암호분석 HW1 3번-20192233 박진철

# (a)

1. 파일에서 암호문 받아오기

```
#1. 파일에서 암호문 받아오기
in_file = "CIPHER-2.txt"
InFileObj=open(in_file, 'rt', encoding='UTF8')
CT = InFileObj.read()
InFileObj.close()
#============================================
```

2. 임의의 키 설정

```
#2. 임의의 키 설정
my_key = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
#============================================
```

분석을 통해 얻은 키값들은 my_key에 저장

3. 빈도수 조사

```
#3. 빈도수 조사
UpAlphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
def getLetterCount(message):
    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 char in message.upper():
        if char in UpAlphabet:
            letterCount[char] += 1
    return letterCount

def getItemAtIndexZero(items):
    return items[0]

ETAOIN = 'ETAOINSHRDLCUMWFGYPBVKJXQZ'

def findfreq(message):
    letter2freq = getLetterCount(message)
    freq2letter = {}
    for char in UpAlphabet:
        if letter2freq[char] not in freq2letter:
            freq2letter[letter2freq[char]] = [char]
        else:
            freq2letter[letter2freq[char]].append(char)

    for freq in freq2letter:
        freq2letter[freq].sort(key=ETAOIN.find, reverse=False)
        freq2letter[freq] = ''.join(freq2letter[freq])
    freqPairs = list(freq2letter.items())
    freqPairs.sort(key=getItemAtIndexZero, reverse=True)
    freqOrder = []
    for freq_pair in freqPairs:
        freqOrder.append(freq_pair[1])
        freq_order_str = ''.join(freqOrder)
    return freq_order_str

CTfreq = findfreq(CT)
print('CT frequency=', CTfreq)
#============================================
```

앞서 2번에서 만든 빈도 분석 함수를 이용해 암호문의 빈도수 조사

```
CT frequency= TKAQVZPGWDHUNSYXOMFLRCIBEJ
```

암호문의 빈도수 출력

4. 빈도수를 통해 e 찾기

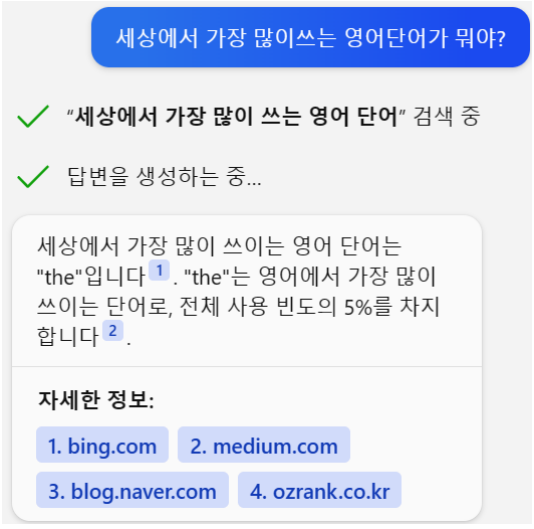
```
53  #4. 빈도를 통해 E 찾기
54  CTL = CT.upper()
55  #암호된 부분은 대문자, 복호화된 부분은 소문자로 구분함
56
57  my_key = my_key.replace('E', CTfreq[0].lower())
58  #가장 빈도수가 높은 알파벳을 e로 변경
59
60  CTL = CTL.replace(CTfreq[0], 'e')
61  #암호문에서 빈도수가 높은 알파벳을 e로 변경
62
63  #print(CTL)
64  #============================================
```

암호화된 부분은 대문자, 복호화된 부분은 소문자로 구분 -> 암호문을 분석할 때 식별하기 좋음

일반적으로 e가 가장 많이 나오므로, 가장 많이 나온 알파벳인 T를 e로 설정

후보키에서 E부분을 t로 변경, CTL에서 T를 e로 변경

5. the 찾기




세상에서 가장 많이 쓰이는 단어는 the->e를 찾았으므로 the를 찾을 수 있을 것

```python
#5. the 찾기
CTLslice=CTL.split()

findxxe=[]
for char in CTLslice:
    if 'e' in char:
        if len(char) == 3: #e가 들어있고, 글자가 3개인 단어들을 선택
            findxxe.append(char)

countxxe={}
for i in findxxe:
    try: countxxe[i] += 1
    except: countxxe[i]=1
print('find the=', countxxe) #선택된 단어들의 등장 횟수를 사전형태로 저장

max_xxe=max(countxxe, key=countxxe.get) #가장 많이 나온 단어 찾기

my_key = my_key.replace('T', max_xxe[0].lower())
my_key = my_key.replace('H', max_xxe[1].lower())

CTL = CTL.replace(max_xxe[0], 't')
CTL = CTL.replace(max_xxe[1], 'h')
#print(CTL)
#==========================================
```

CTLslice에 분석중인 글의 단어들만 추출

findxxe에 e가 들어있고, 글자 수가 3개인 단어들만 저장

선택된 단어들 중 가장 많이 나온 단어를 max_xxe에 저장

```
find the= {'HWe': 3, 'KSe': 15, 'QeY': 1}
```
KSe==the로 추측

