**云南大学数学与与统计学院**

**上机实践报告**

|  |  |  |
| --- | --- | --- |
| **课程名称**：近代密码学实验 | **年级**：2013 | **上机实践成绩**： |
| **指导教师**：陆正福 | **姓名**：金洋 |  |
| **上机实践名称**：纠错密码学实验 | **学号**：20131910023 | **上机实践日期**：11.29 |
| **上机实践编号**：No.5 | **组号**： | **上机实践时间**： **18:56** |

**一、实验目的**

1.熟悉纠错密码学的基本概念

2.掌握纠错密码学编程的基本方法

3.理解纠错码用于密码体制设计的基本思路

4.查阅资料，尽可能将原理级的验证型实验逐渐演化为实现级的设计型实验

**二、实验内容**

1. 复习“信息论基础实验”课程中Hamming码实验，做简单测试与分析。

2. 以Hamming码为基础，**设计**McEliece密码体制：密钥对生成算法，加密算法，解密算法。

3. 分析以Hamming码为基础的纠错密码体制的局限性。

4. 通过实验说明，纠错密码学中的基础纠错码应该具备哪些要点才能保证密码体制的安全性。

5. 分析Goppa码用于纠错密码体制有何优势？

**三、实验平台**

硬件平台：不做限制，可以是各类计算机。

操作系统：不做限制，如Windows，Linux，MacOSX等。

编程平台：不做限制，如Java/Scala+大整数库，C/C++/Python+GMP+NTL库，数学软件（Sage，Matlab，Maple，Mathematica，Magma等）。

说明：

1. 与前四个实验有所变化的是：前四个实验与大整数有关，本实验的必做部分则与有限域、有限域上的矩阵有关。
2. 部分基础纠错码与有限域上的多项式（或有理分式）有关。如果要做安全性更强的实验，则需要选择合适的配套代码库，或做与数据结构与算法设计有关的基础编程。

**四、实验记录与实验结果分析**

（注意记录实验中遇到的问题。实验报告的评分依据之一是实验记录的细致程度、实验过程的真实性、实验结果的解释和分析。**如果涉及实验结果截屏，应选择白底黑字。**）

1. 复习“信息论基础实验”课程中Hamming码实验，做简单测试与分析。

(1) (15,11,3)Hamming 码编码和译码算法的实现

**Hamming.java**

**package** MC05;

**import** java.util.ArrayList;

**public** **class** Hamming {

/\*校验矩阵\*/

**public** **int**[][] checkMatrix(**int** n) {

ArrayList<Integer> array=**new** ArrayList<Integer>();

**int** m=(**int** )(Math.*pow*(2,n)-1);

**int** [][] H=**new** **int** [n][m];

**for** (**int** i=0;i<n;i++)

**for** (**int** j=0;j<n;j++)

H[i][j]=0;

**for** (**int** i=0;i<n;i++)

array.add((**int** )(Math.*pow*(2,i)));

**int** p=0,q=0,mm=1;

String b=**new** String();

**for** (;q<m-n;) {

b=Integer.*toBinaryString*(mm);

**if** (!array.contains(mm)) {

**for** (**int** j=n-1;j>=0;j--) {

**if** (p==b.length()) **break**;

**else** { H[j][q]=Integer.*parseInt*(b.substring(b.length()-(p+1),b.length()-p));

p++;

}

}

q++;

p=0;

}

mm++;

}

**for** (**int** i=0;i<n;i++) {

**for** (**int** j=0;j<n;j++)

**if** (i==j)

H[i][j+(**int**)(Math.*pow*(2,n)-1-n)]=1;

}

**return** H;

}

/\*生成矩阵\*/

**public** **int**[][] creatMatrix(**int** H[][],**int** n){

**int** i,j,m=(**int**)(Math.*pow*(2, n)-1);

**int** mm=(**int**) (Math.*pow*(2, n)-1-n);

**int**[][] G=**new** **int**[mm][m];

**for** (i=0;i<mm;i++)

**for** (j=0;j<mm;j++)

**if** (i==j) G[i][j]=1;

**for** (i=0;i<n;i++)

**for** (j=0;j<mm;j++)

G[j][i+mm]=H[i][j];

