实验报告（DES）

【实验目的】

1、 理解 Feistel 结构及 DES 算法的原理，并掌握加解密流程；
2、 了解弱密钥与半弱密钥；
3、 对算法进行深入理解和思考，开始尝试算法的优化与改良。

【实验环境】

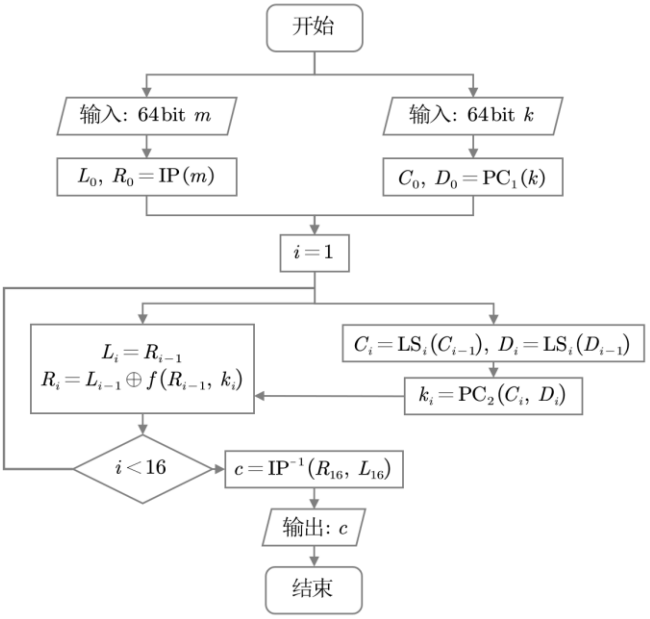
- Python 3.9.7 64-bit

【实验内容】

一、DES 算法

1、算法流程

流程图：



伪代码：

Algorithm 1：密钥生成

Input: 初始密钥k
Output: 子密钥集合$K = (k_1, k_2, \cdots k_{16})$

1. **function** "KeyGen(k)"
2. $(C, D) \leftarrow \mathrm{PC}_1(k)$
3. **for** $i \leftarrow 1$ to 16 **do**
4. $C \leftarrow \mathrm{LS}_i(C)$
5. $D \leftarrow \mathrm{LS}_i(D)$
6. $k_i \leftarrow \mathrm{PC}_2(C, D)$
7. **end for**
8. **return** $K \leftarrow (k_1, k_2, \cdots k_{16})$
9. **end function**

Algorithm 2： DES加密

Input: 明文m, 密钥k
Output: 密文c

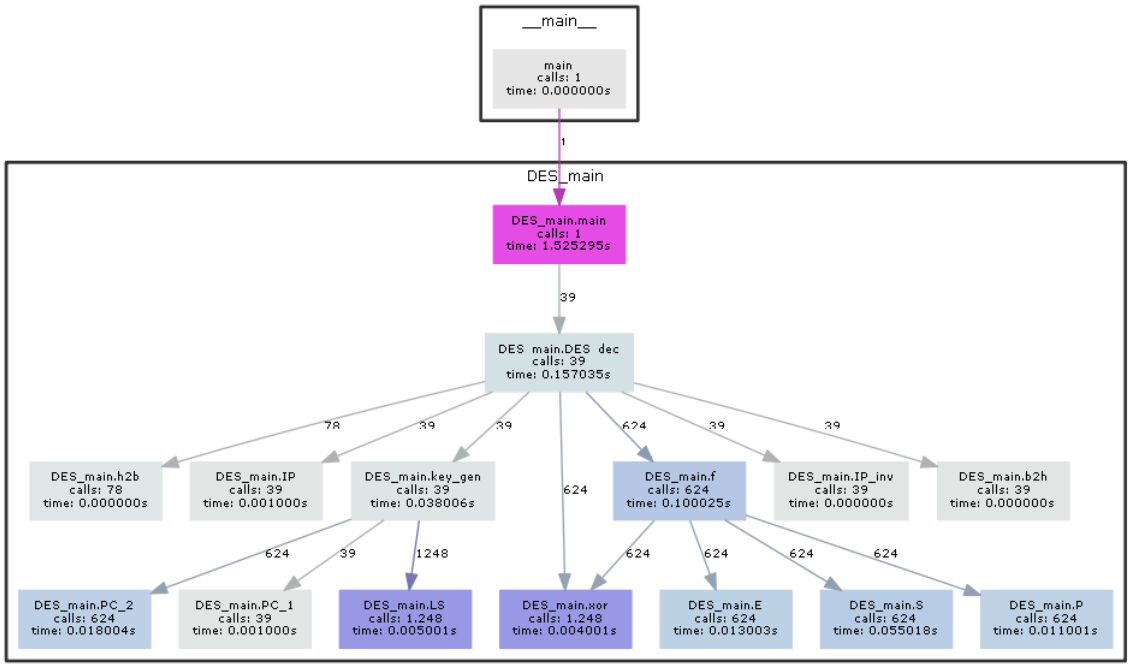
function "DES_enc$(m,\ k)$"
    $(L,R) \leftarrow \mathrm{IP}(m)$
    $K \leftarrow \mathrm{KeyGen}(k)$
    for $i \leftarrow 1$ to 16 **do**
        $L,\ R \leftarrow R,\ L \oplus f(R,\ k_i)$
    end for
    $c \leftarrow \mathrm{IP}^{-1}(R,\ L)$
    return c
end function