후보키에서 T부분을 k로 변경, CTL에서 K를 t로 변경

후보키에서 H부분을 s로 변경, CTL에서 S를 h로 변경

6. to 찾기

| Word | Parts of speech | OEC rank | COCA rank[9] | Dolch level | Polysemy |
|---|---|---|---|---|---|
| the | Article | 1 | 1 | Pre-primer | 12 |
| be | Verb | 2 | 2 | Primer | 21 |
| to | Preposition | 3 | 7, 9 | Pre-primer | 17 |
| of | Preposition | 4 | 4 | Grade 1 | 12 |
| and | Conjunction | 5 | 3 | Pre-primer | 16 |
| a | Article | 6 | 5 | Pre-primer | 20 |
| in | Preposition | 7 | 6, 128, 3038 | Pre-primer | 23 |

the 다음으로 be, to, of, and 등이 많이 쓰임

be는 be동사의 원형이라 안나올 가능성이 있음->t를 이용해 to 찾기

```
#6. to 찾기
CTLslice=CTL.split()
findtx=[]
for char in CTLslice:
    if 't' in char:
        if len(char) == 2:
            findtx.append(char)

counttx={}
for i in findtx:
    try: counttx[i] += 1
    except: counttx[i]=1
print('find to=', counttx)

max_tx=max(counttx, key=counttx.get)
my_key = my_key.replace('O', max_tx[1].lower())

CTL = CTL.replace(max_tx[1], 'o')
#print(CTL)
#=============================================
```

findtx에 t가 들어있고, 글자 수가 2개인 단어들만 저장

선택된 단어들 중 가장 많이 나온 단어를 max_tx에 저장

```
find to= {'tA': 11}
```
tA == to로 추측

후보키에서 O부분을 a로 변경, CTL에서 A를 o로 변경

7. of 찾기

o를 찾았으므로 of를 찾기

```
#7. of 찾기
CTLslice=CTL.split()
findox=[]
for char in CTLslice:
    if 'o' in char:
        if len(char) == 2 and 't' not in char :
            findox.append(char)

countox={}
for i in findox:
    try: countox[i] += 1
    except: countox[i]=1
print('find of=',countox)

max_ox=max(countox, key=countox.get)
my_key = my_key.replace('F', max_ox[1].lower())

CTL = CTL.replace(max_ox[1], 'f')
#print(CTL)
#=============================================
```

findox에 o가 들어있고, 글자 수가 2개이고, to가 아닌 단어들만 저장

선택된 단어들 중 가장 많이 나온 단어를 max_ox에 저장

```
find of= {'Do': 3, 'Go': 5, 'oP': 9, 'oW': 6, 'Mo': 1, 'No': 1, 'oQ': 1}
```

oP == of로 추측

후보키에서 F부분을 p로 변경, CTL에서 P를 f로 변경

8. 자연스러운 문장 만들기

```
theWe HWe thWee IeQeWZX foWUN of XWDFtHQHGDNZN: XZFheWteEt-oQGD,
VQoYQ XZFheWteEt/FGHZQteEt FHZWN HQM XhoNeQ FGHZQteEt oW XhoNeQ XZFheWteEt.
```

CTL의 문장 중 theWe HWe thWee 발견 -> there are three라는 문장으로 바꾸면 자연스러워짐

```
#8. theWe HWe thWee라는 문장 발견
#W->r, H->a로 바꾸면 there are three라는 자연스러운 문장이 됨
my_key = my_key.replace('R', 'w')
my_key = my_key.replace('A', 'h')

CTL = CTL.replace('W', 'r')
CTL = CTL.replace('H', 'a')
#print(CTL)
#============================================
```

후보키에서 R부분을 w로 변경, CTL에서 W를 r로 변경

후보키에서 A부분을 h로 변경, CTL에서 H를 a로 변경

9. and 찾기

a를 이용하여 and 찾기

```
#9. and 찾기
CTLslice=CTL.split()
findaxx=[]
for char in CTLslice:
    if 'a' in char:
        if len(char) == 3:
            findaxx.append(char)

countaxx={}
for i in findaxx:
    try: countaxx[i] += 1
    except: countaxx[i]=1
print('find and=', countaxx)

max_axx=max(countaxx, key=countaxx.get)
print(max_axx)
my_key = my_key.replace('N', max_axx[1].lower())
my_key = my_key.replace('D', max_axx[2].lower())

CTL = CTL.replace(max_axx[1], 'n')
CTL = CTL.replace(max_axx[2], 'd')
#print(CTL)
#============================================
```

findaxx에 a가 들어있고, 글자 수가 3개인 단어들만 저장

선택된 단어들 중 가장 많이 나온 단어를 max_axx에 저장

```
find and= {'Mfa': 1, 'are': 3, 'aQM': 7, 'haN': 1, 'UaD': 1, 'XaQ': 2, 'GaY': 2, 'Far': 1}
```

aQM == and로 추측

후보키에서 N부분을 q로 변경, CTL에서 Q를 n로 변경

후보키에서 D부분을 m로 변경, CTL에서 M을 d로 변경

10~14. 자연스러운 문장/단어 만들기

```
Yhat doeN a XrDFtanaGDNt do?
```

CTL의 문장 중 Yhat doeN a XrDFtanaGDNt do? 발견

->what does a XrDFtanaGDst do? 라는 문장으로 바꾸면 자연스러워짐

```
#10. Yhat doeN a XrDFtanaGDNt do?라는 문장 발견
#Y->w, N->s로 바꾸면 what does a XrDFtanaGDst do?라는 자연스러운 문장이 됨
my_key = my_key.replace('W', 'y')
my_key = my_key.replace('S', 'n')

CTL = CTL.replace('Y', 'w')
CTL = CTL.replace('N', 's')
#print(CTL)
#=============================================
```

