##### 补充练习1

/\*思路

    要计算这个问题，可以将A连乘b次，每次都对m求余，但这种方法特别慢，当b较大时无法使用。下面给出一种较快的算法(用A^b表示A的b次方)

　　若b=0，则A^b%m=I%m。其中I表示单位矩阵。

　　若b为偶数，则A^b%m=(A^(b/2)%m)^2%m，即先把A乘b/2次方对m求余，然后再平方后对m求余。

　　若b为奇数，则A^b%m=(A^(b-1)%m)\*a%m，即先求A乘b-1次方对m求余，然后再乘A后对m求余。

\*/

#include<iostream>

using namespace std;

int n,b,m;

//以为矩阵的乘法不具有互换性，即矩阵相乘是要有顺序的

//init函数用来初始化，也就是保持p矩阵是先乘的矩阵

//矩阵h是零矩阵，也就是所有元素为0

void init(int\* p,int\* h)

{

    for(int i=0;i<n;i++)

    for(int j=0;j<n;j++){

    \*(p+10\*i+j)=\*(h+10\*i+j);

    \*(h+10\*i+j)=0;

    }

}

//f函数是用来求矩阵相乘以后得到的新矩阵h

//函数中的h+row\*10+col也就是h[row][col]的地址

void f(int\* a,int\* p,int\* h)

{

        for(int row=0;row<n;row++)

        for(int col=0;col<n;col++)

        {

            for(int j=0;j<n;j++)

            \*(h+row\*10+col)+=(\*(p+row\*10+j))\*(\*(a+j\*10+col));

        }

}

//qiuyu函数为求余函数，即对矩阵中的每一个数进行求余

void qiuyu(int\* h,int m)

{

    for(int row=0;row<n;row++)

    for(int col=0;col<n;col++)

    \*(h+10\*row+col)%=m;

}

int main()

{

    scanf("%d%d%d",&n,&b,&m);

    //定义三个数组，a是矩阵，p是乘号前的矩阵，h是0矩阵

    //但在最后没有使用init函数，故最后的h矩阵就是要求的矩阵

    int a[10][10],p[10][10],h[10][10];

    for(int i=0;i<n;i++)

    for(int j=0;j<n;j++){

        scanf("%d",&a[i][j]);

        p[i][j]=a[i][j];     ////开始时将p矩阵复制成a矩阵

    }

    //若b=0，则A^b%m=I%m。其中I表示单位矩阵。

    if(b==0){

        for(int i=0;i<n;i++)

        for(int j=0;j<n;j++)h[i][j]=1%m;

    }

    //若b=1，则A^b%m=A%m

    else if(b==1){

         init(h[0],p[0]);

         qiuyu(h[0],m);

    }

    //若b=2，则A^b%m=A^2%m

    else if(b==2){

        f(a[0],a[0],h[0]);

        qiuyu(h[0],m);

    }

    //若b为大于2的偶数，则A^b%m=(A^(b/2)%m)^2%m，即先把A乘b/2次方对m求余，然后再平方后对m求余。

    else if(b%2==0){

        for(int s=1;s<=b/2-1;s++)

        {

            f(a[0],p[0],h[0]);

            init(p[0],h[0]);

            //每次循环中都初始化一下，也就是将p矩阵始终视作乘号前的矩阵，h是0矩阵，下同

        }

        qiuyu(p[0],m);         //把A乘b/2次方对m求余

        f(p[0],p[0],h[0]);     //再平方

        qiuyu(h[0],m);

    }

    //若b为大于1的奇数，则A^b%m=(A^(b-1)%m)\*a%m，即先求A乘b-1次方对m求余，然后再乘A后对m求余。

    else {

        for(int s=1;s<=b-2;s++){

            f(a[0],p[0],h[0]);

            init(p[0],h[0]);

        }

        qiuyu(p[0],m);

        f(a[0],p[0],h[0]);    //再乘A后对m求余

        qiuyu(h[0],m);

    }

    //因为最后并没有调用init函数，所以h矩阵就是最后求余得到的矩阵

    for(int i=0;i<n;i++)

