工科高等代数II应用案例

题目：矩阵算法在信息加密中的实现

小组成员及专业班级：

组长：

成员分工及具体贡献：

目录

矩阵算法在信息加密中的实现

1. 问题描述

在当今这个数字化时代，信息已经成为了一种极其重要的资源，它不仅关系到个人的日常生活，也直接影响着企业的运营和国家的发展. 信息加密是保护个人和国家信息安全的常用手段，人们在设定的秘钥下将信息进行加密传输，保证了传输交换过程中的安全性和隐秘性.

在本次建模中，我们结合矩阵的相关知识和希尔密码作为基础，设计出了矩阵加密的算法，并通过 java 程序模拟了整个加密过程.

2. 解决问题的思路及方法

2.1 建模与求解

2. 1.1 模型的建立

希尔密码是利用矩阵乘法来实现信息变换的一种加密方式，在希尔密码体制中，26 个英文字母从 A 至 Z 依此被赋予了 0 至 25 这 26 个整数，将需要加密的字符串按每 n 个字符一组构成一个 n 维向量，然后与一个 n 阶方阵相乘得到密文. 以上说明，希尔密码是在模 26 的剩余类环上进行运算的[1]. 因此，我们按照如下步骤设计程序，完成建模：

（1）获取一串长度为 n 的字符，将字符按照字母表顺序转化成 $x_1, x_2, \dots, x_n$ 这 n 个分布在 0~25 的正整数；

（2）设计符合条件的秘钥矩阵，记为 $\boldsymbol{P}$ 矩阵，$\boldsymbol{P}$ 矩阵为方阵并且阶数为 r，将步骤(1)中得到的 n 个正整数按照 r 个一组进行拆分，可以得到 $t = \left[\frac{n}{r}\right]$ 组 $1 \times r$ 的列向量 $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \dots, \boldsymbol{\alpha}_t$；

（3）对于剩下的 $n - \left[\frac{n}{r}\right] \cdot r$ 个元素也分配一个新的 $1 \times r$ 的列向量中，余下的空位用固定字母 X 进行填充. 这样，我们就保证对于任何长度的字符串，都可以使用同一个矩阵进行加密.

（4）对于得到的 $t+1$ 个列向量，每个列向量 $\boldsymbol{\alpha}_i$ 与秘钥矩阵 $\boldsymbol{P}$ 相乘，得到矩阵 $\boldsymbol{\alpha}_i\boldsymbol{P}$ 依然为一个 $1 \times r$ 的列向量 $\boldsymbol{\beta}_i$. 对于 $\boldsymbol{\beta}_i$ 的每一个元素取模 26 的余数，这可以保证 $\boldsymbol{\beta}_i$ 最终的每一个元素落在 0~25 区间内，再根据 $\boldsymbol{\beta}_i$ 的各个元素找出对应的英文字母. 这样，我们就可以将任意长度的文字加密得到一串新的字符串，新的字符串依然由 A~Z 这 26 个字母构成.

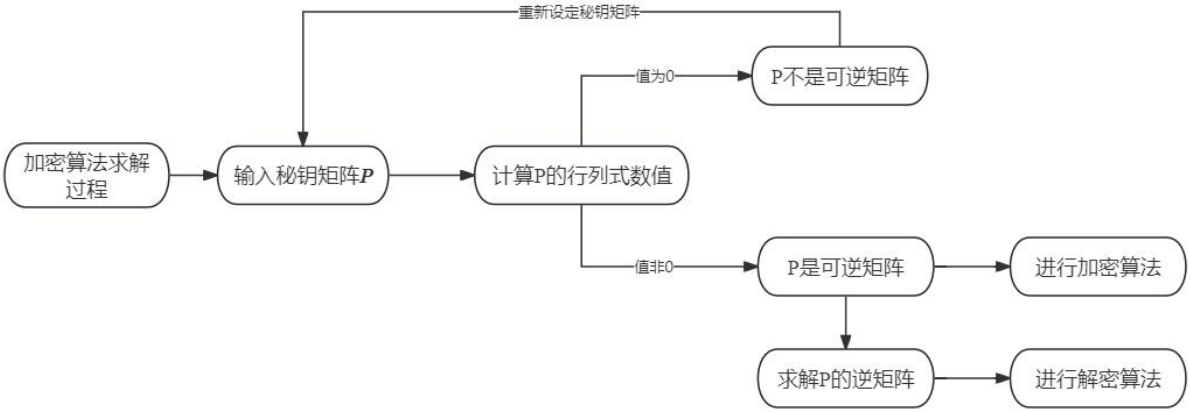
（5）对加密后的文字进行解密，即已知 $\boldsymbol{P}$ 和 $\boldsymbol{\beta}_i$ 的条件下求出对应的 $\boldsymbol{\alpha}_i$，由于 $\boldsymbol{\alpha}_i\boldsymbol{P} = \boldsymbol{\beta}_i$，因此

