 100000$ ), 表示查询次数。

接下来输入  $q$  行, 每行一个正整数  $x$ , 查询  $x$  是否存在。

#### 输出描述:

如果  $x$  存在, 请输出 find, 如果不存在, 请输出 no, 并将  $x$  加入到集合中。

#### 输入样例#:

5

1 2 3 4 5

3  
6  
6  
3

输出样例#:

no  
find  
find

题目来源:

DreamJudge 1477

**题目解析:** 通过分析题目我们可以发现, 这道题有一个特点就是, 数的集合在不断的改变。如果我们用先排序再二分的方法就会遇到困难, 因为加入新的数的时候我们需要去移动多次数组, 才能将数插入进去, 最坏情况每次插入都是  $O(n)$  的复杂度, 这是无法接受的。当然也不是说就不能用这样的方法来解决, 可以用离线的方法来解决这个问题, 但是这样做太复杂, 不适合在考试中使用。那么我们考虑用 map 来解决这个问题。

参考代码

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. int main(){
5.     int n,q,x;
6.     map<int, int> M;//定义一个map映射
7.     scanf("%d", &n);
8.     for (int i = 0; i < n; i++) {
9.         scanf("%d", &x);
10.        M[x]++;//记录集合中x有多少个
11.    }
12.    scanf("%d", &q);
13.    for (int i = 0; i < q; i++) {
14.        scanf("%d", &x);
```



```

15.         if (M[x] == 0) { //如果 x 的个数为 0
16.             printf("no\n");
17.             M[x]++; //将 x 加入到集合中
18.         }
19.         else printf("find\n");
20.     }
21.     return 0;
22. }

```

看了上面的代码，是不是发现用 map 来解决问题真是超级简单。所以学会灵活使用 map 将能极大的拉近你和大佬之间的距离，我们一起来学 map 吧！

当然不是说二分查找就没用了，我们也需要了解二分查找的原理，只不过。二分的前提是单调性，只要满足单调性就可以二分，不论是单个元素还是连续区间。下面我们也给出一个基本的二分查找代码，供大家参考。

```

1.  #include <bits/stdc++.h>
2.  using namespace std;
3.
4.  int a[10005];
5.  int main(){
6.      int n,x;
7.      scanf("%d", &n); //输入 n 个数
8.      for (int i = 1; i <= n; i++) {
9.          scanf("%d", &a[i]);
10.     }
11.     sort(a+1, a+1+n); //排序保持单调性
12.     scanf("%d", &x); //要查找的数 x
13.     int l = 1, r = n;
14.     while (l < r) {
15.         int mid = (l + r) / 2;
16.         if (a[mid] == x) {
17.             printf("find\n");
18.             return 0;
19.         }
20.         if (a[mid] > x) { //如果 x 比中间数小
21.             r = mid - 1; //说明在左区间
22.         }
23.         else l = mid + 1; //否则在右区间内

```

```
24.    }  
25.    printf("not find\n");  
26.    return 0;  
27. }
```

### 练习题目

DreamJudge 1177 查找学生信息

DreamJudge 1388 查找 1

DreamJudge 1387 查找 - 北邮

DreamJudge 1383 查找第 K 小数



## 2.8 贪心类问题

贪心类问题是很常见的考点，贪心算法更重要的是一种贪心的思想，它追求的是当前最优解，从而得到全局最优解。贪心类问题基本上算是必考题型之一，它能更好的考察出学生的思维能力以及对问题的分析能力，很多学校的出题人都非常爱出贪心类的题目。

贪心算法的定义：

贪心算法是指在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，只做出在某种意义上的局部最优解。

贪心算法不是对所有问题都能得到整体最优解，关键是贪心策略的选择，选择的贪心策略必须具备无后效性，即某个状态以前的过程不会影响以后的状态，只与当前状态有关。

贪心可以很简单，简单到让所有人一眼就能看出来该怎么做。贪心也可以很难，难到让你没办法去证明这样贪心的正确性。所以要想解决贪心这类问题，主要还是看你的悟性，看你对题目的分析能力如何，下面我们举例说明。

**例子 1：**地上有 3 张纸币，分别是 5 元、1 元和 10 元，问你只能拿一张，最多能拿多少钱？

解析：很明显，10 元。

**例子 2：**地上有  $n$  张纸币，有 1 元的，有 5 元的，还要 10 元的，问你只能拿一张，最多能拿多少钱？

解析：很明显，还是 10 元。

**例子 3：**地上有很多纸币，有  $a$  张 1 元的，有  $b$  张 5 元的，还要  $c$  张 10 元的，问你从中拿  $x$  张，最多能拿多少钱？

解析：大家应该都能想到，肯定是优先拿 10 元的，如果 10 元的拿完了，再拿 5 元的，最后才会拿 1 元的。这就是贪心的思想，所以贪心其实是很容易想到的。

**例子 4：**有  $n$  个整数构成的集合，现在从中拿出  $x$  个数来，问他们的和最大能是多少？

解析：相信大家都能想到，优先拿大的，从大到小一个个拿，这样组成的和最大。那么在解决这个问题之前，我们需要先排序，从大到小的排好序，然后将前  $x$  个数的和累加起来就是答案。

从上面几个例子中, 相信大家对贪心已经有了初步的了解。我们使用贪心的时候, 往往需要先按照某个特性先排好序, 也就是说贪心一般和 sort 一起使用。

### 喝饮料

#### 题目描述:

商店里有  $n$  中饮料, 第  $i$  种饮料有  $m_i$  毫升, 价格为  $w_i$ 。

小明现在手里有  $x$  元, 他想吃尽量多的饮料, 于是向你寻求帮助, 怎么样买才能吃的最多。

请注意, 每一种饮料都可以只买一部分。

#### 输入描述:

有多组测试数据。

第一行输入两个非负整数  $x$  和  $n$ 。

接下来  $n$  行, 每行输入两个整数, 分别为  $m_i$  和  $w_i$ 。

所有数据都不大于 1000。

$x$  和  $n$  都为 -1 时程序结束。

#### 输出描述:

请输出小明最多能喝到多少毫升的饮料, 结果保留三位小数。

#### 输入样例#:

233 6

6 1

23 66

32 23

66 66

1 5

8 5

-1 -1

#### 输出样例#:

136.000

#### 题目来源:

DreamJudge 1478

题目解析：通过分析之后我们可以发现，小明想要喝尽量多的饮料的话，肯定优先选择性价比最高的饮料喝，也就是说 1 毫升的价格最低的饮料先喝，那么我们就需要去比较，每种饮料 1 毫升的价格是多少。然后按照这个单价从低到高依次排序，然后一个一个往后喝，这样可以保证小明能喝到最多的饮料。

### 参考代码

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. struct node {
5.     double w, m;
6. }p[1005];
7. bool cmp(node a, node b) {
8.     //按照每毫升的价格从低到高排序
9.     return a.w/a.m < b.w/b.m;
10. }
11. int main(){
12.     int n,x;
13.     while (scanf("%d%d", &x, &n) != EOF) {
14.         if (x == -1 && n == -1) break;
15.         for (int i = 1; i <= n; i++) {
16.             scanf("%lf%lf", &p[i].m, &p[i].w);
17.         }
18.         sort(p+1, p+1+n, cmp);
19.         double ans = 0;
20.         for (int i = 1; i <= n; i++) {
21.             if (x >= p[i].w) {//如果剩余的钱能全买
22.                 ans += p[i].m;
23.                 x -= p[i].w;
24.             }
25.             else {//如果剩余的钱买不完这种饮料
26.                 ans += (p[i].m*x/p[i].w);
27.                 break;//到这里 x 已经为 0 了
28.             }
29.         }
30.         printf("%.3lf\n", ans);
31.     }
32.     return 0;
```

