DES算法的Python（3.7）实现

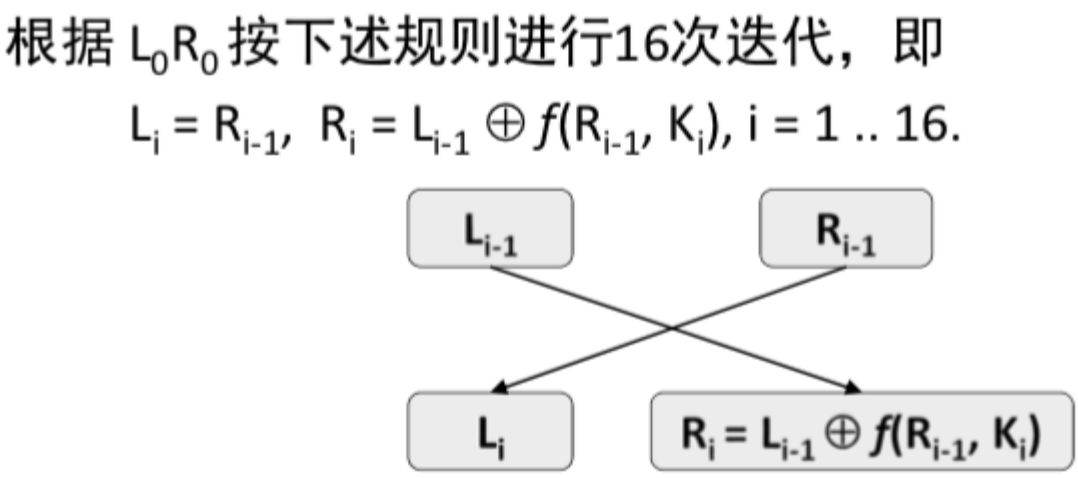
算法描述

DES算法是利用56 + 8奇偶校验位 = 64位的密钥对以64位为单位的数据块进行加密和解密。它的加密与解密实现首先是要生成一套加密密钥，用户输入一个64位长的密码，然后通过等分、移位、选取和迭代形成一套16个加密密钥，分别用于之后的每一轮运算。

加密

DES对以64位(bit)为单位的明文分组M进行操作，M经过初始置换IP，成为M_0 。将M_0明文分为左半部分和右半部分，$M_0 = (L_0 ,R_0)$，各32位长。然后进行16轮完全相同的迭代运算，即 ***Feistel*** 轮函数；在每一轮运算过程中数据与相应的密钥结合进行操作。

每一轮轮函数运算中，密钥位移位，然后再从密钥的56位中选出48位。通过一个扩展置换将数据的右半部分扩展成48位，并通过一个异或操作替代成新的48位数据，再将其压缩置换成32位。做完这些操作后，通过另一个异或运算，轮函数的输出与左半部分结合，其结果成为新的右半部分，原来的右半部分成为新的左半部分。此操作重复16次，最后输出$R_{16}L_{16}$。