$$\boldsymbol{\alpha}_i = \boldsymbol{\beta}_i \boldsymbol{P}^{-1} \qquad (2.1.1)$$

显然，若要使(2.1.1)成立，必要条件为 $\boldsymbol{P}$ 为可逆矩阵. 因此，我们需要通过判断 $\boldsymbol{P}$ 的行列式值是否非零，这样才能保证加密算法的正常计算.

2. 1.2 模型的求解

模型的求解过程图如下：



在计算 ***P*** 的行列式数值过程中，我们按照行列式按第一行展开的计算方式，将 *n* 阶行列式 ***P*** 进行降阶处理得到 *n* 个 *n* - 1 阶行列式；再将每个 *n* - 1 阶行列式第一行展开的计算方式得到若干 *n* - 2 阶行列式. 依次循环，直到得到二阶行列式，在计算得到的若干二阶行列式之和，即可得到 ***P*** 的行列式数值. 在 java 程序中，我们采用递归算法实现：

```
public int determinant(int[][] matrix) {
    int n = matrix.length;
    if (n == 1) {
        return matrix[0][0];
    }
    if (n == 2) {
        return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
    }
    int det = 0;
    for (int p = 0; p < n; p++) {
        int[][] submatrix = new int[n - 1][n - 1];
        for (int i = 1; i < n; i++) {
            int subRow = i - 1;
            for (int j = 0; j < n; j++) {
                if (j != p) {
                    submatrix[subRow][j < p ? j : j - 1] = matrix[i][j];
                }
            }
        }
        det += matrix[0][p] * (p % 2 == 0 ? 1 : -1) * determinant(submatrix);
    }
    return det;
}
```

于是，我们便可以通过 ***P*** 的行列式数值确认 ***P*** 是否可逆. 一旦检测到行列式数值为 0，就要抛出异常，警告用户秘钥设计失败，需要再次设计新的秘钥矩阵.

在求 $\boldsymbol{P}$ 的逆矩阵过程中，我们依据公式(2.2.1)

$$\boldsymbol{P}^{-1} = |\boldsymbol{P}|^{-1}\boldsymbol{P}^{*} \qquad (2.2.1)$$

先计算出 $\boldsymbol{P}$ 的伴随矩阵，再结合上一步计算得到的 $\boldsymbol{P}$ 行列式数值即可获得 $\boldsymbol{P}$ 的逆矩阵. 我们先根据伴随矩阵的定义，结合上一步的行列式求法，我们可以写出 $\boldsymbol{P}$ 的伴随矩阵计算函数，其中*表达式就调用了计算行列式的函数 **int** *determinant* (**int**[][] *matrix*).

```
public int[][] adjugate(int[][] matrix) {
    int n = matrix.length;
    int[][] adj = new int[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            int[][] submatrix = new int[n - 1][n - 1];
            for (int x = 0; x < n; x++) {
                for (int y = 0; y < n; y++) {
                    if (x != i && y != j) {
                        submatrix[x < i ? x : x - 1][y < j ? y : y - 1] = matrix[x][y];
                    }
                }
            }
            adj[i][j] = (int)(Math.pow(-1, i + j) * determinant(submatrix));       // *
        }
    }
    return adj;
}
```

将得到的伴随矩阵 $\boldsymbol{P}^{*}$ 数乘$|\boldsymbol{P}|^{-1}$即可得到 $\boldsymbol{P}$ 的逆矩阵. 这样，我们就可以针对同一个秘钥矩阵进行加密运算和解密运算. 同时，我们也得到了加密算法的副产品——逆矩阵，我们在主函数中设计了一个接口专门处理逆矩阵的计算问题.

