北航历年机试题汇总

2008 机试题目

1.素数

输入一个整数，要求输出所有从 1 到这个整数之间个位为 1 的素数，如果没有则输出-1（30 分）

2.旋转矩阵

任意输入两个 9 阶以下矩阵，要求判断第二个是否是第一个的旋转矩阵，如果是，输出旋转角度（0、90、180、270），如果不是，输出-1。
要求先输入矩阵阶数，然后输入两个矩阵，每行两个数之间可以用任意个空格分隔。行之间用回车分隔，两个矩阵间用任意的回车分隔。（60 分）

3.字符串匹配

从 string.in 中读入数据，然后用户输入一个短字符串。要求查找 string.in 中和短字符串的所有匹配，输出行号、匹配字符串到 string.out 文件中。匹配时不区分大小写，并且可以有一个用中括号表示的模式匹配。如 “aa[123]bb” ，就是说 aa1bb、aa2bb、aa3bb 都算匹配。（60 分）

2009 机试题目

1 立方根逼近

给出立方根的逼近迭代方程 y(n+1) = y(n)*2/3 + x/(3*y(n)*y(n)),其中 y0=x.求给定的 x 经过 n 次迭代后立方根的值。

要求:double 精度,保留小数点后面六位。(送分题)

输入:x n

输出:迭代 n 次后的立方根

```
sample
input: 3000000 28
output:144.224957
```

2 数组排序

输入一个数组的值,求出各个值从小到大排序后的次序。

输入:输入的第一个数为数组的长度,后面的数为数组中的值,以空格分割

输出:各输入的值按从小到大排列的次序。

```
sample
input:
4
-3 75 12 -3
output:
1 3 2 1
```

3 字符串的查找删除

给定文件 filein.txt 按要求输出 fileout.txt。

输入: 无空格的字符串

输出: 将 filein.txt 删除输入的字符串(不区分大小写),输出至 fileout.txt

sample

输入:in

输出:将 filein.txt 中的 In、IN、iN、in 删除,每行中的空格全部提前至行首,输出至 fileout.txt

filein.txt 中的值为:

```
#include <stdio.h>
int main()
{

printf(" Hi ");
}
```

输出的 fileout.txt 为

```
#clude<stdio.h>
tma()
{

prtf("Hi");
}
```

2010 机试题目

1.泰勒公式。

利用泰勒公式求 cos(x)，公式已给，重要的就是注意细节（比如阶乘的存储最好用 double 类型），二级 C 语言的难度...

2.归并字符串。

归并两个有序字符串，要求输出不能有重复字符（数据结构上做过 N 遍的 Merge 函数）

3.数组是否相等。

两个整数数组（无序，可有重复元素），判断两个整数数组是否完全相同（重复元素的话，重复次数也要相同）

2011 机试题目

共有三道编程题，第一道题 20 分，第二道题 15 分，第三道题 15 分，总分 50 分。考试时间：2 个小时。注意:所编程序必须符合标准 C 语言要求，提交程序名必须遵循题中说明。程序中输入/输出必须按照程序要求(可参见输入/输出样例)，不要填加任何额外信息。如果提交 C++程序，必须先选择 C++语言。

1.求孪生数

【问题描述】

孪生数定义：如果 A 的约数（因数，包含 1，但不包含 A 本身）之和等于 B，B 的约数（因数）之和等于 A，A 和 B 称为孪生数（A 和 B 不相等）。试找出正整数 M 和 N 之间的孪生数。

【输入形式】

从控制台输入两个正整数 M 和 N（1<=M<N<=20000），中间用一个空格分隔。

【输出形式】

在标准输出上输出符合题目描述的 M 和 N 之间的全部孪生数对(包括 M 和 N)。每行输出一对孪生数，用一个空格隔开，小的先输出；各行孪生数按照第一个数从小到大的顺序输出，一对孪生数只输出一次。如果没有符合要求的孪生数对，则输出字符串 “NONE” 。

【输入样例 1】

```
20 300
```

【输出样例 1】

```
220 284
```

【输入样例 2】

```
200 250
```

【输出样例 2】

```
NONE
```

【样例说明】

样例 1 输入的区间为[20,300]，其间有一对孪生数对，即：220（1+2+4+5+10+11+20+22+44+55+110=284）和 284（1+2+4+71+142=220）。

样例 2 输入的区间是[200,250]，其间没有孪生数对，所以输出字符串：NONE。

【评分标准】

该题要求输出区间中的所有孪生数对，共有 5 个测试点，提交程序文件名为 example1.c 或 example1.cpp。

```
/**************************************************************************
****
* Problem: 北航 2011 上机题 1
* Copyright 2011 by Yan
* DATE:
* E-Mail: yming0221@gmail.com
**************************************************************************
*/

#include<stdio.h>

int tab[20001];/*存储每个数的因数之和*/

int main()
{
    int m,n;
```

```
    int i,j;
    int flag;
    for(i=1;i<20001;i++)
    {
        for(j=1;j<i;j++)
        {
            if(i%j==0) tab[i]+=j;
        }
    }
    while(scanf("%d %d",&m,&n)!=EOF)
    {
        flag=0;
        for(i=m;i<=n;i++)
        {
            for(j=m;j<i;j++)
            {
                if(tab[i]==j && tab[j]==i) printf("%d %d ",j,i),flag=1;
            }
        }
        if(flag==1) printf("\n");
        if(flag==0) printf("NONE\n");
    }
    return 0;
}
```

上面使用打表的方式，这样打表只计算一遍，如果是多重输入的话推荐这样，如果是单重输入，可以将打表放在输入后，减少打表时间复杂度。

2.矩阵替换

【问题描述】

先输入两个矩阵 A 和 B，然后输入替换位置（左上角），编写程序将矩阵 A 中从替换位置开始的子矩阵（与 B 同样大小）替换为 B，并输出替换后的矩阵。

【输入形式】

从控制台先输入矩阵 A 的行数和列数(行数和列数均大于等于 1,小于等于 20)，然后在新的行上输入矩阵 A 的各行数字（以一个空格分隔的整数)。再以同样的

方式输入矩阵 B。最后输入替换位置（用一个空格分隔的两个整数表示，行数和列数都从 1 开始计数，因此两个整数都大于等于 1）。若替换位置超出了矩阵 A 的行数或列数，则原样输出矩阵 A。

【输出形式】

在标准输出上分行输出替换后的矩阵，每行中各数字之间以一个空格分隔。

【输入样例 1】

```
5 6
10 2 34 -1 800 90
2 76 56 -200 23 1
35 0 0 98 8 3000
2000 100 -1 1 2 0
8 7 85 963 496 8
2 3
9 9 9
9 9 9
3 3
```

【输出样例 1】

```
10 2 34 -1 800 90
2 76 56 -200 23 1
35 0 9 9 9 3000
2000 100 9 9 9 0
8 7 85 963 496 8
```

【样例 1 说明】

输入的矩阵 A 为 5 行 6 列，矩阵 B 是 2 行 3 列，替换位置为第 3 行的第 3 列，即：将 A 中第 3 行第 3 列开始的、行数为 2 列数为 3 的子矩阵替换为 B。

【输入样例 2】

```
3 4
10 2 34 -1
2 76 56 -200
35 0 0 98
2 3
9 9 9
9 9 9
```

```
2 3
```

【输出样例 2】

```
10 2 34 -1
2 76 9 9
35 0 9 9
```

【样例 2 说明】

输入的矩阵 A 为 3 行 4 列，矩阵 B 是 2 行 3 列，替换位置为第 2 行的第 3 列，即：将 A 中第 2 行第 3 列开始的、行数为 2 列数为 3 的子矩阵替换为 B。但该子矩阵超出了 A 的范围，所以只实现了部分替换。

【评分标准】

该题要求输出替换后的矩阵，共有 5 个测试点，提交程序文件名为 example2.c 或 example2.cpp。

```
/**************************************************************************
****
* Problem: 2011 北航上机 2 题
* Copyright 2011 by Yan
* DATE:
* E-Mail: yming0221@gmail.com
**************************************************************************
*/

#include<stdio.h>
#define MAX 21
#define MIN(a,b) (a)>(b)?(b):(a)
int value[MAX][MAX];
int sub[MAX][MAX];

int main()
{
    freopen("input","r",stdin);
    int m,n;/*原矩阵大小*/
    int i,j;
    int x,y;/*替换矩阵大小  x 行、y 列*/
    int posx,posy;
```

```
    int tmp1,tmp2;
    while(scanf("%d %d",&m,&n)!=EOF)
    {
        for(i=0;i<m;i++)
            for(j=0;j<n;j++) scanf("%d",&value[i][j]);
        scanf("%d %d",&x,&y);
        for(i=0;i<x;i++)
            for(j=0;j<y;j++) scanf("%d",&sub[i][j]);

        scanf("%d %d",&posx,&posy);

        tmp1=MIN(posx+x,m);tmp2=MIN(posy+y,n);

        for(i=posx-1;i<tmp1;i++)
            for(j=posy-1;j<tmp2;j++)
                value[i][j]=sub[i-posx+1][j-posy+1];
        for(i=0;i<m;i++)
        {
            for(j=0;j<n;j++)
                printf("%d ",value[i][j]);
            printf("\n");
        }

    }
    return 0;
}
```

3.字符串扩展

【问题描述】

从键盘输入包含扩展符'-'的字符串，将其扩展为等价的完整字符，例如将 a-d 扩展为 abcd，并输出扩展后的字符串。

要求：只处理[a-z]、[A-Z]、[0-9]范围内的字符扩展，即只有当扩展符前后的字符同时是小写字母、大写字母或数字时才进行扩展，其它情况不进行扩展，原样输

出。例如：a-R、D-e、0-b、4-B 等字符串都不进行扩展。

【输入形式】

从键盘输入包含扩展符的字符串

【输出形式】

输出扩展后的字符串

【输入样例 1】

ADEa-g-m02

【输出样例 1】

ADEabcdefghijklm02

【输入样例 2】

cdeT-bcd

【输出样例 2】

cdeT-bcd

【样例说明】

将样例 1 的输入 ADEa-g-m02 扩展为 :ADEabcdefghijklm02 ;样例 2 的输入 cdeT-bcd 中，扩展符前的字符为大写字母，扩展符后的字符为小写字母，不在同一范围内，所以不进行扩展。

