problem set 4实验报告

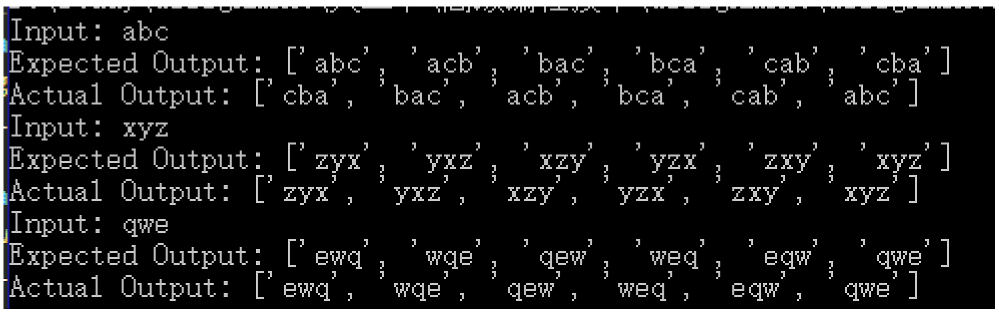
16337183 孟衍璋

ps4a

这部分需要编写一个函数，计算得出一个字符串的全排列。需要用到递归的思想，即一个字符串的全排列就是先计算除去第一个字符之后，剩下的子串的全排列，再将第一个字符依次插入子串的全排列中，就能得出该字符的全排列。代码实现如下：

```python
def get_permutations(sequence):
    prmutations = []
    # 如果只含有一个字符
    if len(sequence) == 1:
        permutations.append(sequence)
    # 如果含有多个字符
    else:
        # 首先获得去掉第一个字符的子串的全排列
        sub_permutations = get_permutations(sequence[1:])
        # 在每一个子串的全排列中，将第一个字符插入其中
        for item in sub_permutations:
            for pos in range(len(item) + 1):
                # 插入第一个字符的位置共有len(item)+1个
                new_str = item[pos:] + sequence[:1] + item[:pos]
                permutations.append(new_str)
    return permutations
```

实验结果：



```
Input: abc
Expected Output: ['abc', 'acb', 'bac', 'bca', 'cab', 'cba']
Actual Output: ['cba', 'bac', 'acb', 'bca', 'cab', 'abc']
Input: xyz
Expected Output: ['zyx', 'yxz', 'xzy', 'yzx', 'zxy', 'xyz']
Actual Output: ['zyx', 'yxz', 'xzy', 'yzx', 'zxy', 'xyz']
Input: qwe
Expected Output: ['ewq', 'wqe', 'qew', 'weq', 'eqw', 'qwe']
Actual Output: ['ewq', 'wqe', 'qew', 'weq', 'eqw', 'qwe']
```

ps4b

这部分需要实现凯撒密码，指定一个移位值，按照字母表的顺序将每个字母进行移位，得到加密后的字符串，再根据穷举26个移位值，计算哪个移位值对应得到的正确单词数最多，来进行解密。

部分函数实现如下：

class Message(object)

```
def __init__(self, text):
    self.message_text = text
    self.valid_words = load_words("./words.txt")
```

```
def get_message_text(self):
    # 返回消息
    return self.message_text
```

```
def get_valid_words(self):
    # 返回合法词语列表
    temp = self.valid_words.copy()
    return temp
```

```
def build_shift_dict(self, shift):
    # 首先初始化一个字典，包含所有大写字母与小写字母
    dictionary = {}
    for letter in string.ascii_lowercase:
        dictionary[letter] = letter
    for letter in string.ascii_uppercase:
        dictionary[letter] = letter
    # dictionary = {'a':'a', 'b':'b', 'c':'c', 'd':'d', 'e':'e', 'f':'f',
'g':'g', 'h':'h', 'i':'i', 'j':'j', 'k':'k', 'l':'l', 'm':'m', 'n':'n',
'o':'o', 'p':'p', 'q':'q', 'r':'r', 's':'s', 't':'t', 'u':'u', 'v':'v',
'w':'w', 'x':'x', 'y':'y', 'z':'z', 'A':'A', 'B':'B', 'C':'C', 'D':'D',
'E':'E', 'F':'F', 'G':'G', 'H':'H', 'I':'I', 'J':'J', 'K':'K', 'L':'L',
'M':'M', 'N':'N', 'O':'O', 'P':'P', 'Q':'Q', 'R':'R', 'S':'S', 'T':'T',
'U':'U', 'V':'V', 'W':'W', 'X':'X', 'Y':'Y', 'Z':'Z'}
    # 运用ascii码，将移位后的字符对应起来
    for letter in string.ascii_lowercase:
        dictionary[letter] = chr((ord(letter) + shift - ord('a')) % 26 +
ord('a'))
    for letter in string.ascii_uppercase:
        dictionary[letter] = chr((ord(letter) + shift - ord('A')) % 26 +
ord('A'))
    return dictionary
```

```
def apply_shift(self, shift):
    message = self.get_message_text()
    dictionary = self.build_shift_dict(shift)
    new_message = ""
    for i in message:
        # 如果是字母，则计算其移位后的字母加入新字符串中
        if i in dictionary:
            new_message += dictionary[i]
        # 如果不是字母，则保留
        else:
            new_message += i
    return new_message
```

