# Cryptanalysis (암호분석)

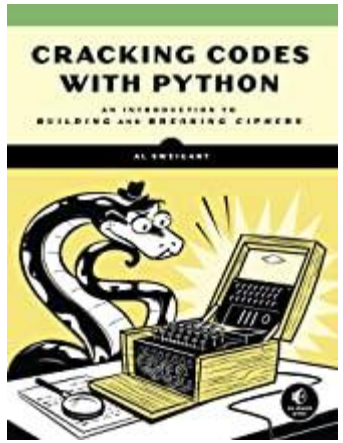
# – Python (Part 3) –

## (Substitution Cipher)

2020. 4

# 목차

1. 치환암호 Subst Cipher
2. 문자열의 빈도
3. 난수생성 및 활용
4. 단어사전을 이용한 치환암호 Subst Cipher 해독



# 치환암호 Subst Cipher

- 암호화 함수

```
Alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

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
```

```
my_key = 'VWXABCDEIJKFGHLMQRSNOPTUYZ'
ciphertext = subst_encrypt(my_key, message)
print(ciphertext)
```

message =
'Cryptography is the practice and study of techniques for secure communication in the presence of third parties (called adversaries). More generally, it is about constructing and analyzing protocols that overcome the influence of adversaries and which are related to various aspects in information security such as data confidentiality, data integrity, and authentication.'

ciphertext =
'Xrymnldrvmey is neb mrvxnixb vha snoay lc nbxehiqobs clr sbxorb xlggohixvnilh ih neb mrbsbhxb lc neira mvrnibs (xvffba vapbrsvribs). Glrb dbhbrvffy, in is vwlon xlhsnroxnihd vha vhvfyzihd mrlnlxlfs nevn lpbrxlgb neb ihcfobhxb lc vapbrsvribs vha teixe vrb rbfvnba nl pvrilos vsmbxns ih ihclrgvnilh sbxoriny soxe vs avnv xlhciabhnivfiny, avnv ihnbdriny, vha vonebhnixvnilh.'

# 치환암호 Subst Cipher

- 복호화 함수

```
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大
```

암호화와 복호화는 서로
InSet 과 OutSet의 역할만 바뀐다.

```
my_key = 'VWXABCDEIJKFGHLMQRSNOPTUYZ'
message = subst_encrypt(my_key, ciphertext)
print(message)
```

ciphertext =
'Xrymnldrvmey is neb mrvxnixb vha snoay lc nbxehiqobs clr sbxorb xlggohixvnilh ih neb mrbsbhxb lc neira mvrnibs (xvffba vapbrsvribs). Glrb dbhbrvffy, in is vwlon xlhsnroxnihd vha vhvfyzihd mrlnlxlfs nevn lpbrxlgb neb ihcfobhxb lc vapbrsvribs vha teixe vrb rbfvnba nl pvrilos vsmbxns ih ihclrgvnilh sbxoriny soxe vs avnv xlhciabhnivfiny, avnv ihnbdriny, vha vonebhnixvnilh.'

message =
'Cryptography is the practice and study of techniques for secure communication in the presence of third parties (called adversaries). More generally, it is about constructing and analyzing protocols that overcome the influence of adversaries and which are related to various aspects in information security such as data confidentiality, data integrity, and authentication.'

# 빈도 분석 – 빈도 사전 만들기

- 문자열에서 각 알파벳의 출현 횟수를 계산하는 함수

```
LETTERS = '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 LETTERS:
            letterCount[char] += 1

    return letterCount
```

(복습)
사전(dictionary) 데이터 타입
Dic = { key1 : value1, key2 : value2, ….}
참조 방식: Dic[key1] = value1

함수의 리턴 값으로
'사전'도 가능하다

```
{'A': 166, 'B': 24, 'C': 115, 'D': 76, 'E': 270, 'F': 43, 'G': 43,
 'H': 87, 'I': 174, 'J': 0, 'K': 10, 'L': 73, 'M': 69, 'N': 160,
 'O': 160, 'P': 63, 'Q': 5, 'R': 151, 'S': 125, 'T': 204, 'U': 58,
 'V': 22, 'W': 19, 'X': 3, 'Y': 54, 'Z': 2}
```

# 사전 반대로 만들기

영한사전 → 한영사전

- 사전의 key와 value의 역할 서로 바꾸기
  - Dic1 = { key1 : value1, key2 : value2, ….}
  - Dic2 = {value1 : key1, value2 key2, ….}

키, 값의 역할 바꾸기

```
letter2freq = getLetterCount(message)
freq2letter = {}

for char in LETTERS:

    if letter2freq[char] not in freq2letter:

            freq2letter[letter2freq[char]] = [char]
    else:
            freq2letter[letter2freq[char]].append(char)
```

처음 나오는 빈도(key)인가?