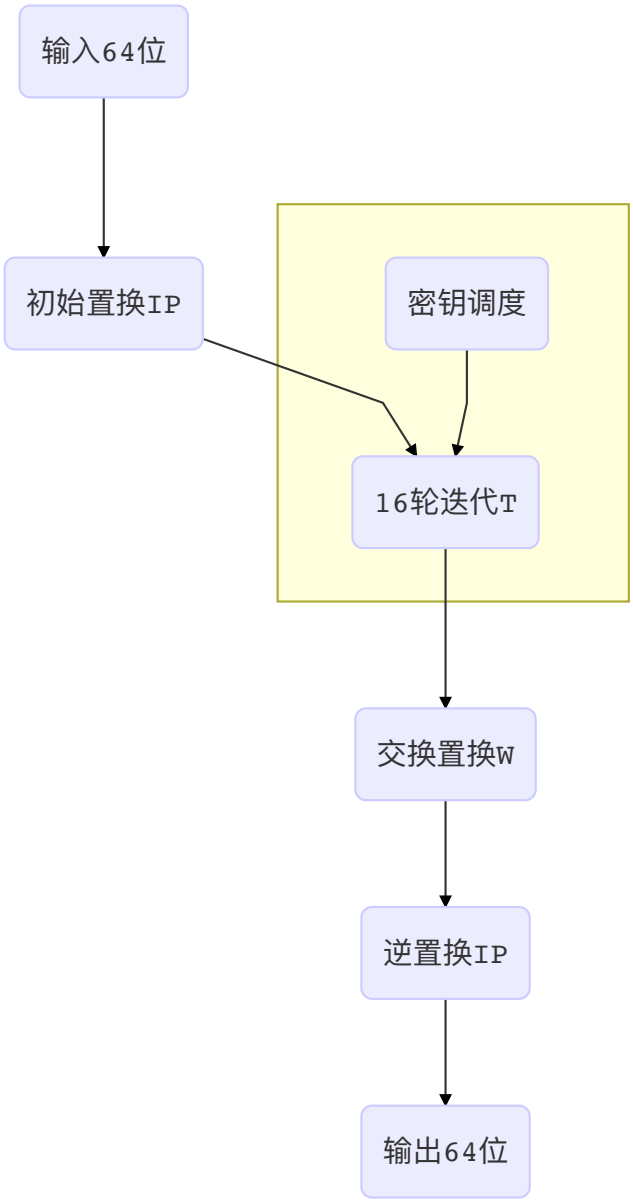


至此，16轮迭代结束后，左右部分结合在一起后，再经过一次IP逆置换，加密过程结束。

解密

DES解密并不是加密过成的逆运算，而是与加密算法相同算法。唯一的不同是密钥的次序相反，加密时是$K_1K_2...K_{16}$的顺序，解密时为$K_{16}K_{15}...K_1$的顺序

总体结构



输入64位明文M时，子密钥调度次序按照$K_1K_2...K_{16}$的顺序调度，为加密过程；

输入64位密文时，子密钥按$K_{16}K_{15}...K_1$次序调度，为解密过程。

代码实现

在此算法的实现中，我使用python是因为其中的List这种数据类型，用这种类型的数据结构存放在算法中要用到的IP置换表，IP^{-1}置换表，8个S盒，P盒以及比特-选择表（E-扩展规则）等，python中对这些List的转换、移位操作非常方便，而不用像c，c++中那样构造二维数组。

代码由两个todo.py和go.py两部分构成。其中todo.py主要实现算法中各种表的置换、数据类型的转换以及子密钥的生成和其他一些功能函数；go.py中主要实现算法的具体运算，包括加密与解密，以及输入等操作。

初始化部分

两个IP置换表的实现：

```
# 初始IP置换表，返回置换后的list
def IpTable(text):
    IP = [58, 50, 42, 34, 26, 18, 10, 2,
          60, 52, 44, 36, 28, 20, 12, 4,
          62, 54, 46, 38, 30, 22, 14, 6,
          64, 56, 48, 40, 32, 24, 16, 8,
          57, 49, 41, 33, 25, 17, 9, 1,
          59, 51, 43, 35, 27, 19, 11, 3,
          61, 53, 45, 37, 29, 21, 13, 5,
          63, 55, 47, 39, 31, 23, 15, 7]
    return [text[IP[i] - 1] for i in range(64)]
    #由于IP表里的初始下标为1，而list索引是从0开始，所以是text[IP[i]-1]

# 逆初始IP置换表，返回置换后的list
def InvereIpTable(text):
    INVERSE_IP = [40, 8, 48, 16, 56, 24, 64, 32,
                  39, 7, 47, 15, 55, 23, 63, 31,
                  38, 6, 46, 14, 54, 22, 62, 30,
                  37, 5, 45, 13, 53, 21, 61, 29,
                  36, 4, 44, 12, 52, 20, 60, 28,
                  35, 3, 43, 11, 51, 19, 59, 27,
                  34, 2, 42, 10, 50, 18, 58, 26,
                  33, 1, 41, 9, 49, 17, 57, 25]
    return [text[INVERSE_IP[i] - 1] for i in range(64)]
```

