

信息与软件工程学院上机实验报告

(第 1 次)

一、实验名称

实验一：古典密码算法

二、实验目的及要求

1. 实验目的

通过编程实现代替密码算法和置换密码算法，加深对古典密码体制的了解。

2. 实验要求

古典密码算法曾被广泛应用，大都比较简单，使用手工和机械操作来实现加密和解密。

1、代替密码

代替密码算法的原理是使用代替法进行加密，就是对明文中的字符用其他字符代替后形成密文。例如，明文字母 a, b, c, d ，用 d, e, f, g 做对应代替后形成密文。代替密码包括多种类型，如单表代替密码，多表代替密码，多字母代替密码等。

试编程实现一种典型的单表代替密码—凯撒（Caesar）密码。它的加密方法是将明文中的每个字母用此字符在字母表中后面的第 k 个字母代替。它的加密过程可以表示为下面的函数：

$$E(k) = (m + k) \bmod n$$

其中， m 为明文字母在字母表中的位置数， n 为字母表中的字母个数， k 为密钥， $E(k)$ 为密文字母在字母表中对应的位置数。解密过程类推。

2、置换密码

置换密码算法的原理是不改变明文字符，只将字符在明文中的排列顺序改

变，从而实现明文信息的加密。置换密码也叫换位密码。

试编程实现矩阵置换密码。它的加密方法是将明文中的字母按照给定的顺序安排在一个矩阵中，然后根据密钥提供的顺序重新组合矩阵中的字母，形成密文。例如，明文为 **attack begins at five**，密钥为 **cipher**，将明文按照每行 6 个字母的形式排在矩阵中，如下形式：

```
a t t a c k
b e g i n s
a t f i v e
```

根据密钥 **cipher** 中各字母在字母表中出现的先后顺序，给定一个置换：

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 4 & 5 & 3 & 2 & 6 \end{pmatrix}$$

根据上面的置换，将原有矩阵中的字母按照第 1、4、5、3、2、6 的顺序排列，则有下列形式：

```
a a c t t k
b i n g e s
a i v f t e
```

从而得到密文：**aacttkbingesaivfte**

解密过程类推。

三、实验环境

1. 操作系统：macOS / Linux
2. 编程语言：Python 3.13
3. 环境管理：uv
4. 开发工具：Visual Studio Code

四、实验设计

1. 实验步骤

程序整体采用菜单式交互：先选择密码算法，再选择加密或解密，最后输入数据并输出结果。整个实现分为两个部分：一部分负责交互流程与输入检查，另一部分负责凯撒密码和矩阵置换密码的具体算法实现。

1.1 设计思想

具体设计思想如下：

1. 对凯撒密码，只处理英文字母，其余字符保持不变，这样既符合算法原理，也方便处理包含空格和标点的输入。
2. 对矩阵置换密码，先根据字符串密钥生成列置换序列，再将文本按固定列数分行，最后完成列重排或逆重排。
3. 中间矩阵和置换序列会被直接输出，便于观察算法执行过程，也便于实验报告展示。

1.2 实现流程

程序的整体执行流程如图所示。

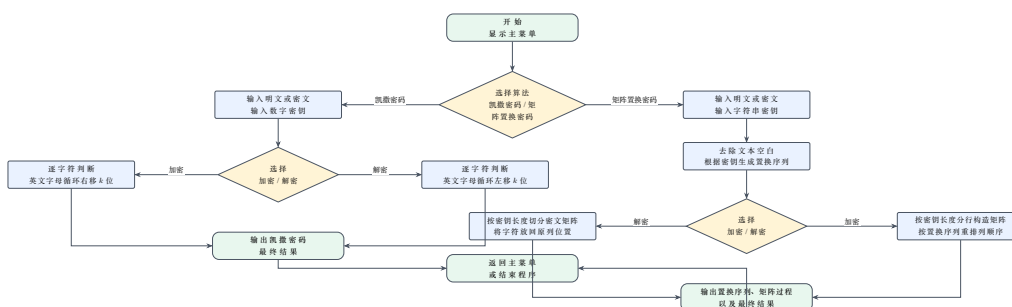


图 1: 古典密码实验程序实现流程图

1.3 核心算法说明

程序中最关键的两类算法如下：

1. 凯撒密码算法：逐个读取输入字符。若当前字符是英文字母，则先计算其相对于 **a** 或 **A** 的偏移量，再通过模 26 运算实现循环位移；若不是英文字

