암호분석 HW1 1번-20192233 박진철

(a)

1. Vigenere암호화/복호화를 수행하는 VigenereLib.py 라이브러리 생성

```python
Alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

#-- Vigenere 암호화
def vigenere_encrypt(key, msg):
    result = ''
    key_list = list(key.upper())
    key_pos= 0
    for ch in msg:
        if ch.upper() in Alphabet:
            key_ch = Alphabet.find(key_list[key_pos])
            idx = Alphabet.find(ch.upper())
            if ch.isupper():
                result += Alphabet[(idx+key_ch)%26].upper()
            else:
                result += Alphabet[(idx+key_ch)%26].lower()
        else:
            result += ch
        key_pos = (key_pos + 1) % len(key)
    return result
```

Vigenere암호화를 수행하는 vigenere_encrypt 함수

```python
#-- Vigenere 복호화
def vigenere_decrypt(key, msg):
    result = ''
    key_list = list(key.upper())
    key_pos= 0
    for ch in msg:
        if ch.upper() in Alphabet:
            key_ch = Alphabet.find(key_list[key_pos])
            idx = Alphabet.find(ch.upper())
            if ch.isupper():
                result += Alphabet[(idx-key_ch)%26].upper()
            else:
                result += Alphabet[(idx-key_ch)%26].lower()
        else:
            result += ch
        key_pos = (key_pos + 1) % len(key)
    return result
```

Vigenere복호화를 수행하는 vigenere_decrypt 함수

2. TEXT-1.txt 파일 암호화

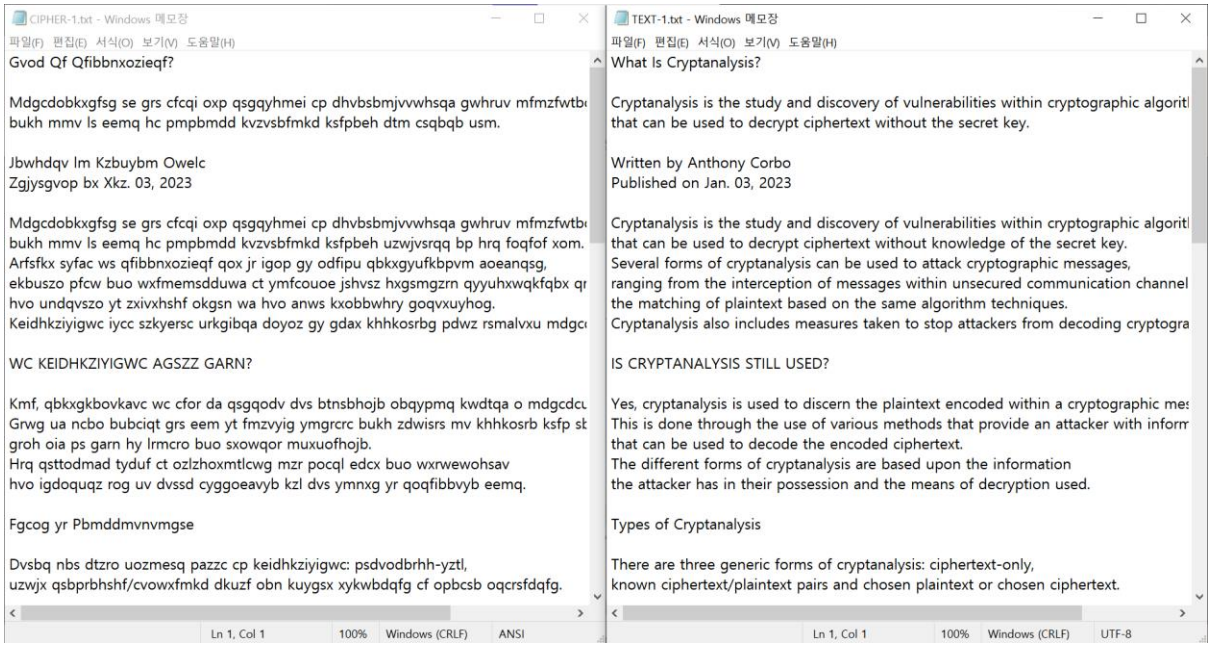
```python
import VigenereLib
import os, sys
import EngDicLib
import caesar_funLib


#1-(a)
in_file = "TEXT-1.txt"

InFileObj=open(in_file, 'rt', encoding='UTF8')
PT = InFileObj.read() #TEXT-1.txt 데이터를 PT에 저장
InFileObj.close()

key = 'KOOKMIN' #암호키: KOOKMIN
CT = VigenereLib.vigenere_encrypt(key, PT)
out_file = 'CIPHER-1.txt'
OutFileObj = open(out_file, 'w')
OutFileObj.write(CT) #암호문 CT를 CIPHER-1.txt에 저장
OutFileObj.close()
#============================================
```