(key: value) = (빈도:문자)
예: 166: ['A']
추가하기

이미 있는 값에 추가하기
예: 43: ['F'] → 43: ['F', 'G']

**letter2freq**

{'A': 166, 'B': 24, 'C': 115, 'D': 76, 'E': 270, 'F': 43, 'G': 43,
'H': 87, 'I': 174, 'J': 0, 'K': 10, 'L': 73, 'M': 69, 'N': 160,
'O': 160, 'P': 63, 'Q': 5, 'R': 151, 'S': 125, 'T': 204, 'U': 58,
'V': 22, 'W': 19, 'X': 3, 'Y': 54, 'Z': 2}

**freq2letter**

{166: ['A'], 24: ['B'], 115: ['C'], 76: ['D'], 270: ['E'], **43: ['F', 'G']**,
87: ['H'], 174: ['I'], 0: ['J'], 10: ['K'], 73: ['L'], 69: ['M'], **160: ['N', 'O']**,
63: ['P'], 5: ['Q'], 151: ['R'], 125: ['S'], 204: ['T'], 58: ['U'], 22: ['V'], 19:
['W'], 3: ['X'], 54: ['Y'], 2: ['Z']}

# 리스트 정렬하기 (sort)

- 리스트 정렬 함수 – sort()
  - key: 정렬의 기준이 될 값을 계산하는 **함수** 이름
  - reverse: 순방향(True)/역방향(False) 선택

```
# 영문자 빈도순서
ETAOIN = 'ETAOINSHRDLCUMWFGYPBVKJXQZ'

list1 = [ 'F', 'G', 'N', 'O']
list1.sort(key= ETAOIN.find, reverse=True)
print(list1)
```

큰 값을 앞쪽에

```
['G', 'F', 'N', 'O']
```

ETAOIN.find

```
ETAOIN.find('F') = 15
ETAOIN.find('G') = 16
ETAOIN.find('N') = 5
ETAOIN.find('O') = 3
```

```
print(''.join(list1))
```

리스트를 문자열로

```
'GFNO'
```

# 빈도사전의 정렬(1) – 리스트(값) 정렬

- 각 값(문자의 리스트) 정렬
  - 동일한 출현 빈도(key)인 문자들(동점자들)을 정렬하는 방법
  - 일반적인 영문빈도의 순서로 정렬하고 문자열로 변경

```
freq2letter = {166: ['A'], 24: ['B'], 115: ['C'], 76: ['D'], 270: ['E'],
   43: ['F', 'G'], 87: ['H'], 174: ['I'], 0: ['J'], 10: ['K'], 73: ['L'],
   69: ['M'], 160: ['N', 'O'], 63: ['P'], 5: ['Q'], 151: ['R'], 125: ['S'],
   204: ['T'], 58: ['U'], 22: ['V'], 19: ['W'], 3: ['X'], 54: ['Y'], 2: ['Z']}

for freq in freq2letter:

    freq2letter[freq].sort(key=ETAOIN.find, reverse=False)

    freq2letter[freq] = ''.join(freq2letter[freq])

print(freq2letter)
```

작은 값을 앞쪽에

동일한 빈도(key)에
두개 이상의 문자가 대응된
경우만 정렬된다.

