实验一：古典密码 实验报告

> 17341125 蒙亚愿 信息安全

一、 实验目的

> 通过实验掌握维吉尼亚密码、希尔密码算法及其分析，加深对其认识。

二、实验要求

> 1，用C++编程实现分析课本31页1.21(b)的维吉尼亚密码的密文，给出清晰的分析过程。
> 2，用C++编程实现课本33页1.24。

三、 实验介绍

• 3.1 实验综述

◦ 实验（1）

已知维吉尼亚密码的明文，先通过指数重合法求出密钥长度 m，再把密文串等分为 m 串，在每一个子串中统计 M_g 值，最接近0.065的 M_g 的 g 值就是密钥之一。确定了密钥之后，就可以与密文$mod\ 26$减法计算出明文。

◦ 实验（2）

已知明文、密文以及密钥长度 m ，把明文和密文 m 等分，再每 m 个组成一组，形成一个 $m \times m$ 的方阵。利用矩阵减法和矩阵求逆、相乘得出密钥矩阵，从而就可以求出 B 。

- 3.2 开发环境
  - 运行环境：Windows
  - 使用语言：C++

四、 算法原理

- 4.1 实验（1）
  - **求密钥长度：指数重合法**

假设维吉尼亚密码加密的密文串为 y，长度为n。将y 分割成m个长度相等的子串，长度为l，分别为$y_1, y_2, \ldots\ldots y_m$ ，组成一个$m \times \frac{n}{m}$矩阵，矩阵的第i行对应于子串i，形式如下：

$$Y_1 = y_1 y_{m+1} y_{2m+1} \cdots\cdots$$
$$Y_2 = y_2 y_{m+2} y_{2m+2} \cdots\cdots$$

.....