【评分标准】

结果完全正确得 15 分，共 5 个测试点，每个测试点 3 分，提交程序文件 expand.c 或 expand.cpp。

```
/**************************************************************************
****
* Problem: 2011 北航上机题 3
* Copyright 2011 by Yan
* DATE:
* E-Mail: yming0221@gmail.com
**************************************************************************
*/

#include<stdio.h>

char str[100];

int main()
{
    freopen("input","r",stdin);
    int i,j;
    while(scanf("%s",str)!=EOF)
    {
        for(i=0;str[i]!='\0';i++)
        {
            if(str[i]=='-')
            {
                if(str[i-1]>='A' && str[i-1]<='Z' && str[i+1]>='A'
                    && str[i+1]<='Z' && str[i+1]>str[i-1])/*大写字母扩展*/
                {
                    for(j=str[i-1]+1;j<str[i+1];j++) printf("%c",j);
                }
                else if(str[i-1]>='a' && str[i-1]<='z' && str[i+1]>='a'
                    && str[i+1]<='z' && str[i+1]>str[i-1])/*小写字母扩展*/
                {
                    for(j=str[i-1]+1;j<str[i+1];j++) printf("%c",j);
                }
                else if(str[i-1]>='0' && str[i-1]<='9' && str[i+1]>='0'
                    && str[i+1]<='9' && str[i+1]>str[i-1])/*数字扩展*/
                {
                    for(j=str[i-1]+1;j<str[i+1];j++) printf("%c",j);
                }
                else printf("-");
            }
            else printf("%c",str[i]);/*不扩展*/
        }
        printf("\n");
    }
```

```
    return 0;
}
```

2012 机试题目

1.某数分解成若干连续整数的和

15=1+2+3+4+5

15=4+5+6

15=7+8

不能分解则输出 NONE

```
#include<stdio.h>
int main(){
    int i,j,k,n,sum;
    bool f=0;
    scanf("%d",&n);
    for(i=1;i<n;i++)
        for(j=i+1;j<n;j++){
           sum=(j-i+1)*(j+i)/2;
           if(sum==n){
               f=1;
               for(k=i;k<=j;k++)printf("%d ",k);
           printf("\n");
           }

        }
        if(f==0){
            printf("NONE\n");
        }
    return 0;
}
```

2.小岛面积

```
1 1 1 1 1 1
1 1 0 0 0 1
1 0 0 0 1 0
1 1 0 1 1 1
0 1 0 1 0 0
1 1 1 1 1 1
```

上面矩阵的中的 1 代表海岸线，0 代表小岛。求小岛面积（即被 1 中包围的 0 的个数)。注意：仅求这样的 0，该 0 所在行中被两个 1 包围，该 0 所在列中被两个 1 包围。

输入：

第一行输入一个整数 N，表示输入方阵的维数,输入一个 N 维方阵

输出：

小岛面积

样例输入：

```
6
1 1 1 1 1 1
1 1 0 0 0 1
1 0 0 0 1 0
1 1 0 1 1 1
0 1 0 1 0 0
1 1 1 1 1 1
```

样例输出：8

```
#include<stdio.h>
#include<string.h>
int main(){
    int n,i,j,tol=0;
    int a[100][100],dir[100][4];
    scanf("%d",&n);
    for(i=0;i<n;i++)
```

```
        for(j=0;j<n;j++){
            scanf("%d",&a[i][j]);
        }
    memset(dir,-1,sizeof(dir));
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
        {
            if(a[i][j]==1){
                if(dir[i][0]==-1)dir[i][0]=j;
                if(dir[j][2]==-1)dir[j][2]=i;
                dir[i][1]=j;
                dir[j][3]=i;
            }
        }
        for(i=0;i<n;i++){
            for(j=0;j<n;j++)
            {
                if(a[i][j]==0){
                   if(i<dir[j][3]&&i>dir[j][2]&&j<dir[i][1]&&j>dir[i][0])
                        tol++;
                }
            }
        }
        printf("%d\n",tol);
    return 0;
}
```

3.统计关键字出现位置

输入：

一行标准 c 语言代码(字符个数小于 300)，统计出该字符串中关键字的 if，while，for 所在的位置，按照关键字出现的顺序依次输出。注意双引号内的不需要统计。

输入：一行标准 c 语言代码，字符个数小于 300

输出：

关键字 if，while，for 对应的位置，按照关键字出现的顺序依次输出。输出格式

为：关键字，后跟冒号，然后是出现的位置。扫描到关键字就输出，每个输出占一行。

```
#include<stdio.h>
#include<string.h>
int judge(char c){
    if(c>='0'&&c<='9')return 1;
    else if(c>='a'&&c<='z')return 2;
    else if(c=='_')return 3;
    else if(c=='"')return 4;
    else return 5;
}
int main(){
    char a[500];
    //FILE *fp=fopen("data.txt","r");
    gets(a);
    puts(a);
    int i,len=strlen(a)-1,yin=0,qian=0;
    for(i=0;i<len;i++){
        if(judge(a[i])==4||yin==0){
            if(judge(a[i])==5)qian=1;
            else if(qian==1&&judge(a[i])==2){

if(a[i]=='i'&&a[i+1]=='f'&&judge(a[i+2])==5&&i+2<len)printf("if:%d\n",i
+1);
                else
if(a[i]=='w'&&a[i+1]=='h'&&a[i+2]=='i'&&a[i+3]=='l'&&a[i+4]=='e'&&judge
(a[i+5])==5&&i+5<len)printf("while:%d\n",i+1);
                else
if(a[i]=='f'&&a[i+1]=='o'&&a[i+2]=='r'&&judge(a[i+3])==5&&i+3<len)print
f("for:%d\n",i+1);
                else qian=0;
            }
            else if(judge(a[i])==4){
            if(yin==0){
               yin=1;
            }
            else yin=0;
            }

        }
```

```
    }
    return 0;
}
```

2013 机试题目

1.真分数约分

题目描述：给一个真分数的分子分母，输出约分后的分子分母。

输入	输出
3 6	1/2
15 25	3/5

```
#include<stdio.h>
int main(){
    int a,b;
    while(scanf("%d%d",&a,&b)!=EOF){
        if(b==0){//需要注意的！分子不能为零
            printf("error");
        }
        for(int i=a;i>1;i--){
            if(a%i==0&&b%i==0){
                a=a/i;
                b=b/i;
            }
        }
        printf("%d/%d\n",a,b);
    }
return 0;
}
```

2.简单八皇后问题

题目：简单八皇后

描述：如何能够在 8 X 8 的国际象棋棋盘上放置八个皇后，使得任何一个皇后都无法直接吃掉其他已经正确地放置了七个皇后，编写程序求出最后一个皇后的位置。

注意： 皇后能横、竖、斜着走

输入：按照棋盘行从小到大的顺序从控制台输入已正确放置的皇后的列数，未放置皇后的行用字符 “” 来表示。各列数和字符 “” 之间没有任何其他字符分隔，在输入末尾有回车换行符。假如上图第四行未放置皇后，其他七个皇后都已经放置完毕，则输入的形式为：627*4853

输出：在标准输出上输出最后一个皇后放置的列数，若无解，输出字符串 “No Answer”

样例：

输入 1：

627*4853

输出 1：

1

输入 2：

3681*752

输出 2：

4

输入 3：

1357246*

输出 3：

No Answer

```
#include<stdio.h>
#include<math.h>
char s[20];
int a[10]={0};
void change(int &p,int &f){//字符转换整形数字
    for(int i=0;i<8;i++){
        if(s[i]>='1'&&s[i]<='8'){
            a[i+1]=s[i]-'0';
        }
        else if(s[i]=='*'){
            p=i+1;
        }
        else {
            printf("No Answer\n");
            f=0;
        }
    }
}
void fun(int p){
    int i,j;
    for(i=1;i<=8;i++){//判断其他行是否有冲突
        if(i<p||i>p){//*之前某一行
            for(j=1;j<i;j++){//之前之前的一行
                if(a[i]==a[j])break;//错误
                if(abs(i-j)==abs(a[i]-a[j]))break;
            }
        }
    }
    if(i<8){
        printf("No Answer");
        return;
    }
    int temp=0;
    for(i=1;i<=8;i++){//*表示的那一行旗子放在哪一列
```

```
        for(j=1;j<=8;j++){//与其他列对比
            if(j!=p){
                if (i==a[j]||abs(p-j)==abs(i-a[j]))break;
            }
        }
        if(j==9){
            printf("%d\n",i);
            return;
        }
    }
    if(i==9){
        printf("No Answer\n");
        return;
    }
}
int main(){
    while(scanf("%s",s)!=EOF){
        int f=1;//输入是否合法；
        int p=0;//指示*的位置；
        change(p,f);
        if(f)fun(p);
    }
return 0;
}
```

3.科学计数法

描述：给出一个标准输入的正数（开头末尾没有多余的 0），输出其科学计数法表示结果.