在确定秘钥矩阵合理的情况下，我们完成加密算法. 加密算法的核心是计算两个矩阵的乘法，我们参考数据与算法的相关书籍，设计了矩阵乘法的计算函数[2]：

```
private int[] encryptBlock(String block) {
    int[] encryptedBlock = new int[matrixSize];
    for (int row = 0; row < matrixSize; row++) {
        for (int col = 0; col < matrixSize; col++) {
            encryptedBlock[row] += keyMatrix[row][col] * (block.charAt(col) - 'A');
        }
        encryptedBlock[row] = (encryptedBlock[row] % 26 + 26) % 26;
                                                        // 确保结果在 0-25 之间
    }
    return encryptedBlock;
}
```

在调用这个函数之前，我们会先对字符串按照既定设计分组构成列向量，并将列向量传入到矩阵乘法函数中即可完成加密.

2.2 高等代数知识点

涉及的高等代数知识点如下：

（1）n 阶行列式的计算；

（2）n 阶伴随矩阵，可逆矩阵的定义和计算；

（3）线性方程 $\boldsymbol{Ax}=\boldsymbol{b}$ 的求解；

（4）可逆矩阵的判定定理；

（5）矩阵乘法的条件和计算公式.

（6）

3.结果演示和结论

测试案例 1：秘钥矩阵为$\begin{pmatrix}1&4&9\\0&1&4\\0&0&1\end{pmatrix}$，加密的文字为 *ABCDEF*.

```
Please enter the size of the key matrix (for example, 3 represents a 3x3 matrix):
3
Please enter the key matrix (input for 3 each row):
1 4 9
0 1 4
0 0 1
```

现将加密的文字转化成 0~25 的整数，依次为(0, 1, 2, 3, 4, 5). 由于矩阵的阶数为 3，因此每次对三个字母构成的列向量进行矩阵乘法加密，加密运算为：

$$\begin{pmatrix}1&4&9\\0&1&4\\0&0&1\end{pmatrix}\begin{pmatrix}0\\1\\2\end{pmatrix}=\begin{pmatrix}22\\9\\2\end{pmatrix}\quad,\quad\begin{pmatrix}1&4&9\\0&1&4\\0&0&1\end{pmatrix}\begin{pmatrix}3\\4\\5\end{pmatrix}=\begin{pmatrix}64\\24\\5\end{pmatrix}\qquad(3.1.1)$$

按照(3. 1. 1)所示的计算结果，对应的秘文依次为字母表中的第 22,9,2,12,24,5 号的字母，由于我们的编号从 0 开始，即 A 为 0 号字母，以此类推，因此最终的字母为 W, J, C, M, Y, F. 对比程序运行结果可知，程序运行正确.

```
Please enter the English sentence that needs to be encrypted:
ABCDEF
The encrypted ciphertext is:WJCMYF
```

测试案例 2：计算矩阵$\begin{pmatrix}1&2&3\\2&2&1\\3&4&3\end{pmatrix}$的逆矩阵.

答案为$\begin{pmatrix}1&3&-2\\-3/2&-3&5/2\\1&1&-1\end{pmatrix}$. 由于算法需要，这里我们取它的逆矩阵的转置矩阵进行核验.

```
Input the number of line:
3
Input the elements:
1 2 3
2 2 1
3 4 3
The determinant of the matrix is:2
```

```
The inverse matrix is:
1   -3/2    1
3   -3  1
-2  5/2 -1
```

4.小结和问题引申

根据两个测试案例，我们可以暂且认为程序可以正确进行计算矩阵的逆阵（结果以其转置矩阵的形式展现）和进行明文加密. 我们后续还可以编写更多测试案例，这样可以帮助程序的完善. 尽管在矩阵求逆的环节中，我们加入了对分数的输出处理，但是用户输入的矩阵必须各个元素均为整数，在实际问题中会收到影响. 另外，我们在设定秘钥矩阵的过程中，必须要求用户输入的秘钥矩阵 $\boldsymbol{P}$ 的行列式值为 1，这会极大约束秘钥的个数.