$$Y_m = y_m y_{2m} y_{3m} \cdots\cdots$$

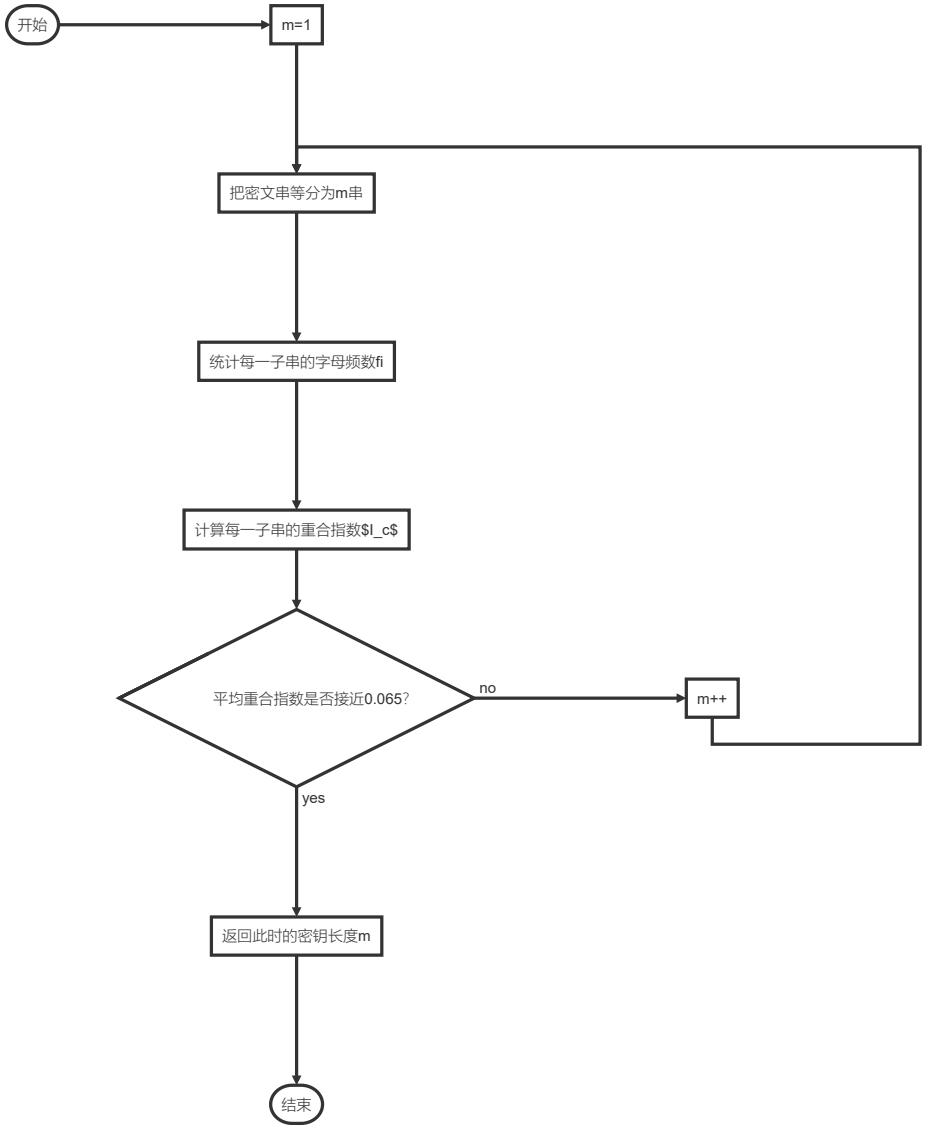
统计每一子串中各个字母出现的频数f_i，计算每一子串的重合指数I_c：

$$I_c(x) = \frac{\sum_{i=0}^{25} f_i(f_i-1)}{l(l-1)}$$

（$f_0, f_1, \ldots\ldots f_{25}$分别表示$A, B, \ldots\ldots Z$出现的频数，共有$C_n^2$种方法来选择$x$中的任两个元素）

当子串们的平均重合指数接近0.065时（小于指定误差），确定了密钥长度m。

流程图如下：


开始
m=1
把密文串等分为m串
统计每一子串的字母频数fi
计算每一子串的重合指数I_c
平均重合指数是否接近0.065?
no
m++
yes
返回此时的密钥长度m
结束

- 求密钥字

  确定了密钥长度m之后，还是对于以上的m个子串，每个子串的长度为l，对每个子串统计$A, B, \ldots\ldots Z$出现的频数，求26个字母再子串y_i中的概率分布：