Algorithm 3： DES解密

Input: 密文c, 密钥k
Output: 明文m

function "DES_dec$(c,\ k)$"
    $(R,L) \leftarrow \mathrm{IP}(c)$
    $K \leftarrow \mathrm{KeyGen}(k)$
    for $i \leftarrow 16$ to 1 **do**
        $R_{last} \leftarrow R$
        $R \leftarrow L$
        $L \leftarrow R_{last} \oplus f(R,\ k_i)$
    end for
    $m \leftarrow \mathrm{IP}^{-1}(L,\ R)$
    return m
end function

函数调用关系图如下（以解密为例）：



函数名	函数功能
DES_enc(),DES_dec()	DES 加密主函数、DES 解密主函数
h2b(),b2h()	十六进制转二进制、二进制转十六进制
key_gen()	密钥生成主函数
IP(),IP_inv()	IP 置换、IP 逆置换
PC_1(),PC_2()	选择置换 1、选择置换 2
LS()	循环左移
E(),S(),P()	扩展置换 E、S 盒代替、置换运算 P
xor()	异或运算

2、测试样例及结果截图

```
39
0x3fc8d0410de97f5e
0xee7bfa825f654293
0
0xad0ea0c458594e1e

Process finished with exit code 0
```

```
1
0x900eb9584f9321c0
0x84f5ca4805eb1ba3
1
0xb144bfab09125c7b

Process finished with exit code 0
```

```
1
0xf0af67754a865148
0x4b771d5972ff5399
0
0xbeead9f0ce6aeac1

Process finished with exit code 0
```

```
39
0x5ea6454a04d1e28c
0x41441b3cac8e2e35
1
0x0d317c43dcbf42a6

Process finished with exit code 0
```

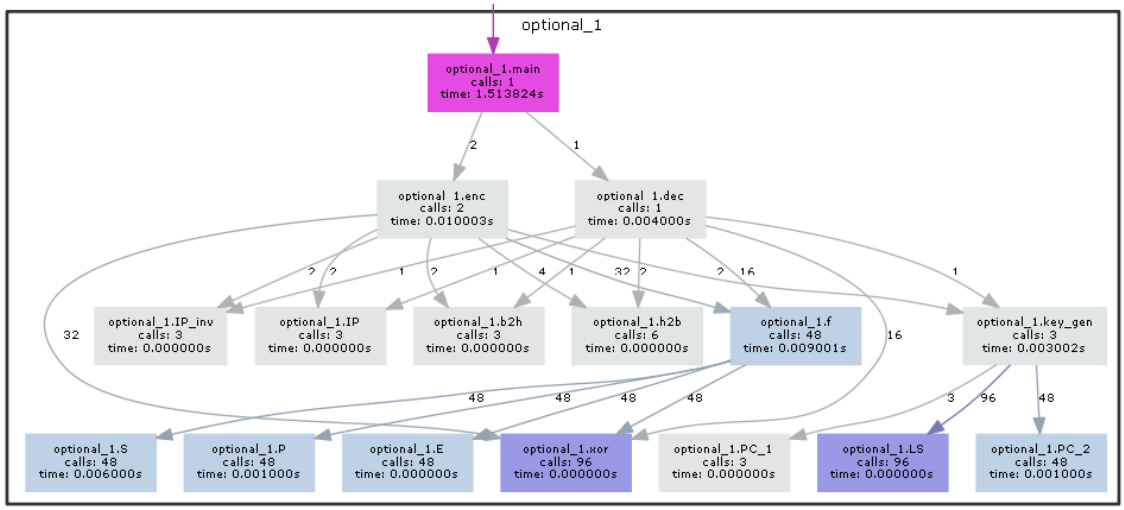
3、讨论与思考

- 对于明/密文、密钥均用 Python 的字符串储存，并用'+'进行字符串拼接操作，这比 Python 中的 list 运行效率低，是此程序可以优化的一点；
- 对于置换操作，该程序多次查找置换表（list），这比 Python 中的字典（dict）运行效率低，也是程序可以优化的一点。