후보키에서 W부분을 y로 변경, CTL에서 Y를 w로 변경

후보키에서 S부분을 n로 변경, CTL에서 N를 s로 변경

```
Vnown XZFherteEt/FGaZnteEt FaZrs and Xhosen FGaZnteEt or Xhosen XZFherteEt.
```

```
XrDFtanaGDsts are often assoXZated wZth IoOernUent aIenXZes or Gaw enforXeUent,
hZred to ensRre aIenXD enXrDFtZon Uethods are RF to Far wZth the XRrrent standards
```

CTL의 단어 중 Vnown, Xhosen, wZth이 자주발견

->known, chosen, with라는 단어로 바꾸면 자연스러워짐

```
#11. Vnown, Xhosen, wZth이라는 단어가 자주 보임
#V->k, X->c, Z->i로 바꾸면 known, chosen, with이라는 자연스러운 단어가 됨
my_key = my_key.replace('K', 'v')
my_key = my_key.replace('C', 'x')
my_key = my_key.replace('I', 'z')

CTL = CTL.replace('V', 'k')
CTL = CTL.replace('X', 'c')
CTL = CTL.replace('Z', 'i')
#print(CTL)
#=============================================
```

후보키에서 K부분을 v로 변경, CTL에서 V를 k로 변경

후보키에서 C부분을 x로 변경, CTL에서 X를 c로 변경

후보키에서 I부분을 z로 변경, CTL에서 Z를 i로 변경

```
deciFherinI this strinI of teEt wiGG aGso aGGow the attacker to decrDFt FGainteEt
```

CTL의 문장 중 wiGG aGso aGGow 발견->will also allow 라는 문장으로 바꾸면 자연스러워짐

```
#12. wiGG aGso aGGow라는 문장 발견
#G->l로 바꾸면 will also allow라는 자연스러운 단어가 됨
my_key = my_key.replace('L', 'g')

CTL = CTL.replace('G', 'l')
#print(CTL)
#============================================
```

후보키에서 L부분을 g로 변경, CTL에서 G를 l로 변경

```
known ciFherteEt/FlainteEt Fairs and chosen FlainteEt or chosen ciFherteEt.
```

CTL의 단어 중 ciFherteEt/FlainteEt이 자주발견

->ciphertext/plaintext라는 단어로 바꾸면 자연스러워짐

```
#13. ciFherteEt/FlainteEt라는 단어 발견
#F->p, E->x로 바꾸면 ciphertext/plaintext라는 자연스러운 단어가 됨
my_key = my_key.replace('P', 'f')
my_key = my_key.replace('X', 'e')

CTL = CTL.replace('F', 'p')
CTL = CTL.replace('E', 'x')
#print(CTL)
#============================================
```

후보키에서 P부분을 f로 변경, CTL에서 F를 p로 변경

후보키에서 X부분을 e로 변경, CTL에서 E를 x로 변경

```
what does a crDptanalDst do?
```

CTL의 단어 중 crDptanalDst이 자주발견

->cryptanalyst라는 단어로 바꾸면 자연스러워짐

```
#14. crDptanalDst라는 단어 발견
#D->y로 바꾸면 cryptanalyst라는 자연스러운 단어가 됨
my_key = my_key.replace('Y', 'd')

CTL = CTL.replace('D', 'y')
#print(CTL)
#================================================
```

후보키에서 Y부분을 d로 변경, CTL에서 D를 y로 변경

15. 남은 단어들 확인

아직 해결되지 않은 단어들을 찾아내기

```
#15. 남은 단어들 확인
CTLslice=CTL.split()
lea_word=[]
for char in CTLslice:
    if char.islower() == False:
        lea_word.append(char)
    #대문자가 하나라도 있는 단어를 찾음

count_lword={}
for i in lea_word:
    try: count_lword[i] += 1
    except: count_lword[i]=1
    #찾은 단어들의 등장횟수를 사전 형태로 저장
print('remain word=',count_lword)
#================================================
```