### 解题的通用步骤

- 1、建立数学模型来描述问题；
- 2、把求解的问题分成若干个子问题；
- 3、对每一子问题求解，得到子问题的局部最优解；
- 4、把子问题的局部最优解合成原来问题的一个解。

### 题型总结

贪心问题在很多机试难度低的学校，可以成为压轴题，也就是通过人数最少的题目。在机试难度高的学校也是中等难度及以上的题目，为什么明明贪心看起来这么容易的题目，却成为大多数学生过不去的坎呢？原因有二，一是很多同学根本就没有想到这个题目应该用贪心算法，没能将题目抽象成数学模型来分析，简单说就是没有读懂题目隐藏的意思。二是读懂题了，知道应该是贪心算法解这个题目，但是排序的特征点却没有找准，因为不是所有题目都是这么明显的看出来从小到大排序，有的题目可能隐藏的更深，但是这种难度的贪心不常见。所以机试中的贪心题，你要你反应过来这是一个贪心，99%的情况下都能解决。

### 练习题目

DreamJudge 1307 组队刷题

DreamJudge 1347 To Fill or Not to Fill

## 2.9 链表类问题

链表类问题属于选读章节, 对于使用 OJ 测评的院校的同学来说, 这类问题可以用数组来实现, 没有必要用链表去实现, 写起来慢不说, 还容易出错, 所以我们一般都直接用数组来实现, 反正最后 OJ 能 AC 就行, 建议这类同学跳过本节或仅做了解即可。但是对于非 OJ 测评的院校来说, 链表类问题可以说是必考的题型。

一般来说有以下三种常见考点

### 1、猴子报数

解析: 循环链表建立之后, 按照题意删除节点。

### 2、两个有序链表合并为一个

解析: 这个和两个有序数组合并为一个有序数组原理一样。

### 3、链表排序

解析: 使用冒泡排序进行链表排序, 因为冒泡排序是相邻两个元素进行比较交换, 适合链表。

### 猴子报数

#### 题目描述:

$n$  个猴子围坐一圈并按照顺时针方向从 1 到  $n$  编号, 从第  $s$  个猴子开始进行 1 到  $m$  的报数, 报数到第  $m$  的猴子退出报数, 从紧挨它的下一个猴子重新开始 1 到  $m$  的报数, 如此进行下去知道所有的猴子都退出为止。求给出这  $n$  个猴子的退出的顺序表。

#### 输入描述:

有做组测试数据。每一组数据有两行, 第一行输入  $n$  (表示猴子的总数最多为 100) 第二行输入数据  $s$  (从第  $s$  个猴子开始报数) 和数据  $m$  (第  $m$  个猴子退出报数)。当输入 0 0 0 时表示程序结束。

#### 输出描述:

每组数据的输出结果为一行, 中间用逗号间隔。

#### 输入样例#:

10

2 5

5  
2 3  
0  
0 0

输出样例#:

6, 1, 7, 3, 10, 9, 2, 5, 8, 4  
4, 2, 1, 3, 5

题目来源:

DreamJudge 1081

**题目解析:** 我们需要创建一个首尾相连的循环链表, 然后先走  $s$  步, 再开始循环遍历链表, 每走  $m$  步删除一个节点, 知道链表中只能下一个节点时结束循环。只能一个节点的判断条件是, 它的下一个指针指向的是它, 说明它自循环了。

```
1. #include <stdio.h>
2. #include <malloc.h>
3. struct node {
4.     int num;
5.     struct node *next;
6. };
7. int n, s, m;
8. //创建循环链表
9. struct node* create() {
10.     struct node *head, *now, *pre;
11.     for (int i = 1; i <= n; i++) {
12.         now = (struct node *)malloc(sizeof(node));
13.         if (i == 1) { //第一个节点需要处理
14.             head = now; //头结点指向第一个节点
15.             pre = now; //上一个节点也指向它
16.         }
17.         now->num = i;
18.         now->next = head;
19.         pre->next = now;
20.         pre = now; //将当前节点作为上一个节点
21.     }
22.     return head;
```



```
23. };
24. //按照题目要求输出
25. void print(struct node *head) {
26.     struct node *p;
27.     p = head;
28.     while (s--) { //先走 s 步
29.         p = p->next;
30.     }
31.     int i = 1;
32.     while (p != NULL) {
33.         if (p == p->next) { //只剩最后一个
34.             printf("%d\n", p->num);
35.             break;
36.         } //这里有个小技巧 我们遍历到满足条件的上一个
37.         if ((i+1) % (m-1) == 0) { //然后输出下一个
38.             printf("%d,", p->next->num); //这样方便删除
39.             p->next = p->next->next; //删除下一个节点
40.         }
41.         p = p->next;
42.         i++;
43.     }
44. }
45. int main(){
46.     while (scanf("%d%d%d", &n, &s, &m) != EOF) {
47.         if (n==0&&s==0&&m==0) break;
48.         struct node *head;
49.         head = create();
50.         print(head);
51.     }
52.     return 0;
53. }
```

## 练习题目

DreamJudge 1015 单链表

DreamJudge 1018 击鼓传花

DreamJudge 1025 合并链表

DreamJudge 1405 遍历链表

## 第三章 数学

本章我们重点讲解一些常见的数学题型，包括同模余定理、最大公约数（GCD）、最小公倍数（LCM）、斐波那契数列、素数判定、素数筛选、分解素因数、二分快速幂、常见数学公式总结、规律神器 OEIS 等内容。希望能帮助读者更好的掌握计算机考研机试中所涉及到的数学问题。

数学算是一个很常考的类别，毕竟计算机的基础是数学，就如人类的本质是复读机一样。所以，出题人为了展现自己的计算机深厚的功底，一般都会出个数学题来刷一下存在感。

所以，认真看完这一章的内容对你的帮助会很大，加油，fighting!