样例：

输入 0.000002，输出 2e-6；

输入 123.456，输出 1.23456e2；

输入 123456，输出 1.23456e5

输入 1.2354，输出 1.2345

输入 10000，输出 1e4；

输入 0.0000/0，输出 0

输入 1.000/1，输出 1

//细节非常重要！好多种情况要考虑，可以先从最常见的下手 12.356，然后再加上各种特殊情况

```
#include<stdio.h>
#include<string.h>
char a[100];
double b;
int main(){
    while(scanf("%s",&a)!=EOF){
        int i,p,e=0;
        int len=strlen(a),len1=len;
        for(i=0;i<len;i++){
            if(a[i]=='.')break;
        }
        p=i;//记录.的位置,若没有.则 p=len;

        //无整数部分(0.00/0/0.123/0.1/0.00023/0.00020)
        if(a[0]=='0'){
            for(i=2;i<len;i++){//e 作为指数计数
                if(a[i]=='0')e--;//0.0002300
                else break;
            }
            e--;
            if(i==len){//说明全 0
                printf("0\n");//0.0000，不输出‘.’和 e
                continue;//下一轮输入
            }
            else{//不全为 0
                for(i=len-1;i>0;i--) {//0.00200
                    if(a[i]=='0') len1--;//字符串有效长度 len1
                    if(a[i]!='0')break;//直到倒数的位数不为 0
                }
                if(len1==((-e)+2))printf("%c",a[(-e)+1]);//0.002 不输出‘.’和 e
                else printf("%c.",a[(-e)+1]);//0.0023
```

```
                for(i=(-e)+2;i<len1;i++)  printf("%c",a[i]);//其他位数依次输出
                printf("e%d\n",e);
                continue;  //下一轮输入
            }
        }

        //有整数部分（1/1.00/1000/1.02300/12345/12.350/1230）
        //printf("%c.",a[0]);
        for(i=len-1;i>0;i--) {//除去结尾的 0
            if(a[i]=='0') len1--;
            if(a[i]!='0')break;
        }
        if(len1==1||(len1==2&&a[1]=='.')){//1 /1.0000/1000（写成科学计数法之后只有一个有效位，因为 0 都除去了）
            if(p==len&&len!=1)printf("%ce%d\n",a[0],len-1);//1000
            else printf("%c\n",a[0]);//1/1.00
            continue;
        }
        else printf("%c.",a[0]);//打印第一个元素
        for(i=1;i<len1;i++){//一次打印后面的，‘.’不打印
            if(a[i]=='.')continue;
            printf("%c",a[i]);
        }
        if(p-1!=0)printf("e%d\n",p-1);//指数与 p 的位置有关
        else printf("\n");//1.0256，当小数点前只有一位时不打印 e
    }
return 0;
}
```

2014 机试题目

1.阶乘数

输入一个正整数，输出时，先输出这个数本身，跟着一个逗号，再输出这个数的各位数字的阶乘和，等号，阶乘和的计算结果，并判断阶乘和是否等于原数，如果相等输出 Yes，否则输出 No。题目说明输入的正整数以及其各位阶乘和都不会超出 int 型的表示范围。

输入样例 1：

```
145
```

输出样例 1：

```
145,1!+4!+5!=145
Yes
```

输入样例 2：

```
1400
```

输出样例 2：

```
1400,1!+4!+0!+0!=27
No
```

```
#include<stdio.h>
int fun(int b){//求阶乘
    int mul=1;
    for(int i=1;i<=b;i++){
        mul*=i;
    }
    return mul;
}
int main(){
    int a;
    while(scanf("%d",&a)!=EOF){
        int sum=0,c[100];//c 倒序放 a 的每一位
        printf("%d,",a);
        int temp=a;
        for(int i=0;i<100;i++){//将 a 的每位放入 c 数组，i 计数位数
            if(temp>0){
                c[i]=temp%10;
                sum=sum+fun(c[i]);//同时计算
                temp=temp/10;
            }
            else break;//放完了
        }
        for(int j=i-1;j>=0;j--){//倒序输出
            if(j!=0) printf("%d!+",c[j]);//注意格式处理
            else printf("%d!=",c[j]);
        }
```

```
        printf("%d\n",sum);
        if(a==sum)printf("Yes\n");//注意换行符
        else printf("No\n");
    }
    return 0;
}
```

2.五子棋

【输入】一个 19*19 的矩阵，只包含数字 0、1、2，表示两人下五子棋的棋牌状态，1、2 分别表示两人的棋子，0 表示空格。要求判断当前状态下是否有人获胜（横向、竖向或者斜线方向连成 5 个同色棋子）

【输出】输入样例保证每条线上至多只有连续 5 个同色棋子，并且保证至多只有 1 人获胜。如果有人获胜，输出获胜者（1 或 2）加一个冒号，接着输出获胜的五连珠的第一个棋子的坐标，从上到下从左到右序号最小的为第一个，序号从 1 开始编号。如果无人获胜，输出 no

```
#include<stdio.h>
struct node{//结构体数组
    int x;
    bool right,down,right_down,left_down;
}a[20][20];
int main(){
    //FILE *fp=fopen("test.txt","r");//打开文件，也可以直接复制到输入台
    int i,j;
    for(i=1;i<20;i++){
        for(j=1;j<20;j++){//初始化数组
            //fscanf(fp,"%d",&a[i][j].x);//从文件中将数据读入数组；//
            scanf("%d",&a[i][j].x);

    a[i][j].down=a[i][j].left_down=a[i][j].right=a[i][j].right_down=fals
e;//初始化
        }
    }
//  fclose(fp);//关闭文件
    int temp;//记录当前旗子的值
```

```
    bool result=false;
    for(i=1;i<20;i++){
        for(j=1;j<20;j++){
            temp=a[i][j].x;
            int k;
            if(temp==0) continue;//没有旗子
            if(j<=15&&a[i][j].right==false){//左右方向还没被标记；遍历右边
                for(k=1;k<5;k++){
                    if(a[i][j+k].x==temp){
                        a[i][j+k].right=true;
                    }
                    else break;//颜色不相同
                }
                if(k==5){
                    result=true;//找到了获胜组合
                    break;
                }
            }
            if(j<=15&&i<=15&&a[i][j].right_down==false){//左上右下方向还没被标记；遍历右下
                for(k=1;k<5;k++){
                    if(a[i+k][j+k].x==temp){
                        a[i+k][j+k].right_down=true;
                    }
                    else break;//颜色不相同
                }
                if(k==5){
                    result=true;
                    break;
                }
            }
            if(i<=15&&a[i][j].down==false){//遍历下边
                for(k=1;k<5;k++){
                    if(a[i+k][j].x==temp){
                        a[i+k][j].down=true;
                    }
                    else break;//颜色不相同
                }
                if(k==5){
                    result=true;
                    break;
                }
            }
            if(j>=5&&i<=15&&a[i][j].left_down==false){//遍历左下
```

```
                for(k=1;k<5;k++){
                    if(a[i+k][j-k].x==temp){
                        a[i+k][j-k].left_down=true;
                    }
                    else break;//颜色不相同
                }
                if(k==5){
                    result=true;
                    break;
                }
            }

        }
        if(result==true)break;//两重循环要退出两次
    }
    if(result==true){
        printf("%d:(%d,%d)\n",temp,i,j);
    }
    else{
        printf("no\n");
    }
    return 0;
}
```

3.排版

【输入】输入若干行字符，表示某电影的演职员表，每行只有一个冒号，冒号前面是职位，冒号后面是姓名

【输出】要求把各行冒号对齐，删除多余空格后输出.

【说明】

o 先输入一个数字，表示排版要求的冒号位置，该位置号保证比各行冒号前的最大字符数还要大

o 再输入若干行字符，最多 50 行，每行最多 100 个字符，除空格、制表符和回车之外都是有效字符;

要求每行的冒号处于格式要求的位置，冒号两边与有效单词之间各有一个空格，

冒号前面的单词之间只有一个空格（删除多余的空格和制表符）；

o 在冒号左边右对齐，前面全由空格填充，冒号后面的单词之间也只有一个空格，在冒号右边左对齐，最后一个单词后不加空格直接换行。

```
#include<stdio.h>
#include<string.h>
int main(){
    int n;
    char a[60][120];
    scanf("%d",&n);//冒号位置
    int i=0,j,b[60];//记录冒号的位置
    getchar();//吸收换行
    while(gets(a[i])!=NULL){
        i++;
    }
    int line=i;//记录行下标最大值从 0 开始
    int c[60]={0};//记录冒号之前有效位数，
    int d[60]={0};//记录单词的个数，与 c[]共同决定从哪里开始输出
    bool f=false;//标志现在是单词
    for(i=0;i<line;i++){
        for(j=0;j<(int)strlen(a[i]);j++){
            if(a[i][j]==':'){
                b[i]=j;
                break;
            }
            else if(a[i][j]!=' '&&a[i][j]!='\t') {
                c[i]++;
                if(a[i][j+1]=='       '||a[i][j+1]=='\t'||a[i][j+1]==':')
d[i]++;
            }
        }
    }
    for(i=0;i<line;i++){//输出注意格式控制
        for(int k=0;k<n-c[i]-d[i]-1;k++)printf(" ");//输出空格
        for(j=0;j<(int)strlen(a[i]);j++){
            if(j<b[i]){//冒号之前，重要的是前面的空格输出几个
                if(a[i][j]==' '||a[i][j]=='\t') continue;
                else                                        if(a[i][j+1]=='
'||a[i][j+1]=='\t'||a[i][j+1]==':')//一个单词结尾
                    printf("%c ",a[i][j]);
```

```
                else printf("%c",a[i][j]);
            }
            else if(j==b[i]){//
                printf(": ");
            }
            else{//重要的是结尾的控制
                if(a[i][j]==' '||a[i][j]=='\t') continue;
                else if(a[i][j+1]==' '||a[i][j+1]=='\t')
                    printf("%c ",a[i][j]);
                else printf("%c",a[i][j]);
                if(j==(int)strlen(a[i])-1) printf("\n");
            }
        }
    }
return 0;
}
```

2015 机试题目

1.相亲数

输入两个正整数 a 和 b，若 a 的所有约数（包括 1，不包括 a 本身）的和等于 b，且 b 的所有约数（包括 1，不包括 b 本身）的和等于 a，则两个数是相亲数。要求分别输出两个正整数的约数和的式子，再换行后输出 1 或 0，表示这两个数是否为 “相亲数” 。

o 编写一个程序计算 x 和 y 分别除了本身以外的因子之和，并判断 x 和 y 是不是一对相亲数。x 和 y 为大于 1 的 int 范围内的整数。

【样例输入】

220 284

【样例输出】

220,110+55+44+22+20+11+10+5+4+2+1=284

284,142+71+4+2+1=220

1

输入: x 和 y，空格隔开。

输出:

第一行输出 x，一个逗号，x 的除了本身以外的因子之和的计算过程（见题意，要求降序输出每个因子），不要有多余的空格。

第二行输出 y，一个逗号，y 的除了本身以外的因子之和的计算过程（见题意，要求降序输出每个因子），不要有多余的空格。

第三行，如果 x 和 y 是一对相亲数输出 1，否则输出 0。文末换行可有可无。

```
#include<stdio.h>
void deal(int a,int &sum){
    printf("%d,",a);
    for(int i=a-1;i>=1;i--){
        if(a%i==0){
            sum+=i;
            if(i!=1)printf("%d+",i);//注意格式调整
            else printf("%d=",i);
        }
    }
    printf("%d\n",sum);

}
int main(){
    int a,b;
    while(scanf("%d%d",&a,&b)!=EOF){//EOF 是文档的结尾
        int sum=0,sum1=0;
        deal(a,sum);
        deal(b,sum1);
        if(sum==b&&sum1==a)printf("1\n");//&&两个同时都要满足
        else printf("0\n");

    }
return 0;
```

}

2.窗口模拟点击

题目：窗口点击模拟

描述：在计算机屏幕上，有 N 个窗口。窗口的边界上的点也属于该窗口。窗口之间有层次的区别，在多于一个窗口重叠的区域里，只会显示位于顶层的窗口里的内容。当你用鼠标点击某个点的时候，若其在窗口内，你就选择了处于被点击位置所属的最顶层窗口，并且这个窗口就会被移到所有窗口的顶层，而剩余的窗口的层次顺序不变，如果你点击的位置不属于任何窗口计算机就会忽略你这次点击。

编写一个程序模拟点击窗口的过程：先从标准输入读入窗口的个数，窗口编号和位置（以窗口的左上角和右下角的坐标表示，先输入的窗口层次高），然后读入点击的次数和位置（以点击的坐标表示），编写程序求得经过上述点击后的窗口叠放次序。

假设：

屏幕左下角作为 X 轴和 Y 轴坐标原点，即坐标为（0，0），所有输入的坐标数值都是整数，并且都大于等于 0，小于等于 1000。

输出窗口的叠放次序时从最后点击后最顶层的窗口编号开始按层次依次输出；

输入的窗口个数大于 0 并且小于等于 10，点击次数大于 0 并且小于等于 20。

输入：第一行窗口个数 n，接下来 n 行每行一个窗口的编号、左下角坐标、右上角坐标。 接下来一行点击次数 k，接下来 k 行每行一个点击坐标。

输出：一行 n 个数字，表示 K 次点击后按层次排列的窗口编号，空格隔开。 行末空格与文末换行可有可无。

样例：

输入：

```
4
1 43 31 70 56
2 50 24 80 50
3 23 13 63 42
4 57 36 90 52
5
47 28
73 40
68 32
82 43
27 49
```

输出：

```
4 2 3 1
```

```
//注意题目中先输入的层次高是指先收入的在底层
#include<stdio.h>
int main(){//或者用结构体、
    int n,m,i,j,x,y;
    int a[10][10];//存每个窗口的编号和坐标
    int b[10];//依次放窗口叠放次序从顶层到底层的窗口在 a 中的下标
    while(scanf("%d",&n)!=EOF){
        for(i=n-1;i>=0;i--){//输入窗口，a[0]的层次最低（在最顶层）

    scanf("%d%d%d%d%d",&a[i][0],&a[i][1],&a[i][2],&a[i][3],&a[i][4]);
            b[i]=i;//存放窗口编号
        }
        scanf("%d",&m);
        for(i=0;i<m;i++){//m 个输入
            scanf("%d%d",&x,&y);
            for(j=0;j<n;j++){//在输入的窗口之内,j 表示在上面的窗口个数

    if(x>=a[b[j]][1]&&x<=a[b[j]][3]&&y>=a[b[j]][2]&&y<=a[b[j]][4]){//  交
换
                    int temp=b[j];
                    for(int k=j;k>=1;k--){
                        b[k]=b[k-1];
                    }
                    b[0]=temp;
                    break;
                }
```

```
            }

        }
        for(i=0;i<n;i++){
            printf("%d ",a[b[i]][0]);
            if(i==n-1)printf("\n");
        }
    }
return 0;
}
```

3.文章识别

描述：输入一篇可能未经排版的文章，挑选出其中的单词【单词中不包含“(”等特殊符号】，然后按字典序输出。

输入：从文件中读取文章

输出：按字典序输出单词

样例：

输入：

When most kids go to school and study the knowledge, few special kids have made their history. A small boy from America wins the international contest by defeating adult competitors. He becomes the youngest winner. He has the gift and so many parents want to have such a kid. Though we are not that smart, we still can study hard to realize our dreams.

输出：

见代码后面的截图

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

```
#include<algorithm>
using namespace std;
struct wordtype{
    char word[30];
    int count;
}w[100];
int wordcount=0;//记录数组实际存放的单词数（去重）
bool cmp(wordtype a,wordtype b){//结构体的排序
    if (strcmp(a.word,b.word)<0)return true;
    else return false;
}
int find(char temp[]){//查找以前是否有过这个单词
    for(int i=0;i<wordcount;i++){
        if(strcmp(w[i].word,temp)==0){
            return i;
        }
    }
    return -1;
}
int main(){
    char c;
    char temp[30];
    int a=0;
    while(scanf("%c",&c)!=EOF){
        if(isalpha(c)) temp[a++]=c;
        else{
            temp[a]=0;
            if(a==0) continue;//多余的空格，不处理并读入下一个
            else{//上一个单词结束了
                a=0;//不要忘记清 0
                int i=find(temp);
                if (i!=-1){//以前有过此单词
                    w[i].count++;
                }
                else{//建立一个结构体并把它放在结构体数组中
                    wordtype tmp;
                    strcpy(tmp.word,temp);
                    tmp.count=1;
                    w[wordcount]=tmp;
                    wordcount++;
                }
            }
        }
    }
```

```
    sort(w,w+wordcount,cmp);//结构体数组排序，需要实现 cmp 函数
    for(int i=0;i<wordcount;i++){//输出
        printf("%s %d\n",w[i].word,w[i].count);
    }
    return 0;
}
```

2016 机试题目

1.逆序数

描述：给定一个数 n，将这个数的各位顺序颠倒，称为逆序数 m。 例如 1234 的逆序数是 4321。

输入：输入一个数 n, n 开头无多余的 0（0 < n > 1000000000）

输出：如果 m 是 n 的 k 倍（k 为整数），那么输出 n*k=m。 如果 m 不是 n 的整数倍，那么输出 n 和 n 的逆序数。

样例：

输入 1：1204

输出 1：1204 4201

输入 2：1089

输出 2：1089*9=9801

输入 3：23200

输出 3：23200 00232

```
#include<stdio.h>
#include<algorithm>
using namespace std;
int main(){
    char s[31];
```

```
    while(scanf("%s",&s)!=EOF){
        int n=atoi(s);//字符向 int 转换
        reverse(s,s+strlen(s));//用反转函数头文件 algorithm
        int m=atoi(s);
        if(m%n==0) printf("%d*%d=%d\n",n,m/n,m);
        else printf("%d %d\n",n,m);
    }
return 0;
}

//这是纯自己编不借用函数的做法
#include<stdio.h>
#include<string.h>
int main(){
    char a[12];
    int n,m;
    while(scanf("%s", &a)!=EOF){
        int len=strlen(a);
        n=m=0;
        int i, tmp=1;
        for(i=0;i<len;i++){
            n=n*tmp+a[i]-'0';
            m=m*tmp+a[len-i-1]-'0';
            tmp=tmp*10;
        }
        if(n!=0 && m%n==0){
            printf("%d*%d=%d\n", n, m/n, m);
        }else{
            printf("%s ", a);
            for(i=len-1;i>=0;i--){
                printf("%c", a[i]);
            }
            printf("\n");
        }
    }
    return 0;
}
```

2.enum 定义语句字符串解析

描述：给一个 c 语言的 enum 定义语句，输出 enum 中规定的各项及其对应的数

值。

输入 1：enum BOOL{true,false};

输出 1：true 0 false 1

输入 2：

```
enum:date{JAN=1,FEB,MAR,APR,MAY,JUN,JULY,AUG,SEP,OCT,NOV,DEC,MON=1,TUE,
WED,THU,FRI,SAT,SUN,found=1949};
```

输出 2：

```
JAN 1
FEB 2
MAR 3
APR 4
MAY 5
JUN 6
JULY 7
AUG 8
SEP 9
OCT 10
NOV 11
DEC 12
MON 1
TUE 2
WED 3
THU 4
FRI 5
SAT 6
SUN 7
found=1949
```

```
#include<stdio.h>
#include<string.h>
#include<cctype>
#include<algorithm>
using namespace std;
int main(){
    char a[100];//存放字符串
    while(gets(a)!=NULL){
        char word[10];//存放当前解析的单词
```

```
int i,flag=0;//flag 标志已经读取到‘{‘之后
int wordindex=0;//当前单词的下标
char num[5];//存‘=’之后的数值
int numindex=0;//存放数值的下标
int n=0;
int len=strlen(a);
for(i=0;i<len;i++){
    if(a[i]=='{'){
        flag=1;
    }
    else if(flag==0) continue;//‘{’之前的不计入单词
    else {
        if (isalpha(a[i])){
            word[wordindex++]=a[i];
        }
        else if(a[i]=='='){
            word[wordindex]='\0';//单词结束
            wordindex=0;
            printf("%s ",word);   //打印单词
            i++;
            while(isdigit(a[i])==0){//空格等
                i++;
            }
            while(isdigit(a[i])){//数字放入字符数组
                num[numindex++]=a[i++];
            }
            num[numindex]='\0';//数字结束
            numindex=0;
            n=atoi(num);//转成 int
            printf("%d\n",n);
            while(a[i]!=',')i++;//=之后的，特殊处理
            n++;

        }
        else if(a[i]==','){

            word[wordindex]='\0';//单词结束
            wordindex=0;
            printf("%s ",word);   //打印单词
            printf("%d\n",n);
            n++;
            while(isalpha(a[i+1])==0){//空格等
                i++;
            }
```

```
                }
                else if(a[i]=='}'){

                    word[wordindex]='\0';//单词结束
                    wordindex=0;
                    printf("%s ",word);   //打印单词
                    printf("%d\n",n);
                }
                //else 空格等无用字符不作处理继续处理下一个字符
            }
        }
    }
    return 0;
}
```