**return** G;

}

/\*Hamming码\*/

**public** **int**[] HammingCoding(**int**[][] g,ArrayList<Integer> array) {

**int** i,j,num;

**int**[] code=**new** **int**[g[1].length];

**for** (i=0;i<g[1].length;i++) {

num=0;

**for** (j=0;j<g.length;j++)

num+=array.get(j)\*g[j][i];

**if** ((num & 1)==1) code[i]=1;

**else** code[i]=0;

}

**return** code;

}

/\*检错纠错\*/

**public** **void** checkCorrectError(**int** h[][],ArrayList<Integer> array) {

**int** i,j,num;

**int** [] HX=**new** **int**[h.length];

**for** (i=0;i<h.length;i++) {

num=0;

**for** (j=0;j<array.size();j++)

num+=h[i][j]\*array.get(j);

**if** ((num & 1)==1) HX[i]=1;

**else** HX[i]=0;

}

print("H\*X=[");

**for** (i=0;i<HX.length;i++) {

print (HX[i]);

**if** (i!=HX.length-1) print(",");

}

print("]\n");

**boolean** flag=**true**;

**for** (i=1;i<HX.length;i++) {

**if** (HX[i]!=0) {

flag=**false**;

**break**;

}

}

**if** (! flag) {

**int** m=0;

m=researchIndex(h,HX);

**if** (m!=0) {

print("接收的信息出错，出错位置为第"+m+"位.\n");

print("纠错前的信息为:");

**for** (i=0;i<array.size();i++)

print(array.get(i));

/\*纠错\*/

m=m-1;

array.set(m, array.get(m)^1);

print("\n纠错后的信息为：");

**for** (i=0;i<array.size();i++) print(array.get(i));

print("\n");

}

}

**else** {

print("接收的信息正确，信息为：");

**for** (i=0;i<array.size();i++) print(array.get(i));

print("\n");

}

}

**public** **int** researchIndex(**int**[][] h, **int**[] hX) {

**int** i,j,num;

**for** (i=0;i<h[1].length;i++) {

num =0;

**for** (j=0;j<h.length;j++) {

**if** (hX[j]==h[j][i]) num++;

}

**if** (num==h.length) **return** i+1;

}

**return** 0;

}

**public** **void** print(**int** str) {

System.***out***.print(str);

}

**public** **void** print(String str) {

System.***out***.print(str);

}

}

**TestHamming.java**

**package** MC05;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**public** **class** TestHamming {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Hamming HM=**new** Hamming();

ArrayList<Integer> array1=**new** ArrayList<Integer>();

ArrayList<Integer> array2=**new** ArrayList<Integer>();

**int** in,k;

**int**[][] H;//校验矩阵

**int**[][] G;//生成矩阵

String input;

**boolean** flag1=**true**,flag2=**true**,flag3=**true**,flag4=**true**;

**while** (1==1) {

k=0;

array1.clear();

array2.clear();

HM.print("继续? Y/N:");

Scanner s=**new** Scanner(System.***in***);

input=s.next();

**if** (input.equals("N")) **break**;

HM.print("请输入待编码的二进制字符串,以2结束");

in=s.nextInt();

**if** (!(in==1 || in ==0)) HM.print("输入错误，请重新输入\n");

**while** (in!=2 &&(in==1||in==0)) {

array1.add(in);

in=s.nextInt();

**if** ((in==1 || in ==0 || in==2)==**false**)

flag1=**false**;

}

**if** (!flag1) HM.print("输入错误，请重新输入\n");

**for** (**int** i=0;;i++) {

**if** (Math.*pow*(2,k)-k==array1.size()+1) {

flag3=**true**;

**break**;

}

**if** (Math.*pow*(2,k)-k>array1.size()+1) {

flag3=**true**;

**break**;

}

k++;