N 诺  
noobdream.com

### 3.1 同模余定理

同余模是一个大家容易忽视的点, 它一般不会单独作为考点出现, 而是在其他类型的题目中作为一个常识的形式出现。

#### 定义

所谓的同余, 顾名思义, 就是许多的数被一个数  $d$  去除, 有相同的余数。 $d$  数学上的称谓为模。如  $a = 6$ ,  $b = 1$ ,  $d = 5$ , 则我们说  $a$  和  $b$  是模  $d$  同余的。因为他们都有相同的余数 1。

数学上的记法为:

$$a \equiv b \pmod{d}$$

可以看出当  $n < d$  的时候, 所有的  $n$  都对  $d$  同商, 比如时钟上的小时数, 都小于 12, 所以小时数都是模 12 的同余。对于同余有三种说法都是等价的, 分别为:

- (1)  $a$  和  $b$  是模  $d$  同余的.
- (2) 存在某个整数  $n$ , 使得  $a = b + nd$ .
- (3)  $d$  整除  $a - b$ .

可以通过换算得出上面三个说话都是正确而且是等价的.

#### 定律

同余公式也有许多我们常见的定律, 比如相等律, 结合律, 交换律, 传递律... 如下面的表示:

- 1)  $a \equiv a \pmod{d}$
- 2)  $a \equiv b \pmod{d} \rightarrow b \equiv a \pmod{d}$
- 3)  $(a \equiv b \pmod{d}, b \equiv c \pmod{d}) \rightarrow a \equiv c \pmod{d}$

如果  $a \equiv x \pmod{d}, b \equiv m \pmod{d}$ , 则

- 4)  $a+b \equiv x+m \pmod{d}$
- 5)  $a-b \equiv x-m \pmod{d}$
- 6)  $a*b \equiv x*m \pmod{d}$

#### 应用

$$\begin{aligned}(a+b) \% c &= (a \% c + b \% c) \% c; \\ (a-b) \% c &= (a \% c - b \% c) \% c; \\ (a*b) \% c &= (a \% c * b \% c) \% c;\end{aligned}$$

## 求 $S(n)$

### 题目描述:

$$S(n) = n^5$$

求  $S(n)$  除以 3 的余数

### 输入描述:

每行输入一个整数  $n$ , ( $0 < n < 1000000$ )

处理到文件结束

### 输出描述:

输出  $S(n) \% 3$  的结果并换行

### 输入样例#:

1  
2

### 输出样例#:

1  
2

### 题目来源:

DreamJudge 1500

**题目解析:** 通过分析我们可以发现,  $n$  虽然不大, 但是  $n^5$  却超过 `long long` 的范围, 所幸的是题目只要我们对答案 $\%3$ , 这时候我们就可以运用同余模定理。

$$S(n) \% 3 = (n^5) \% 3 = (n * n * n * n * n) \% 3 = ((n \% 3) * (n \% 3) * (n \% 3) * (n \% 3) * (n \% 3)) \% 3$$

### 参考代码

```
1. #include<stdio.h>
2. #include<iostream>
3. using namespace std;
4.
5. int main() {
6.     long long int n;
7.     while(~scanf("%lld",&n)) {
8.         long long int s=n;
9.         // 同余模定理:
10.        for(int i=1;i<5;i++) {
```

```

11.         s=((s%3)*(n%3));
12.     }
13.     printf("%lld\n",s%3);
14. }
15.     return 0;
16. }
```

另一类考点就是对大数进行取模

对于大数的求余, 联想到进制转换时的方法, 得到

举例如下, 设大数  $m=1234$ , 模  $n$

就等于

$((((1 * 10) \% n + 2 \% n) \% n * 10 \% n + 3 \% n) \% n * 10 \% n + 4 \% n) \% n$

参考代码 (大数取余)

```

1.  #include <stdio.h>
2.
3.  char s[1000];
4.  int main() {
5.      int i, j, k, m, n;
6.      while(~scanf("%s", s, &n)) {
7.          m = 0;
8.          for(i = 0; s[i] != '\0'; i++)
9.              m = ((m * 10) \% n + (s[i] - '0') \% n) \% n;
10.         printf("%d\n", m);
11.     }
12.     return 0;
13. }
```

**特别提醒:** 同余模定理的运算不适用于除法, 对于除法取模的运算我们一般使用逆元来解决问题, 后面的章节中会详细给大家讲解。

## 3.2 最大公约数 (GCD)

最大公约数, 相信大家高中的时候就知道了, 如何用计算机来快速求解两个数的最大公约数是一个很常见的数学问题。

如果题目要求我们求两个数  $x$  和  $y$  的最大公约数, 我们只需要记住下面这种方法就行了。

```
1. #include <stdio.h>
2.
3. int gcd(int a, int b) {
4.     if (b == 0) return a;
5.     else return gcd(b, a % b);
6. }
7. int main() {
8.     int x, y;
9.     scanf("%d%d", &x, &y);
10.    printf("%d\n", gcd(x, y));
11. }
```

显然, 大部分时候题目都不会出的这么直接, 那么对于最大公约数一般会有哪些变形考法呢?

### 最简真分数

#### 题目描述:

给出  $n$  个正整数, 任取两个数分别作为分子和分母组成最简真分数, 编程求共有几个这样的组合。

#### 输入描述:

每组包含  $n$  ( $n \leq 600$ ) 和  $n$  个数, 整数大于 1 且小于等于 1000。

#### 输出描述:

每行输出最简真分数组合的个数。

#### 输入样例#:

```
7
3 5 7 9 11 13 15
```

#### 输出样例#:

```
17
```

#### 题目来源:

DreamJudge 1180

题目解析：通过分析题意可以发现，最简真分数的必要条件就是不可以继续约分，那么不可以继续约分，就说明分子和分母的最大公约数为 1。因此，我们只需要枚举所有组合的情况然后判断 GCD 即可。

### 参考代码

```
1. #include <stdio.h>
2.
3. int gcd(int a, int b) {
4.     if(b==0) return a;
5.     else return gcd(b, a%b);
6. }
7. int main() {
8.     int buf[605];
9.     int ans, n;
10.    while(scanf("%d", &n)!=EOF) {
11.        for(int i=0; i<n; i++)
12.            scanf("%d", &buf[i]);
13.        ans=0; //答案个数
14.        for(int i=0; i<n; i++)
15.            for(int j=i+1; j<n; j++)
16.                if (gcd(buf[i], buf[j])==1) ans++;
17.        printf("%d\n", ans);
18.    }
19.    return 0;
20. }
```