  $$\frac{f_0}{l}, \frac{f_1}{l}, \ldots\ldots \frac{f_{25}}{l},$$

  考虑到子串y_i是由对应的待加密的明文子集中的字母移动k_i个位置得到的。因此，移位后的概率分布为：

  $$\frac{f_{k_i}}{l}, \frac{f_{k_i+1}}{l} \cdots\cdots \frac{f_{k_{i+25}}}{l}$$

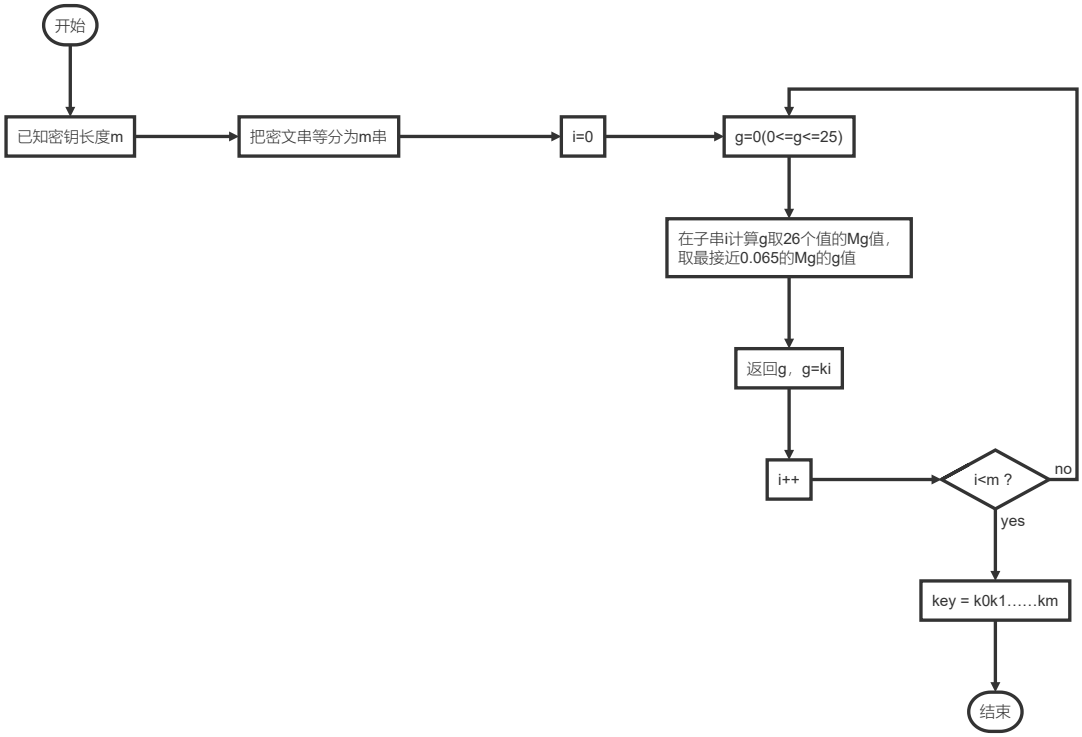
  会近似于英文字母的概率分布。

  所以对于$0 \leq g \leq 25$，计算

  $$M_g = \sum_{i=0}^{25} \frac{p_i f_{i+g}}{l}$$

  选组26个M_g中最接近0.065的g作为k_i，从而每个子串都会得到密钥字。

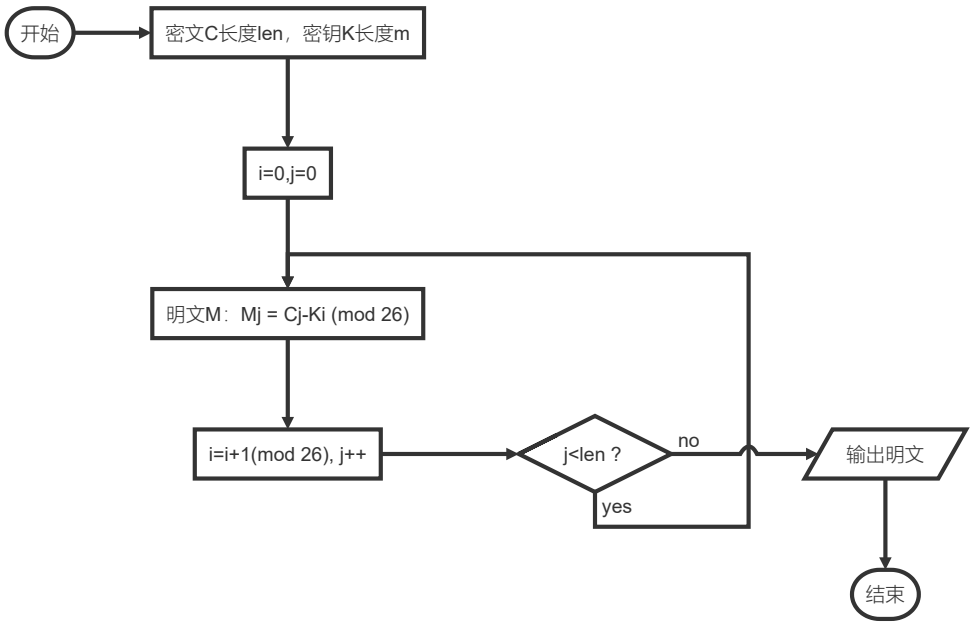
  流程图如下：



- **求明文**

  求出完整的密钥字之后，对于已知密文，就可以用密文减密钥模26得出明文。

  流程图：



- ## 4.2 实验（2）

已知明文P、密文C、密钥长度m，把明文和密文的字符每m个作为一组，每m组形成一个$m \times m$的方阵，从而形成方程组

$$\begin{bmatrix} y_0 & y_1 & \cdots & y_{m-1} \\ y_m & y_{m+1} & \cdots & y_{2m-1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m(m-1)} & y_{m(m-1)+1} & \cdots & y_{m^2-1} \end{bmatrix} = \begin{bmatrix} x_0 & x_1 & \cdots & x_{m-1} \\ x_m & x_{m+1} & \cdots & x_{2m-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m(m-1)} & x_{m(m-1)+1} & \cdots & x_{m^2-1} \end{bmatrix} \begin{bmatrix} l_{0,0} & l_{0,1} & \cdots & l_{0,m-1} \\ l_{1,0} & l_{1,1} & \cdots & l_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ l_{m-1,0} & l_{m-1,1} & \cdots & l_{m-1,m-1} \end{bmatrix} + \begin{bmatrix} b_0 & b_1 & \cdots & b_{m-1} \\ b_0 & b_1 & \cdots & b_{m-1} \\ \vdots & \vdots & \ddots & \vdots \\ b_0 & b_1 & \cdots & b_{m-1} \end{bmatrix}$$

求密钥K矩阵：通过矩阵相减得到$Y' = X'K$, 求X的逆X^{-1}再与Y'相乘，求得K，代入方程组之一求得B。

值得注意的是

1. 在所有的运算中, 都需要$mod\ 26$, 求出的行列式，伴随矩阵，矩阵乘法, 矩阵减法等都需要$mod\ 26$, 确保得出的每一个　　数值都在区间$[0, 25]$中.
2. 在计算逆X^{-1}时, 算出的行列式det在$mod\ 26$之后, **还要检验det与26是否互素**, 只有两者互素时, X才存在逆, 本来一个矩阵可逆是指其行列式不等于零,

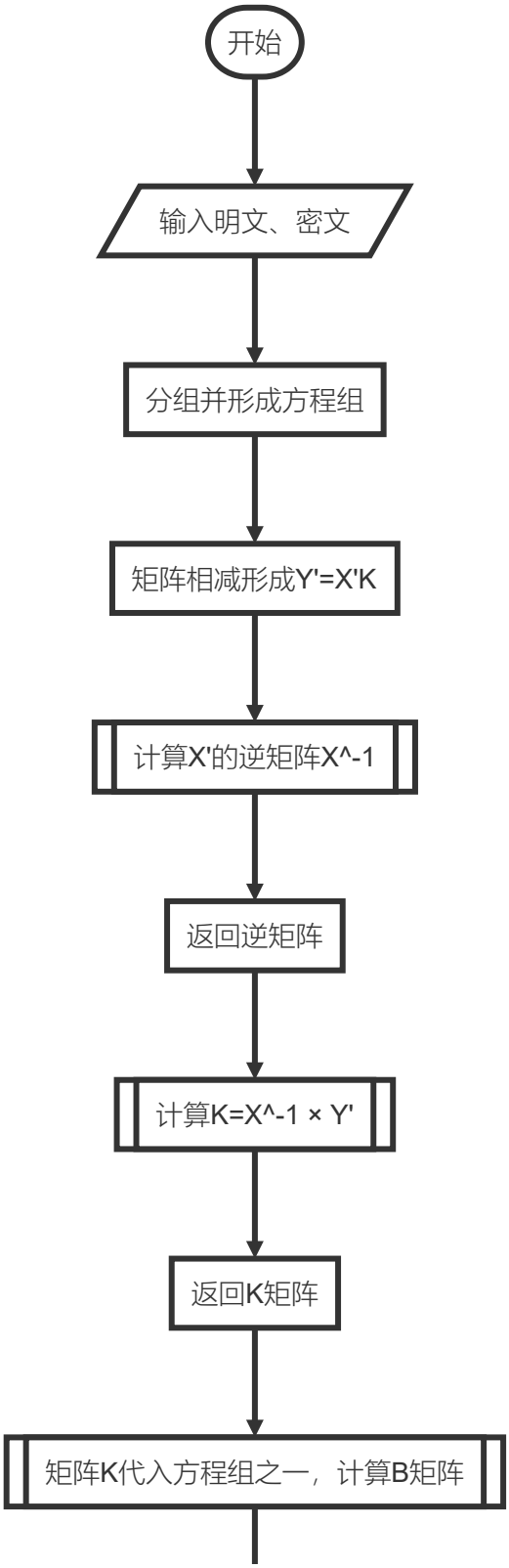
但是由于所有的运算都是在Z_{26}上进行的, 所以可逆的充要条件就是$gcd(det, 26) = 1$.