由于时间紧张，我们没有给出完整的解密类，仅仅写出了解密类的关键函数，下面我们会在这些函数的基础上完善我们的代码，实现加密和解密的双向支持.

```
public String decrypt(String ciphertext) {                          //to do
    MatrixKey matrixKey = new MatrixKey();
    int[][] inverseMatrix = matrixKey.DecryptionInverse(keyMatrix);

    StringBuilder plaintext = new StringBuilder();
    for (int i = 0; i < ciphertext.length(); i += matrixSize) {
        String block = ciphertext.substring(i, i + matrixSize);
        int[] decryptedBlock = decryptBlock(block, inverseMatrix);
        for (int value : decryptedBlock) {
            plaintext.append((char) (value + 'A'));
        }
    }
    // 移除由于加密添加的填充字符
    return plaintext.toString().replaceAll("X+$", "");
}

private int[] decryptBlock(String block, int[][] inverseMatrix) {
    int[] decryptedBlock = new int[matrixSize];
    for (int row = 0; row < matrixSize; row++) {
        for (int col = 0; col < matrixSize; col++) {
            decryptedBlock[row] += inverseMatrix[row][col] * (block.charAt(col) - 'A');
        }
        decryptedBlock[row] = (decryptedBlock[row] % 26 + 26) % 26;
    }
    return decryptedBlock;
}
```

正在完善的解密类

5.参考文献

[1] 孙兵，刘国强，海昕. 线性代数教学中的希尔密码案例设计. 大学数学. 2023 ,39 (01)；
[2] 吴及，陈健生，白铂. 《数据与算法》. 清华大学出版社.