class PlaintextMessage(Message)

```
def __init__(self, text, shift):
    self.message_text = text
    self.shift = shift
```

```
def get_shift(self):
    return self.shift
```

```
def get_encryption_dict(self):
    dictionary = self.build_shift_dict(get_shift())
    return dictionary
```

```
def get_message_text_encrypted(self):
    message = self.get_message_text()
    shift = self.get_shift()
    return self.apply_shift(shift)
```

```python
def change_shift(self, shift):
    self.shift = shift
```

class CiphertextMessage(Message)

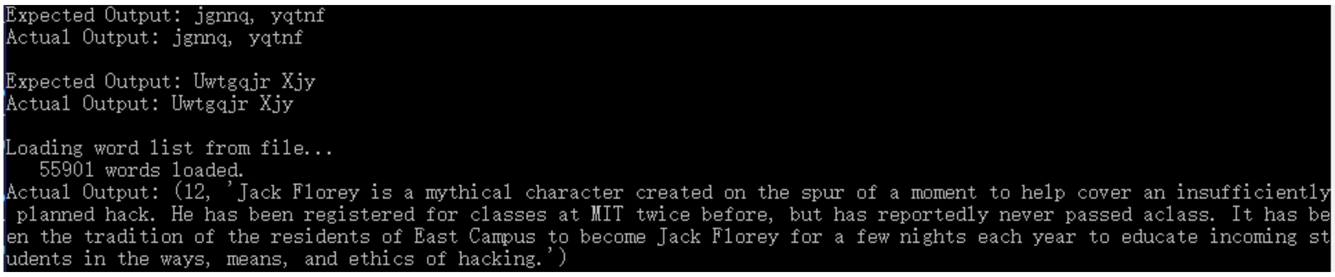
```python
def __init__(self, text):
    self.message_text = text
    self.valid_words = load_words("./words.txt")
```

```python
def decrypt_message(self):

    # count是一个列表，存储每一个移位值得到的合法字符串的数目
    count = []
    for s in range(26):
        message = self.apply_shift(s)
        # print('s =', s) #.
        # print(message) #.
        message = message.split()  # 将移位后的字符串分割为一个一个单词
        # 计算得到的单词中合法的单词的数目
        flag = 0
        for word in message:
            # print("w =", word) #.
            if is_word(self.valid_words, word):
                flag += 1
        count.append(flag)

    # 得到合法的单词数最多的一个移位值
    ind = count.index(max(count))
    # print(count) #.
    # print(self.apply_shift(count[ind])) #.
    message = self.apply_shift(ind)
    return (ind, message)
```

实验结果



```
Expected Output: jgnnq, yqtnf
Actual Output: jgnnq, yqtnf

Expected Output: Uwtgqjr Xjy
Actual Output: Uwtgqjr Xjy

Loading word list from file...
   55901 words loaded.
Actual Output: (12, 'Jack Florey is a mythical character created on the spur of a moment to help cover an insufficiently
 planned hack. He has been registered for classes at MIT twice before, but has reportedly never passed aclass. It has be
en the tradition of the residents of East Campus to become Jack Florey for a few nights each year to educate incoming st
udents in the ways, means, and ethics of hacking.')
```

