A Hide and Seek

时间限制：1000ms　内存限制：65536kb

通过率：904/1154 (78.34%)　正确率：904/4084 (22.14%)

题目描述

给定一个长度为 n 的**单调非下降有序数组**（可能有重复元素），以及 k 个询问，每个询问包含一个整数 m。
对于每个询问，输出数组中整数 m 出现的最大下标(**下标从 0 开始**)，若数组中没有 m，则输出 −1。

输入

第一行为两个正整数 n,k(n,k≤10^5)，n 为数组长度，k 为询问个数。
第二行为 n 个整数 a_i，为数组的 n 个元素，保证在 `int` 范围内，且这 n 个数是**单调非下降的**。
第 3~k+2 行每行一个整数 m_i，保证在 `int` 范围内，依次为 k 个询问对应的整数 m。

输出

对于每个询问，输出一行一个整数 index，为该询问的答案。

输入样例

```
10 3
1 2 3 3 4 5 6 8 8 8
5
3
8
```

输出样例

```
5
3
```

```
9
```

HINT

使用线性查找可能会超时

考察知识点

数组查找
考察难度 4

解题思路

注意到数据范围是 n,k<=100000，如果使用线性查找的话，最坏情况下时间复杂度将达到 O(nk)，将会超时，因此我们选取二分查找，每次查找的时间复杂度为 O(logn)，这样就不会超时了。
于是对每个询问 m，使用二分查找来寻找 m 出现的一个下标，并按这个下标开始往后推，寻找到 m 的最大下标，之后输出即可。

代码

```
#include<stdio.h>
int i,n,k,m,a[100005];
int getIndex(int m){
    int index = -1,l = 0,r = n - 1,mid;
    while (l <= r){
        mid = (l + r) / 2;
        if (a[mid] == m){
            index = mid;
            break;
        }
        else if (a[mid] < m)
            l = mid + 1;
        else
            r = mid - 1;
    }
    if (index != -1)
```

```
        while (index < n - 1 && a[index + 1] == a[index])
            index++;
    return index;
}
int main(){
    scanf("%d%d",&n,&k);
    for (i = 0;i < n;i++)
        scanf("%d",&a[i]);
    for (i = 0;i < k;i++){
        scanf("%d",&m);
        printf("%d\n",getIndex(m));
    }
    return 0;
}
```

B Terry 的括号匹配

时间限制：1000ms　内存限制：65536kb

通过率：791/1002 (78.94%)　正确率：791/2772 (28.54%)

题目描述

Terry 现在有一个括号序列，问它们是否可以完全匹配。

输入

一行字符串，长度小于 100，只包括 ()[]<> 中的字符。

输出

输出一行，如果可以完全匹配，输出 Yes，失配则输出 No。

输入样例

```
<>>
```

输出样例

```
No
```

考察知识点

栈的使用
难度系数 2

解题思路

本题是书上 P119 的例 6-14 括号匹配改了一点点，只需要将书上的代码修改一点点加上输入输出，调用此函数就可以 AC。具体改动见标程。

标程

```
#include <stdio.h>

int stack[500]={0},s_top=0;

void push(int v) {
    stack[s_top++]=v;
}
int pop() {
    return stack[--s_top];
}
int stack_empty() {
    return s_top == 0;
}

int paran_match(char* s) {
    char *p;
    int d;
    for(p=s;*p!='\0';p++) {
        if(*p=='('||*p=='['||*p=='<')
            push(*p);
```

```
        else if(*p==')'||*p==']'||*p=='>') {
            if(stack_empty())
                return 0;
            d=pop();

    if((*p==')')&&(d=='(')||(*p==']')&&(d=='[')||(*p=='>')&&(d=='<
'))
                continue;
            return 0;
        }
    }
    return stack_empty()?1:0;
}

int main() {
    char a[200];
    scanf("%s",a);
    if(paran_match(a))
        printf("Yes\n");
    else
        printf("No\n");
}
```

C N 位超级质数改

时间限制：1000ms　内存限制：65536kb

通过率：580/692 (83.82%)　正确率：580/1301 (44.58%)

题目描述

从标准输入读入一个整数 n(n≤6)，生成所有满足下列条件的 n 位超级质数：从最右侧开始的任意连续位均是质数。
例如，13,113 就分别是这样的 2,3 位超级质数。**注意题目，是从右侧开始的。**
本题不允许前导零存在，例如 103， 103 和 33 是质数，但 03 的数字形式不合法，所以 103 不算 3 位的超级质数。

输入

一个正整数 n(n≤6)