대문자가 하나라도 있는 단어를 lea_word에 저장

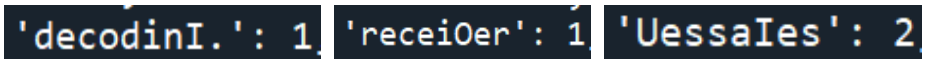
찾은 단어들의 등장횟수를 count_lword에 저장

```
remain word= {'iUUxlnR': 1, 'BUw': 4, 'CfBfyko': 1, 'oOaoklkOklUn': 7, 'hUOiC': 1, 'LUO':
10, 'oORRfok': 1, 'oUef?': 1, 'ofJfwyi': 2, 'CfBfyk': 2, 'Bfd': 1, 'UtklUno:': 1,
'sOltslOt': 1, '-': 4, 'yiiUdo': 4, 'kU': 10, 'fnhwLtkfC': 2, 'efooyRf': 4, 'ynC': 7,
'yOkUeyklhyiiL': 2, 'CfhwLtk': 3, 'OolnR': 1, 'JywlfkL': 4, 'UB': 8, 'kfhrnlsOfo,': 1,
'lnhiOClnR': 3, 'BwfsOfnhL': 1, 'ynyiLolo': 1, 'ClhklUnywL': 1, 'yioU': 3, 'eynOyiiL': 3,
'lntOk': 4, 'oOaoklkOklUno': 2, 'CfhwLtklUn': 3, 'twUhfoo.': 2, 'wOexln': 1, 'UBBfwo': 2,
'fnhwLtklUn': 2, 'kUUio,': 1, 'kUUio': 2, 'CfBfyklnR': 2, 'oUiJfw': 2, 'orUd': 1,
'CfhwLtkfC': 1, 'hrynRfo.': 1, 'aUcfnkwls': 1, 'oOaoklkOklUno.': 2, 'twUJlCfo': 1, 'oUef':
1, 'ROlCynhf': 1, 'Un': 2, 'rUd': 2, 'yttwUyhr': 1, 'hwLtkUhwyhx': 1, 'CUdniUyCyaif': 1,
'twURwye': 1, 'OofC': 1, 'Ukrfw': 1, 'kLtfo': 1, 'fnhwLtklUn.': 1, 'Oofo': 1, 'kfhrnlsOfo':
1, 'efooyRf,': 1, 'eynOyi': 1, 'nUkf': 1, 'rfitBOi': 1, 'nU': 1, 'ROywynkff': 1, 'krfL': 1,
'yidyLo': 1, 'oOhhfooBOi.': 1, 'okwfnRkr': 1, 'CftfnCo': 1, 'ByhkUwo,': 1, 'ifnRkr': 1,
'xfL': 1, 'otfhlBlh': 1, 'yiRUwlkre': 1, 'OofC.': 1, 'CU': 2, 'CfhLtk': 1, 'BUiiUdlnR': 1,
'efooyRf?': 1, 'Ieneric': 1, 'forUs': 1, 'aOailaJle': 1, 'theU': 1, 'decodinI.': 1, 'soUe':
1, 'eleUent': 1, 'Je': 3, 'aJle': 1, 'Uatch': 1, 'eleUents': 1, 'exaUple,': 1, 'coUpRter':
1, 'Uay': 1, 'JeIin': 1, '"loI': 1, 'decipherinI': 2, 'strinI': 1, 'Uatches': 1,
'throRIhoRt': 1, 'UessaIe.': 1, 'occRrs': 1, 'RnwittinIly': 1, 'caRses': 1, 'transUitter':
1, 'receiOer': 1, 'proOides': 1, 'aJRndance': 1, 'knowledIe,': 1, 'possiJly': 1, 'eOen': 1,
'knowledIe': 1, "UessaIe's": 1, 'twUayaiL,': 1, 'rywC': 1, 'kUU': 1, 'orUwk?': 1, 'yaUOk':
1, 'secRrity': 1, 'caRses,': 1, 'discoOer': 1, 'eOidence': 2, 'froU': 1, 'UessaIes': 2,
'Uore.': 2, 'IoOernUent': 2, 'aIencies': 2, 'enforceUent,': 1, 'ensRre': 1, 'aIency': 1,
'Uethods': 1, 'Rp': 1, 'cRrrent': 1, 'cyJersecRrity': 1, 'enIaIe': 1, 'UessaIes.': 1, 'Jy':
1, 'pRrposefRlly': 1, 'exploitinI': 1, 'Uentioned,': 1, 'orIaniLations': 1, 'eUploy': 1,
```

남은 단어들을 분석하여 암호키 찾기

대문자에 모든 알파벳을 대입해도 안되는 것들이 존재->암호문이 또 숨어있는 것을 알 수 있음

16~17. 자연스러운 단어 만들기




CTL의 단어 중 decodinI, receiOer, UessaIes이 발견

-> decoding, receiver, message 라는 단어로 바꾸면 자연스러워짐

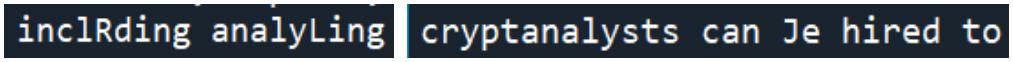
```
#16. decodinI, receiOer, UessaIe라는 단어 발견
#I->g, O->v, U->m으로 바꾸면 decoding, receiver, message라는 자연스러운 단어가 됨
my_key = my_key.replace('G', 'i')
my_key = my_key.replace('V', 'o')
my_key = my_key.replace('M', 'u')

CTL = CTL.replace('I', 'g')
CTL = CTL.replace('O', 'v')
CTL = CTL.replace('U', 'm')
#print(CTL)
#============================================
```

후보키에서 G부분을 i로 변경, CTL에서 I를 g로 변경

후보키에서 V부분을 o로 변경, CTL에서 O를 v로 변경

후보키에서 M부분을 u로 변경, CTL에서 U를 m로 변경




CTL의 단어 중 analyLing, inclRding, can Je이 발견

-> analyzing, including, can be라는 단어로 바꾸면 자연스러워짐

```
#17. analyLing, inclRding, can Je라는 단어 발견
#L->z, R->u, J->b로 바꾸면 analyzing, including, can be라는 자연스러운 단어가 됨
my_key = my_key.replace('Z', 'l')
my_key = my_key.replace('U', 'r')
my_key = my_key.replace('B', 'j')

CTL = CTL.replace('L', 'z')
CTL = CTL.replace('R', 'u')
CTL = CTL.replace('J', 'b')
#print(CTL)
#============================================
```