### 另一种变形考法是：分数化简

例如：给你一个分数 12/30，让你将它化简，很明显，我们都知道它的答案是 2/5。  
那么：如果给你一个分数 x/y 呢？如何化简？

我们可以得出： $x/y = (x/\text{gcd}(x, y)) / (y/\text{gcd}(x, y))$

那么只用求出他们的最大公约数，然后除一下就可以得到答案了。

### 3.3 最小公倍数 (LCM)

对于求两个数的最小公倍数，只需要记住下面这个公式即可。

$$\text{LCM}(x, y) = x * y / \text{GCD}(x, y)$$

翻译一下就是：两个数的最小公倍数等于两个数的乘积除以两个数的最大公约数。

所以要求两个数的最小公倍数，我们只需求出他们的最大公约数即可。

上面的式子经过变形，可以很容易得到下面这个式子

$$x * y = \text{LCM}(x, y) * \text{GCD}(x, y)$$

延伸出考点：

1、给你两个数的乘积和这两个数的最小公倍数，问你这两个数的最大公约数是多少？

**解析：**很明显，我们通过上面的公式可知，乘积除以最小公倍数就是答案。

2、给你两个数的最大公约数和最小公倍数，问你这两个数的和最大和最小可能是多少？

**解析：**这个问题留给读者思考。



### 3.4 斐波那契数列

#### 定义

斐波那契数列指的是这样一个数列 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368.....  
这个数列从第 3 项开始, 每一项都等于前两项之和。

#### 公式

$$F(n) = F(n-1) + F(n-2)$$

斐波那契一般不会作为一个直接考点出现, 都会结合一些规律让你去推导, 然后发现题目的规律原来是一个斐波那契数列。

在本章最后一个小节中, 用了一个斐波那契的题目作为例子。

我们需要注意的考点是

- 1、这个数列的上升速度非常快, 很容易超过 `int` 和 `long long` 的范围
- 2、如果对答案取模, 题目就可能要求我们计算第 10000000 项的值, 我们就可以直接使用公式求解, 后面的章节也会讲到用矩阵快速幂求解的方法。

$$\therefore F(n) = 1/\sqrt{5} \left[ \left( \frac{1+\sqrt{5}}{2} \right)^n - \left( \frac{1-\sqrt{5}}{2} \right)^n \right]$$

- 3、如果给你一个数列:  $a(1) = 1, a(n+1) = 1 + 1/a(n)$ 。  
那么它的通项公式为:  $a(n) = \text{fib}(n+1) / \text{fib}(n)$ 。

## 3.5 素数判定

如何判断一个数  $x$  是不是素数呢?

我们可以根据素数的定义从 2 到小于这个数  $x$  的每个数去除, 看是否能除尽。一般情况下, 我们没有必要判断这么多个数, 只用判断到  $\text{sqrt}(x)$  就停止了。  
**因为:** 如果比  $\text{sqrt}(x)$  大的数能除尽的话, 就必然存在一个比  $\text{sqrt}(x)$  小的数能被除尽。

### 判断素数

**题目描述:**

输入一个整数, 判断该整数是否为素数, 若是, 输出该整数, 若否, 输出大于该整数的第一个素数。(例如, 输入为 14, 输出 17, 因为 17 是大于 14 的第一个素数)

**输入描述:**

输入一个整数  $n$ ,  $n$  最大为 10000。

**输出描述:**

按题意输出。

**输入样例#:**

14

**输出样例#:**

17

**题目来源:**

DreamJudge 1013

**题目解析:** 我们首先判断输入的  $n$  是不是一个素数, 如果是的话就直接输出。如果不是的话, 我们从  $n+1$  开始, 对每个数去判断它是不是一个素数, 直到找到一个素数的时候终止。

**参考代码**

```
1. #include <stdio.h>
2. #include <math.h>
3.
4. int main() {
5.     int n;
6.     scanf("%d", &n);
7.     if (n == 1) n++; //1 不是素数
```

```
8.     for (int i = n; ; i++) {  
9.         int flag = 0;  
10.        for (int j = 2; j <= sqrt(i); j++) {  
11.            if (i % j == 0) { //如果找到了约数  
12.                flag = 1; //说明不是素数  
13.                break;  
14.            }  
15.        }  
16.        if (flag == 0) {  
17.            printf("%d\n", i);  
18.            break;  
19.        }  
20.    }  
21.    return 0;  
22. }
```



### 练习题目

DreamJudge 1355 素数判定 - 哈尔滨工业大学

noobdream.com

### 3.6 素数筛选

有的时候，题目要求我们筛选出一段区间内的素数，我们就需要掌握一种素数筛选的方法。

如果用上一节素数判定的方法去判定每一个数是不是素数的话。  
复杂度是  $O(n \cdot \sqrt{n})$ ，大概能处理到 10000 以内的数。

如果题目要求的范围更大，那么我们就需要一种更为高效的筛选法。  
掌握下面这一种线性复杂度的筛选方法就足够我们应对任何情况。

```
1. // 线性素数筛选 prime[0]存的是素数的个数
2. const int maxn = 1000000 + 5;
3. int prime[maxn];
4. void getPrime() {
5.     memset(prime, 0, sizeof(prime));
6.     for (int i = 2; i <= maxn; i++) {
7.         if (!prime[i]) prime[++prime[0]] = i;
8.         for (int j = 1; j <= prime[0] && prime[j] * i <= maxn; j++) {
9.             prime[prime[j] * i] = 1;
10.            if (i % prime[j] == 0) break;
11.        }
12.    }
13. }
```

#### 素数判定

**题目描述:**

给你两个数 a、b, 现在的问题是要判断这两个数组成的区间内共有多少个素数

**输入描述:**

多组测试数据。 每个测试数据输入两个数 a、b。 ( $2 \leq a, b \leq 1000$ )

**输出描述:**

输出该区间内素数的个数。

**输入样例#:**

2 4

4 6

输出样例#:

2

1

题目来源:

DreamJudge 1102

**题目解析:** 这道题的数据范围不大, 我们可以用挨个暴力判断的方法来解决。我们假设这道题的数据范围很大, 使用素数筛选的方法来解决这个问题。

参考代码

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. // 线性素数筛选 prime[0]存的是素数的个数
5. const int maxn = 1000000 + 5;
6. int prime[maxn];
7. void getPrime() {
8.     memset(prime, 0, sizeof(prime));
9.     for (int i = 2; i <= maxn; i++) {
10.         if (!prime[i]) prime[++prime[0]] = i;
11.         for (int j = 1; j <= prime[0] && prime[j] * i <= maxn; j++) {
12.             prime[prime[j] * i] = 1;
13.             if (i % prime[j] == 0) break;
14.         }
15.     }
16. }
17. int main() {
18.     getPrime(); // 先进行素数筛选预处理
19.     int a, b;
20.     while (scanf("%d%d", &a, &b) != EOF) {
21.         if (a > b) swap(a, b); // a 有可能比 b 大
22.         int ans = 0;
23.         for (int i = 1; i <= prime[0]; i++) {
24.             if (prime[i] >= a) ans++; // 素数大于 a 答案加一
25.             if (prime[i] > b) {
26.                 ans--; // 大于 b 要减回来
27.                 break;
            }
```