输出

输出所有满足条件的 n 位超级质数，每行一个。
按照从右到左来比较数的大小输出，例如 913,523 即先比较 3 和 3，再比较 1 和 2，这样 913 排在 523 前面。
(P.S:输出顺序按照课上方法的质数表正常构造即满足条件)

(P.S:因为是往前面填数，填数的时候范围不同了)

输入样例

```
2
```

输出样例

```
13
23
43
53
73
83
17
37
47
67
97
```

考察知识点

队列
难度系数 3

思路解析:

本题思路和 N 位超级质数基本一致.首先,我们找出长度为一位的 4 个质数 2,3,5,7.把这 4 个质数放在队列之中,因为这次是在质数的最高位添加新数,所以 1-9 这 9 个数字都是可选的.首先,取出队列开头的数字,记为 m,枚举 1-9,乘上此时 10 的 n-1 次方.(n 为正在构造的质数的长度),这个乘积记为 digits[i]*tmp[len],m+digits[i]*tmp[len]就得到了一个新的可能的答案,判断这个答案是否是质数,如果是,则加入队列之中.这样每次取出队列开头的质数,构造更多的质数加入队列尾巴.

举 n=2 时候的例子,首先取出队列首元素 2,构造出 12,22.....,92 等,都不是质数,队首向后移动,取出现在队首元素 3,构造 13,23,33.....等,然后其中 13,23...是质数,把它加进队列的最末尾.以此类推

题目代码:

```
#include<stdio.h>
#include<math.h>
#define NumOf(a) (sizeof(a)/sizeof(a[0]))
#define MAX_Q 10000
int hd = 0, tail=4, queue[MAX_Q]={2, 3, 5, 7};//queue 中初始只包含一个质数
int digits[]={1,2,3,4,5,6,7,8,9}; //可用于前一位的数字表
int tmp[20];
int isPrime(int x){
    int t = sqrt(x)+0.5,i;
    for(i = 2;i <= t;i++)
        if(x % i == 0)
            return 0;
    return 1;
}
int main()
{
    int i, m, n, len, end=0; //end 用于保存长度为 len 的最后一个种子的位置
    tmp[0] = 1;
    for(i = 1;i < 10;i++)
        tmp[i] = tmp[i-1]*10;
    scanf("%d",&n);
    for(len = 1; len < n; len++){
        for(end = tail; hd<end; hd++){
        m = queue[hd];
        for(i=0; i<NumOf(digits); i++)
        if(isPrime(m+digits[i]*tmp[len]))
            queue[tail++]=m+digits[i]*tmp[len];
        //生成并存储比种子长一位的质数
        }
    }
```

```
    for(i = end;i < tail;i++)
           printf("%d\n",queue[i]);
}
```

D 首个出现三次的字母

时间限制: 1000 ms 内存限制: 65536 kb
总通过人数: 904 总提交人数: 996

题目描述

读入一个字符串，查找字符串中首个出现 3 次的小写字母（即第一个满足出现 3 次的字母）。输出该字母及其在字符串中第一次，第二次和第三次出现的位置。字符位置从 1 开始。

输入

一个字符串，保证只有小写字母，保证一定有出现三次的字母。长度小于 100。

输出

输出该字母及其在字符串中第一次，第二次和第三次出现的位置，三次出现的位置及其与字母间以空格分隔。

输入样例

```
helloworld
```

输出样例

```
l 3 4 9
```

考察知识点：

字符串, hash 表

解题思路：

当首次遇到出现三次的字母的时候就可以输出了，因此只需记录前两次出现的位置。可以使用两个一维数组构造一个散列表，以字母在字母表中的顺序作为键值，同时还需要一个数组来记录当前字母出现了几次。

代码：

```
#include<stdio.h>
#define MAX_N (30)
int char_pos1[MAX_N], char_pos2[MAX_N];
int char_cnt[MAX_N];
int main() {
    int i, c;
    for (i = 1; (c = getchar()) != EOF; i++) {
        int pos = c - 'a';
        if (char_cnt[pos] == 2) {
            printf("%c %d %d %d\n", c, char_pos1[pos], char_pos2[pos],
i);
            return 0;
        }
        if (char_cnt[pos]++ == 0) {
            char_pos1[pos] = i;
        } else {
            char_pos2[pos] = i;
        }
    }
    return 0;
}
```

