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

```
Abcdefg.
.gfedcbA
```

2.

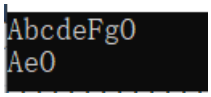
```cpp
#include<iostream>
#include<cstring>
#include<vector>
using namespace std;

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

```
第1行第2个空的可能答案有1 2 4 6
第1行第3个空的可能答案有2 4 5 6 9
第1行第4个空的可能答案有2 3 4 7
第1行第5个空的可能答案有1 2 3 5 7
第1行第6个空的可能答案有1 2 3 4
第1行第7个空的可能答案有1 3 5 6 9
第1行第8个空的可能答案有4 5 7 9
第1行第9个空的可能答案有1 3 4 5 6 7 9
第2行第1个空的可能答案有1 2 4 5 9
第2行第2个空的可能答案有1 2 4
第2行第5个空的可能答案有1 2 5 7 8
第2行第6个空的可能答案有1 2 4 8
第2行第7个空的可能答案有1 5 8 9
第2行第8个空的可能答案有4 5 7 8 9
第2行第9个空的可能答案有1 4 5 7 9
第3行第1个空的可能答案有1 4 5 6
第3行第3个空的可能答案有4 5 6
第3行第4个空的可能答案有3 4 8
第3行第6个空的可能答案有1 3 4 8
第3行第8个空的可能答案有4 5 8
第3行第9个空的可能答案有1 3 4 5 6
第4行第1个空的可能答案有1 2 3 4 6 9
第4行第3个空的可能答案有2 4 6 9
第4行第4个空的可能答案有2 3 8 9
第4行第5个空的可能答案有2 3 6 8
第4行第7个空的可能答案有1 6 8 9
第4行第8个空的可能答案有2 4 8 9
第4行第9个空的可能答案有1 2 4 6 9
```

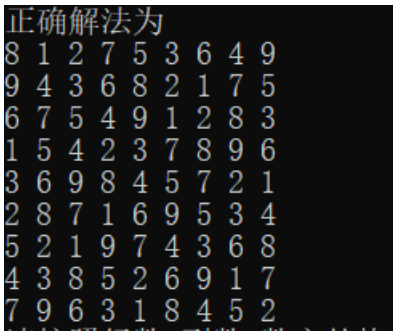
```
第5行第1个空的可能答案有1 2 3 6 9
第5行第2个空的可能答案有1 2 3 6 8
第5行第3个空的可能答案有2 6 9
第5行第4个空的可能答案有2 3 8 9
第5行第8个空的可能答案有2 8 9
第5行第9个空的可能答案有1 2 6 9
第6行第1个空的可能答案有2 4 6 7 9
第6行第2个空的可能答案有2 4 6 8
第6行第3个空的可能答案有2 4 6 7 9
第6行第5个空的可能答案有2 6 8
第6行第6个空的可能答案有2 6 8 9
第6行第7个空的可能答案有5 6 8 9
第6行第9个空的可能答案有2 4 5 6 9
第7行第1个空的可能答案有2 3 4 5 7
第7行第2个空的可能答案有2 3 4
第7行第4个空的可能答案有2 3 4 7 9
第7行第5个空的可能答案有2 3 7
第7行第6个空的可能答案有2 3 4 9
第7行第7个空的可能答案有3 5 9
第8行第1个空的可能答案有2 3 4 6 7
第8行第2个空的可能答案有2 3 4 6
第8行第5个空的可能答案有2 3 6 7
第8行第6个空的可能答案有2 3 4 6 9
第8行第7个空的可能答案有3 9
第8行第9个空的可能答案有2 3 7 9
第9行第1个空的可能答案有2 3 5 6 7
第9行第3个空的可能答案有2 5 6 7
第9行第4个空的可能答案有2 3 7 8
第9行第5个空的可能答案有1 2 3 6 7 8
第9行第6个空的可能答案有1 2 3 6 8
第9行第8个空的可能答案有2 5 7
第9行第9个空的可能答案有2 3 5 7
正确解法为
8 1 2 7 5 3 6 4 9
9 4 3 6 8 2 1 7 5
6 7 5 4 9 1 2 8 3
1 5 4 2 3 7 8 9 6
3 6 9 8 4 5 7 2 1
2 8 7 1 6 9 5 3 4
5 2 1 9 7 4 3 6 8
4 3 8 5 2 6 9 1 7
7 9 6 3 1 8 4 5 2
请按照行数+列数+数字的格式输入
```

```
正确解法为
8 1 2 7 5 3 6 4 9
9 4 3 6 8 2 1 7 5
6 7 5 4 9 1 2 8 3
1 5 4 2 3 7 8 9 6
3 6 9 8 4 5 7 2 1
2 8 7 1 6 9 5 3 4
5 2 1 9 7 4 3 6 8
4 3 8 5 2 6 9 1 7
7 9 6 3 1 8 4 5 2
请按照行数+列数+数字的格式输入1 2 1
正确
8 1 0 0 0 0 0 0 0
0 0 3 6 0 0 0 0 0
0 7 0 0 9 0 2 0 0
0 5 0 0 0 7 0 0 0
0 0 0 0 4 5 7 0 0
0 0 0 1 0 0 0 3 0
0 0 1 0 0 0 0 6 8
0 0 8 5 0 0 0 1 0
0 9 0 0 0 0 4 0 0
请按照行数+列数+数字的格式输入1 3 4
错误
8 1 0 0 0 0 0 0 0
0 0 3 6 0 0 0 0 0
0 7 0 0 9 0 2 0 0
0 5 0 0 0 7 0 0 0
0 0 0 0 4 5 7 0 0
0 0 0 1 0 0 0 3 0
0 0 1 0 0 0 0 6 8
0 0 8 5 0 0 0 1 0
0 9 0 0 0 0 4 0 0
请按照行数+列数+数字的格式输入1 3 2
正确
8 1 2 0 0 0 0 0 0
0 0 3 6 0 0 0 0 0
0 7 0 0 9 0 2 0 0
0 5 0 0 0 7 0 0 0
0 0 0 0 4 5 7 0 0
0 0 0 1 0 0 0 3 0
0 0 1 0 0 0 0 6 8
0 0 8 5 0 0 0 1 0
0 9 0 0 0 0 4 0 0
请按照行数+列数+数字的格式输入
```

如此一直输入下去，就能得到最终解法

```
正确解法为
8 1 2 7 5 3 6 4 9
9 4 3 6 8 2 1 7 5
6 7 5 4 9 1 2 8 3
1 5 4 2 3 7 8 9 6
3 6 9 8 4 5 7 2 1
2 8 7 1 6 9 5 3 4
5 2 1 9 7 4 3 6 8
4 3 8 5 2 6 9 1 7
7 9 6 3 1 8 4 5 2
```

将最终解法与实际结果相比较



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

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

```
5
abcdecccabcdemm
abcde
```

符合题意