후보키에서 Z부분을 I로 변경, CTL에서 L를 z로 변경

후보키에서 U부분을 r로 변경, CTL에서 R를 u로 변경

후보키에서 B부분을 j로 변경, CTL에서 J를 b로 변경

18. 남은 키값 찾기

```
#18. 남은 키 찾기
new_key=my_key.upper()
othkey='ABCDEFGHIJKLMNOPQRSTUVWXYZ'

for ch in new_key:
    for i in UpAlphabet:
        if ch==i:
            othkey=othkey.replace(ch, '')
    #my_key에 아직 찾지 못한 알파벳을 othkey에 저장

ncha_key=[]
for ch in my_key:
    if ch.isupper()==True:
        ncha_key.append(ch)
        #my_key에서 대문자로 되어있는 알파벳을 ncha_key에 저장

print('remainkey=', othkey)
print('n_changedkey=', ncha_key)
print('my_key=', my_key)
#============================================
```

new_key에 my_key의 대문자버전을 넣어주고 newkey에 없는 알파벳을 othkey에 저장

my_key에서 아직 못 찾은 키값을 ncha_key에 저장

```
remainkey= BC
n_changedkey= ['J', 'Q']
my_key= hjxmtpiszJvguqafQwnkroyedl
```

B와 C가 아직 CLT에 남아있음, 후보키에서 J와 Q가 아직 치환이 되지 않음

19. 남은 영단어 찾기

15번과 같이 아직 해결되지 않은 단어들 찾아내기

```
#19. 남은 영단어 찾기
CTLslice=CTL.split()
engwords=[]
for char in CTLslice:
    if char.islower() == False:
        engwords.append(char)

engdic={}
for i in engwords:
    try: engdic[i] += 1
    except: engdic[i]=1
print('remainword=', engdic)
#============================================
```

대문자가 하나라도 있는 단어를 engword에 저장

찾은 단어들의 등장횟수를 engdic에 저장

```
remainword= {'Bmw': 4, 'CfBfyko': 1, 'hmviC': 1, 'CfBfyk': 2, 'Bfd': 1, '-': 4,
'fnhwztkfC': 2, 'ynC': 7, 'Cfhwztk': 3, 'mB': 8, 'lnhivClnu': 3, 'Bwfsvfnhz': 1,
'Clhklmnywz': 1, 'Cfhwztklmn': 3, 'mBBfwo': 2, 'CfBfyklnu': 2, 'CfhwztkfC': 1,
'twmblCfo': 1, 'uvlCynhf': 1, 'CmdnimyCyaif': 1, 'vofC': 1, 'rfitBvi': 1,
'ovhhfooBvi.': 1, 'CftfnCo': 1, 'Byhkmwo,': 1, 'otfhlBlh': 1, 'vofC.': 1, 'Cm': 2,
'Cfhztk': 1, 'Bmiimdlnu': 1, 'rywC': 1, 'ClBBlhvik?': 1, 'Bwvokwykfc.': 1}
```

CLT와 남은 영단어를 확인해보면 모두 암호문부분이 해석이 안됨

->my_key의 J와 Q에 b와 c중 아무거나 치환해도 암호문은 풀림

20. 임의로 키 설정하기

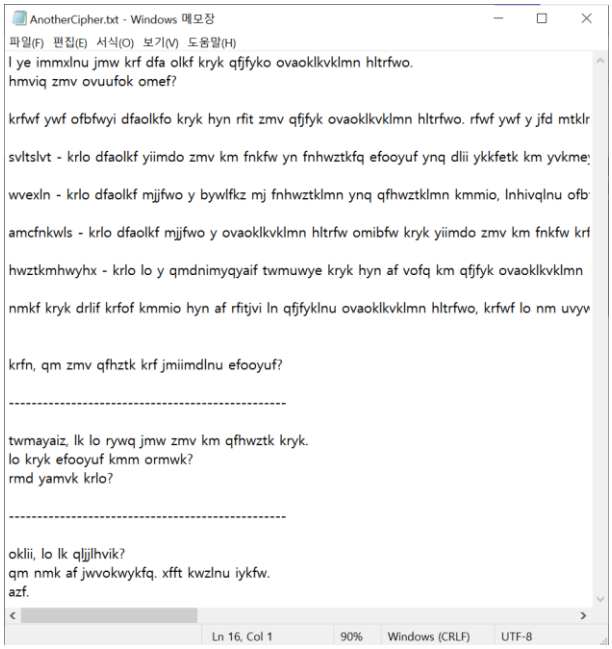
```
291    #20. 임의로 키 설정하기
292    #남은 것들은 또 다른 암호문에 있는 것들 뿐
293    #임의로 B->j, C->q로 변경
294    my_key = my_key.replace('J', 'b')
295    my_key = my_key.replace('Q', 'c')
296
297    CTL = CTL.replace('B', 'j')
298    CTL = CTL.replace('C', 'q')
299
300    #print(CTL)
301    print('key=',my_key)
```