二进制的明文块，通过置换后重排明文块的二进制串

子密钥的生成：

```
#PC_1置换规则，不含校验位下标
PC_1 = [57, 49, 41, 33, 25, 17, 9,
        1, 58, 50, 42, 34, 26, 18,
        10, 2, 59, 51, 43, 35, 27,
        19, 11, 3, 60, 52, 44, 36,
```

```
        63, 55, 47, 39, 31, 23, 15,
        7, 62, 54, 46, 38, 30, 22,
        14, 6, 61, 53, 45, 37, 29,
        21, 13, 5, 28, 20, 12, 4]
#PC_2置换规则
PC_2 = [14, 17, 11, 24, 1, 5,
        3, 28, 15, 6, 21, 10,
        23, 19, 12, 4, 26, 8,
        16, 7, 27, 20, 13, 2,
        41, 52, 31, 37, 47, 55,
        30, 40, 51, 45, 33, 48,
        44, 49, 39, 56, 34, 53,
        46, 42, 50, 36, 29, 32]
# 子密钥的生成
def createSubKey(preKey):
    result = []
    # 对56个非校验位进行PC_1置换，校验位下标不参与置换
    key56 = [preKey[PC_1[i]-1] for i in range(56)]
    # 移动的步长，当i=0，1，8，15时，二进制串循环左移一位，否则循环左移两位
    step = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]
    #生成16组子密钥
    for i in range(16):
        key_left = move(key56[:28], step[i]) # 前28位
        key_right = move(key56[28:], step[i]) # 后28位
        key56 = key_left + key_right
        #对56位的结果进行压缩置换
        key48 = [key56[PC_2[i]-1] for i in range(48)]
        #将每一次生成的子密钥Ki放入result中
        result.append(key48)
    return result
# 移动，对给定的list左移指定次数，并返回移动后的序列
def move(text, times):
    return text[times:] + text[:times]
```

轮函数部分

E-扩展

```
# E扩展置换，输出48位
def E_extend(text):
    E = [32, 1, 2, 3, 4, 5,
         4, 5, 6, 7, 8, 9,
         8, 9, 10, 11, 12, 13,
         12, 13, 14, 15, 16, 17,
         16, 17, 18, 19, 20, 21,
         20, 21, 22, 23, 24, 25,
         24, 25, 26, 27, 28, 29,
         28, 29, 30, 31, 32, 1]
    return [text[E[i] - 1] for i in range(48)]
```

S-盒置换：

```
# s盒置换，将48位输入均分成长度为6的8个小组，每个小组按顺序进入相应的S盒各得到4位输出，返回合并后的32位结果。
def s_box_change(text):
    result = []
    for i in range(0, 8):
        # 将输入以6为单位切分为8组，分别使用一个S_box的转换规则
        temp = text[i*6:(i+1)*6]
        # list索引时数据类型为int，所以进行强制类型转换
        # 由第一位和最后一位确定S盒的行号
        row = int(str(temp[0])+str(temp[-1]), 2)
        # 由中间四位确定S盒列号
        col = int(str(temp[1])+str(temp[2])+str(temp[3])+str(temp[4]), 2)
        message = S_box[i][row][col]
        # 由于S盒中的数据是十进制数，故将其转化为四位二进制数后加入result中
        result.append(dec2bin4(message))

    return [int(x) for x in ''.join(result)]
```

P-盒置换

```
# P-盒置换：将32位输入按 P 规则置换后返回32位结果。
def p_box_change(text):
    P = [16, 7, 20, 21, 29, 12, 28, 17,
         1, 15, 23, 26, 5, 18, 31, 10,
         2, 8, 24, 14, 32, 27, 3, 9,
         19, 13, 30, 6, 22, 11, 4, 25]
    return [text[P[i] - 1] for i in range(32)]
```

异或

```
# 异或操作，对序列进行各位异或操作，并返回所有结果
def xor(m, n):
    return [a ^ b for a, b in zip(m, n)]
```

F函数

```
    for i in range(16):
        l, r = text[:32], text[32:]
        r_extend = E_extend(r)  # 对右半部分进行E扩展运算，转换成48位的串
        xor1 = xor(r_extend, subkeys[i])  # 对得到的48位串与48位的子密钥进行异或
运算
        s_box_result = s_box_change(xor1)  # 异或得到的结果进行S盒转换
        p_box_result = p_box_change(s_box_result)  # p置换，得到轮函数的输出
        xor2 = xor(l, p_box_result)  # 将左半部分与轮函数输出做异或，得到新串的右
半部分
        text = r + xor2  # 左右部分结合，形成新串
```