2017 机试题目

1.查找中位数

先输入一个整形数字 N，接着输入 N 个无序的数字。要求输出升序排列后的中位数，以及该中位数输入的次序。如果 N 为偶数，则输出有二个中位数，如果 N 为奇数，输出最中间的数即可。

样例 1：

输入：5

9 2 7 1 6

输出：6 5

样例 2：

输入：6

9 6 7 1 2 3

输出：3 6

6 2

```
#include<stdio.h>
#include<algorithm>
using namespace std;
struct num{//结构体
    int value;
    int index;
}a[200];
bool cmp(num a,num b){//cmp 函数
    if(a.value<=b.value)return 1;
    else return 0;
}
int main(){
    int n;
    while(scanf("%d",&n)!=EOF){
        for(int i=0;i<n;i++){
            scanf("%d",&a[i].value);
            a[i].index=i+1;
        }
        sort(a,a+n,cmp);//使用排序函数
        int m=n/2;
        if(n%2==0){//偶数
            printf("%d %d\n",a[m-1].value,a[m-1].index);
            printf("%d %d\n",a[m].value,a[m].index);
        }
        else printf("%d %d\n",a[m].value,a[m].index);
    }
return 0;
}

//当不使用 sort 函数，自己写排序函数
#include<stdio.h>
struct num{
    int value;
    int index;
}a[200];
void sort(int n){
    for(int i=0;i<n;i++){//冒泡
        for(int j=0;j<n-i-1;j++){
            num temp;
            if(a[j].value>a[j+1].value){
                temp=a[j+1];
                a[j+1]=a[j];
```

```
                a[j]=temp;
            }
        }
    }
}
int main(){
    int n;
    while(scanf("%d",&n)!=EOF){
        for(int i=0;i<n;i++){
            scanf("%d",&a[i].value);
            a[i].index=i+1;
        }
        //sort(a,a+n,cmp);
        sort(n);
        int m=n/2;
        if(n%2==0){//偶数
            printf("%d %d\n",a[m-1].value,a[m-1].index);
            printf("%d %d\n",a[m].value,a[m].index);
        }
        else printf("%d %d\n",a[m].value,a[m].index);
    }
return 0;
}
```

2.词法分析，查找未定义变量

输入两个 C 语言语句，第一句为正常的 C 语言变量定义语句，符合 C 语言语法要求，变量间可以有多个空格，包含数组，指针定义等。第二句为变量运算语句，要求输出第二个 C 语言语句中未定义的变量。

样例：

输入：

```
int x12, y=1, num_stu=89, a[30], p; int x12, y=1;
Sum=num+x12y;
```

输出：

```
Sum num
```

```
#include "stdafx.h"
#include <iostream>
#include <string>
#include "math.h"
#include "stdio.h"
#include "string.h"
#include <vector>
#include <queue>
#include <map>
#include <algorithm>

using namespace std;
int max(int x,int y){
    return x>y?x:y;
}

int nextvar(string str,int &curindex,string &nextstr){
    if(curindex>=str.size()-1) return 0;
    int startindex,endindex;
    int delindex;
    while(str[curindex]=='*'||str[curindex]==' ')  curindex++;      //è
￥3y±?á????º?àóàμ?????oí*
    startindex=curindex;
    endindex=str.find(",",startindex);
    if(endindex==string::npos) endindex=str.find(";",startindex);

    curindex=endindex+1;
    if(!(endindex>startindex)) return 0;
    nextstr=str.substr(startindex,endindex-startindex);

    delindex=nextstr.find("[");
    if(delindex!=string::npos){
        nextstr.erase(delindex,nextstr.size());
    }
    delindex=nextstr.find("=");
    if(delindex!=string::npos){
        nextstr.erase(delindex,nextstr.size());
    }
    if(!nextstr.empty()) return 1;
    else return 0;
}
```

```
int nextvar2(string str,int &curindex,string &nextstr){
    if(curindex>=str.size()-1) return 0;
    int startindex,endindex;
    while(str[curindex]=='*'||str[curindex]==' ')  curindex++;      //è
￥3y±?á????°?àóàμ?????oí*
    startindex=curindex;
    for(int i=startindex;i<str.size();i++){
        if(!( ((str[i]>='a')&&(str[i]<='z'))
              ||((str[i]>='A')&&(str[i]<='Z'))
              ||((str[i]>='0')&&(str[i]<='9'))
              ||(str[i]=='_')))
        break;
    }
    endindex=i;
    curindex=endindex;
    if(str[curindex]=='[') {
        curindex++;
        while(str[curindex]!=']') curindex++;
    }
    if(str[curindex]==']') curindex++;
    curindex++;
    while((str[curindex]>='0')&&(str[curindex]<='9')) curindex++;

    //cout<<curindex<<" "<<str[curindex]<<endl;
    if(!(endindex>startindex)) return 0;
    nextstr=str.substr(startindex,endindex-startindex);
    if(!nextstr.empty()) return 1;
    else return 0;
}
int main(int argc, char* argv[])
{
    string str1,str2,nextstr;
    char s[100];
    vector<string> var;

    gets(s);
    str1=s;
    gets(s);
    str2=s;
    int length=str1.size();
    int curindex=0;
    while(str1[curindex]==' '){
        str1.erase(curindex,curindex+1);
    }
```

```
    curindex=str1.find(" ");
    while(nextvar(str1,curindex,nextstr)){
        var.push_back(nextstr);
    }

    curindex=0;
    while(str2[curindex]==' '){
        str2.erase(curindex,curindex+1);
    }
    while(nextvar2(str2,curindex,nextstr)){
        int result=0;
        for(int i=0;i<var.size();i++){
            if(var[i]==nextstr) {
                result=1;
                break;
            }
        }

        if (result==0) cout<<nextstr<<" ";
    }


    //for(vector<string>::iterator
iter=var.begin();iter!=var.end();iter++)
    //  cout<<*iter<<endl;

    return 0;
}
```

3.找家谱成员

输入若干行，每一行的第一个输入为家谱中的某成员，该行接着输入的信息为每个孩子姓名。最后一行的输入为要求查找的两个家谱成员的关系。要求：根据输入的家谱成员信息，建立二叉树家谱关系图，并输出二位待查找成员在家谱中的关系，包括输出他们最邻近的共同祖先以及在家谱中相差的层次数。

样例：

输入：

YE SHU MEI

SHU GE MEI1

BA SELF MEI2

GE SON1 SON2

SON2 MEI1

输出：

SHU 1

//代码是参考别人的，也有不严谨的地方

#include<stdio.h>

#include<string.h>

const int namesize=20;

struct treenode{

char name[namesize];

char parent[namesize];

int gradation;

```
    treenode* leftchild;
    treenode* rightchild;
};
 treenode* findtreenode(treenode* root,char parent[]);
int getname(char str[],char parent[],char leftc[],char rightc[] ){
    int len=strlen(str);
    int index=0;
    int flag=0;//标记输入的格式，输入了两个还是三个名字。
```

```
    for(int i=0;i<len;){
        while(str[i]==' ') i++;
        while(str[i]!=' ')
            parent[index++]=str[i++];
        parent[index]='\0';//勿忘！

        while(str[i]==' ') i++;
        index=0;
        while(str[i]!=' '&&i<len)
            leftc[index++]=str[i++];
        leftc[index]='\0';

        while(str[i]==' ') i++;
        index=0;
        while(str[i]!=' '&&i<len){
            rightc[index++]=str[i++];
            flag=1;
        }
        rightc[index]='\0';
        while(str[i]==' ') i++;
    }
    return flag;
}

treenode*  addtreenode(treenode*  root,char  parent[],char  leftc[],char
rightc[]){
    if(root==NULL){//根节点
        root=new treenode;
        strcpy(root->name,parent);
        root->parent[0]='\0';
        root->gradation=0;
        root->leftchild=NULL;
        root->rightchild=NULL;
    }

    treenode* temp=findtreenode(root,parent);//调用函数，根据结点双亲名寻找
结点 temp；

    treenode *lc=new treenode;//创建左孩子
    strcpy(lc->name,leftc);
    strcpy(lc->parent,parent);
    lc->gradation=temp->gradation+1;//父节点的层数+1
    lc->leftchild=NULL;
    lc->rightchild=NULL;
```

```
    treenode *rc=new treenode;//创建右孩子
    strcpy(rc->name,rightc);
    strcpy(rc->parent,parent);
    rc->gradation=temp->gradation+1;
    rc->leftchild=NULL;
    rc->rightchild=NULL;

    temp->leftchild=lc;//连接
    temp->rightchild=rc;

    return root;//返回根节点
}

treenode* findtreenode(treenode* root,char parent[]){
    if(root==NULL) return NULL;
    if(strcmp((root->name),parent)==0) return root;
    else{
        treenode* result =findtreenode(root->leftchild,parent);
        if(result==NULL)
            result=findtreenode(root->rightchild,parent);
        return result;
    }
}
void findrelation(treenode* root,char person1[],char person2[]){
    treenode* position1=findtreenode(root,person1);
    treenode* position2=findtreenode(root,person2);
    treenode* min,*max;
    if(position1->gradation>position2->gradation){//第一个位置的深度大
        min=position1;max=position2;}
    else {
        min=position2;max=position1;
    }
    int level=min->gradation-max->gradation;//层数差
    while(min->gradation>max->gradation){//层数高的晚辈向上爬，直到层数相同
        char parent[namesize];
        strcpy(parent,min->parent);
        min=findtreenode(root,parent);
    }
    while(strcmp(min->parent,max->parent)!=0){
        char parent[namesize];
        strcpy(parent,min->parent);
        min=findtreenode(root,parent);//寻找根节点
        strcpy(parent,max->parent);
```

```
        max=findtreenode(root,parent);
    }
    if(min->gradation==0)
        printf("没有共同祖先");
    else
        printf("%s %d\n",min->parent,level);
}
int main(){
treenode* root=NULL;
char parent[namesize],leftc[namesize],rightc[namesize];
    char str[namesize*3];//存放一行字符串
while(gets(str)!=0){
    parent[0]=leftc[0]=rightc[0]='\0';
    int flag;
    flag=getname(str,parent,leftc,rightc);//调用函数
    if(flag==1)
        root=addtreenode(root,parent,leftc,rightc);//调用函数向树中添加结
点
    if(flag==0){//输出关系
        findrelation(root,parent,leftc);
    }
}

return 0;
}
```