母，则直接原样输出。

2. 矩阵置换密码算法：先按字母表顺序为密钥生成置换序列，再按密钥长度把文本切分成多行。加密时按置换序列重新读取每行字符；解密时则把密文字符按逆过程放回原来的列位置。

对于本实验使用的去重后密钥 `houkaife`，程序生成的置换序列为：

(4, 7, 8, 6, 1, 5, 3, 2)

这意味着加密时每一行字符按“第 4 列、第 7 列、第 8 列、第 6 列、第 1 列、第 5 列、第 3 列、第 2 列”的顺序重新取出。

2. 调试过程及实验结果

调试时，我主要关注三类问题：第一，凯撒密码的循环位移是否正确；第二，矩阵置换密码是否与题目示例一致；第三，使用实验要求中的个人数据时，加密和解密是否互为逆过程。

2.1 测试样例

为了验证程序实现正确，调试时选取了两类测试数据：

1. 原理验证型样例：例如用 `abcxyz` 和密钥 3 检查凯撒密码是否正确处理字母表回绕；用题目给出的 `attack begins at five` 和 `cipher` 检查矩阵置换密码是否得到示例密文。
2. 实验演示型样例：使用包含本人姓名的字符串 作为明文，使用学号后两位 18 作为凯撒密码密钥，使用去重后的 `houkaife` 作为矩阵置换密码密钥。

2.2 运行结果

调试与最终运行结果如表所示。

表 1: 测试样例与运行结果汇总

编号	测试内容	输入数据	运行结果
1	凯撒密码回绕测试	明文 <code>abcxyz</code> ，密钥 3	得到密文 <code>defabc</code> ，说明循环位移正确。
2	矩阵置换题目示例测试	明文 <code>attack begins at five</code> ，密钥 <code>cipher</code>	得到密文 <code>aacttkbingesaivfte</code> ，与题目示例一致。
3	凯撒密码实验演示	明文 <code> </code> ，密钥 18	加密得到 <code>kzmrzsfzms</code> ，解密后恢复 <code> </code> 。
4	矩阵置换实验演示	明文 <code> </code> ，密钥 <code>houkaife</code>	加密得到 <code>zhashuhua</code> ，解密后恢复 <code> </code> 。

表 2: 最终实验结果汇总

编号	实验内容	输入数据	输出结果
1	凯撒密码加密	明文 <code> </code> ，密钥 18	密文 <code>kzmrzsfzms</code>
2	凯撒密码解密	密文 <code>kzmrzsfzms</code> ，密钥 18	明文 <code> </code>
3	矩阵置换密码加密	明文 <code> </code> ，密钥 <code>houkaife</code>	密文 <code>zhashuhua</code>
4	矩阵置换密码解密	密文 <code>zhashuhua</code> ，密钥 <code>houkaife</code>	明文 <code> </code>

2.3 运行截图

为直观展示程序运行效果，下面给出加密与解密过程的截图。

```
Code/IS/Lab01 [📦 v0.1.0][🐍 v3.13.12][💻 1m33s]
> python main.py

===== 实验一：古典密码算法 =====
1. 凯撒密码
2. 矩阵置换密码
0. 退出程序
请选择功能：1

===== 凯撒密码 =====
请选择操作：1. 加密 2. 解密
你的选择：1
请输入明
请输入数字密钥 k: 18

操作：凯撒密码加密
明文
密钥 k: 18
密文：kzmrzsfzms
说明：程序只移动英文字母，其他字符保持不变。
```