    {

    for(int j=0;j<n;j++)

    printf("%d ",h[i][j]);

    printf("\n");

    }

    return 0;

}

**测试数据：**

**样例1输入：**

2 2 2

1 1

0 1

**样例1输出**

1 0

0 1

**样例2输入**

**3 3 3**

**1 2 3**

**4 5 6**

**7 8 9**

**样例3输出**

**0 0 0**

**2 1 0**

**0 0 0**

##### 补充练习2

**1.**

#include<iostream>

#include<cstring>

#include<vector>

using namespace std;

//创建vector数组来存放字符串a逆序后的字符串

vector<char> b;

void nixu(string a)

{

    for(int i=a.size()-1;i>=0;i++)

    b.push\_back(a[i]);//逆序存放a中的字符

}

int main()

{

    string a;

    cin >> a;

    nixu(a);

    for(int i=0;i<a.size();i++)cout << b[i] ;//按序输出

    return 0;

}

**测试数据**

**样例输入**

**Abcdefg.**

**样例输出**



**2.**

#include<iostream>

#include<cstring>

#include<vector>

using namespace std;

vector<char> b;    //b数组是用来存放a中的元音字母

//check函数使用来判断字符是不是元音字母的

bool check(char ch)

{

    if(ch=='a' || ch=='A' || ch=='e' || ch=='E'|| ch=='i' || ch=='I' || ch=='o' || ch=='O' ||ch=='u' || ch=='U')

    return true;

    return false;

}

int main()

{

    string a;

    cin >> a;

    for(int i=0;i<a.size();i++)

    if(check(a[i]))b.push\_back(a[i]);  //若a[i]时元音字母，则将其存进b中

    for(int i=0;i<b.size();i++)cout << b[i];

    return 0;

}

**测试数据：**

**样例输入**

**AbcdeFgO**

**样例输出**

**AeO**

**如图**



##### 补充练习3

/\*

思路：本体采用深度优先搜索，从棋盘的第一个位置开始搜索

首先调用pos函数存储每个空位上可能填的数

再从第一个空位开始，深度搜索所有的结果

当成功搜完最后一个元素时，回溯

这样就能完成解法

\*/

#include<iostream>

using namespace std;

char sudoku[9][9];//定义数独棋盘，0表示此处为空，需要填数

char a[9][9];     //a棋盘为最终的正确答案棋盘

int ans[9][9][9]; //ans数组存放着9\*9棋盘上每个位置上可以填的数

int res[9][9];    //res数组表示9\*9棋盘上每个位置上可以填的数字的个数

int row,col,num;  //与后面goin函数有关

//check\_row函数是用来查询与空位在相同行上某个数字(不含0)的个数

int check\_row(int row,char ch)

{

    int f1=0;

    for(int i=0;i<9;i++){

        if(a[row][i]==ch)f1++;

    }

    return f1;

}

//check\_col函数是用来查询与空位在相同列上某个数字(不含0)的个数

int check\_col(int col,char ch)

{

    int f2=0;

    for(int i=0;i<9;i++){

        if(a[i][col]==ch)f2++;

    }

    return f2;

}

//check\_nine函数是用来查询空位所在九宫格上某个数(不含0)的个数

int check\_nine(int i,int j,char ch)

{

    int f3=0;

    for(int x=(i/3)\*3;x<(i/3)\*3+3;x++)

    for(int y=(j/3)\*3;y<(j/3)\*3+3;y++)

        if(a[x][y]==ch)f3++;

    return f3;

}

//pos函数是用来搜索某个空位上可能填的数字,并存储下来

void pos(int i,int j,bool flag)

{

    if(flag)printf("第%d行第%d个空的可能答案有",i+1,j+1);

    int r=0;

    for(char x='1';x<='9';x++)

    {

        int f1=check\_row(i,x);

        int f2=check\_col(j,x);

        int f3=check\_nine(i,j,x);

        if(f1>1 || f2>1 || f3>1){cout << "此题无解" <<endl;return;}

        if(!f1 && !f2 && !f3){if(flag)cout << x << ' ';ans[i][j][r++]=x-'0';res[i][j]++;}

