암호분석 과제 보고서 : 1번, 2번 문제 해설

20192210 김민기

1번 문제:

```
mingi 🔥  ~/Desktop/mg00/2023_1/ambun2023/HW1  python3 AttackVigenere.py
input file (TEXT-1.txt) : What Is Cr ... d more.



This will overwrite the file CIPHER-1.txt. (C)ontinue or (Q)uit
> C
Output file (CIPHER-1.txt) : Dlle Pw Nf ... k xzfl.



input file (CIPHER-1.txt) : Dlle Pw Nf ... k xzfl.
```

TEXT-1.txt 파일을 암호화해서 CIPHER-1.txt 파일로 만들었다.

key는 'HELLO' 를 사용했다.

```
50  #-- 암호문 만들기
51
52  key = 'HELLO'
53  CT = VigenereLib.vigenere_encrypt(key, PT)
```

1) 키 길이 찾기

```
89  MAX_KEY_LENGTH = 8 # 암호키의 최대길이 설정(가정)
90  '''
91  PT = This is a
92  key CBDACBDAC
93  num 213021302
94  CT = Vils jv c
95  '''
96
97  keylen_candidate = 0    #후보 키 길이
98  max_ic = 0.0    # Index of Coincedence : 랜덤이면 낮고, 영문이면 높다
99  for key_len in range(1,MAX_KEY_LENGTH+1):
100     sub_msg = ''        #일정한 간격(key_len)으로 암호문을 추출
101     idx = 0
102     while idx < len(CT):    # index 사이 것들은 버리고 sub에 넣기
103         sub_msg += CT[idx]
104         idx += key_len
105
106     sub_ic = EngDicLib.IC(sub_msg)
107     if max_ic < sub_ic:
108         max_ic = sub_ic
109         keylen_candidate = key_len
110     print('key_len =', key_len, ':', end='')
111     print('sub_msg', sub_msg[:10],"...", '( length =', len(sub_msg), ')\t', end='')
112     print('IC(sub_msg)= %6.4f' %(sub_ic))
113
114 # 찾은 키 길이 출력
115 print('key length =', keylen_candidate)
```

최대 키 길이를 8로 정해놓고 키 길이 간격으로 추출한 암호문의 IC를 조사한다.
만약 키 길이가 8이하라면, 1~8중에 영문에 근접한 IC를 나타내는 추출 암호문이 존재할 것이다.

```
 ( length = 4075 )
key_len = 1 :sub_msg Dlle Pw Nf ... ( length = 4075 )   IC(sub_msg)= 0.0444
key_len = 2 :sub_msg Dl wNfebpd ... ( length = 2038 )   IC(sub_msg)= 0.0446
key_len = 3 :sub_msg Dewfehd?Qa ... ( length = 1359 )   IC(sub_msg)= 0.0442
key_len = 4 :sub_msg D NepzQeww ... ( length = 1019 )   IC(sub_msg)= 0.0435
key_len = 5 :sub_msg DPfhzyup z ... ( length = 815 )    IC(sub_msg)= 0.0693
key_len = 6 :sub_msg DwedQuwso  ... ( length = 680 )    IC(sub_msg)= 0.0436
key_len = 7 :sub_msg D b?epsmmj ... ( length = 583 )    IC(sub_msg)= 0.0453
key_len = 8 :sub_msg DNpQw om p ... ( length = 510 )    IC(sub_msg)= 0.0436
key length = 5
```

키 길이를 5라고 가정하고 다음 단계로 넘어간다.

2) 암호문 첫 글자 기준으로 키의 상대적 인덱스 찾기

암호문의 첫 글자와 다음 글자의 관계를 생각해보자.
CT[0] = PT[0] + K[0] 이고, CT[1] = PT[1] + K[1] 이다.
K[1] = (K[0] + k) mod 26 인 k가 존재할 것이다. → K[1] – k = K[0]로 생각하자.
CT[1] – k = CT'[1] = PT[1] + K[1] – k = PT[1] + K[0] 를 만족하는 k를 찾아야한다.
암호문의 첫 글자와 두번째 글자를 예시로 들어보면, 키 길이만큼 건너뛰며 암호문을 해독한 결과를 저장하는데, 이때 해독 키를 0부터 25까지 모두 해본다. 26가지 중 가장 IC가 높은 것을 택한다. 이렇게 진행하면 암호키의 첫번째 글자가 무엇인진 모르지만, 암호키의 첫번째 글자와 두번째 글자의 인덱스 차이를 알 수 있다. 이 작업을 1부터 키 길이까지 반복한다.

```
key[1] : key_ch_candidate = 23
key[2] : key_ch_candidate = 4
key[3] : key_ch_candidate = 4
key[4] : key_ch_candidate = 7
```

key 리스트에는 [0, 23, 4, 4, 7] 이 저장된다.