6. 附录：程序源代码

（1）矩阵计算类：包括计算行列式，伴随矩阵和可逆矩阵

```
public class MatrixKey
{
    public int determinant(int[][] matrix) {
        int n = matrix.length;
        if (n == 1) {
            return matrix[0][0];
        }
        if (n == 2) {
            return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
        }
        int det = 0;
        for (int p = 0; p < n; p++) {
            int[][] submatrix = new int[n - 1][n - 1];
            for (int i = 1; i < n; i++) {
                int subRow = i - 1;
                for (int j = 0; j < n; j++) {
                    if (j != p) {
                        submatrix[subRow][j < p ? j : j - 1] = matrix[i][j];
                    }
                }
            }
            det += matrix[0][p] * (p % 2 == 0 ? 1 : -1) * determinant(submatrix);
        }
        return det;
    }
    public int[][] adjugate(int[][] matrix) {                    //Programmed by Lzc
        int n = matrix.length;
        int[][] adj = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                int[][] submatrix = new int[n - 1][n - 1];
                for (int x = 0; x < n; x++) {
                    for (int y = 0; y < n; y++) {
                        if (x != i && y != j) {
                            submatrix[x < i ? x : x - 1][y < j ? y : y - 1] = matrix[x][y];
                        }
                    }
                }
                adj[i][j] = (int)(Math.pow(-1, i + j) * determinant(submatrix));
            }
        }
```

```
        return adj;
    }
    public boolean isMatrixInvertible(int[][] matrix) {
        return determinant(matrix) != 0;
    }
    public static int gcd(int m, int n) {
        int mm = m;
        int nn = n;
        if (mm < nn) {
            int c = mm;
            mm = nn;
            nn = c;
        }
        int w = 1;
        while (w != 0) {
            w = mm % nn;
            mm = nn;
            nn = w;
        }
        return mm;
    }
    public void printAnswer(int m, int n)
    {
        if (n * m < 0) {
            System.out.printf("-");
            printAnswer(Math.abs(m), Math.abs(n));
            return;
        }
        MatrixKey matrix = new MatrixKey();
        if (m == 1)
            System.out.print(n + "\t");
        else if (n % m == 0)
            System.out.print(n / m + "\t");
        else
            System.out.print((n / matrix.gcd(m, n)) + "/" + (m / matrix.gcd(m, n)) + "\t");
    }
public int[][] inverse(int[][] matrix) {
        if (!isMatrixInvertible(matrix)) {
            throw new IllegalArgumentException("Matrix is not invertible");
        }
        int det = determinant(matrix);
        int[][] adj = adjugate(matrix);
        int[][] inv = new int[adj.length][adj[0].length];
        for (int i = 0; i < adj.length; i++) {
```

```
                for (int j = 0; j < adj[i].length; j++) {
                        printAnswer((int)det,(int)adj[i][j]);
                }
                System.out.println();
        }
        return inv;
}
public void print(int[][] matrix) {
        int n = matrix.length;
        for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                        System.out.printf(matrix[i][j] + " ");
                }
                System.out.println();
        }
}

public int[][] DecryptionInverse(int[][] matrix) {
        if (!isMatrixInvertible(matrix)) {
                throw new IllegalArgumentException("Matrix is not invertible");
        }
        int det = determinant(matrix);
        int[][] adj = adjugate(matrix);
        int[][] inv = new int[adj.length][adj[0].length];
        int detInv = modInverse(det, 26);

        for (int i = 0; i < adj.length; i++) {
                for (int j = 0; j < adj[i].length; j++) {
                        inv[i][j] = (int) ((adj[i][j] * detInv) % 26);
                }
        }
        return inv;
}
private int modInverse(int a, int mod) {
        int m0 = mod, t, q;
        int x0 = 0, x1 = 1;

        if (mod == 1) return 0;

        while (a > 1) {
                q = a / mod;
                t = mod;
                mod = a % mod;
                a = t;
```

```
            t = x0;
            x0 = x1 - q * x0;
            x1 = t;
        }
        if (x1 < 0) x1 += m0;
        return x1;
    }
}
```

（2）信息加密类：将一串英文字母按照设计的算法进行加密

```
public class MatrixEncryption
{
    private int[][] keyMatrix; // 秘钥矩阵
    private int matrixSize; // 矩阵的大小
    // 构造函数，初始化秘钥矩阵
    public MatrixEncryption(int[][] keyMatrix)
    {
        this.keyMatrix = keyMatrix;
        this.matrixSize = keyMatrix.length; // 是一个方阵
    }
    // 加密方法
    public String encrypt(String plaintext) {
        plaintext = removeSpacesAndToUpper(plaintext);
        int padding = calculatePadding(plaintext);
        plaintext = padText(plaintext, padding);
        return processEncryption(plaintext);
    }
    private String removeSpacesAndToUpper(String text) {
        return text.replaceAll("\\s", "").toUpperCase();
    }
    private int calculatePadding(String text) {
        return matrixSize - (text.length() % matrixSize);
    }
    private String padText(String text, int padding) {
        if (padding != matrixSize) {
            return text + "X".repeat(padding);
        }
        return text;
    }
    private String processEncryption(String cleanedText) {
        StringBuilder ciphertext = new StringBuilder();
        for (int i = 0; i < cleanedText.length(); i += matrixSize) {
            String block = cleanedText.substring(i, i + matrixSize);
            int[] encryptedBlock = encryptBlock(block);
            for (int value : encryptedBlock) {
```

```
                    ciphertext.append((char) (value + 'A'));
                }
            }
            return ciphertext.toString();
        }
    private int[] encryptBlock(String block) {
            int[] encryptedBlock = new int[matrixSize];
            for (int row = 0; row < matrixSize; row++) {
                for (int col = 0; col < matrixSize; col++) {
                    encryptedBlock[row] += keyMatrix[row][col] * (block.charAt(col) - 'A');
                }
                encryptedBlock[row] = (encryptedBlock[row] % 26 + 26) % 26;
            }
            return encryptedBlock;
        }
}
```

（3）主函数类：将会满足以下三个功能

1. 输入 n 阶方阵 $\boldsymbol{A}$，计算 $\boldsymbol{A}$ 的行列式、伴随矩阵和可逆矩阵；
2. 设定可逆方阵 $\boldsymbol{A}$，输入一段英文，得到加密算法后秘文并输出；
3. 在设定的可逆方阵 $\boldsymbol{A}$ 下，输入加密后的秘文，并输出明文.