```
28.         }  
29.         }  
30.         printf("%d\n", ans);  
31.     }  
32.     return 0;  
33. }
```

### 练习题目

DreamJudge 1375 素数

N 诺  
noobdream.com

### 3.7 分解素因数

我们知道，对于任意一个大于 1 的整数，它都可以分解成多个质因子相乘的形式。

#### 质因数个数

##### 题目描述：

求正整数  $N(N>1)$  的质因数的个数。相同的质因数需要重复计算。如  $120=2*2*2*3*5$ ，共有 5 个质因数。

##### 输入描述：

可能有多组测试数据，每组测试数据的输入是一个正整数  $N$ ， $(1<N<10^9)$ 。

##### 输出描述：

对于每组数据，输出  $N$  的质因数的个数。

##### 输入样例#：

120

##### 输出样例#：

5

##### 题目来源：

DreamJudge 1156

**题目解析：**我们可以通过上一节学会的素数筛选，先将所有的素数筛选出来。然后再不断的去分解素数，直到将数分解到 1 为止。由于我们的素数筛选只能到 1000000，对于更大的素因子我们可以不继续分解，因为不会存在两个大于 1000000 的素因子，如果存在，那么这个数的范围一定大于题目所给的范围  $10^9$ 。

## 参考代码

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. // 线性素数筛选 prime[0]存的是素数的个数
5. const int maxn = 1000000 + 5;
6. int prime[maxn];
7. void getPrime() {
8.     memset(prime, 0, sizeof(prime));
9.     for (int i = 2; i <= maxn; i++) {
10.         if (!prime[i]) prime[++prime[0]] = i;
11.         for (int j = 1; j <= prime[0] && prime[j] * i <= maxn; j++) {
12.             prime[prime[j] * i] = 1;
13.             if (i % prime[j] == 0) break;
14.         }
15.     }
16. }
17. int main() {
18.     getPrime(); // 先进行素数筛选预处理
19.     int n;
20.     while (scanf("%d", &n) != EOF) {
21.         int ans = 0;
22.         for (int i = 1; i <= prime[0]; i++) {
23.             while (n % prime[i] == 0) {
24.                 n /= prime[i];
25.                 ans++;
26.             }
27.         }
28.         if (n > 1) ans++;
29.         printf("%d\n", ans);
30.     }
31.     return 0;
32. }
```

## 练习题目

DreamJudge 1284 整除问题



## 3.8 二分快速幂

有一类题目是这样的

求  $(x^y) \% p$

当  $y$  很大的时候, 我们不能直接用 for 去一个一个的乘, 因为这样的方法复杂度是  $O(N)$  的。那么有没有更高效的方法呢?

遇到这类问题的时候, 我们就可以使用二分快速幂的方法来求解。

举个例子

$$3^7 = 3^4 * 3^2 * 3^1$$

首先我们要知道, 对于任意一个数  $s$ , 它的二进制代表了它可以由 2 的次幂的累加和来表示。

比如 7 的二进制是 111

那么它就是说  $7 = 2^2 + 2^1 + 2^0$

再比如一个数 12 的二进制是 1101

$$13 = 2^3 + 2^2 + 2^0$$

所以对于任意一个  $x^y$ , 我们都可以将  $y$  分解成 2 的幂次的形式。

$$\text{例如 } 5^{13} = 5^8 * 5^4 * 5^1$$

这样做有什么好处呢?

1..2..4..8..16..32..64..128.....1024.....

你发现其中的奥秘了吗?