(6 条消息) 北航机试题 2017（题目+代码）_mingzhiqing 的博客-CSDN 博客

2018 机试题目

1.最长折线段

在直角坐标系中有若干线段，有的线段会和其他线段的某一段点重合，即某一端点的坐标相同，所以这些线段会形成含两条或者两条以上线段的折线。求若干折线中，含有线段条数最多的折线中包含的线段数目，并输出该折线最左端的坐标。

输入

9

1 3 3 1

2 3 3 4

3 4 6 3

3 1 5 2

4 2 5 2

4 2 5 4

7 3 8 4

8 4 9 0

9 0 10 0

输出

4: (1,3) (3,1)

我猜是这样输出，具体是什么格式我也不知道

```
#include <iostream>
#include <cstdio>
#include <algorithm>
using namespace std;
int N;
struct segment{
    int srcx, dstx; // x 坐标
```

```
    int srcy, dsty; // y 坐标
    int segCnt; // 折线的最大联通条数
    bool updatesegCnt; // 更新标记，dfs 后会得到联通的线段条数，然后再更新线段条数
};
bool isConnect(const segment &a, const segment &b) { // 两条线段是否联通
    bool ret = false;
    if (a.dstx == b.srcx && a.dsty == b.srcy) ret = true;
    else if (a.dstx == b.dstx && a.dsty == b.dsty) ret = true;
    else if (a.srcx == b.srcx && a.srcy == b.srcy) ret = true;
    else if (a.srcx == b.dstx && a.srcy == b.dsty) ret = true;
    return ret;
}
void dfs(segment seg[], bool visit[], int index, int &cnt) { // 深搜遍历联通的线段
    cnt++;
    visit[index] = true;
    for (int i = 0; i < N; i++) {
        if (visit[i] == false && isConnect(seg[index], seg[i])) {
            dfs(seg, visit, i, cnt);
        }
    }
}
```

```
void updateSegCnt(segment seg[], bool visit[], const int &cnt) { // 更新线段条数
    for (int i = 0; i < N; i++) {
        if (visit[i] == true && seg[i].updatesegCnt == false) { // visit 为 true 说明访问过，updatesegCnt 为 false 说明是刚刚访问过的
            seg[i].updatesegCnt = true;
            seg[i].segCnt = cnt;
        }
    }
}
bool segCmp(const segment &a, const segment &b) { // 比较函数
    if (a.segCnt == b.segCnt) {
        return a.srcx < b.srcx;
    }
    else return a.segCnt > b.segCnt;
}
int main()
{
    cin >> N;
    segment *seg = new segment[N]; // n 条线段
    bool *visit = new bool[N]; // 访问标记
    for (int i = 0; i < N; i++) {
        cin >> seg[i].srcx >> seg[i].srcy >> seg[i].dstx >> seg[i].dsty;
        seg[i].segCnt = 1;
        visit[i] = false;
        seg[i].updatesegCnt = false;
    }
    for (int i = 0; i < N; i++) {
        if (visit[i] == false) {
            int cnt = 0;
            dfs(seg, visit, i, cnt);
            updateSegCnt(seg, visit, cnt);
        }
    }
    sort(seg, seg+N, segCmp);
    /*
    for (int i = 0; i < N; i++) {
        printf("%d:  (%d,%d)  (%d,%d)\n",  seg[i].segCnt,  seg[i].srcx, seg[i].srcy, seg[i].dstx, seg[i].dsty);
    }
    */
    putchar('\n');
    printf("%d:   (%d,%d)   (%d,%d)\n",   seg[0].segCnt,   seg[0].srcx, seg[0].srcy, seg[0].dstx, seg[0].dsty);
```

```
    return 0;
}
```

2.三叉树分支最多的节点高度

一个三叉树，第一行输入结点个数，其余各行输入每个结点的序号和三个孩子的序号，除第一个结点外，其他组的数据第一个就是父结点，后三个数是孩子（孩子没有先后顺序）。之后计算分支最多的结点的高度，输出其编号，高度相同，按前序输出第一个。

孩子为 0 表示没有这个孩子

```
#include <iostream>
#include <queue>
using namespace std;
struct Node { //节点
    int id; // 节点 id
    Node *left; // 左中右儿子
    Node *middle;
    Node *right;
    int layer; // 第 layer 层
    int preOrderPriority; // 前序访问的次序（没用到）
    int childCnt; // 儿子的个数
    Node (int id) : id(id) {
        left = middle = right = NULL;
        layer = 1; // 单纯的一个节点的层次定义为 1
        preOrderPriority = 0;
        childCnt = 0;
    }
```

```
};
class Tree {
protected:
    Node *root; // 根
    void rprint(Node *r, int h) { // 递归打印树，肉眼检查错误
        for (int i = 0; i < h; i++) cout << "   ";
        if (r == NULL) {
            cout << "[/]" << endl;
        }
        else {
            cout << r->id << "(" << r->childCnt << ")" << r->layer << endl;
            rprint(r->left, h+1);
            rprint(r->middle, h+1);
            rprint(r->right, h+1);
        }
        return;
    }
    // 前序遍历得到满足题意的节点
    void preOrderGetNode(Node *r, Node *&tmp) { // 因为要改变 tmp 的值所以
这里需要的是引用（tmp 是引用，引用的类型是 Node*）
        if (r == NULL) return;
        if (r->childCnt > tmp->childCnt) { // 首先找儿子最多的
            tmp = r;
        }
        else if (r->childCnt == tmp->childCnt) { // 如果儿子个数相同
            if (r->layer > tmp->layer) { // 找层次最大的
                tmp = r;
            }
            // 如果层次也相同，不用管了，因为是按照前序遍历访问的
        }
        preOrderGetNode(r->left, tmp);
        preOrderGetNode(r->middle, tmp);
        preOrderGetNode(r->right, tmp);
        return;
    }
public:
    Tree() {
        root = NULL;
    }
    void insertNode(int parent, int l, int m, int r) { // 插入节点
        Node *tmp;
        if (root == NULL) tmp = new Node(parent); // 插入的节点是根
        else  {
            tmp = findById(parent); // 不是根找父亲
```

```
            if (tmp == NULL || noChild(tmp) == false) return; // 没找到或者找到的父亲已经有儿子了
        }
        // l m r 如果为 0 说明没有这个孩子
        // 如果有孩子，把孩子插上，父亲的孩子数量加一，孩子的层次比父亲多一
        if (l != 0) {
            tmp->left = new Node(l);
            tmp->childCnt++;
            tmp->left->layer += tmp->layer;
        }
        if (m != 0) {
            tmp->middle = new Node(m);
            tmp->childCnt++;
            tmp->middle->layer += tmp->layer;
        }
        if (r != 0) {
            tmp->right = new Node(r);
            tmp->childCnt++;
            tmp->right->layer += tmp->layer;
        }
        if (root == NULL) root = tmp;
    }
    Node* findById(int id) { // 按照 id 查找
        Node* r = root;
        queue<Node*> q; // 用队列一层一层遍历
        if (root != NULL) {
            q.push(r);
        }
        while (q.empty() == false) {
            Node *tmp = q.front();
            q.pop();
            if (tmp->id == id) return tmp; // 找到了就返回节点指针
            if (tmp->left != NULL) q.push(tmp->left);
            if (tmp->middle != NULL) q.push(tmp->middle);
            if (tmp->right != NULL) q.push(tmp->right);
        }
        return NULL; // 没找到
    }
    bool noChild(Node *t) { // 检查是否有孩子
        bool ret;
        if (t->childCnt == 0) ret = true;
        else ret = false;
        return ret;
    }
```

```
    void printIdSatisfyQuestion() { // 打印满足题目的结果
        Node *tmp = root;
        preOrderGetNode(root, tmp); // 前序递归寻找满足的节点
        cout << tmp->id << endl;
    }
    void print() {
        rprint(root, 0);
    }
};
int main()
{
    int N;
    Tree t;
    cin >> N;
    int id, l, m, r;
    for (int i = 0; i < N; i++) {
        cin >> id >> l >> m >> r;
        t.insertNode(id, l, m, r);
    }
    t.printIdSatisfyQuestion();
    t.print();
    return 0;
}
/*
孩子个数 > 层数 > 前序顺序
4
1 0 3 4
3 6 7 8
4 2 9 5
9 0 11 10

5
1 0 3 4
3 6 7 8
4 2 9 5
9 0 11 10
5 12 13 14

*/
```