E 字符统计.改

时间限制: 1000 ms 内存限制: 65536 kb
总通过人数: 593 总提交人数: 738

题目描述

从标准输入读入一句话，统计这句话中：每个小写字母出现次数；单词总数；每个大写字母出现次数与字符总数（空格与标点不计入其内）。每个单词用空格、句号或者逗号隔开，两个单词之间可能不止一个分隔符号。输入输出均使用的英文符号。

输入

输入一行，为所需要统计的语句。

输入长度不超过 200200 。
输入只由大小写字母、空格、逗号、英文句号组成。

输出

对于每一句话，先输出每个小写字母出现次数，再输出大写字母出现次数，两者均按字母表顺序输出，再输出单词总数与字符总数。

每一个数据都占一行，具体请看样例。

没有出现的字符不输出。

输入样例

```
I. love C, Programming.
```

输出样例

```
a:1
e:1
g:2
i:1
l:1
m:2
n:1
o:2
r:2
v:1
C:1
I:1
P:1
4
17
```

考察知识点

> 字符串，hash 表
> 难度系数 4

解题思路

首先，从输入中一个一个字符的读入，如果为大小写字母，那么将字母在字母表中的顺序作为数组下标，数组内容为字母出现次数来进行统计。对于单词的统计，需要在每次读入一个非字母时，判断前一个字符是否为字母，如果是，则可以认定这是两个单词间的分隔符，即单词数量+1。标程里借助了一个 flag 变量作为指示性变量，当读入一个字母时，将其的值设为 1，读入一个非字母时，设为 0。因此，每次只需判断 flag 的值即可完成单词的统计。

标程

```
#include <stdio.h>
#include<ctype.h>
#define N 26
int main(){
    int i,c,total=0,lower[N]={0},upper[N]={0},flag=0,words=0;
    while((c=getchar())!=EOF){
        if(islower(c)){
            lower[c-'a']++;
            total++;
            flag=1; //读入字母时设为 1
        }
        else if(isupper(c)){
            upper[c-'A']++; //借助 ASCII 码来判断是第几个字母
            total++;
            flag=1;
        }
        else{   //读入的字符并非字母时
            if(flag){  //如果 flag 为 1，即上一个字符为字母
                words++;  //意味着出现了一个单词
                flag=0;//将 flag 的值再次设为 0
            }
        }
    }
```

```
    for(i=0;i<N;i++){
        if(lower[i]!=0)
            printf("%c:%d\n",i+'a',lower[i]);
    }
    for(i=0;i<N;i++){
        if(upper[i]!=0)
            printf("%c:%d\n",i+'A',upper[i]);
    }
    printf("%d\n%d",words,total);
}
```

F PPZ 的单身狗区间搜寻

时间限制：1000ms　内存限制：65536kb

通过率：333/454 (73.35%)　正确率：333/1147 (29.03%)

题目描述

PPZ 的室友由于脱单无望，于是开始疯狂的搜刮他的“单身队友”，这个人甚至连区间也不放过。。。现在有 n 个闭区间，需要你帮他找出其中的单身狗区间，即与其他区间不相交的区间，区间端点重合也算相交。

输入

输入共 n 行 (0<n≤100)
n 行输入中每行两个整数(int 范围内)，表示区间的左右端点，两个整数之间以空格隔开
区间端点 aa 满足(0≤a≤100000)，保证右端点严格大于左端点。

输出

按照输入顺序，输出所有的单身狗区间，每个区间占一行，两个端点整数之间以空格分隔。

保证至少存在一个单身狗区间。

输入样例

```
1 4
11 13
```

```
6 7
8 9
10 15
1 5
```

输出样例

```
6 7
8 9
```

HINT

```
散列表
```

考察知识点

```
数组
难度系数 4
```

解题思路

类似课件中区间合并这道题的思路，针对这道题做一些新的改动即可。输入时为了判断左端点相同的区间，设立 flag 数组用 flag[i]表示是否以 i 为左端点的区间有多个。在遍历整个 range 数组的过程中，对于每一个区间，start 保存其初始左端点，judge 是标志位，从其左端点开始，判断其覆盖范围（包含两个端点）是否有其他区间，如果该区间完整遍历后无其他区间且 flag[start]标志位为 0，就是一个“单身”区间，打印输出。如果有其他区间，选择两个区间右端点中较大的值作为新的循环结束值。