    }

    if(flag)cout << endl;

}

bool flag; //定义回溯的标志，初始为false

void dfs(int i,int j)

{

    if(flag)return; //若标志成立，则一直返回

    //若当前位置不是空位，则搜索下一位

    if(a[i][j]!='0'){

        if(j<8)dfs(i,j+1);

        else if(i<8)dfs(i+1,0);

        else if(i==8){flag=true;return;}//

        //若当前位置不是空位，且是最后一位，则标志成立，返回

    }

    //若当前位置是空位，开始搜索

    else{

        //每个空位上的可能填的数的数量为res[i][j]

        for(int x=0;x<res[i][j];x++)

        {

            char ch='0'+ans[i][j][x];

            int f1=check\_row(i,ch);//查询相同行上的x的数量

            int f2=check\_col(j,ch);//查询相同列上的x的数量

            int f3=check\_nine(i,j,ch);//查询所在九宫格上的x的数量

            if(f1>1 || f2>1 || f3>1)return;//若有一处d的数量大于1，返回

            //若当前位置可填x，继续搜索

            if(!f1 && !f2 && !f3){

                a[i][j]=ch;       //填进去

                if(j<8)dfs(i,j+1);//搜索下一位

                else if(i<8)dfs(i+1,0);

                else if(i==8){flag=true;return;}//若正好是最后以为，标志成立，返回

                if(flag)break;//若标志成立，直接break跳出循环并返回

                a[i][j]='0';//若未搜索成功,

            }

        }

    }

    return;

}

//goin函数是用来输入答案的

void goin()

{

    printf("请按照行数+列数+数字的格式输入");

    cin >> row >> col >> num;

}

//goout函数用来，当输入答案正确时，输出现在的sudoku棋盘

void goout(char\* p)

{

    for(int i=0;i<9;i++){

        for(int j=0;j<9;j++)

            cout << \*(p+9\*i+j)  <<' ';

        cout << endl;

    }

}

int main(void)

{

    int t1=0,t2=0;        //t1记录空位的数量,t2记录已经填入的空位的数量

    for(int i=0;i<9;i++)cin >> sudoku[i];//输入棋盘

    for(int i=0;i<9;i++)

    for(int j=0;j<9;j++)

    {

        a[i][j]=sudoku[i][j]; //将原来的棋盘复制成a,即确定a的初始状态

        if(sudoku[i][j]=='0')t1++;

    }

    for(int i=0;i<9;i++)

    for(int j=0;j<9;j++)

    if(a[i][j]=='0')pos(i,j,true); //第一遍的时候，输出每个位置上可以填入的数

    dfs(0,0);//深搜答案

    printf("正确解法为\n");//输出最终解法

    for(int i=0;i<9;i++){

        for(int j=0;j<9;j++)

            cout<< a[i][j];

        cout << endl;

        }

    while(1){

        //当t2=t1时，表示已经完成，解题成功

        if(t2==t1){

            printf("解题成功，最终答案为\n");

                goout(sudoku[0]);

        }

        goin();//输入解法

        if('0'+num==a[row-1][col-1]){

            printf("正确\n");

            t2++;

            sudoku[row-1][col-1]='0'+num; //填入答案

            goout(sudoku[0]);

        }

        else {

            printf("错误\n");

            for(int i=0;i<9;i++){

                for(int j=0;j<9;j++)

                cout << sudoku[i][j] <<' ';

                cout << endl;