算法实现部分

```
# 加密解密共用的算法，解密时只需反转子密钥序列即可
def cipher(message, key, mode='encrypt'):
    # 初始化明文和密钥，将其转化为二进制串
    message = string2bin(message)
    key = string2bin(key)
    # 子密钥的生成，当mode为'decrypt'时，反转子密钥序列
    subkeys = createSubKey(key) if mode == 'encrypt' else createSubKey(key)
[::-1]
    # 对明文进行IP置换
    text = IpTable(message)
    # F函数的实现
    for i in range(16):
        l, r = text[:32], text[32:]
        r_extend = E_extend(r)  # 对右半部分进行E扩展运算，转换成48位对串
        xor1 = xor(r_extend, subkeys[i])  # 对得到对48位串与48位对子密钥进行异或
运算
        s_box_result = s_box_change(xor1)  # 异或得到对结果进行S盒转换
        p_box_result = p_box_change(s_box_result)  # p置换，得到轮函数的输出
        xor2 = xor(l, p_box_result)  # 将左半部分与轮函数输出做异或，得到新串对右
半部分
        text = r + xor2  # 左右部分结合，形成新串

    text = text[32:] + text[:32]
    # 将二进制密文转换为字符串后返回
    return bin2string(InvereIpTable(text))
```

考虑到用户输入的明文不一定都满足规定的位数，所以需要对输入的明文进行一定的操作，对于不满足要求的进行补位操作。

```
def fill(string):  # 如果字符串的长度不是8的倍数，则补满空缺位数
    mod = len(string) % 8
    space = 8 - mod
    return string + bytes(space).decode('utf-8')
```

DES类的封装

```
class DES:
    def __init__(self, message, key):
        self.message = message
        self.key = key

    @property
    def ciphertext(self):  # 密文
        return self.__encrypt()
    @property
    def plaintext(self):  # 明文
        return self.__decrypt()

    def __encrypt(self):  # 加密
        output = []
        length = len(self.message)
        # 按64bit为一组(8bytes)进行切分
        times, mod = length // 8, length % 8
        if mod:  # 如果mod不为0，进行补位
            self.message = fill(self.message)
            times += 1
        # 对明文对每一组进行操作
        for i in range(times):
            result = cipher(self.message[i * 8:i * 8 + 8], self.key,
'encrypt')
            output.append(result)
        return ''.join(output)

    def __decrypt(self):  # 解密
        output = []
        length = len(self.message)
        times, mod = length // 8, length % 8
        if not times:
            return None
        if mod:
            self.message = fill(self.message)
            length += 1
        for i in range(times):
```

```
            result = cipher(self.message[i * 8:i * 8 + 8], self.key,
'decrypt')
            output.append(result)
        return ''.join(output).rstrip(b'\x00'.decode('utf-8'))
```

main函数

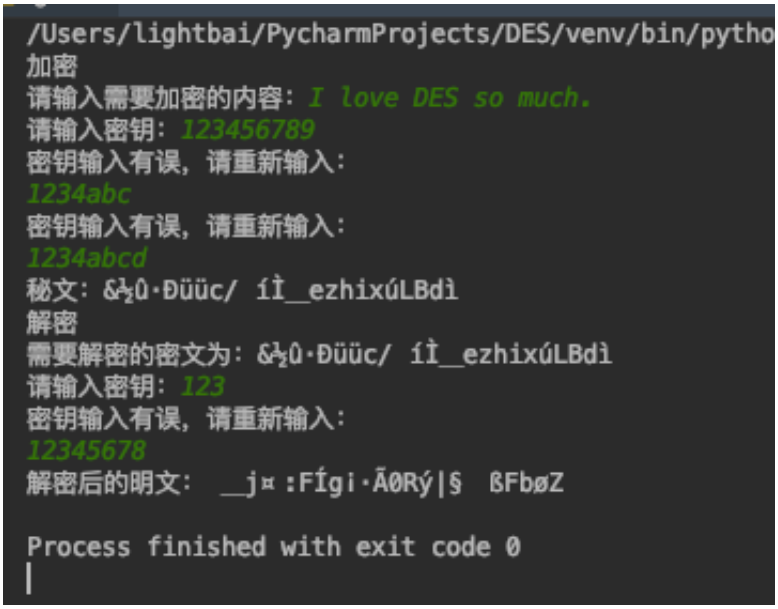
```
if __name__ == '__main__':

    mess = input("请输入需要加密的内容：")
    key = input("请输入密钥：")
    while len(key) != 8:
        print("密钥输入有误，请重新输入：")
        key = input()
    cipher1 = DES(mess, key)
    print("秘文："+cipher1.ciphertext)

    cipher2 = DES(cipher1.ciphertext, key)

    print("解密后的明文："+cipher2.plaintext)
```

运行结果

测试明文：I love DES so much.

密钥：1234abcd

运行截图：



当密钥长度不为8时，会有错误提示，如果加密解密不使用同一密钥，则解密得到的明文与原明文不同。