例程

```
#include<stdio.h>
#define RANGE (100000+1)
int range[RANGE],flag[RANGE];
void single_search();
```

```
int main()
{
    int a,b;
    while(scanf("%d%d",&a,&b)==2){
        if(b>range[a]){
            if(range[a])
                flag[a]=1;
            range[a]=b;
        }
    }
    single_search();
    return 0;
}
void single_search()
{
    int i,n,start,judge;
    for(i=0;i<RANGE;++i){
        if(range[i]!=0){
            judge=0;
            start=i;
            for(n=range[i],++i;i<=n;++i){
                if(range[i]!=0){
                    n=n>range[i]?n:range[i];
                    judge=1;
                }
            }
            if(!judge&&!flag[start])
                printf("%d %d\n",start,range[start]);
            --i;
        }
    }
}
```

G 温度转换

时间限制: 1000 ms 内存限制: 65536 kb
总通过人数: 1615 总提交人数: 1642

题目描述

已经学习了如何将摄氏温度 ℃ 转换为华氏温度 ℉，现在给出一个整数，请输出从 0°C 到这个目标温度中所有整数温度对应的华氏温度
(F=C*9/5+32)(

输入

一个整数 a（0<a<100)

输出

a+1 行，从到 0℃到 a℃所有整数对应的华氏温度，结果向下取整。

输入样例

```
3
```

输出样例

```
0 C = 32 F
1 C = 33 F
2 C = 35 F
3 C = 37 F
```

HINT

不需要长度控制,确认输出格式与样例一致

考察知识点

循环，难度系数 2

解题思路：

考察的是简单循环输出，读入需要输出的上限，循环从 0 到上限按公式输出即可。int 型会忽略结果的小数部分，实际就是向下取整，注意输出结果的空格和符号正确。

代码

```
#include<stdio.h>
int main()
{
    int num;
    scanf("%d",&num);
    for(int i=0;i<=num;i++)
    {
        printf("%d C = %d F\n",i,i*9/5+32);
    }
    return 0;
}
```

H 绝对值排序问题

时间限制: 1000 ms 内存限制: 65536 kb
总通过人数: 996 总提交人数: 1094

题目描述

橙橙在学习排序的过程中，有了一些思考，所以打算在这里和大家分享一下。

课上所学的排序方法，比较朴素，仅仅是按数字的大小进行排序，那么如果按照数字的其他性质进行排序，例如绝对值，该怎样安排算法的设计呢？

请你帮助橙橙，设计出一个将若干数据按绝对值进行排序并输出的程序，你能做到吗？

加油吧！

输入

输入分为两行。

第一行是数据数量 n(1≤n≤100)。
第二行是 n 个数据(全部在 `int` 范围内)。

输出

输出这 n 个数据按绝对值从小到大的排序。
(对绝对值相同的两个相反数排序时，负数在先。)

输入样例

```
3
-2 3 1
```

输出样例

```
1 -2 3
```

考察知识点

```
数组排序 （例如冒泡排序）
难度系数 2
```

解题思路：

本题解法很多，下面以冒泡排序为例进行说明。

解题时，将普通的冒泡排序稍作即可，将判断条件中的直接比较大小换成比较绝对值的大小。

注意两个相反数相邻的情况，这种情况是否交换次序取决于是负数在前还是正数在前。题目要求负数在前，所以如果负数在前就不需要交换，如果整数在前就需要交换。也可以选择下面代码中的实现方法，即无论正数负数哪一个在前，都更新为负数在前。

AC 代码：

```
#include<stdio.h>
#include<math.h>
#define N(x) (sizeof(x)/sizeof(x[0]))
void bub(int[], int );

int main()
{
    int m;
    scanf("%d",&m);
    int a[m],i;
    for(i=0;i<m;i++)
        scanf("%d",&a[i]);
    bub(a,N(a));
    for(i=0;i<m;i++)
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}

void bub(int a[], int n)
{
    int i,j,h;
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(abs(a[j])>abs(a[j+1]))
            {
                h=a[j];
                a[j]=a[j+1];
                a[j+1]=h;
            }
            if( abs(a[j])==abs(a[j+1]) && a[j]!=a[j+1] )
            {
                a[j]=0-abs(a[j]);
                a[j+1]=abs(a[j]);
            }
        }
    }
}
```

时间限制：1000ms　内存限制：65536kb

通过率：254/316 (80.38%)　正确率：254/974 (26.08%)

题目描述

珠心算是一种通过在脑中模拟算盘变化来完成快速运算的一种计算技术。珠心算训练，既能够开发智力，又能够为日常生活带来很多便利，因而在很多学校得到普及。

Terry 想到一种锻炼珠心算加法能力的方法。他随机生成一个正整数集合，集合中的数各不相同，然后请你回答：其中有多少个数，恰好等于集合中另外两个（不同的）数之和？