}

**if** (flag3==**false**) HM.print("字符个数输入出错，请重新输入.\n");

**if** (flag3) {

H=HM.checkMatrix(k);

G=HM.creatMatrix(H, k);

HM.print("校验矩阵为：\n");

**for** (**int** i=0;i<H.length;i++) {

**for** (**int** j=0;j<H[i].length;j++)

System.***out***.print(" "+H[i][j]);

System.***out***.println();

}

HM.print("生成矩阵为：\n");

**for** (**int** i=0;i<G.length;i++) {

**for** (**int** j=0;j<G[i].length;j++)

System.***out***.print(" "+G[i][j]);

System.***out***.println();

}

HM.print("Hamming编码为\n");

**int**[] code=HM.HammingCoding(G, array1);

**for** (**int** i=0;i<code.length;i++)

HM.print(code[i]+" ");

**int** mm=(**int**)(Math.*pow*(2, k)-1);

**while** (flag4) {

HM.print("\n请输入"+mm+"位接收方接收的字符：");

**for** (**int** i=0;i<mm;i++) {

in=s.nextInt();

**if** ((in==1 || in==0)==**false**) {

HM.print("输入错误，请重新输入");

**break**;

}

**else** {

**if** (i==mm-1) flag4=**false**;

array2.add(in);

}

}

}

flag4=**true**;

HM.checkCorrectError(H, array2);