二、三重 DES 算法（选做一）

1、算法流程

函数调用关系图如下：



2、测试样例及结果截图

```
0x78e2025d31d06fde
0xa530af81d46635e4
0xfc6aa9db1d2b1224
1
0xf1b98dac6657545c

Process finished with exit code 0
```

```
0x618f1b32353b4f35
0xc50f4b2e21282135
0x35becf7d711ee29a
0
0xe5388d6e1eda0fad

Process finished with exit code 0
```

3、讨论与思考

- 嵌套调用 DES_enc()、DES_dec()即可。

三、弱密钥与半弱密钥（选做二）

1、算法流程

a、弱密钥

- 记录 64 bit 密钥每一位的下标；
- 生成 16 轮子密钥，其中为一位均为原密钥的下标索引；
- 弱密钥要求每轮子密钥相同，即每一位对应原密钥的下标索引均为 0 或 1；
- 集合求交求并得到 2 组下标，其中每一组下标对应的元素（0 或 1）相同；
- 剩下的下标索引为奇偶校验位；
- 共2^3个弱密钥。

b、半弱密钥

- 半弱密钥要求第i轮子密钥与第$16-i$轮子密钥相同；
- 将第i轮子密钥下标标记为 0~63，第$16-i$轮子密钥下标标记为 64~127，将两者进行下标配对；
- 集合求交求并得到 4 组下标，其中每一组下标对应的元素（0 或 1）相同；
- 剩下的下标索引为奇偶校验位；
- 共2^5-8对半弱密钥。

2、测试样例及结果截图

弱密钥下标（0 ~ 63）分组：

```
[0, 1, 2, 8, 9, 10, 16, 17, 18, 24, 25, 26, 32, 33, 34, 35, 40, 41, 42, 43, 48, 49, 50, 51, 56, 57, 58, 59]
[3, 4, 5, 6, 11, 12, 13, 14, 19, 20, 21, 22, 27, 28, 29, 30, 36, 37, 38, 44, 45, 46, 52, 53, 54, 60, 61, 62]
[7, 15, 23, 31, 39, 47, 55, 63]
```

半弱密钥下标（0 ~ 127）分组：

```
[0, 1, 2, 16, 17, 18, 32, 33, 34, 35, 48, 49, 50, 51, 72, 73, 74, 88, 89, 90, 104, 105, 106, 107, 120, 121, 122, 123]
[3, 4, 5, 6, 19, 20, 21, 22, 36, 37, 38, 52, 53, 54, 75, 76, 77, 78, 91, 92, 93, 94, 108, 109, 110, 124, 125, 126]
[8, 9, 10, 24, 25, 26, 40, 41, 42, 43, 56, 57, 58, 59, 64, 65, 66, 80, 81, 82, 96, 97, 98, 99, 112, 113, 114, 115]
[11, 12, 13, 14, 27, 28, 29, 30, 44, 45, 46, 60, 61, 62, 67, 68, 69, 70, 83, 84, 85, 86, 100, 101, 102, 116, 117, 118]
[7, 15, 23, 31, 39, 47, 55, 63, 71, 79, 87, 95, 103, 111, 119, 127]
```

弱密钥：

0x0101010101010101	0x0000000000000000	0x1f1f1f1f0e0e0e0e	0x1e1e1e1e0f0f0f0f
0xe0e0e0e0f1f1f1f1	0xe1e1e1e1f0f0f0f0	0xfefefefefefefefe	0xffffffffffffffff

半弱密钥：

0x1f011f010e010e01	0x011f011f010e010e	0xfe1ffe1ffe0efe0e	0x1ffe1ffe0efe0efe
0x1e001e000f000f00	0x001e001e000f000f	0xff1eff1eff0fff0f	0x1eff1eff0fff0fff
0xe001e001f101f101	0x01e001e001f101f1	0xfe01fe01fe01fe01	0x01fe01fe01fe01fe
0xe100e100f000f000	0x00e100e100f000f0	0xff00ff00ff00ff00	0x00ff00ff00ff00ff
0xfee0fee0fef1fef1	0xe0fee0fef1fef1fe	0xe01fe01ff10ef10e	0x1fe01fe00ef10ef1
0xffe1ffe1fff0fff0	0xe1ffe1fff0fff0ff	0xe11ee11ef00ff00f	0x1ee11ee10ff00ff0

3、讨论与思考

- 密钥生成只涉及移位运算，故直接用 int 型记录。
- Python 中可以直接用 set 类进行集合求交求并操作，但对于本题而言，尤其是半弱密钥，其集合量过多，用 set 类无论从时间上还是空间上效率都比较低，因此笔者直接对 list 进行查找达到分组效果。