TEXT-1.txt.에 있는 문장들을 Vigenere암호화하여 CIPHER-1.txt에 저장



암호문이 정상적으로 CIPHER-1.txt에 저장됨

(b)

1. CIPHER-1.txt파일 읽기

```
#1-(b)
in_file = 'CIPHER-1.txt'
InFileObj = open(in_file)
CT = InFileObj.read() #CIPHER-1.txt 데이터를 CT에 저장
InFileObj.close()
```

CIPHER-1.txt 문장을 CT에 저장

2. 암호키 길이 찾기

Index of Coincidence를 이용하여 암호키의 길이 찾기

IC(Index of Coincidence): 주어진 메시지에서 두 글자를 비복원 추출할 때 같은 글자가 나올 확률

각 행은 Caesar 암호 치환법이기 때문에, 키 길이만큼의 간격으로 암호문에서 문자를 얻어와 IC값 값 얻었을 때, 가장 높은 IC값이 나오는 길이가 암호키의 길이로 추측할 수 있음

```
#IC: Index of Coincidence
def IC(msg):
    AlphaDic = {}
    for ch in UpAlphabet:
        AlphaDic[ch] = 0
    num_alpha = 0
    for ch in msg:
        if ch.upper() in UpAlphabet:
            AlphaDic[ch.upper()] +=1
            num_alpha += 1
    ic = 0
    for ch in UpAlphabet:
        ic += AlphaDic[ch] * (AlphaDic[ch]-1)
    ic /= num_alpha*(num_alpha-1)
    return ic
```

문자열의 IC를 구하는 함수 IC함수를 EngDicLib.py 라이브러리에 저장

```
MAX_KEY_LENGTH = 8 # 암호키의 최대길이
keylen_candidate = 0 #후보키의 길이
max_ic = 0.0 #영문일수록 높아짐
for key_len in range(1,MAX_KEY_LENGTH+1):
    sub_msg = ''
    idx = 0
    while idx < len(CT):
        sub_msg += CT[idx] #키 길이만큼의 간격으로 암호문을 sub_msg에 저장
        idx += key_len
    sub_ic = EngDicLib.IC(sub_msg) #키 길이만큼의 간격으로 저장된 암호문의 IC값 계산
    if max_ic < sub_ic:
        max_ic = sub_ic
        keylen_candidate = key_len #IC값이 가장 높을때의 키길이를 후보키의 길이로 설정
    print('key_len =', key_len, ':', end='')
    print('sub_msg', sub_msg[:10],"...", '( length =', len(sub_msg), ')\t', end='')
    print('IC(sub_msg)= %6.4f' %(sub_ic))

print('------------------------------------')
```

각 길이당 IC값을 구하고, 가장 큰 IC값을 가지는 길이를 후보키의 길이로 설정

```
key_len = 1 :sub_msg Gvod Qf Qf ... ( length = 4075 )   IC(sub_msg)= 0.0443
key_len = 2 :sub_msg Go fQibxze ... ( length = 2038 )   IC(sub_msg)= 0.0448
key_len = 3 :sub_msg Gdffboe?Mc ... ( length = 1359 )   IC(sub_msg)= 0.0453
key_len = 4 :sub_msg G QbzfMdxg ... ( length = 1019 )   IC(sub_msg)= 0.0441
key_len = 5 :sub_msg GQiofdbs c ... ( length = 815 )    IC(sub_msg)= 0.0454
key_len = 6 :sub_msg GfbeMbgrq  ... ( length = 680 )    IC(sub_msg)= 0.0457
key_len = 7 :sub_msg G x?dsrisi ... ( length = 583 )    IC(sub_msg)= 0.0626
key_len = 8 :sub_msg GQzMx qs s ... ( length = 510 )    IC(sub_msg)= 0.0461
------------------------------------
```