}

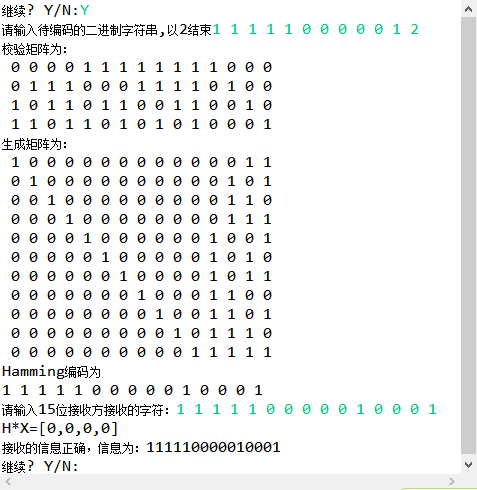
}

}

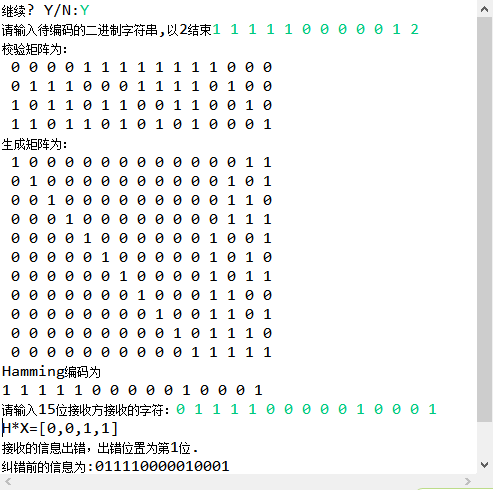
}

测试结果：

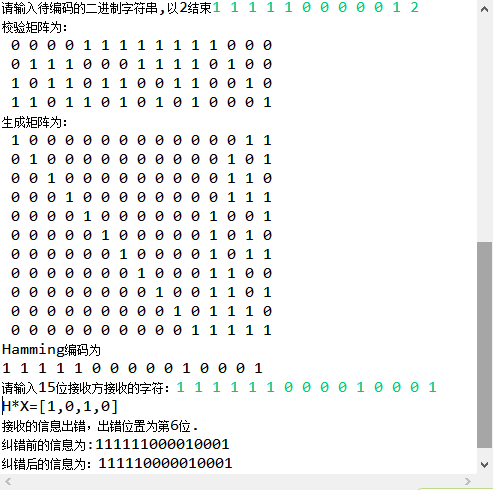
①当正确接收时



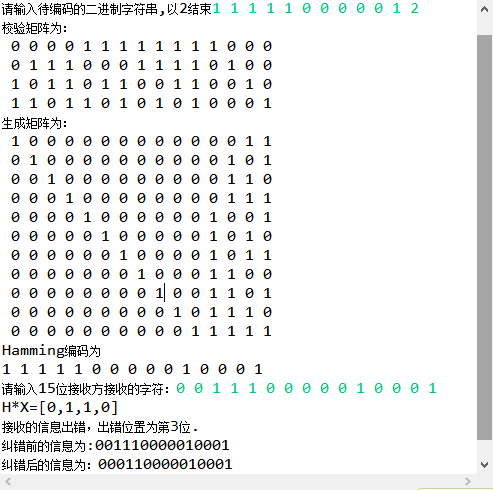
② 当第一位接收出错时：



③当第6接收出错时



④当第1,2位接收出错时



检测出有出错，但是没有正确给出出错位数，纠错也失败。

1. 以Hamming码为基础，设计McEliece密码体制：密钥对生成算法，加密算法，解密算法。

**ME.java**

**package** MC05;

**import** java.util.Scanner;

**import** java.util.Random;

/\*

\*

\* @see 实现ME

\*/

**public** **class** ME {

**static** **int** *k*, *m*, *n*;

Scanner sc;

**public** **static** **void** main(String[] args) {

ME MC = **new** ME();

MC.start(MC);

}

**public** **void** start(ME MC) {

System.***out***.print("请输入校验位数 k = ");

sc = **new** Scanner(System.***in***);

*k* = sc.nextInt();// 3

*n* = (**int**) Math.*pow*(2, *k*) - 1;// 7

*m* = *n* - *k*;// 4

HammingCode HMC = **new** HammingCode();

HMC.Create(HMC);// 得到H,G

Matrix S = **new** Matrix();

S = S.GenerateS();

Matrix P = **new** Matrix();

P = P.GenerateP();

Matrix G = **new** Matrix(*k*, *n*);

G.Mt = HMC.G;

Matrix G1 = **new** Matrix();

G1 = G1.MultipleMatrix(S, G);

G1 = G1.MultipleMatrix(G1, P);

System.***out***.println("公钥G1:");

**for** (**int** i = 0; i < *m*; i++) {

**for** (**int** j = 0; j < *n*; j++) {

System.***out***.print(G1.Mt[i][j] + " ");

}

System.***out***.println();

}

**int** send[] = **new** **int**[*m*];

// 输入字符进行编码

System.***out***.print("请输入需编码的字符（0、1）" + *m* + "个：");

**for** (**int** i = 0; i < *m*; i++) {

send[i] = sc.nextInt();

}

sc.close();

**int** receive[] = **this**.Encrypt(send,HMC, G1);

**int** send0[] =**this**.Decrypt(receive, P, S, HMC);

**for**(**int** i=0 ; i<send0.length ; i++){

**if**(send0[i]!=send[i]){

System.***out***.println("解密错误！");

**return**;

}

}

System.***out***.println("解密正确");

**return**;

}

**public** **int**[] Encrypt(**int** send[],HammingCode HMC, Matrix G1) {

**int** receive[] = **new** **int**[*n*];

**int** e;

Random rnd = **new** Random();

e = rnd.nextInt(*n*); //e:0~n-1

// 编码

receive = HMC.encode(G1.Mt, send);

// 加入噪声

receive[e] = (receive[e] + 1) % 2;

// 打印y=x\*G1+e

System.***out***.print("y=x\*G1+e:");

**for** (**int** i = 0; i < *n*; i++) {

System.***out***.print(receive[i] + " ");

}

System.***out***.println();

**return** receive;

}

**public** **int**[] Decrypt(**int** receive[], Matrix P, Matrix S, HammingCode HMC) {

**int**[] send = **new** **int**[*m*];

**int**[] y1 = **new** **int**[*n*];

P.Mt = P.Invert(P);

y1 = HMC.encode(P.Mt, receive);// y1=y\*p^-1

// 利用校验矩阵H求得y1的最近码字

**int** e1 = HMC.checkAndDecode(HMC.H, y1);//e1=H\*y1

y1[e1] = (y1[e1]+1)%2;