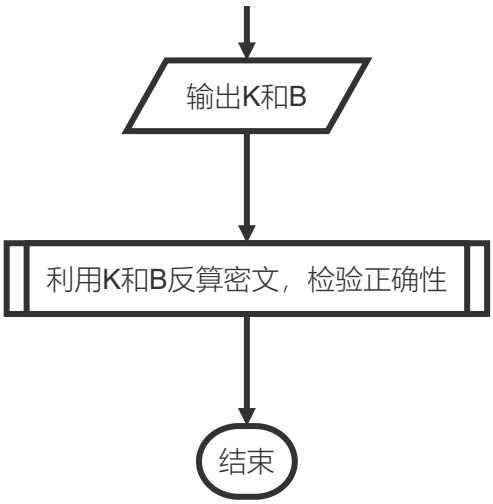
3. 在式子$X^{-1} = \frac{X^*}{det}$我们需要记住所有的运算都是在Z_{26}上进行的, 所以式子中的除以det实际上就是乘上det模26的逆det^{-1}, 即

$$X^{-1} = X^* \times det^{-1}$$

最后得出的结果也是需要$mod\ 26$。

流程图:





五、 代码设计

- 5.1 实验（1）
  - A. main模块

    main函数的作用主要用于输入密文，统一以下各个模块，输出密钥长度、密钥、明文。

    方便起见，变量 $code$ (密文) 和 $length$ (密文长度) 都是全局变量。

```
int main(){

cin>>code;
    length = code.length();

    const int keyLen = getKeyLen();
    string keyword = getKeyword(keyLen);
    string word = getMessenge(keyword, keyLen);

    cout<<"密钥长度为："<<keyLen<<endl;
    cout<<"密钥："<<keyword<<endl;
    cout<<"明文："<<word<<endl;
}
```

  - B. getKeyLen模块

    这个模块用于计算并返回密钥长度。

依照算法原理，假设密钥长度m从1开始，把密文等分为m串，统计每一串各个字母出现的频数和每一串的长度。

```
            for(int j=0; i+j*keyLen<length; j++)
                cnt[(int)(code[i+j*keyLen]-'A')]++;

            for(int k=0;k<26;k++)  size+=cnt[k];
```

接着根据公式计算每个字母串的重合指数，当平均重合指数$avgI_c \in [0.06, 0.07]$中时，可以认为$avgI_C$很接近0.065，即可返回此时的m值，否则，m增加1继续计算。

```
            size*=(size-1);
            double temp=0.00f;
            for(int u=0;u<26;u++){
                if(cnt[u]!=0)
                    temp += ((double)cnt[u]*(double)
(cnt[u]-1))/(double)size;
            }
            IC[i]=temp;
        }

        double avgIC = 0;
        for(int w=0;w<keyLen;w++)  avgIC+=IC[w];
        avgIC /= keyLen ;
        if(avgIC >= 0.06 && avgIC <= 0.07){
            return keyLen;
        }
        keyLen++;

```

- C. getKeyword模块

这个模块计算并返回密钥字。

首先记下实际生活中各个字母出现的频率，用于之后M_g的计算。

```
double p[26]={0.082, 0.015, 0.028, 0.043, 0.127, 0.022, 0.020, 0.061,
0.070, 0.002, 0.008, 0.040, 0.024, 0.067, 0.075, 0.019, 0.001, 0.060,
0.063, 0.091, 0.028, 0.010, 0.023, 0.001, 0.020, 0.001 } ;
```