1、每一个数都是它前一个数的 2 倍

2、它的迭代速度超级快

### 幂次方

**题目描述:**

对任意正整数  $N$ , 求  $X^{N\%233333}$  的值。

要求运算的时间复杂度为  $O(\log N)$ 。

例如

$$X^{30} = X^{15} * X^{15}$$

$$X^{15} = X^7 * X^7 * X$$

$$X^7 = X^3 * X^3 * X$$

$$X^3 = X * X * X$$

共 7 次乘法运算完毕。

#### 输入描述:

输入两个整数 X 和 N, 用空格隔开, 其中  $X, N \leq 10^9$ 。

#### 输出描述:

输出  $X^N$  对 233333 取模的结果。

#### 输入样例#:

2 5

#### 输出样例#:

32

#### 题目来源:

DreamJudge 1017

题目解析: 运用上面讲过的二分快速幂思想和同余模定理即可。

#### 参考代码

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. typedef long long ll;
5. ll mod_pow(ll x, ll y, ll mod) {
6.     ll res = 1;
7.     while (y > 0) {
8.         //如果二进制最低位为 1、则乘上  $x^{(2^i)}$ 
9.         if (y & 1) res = res * x % mod;
10.        x = x * x % mod; // 将 x 平方
11.        y >>= 1;
12.    }
13.    return res;
14. }
15.
16. int main() {
17.    ll x, n; //注意  $x*x$  会超出 int 范围
18.    scanf("%lld%lld", &x, &n);
19.    printf("%lld\n", mod_pow(x, n, 233333));
20.    return 0;
21. }
```

### 3.9 常见数学公式总结

#### 错排公式

问题：十本不同的书放在书架上。现重新摆放，使每本书都不在原来放的位置。有几种摆法？

递推公式为： $D(n) = (n - 1) * [D(n - 1) + D(n - 2)]$

#### 海伦公式

$$S = \sqrt{p * (p - a) * (p - b) * (p - c)}$$

公式描述：公式中  $a, b, c$  分别为三角形三边长， $p$  为半周长， $S$  为三角形的面积。

#### 组合数公式

$$C(n, m) = p(n, m)/m! = n!/((n - m)! * m!)$$

公式描述：

组合数公式是指从  $n$  个不同元素中，任取  $m(m \leq n)$  个元素并成一组，叫做从  $n$  个不同元素中取出  $m$  个元素的一个组合。

#### 两点之间的距离公式

$$|AB| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

公式描述：公式中  $(x_1, y_1), (x_2, y_2)$  分别为  $A, B$  两个点的坐标。

扇形面积： $S = 1/2 \times \text{弧长} \times \text{半径}$ ， $S_{\text{扇}} = (n/360) \pi R^2$

## 卡特兰数

原理:

令  $h(0)=1, h(1)=1$ , catalan 数满足递归式:

$$h(n) = h(0) * h(n-1) + h(1) * h(n-2) + \dots + h(n-1)h(0) \quad (\text{其中 } n \geq 2)$$

另类递推公式:

$$h(n) = h(n-1) * (4 * n - 2) / (n + 1)$$

该递推关系的解为:

$$h(n) = C(2n, n) / (n + 1) \quad (n=1, 2, 3, \dots)$$

卡特兰数的应用实质上都是递归等式的应用

前几项为: 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452, ...

所有的奇卡特兰数  $C_n$  都满足  $n = 2^k - 1$ 、所有其他的卡特兰数都是偶数。

## 应用

- 1、12 个高矮不同的人, 排成两排, 每排必须是从矮到高排列, 而且第二排比对应的第一排的人高, 问排列方式有多少种?
- 2、括号化问题。矩阵链乘:  $P=A_1 \times A_2 \times A_3 \times \dots \times A_n$ , 依据乘法结合律, 不改变其顺序, 只用括号表示成对的乘积, 试问有几种括号化的方案?
- 3、将多边形划分为三角形问题。将一个凸多边形区域分成三角形区域(划分线不交叉)的方法数?
- 4、出栈次序问题。一个栈(无穷大)的进栈序列为 1、2、3、...、n, 有多少个不同的出栈序列?
- 5、通过连结顶点而将  $n + 2$  边的凸多边形分成三角形的方法个数。
- 6、所有在  $n \times n$  格点中不越过对角线的单调路径的个数。
- 7、有  $n+1$  个叶子的二叉树的个数。

更多的变化不胜枚举, 遇到这类规律题建议使用下一节讲的 OEIS 来解决问题。

### 3.10 规律神器 OEIS

下面给大家隆重介绍一个神器，所有的考生，看这里：OEIS。

对，就是它，**考试必备神器**，你值得拥有。有了它，你可以变得更自信，你可以满怀信心的走进考场，你可以飞快解出那些所谓推公式的规律难题。

它的网址：<http://oeis.org/>

它究竟有多变态，只要你输入前几项，它就可以给你找出规律来，并且自动给你推出公式。所以对于找规律的题目，你只需要手动计算出前几项的值，或者暴力打表求出前几项的值，然后输入 OEIS，他就可以告诉你公式是什么。

怎么样，超级有用吧！妈妈再也不用担心我的数学了^\_^

What!你说它考试的时候没用？难道如何优雅的上厕所也需要教？

所有的考生，一定要学会，用它！用它！一定要用它！

只用花一分钟你就学会用这个东西了，如果你不会用，而你的竞争对手会用，那你就惨了，说不定你推到死都推不出来的公式，别人几秒钟就搞定了。反之，你学会了用这个东西，而你的竞争对手不会，到时候你会感觉爽翻天(\*^▽^\*)

快来用这个神器做一下下面这个题练练手吧

#### 01 字符串

##### 题目描述：

给你一串长度为  $n$  的全为 0 的字符串，你可以进行一个压缩操作，将两个相邻的 0 压缩成一个 1。请问最多会有多少种组合出现？

例如  $n$  为 3 则有下面 3 种组合：

000

10

01

### 输入描述:

输入一个正整数  $n$  ( $1 \leq n \leq 10000$ )。

### 输出描述:

输出最多有多少种组合出现, 由于结果可能过大, 请将答案对 2333333 取模。

### 输入样例#:

3

### 输出样例#:

3

### 题目来源:

DreamJudge 1479

**题目解析:** 对于这类让我们去找有多少种情况的题目, 第一反应就是他是有规律的, 它的规律可能简单, 也可能很难。这时候我们先推出它的前几项值,  $n$  为 1 时答案等于 1,  $n$  为 2 时答案等于 2,  $n$  为 3 时答案等于 3,  $n$  为 4 时答案等于 5。然后将这前几项值 1, 2, 3, 5 输入到 OEIS 中, 它立马告诉我们这个规律是斐波那契数列,  $f[i] = f[i-1] + f[i-2]$ 。

### 参考代码

```
1. #include <stdio.h>
2.
3. long long f[10005] = {0};
4. int main(){
5.     int n;
6.     scanf("%d", &n);
7.     f[0] = 1; f[1] = 1;
8.     for (int i = 2; i <= n; i++) {
9.         f[i] = f[i-1] + f[i-2];
10.        f[i] %= 2333333;
11.    }
12.    printf("%lld\n", f[n]);
13.    return 0;
14. }
```

上面这个题目规律很简单, 可能你不借助 OEIS 都能推出来, 但是很多时候规律的公式很复杂, 涉及到组合数之类的, 手推是非常难推出来的。既然能几秒钟解决的问题, 为什么要花大量时间去手推公式呢? 多留点时间做其他题不好吗?

## 第四章 高精度问题

本章我们重点讲解一些常见的高精度题型，包括 python 解法、java 解法、C/C++解法等内容。希望能帮助读者更好的掌握计算机考研机试中所涉及到的高精度问题。



## 4.1 Python 解法

人生苦短，我用 python！

如果你的考试院校支持用 python，那真是太好不过了。Python 真是超级棒的一门语言，虽然 python 速度慢，但是那及其丰富的库和框架，再加上简洁的语法，真是美妙至极。

对于高精度的大数类问题，对 python 来说简直来说就是小菜一碟。

什么？你在问我 python 大数应该怎么用？

Are you kidding me? 在 python 眼中就没有大数这个东西。

我们还是假装给一下两个大数相加的代码。

```
1. while True:
2.     try:
3.         a, b = map(int, input().split())
4.         c = a+b
5.         print(c)
6.     except:
7.         break
```

所以只要你会用 python 就可以了。

### 练习题目

DreamJudge 1474 大整数加法

DreamJudge 1475 大整数乘法



## 4.2 Java 解法

基本上所有的 OJ 都支持 Java，所以建议大家使用 Java 来解决高精度的题目。

BigDecimal（表示浮点数）和 BigInteger（表示整数）加上

```
import java.math.*
```

```
1.  valueOf(paramant); //将参数转换为指定类型
2.  add(); //大数加法
3.  subtract(); //减法
4.  multiply(); //乘法
5.  divided(); //相除取整
6.  remainder(); //取余
7.  pow(); //a.pow(b) = a ^ b
8.  gcd(); //最大公约数
9.  abs(); //绝对值
10. negate(); //取反数
11. mod(); //a.mod(b) = a % b = a.remainder(b)
12. max(); min();
13. public int compareTo(); //比较
14. boolean equals(); //比较是否相等
```

参考代码

```
1.  import java.math.BigInteger;
2.  import java.util.Scanner;
3.
4.  public class Main {
5.
6.      public static void main(String[] args) {
7.          Scanner sr=new Scanner(System.in);
8.          while (sr.hasNext()) {
9.              BigInteger a,b;
10.             a=sr.nextBigInteger();
11.             b=sr.nextBigInteger();
12.             System.out.println(a.add(b));
13.         }
14.     }
15. }
```

## 4.3 C/C++解法

C/C++可以通过模拟的方法解决高精度的问题，但是我们不是特别建议在考试的时候自己手动去模拟大整数的问题，这样很容易出现失误。

当然，如果是很简单的加减法运算，用 C/C++模拟也是挺不错的，毕竟更换 IDE 也挺麻烦的。

下面给出 C/C++大数加法的代码