其中，功能 3 正在完善当中.

```
import java.util.Scanner;
public class MainFunction {
    public static void matrixCalculator() {
        Scanner scanner = new Scanner(System.in);
        MatrixKey matrixKey = new MatrixKey();
        while (true) {
            System.out.println("Input the number of line:");
            int lineOfMatrix = scanner.nextInt();
            if (lineOfMatrix != 0) {
                System.out.println("Input the elements:");
                int[][] matrix = new int[lineOfMatrix][lineOfMatrix];
                for (int i = 0; i < lineOfMatrix; i++) {
                    for (int j = 0; j < lineOfMatrix; j++) {
                        matrix[i][j] = scanner.nextInt();
                    }
                }
                System.out.println("The determinant of the matrix is:" + matrixKey.determinant(matrix));
                System.out.println("The adjoint matrix is:");
                matrixKey.print(matrixKey.adjugate(matrix));
                System.out.println("The inverse matrix is:");
                int[][] inverseMatrix = matrixKey.inverse(matrix);
                matrixKey.print(inverseMatrix);
            } else {
```

```
                    System.out.println("Invalid Input.");
                    break;
                }
            }
        }
        public static void encryptionSentence() {
            Scanner scanner = new Scanner(System.in);
            System.out.println("Please enter the size of the key matrix (for example, 3 represents a 3x3
matrix):");
            int size = scanner.nextInt();
            scanner.nextLine(); // 消耗掉换行符
            int[][] keyMatrix = new int[size][size];
            MatrixKey matrixKey = new MatrixKey();

            int matrixValue = 0;
            while (matrixValue == 0) {
                System.out.println("Please enter the key matrix (input for " + size + " each row):");
                for (int i = 0; i < size; i++) {
                    for (int j = 0; j < size; j++) {
                        keyMatrix[i][j] = scanner.nextInt();
                    }
                }
                matrixValue = matrixKey.determinant(keyMatrix);
                if (matrixValue == 0) {
                    System.out.println("Invalid Matrix! The determinant is zero. Please input again.");
                }
            }
            System.out.println("The password is established successfully.");
            System.out.println("(1) Input 1 to enter the English sentence that needs to be encrypted.");
            System.out.println("(2) Input 2 to enter the English sentence that needs to be decrypted.");
            System.out.println("(3) Input 0 to back the main menu.");
            int operation;
            while (true) {
                operation = scanner.nextInt();
                if (operation == 1) {
                    scanner.nextLine(); // 消耗掉换行符
                    System.out.println("Please enter the English sentence that needs to be encrypted:");
                    String plaintext = scanner.nextLine();
                    MatrixEncryption encryption = new MatrixEncryption(keyMatrix);
                    String ciphertext = encryption.encrypt(plaintext);
                    System.out.println("The encrypted ciphertext is:" + ciphertext);
                    if (plaintext.equals("Exit")) break;
                }
```

```
                if (operation == 2) {
                    //即第 3 个功能，to do.
                }
                if (operation == 0) break;
            }
        }

        public static void main(String[] args) {
            Scanner scanner = new Scanner(System.in);
            System.out.println("(1) Input 1 to enter the matrix Calculator.");
            System.out.println("(2) Input 2 to enter the encryption software.");
            System.out.println("(3) Input 0 to exit the system.");
            int operation;
            while (true) {
                operation = scanner.nextInt();
                if (operation == 1) matrixCalculator();
                if (operation == 2) encryptionSentence();
                if (operation == 0) System.exit(0);
                else System.out.println("Invalid Operation! Please try again.");
            }
        }
}
```