ps4c

这部分要实现代换密码，即将字符串中的所有元音进行替换，替换的顺序是元音字母的任意重排序，实现代码如下：

class SubMessage(object)

```python
def __init__(self, text):
    self.message_text = text
    self.valid_words = load_words('./words.txt')
```

```python
def get_message_text(self):
    return self.message_text
```

```python
def get_valid_words(self):
    temp = self.valid_words.copy()
    return temp
```

```python
def build_transpose_dict(self, vowels_permutation):
    # 首先初始化一个字典，包含所有大写字母与小写字母
    dictionary = {}
    for letter in string.ascii_lowercase:
        dictionary[letter] = letter
    for letter in string.ascii_uppercase:
        dictionary[letter] = letter
    # 将元音替换掉
    dictionary['a'] = vowels_permutation[0]
    dictionary['e'] = vowels_permutation[1]
    dictionary['i'] = vowels_permutation[2]
    dictionary['o'] = vowels_permutation[3]
    dictionary['u'] = vowels_permutation[4]
    return dictionary
```

```
def apply_transpose(self, transpose_dict):
    message = self.get_message_text()
    new_message = ""
    for letter in message:
        if letter in transpose_dict:
            new_message += transpose_dict[letter]
        else:
            new_message += letter
    return new_message
```

class EncryptedSubMessage(SubMessage)

```
def __init__(self, text):
    self.message_text = text
    self.valid_words = load_words('./words.txt')
```

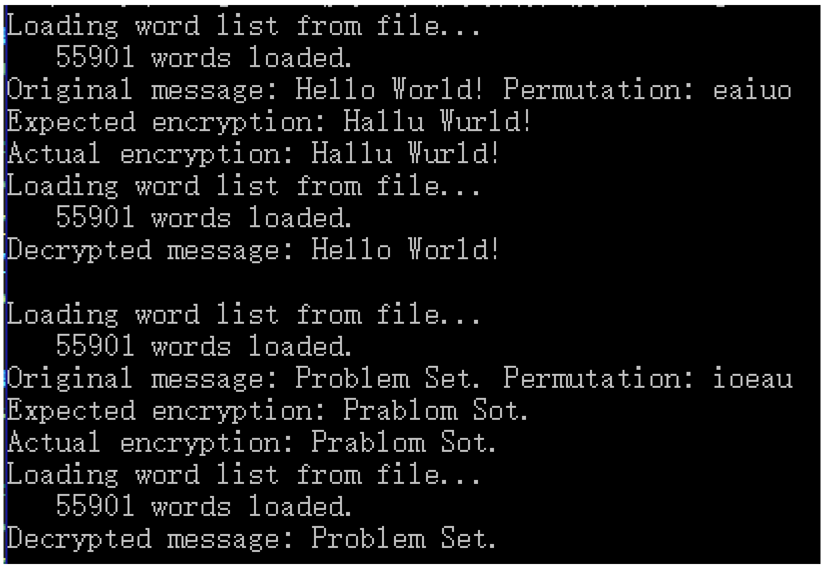
```
def decrypt_message(self):
    # 由之前编写的函数得到元音的全排列
    permutations = get_permutations(VOWELS_LOWER)
    # 尝试每种排列的元音组合
    count = []
    for permutation in permutations:
        dictionary = self.build_transpose_dict(permutation)
        message = self.apply_transpose(dictionary)
        message = message.split()
        flag = 0
        for word in message:
            if is_word(self.valid_words, word):
                flag += 1
        count.append(flag)

    # 如果所有的情况都没有正确的单词，返回原字符
    if not any(count):
        return self.message_text

    # 选出正确的单词数最多的一种
    ind = count.index(max(count))
    dictionary = self.build_transpose_dict(permutations[ind])
```

```python
        message = self.apply_transpose(dictionary)
        return message
```

实验结果



```
Loading word list from file...
   55901 words loaded.
Original message: Hello World! Permutation: eaiuo
Expected encryption: Hallu Wurld!
Actual encryption: Hallu Wurld!
Loading word list from file...
   55901 words loaded.
Decrypted message: Hello World!

Loading word list from file...
   55901 words loaded.
Original message: Problem Set. Permutation: ioeau
Expected encryption: Prablom Sot.
Actual encryption: Prablom Sot.
Loading word list from file...
   55901 words loaded.
Decrypted message: Problem Set.
```