在之前的模块已经计算出密钥长度m，同样是把密文等分成m串，对于每一串都统计其各个字母出现的频数。

```
    for(int x=0;w+x*keyLen<length;x++){   //统计目前这一串的各个字母的个数
        cnt[code[w+x*keyLen]-'A']++, size++;
    }
```

在每一个子串内，利用公式计算k遍历0到26，计算移动k位后的M_g值，取最接近0.065的k值，作为$keyword$之一，从而计算出$keyword$。

```
    for(int k=0;k<26;k++){     //移动k个试试看
        double temp=0.0 ;
        //移k位后的MG
        for(int a=0;a<26;a++){
            temp+=(double)(p[a]*cnt[(k+a)%26]) / (double)size ;
        }
        MG[k] = temp ;
        if(fabs(MG[k]-0.065) <= diff )
              diff=fabs(MG[k]-0.065), mark=k ;
    }
    word[w]=(char)(mark+'A');
    keyword += word[w];
```

- D. getMessenge模块

  在之前的模块中计算出了密钥字之后，就可以根据加密算法反推解密算法，即通过密文减密钥$(mod\ 26)$得到明文。

```
string getMessenge(string keyword, int keyLen){
    char  m[1000];
    string meg="";
    int k=0;

    for(int i=0;i<length;i++){
        m[i]=(code[i]-keyword[k]+26)%26 ;
        m[i]=(char)(m[i]+'a') ;  // 转换成字符
        k++;
        k%=keyLen ;  //密钥轮回
        meg += m[i];
    }
    return meg ;
}
```

- 5.2 实验（2）
  - A. main模块

这个模块主要用于输入明文、密文，形成方程组。

分别输入明文、密文到相应的二维$1000 \times m$数组中（$m = 3$），每m行形成一个$Y = XK + B$方程组，再通过两个方程相减形成$Y' = X'K$的方程。

```
    cout<<" 请输入明文:";   cin>>word;
    for(int k=0;k<word.length();k++){
        x[i][j] = (int)(word[k]-'a');
        j++;
        if(j==m)  j=0,i++;
    }
```

```
    int xdiff[m][m]={0}, ydiff[m][m]={0} ;
    for(int k=0;k<m;k++){
        for(int u=0;u<m;u++){
            xdiff[k][u] = x[k+m][u]-x[k][u] ;
            ydiff[k][u] = y[k+m][u]-y[k][u] ;
        }
    }

```

- B. getInverse模块

函数$getInverse()$用于求出X'的逆X'^{-1}。

根据公式$X^{-1} = X^* \times det^{-1}$，需要计算行列式的逆$det^{-1}$以及伴随矩阵$X^*$。

1. 求行列式的逆det^{-1}

a) 计算行列式

对于矩阵X'的第一行，求每一个元素的余子式：划去该元素所在的行与列的元素，剩下的元素按原样排列，得 到新的行列式，再继续递归求解。

求出之后，再求该元素的代数余子式：余子式与该元素对应的正负符号的乘积。

行列式就等于按照第一行展开成元素与其对应的代数余子式的乘积之和(线性代数有学，不多加叙述)。

```
int getDet(int a[m][m],const int size){
    if(size==1)  return a[0][0];

```

```
    int det=0;
    int temp[m][m] = {0} ;

    for(int i=0;i<size;i++){
        for(int j=0;j<size;j++){
            for(int k=0;k<size;k++){
                if(k>=i)  temp[j][k] = a[j+1][k+1];
                else temp[j][k] = a[j+1][k];
            }
        }
        int t = getDet(temp, size-1);
        if(i%2==0)  det+=a[0][i]*t;
        else det-=a[0][i]*t;
    }
    return det;
}
```

b) 计算行列式的逆并检验

利用逆的定义：

$$a \cdot a^{-1} = 1(mod 26)$$

计算逆时，由于26不是很大，使用最简单的方法循环计算。

```
int getDetInver(const int det){
    for(int i=0;i<26;i++){
        int temp = i*det ;
        if(temp%26==1)  return  i;
    }
    return -1;
}
```