3) 키의 인덱스 관계를 활용하여 복호화 하기
첫 암호키를 A ~ Z 모두 대입해본다.

```
204     eng_percent = EngDicLib.percentEnglishWords(dec_msg)
```

영어 사전에 몇 퍼센트정도 있는지 확인해 본 후에 제일 높은 복호화 결과를 키로 확정 짓는다.

```
right key = HELLO
```

2번 문제:

```python
def subst_encrypt(key, msg):
    result = ''
    Inset = Alphabet
    OutSet = key

    for ch in msg:
        if ch.upper() in Inset:
            idx = Inset.find(ch.upper())
            if ch.isupper():
                result += OutSet[idx].upper()
            else:
                result += OutSet[idx].lower()
        else:
            result += ch

    return result
```

```python
def subst_decrypt(key, msg):
    result = ''
    Inset = key
    OutSet = Alphabet

    for ch in msg:
        if ch.upper() in Inset:
            idx = Inset.find(ch.upper())
            if ch.isupper():
                result += OutSet[idx].upper()
            else:
                result += OutSet[idx].lower()
        else:
            result += ch

    return result
```

암호화와 복호화는 Inset과 Outset을 바꾸어 넣으면 된다.

```python
def make_random_subst_key():
    alpha_list = list(Alphabet)
    random.seed(time.time())
    random.shuffle(alpha_list)
    shuffled = ''.join(alpha_list)
    print('Shuffled random key =', shuffled)

    return shuffled
```

랜덤으로 키를 생성해주는 함수이다. seed를 타임 함수로 초기화 시키고 알파벳을 섞는다.
섞인 리스트를 join함수로 str형으로 만들어 준 뒤에 반환한다.

```python
# Key가 유효한지 확인하는 함수
def confirm_key(key):
    letterCount = {'A':0, 'B':0, 'C':0, 'D':0, 'E':0, 'F':0, 'G':0, 'H':0,
                   'I':0, 'J':0, 'K':0, 'L':0, 'M':0, 'N':0, 'O':0, 'P':0,
                   'Q':0, 'R':0, 'S':0, 'T':0, 'U':0, 'V':0, 'W':0, 'X':0,
                   'Y':0, 'Z':0}
    for ch in key.upper():
        if ch in Alphabet:
            letterCount[ch] += 1

    for ch in Alphabet:
        if letterCount[ch] != 1:
            return False
    return True
```

만들어진 키가 유효한지 확인하는 함수이다.
알파벳 사전을 만든 뒤에 키 안에 들어있는 알파벳을 순서대로 대입하고 사전의 Value가 1이 아니면 False를 반환한다.

```
def get_letter_count(msg):
    letterCount = {'A':0, 'B':0, 'C':0, 'D':0, 'E':0, 'F':0, 'G':0, 'H':0,
                   'I':0, 'J':0, 'K':0, 'L':0, 'M':0, 'N':0, 'O':0, 'P':0,
                   'Q':0, 'R':0, 'S':0, 'T':0, 'U':0, 'V':0, 'W':0, 'X':0,
                   'Y':0, 'Z':0}
    for ch in msg.upper():
        if ch in Alphabet:
            letterCount[ch] += 1
```

마찬가지로 사전을 만들어 알파벳을 세는 함수이다.

```
def get_freq_order(msg):
    # (letter, freq 사전 만들기)
    letter2freq = get_letter_count(msg)
    # (freq, letter 사전 만들기)
    freq2letter = {}
    for ch in Alphabet:
        if letter2freq[ch] not in freq2letter:
            freq2letter[letter2freq[ch]] = [ch]   # [ch]로 하면 리스트로 저장됨?
        else:
            freq2letter[letter2freq[ch]].append(ch)

    for freq in freq2letter:
        freq2letter[freq].sort(key=Alpha_freq.find, reverse = False)
        freq2letter[freq] = ''.join(freq2letter[freq])  # value들을 str로 바꿔주기
    #print(freq2letter)

    freqpairs = list(freq2letter.items())
    #print('FreqPairs = ', freqpairs)
    freqpairs.sort(key=lambda x : x[0], reverse = True)
    # freqPairs.sort(key=getItemAtIndexZero, reverse=True) 이건 ppt
    freq_order_list = []
    for freq_pair in freqpairs:
        freq_order_list.append(freq_pair[1])
    freq_order_str = ''.join(freq_order_list)
    #print('freq_order_str =', freq_order_str)

    return freq_order_str
```

주어진 메시지에서 알파벳 빈도를 확인하는 함수이다.
key를 알파벳으로 갖는 사전을 생성한 뒤에 숫자를 증가시켜 letter : freq사전을 만든다.
그 후 key와 value를 바꾸는 작업을 진행한다 → freq : letter 사전이 만들어진다.
list로 저장된 letter를 str로 바꾸기 전에 알파벳 빈도 (ETAOINSHRDLCUMWFGYPBVKJXQZ)를 기준으로 정렬한다. (빈도가 높은 것이 앞에 오게 → find() 함수에서는 작은 순이므로 reverse = False)
사전을 리스트로 변환하고 각 리스트의 첫번째 인자를 기준으로 정렬한다.
내가 쓴 함수 : freqpairs.sort(key=lambda x : x[0], reverse = True) # 빈도가 큰 순으로 정렬됨

만들어진 리스트의 두번째 인자 값만 모아서 str로 만들어서 반환한다.

```
Shuffled random key = ZPWFGQXBKHMLCNJSIVEOURDAYT
PT frequency order : ETAINSORCHLPDYGMUFWXKBVQZJ
CT frequency order : GOZKNEJVWBLSFYXCUQDAMPRITH
Alphabet frequency order : ETAOINSHRDLCUMWFGYPBVKJXQZ
```

실행 결과 이다.