남은 것들을 치환하여 my_key 완성

21. 또다른 암호문 해석

21-1. 또다른 암호문 저장

또다른 암호문을 AnotherCipher.txt에 저장



21-2. 공격을 위한 코드 생성

attack_anotherCipher.py 생성

```python
import SubstLib

in_file = "AnotherCipher.txt"
InFileObj=open(in_file, 'rt', encoding='UTF8')
CT = InFileObj.read()
InFileObj.close()

my_key = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' #임의의 키

UpAlphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

앞의 코드와 동일하게 CT저장, 임의의 키 지정

```python
def getLetterCount(message):
    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 char in message.upper():
        if char in UpAlphabet:
            letterCount[char] += 1
    return letterCount

#배열의 첫번째 원소를 주는 함수
def getItemAtIndexZero(items):
    return items[0]
```

```python
#빈도순서를 문자열로 정렬해주는 함수
def findfreq(message):
    letter2freq = getLetterCount(message)
    freq2letter = {}
    for char in UpAlphabet:
        if letter2freq[char] not in freq2letter:
            freq2letter[letter2freq[char]] = [char]
        else:
            freq2letter[letter2freq[char]].append(char)

    for freq in freq2letter:
        freq2letter[freq].sort(key=ETAOIN.find, reverse=False)
        freq2letter[freq] = ''.join(freq2letter[freq])
    freqPairs = list(freq2letter.items())
    freqPairs.sort(key=getItemAtIndexZero, reverse=True)
    freqOrder = []
    for freq_pair in freqPairs:
        freqOrder.append(freq_pair[1])
        freq_order_str = ''.join(freqOrder)
    return freq_order_str
```

```
#남은 단어들 확인하는 함수
def checkremainword(msg):
    msgslice=msg.split()
    lea_word=[]
    for char in msgslice:
        if char.islower() == False:
            lea_word.append(char)

    count_lword={}
    for i in lea_word:
        try: count_lword[i] += 1
        except: count_lword[i]=1
    #찾은 단어들의 등장횟수를 사전 형태로 저장
    print('remain word=',count_lword)
```

남은 단어들을 확인하는 checkremainword함수 추가

```
CTfreq = findfreq(CT) #CT의 빈도순서
print('CT frequency=', CTfreq)

CTL=CT.upper()
```

```
CT frequency= KFOMLYNWIVRHTZQJAUEDBXSCGP
```

21-3. e 찾기

처음에는 K를 e로 하려 했으나, the를 찾는 과정에서 이상이 생김

```
find the= {'eRF': 11, 'eMM': 1, 'NMe': 1}
```

K를 e로 바꿀 경우, e가 들어간 3글자 단어가 위의 3개가 나오는데, 이럴경우 the가 하나밖에 없다는 결과가 나옴

->K가 아닌 그 다음 F를 e로 설정

```
#21-3. e 찾기
my_key = my_key.replace('E', CTfreq[1].lower())

CTL = CTL.replace(CTfreq[1], 'e')
#=============================================
```

후보키에서 E부분을 f로 변경, CTL에서 F를 e로 변경

21-4. the 찾기

```
#21-4. the 찾기
CTLslice=CTL.split()

findxxe=[]
for char in CTLslice:
    if 'e' in char:
        if len(char) == 3: #e가 들어있고, 글자가 3개인 단어들을 선택
            findxxe.append(char)

countxxe={}
for i in findxxe:
    try: countxxe[i] += 1
    except: countxxe[i]=1
print('find the=', countxxe) #선택된 단어들의 등장 횟수를 사전형태로 저장

max_xxe=max(countxxe, key=countxxe.get) #가장 많이 나온 단어 찾기

my_key = my_key.replace('T', max_xxe[0].lower())
my_key = my_key.replace('H', max_xxe[1].lower())

CTL = CTL.replace(max_xxe[0], 't')
CTL = CTL.replace(max_xxe[1], 'h')
#print(CTL)
#============================================
```

```
find the= {'KRe': 11, 'DeA': 1, 'YWe': 2, 'JeD': 1, 'XeZ': 1}
```

후보키에서 T부분을 k로 변경, CTL에서 K를 t로 변경

후보키에서 H부분을 r로 변경, CTL에서 R를 h로 변경

21-5. to 찾기

```
#21-5. to 찾기
CTLslice=CTL.split()
findtx=[]
for char in CTLslice:
    if 't' in char:
        if len(char) == 2:
            findtx.append(char)

counttx={}
for i in findtx:
    try: counttx[i] += 1
    except: counttx[i]=1
print('find to=', counttx)

max_tx=max(counttx, key=counttx.get)
my_key = my_key.replace('O', max_tx[1].lower())

CTL = CTL.replace(max_tx[1], 'o')
#print(CTL)
#============================================
```

```
find to= {'tM': 10, 'Lt': 5}
```
tM == to로 추측

후보키에서 O부분을 m로 변경, CTL에서 M을 o로 변경

21-6. of 찾기

```
#21-6. of 찾기
CTLslice=CTL.split()
findox=[]
for char in CTLslice:
    if 'o' in char:
        if len(char) == 2 and 't' not in char :
            findox.append(char)

countox={}
for i in findox:
    try: countox[i] += 1
    except: countox[i]=1
print('find of=',countox)

max_ox=max(countox, key=countox.get)
my_key = my_key.replace('F', max_ox[1].lower())