**for** (**int** i = 0; i < *n*; i++) {

//y1[i] = y1[i] ^ P.Mt[i][e1];

**if** (i < *m*) {

send[i] = y1[i];//取y1前m位放入send数组中

}

}

S.Mt = S.Invert(S); //S^-1

send = HMC.encode(S.Mt, send);//x0 = send \* S^-1

System.***out***.print("解密出的明文：");

**for** (**int** i = 0; i < *m*; i++) {

System.***out***.print(send[i]+" ");

}

**return** send;

}

**class** HammingCode {

**int**[][] H;

**int**[][] G;

/\*\*

\* **@param** start

\* Hamming码的具体实现

\* **@param** HMC

\* 类的实例

\*/

**public** **void** Create(HammingCode HMC) {

// Hamming 码的校验矩阵生成

H = HMC.CreateCheckArray();

// Hamming 码的生成矩阵生成

G = HMC.GenerateArray();

}

/\*\*

\* **@param** ToBinary

\* 校验矩阵 十进制数转为二进制矩阵

\* **@return** 返回校验矩阵

\*/

**public** **int**[][] CreateCheckArray() { // 二进制 校验矩阵

**int** m1 = 1;

**int** l = 0;

**int**[][] b = **new** **int**[*k*][*n*];

**for** (**int** i = 0; i < b.length; i++) {

**for** (**int** j = 0; j < b[i].length; j++) {

b[i][j] = 0;

}

}

**for** (**int** i = 1; i <= *n*; i++) { // 二进制

m1 = i;

**if** (i != 1 && i != 2 && i != 4 && i != 8) {

**for** (**int** j = *k* - 1; j >= 0; j--) {

b[j][l] = m1 % 2;

m1 = m1 / 2;

**if** (m1 < 0) {

**break**;

}

}

l++;

}

}

**for** (**int** j = 0; j < *k*; j++) { // 二进制

**for** (**int** i = *m*; i < *n*; i++) {

**if** (i - *m* == j) {

b[j][i] = 1;

}

}

}

System.***out***.println("(" + *n* + "," + *m* + ")HammingCode校验矩阵生成:");

System.***out***

.println("----------------------------------------------------------");

**for** (**int** i = 0; i < *k*; i++) {

**for** (**int** j = 0; j < *n*; j++) {

System.***out***.print(b[i][j] + " ");

}

System.***out***.println();

}

System.***out***

.println("----------------------------------------------------------");

**return** b;

}

/\*\*

\* **@param** Generate

\* 生成矩阵 十进制数转为二进制矩阵

\* **@return**

\*/

**public** **int**[][] GenerateArray() {// 生成矩阵 增加还没有做

**int** g[][] = **new** **int**[*k*][*m*];

**int** l = 0;

**int** m1;

**for** (**int** i = 1; i <= *n*; i++) { // 二进制

m1 = i;

**if** (i != 1 && i != 2 && i != 4 && i != 8) {

**for** (**int** j = *k* - 1; j >= 0; j--) {

g[j][l] = m1 % 2;

m1 = m1 / 2;

**if** (m1 < 0) {

**break**;

}

}

l++;

}

}

// G 生成

**int**[][] G = **new** **int**[*m*][*n*];

**for** (**int** i = 0; i < G.length; i++) {

**for** (**int** j = 0; j < G[i].length; j++) {

**if** (j == i) {

G[i][j] = 1;

} **else** {

G[i][j] = 0;

}

}

}

**for** (**int** j = *m*; j < *n*; j++) {

**for** (**int** i = 0; i < G.length; i++) {

G[i][j] = g[j - *m*][i];

}

}

System.***out***.println("(" + *n* + "," + *m* + ")HammingCode生成矩阵生成:");

System.***out***

.println("----------------------------------------------------------");

**for** (**int** i = 0; i < *m*; i++) {

**for** (**int** j = 0; j < *n*; j++) {

System.***out***.print(G[i][j] + " ");

}

System.***out***.println();