출력 결과, 가장 높은 값을 가진 7이 키의 길이로 예상됨

우리가 알고있는 암호키인 KOOKMIN도 키의 길이는 7

3. 상대적인 키 찾기

키의 각 글자가 첫번째 글자에 비해서 얼마나 떨어져 있는가를 확인

```
key_list = [0]*keylen_candidate
for key_pos in range(1,keylen_candidate):
    key_ch_candidate = 0
    max_ic = 0.0
    for key_ch in range(0,26):
        sub_msg = ''
        idx = 0
        while idx < len(CT):
            sub_msg += CT[idx] #키의 첫번째 글자와의 차이값만큼의 간격으로 암호문을 sub_msg에 저장
            if (idx+key_pos) < len(CT):
                sub_msg += caesar_funLib.caesar_dec(key_ch, CT[idx+key_pos])
            idx += keylen_candidate
        sub_ic = EngDicLib.IC(sub_msg)
        if max_ic < sub_ic:
            max_ic = sub_ic
            key_ch_candidate = key_ch #IC값이 가장 높을때의 값을 키 첫 글자와의 상대적 거리로 설정
    key_list[key_pos] = key_ch_candidate
    print('key[%d] : key_ch_candidate = %d' %(key_pos, key_ch_candidate))

print('------------------------------------')
```

키의 각 문자는 첫번째 글자와 얼마나 떨어져 있는가를 조사

첫 글자와의 차이값 만큼 IC값을 구하고, 가장 높은 IC값을 가진 수를 상대적 거리로 설정

```
key[1] : key_ch_candidate = 4
key[2] : key_ch_candidate = 4
key[3] : key_ch_candidate = 0
key[4] : key_ch_candidate = 2
key[5] : key_ch_candidate = 24
key[6] : key_ch_candidate = 3
-------------------------------------
```

우리가 알고있는 암호키인 KOOKMIN과 비교했을 때, 상대적 거리가 맞음

4. 암호키 찾기

이제 상대적 거리를 이용해 후보키를 만들고, 가장 영단어가 많이 나온 후보키를 암호키로 예상

```
for key_ch in range(0,26):
    dec_msg = ''
    key_pos= 0
    for ch in CT: #키의 첫번째 값을 A~Z로 설정
        key_now = (key_ch + key_list[key_pos]) % 26
        dec_msg += caesar_funLib.caesar_dec(key_now, ch)
        key_pos = (key_pos + 1) % keylen_candidate
    eng_percent = EngDicLib.percentEngWords(dec_msg)
    #후보키로 복호화한 문장에 영단어가 얼마나 있는지 계산

    print('key_ch =', key_ch, ':', end='')
    print('dec_msg', dec_msg[:10],"...", '( length =', len(dec_msg), ')\t', end='')
    print('Eng(dec_msg)= %5.2f %%' %(eng_percent*100))

    if EngDicLib.isEnglish(dec_msg):
        key_0_candidate, rightPT = key_ch, dec_msg
        #자연스러운 영문장이 맞다면, 해당 후보키가 암호키가 맞다고 예상

if key_0_candidate >= 0:
    rightkey = ''
    for idx in key_list:
        rightkey += VigenereLib.Alphabet[(key_0_candidate + idx) % 26]
        #rightkey에 예상한 암호키를 저장

    print('right key =', rightkey)
    print('PT = ', rightPT[:20], '...', rightPT[-10:])
#============================================
```

키의 첫번째 값을 A~Z로 설정하고, Caesar 복호화를 진행하여 영단어가 얼마나 있는지를 계산

```
#복호화한 문서에서 영어단어의 비율
def percentEngWords(msg):
    msg = msg.lower()
    msg = removeNonletters(msg)
    possibl_words = msg.split() #문자열을 리스트로
    if possibl_words == []: #0으로 나누기 방지
        return 0.0
    count_words = 0
    for word in possibl_words:
        if word in EnglishDic: #사전에 있는 단어인가?
            count_words +=1
    return float(count_words)/len(possibl_words)
```