CTL = CTL.replace(max_ox[1], 'f')
print(CTL)
#=============================================
```

```
find of= {'oJ': 8, 'oN': 2, 'No': 1, 'Qo': 2}
```
oJ == of로 추측

후보키에서 F부분을 j로 변경, CTL에서 J를 f로 변경

21-7. 자연스러운 문장 만들기

```
theWe YWe OeBeWYI DeAOLteO thYt HYN heIT ZoV QefeYt OVAOtLtVtLoN HLTheWO.
```

CTL의 문장 중 theWe YWe 발견 -> there are라는 문장으로 바꾸면 자연스러워짐

```
#21-7. theWe YWe라는 문장 발견
#W->r, Y->a로 바꾸면 there are라는 자연스러운 문장이 됨
my_key = my_key.replace('R', 'w')
my_key = my_key.replace('A', 'y')

CTL = CTL.replace('W', 'r')
CTL = CTL.replace('Y', 'a')
#print(CTL)
#=============================================
```

후보키에서 R부분을 w로 변경, CTL에서 W를 r로 변경

후보키에서 A부분을 y로 변경, CTL에서 Y를 a로 변경

21-8. and 찾기

```
#21-8. and 찾기
CTLslice=CTL.split()
findaxx=[]
for char in CTLslice:
    if 'a' in char:
        if len(char) == 3:
            findaxx.append(char)

countaxx={}
for i in findaxx:
    try: countaxx[i] += 1
    except: countaxx[i]=1
print('find and=', countaxx)

max_axx=max(countaxx, key=countaxx.get)
print(max_axx)
my_key = my_key.replace('N', max_axx[1].lower())
my_key = my_key.replace('D', max_axx[2].lower())

CTL = CTL.replace(max_axx[1], 'n')
CTL = CTL.replace(max_axx[2], 'd')
print(CTL)
#============================================
```

```
find and= {'are': 2, 'HaN': 4, 'aNQ': 7}
```
aNQ == and로 추측

후보키에서 N부분을 n로 변경, CTL에서 N를 n로 변경

후보키에서 D부분을 Q로 변경, CTL에서 Q을 d로 변경

21-9. 자연스러운 단어 만들기

```
thLO LO a doDnIoadaAIe TroUraE that Han Ae VOed to defeat OVAOtLtVtLon
```

CTL의 단어 중 doDnIoadaAIe 발견-> downloadable라는 단어로 바꾸면 자연스러워짐

```
#21-9. 자연스러운 단어 만들기
#doDnIoadaAIe라는 단어 발견
#D->w, I->l, A->b로 바꾸면 downloadable라는 자연스러운 단어가 됨
my_key = my_key.replace('W', 'd')
my_key = my_key.replace('L', 'i')
my_key = my_key.replace('B', 'a')

CTL = CTL.replace('D', 'w')
CTL = CTL.replace('I', 'l')
CTL = CTL.replace('A', 'b')
```

후보키에서 W부분을 d로 변경, CTL에서 D를 w로 변경

후보키에서 L부분을 i로 변경, CTL에서 I을 L로 변경

후보키에서 B부분을 a로 변경, CTL에서 A을 b로 변경

```
HWZTtMHWYHX - thLO LO Y QMDNIMYQYAIF TWMUWYE thYt HYN AF VOFQ tM QFJFYt OVAOtLtVtLMN
```

CTL의 문장 중 thLO LO 발견 -> this is라는 문장으로 바꾸면 자연스러워짐

```
190    #thLO LO라는 문장 발견
191    #L->i, O->s로 바꾸면 this is라는 자연스러운 단어가 됨
192    my_key = my_key.replace('I', 'l')
193    my_key = my_key.replace('S', 'o')
194
195    CTL = CTL.replace('L', 'i')
196    CTL = CTL.replace('O', 's')
197    #=============================================
```

후보키에서 I부분을 l로 변경, CTL에서 L를 i로 변경

후보키에서 S부분을 o로 변경, CTL에서 O을 s로 변경

```
HrZTtoHraHX - this is a downloadable TroUraE that Han be Vsed to defeat sVbstitVtion
```

CTL의 문장 중 Han be Vsed 발견 -> can be used라는 문장으로 바꾸면 자연스러워짐

```
197    #Han be Vsed라는 문장 발견
198    #H->c, V->u로 바꾸면 can be used라는 자연스러운 단어가 됨
199    my_key = my_key.replace('C', 'h')
200    my_key = my_key.replace('U', 'v')
201
202    CTL = CTL.replace('H', 'c')
203    CTL = CTL.replace('V', 'u')
204    print(CTL)
205    #=============================================
```

후보키에서 C부분을 h로 변경, CTL에서 H를 c로 변경

후보키에서 U부분을 v로 변경, CTL에서 V를 u로 변경

21-10. 남은 단어로 유추하기