四、DES 算法的优化（选做三）

1、算法流程

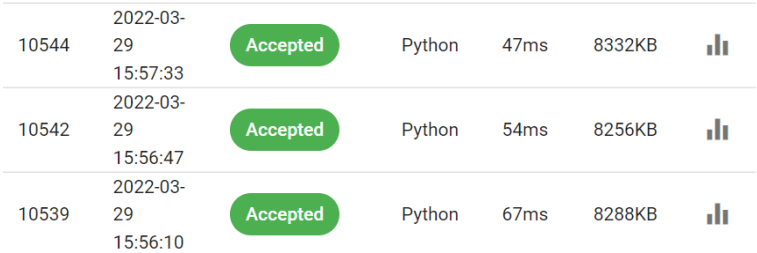
- 以 C++语言对 DES 进行重现。

2、测试样例及结果截图

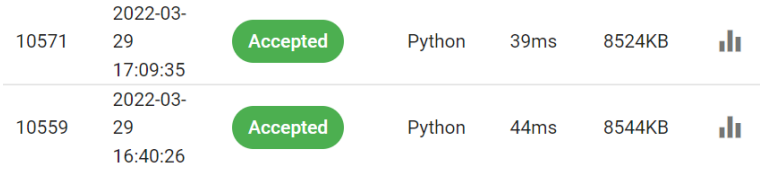
```
39
0x5ea6454a04d1e28c
0x41441b3cac8e2e35
0x0d317c43dcbf42a6

Process returned 0 (0x0)   execution time : 2.315 s
Press any key to continue.
```

3、讨论与思考

- 笔者尝试基于 Python 进行优化，首先将之前用的 str 类替换为 list 类，将 list 查找替换为 dict 查找，结果并未有本质上的提升（测试结果见下图）。

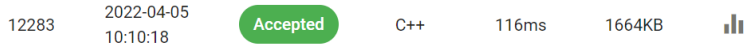


- 另一方面，笔者对编程细节上进行了优化：首先将每个置换列表都提到了函数外，减少每次调用函数时生成列表的次数；另外，对 S 盒进行了优化，由于一个 6 位的输入对应一个确定的 4 位输出，笔者根据这个对应关系写了一个映射函数。运行发现，这两方面的优化仍未使程序效率有本质上的提升。显然，这是达不到程序的优化要求的。根本原因还是在于 Python。



- 于是，笔者通过更换编程语言来解决这个问题，运用 C++，笔者很容易便满足了题目的需求

- 但笔者并没有在算法上进行优化，终于理所应当的在一周后当了排行榜的垫底。
- 这显然是满足不了我的好奇心的，难道不同于 Python，优化 S 盒在 C++里面真的能变快？笔者按照前面的步骤直接进行了 SP 优化，mission success。



【收获与建议】

1、收获

- ~~加深了对 Feistel 结构及 DES 算法的理解。~~
- ~~提高了对 Python 的熟练度和 C++的编程能力。~~
- ~~巩固了排版能力。~~
- 收获了劳累与繁忙。

2、建议

- 希望老师和助教能给一个学期成绩的稍微具体一点的衡量标准（譬如必做占多少、选做占多少、实验报告占多少、对哪一项的重视程度大一些），这样可以让大家都合理地做取舍。（笔者感觉花在选做上的时间太多了，其他事情忙不过来，但又舍不得不做选做）

【思考题】

1、 目前对于 DES 的密码分析都做了哪些工作，有什么成果。

- 随着攻击技术的发展，DES 本身又有发展，衍生出可抗差分分析攻击的变形 DES 以及密钥长度为 128 比特的三重 DES 等。
- 1990 年 S.Biham 和 A.Shamir 提出了差分攻击的方法，采用选择明文2^{47}攻击，最终找到可能的密钥。
- M.Matsui 提出的线性分析方法，利用2^{43}个已知明文，成功地破译了 16 圈 DES 算法，到目前为止，这是最有效的破译方法。
- 从 1997 年开始，RSA 公司发起了一个称作“向 DES 挑战”的竞技赛。在首届挑战赛上，罗克·维瑟用了 96 天时间破解了用 DES 加密的一段信息。
- 1999 年 12 月 22 日，RSA 公司发起“第三届 DES 挑战赛（DES Challenge III）”。2000 年 1 月 19 日，由电子边疆基金会组织研制的 25 万美元的 DES 解密机以 22.5 小时的战绩，成功地破解了 DES 加密算法。

2、 DES 为何不再适用于现代。

- 密钥空间小，加/解密处理简单，加密密钥与解密密钥相同。
- 随着计算机效率的提升，强力攻击（2^{55}次尝试）、差分密码分析（2^{47}次尝试）、线性密码分析（2^{43}次尝试）在现实应用方面已成为可能。使得 DES 安全性受到极大的挑战。