}

System.***out***

.println("----------------------------------------------------------");

**return** G;

}

/\*\*

\* **@param** encode

\* 用生成矩阵编码

\* **@param** G

\* 代表生成矩阵

\* **@param** send

\* 输入的字符诸位 1、0 等

\* **@return**

\*/

**public** **int**[] encode(**int**[][] G, **int**[] send) { // 利用生成矩阵 编码

**int**[] newArray = **new** **int**[G[1].length]; // 得到X\*G的一个矩阵

**for** (**int** i = 0; i < newArray.length; i++) {

newArray[i] = 0;

**for** (**int** j = 0; j < G.length; j++)

newArray[i] = (newArray[i]+send[j] \* G[j][i])%2;

}

**return** newArray;

}

/\*\*

\*

\* **@param** H

\* 校验矩阵错误

\* **@param** receive

\* 收到的信息

\* **@return** 返回错误位置

\*/

**public** **int** checkAndDecode(**int**[][] H, **int**[] receive) { // 检测错误

**int**[] e = **new** **int**[H.length]; // H\*x得到

**int** flag ;

**for** (**int** i = 0; i < e.length; i++) { // 得到Hx的一个矩阵

e[i] = 0;

**for** (**int** j = 0; j < receive.length; j++) {

e[i] = e[i] ^ (receive[j] & H[i][j]);

}

}

// 计算H\*x

// flag 得到位错的下标

flag = -1;

**for** (**int** i = 0; i < receive.length; i++) {

**int** k = 0;

**for** (**int** j = 0; j < e.length; j++) {

**if**(e[j] == H[j][i]){

k++;

} **else**{

**break**;

}

}

**if**(k == e.length){

flag = i;

**break**;

}

}

**return** flag;

}

}

**class** Matrix {

**int** r, c;

**int**[][] Mt;

Matrix() {

}

Matrix(**int** x, **int** y) {

r = x;

c = y;

Mt = **new** **int**[r][c];

**for** (**int** i = 0; i < r; i++) {

**for** (**int** j = 0; j < c; j++) {

Mt[i][j] = 0;

}

}

}

// 求行列式

**public** **int** Det(Matrix M) {

**int** switchtime = 0, result;

**int** t[][] = **new** **int**[M.c][M.c];

**for** (**int** i = 0; i < M.c; i++) {

**for** (**int** j = 0; j < M.c; j++) {

t[i][j] = M.Mt[i][j];

}

}

**for** (**int** row = 0; row < M.c - 1; row++) {

**int** nextrow = row + 1;

**if** (t[row][row] == 0) {

**while** (t[nextrow][row] == 0 && nextrow < M.c - 1) {

nextrow++;

}

**if** (nextrow == M.c - 1) {

**return** 0;

} **else** {

switchtime++;

**for** (**int** col = 0; col < M.c; col++) {

**int** temp1 = t[row][col];

t[row][col] = t[nextrow][col];

t[nextrow][col] = temp1;

}

}

}

**for** (nextrow = row + 1; nextrow < M.c; nextrow++) {

**if** (t[nextrow][row] != 0) {

**int** temp2 = t[nextrow][row] / t[row][row];

**for** (**int** col = row; col < M.c; col++) {

t[nextrow][col] += -temp2 \* t[row][col];

}

} **else** {

nextrow++;

}

}

}

result = 1;

**for** (**int** row = 0; row < M.c; row++) {

result \*= t[row][row];

}

**if** (switchtime % 2 == 1) {

**return** -result;

} **else** {

**return** result;

}

}

// 矩阵乘积

**public** Matrix MultipleMatrix(Matrix A, Matrix B) {

Matrix C = **new** Matrix(A.r, B.c);

**for** (**int** i = 0; i < A.r; i++) {

**for** (**int** j = 0; j < B.c; j++) {

**for** (**int** k = 0; k < A.c; k++) {

C.Mt[i][j] += A.Mt[i][k] \* B.Mt[k][j];

}

C.Mt[i][j] = C.Mt[i][j] % 2;