图 2: 凯撒密码加密运行结果

```
===== 实验一：古典密码算法 =====
1. 凯撒密码
2. 矩阵置换密码
0. 退出程序
请选择功能：1

===== 凯撒密码 =====
请选择操作：1. 加密 2. 解密
你的选择：2
请输入密文：kzmrzsfzms
请输入数字密钥 k: 18

操作：凯撒密码解密
密文：kzmrzsfzms
密钥 k: 18
明文
说明：程序只移动英文字母，其他字符保持不变。
```

图 3: 凯撒密码解密运行结果

```

===== 实验一：古典密码算法 =====
1. 凯撒密码
2. 矩阵置换密码
0. 退出程序
请选择功能：2

===== 矩阵置换密码 =====
请选择操作：1. 加密 2. 解密
你的选择：1
请输入明
请输入字符串密钥：houkaife

操作：矩阵置换密码加密
明文（原始输入）：
密钥：houkaife
密钥字符：h o u k a i f e
置换序列：4 7 8 6 1 5 3 2
加密前矩阵：
列号： 1 2 3 4 5 6 7 8
第1行： s h u z h a n h
第2行： u a . . . . .
按置换重排后的矩阵：
列号： 1 2 3 4 5 6 7 8
第1行： z n h a s h u h
第2行： u a . . . . .
明文： znhashuhua

```

图 4: 矩阵置换密码加密运行结果

```

===== 实验一：古典密码算法 =====
1. 凯撒密码
2. 矩阵置换密码
0. 退出程序
请选择功能：2

===== 矩阵置换密码 =====
请选择操作：1. 加密 2. 解密
你的选择：2
请输入密文：znhashuhua
请输入字符串密钥：houkaife

操作：矩阵置换密码解密
密文（原始输入）：znhashuhua
密钥：houkaife
密钥字符：h o u k a i f e
置换序列：4 7 8 6 1 5 3 2
解密时的密文矩阵：
列号： 1 2 3 4 5 6 7 8
第1行： z n h a s h u h
第2行： u a . . . . .
还原后的明文矩阵：
列号： 1 2 3 4 5 6 7 8
第1行： s h u z h a n h
第2行： u a . . . . .
明文：

```

图 5: 矩阵置换密码解密运行结果

2.4 结果分析

从运行结果可以看出，程序已经完整覆盖实验要求。首先，凯撒密码能够正确处理字母循环位移问题，并在解密时准确恢复原始明文。其次，矩阵置换密码不仅能够得到与题目示例一致的标准结果，也能在最后一行字符不足一整行时保持可逆。最后，在使用实验要求中的个人数据进行演示时，两种密码算法都能够顺利完成加密和解密，说明程序实现正确。

3. 总结

3.1 对上机实践结果进行分析

本次实验完成了凯撒密码和矩阵置换密码两种古典密码算法的程序实现，并完成了加密与解密演示。从运行结果看，凯撒密码能够正确完成字母循环位移，矩阵置换密码能够按照给定置换序列完成字符重排，并在解密时恢复原始明文，说明程序实现达到了实验要求。

3.2 上机的心得体会

通过这次上机，我对古典密码算法的理解比单纯阅读定义时更加具体。以前只知道凯撒密码是“移动字母”，矩阵置换密码是“交换位置”，真正写成程序后才发现，二者虽然都属于加密算法，但一个改变的是字符本身，另一个改变的是字符位置，思想差别非常明显。尤其是在矩阵置换密码中，当明文被排成矩阵再按列重排时，原本抽象的“置换”概念就变得非常直观。