```
1. #include <iostream>
2. #include <string>
3. #include <algorithm>
4. using namespace std;
5.
6. string Add(string a, string b) {
7.     //a 一直为位数较长的字符串
8.     if (a.length() < b.length()) a.swap(b);
9.
10.    string result(a.length(), 0); //初步设置 result 长度为较长字符串长度
11.    b.insert(0, a.length() - b.length(), '0'); //较短的字符串前面补零方便计算
12.    int carry = 0; //进位
13.    for (int i = a.length() - 1; i >= 0; i--) {
14.        int sum = (a[i] - 48) + (b[i] - 48) + carry;
15.        carry = sum / 10;
16.        result[i] = sum % 10 + 48;
17.    }
18.    //若进位不为 0，还要在前面补上进位
19.    if (carry != 0) {
20.        result.insert(result.begin(), carry + 48);
21.    }
22.    return result;
23. }
24.
25. int main() {
26.    string a, b;
27.    while (cin >> a >> b)
28.        cout << Add(a, b) << endl;
29.    return 0;
30. }
```

## 第五章 数据结构

本章我们重点讲解一些常见的数据结构题型，包括栈的应用、哈夫曼树、二叉树、二叉排序树、hash 算法、前缀树等内容。希望能帮助读者更好的掌握计算机考研机试中所涉及到的数据结构问题。



## 5.1 栈的应用

栈是一种只能在一端进行插入和删除操作的数据结构，它满足先进后出的特性。

我们通过 `stack<int> S` 来定义一个全局栈来储存整数的空的 `stack`。当然 `stack` 可以存任何类型的数据，比如 `stack<string> S` 等等。

```
1. #include <iostream>
2. #include <stack>
3. using namespace std;
4. stack<int> S;
5. int main() {
6.     S.push(1);
7.     S.push(10);
8.     S.push(7);
9.     while (!S.empty()) {
10.        cout << S.top() << endl;
11.        S.pop();
12.    }
13.    return 0;
14. }
```

### 括号的匹配

#### 题目描述:

假设表达式中允许包含两种括号:圆括号和方括号。编写一个算法判断表达式中的括号是否正确配对。

#### 输入描述:

由括号构成的字符串, 包含“(“、“)”、“[“和”]“。

#### 输出描述:

如果匹配输出 YES, 否则输出 NO。

#### 输入样例#:

[([[]() )]

#### 输出样例#:

YES

题目来源:

DreamJudge 1501

**题目解析:** 用栈模拟即可, 和栈顶元素匹配就说明配对成功, 将栈顶元素出栈, 否则配对不成功, 就将当前元素入栈。最后查看栈是否为空, 若为空则是 YES, 否则就是 NO。

参考代码