}

}

**return** C;

}

// 高斯若而当列主元求逆矩阵

**public** **int**[][] Max(**int**[][] a, **int** k) {

**int** max, tmp;

**int** m, j;

max = a[k][k];

m = k;

**for** (j = k + 1; j < a.length; j++) {

**if** (Math.*abs*(a[j][k]) > Math.*abs*(max)) {

max = a[j][k];

m = j;

}

}

**for** (j = k; j < a[0].length; j++) {

tmp = a[k][j];

a[k][j] = a[m][j];

a[m][j] = tmp;

}

**return** a;

}

**public** **void** GaussJordanlie(Matrix a) {

**int** i, j;

**for** (**int** k1 = 0; k1 < a.r; k1++) {

a.Mt = Max(a.Mt, k1);

**for** (j = a.c - 1; j >= k1; j--)

a.Mt[k1][j] = a.Mt[k1][j] / a.Mt[k1][k1];

**for** (i = 0; i < a.r; i++) {

**if** (i == k1)

**continue**;

**for** (j = k1 + 1; j < a.c; j++)

a.Mt[i][j] = a.Mt[i][j] - a.Mt[i][k1] \* a.Mt[k1][j];

a.Mt[i][k1] = 0;

}

}

}

// 求逆矩阵

**public** **int**[][] Invert(Matrix a) {

**int** c = 2 \* a.r;

Matrix b = **new** Matrix(a.r, c);

**for** (**int** i = 0; i < a.r; i++) {

**for** (**int** j = 0; j < c; j++) {

**if** (j < a.r) {

b.Mt[i][j] = a.Mt[i][j];

} **else** {

**if** (j == a.r + i) {

b.Mt[i][j] = 1;

} **else** {

b.Mt[i][j] = 0;

}

}

}

}

GaussJordanlie(b);

**for** (**int** i = 0; i < a.r; i++) {

**for** (**int** j = a.r; j < c; j++) {

a.Mt[i][j - a.r] = b.Mt[i][j] % 2;

**if** (a.Mt[i][j - a.r] == -1) {

a.Mt[i][j - a.r] = 1;

}

}

}

**return** a.Mt;

}

// 随机生成S

**public** Matrix GenerateS() {

**long** seed = 0;

Matrix S = **new** Matrix(*m*, *m*);

**do** {

Random rnd = **new** Random(seed);

**for** (**int** i = 0; i < *m*; i++) {

**for** (**int** j = 0; j < *m*; j++) {

S.Mt[i][j] = rnd.nextInt(2);

}

}

seed = seed + 1;

} **while** (Det(S) == 0);

System.***out***.println("S生成:");

**for** (**int** i = 0; i < *m*; i++) {

**for** (**int** j = 0; j < *m*; j++) {

System.***out***.print(S.Mt[i][j] + " ");

}

System.***out***.println();

}

System.***out***

.println("----------------------------------------------------------");

**return** S;

}

// 随机生成P；

**public** Matrix GenerateP() {

Matrix P = **new** Matrix(*n*, *n*);

**int** t[] = **new** **int**[*n*];

**for** (**int** i = 0; i < *n*; i++) {

t[i] = 0;

}

Random rnd = **new** Random();

**for** (**int** j = 0; j < *n*; j++) {

**int** k = rnd.nextInt(*n*);

**while** (t[k] != 0) {

k++;

**if** (k == *n*) {

k = 0;

}

}

P.Mt[k][j] = 1;

t[k] = 1;

}

System.***out***.println("P生成:");

**for** (**int** i = 0; i < *n*; i++) {

**for** (**int** j = 0; j < *n*; j++) {

System.***out***.print(P.Mt[i][j] + " ");

}

System.***out***.println();

}

System.***out***

.println("----------------------------------------------------------");