再计算det^{-1}是否与26互素，只有互素的时候，矩阵X'的逆才存在。

```
int gcd(int x,int y){
    if(y==0)  return x;
    return gcd(y, x%y);
}
```

2. 求伴随矩阵X^*

对于矩阵中的所有元素，都求其余子式形成新的矩阵，该矩阵再转置即为伴随矩阵。

```
void getAd(int a[m][m],int ad[m][m],int size){
    if(size==1){
        ad[0][0]=1;  return ;
    }
    int temp[m][m] = {0};

    for(int i=0;i<size;i++){
        for(int j=0;j<size;j++){
            for(int k=0;k<size-1;k++){
                for(int u=0;u<size-1;u++){
                    temp[k][u] = a[k>=i? k+1:k][u>=j? u+1:u];
                }
            }
            if((i+j)%2==0) ad[j][i] = getDet(temp,size-1);
            else ad[j][i] = -getDet(temp, size-1);  //转置
            ad[j][i] = mod26(ad[j][i]);
        }
    }

}
```

3. 计算逆矩阵X'^{-1}

```
    for(int i=0;i<m;i++){
        for(int j=0;j<m;j++){
            inver[i][j] = ad[i][j]*DetInver ;
            inver[i][j] = mod26(inver[i][j]);
        }
    }
```

- C. 获取密钥矩阵K模块

  通过矩阵Y'与X'^{-1}相乘得出矩阵K。（最后的结果都要模26）

```
    for(int i=0;i<m;i++){
        for(int j=0;j<m;j++){
            key[i][j] = 0 ;
            for(int k=0;k<m;k++){
                key[i][j] += inver[i][k]*yd[k][j] ;
            }
            key[i][j] = mod26(key[i][j]) ;  //mod 26
        }
    }

```

- D.获取B矩阵模块

  以上已经求出了K，只需代入方程组之一即可求出矩阵B。

```
void getB(int y[m][m], int x[m][m], int key[m][m], int b[m][m]){
    int product[m][m] ={0} ;
    multiple(x, key, product) ;
//  cout<<"product"<<endl;
//  print(product);

    for(int i=0;i<m;i++){
        for(int j=0;j<m;j++){
            b[i][j] = y[i][j] - product[i][j] ;
            b[i][j] = mod26(b[i][j]);
        }
    }

    cout<<"B:"<<endl;
    print(b);
}
```

- check模块

  以上已经计算出了矩阵K和B，为了确保计算的正确性，重新计算经过希尔日仿射函数加密后的密文，对照已知密文，即可验证矩阵。

```
bool check(int x[1000][m], int key[m][m], int b[m][m], int y[1000][m],int col){
    int temp[1000][m]={0};
    //加密
    for(int i=0;i<col;i++){
        for(int j=0;j<m;j++){
            temp[i][j] = 0;
            for(int k=0;k<m;k++){
                temp[i][j] += x[i][k]*key[k][j];
            }
            temp[i][j] += b[0][j] ;
            temp[i][j] = mod26(temp[i][j]) ;
        }
    }
    cout<<"经过以上密钥key,b加密后,密文:"<<endl;
    for(int i=0;i<col;i++){
        for(int j=0;j<m;j++)
            cout<<temp[i][j]<<"  ";
        cout<<endl;
    }
```

```
20
21      for(int i=0;i<col;i++){
22          for(int j=0;j<m;j++){
23              if(temp[i][j]!=y[i][j])  return false;
24          }
25      }
26
27      return true;
28  }
```