3.3 改进意见

可以把矩阵置换密码的中间过程展示得更详细，使得更直观地理解加密和解密过程。

4. 附录

为了控制篇幅，附录中只保留与算法原理直接相关的核心实现代码。

4.1 古典密码核心程序 `classical_ciphers.py`

```
1 from __future__ import annotations
2
3 from dataclasses import dataclass
4
5
6 @dataclass
```

```
7 class MatrixCipherResult:
8     """保存一次矩阵置换密码运算的完整结果。"""
9
10    original_text: str
11    normalized_text: str
12    key: str
13    permutation: list[int]
14    source_rows: list[list[str]]
15    target_rows: list[list[str]]
16    result_text: str
17
18
19 def shift_english_letter(char: str, shift: int) -> str:
20     """
21     将单个英文字母循环移动 shift 位。
22
23     说明：
24     1. 只处理 A-Z / a-z
25     2. 其他字符直接原样返回
26     3. 使用模 26 运算保证越界后回到字母表开头
27     """
28     if "a" <= char <= "z":
29         base = ord("a")
30         return chr((ord(char) - base + shift) % 26 + base)
31
32     if "A" <= char <= "Z":
33         base = ord("A")
34         return chr((ord(char) - base + shift) % 26 + base)
35
36     return char
37
38
39 def caesar_encrypt(text: str, key: int) -> str:
40     """使用凯撒密码加密。"""
41     return "".join(shift_english_letter(char, key) for char in
42                     text)
43
44 def caesar_decrypt(text: str, key: int) -> str:
45     """使用凯撒密码解密。"""
46     return caesar_encrypt(text, -key)
47
48
49 def build_caesar_report(text: str, key: int, result: str, mode:
50                          str) -> str:
```

```

50     """把凯撒密码的结果整理成适合终端展示的中文文本。"""
51     mode_label = "加密" if mode == "encrypt" else "解密"
52     input_label = "明文" if mode == "encrypt" else "密文"
53     output_label = "密文" if mode == "encrypt" else "明文"
54
55     lines = [
56         f"操作: 凯撒密码{mode_label}",
57         f"{input_label}: {text}",
58         f"密钥 k: {key}",
59         f"{output_label}: {result}",
60         "说明: 程序只移动英文字母, 其他字符保持不变。",
61     ]
62     return "\n".join(lines)
63
64
65 def remove_whitespace(text: str) -> str:
66     """去掉字符串中的所有空白字符。"""
67     return "".join(char for char in text if not char.isspace())
68
69
70 def build_permutation_from_key(key: str) -> list[int]:
71     """
72     根据密钥生成题目示例风格的置换序列。
73
74     例如:
75     key = "cipher"
76     返回 [1, 4, 5, 3, 2, 6]
77
78     做法是:
79     1. 先按字母表顺序对密钥字符排序
80     2. 再把排序后的名次写回原位置
81     """
82     indexed_key = list(enumerate(key))
83     sorted_key = sorted(indexed_key, key=lambda item: (item[1].casefold(), item[0]))
84
85     permutation = [0] * len(key)
86     for rank, (original_index, _) in enumerate(sorted_key, start=1):
87         permutation[original_index] = rank
88
89     return permutation
90
91
92 def split_rows(text: str, column_count: int) -> list[list[str]]:

```

```

93     """把字符串按固定列数切分成多行。"""
94     return [list(text[index : index + column_count]) for index in
95             range(0, len(text), column_count)]
96
97 def reorder_row_by_permutation(row: list[str], permutation: list[
98     int]) -> list[str]:
99     """
100
101     按题目示例给出的置换序列重排一行字符。
102
103     例如 permutation 为 [1, 4, 5, 3, 2, 6] 时,
104     就按第 1、4、5、3、2、6 列的顺序取字符。
105
106     如果最后一行长度不够, 就只处理实际存在的列。
107     """
108     reordered_row: list[str] = []
109     for column_number in permutation:
110         column_index = column_number - 1
111         if column_index < len(row):
112             reordered_row.append(row[column_index])
113     return reordered_row
114
115 def restore_row_from_permutation(row: list[str], permutation:
116     list[int], row_length: int) -> list[str]:
117     """
118
119     根据置换序列把一行密文还原回原始顺序。
120
121     还原思路是:
122     1. 先找出当前这一行实际使用了哪些列
123     2. 再把密文中的字符放回它们原来所在的位置
124     """
125     restored_row = [''] * row_length
126     used_columns = [column_number for column_number in
127                     permutation if column_number <= row_length]
128
129     for char, column_number in zip(row, used_columns):
130         restored_row[column_number - 1] = char
131
132     return restored_row
133
134 def matrix_encrypt(plaintext: str, key: str) ->
135     MatrixCipherResult:
136     """使用矩阵置换密码加密。"""