(6 条消息) 北航 2018 年机试_我在浪里-CSDN 博客

(6 条消息) 2018 北航计算机学院上机复试题一_dadan 的博客-CSDN 博客

2019 机试题目

前言

大家好，我是 2019 年北航考研的考生，在 3 月 18 号参加了北航的机考，特此在这个 根据自己的回忆给出题目描述，一个测试用例，样例输出，以及本人编写的参考代码, 以及思路讲解。如果我可以得到题目的原本描述, 会更新给大家的。还有做几点说明，北航上机考试建议使用标准 c 编程，解释一下标准 c 编程，不能使用基于特定平台的库，例如 windows.h，linux 系统下的特定库，某个厂家自己开发出的库，这些都是编译不通过的。其他的库都是可以使用的。还有可以使用标准 c++编程，可以使用 STL(例如 vector,set,queue,stack,map 等等)，可以使用<algorithm>库（例如 sort（）函数），可以使用 c++的语言特性，例如重载，动态联编等等。(这个想说，有些学长学姐可能会说不能使用 c++，特此声明，完全可以，像今年和去年的第二道题目，都比较难，涉及三叉树，用 c++的库会方便不少)。

1.素数等差数列

题目一 : 给定闭区间[a,b] ,要求输出 素数的等差序列,三个以上才算是序列，例如 [100,200] 会输出 151 157 163 和 167 173 179

再例如输入[1,100] 会有两个等差序列，3 5 7 和 47 53 59。输出样式行末的空格保留。

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
```

```
int isPrime(int n)
{
    if(n==1) return 0;
    else if(n<5) return n<4;
    else
    {
        if(n%6!=5&&n%6!=1) return 0;
        int len = sqrt(n);
        for(int i=2; i<=len+1; i++)
            if(n%i==0) return 0;
        return 1;
    }
}
int getNextPrime(int n)
{
    while(1)
    {
         if(isPrime(n+1)) return n+1;
        else n++;
    }

}


int main()
{
    int a,b;
    scanf("%d %d",&a,&b);
    int first,second,tmp,dif;

    int x = a-1;
    while(x<=b)
    {
        first = getNextPrime(x);
        x = first;
        if(x>b) break;
        second = getNextPrime(x);
        x = second;
        if(x>b) break;
        dif = second - first;
        tmp = getNextPrime(x);
        x = tmp;

        if(x>b) break;
```

```
        if(tmp-second==dif)
        {
            printf("%d %d %d ",first,second,tmp);
            int tmp2 = second;
            while(tmp-tmp2==dif)
            {
                    tmp2 = tmp;
                    tmp = getNextPrime(x);
                    x = tmp;
                    if(x>b) break;
                   if(tmp-tmp2==dif) printf("%d ",tmp);
                    else x = tmp2;
            }
            printf("\n");
        }else{
            x = first;
        }
    }

}
```

样例输入：141 400

样例输出：151 157 163

167 173 179

199 211 223

251 257 263 269

367 373 379

题目解析：比较中规中矩的算法题，第一点是判断素数的函数，第二点是就是如何控制满足题目描述，并且输出正确的格式，我觉得唯一注意的问题就是注意判断 1 不是素数。

2 三叉树最短路径

题目描述：

一个关于三叉树的题目，小于 100 的值代表树叶，大于 100 的值为分支点，建树的过程是水平方向建树。

输入格式：

先输入 n，代表有 n 组数据，接下来 n 行，输入 4 个数，第一个数代表根节点，接下来分别代表三个子节点，-1 代表子节点不存在，输入的顺序按照层次遍历的次序。接下来，要求寻找叶子节点的最短路径，最短路径是指不经过重复的边。

输入方式，首先输入一个值 m，代表 m 行，接下来 m 行输入 m 个叶子节点和对应的优先级，要求按优先级输出从上次到达的位置到该节点的最短路径，每条路径的最后一个节点要求输出目标叶子节点，最后要求回到根节点。

样例：

输入：

```
10
100 101 108 107
101 1 102 2
108 103 104 105
107 17 109 18
102 3 4 5
103 7 8 9
104 10 106 11
105 15 16 -1
109 19 20 21
106 12 13 14
5
8 1
14 3
16 2
5 0
19 4**
```

输出：

```
100 101 102 5
102 101 100 108 103 8
103 108 105 16
```

```
105 108 104 106 14
106 104 108 100 107 109 19
109 107 100
```

将 lca 问题转为 rmq 问题，找到最近公共祖先， 再用 dfs 找到最近公共祖先和子节点的路径，输出路径。代码如下：

```
#include<cstdio>
#include<vector>
#include<cmath>
#include<algorithm>
using namespace std;
const int maxn = 10010;
int seq[maxn], dep[maxn], pos[maxn] = {0}, hashTable[maxn] = {false};
int N = 0, c = 1;
struct node {
    int isroot;
    vector<int> child;
}Node[maxn];

void DFS(int index, int depth) {
    seq[c] = index;
    dep[c] = depth;
    if(!pos[index]) {
        pos[index] = c;
    }
    c++;
    for(int i = 0; i < Node[index].child.size(); i++) {
        if(!pos[Node[index].child[i]]) {
            DFS(Node[index].child[i], depth + 1);
        }
        seq[c] = index;
        dep[c] = depth;
        c++;
    }
}

int dp[maxn][maxn];
void init() {
    for(int i = 1; i <= N; i++) {
        dp[i][0] = dep[i];
    }
```

```
    for(int j = 1; (1<<j) <= N; j++) {
        for(int i = 1; i + (1<<j) - 1 <= N; i++) {
            dp[i][j] = min(dp[i][j - 1], dp[i + (1<<(j - 1))][j - 1]);
        }
    }
}

int RMQ(int a, int b) {
    a = pos[a], b = pos[b];
    if(a > b) swap(a, b);
    int k = log(b - a + 1) / log(2);
    int mind =  min(dp[a][k], dp[b - (1<<k) + 1][k]);
    for(int i = a; i <= b; i++) {
        if(dep[i] == mind) {
            return i;
        }
    }
}

int t[maxn], path[maxn], len;

void getpath(int index, int numNode, int e) {
    if(Node[index].child.size() == 0) {
        if(index == e) {
            for(int i = 1; i < numNode; i++) {
                path[i] = t[i];
            }
            len = numNode;
            return;
        } else {
            return;
        }
    }
    for(int i = 0; i < Node[index].child.size(); i++) {
        int child = Node[index].child[i];
        t[numNode] = child;
        getpath(child, numNode + 1, e);
    }
}

void printpath(int a, int b) {
    int minfa = seq[RMQ(a, b)];
    path[0] = minfa;
    getpath(minfa, 1, a);
```

```
    for(int j = len - 2; j >= 0; j--) {
        printf("%d ", path[j]);
    }
    getpath(minfa, 1, b);
    for(int j = 1; j < len; j++) {
        printf("%d", path[j]);
        if(j < len) printf(" ");
    }
    printf("\n");
}

int main() {
    for(int i = 0; i < maxn; i++) {
        Node[i].isroot = 0;
    }
    int n, root;
    scanf("%d", &n);
    int id, child;
    for(int i = 0; i < n; i++) {
        scanf("%d", &id);
        Node[id].isroot++;
        if(hashTable[id] == false) {
            N++;
            hashTable[id] = true;
        }
        for(int j = 0; j < 3; j++) {
            scanf("%d", &child);
            if(hashTable[child] == false && child != -1) {
                N++;
                hashTable[child] = true;
            }
            if(child != -1) {
                Node[child].isroot--;
                Node[id].child.push_back(child);
            }
        }
    }
    for(int i = 0; i < maxn; i++) {
        if(Node[i].isroot > 0) {
            root = i;
            break;
        }
    }
    N = 2 * N - 1;
```

```
    DFS(root, 0);
    init();
    int m;
    scanf("%d", &m);
    int z[m];
    int cixu, zhi;
    for(int i = 0; i < m; i++) {
        scanf("%d", &zhi);
        scanf("%d", &cixu);
        z[cixu] = zhi;
    }
    path[0] = root;
    getpath(root, 1, z[0]);
    for(int i = 0; i < len; i++) {
        printf("%d", path[i]);
        if(i < len - 1) printf(" ");
    }
    printf("\n");
    for(int i = 0; i < m - 1; i++) {
        printpath(z[i], z[i + 1]);
    }
    path[0] = root;
    getpath(root, 1, z[m - 1]);
    for(int i = len - 2; i >= 0; i--) {
        printf("%d", path[i]);
        if(i > 0) printf(" ");
    }
    printf("\n");
    return 0;
}
```

(6 条消息) 2019 北航机考第二题题目描述解析以及参考代码（紧接第一题）

_sinat_38425013 的博客-CSDN 博客

思路解析：

第一步肯定是建树，声明数据结构，利用队列建树

```
#include<iostream>
#include<stdio.h>
#include<queue>
using namespace std;

struct triTree{
    int data;
    struct triTree *lchild;
    struct triTree *mchild;
    struct triTree *rchild;
    struct triTree *parent;
    triTree()
    {
        lchild = NULL;
        mchild = NULL;
        rchild = NULL;
        parent = NULL;
    }
    triTree(int x)
    {
        data = x;
        lchild = NULL;
        mchild = NULL;
        rchild = NULL;
        parent = NULL;
    }

};
queue<triTree*> q;
triTree* createTree(triTree* root,int rt,int a,int b,int c)
{
    if(root->parent==NULL)
    {
        root->data= rt;
    }
    if(a!=-1){
        triTree *newNode = new triTree(a);
        root->lchild = newNode;
        newNode->parent = root;
```

```
        if(newNode->data>=100) q.push(newNode);
    }else root->lchild = NULL;
    if(b!=-1){
         triTree *newNode = new triTree(b);
        root->mchild = newNode;
        newNode->parent = root;
        if(newNode->data>=100) q.push(newNode);
    }else root->mchild = NULL;
    if(c!=-1){
         triTree *newNode = new triTree(c);
        root->rchild = newNode;
        newNode->parent = root;
        if(newNode->data>=100) q.push(newNode);
    }else root->rchild = NULL;
    if(!q.empty()){
        triTree* tmp = q.front();
        q.pop();
        return tmp;
    }else return NULL;