输入

输入共两行，第一行包含一个整数 n，表示给出的整数个数。（$3 \le n \le 100$）
第二行有 n 个正整数，每两个正整数之间用一个空格隔开，表示给出的正整数。正整数小于等于 10000。

输出

输出一个整数，表示答案。

输入样例

```
4
1 2 3 4
```

输出样例

```
2
```

样例说明

1+2=3，1+3=4，故满足测试要求的答案为 2。注意，加数和被加数必须是集合中两个不同的数。

考察知识点

本题考察了循环、数组和排序。
难度系数 6

解题思路

本题要计算给出的集合中有多少个元素是另外两个元素的和，我们发现元素个数小于 100 个，所以想到用一种非常朴素的方法就能解决此问题。枚举所有种和的情况，并遍历整个集合，看是否有元素值等于这个和，如果有就让计数器+1。我们将所有元素进行排序，利用元素的有序性可以知道两个数的和的索引值一定在两个数的后边。注意本题求的不是满足条件等式的数量，所以要用一个数组 vk[]来记录重复与否，重复的不计数。具体的操作见代码。

标程

```
#include <stdio.h>

int a[101],ans,n,vk[101];  //a[]来记录所有的数，vk[]表示这个数是否被计数过
int main() {
    int by;           //冒泡排序的中间变量
    ans=0;
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
        scanf("%d",&a[i]);
    for(int i=1;i<=n;i++) {  //先对数据进行排序 方便操作
        for(int j=i+1;j<=n;j++)
            if(a[i]>a[j]) {
                by=a[i];
                a[i]=a[j];
                a[j]=by;
            }
    }
    for(int i=1;i<n-1;i++)
        for(int j=i+1;j<n;j++)//不能与 i 一样
            for(int k=j+1;k<=n;k++)//合数肯定要比加数他的数要大
                if(a[i]+a[j]==a[k]&&vk[k]==0) {//v 判重，0 代表未标记
```

```
                ans++;
                vk[k]=1;
    }
    printf("%d\n",ans);
    return 0;
}
```

J Tarpe 酋长的电话密码_备用

时间限制: 1000 ms 内存限制: 65536 kb
通过率：154/190 (81.05%) 正确率：154/335 (45.97%)

题目描述

Tarpe 酋长的记性很差，自己的密码都记不住，所以一般会在自己的座位前贴一张纸条来提示密码，纸条上只有一串数字。但是 Tarpe 酋长的密码是很长的字符串。

酋长的密码映射规则是这样的（和电话号码一样）：

酋长的问题是，给定一个数字字符串，按字典序返回数字所有可能表示的字符串。

特殊符号不用考虑

输入

一行，一个数字字符串（保证长度小于 10）

输出

按字典序输出所有可能的字符串，每种结果一行，最后需要换行

输入样例

```
23
```

输出样例

```
ad
ae
af
bd
be
bf
cd
ce
cf
```

HINT

```
难题慎入，可参考递归实现全排列
```

考察知识点

```
递归，难度系数 9
```

解题思路

参考全排列问题的求解思路，我们可以使用递归的方法解答此题。

考虑数字串的每一位，在最终的字符串中可能对应 3 或 4 种字母。对比全排列问题：全排列问题的每一位可能对应的是某个未使用过的数字，而本问题中每一位对应的字母的集合是固定的。因此，只需要预先记录好每一种数字对应哪些字母，就可以使用递归的方式计算出所有的字符串。

对于按字典序输出这个需求，有两种解决思路：其一是存储下得到的所有字符串，并且进行排序，但是字符串的数量可能非常多，极限会达到 4^10≈1,000,000 个，存储开销过大。另一种解决方案是在递归的过程中，使用循环枚举每一位字母的时候从小到大循环，这样就可以确保先找到的字符串字典序一定比后找到的要小。

代码

```c
#include <stdio.h>

char map[10][5] = {
    "",
    ""    , "abc", "def",
    "ghi" , "jkl", "mno",
    "pqrs", "tuv", "wxyz"
};

char ans[20];

void dfs(const char *s, int pos)
{
    if (!s[pos]) {
        ans[pos] = 0;
        printf("%s\n", ans);
        return;
    }
    int c = s[pos] - '0';
    int n = (c == 7 || c == 9) ? 4 : 3;
    for (int i = 0; i < n; ++ i) {
        ans[pos] = map[c][i];
        dfs(s, pos + 1);
    }
}

int main()
{
    char str[20];
    scanf("%s", str);
    dfs(str, 0);
    return 0;
}
```