六、 实验结果及分析

- 6.1 实验（1）运行结果

分析：在输入密文之后，先计算出密钥长度为6，接着计算出密钥字：$CRYPTO$，最后对于密文，逐个进行解密、输出。经过手动计算，实验结果正确。

```
密文：KCCPKBGUFDPHQTYAVINRRTMVGRKDNBVFDETDGILTXRGUDDKOTFMBPVGEGLTGCKQRACQCWDNAWCRXIZAKFTLEWRPTYCQKY
VXCHKFTPONCQQRHJVAJUWETMCMSPKQDYHJVDAHCTRLSVSKCGCZQQDZXGSFRLSWCWSJTBHAFSIASPRJAHKJRJUMVGKMITZHFPDIS
PZLVLGWTFPLKKEBDPGCEBSHCTJRWXBAFSPEZQNRWXCVYCGAONWDDKACKAWBBIKFTIOVKCGGHJVLNHIFFSQESVYCLACNVRWBBIRE
PBBVFEXOSCDYGZWPFDTKFQIYCWHJVLNHIQIBTKHJVNPIST

密钥长度为：6

密钥：CRYPTO

明文：ilearnedhowtocalculatetheamountofpaperneededforaroomwheniwasatschoolyoumultiplythesquarefoota
geofthewallsbythecubiccontentsofthefloorandceilingcombinedanddoubleityouthenallowhalfthetotalforope
ningssuchaswindowsanddoorsthenyouallowtheotherhalfformatchingthepatternthenyoudoublethewholethingag
aintogiveamarginoferrorandthenyouorderthepaper
```

- 6.2 实验（2）运行结果

分析：分别输入明文x、密文y，计算输出它们对应的矩阵，用于之后的校验。计算并输出密钥矩阵key以及B矩阵之后，重新对明文进行加密，如果加密出的密文与以上刚输入的密文一致，即可确定计算的key以及B是正确的。经过手动分析，实验结果正确。

```
请输入明文:adisplayedequation
请输入密文:DSRMSIOPLXLJBZULLM
x:
0  3  8
18  15  11
0  24  4
3  4  16
20  0  19
8  14  13

y:
3  18  17
12  18  8
14  15  11
23  11  9
1  25  20
11  11  12

key:
3  6  4
5  15  18
17  8  5

B:
8  13  1
8  13  1
8  13  1

经过以上密钥key,b加密后,密文:
3  18  17
12  18  8
14  15  11
23  11  9
1  25  20
11  11  12
DSRMSIOPLXLJBZULLM

与原密文一致，检验通过!
```

七、实验总结

1. 两个实验都是在充分了解学习了密码体制之后才能编写代码。

   对于维吉尼亚密码，我原以为上课所学的已足够了解，但是真正的编程时却发现其内在算法也只是略知皮毛。再仔细阅读了课本的算法之后，才能成功的编程。

   而希尔仿射密码，在最开始的求解过程中，我也只是简单的按照原来在线性代数所学的求解矩阵方程，于是出现了很多小数，不思其解。在同学的提醒之下想起来需要$mod\ 26$，才知道自己忽略了许多重要的细节，再翻书认真学习这两种密码的计算方法之后，在原来的单纯的求解逆以及矩阵相乘、相减等的基础上，增加了$mod\ 26$，以及求行列式在Z_{26}中的逆，最后成功地求解。

2. 对密码也有了上升字面的理解。

   破译维吉尼亚密码的快慢与破译的成功取决于对密码长度的判断，手动计算时我偏向于使用$Kasiski$测试法，这时就需要在密文中主义重复的字段，他们之间的距离很有可能就是密钥长度的倍数。使用指数重合法的时候，在判断密文中的频率与标准频率是否符合时，不仅要考虑是否低频率的字母出现得多，高频率的出现得少。

同时，我在程序中设置的I_c的逼近误差范围也许不够精确，当密文足够多的时候，设置的范围相应的也要足够小。

希尔密码就是基于线性变换，它相对于移位密码以及仿射密码而言，最大的好处就是隐藏了字符频率的信息，使得传统的通过频率来破译密文的方法失效。

3. 本次实验的难度不是很大，当理解学习了这些密码体制之后，就可以简单的进行编程，但是收获很大，锻炼了编程能力，也加深了课堂上的学习，只是要敢于质疑，要相信经过自己认真学习仔细验证的答案，不要太依赖网上的答案。

   最后辛苦老师和TA啦。