            }

        }

    }

}

**测试数据（号称是世界上最难的数独题）**

**输入9\*9个1~9的数字，0表示空位（默认输入数据有最终解法）**

800000000

003600000

070090200

050007000

000045700

000100030

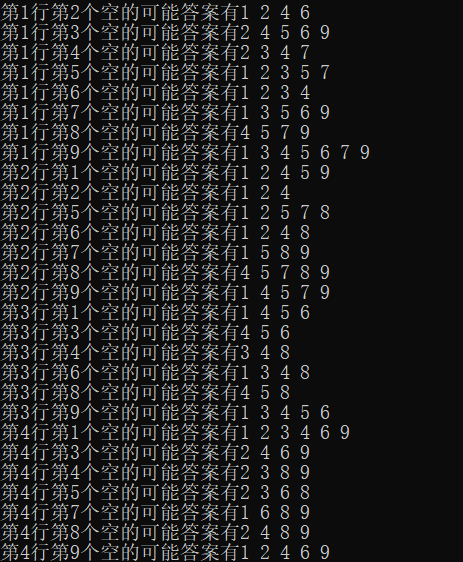
001000068

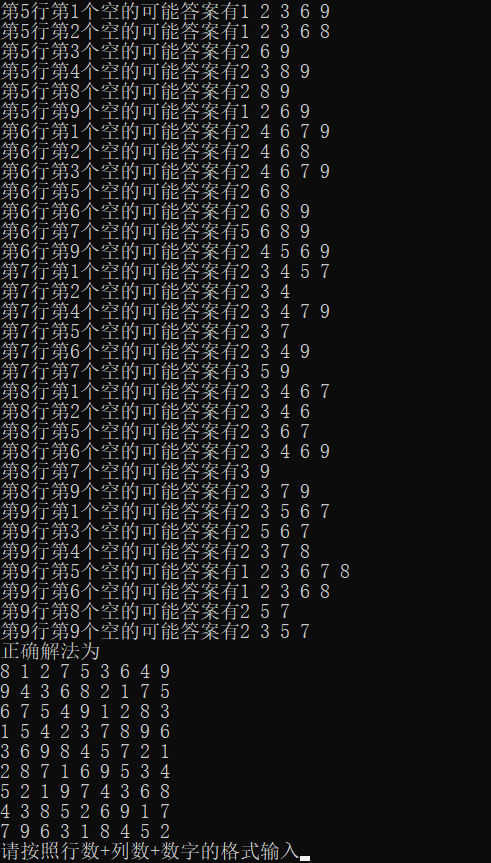
008500010

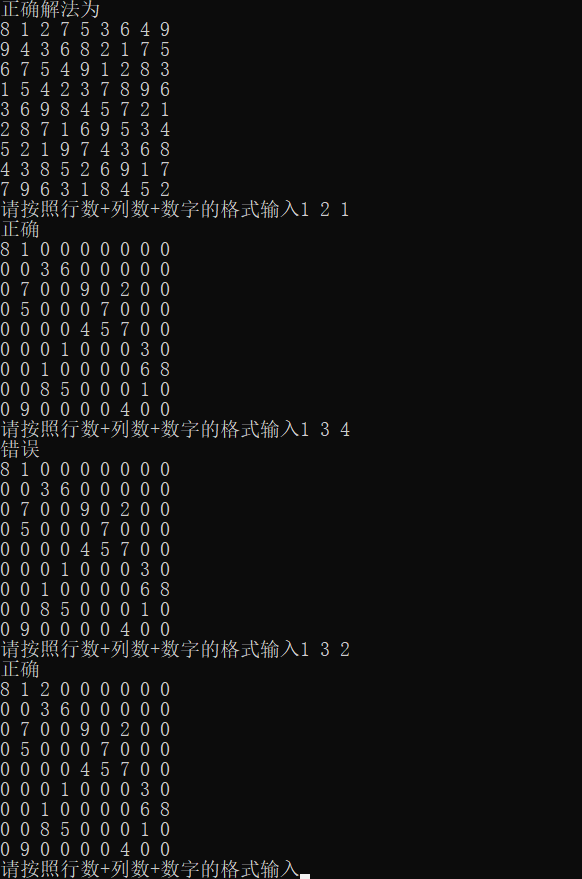
090000400

**运行结果较长**

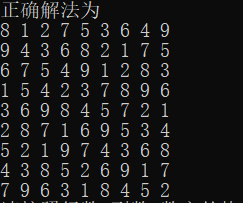
**截图如下**



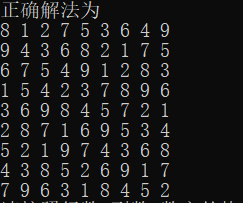


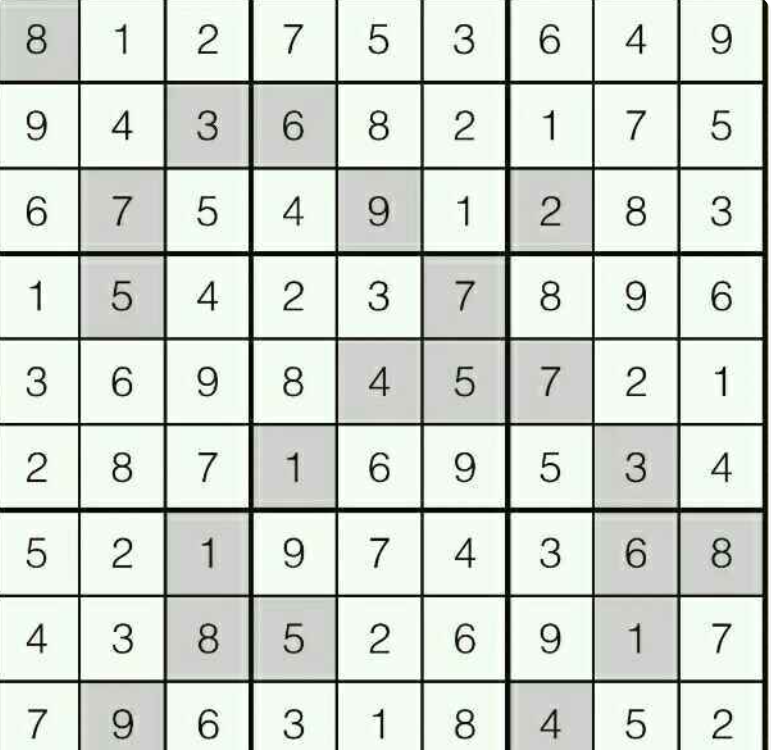


**如此一直输入下去，就能得到最终解法**



**将最终解法与实际结果相比较**





**完全吻合！**

**这样，世界上最难的数独题解出来了**

##### 第二次补充练习

/\*

由于本题中母串str较短，故可以采用暴力搜索的方法，搜索出所有字串在原串中出现的次数

并存贮下来，找到其中的最大值，然后将所有出现次数为最大值的子串输出

因为相同的子串出现的次数相同，故应该从字串的第一个元素对应的母串的下一个元素开始搜

这样就能避免输出相同的字串了

\*/

#include<iostream>

#include<cstring>

using namespace std;

int L;             //定义字串长度为L

string str;

//p函数用来求某个子串在原串中出现的次数

int p(int init,char\* a)

{

    int cnt=0;

    //因为字串在原串中一定会出现1次，故从子串的首元素的下一个元素开始搜索

    for(int i=init+1;i<=str.size()-L;i++)

    {

        int x=0,j=i;

        while(str[j++]==\*(a+x) && x<L)x++;

        if(x==L)cnt++;

    }

    return cnt;

}

int main()

{

    cin >> L;

    cin >> str;

    char t[60];

    int res[60];

    int m=0;

    for(int i=0;i<=str.size()-L;i++)

    {

        int x=i;

        for(int j=0;j<L;j++)t[j]=str[x++];

            res[i]=p(i,t);     //用字串第一个元素对应的母串中的元素的位置来标记子串

                               //存储下来其在母串中出现的次数

            m=max(m,res[i]);  //擂台法找出出现的最大次数

    }

    for(int i=0;i<=str.size()-L;i++)

    {

        int x=i;

        for(int j=0;j<L;j++)t[j]=str[x++];

        //将所有出现出现次数等于最大次数的子串输出

        if(res[i]==m){

            for(int j=0;j<L;j++)

            cout << t[j] ;

            cout <<endl;

        }

    }

    return 0;

}

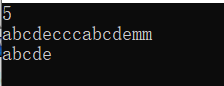
**测试数据**

**输入样例1**

**5**

**abcdecccabcdemm**

**输出**



**符合题意**