```
1. #include <bits/stdc++.h>
2. using namespace std;
3.
4. int main() {
5.     char s[300];
6.     scanf("%s", &s);
7.     int len = strlen(s);
8.     stack<char> st;
9.     for (int i = 0; i < len; i++) {
10.        if (!st.empty()) {
11.            char now = st.top();
12.            if (s[i]=='(' && now=='(' || s[i]==']' && now=='[')
13.                st.pop();
14.            else st.push(s[i]);
15.        }
16.        else st.push(s[i]);
17.    }
18.    if (!st.empty()) printf("NO\n");
19.    else printf("YES\n");
20.    return 0;
21. }
```

其他比较常见的应用方式

- 1、计算表达式的值
- 2、带优先级的括号匹配问题

练习题目

DreamJudge 1067 括号的匹配

DreamJudge 1296 括号匹配问题

## 5.2 哈夫曼树

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 5.3 二叉树

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 5.4 二叉排序树

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日





## 5.5 hash 算法

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 5.6 前缀树

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 第六章 搜索

本章我们重点讲解一些常见的搜索题型，包括暴力枚举、广度优先搜索（BFS）、递归及其应用、深度优先搜索（DFS）、搜索剪枝技巧、终极骗分技巧等内容。希望能帮助读者更好的掌握计算机考研机试中所涉及到的搜索问题。



## 6.1 暴力枚举

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 6.2 广度优先搜索 (BFS)

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 6.3 递归及其应用

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 6.4 深度优先搜索（DFS）

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 6.5 搜索剪枝技巧

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日





## 6.6 终极骗分技巧

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 第七章 图论

本章我们重点讲解一些常见的图论题型，包括暴力枚举、广度优先搜索（BFS）、递归及其应用、深度优先搜索（DFS）、搜索剪枝技巧、终极骗分技巧等内容。希望能帮助读者更好的掌握计算机考研机试中所涉及到的图论问题。



## 7.1 理论基础

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 7.2 图的存储

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 7.3 并查集

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 7.4 最小生成树问题

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 7.5 最短路径问题

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 7.6 拓扑排序

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日





## 7.7 打印路径类问题

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺**兑换中心**进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 第八章 动态规划

本章我们重点讲解一些常见的动态规划题型，包括递推求解、最大子段和、最长上升子序列（LIS）、最长公共子序列（LCS）、背包类问题、记忆化搜索、字符串相关的动态规划等内容。希望能帮助读者更好的掌握计算机考研机试中所涉及到的动态规划问题。



## 8.1 递推求解

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 8.2 最大子段和

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



### 8.3 最长上升子序列 (LIS)

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 8.4 最长公共子序列 (LCS)

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 8.5 背包类问题

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 8.6 记忆化搜索

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日





## 8.7 字符串相关的动态规划

N 诺考研系列相关书籍都将采用电子版**不断更新**的形式发布。

最新版的书籍可以在 N 诺兑换中心进行兑换或购买，**一次兑换，终生免费。**

当前版本更新日期：2019 年 12 月 01 日



## 完结撒花

当你看到这里, 说明你已经读完了本书所有的内容。恭喜你, 学完计算机考研机试攻略 - 高分篇的全部内容, 我们相信你一定会在机试中取得非常不错的成绩。

如果本书对你有所帮助, 希望你能在复试之后将还能记得的机试题目发表在 N 诺的交流区里, 当然也可以直接在 N 诺官方群或机试群里联系管理员。N 诺会根据你提供题目描述进行数据还原, 继续帮助下一届学弟学妹, 让他们可以做到最新的真题, 少走一些弯路, 并且我们会将你的名字或 N 诺 ID 放在题目的后面进行特别鸣谢。

最后, 我们不仅希望你能在机试中取得满分的成绩, 也希望你能如愿以偿的考上心目中理想的院校, **加油! Go!Go!Go!**

一定要做的说明: N 诺出版的考研系列书籍都将以**电子版**的形式进行发布更新, 需要纸质版的同学自行打印学习即可。

## N 诺考研系列书籍为什么只发布电子版不发布纸质版？

**原因一：**发布纸质版需要提前很久将书籍整理成册，时间太赶容易敷衍了事，我们相信慢工出细活。

**原因二：**纸质版一经印刷，便无法修改，就算发现问题或者想对某些内容进行优化也没办法。而电子版可以随时勘误进行修改，灵光一现的时候还能对前面写的不够好的地方进行优化。

**原因三：**电子版少了中间商，可以给同学们节约更多的费用。纸质版的话出版社、印刷商都要从中获取利润，最终羊毛出在羊身上。



## 如何获取 N 诺考研系列书籍？

noobdream.com

访问 N 诺平台（[www.noobdream.com](http://www.noobdream.com)）的兑换中心即可兑换或购买各种你想要的书籍或资料。

另外，本书会不断的进行更新，所以需要最新版的同学，请去官网**兑换中心**进行兑换，只要一次兑换，后续版本更新都可看到，不用重复兑换。

## 考研机试冲刺八套卷

在本书的最后告诉所有同学一个好消息, 由 N 诺出版的考研机试冲刺八套卷将**免费赠送**给所有考研的同学。

虽然本书已经将各种考点和考法都在书中做了介绍, 但是想到很多同学都没有实际机试的经验, 对本书的介绍的各类做题技巧可能没有适应, 所以需要几场模拟考试来将各种技巧**融会贯通**, 在正式的上机考试中才能游刃有余。

N 诺邀请了众多大佬共同出这八套卷的题, 押中原题的话不要惊讶, 这只是常规操作。毕竟这其中曾经把机试题全部押中的大佬, **押题**, 绝对是专业的。当然, 这八套冲刺卷目的不仅仅是为了押题, 更是为了帮助同学们**快速变强**, 题目难度适中(准确说应该是较为简单), 适合所有学校的同学。

最最最重要的是, 在每一场比赛结束之后的 10 分钟以内, 我们会马上在 B 站上直播讲解这套冲刺卷的每一道题的题解(**全程讲解答疑免费**), 以及可以使用什么样的技巧**快速通过**这类题, 毕竟有很多实用的技巧在书上不太好讲述, 实战过程中大家也更容易理解。

**直播题解传送门:** <https://live.bilibili.com/3513962> (可以提前关注哦)

还要一点很重要, 同学们看了直播题解懂了怎么做之后, 一定要去**补题**, 不补题的话, 你绝对会后悔的。对于绝大部分初学者来说, 口头 AC 和实际 AC 中间差了十万八千里。

**温馨提示：**这八套冲刺卷是非常重要的，就算你是一个什么都不会的小白，只要认真做了这八套卷，赛后把题都补了，绝对会拿到机试高分。如果你认真做了却没能拿到高分，我们一起祝N诺明年倒闭。

21 考研的同学也可以来做这八套卷，今年提前做了，明年就轻松了。

### 如何报名？

比赛网址：<http://www.noobdream.com/DreamJudge/Contest/match/>

点击报名即可参加，没有提前报名的话，比赛过程中也可报名。

比赛时长：3 个小时

题目数量：6 个题

### 时间安排

考研机试冲刺八套卷（第一套）：12 月 28 日 晚上 6 点 — 9 点

考研机试冲刺八套卷（第二套）：1 月 4 日 晚上 6 点 — 9 点

考研机试冲刺八套卷（第三套）：1 月 11 日 晚上 6 点 — 9 点

考研机试冲刺八套卷（第四套）：1 月 18 日 晚上 6 点 — 9 点

考研机试冲刺八套卷（第五套）：1 月 25 日 晚上 6 点 — 9 点

考研机试冲刺八套卷（第六套）：2 月 1 日 晚上 6 点 — 9 点

考研机试冲刺八套卷（第七套）：2 月 8 日 晚上 6 点 — 9 点

考研机试冲刺八套卷（第八套）：2 月 15 日 晚上 6 点 — 9 点

即每周六晚 6 点开始，9 点结束，9 点 10 分开始直播讲解每道题的解法。

最后，每一场比赛开始前，我们会在N诺官方群(612180120)和N诺考研机试交流群(960036920)提醒各位同学参加比赛，赛后也会将题解和标程上传到群文件，大家有不会的问题也可以在群里进行交流探讨。

## N 诺网校

在本书的最后给大家安利一下 N 诺网校，N 诺网校的**性价比**可以说是无法用言语形容的。

- 1、N 诺网校没有营销环节，可以为大家节约 **20-30%**的费用。
- 2、N 诺网校没有广告支出，可以为大家节约 **30-50%**的费用。
- 3、N 诺网校不抽取上课老师的分成，可以为大家节约 **50-60%**的费用。
- 4、N 诺网校有渠道给大家找到专业大佬来上课，水平有保证。
- 5、由于网校这一块是亏本运营，所以没有固定开班时间，可以**预报名**，报名人数达到最低开课人数即开始上课，**组团报名**不仅有优惠，而且可以更快开课。



该截图展示了 N 诺网校的课程列表。页面顶部有导航栏，包含“首页”、“DreamJudge”、“历年真题”、“任务平台”、“讨论区”、“考研头条”和“兑换中心”。右侧有“登录”和“注册”按钮。课程列表标题为“N诺网校课程”，下方列出了五门课程：

- 零基础入门课程**：学完本次课程，可以基本掌握C/C++语言技能，能完成大部分基础编程题目和一些简单的算法题目。长期课程，编程入门。¥1，开始时间 2019年8月30日 00:52，报名咨询人数: 245。
- 算法竞赛基础课程**：学完本次课程，可以基本掌握各类算法竞赛基础算法知识，可以继续进阶算法竞赛提高课程的学习。长期课程，算法基础。¥1，开始时间 2019年8月30日 01:30，报名咨询人数: 157。
- 算法竞赛提高课程**：学完本次课程，可以掌握一些进阶算法、数据结构和数学知识，足以应对各类算法竞赛，并获得不错的成绩。长期课程，算法提高。¥1，开始时间 2019年8月30日 01:31，报名咨询人数: 84。
- 考研机试零基础班**：学完本次课程，你可以学会写大部分简单的代码。7天限时，零基础。¥1，开始时间 2019年11月21日 11:03，报名咨询人数: 135。
- 考研机试提高班**：学完本次课程，你可以学会写大部分简单的算法。7天限时，提高。¥1，开始时间 2019年11月21日 11:05，报名咨询人数: 145。

## N 诺网校的课程分为两种模式

- 1、小班课程
- 2、一对一个性化辅导

小班课程除了上课老师以外，还会有 2-3 名助教进行日常答疑，价格也非常实惠，一门课 30 课时左右，一门课仅需 150-300 元，并且附赠培训辅导书以及教学视频。

一对一个性化辅导适合经济宽裕点的同学，虽然没有动辄上万的费用，但是 100 元/小时差不多是最低价格了，低于这个价格几乎不可能有大佬愿意接单。

**最关键的是：**N 诺网校可以上一次课交一次课的费用，这是别的地方不敢想象的。

## N 诺网校的科目

- 1、C 语言
- 2、数据结构
- 3、操作系统
- 4、计算机组成原理
- 5、计算机网络
- 6、数据库
- 7、算法分析
- 8、考研机试

## 如何报名？

在 N 诺官方群联系管理员或者在 N 诺网校的对应科目中点击报名意向咨询即可。