```

```
133     normalized_text = remove_whitespace(plaintext)
134     normalized_key = remove_whitespace(key)
135
136     if not normalized_text:
137         raise ValueError("明文去掉空白后不能为空。")
138     if not normalized_key:
139         raise ValueError("密钥去掉空白后不能为空。")
140
141     permutation = build_permutation_from_key(normalized_key)
142     source_rows = split_rows(normalized_text, len(normalized_key)
143 )
144     target_rows = [reorder_row_by_permutation(row, permutation)
145 for row in source_rows]
146     result_text = "".join("".join(row) for row in target_rows)
147
148     return MatrixCipherResult(
149         original_text=plaintext,
150         normalized_text=normalized_text,
151         key=normalized_key,
152         permutation=permutation,
153         source_rows=source_rows,
154         target_rows=target_rows,
155         result_text=result_text,
156     )
157
158 def matrix_decrypt(ciphertext: str, key: str) ->
159 MatrixCipherResult:
160     """使用矩阵置换密码解密。"""
161     normalized_text = remove_whitespace(ciphertext)
162     normalized_key = remove_whitespace(key)
163
164     if not normalized_text:
165         raise ValueError("密文去掉空白后不能为空。")
166     if not normalized_key:
167         raise ValueError("密钥去掉空白后不能为空。")
168
169     permutation = build_permutation_from_key(normalized_key)
170     source_rows = split_rows(normalized_text, len(normalized_key)
171 )
172     target_rows = [
173         restore_row_from_permutation(row, permutation, len(row))
174 for row in source_rows
175 ]
176     result_text = "".join("".join(row) for row in target_rows)
```

```
173
174     return MatrixCipherResult(
175         original_text=ciphertext,
176         normalized_text=normalized_text,
177         key=normalized_key,
178         permutation=permutation,
179         source_rows=source_rows,
180         target_rows=target_rows,
181         result_text=result_text,
182     )
183
184
185 def format_matrix(rows: list[list[str]], column_count: int) ->
    str:
186     """
187     把矩阵格式化成适合终端阅读的文本。
188
189     用 . 表示当前行不存在的空位，便于观察最后一行是否填满。
190     """
191     if not rows:
192         return "(空矩阵)"
193
194     lines = []
195     header = "列号: " + " ".join(f"{index:>2}" for index in range
    (1, column_count + 1))
196     lines.append(header)
197
198     for row_number, row in enumerate(rows, start=1):
199         padded_row = row + ["."] * (column_count - len(row))
200         lines.append(f"第{row_number}行: " + " ".join(f"{char:>2}"
    for char in padded_row))
201
202     return "\n".join(lines)
203
204
205 def build_matrix_report(result: MatrixCipherResult, mode: str) ->
    str:
206     """把矩阵置换密码的过程整理成易读的中文文本。"""
207     input_label = "明文" if mode == "encrypt" else "密文"
208     output_label = "密文" if mode == "encrypt" else "明文"
209     source_title = "加密前矩阵" if mode == "encrypt" else "解密时的
    密文矩阵"
210     target_title = "按置换重排后的矩阵" if mode == "encrypt" else "
    还原后的明文矩阵"
211
```

```
212     lines = [  
213         f"操作: 矩阵置换密码{'加密' if mode == 'encrypt' else '解密'}"  
214     ,  
215         f"{input_label} (原始输入): {result.original_text}",  
216     ]  
217  
218     if result.original_text != result.normalized_text:  
219         lines.append(f"{input_label} (去掉空白后): {result.  
220 normalized_text}")  
221  
222     lines.extend(  
223     [  
224         f"密钥: {result.key}",  
225         "密钥字符: " + " ".join(result.key),  
226         "置换序列: " + " ".join(str(number) for number in  
227 result.permutation),  
228         f"{source_title}: ",  
229         format_matrix(result.source_rows, len(result.key)),  
230         f"{target_title}: ",  
231         format_matrix(result.target_rows, len(result.key)),  
232         f"{output_label}: {result.result_text}",  
233     ]  
234     )  
235  
236     return "\n".join(lines)
```