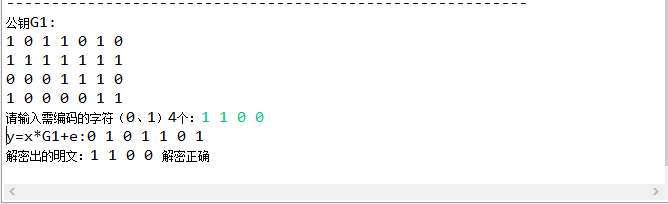
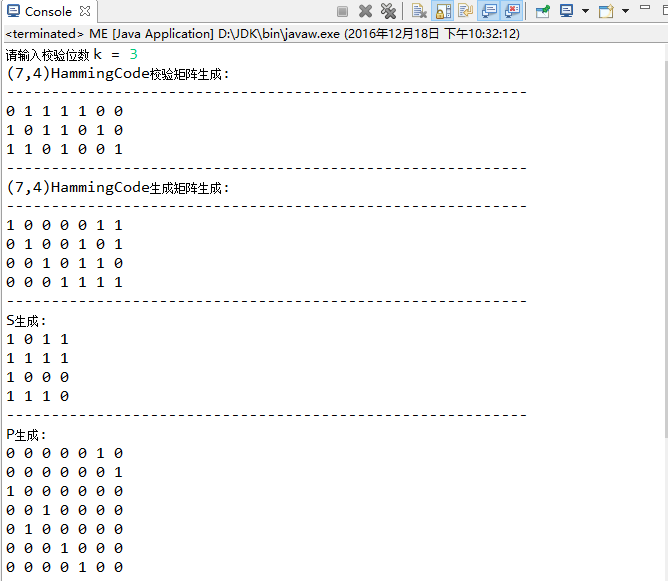
**return** P;

}

}

}

测试结果：



3. 分析以Hamming码为基础的纠错密码体制的局限性。

由于Hamming码纠错能力有限，只能正确纠正一个比特位置上的错误，所以这次试验只是用来模拟McEliece公钥密码体制的相关过程，不具有实用性。并且查阅资料以后，对McEliece有了更多了解。虽然比较简单容易实现，但也存在不容忽视的弊端，比如

1）公钥G1的密钥空间太大。如果使用Goppa码，公钥将有219 个bit密钥空间太大。

2）密文比明文要大得多。这将导致传输中的错误发生的可能性增加。在构造加密体制时应该结合各种差错控制机制以保证加密信息的可靠传输，否则可能无法解密。

1. 通过实验说明，纠错密码学中的基础纠错码应该具备哪些要点才能保证密码体制的安全性。

如果纠错码构造公钥密码体制，则要求码不存在快速的译码算法，而只能采取穷搜索或其他译码复杂性极高的通用密码算法。

5. 分析Goppa码用于纠错密码体制有何优势？

Goppa码最主要优点是它的某些子类能达到香农信道编码定理所给出的性能，并且具有快速译码算法。特别是他的不等价码类数目很大。

一旦码的Goppa多项式及位置集L确定，码也就确定了。因此Goppa码的非等价码类的个数决定于既约多项式的个数，也就是GF（q）中既约多项式的数目。由有限域理论可知，GF（q）上m次既约多项式的数目是



其中为Mobius函数。可知，既约多项式的个数随m指数增长。所以对同一次数m，可以构造非常多的不等价既约Goppa码，事实上码长为2m，纠错t个错误的不同二进制即约Goppa码的数目，由上式可得，大约为



这样就为用Goppa码构造公钥密码体制创造了必要条件。

**五、实验体会**

**（请认真填写自己的真实体会）**

如果纠错码构造公钥密码体制，则要求码不存在快速的译码算法，而只能采取穷搜索或其他译码复杂性极高的通用密码算法。因此在差错控制系统中应用纠错码以提高通信可靠性，与在密码中应用纠错码构造某些密码体制，这二者对于纠错码的译码时截然相反的。

**六、参考文献**

1. 课程群中共享的McEliece密码体制资料

2. 信息论基础实验的Hamming码实验

3. Goppa码的参考文献如《代数几何码》方面的参考文献

4.王新梅，马文平，武传坤，纠错密码理论，北京：人民邮电出版社，2001

**（请实验者列出各类参考文献的具体出处）**