```
207    #21-10. 남은 단어로 유추하기
208    checkremainword(CTL)
```

```
remain word= {'aE': 1, 'looXinU': 1, 'ciThers.': 3, 'Zou': 10, 'suUUest': 1,
'soEe?': 1, 'seBeral': 2, 'helT': 2, 'oTtions:': 1, 'SuiTSiuT': 1, '-': 4,
'encrZTted': 2, 'EessaUe': 4, 'atteETt': 2, 'autoEaticallZ': 2, 'decrZTt': 3,
'usinU': 1, 'BarietZ': 4, 'techniSues,': 1, 'includinU': 3, 'freSuencZ': 1,
'analZsis': 1, 'dictionarZ': 1, 'attacXs.': 1, 'EanuallZ': 3, 'inTut': 4,
'decrZTtion': 3, 'Trocess.': 2, 'ruEXin': 1, 'encrZTtion': 2, 'defeatinU': 2,
'ciTher': 3, 'solBer': 2, 'decrZTted': 1, 'EaXe': 1, 'chanUes.': 1, 'boCentriS': 1,
'TroBides': 1, 'soEe': 1, 'Uuidance': 1, 'aTTroach': 1, 'crZTtocracX': 1, 'TroUraE':
1, 'ciThers': 1, 'tZTes': 1, 'encrZTtion.': 1, 'techniSues': 1, 'EessaUe,': 1,
'Eanual': 1, 'helTful': 1, 'ciThers,': 1, 'Uuarantee': 1, 'theZ': 1, 'alwaZs': 1,
'strenUth': 1, 'deTends': 1, 'lenUth': 1, 'XeZ': 1, 'sTecific': 1, 'alUorithE': 1,
'decZTt': 1, 'followinU': 1, 'EessaUe?': 1,
'---------------------------------------------': 2, 'TrobablZ,': 1, 'XeeT': 1,
'trZinU': 1, 'bZe.': 1}
```

CTL의 단어 중 ciThers, includinU, decrZTtion, attacXs이 발견

-> ciphers, including, decryption, attacks 라는 단어로 바꾸면 자연스러워짐

```
210   #ciThers, includinU, decrZTtion, attacXs라는 단어 발견
211   #T->p, U->g, Z->y, X->k로 바꾸면
212   #ciphers, including, decryption, attacks라는 자연스러운 단어가 됨
213   my_key = my_key.replace('P', 't')
214   my_key = my_key.replace('G', 'u')
215   my_key = my_key.replace('Y', 'z')
216   my_key = my_key.replace('K', 'x')
217
218   CTL = CTL.replace('T', 'p')
219   CTL = CTL.replace('U', 'g')
220   CTL = CTL.replace('Z', 'y')
221   CTL = CTL.replace('X', 'k')
222
223   checkremainword(CTL)
```

후보키에서 P부분을 t로 변경, CTL에서 T를 p로 변경

후보키에서 G부분을 u로 변경, CTL에서 U를 g로 변경

후보키에서 Y부분을 z로 변경, CTL에서 Z를 y로 변경

후보키에서 K부분을 x로 변경, CTL에서 X를 k로 변경

설정한 후 다시 남은 단어 출력

```
remain word= {'aE': 1, 'soEe?': 1, 'seBeral': 2, 'SuipSiup': 1, '-': 4, 'Eessage':
4, 'atteEpt': 2, 'autoEatically': 2, 'Bariety': 4, 'techniSues,': 1, 'freSuency': 1,
'Eanually': 3, 'ruEkin': 1, 'solBer': 2, 'Eake': 1, 'boCentriS': 1, 'proBides': 1,
'soEe': 1, 'prograE': 1, 'techniSues': 1, 'Eessage,': 1, 'Eanual': 1, 'algorithE':
1, 'Eessage?': 1, '--------------------------------------------------': 2}
```

CTL의 단어 중 Eessage, freSuency, seBeral, boCentriS 이 발견

-> message, frequency, several, boxentriq 라는 단어로 바꾸면 자연스러워짐

```
225   #Eessage, freSuency, seBeral, boCentriS라는 단어 발견
226   #E->m, S->q, B->v, C->x로 바꾸면
227   #message, frequency, several, boxentriq라는 자연스러운 단어가 됨
228   my_key = my_key.replace('M', 'e')
229   my_key = my_key.replace('Q', 's')
230   my_key = my_key.replace('V', 'b')
231   my_key = my_key.replace('X', 'C')
232
233   CTL = CTL.replace('E', 'm')
234   CTL = CTL.replace('S', 'q')
235   CTL = CTL.replace('B', 'v')
236   CTL = CTL.replace('C', 'x')
237
238   checkremainword(CTL)
```

후보키에서 M부분을 e로 변경, CTL에서 E를 m로 변경

후보키에서 Q부분을 s로 변경, CTL에서 S를 q로 변경

후보키에서 V부분을 b로 변경, CTL에서 B를 v로 변경

후보키에서 X부분을 c로 변경, CTL에서 C를 x로 변경

설정한 후 다시 남은 단어 출력

```
remain word= {'-': 4, '-----------------------------------------------': 2}
```

모든 단어가 정상적으로 들어감

22. 평문으로 저장하기

만들어진 두 평문을 합하여 decryptedTEXT.txt에 저장

23. 검증하기

```python
import EngDicLib

in_file = "decryptedTEXT.txt"
InFileObj=open(in_file, 'rt', encoding='UTF8')
my_text = InFileObj.read()
InFileObj.close()

if EngDicLib.isEnglish(my_text):
    print('it is correct!')
else:
    print('it is wrong!')
```

만든 해독문을 isEnglish에 넣어 영단어가 제대로 되었는지 확인하는 makePT.py코드 생성

```
it is correct!
```

영단어가 잘 들어간 해독문임을 알 수 있음.