영단어의 퍼센트를 알려주는 함수 percentEngWords를 EngDicLib.py 라이브러리에 저장

자연스러운 영단어가 생긴 후보키를 암호키로 예상

```
62  #영어인지 판정하기
63  def isEnglish(msg, wordPer=20, letterPer=80):
64      wordMatch = percentEngWords(msg)*100 >= wordPer
65
66      numletters = len(removeNonletters(msg))
67      messageLettersPer = float(numletters)*100/len(msg)
68
69      letterMatch = messageLettersPer >= letterPer
70
71      return wordMatch and letterMatch
```

해당 메시지가 영어인지를 확인하는 함수 isEnglish를 EngDicLib.py 라이브러리에 저장

마지막으로 rightkey에 예상한 암호키를 저장하여 복호화

```
key_ch = 0 :dec_msg Grkd Sc Mb ... ( length = 4075 )    Eng(dec_msg)=  0.00 %
key_ch = 1 :dec_msg Fqjc Rb La ... ( length = 4075 )    Eng(dec_msg)=  0.00 %
key_ch = 2 :dec_msg Epib Qa Kz ... ( length = 4075 )    Eng(dec_msg)=  0.35 %
key_ch = 3 :dec_msg Doha Pz Jy ... ( length = 4075 )    Eng(dec_msg)=  0.53 %
key_ch = 4 :dec_msg Cngz Oy Ix ... ( length = 4075 )    Eng(dec_msg)=  0.35 %
key_ch = 5 :dec_msg Bmfy Nx Hw ... ( length = 4075 )    Eng(dec_msg)=  0.35 %
key_ch = 6 :dec_msg Alex Mw Gv ... ( length = 4075 )    Eng(dec_msg)=  0.00 %
key_ch = 7 :dec_msg Zkdw Lv Fu ... ( length = 4075 )    Eng(dec_msg)=  0.00 %
key_ch = 8 :dec_msg Yjcv Ku Et ... ( length = 4075 )    Eng(dec_msg)=  0.00 %
key_ch = 9 :dec_msg Xibu Jt Ds ... ( length = 4075 )    Eng(dec_msg)=  3.86 %
key_ch = 10 :dec_msg What Is Cr ... ( length = 4075 )   Eng(dec_msg)= 52.46 %
key_ch = 11 :dec_msg Vgzs Hr Bq ... ( length = 4075 )   Eng(dec_msg)=  0.18 %
key_ch = 12 :dec_msg Ufyr Gq Ap ... ( length = 4075 )   Eng(dec_msg)=  0.00 %
key_ch = 13 :dec_msg Texq Fp Zo ... ( length = 4075 )   Eng(dec_msg)=  0.00 %
key_ch = 14 :dec_msg Sdwp Eo Yn ... ( length = 4075 )   Eng(dec_msg)=  0.00 %
key_ch = 15 :dec_msg Rcvo Dn Xm ... ( length = 4075 )   Eng(dec_msg)=  0.00 %
key_ch = 16 :dec_msg Qbun Cm Wl ... ( length = 4075 )   Eng(dec_msg)=  0.00 %
key_ch = 17 :dec_msg Patm Bl Vk ... ( length = 4075 )   Eng(dec_msg)=  0.00 %
key_ch = 18 :dec_msg Ozsl Ak Uj ... ( length = 4075 )   Eng(dec_msg)=  0.00 %
key_ch = 19 :dec_msg Nyrk Zj Ti ... ( length = 4075 )   Eng(dec_msg)=  0.00 %
key_ch = 20 :dec_msg Mxqj Yi Sh ... ( length = 4075 )   Eng(dec_msg)=  0.00 %
key_ch = 21 :dec_msg Lwpi Xh Rg ... ( length = 4075 )   Eng(dec_msg)=  0.18 %
key_ch = 22 :dec_msg Kvoh Wg Qf ... ( length = 4075 )   Eng(dec_msg)=  0.00 %
key_ch = 23 :dec_msg Jung Vf Pe ... ( length = 4075 )   Eng(dec_msg)=  2.98 %
key_ch = 24 :dec_msg Itmf Ue Od ... ( length = 4075 )   Eng(dec_msg)=  0.00 %
key_ch = 25 :dec_msg Hsle Td Nc ... ( length = 4075 )   Eng(dec_msg)=  0.00 %
right key = KOOKMIN
PT =  What Is Cryptanalysi ... d more.
```

실행결과, 52.46%로 가장 높은 10(K)가 암호키의 첫번째 글자로 예상

right key = KOOKMIN으로 암호키와 동일함