}
//void pre_test(triTree* test)
//{
//    if(test==NULL) return;
//    cout<<test->data<<" ";
//    pre_test(test->lchild);
//    pre_test(test->mchild);
//    pre_test(test->rchild);
//}
int main()
{
    freopen("in.txt","r",stdin);
    int n;
    cin>>n;
    int rt,a,b,c;
    struct triTree* root = new triTree;
    struct triTree* tmp = root;
    for(int i=0;i<n;i++)
    {
        cin>>rt>>a>>b>>c;
        tmp = createTree(tmp,rt,a,b,c);
    }
    //pre_test(root);
```

}

第二步处理下一组输入，并且排序

```
struct target{
    int target;
    int priority;
    bool operator<( const struct target& t)const
    {
        return priority<t.priority;
    }
};
    int m;
    cin>>m;
    target  t[m];
    for(int i=0;i<m;i++)
    {
        cin>>t[i].target>>t[i].priority;
    }
    sort(t,t+m);
```

第三步，需要四个函数，需要从输入的节点值获取在树中的节点地址（指针），

根据两个节点后序遍历求最近的公共父节点，

从公共节点出发前序遍历找到目标路径的节点和打印函数。

```
struct triTree* search_node(triTree* root,int x)
{
    if(root==NULL||root->data==x) return root;
    struct triTree* tmp = search_node(root->lchild,x);
    if(tmp!=NULL) return tmp;
    else tmp = search_node(root->mchild,x);
    if(tmp!=NULL) return tmp;
     else tmp = search_node(root->rchild,x);
    if(tmp!=NULL) return tmp;
    else return NULL;




}
struct  triTree*  getNearestRoot(triTree*  root,triTree*  node1,triTree
*node2)
{
```

```
    if(root==NULL||root==node1||root==node2) return root;
    triTree* left = getNearestRoot(root->lchild,node1,node2);
    triTree* middle = getNearestRoot(root->mchild,node1,node2);
    triTree* right = getNearestRoot(root->rchild,node1,node2);
    if(left!=NULL&&middle!=NULL) return root;
    if(left!=NULL&&right!=NULL) return root;
    if(middle!=NULL&&right!=NULL) return root;
    if(left!=NULL) return left;
    if(middle!=NULL) return middle;
    if(right!=NULL) return right;
    return NULL;
}
vector<int> result;
triTree* preOrder(triTree* root,int x)
{

    if(root==NULL||root->data==x) return root;
    result.push_back(root->data);
    triTree* tmp = preOrder(root->lchild,x);
    if(root->lchild==NULL);
    else if(tmp==NULL||tmp->data!=x) result.pop_back();
    else return tmp;
    tmp = preOrder(root->mchild,x);
    if(root->mchild==NULL);
    else if(tmp==NULL||tmp->data!=x) result.pop_back();
    else return tmp;
    tmp = preOrder(root->rchild,x);
    if(root->rchild==NULL);
    else if(tmp==NULL||tmp->data!=x) result.pop_back();
    else return tmp;

}

void print()
{
    int len = result.size();
    for(int i=0;i<len;i++)
    {
        cout<<result[i]<<" ";
    }
}
```

第四步：按照输出格式进行输出即可

```
triTree *tmp1 = root;
```

```
    triTree *tmp2,*troot;
    triTree *last = root;
    for(int i=0;i<m;i++)
    {
        result.clear();


        tmp2 = search_node(root,t[i].target);
        troot = getNearestRoot(root,tmp1,tmp2);
        while(last!=troot)
        {
             result.push_back(last->data);
             last = last->parent;
        }
         preOrder(troot,t[i].target);
        print();
        cout<<tmp2->data<<" "<<endl;
        tmp1 = tmp2;
        last = search_node(root,result.back());
    }
    while(last!=NULL)
    {
        cout<<last->data<<" ";
        last = last->parent;
    }
    cout<<endl;
```

这里给出全部代码（注释掉的代码是为了调试，无意义）

```
#include<iostream>
#include<stdio.h>
#include<queue>
#include<algorithm>
#include<vector>
using namespace std;

struct triTree{
    int data;
    struct triTree *lchild;
    struct triTree *mchild;
    struct triTree *rchild;
    struct triTree *parent;
    triTree()
    {
        lchild = NULL;
```

```
        mchild = NULL;
        rchild = NULL;
        parent = NULL;
    }
    triTree(int x)
    {
        data = x;
        lchild = NULL;
        mchild = NULL;
        rchild = NULL;
        parent = NULL;
    }

};
queue<triTree*> q;
vector<int> result;
triTree* createTree(triTree* root,int rt,int a,int b,int c)
{
    if(root->parent==NULL)
    {
        root->data= rt;
    }
    if(a!=-1){
        triTree *newNode = new triTree(a);
        root->lchild = newNode;
        newNode->parent = root;
        if(newNode->data>=100) q.push(newNode);
    }else root->lchild = NULL;
    if(b!=-1){
         triTree *newNode = new triTree(b);
        root->mchild = newNode;
        newNode->parent = root;
        if(newNode->data>=100) q.push(newNode);
    }else root->mchild = NULL;
    if(c!=-1){
         triTree *newNode = new triTree(c);
        root->rchild = newNode;
        newNode->parent = root;
        if(newNode->data>=100) q.push(newNode);
    }else root->rchild = NULL;
    if(!q.empty()){
        triTree* tmp = q.front();
        q.pop();
        return tmp;
```

```
    }else return NULL;

}
//void pre_test(triTree* test)
//{
//    if(test==NULL) return;
//    cout<<test->data<<" ";
//    pre_test(test->lchild);
//    pre_test(test->mchild);
//    pre_test(test->rchild);
//}
struct target{
    int target;
    int priority;
    bool operator<( const struct target& t)const
    {
        return priority<t.priority;
    }
};
struct triTree* search_node(triTree* root,int x)
{
    if(root==NULL||root->data==x) return root;
    struct triTree* tmp = search_node(root->lchild,x);
    if(tmp!=NULL) return tmp;
    else tmp = search_node(root->mchild,x);
    if(tmp!=NULL) return tmp;
    else tmp = search_node(root->rchild,x);
    if(tmp!=NULL) return tmp;
    else return NULL;


}
struct  triTree*  getNearestRoot(triTree*  root,triTree*  node1,triTree
*node2)
{
    if(root==NULL||root==node1||root==node2) return root;
    triTree* left = getNearestRoot(root->lchild,node1,node2);
    triTree* middle = getNearestRoot(root->mchild,node1,node2);
    triTree* right = getNearestRoot(root->rchild,node1,node2);
    if(left!=NULL&&middle!=NULL) return root;
    if(left!=NULL&&right!=NULL) return root;
    if(middle!=NULL&&right!=NULL) return root;
    if(left!=NULL) return left;
    if(middle!=NULL) return middle;
```

```
    if(right!=NULL) return right;
    return NULL;
}
void print()
{
    int len = result.size();
    for(int i=0;i<len;i++)
    {
        cout<<result[i]<<" ";
    }
}
triTree* preOrder(triTree* root,int x)
{

    if(root==NULL||root->data==x) return root;
    result.push_back(root->data);
    triTree* tmp = preOrder(root->lchild,x);
    if(root->lchild==NULL);
    else if(tmp==NULL||tmp->data!=x) result.pop_back();
    else return tmp;
    tmp = preOrder(root->mchild,x);
    if(root->mchild==NULL);
    else if(tmp==NULL||tmp->data!=x) result.pop_back();
    else return tmp;
    tmp = preOrder(root->rchild,x);
    if(root->rchild==NULL);
    else if(tmp==NULL||tmp->data!=x) result.pop_back();
    else return tmp;

}

int main()
{
    //freopen("in.txt","r",stdin);
    int n;
    cin>>n;
    int rt,a,b,c;
    struct triTree* root = new triTree;
    struct triTree* tmp = root;
    for(int i=0;i<n;i++)
    {
        cin>>rt>>a>>b>>c;
        tmp = createTree(tmp,rt,a,b,c);
    }
```

```
    //pre_test(root);
    int m;
    cin>>m;
    target  t[m];
    for(int i=0;i<m;i++)
    {
        cin>>t[i].target>>t[i].priority;
    }
    sort(t,t+m);
//    for(int i=0;i<m;i++)
//    {
//        cout<<t[i].target<<" "<<t[i].priority<<endl;
//    }
//    triTree * node1 = search_node(root,15);
    cout<<node->data<<" "<<node->parent->data<<endl;
//    triTree *node2 = search_node(root,19);
//    triTree *res = getNearestRoot(root,node1,node2);
//    cout<<res->data<<endl;
//    result.clear();
//    preOrder(root,7);
//    print();
    triTree *tmp1 = root;
    triTree *tmp2,*troot;
    triTree *last = root;
    for(int i=0;i<m;i++)
    {
        result.clear();

        tmp2 = search_node(root,t[i].target);
        troot = getNearestRoot(root,tmp1,tmp2);
        while(last!=troot)
        {
             result.push_back(last->data);
             last = last->parent;
        }
        preOrder(troot,t[i].target);
        print();
        cout<<tmp2->data<<" "<<endl;
        tmp1 = tmp2;
        last = search_node(root,result.back());
    }
    while(last!=NULL)
    {
        cout<<last->data<<" ";
```

```
        last = last->parent;
    }
    cout<<endl;
}
```

点评：这个题按以上分析，十分复杂，考察的点非常多，要求关于树的常用算法非常熟悉才可以在最多一个半小时完成。这里说一说都用到哪些数据结构和算法：首先是三叉树的数据结构，（为了处理方便增加 parent 指针），树的建立过程（用队列），结构体排序（sort 函数以及操作符重载），前序遍历求节点，后序遍历求两个节点的最近公共父节点，前序遍历求叶子节点的路径等，细节非常多。近两年北航机考最后一题都是关于树有关的，难度都不低，建议决定报考北航的考生注意这一部分程序设计的实战。

2020 机试题目

注：2020 为远程面试，没有机试。