{166: 'A', 24: 'B', 115: 'C', 76: 'D', 270: 'E', **43: 'FG'**, 87: 'H', 174: 'I', 0: 'J',
10: 'K', 73: 'L', 69: 'M', **160: 'ON'**, 63: 'P', 5: 'Q', 151: 'R', 125: 'S', 204: 'T',
58: 'U', 22: 'V', 19: 'W', 3: 'X', 54: 'Y', 2: 'Z'}

# 빈도사전의 정렬(2) – 사전 정렬

- 빈도사전의 문자를 빈도순으로 정렬하고 문자열로 변환

```
freq2letter = {166: 'A', 24: 'B', 115: 'C', 76: 'D', 270: 'E',
     43: 'FG', 87: 'H', 174: 'I', 0: 'J', 10: 'K', 73: 'L', 69: 'M',
     160: 'ON', 63: 'P', 5: 'Q', 151: 'R', 125: 'S', 204: 'T', 58: 'U',
     22: 'V', 19: 'W', 3: 'X', 54: 'Y', 2: 'Z'}

freqPairs = list(freq2letter.items())
print('FreqPairs = ',freqPairs)
```

사전을 리스트로 변환

```
FreqPairs =  [(166, 'A'), (24, 'B'), (115, 'C'), (76, 'D'), (270, 'E'), (43, 'FG'), (87, 'H'),
  (174, 'I'), (0, 'J'), (10, 'K'), (73, 'L'), (69, 'M'), (160, 'ON'), (63, 'P'), (5, 'Q'),
  (151, 'R'), (125, 'S'), (204, 'T'), (58, 'U'), (22, 'V'), (19, 'W'), (3, 'X'), (54, 'Y'), (2, 'Z')]
```

배열의 첫번째 값을 기준으로 정렬함

```
freqPairs.sort(key=getItemAtIndexZero, reverse=True)
print('FreqPairs = ',freqPairs)
```

```
FreqPairs = [(270, 'E'), (204, 'T'), (174, 'I'), (166, 'A'), (160, 'ON'), (151, 'R'), (125, 'S'),
(115, 'C'), (87, 'H'), (76, 'D'), (73, 'L'), (69, 'M'), (63, 'P'), (58, 'U'), (54, 'Y'), (43, 'FG'),
(24, 'B'), (22, 'V'), (19, 'W'), (10, 'K'), (5, 'Q'), (3, 'X'), (2, 'Z'), (0, 'J')]
```

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

# 빈도순서를 문자열로

- 사전정렬 결과를 이용하여 빈도순서를 문자열로 만들기

```
FreqPairs = [(270, 'E'), (204, 'T'), (174, 'I'), (166, 'A'), (160, 'ON'),
     (151, 'R'), (125, 'S'), (115, 'C'), (87, 'H'), (76, 'D'), (73, 'L'),
     (69, 'M'), (63, 'P'), (58, 'U'), (54, 'Y'), (43, 'FG'), (24, 'B'),
     (22, 'V'), (19, 'W'), (10, 'K'), (5, 'Q'), (3, 'X'), (2, 'Z'), (0, 'J')]

freqOrder = []

for freq_pair in freqPairs:
    freqOrder.append(freq_pair[1])
```

```
['E', 'T', 'I', 'A', 'ON', 'R', 'S', 'C', 'H', 'D', 'L', 'M', 'P', 'U', 'Y', 'FG', 'B', 'V', 'W', 'K', 'Q', 'X', 'Z', 'J']
```

```
freq_order_str =  ''.join(freqOrder)
print('freq_order_str = ', freq_order_str)
```

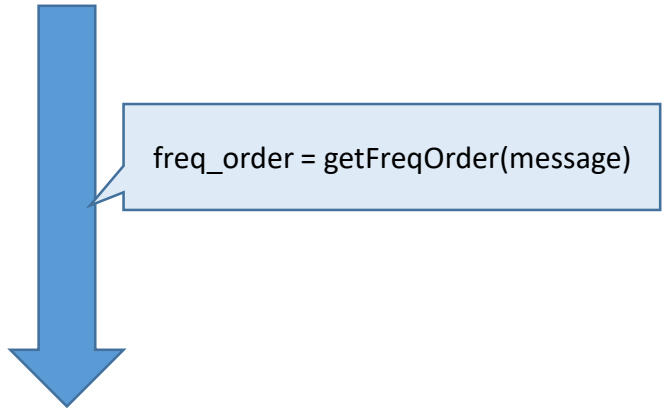
```
freq_order_str = ETIAONRSCHDLMPUYFGBVWKQXZJ
```

이제 이정도는 쉬워야 함!

# 알파벳 출현 빈도 – 종합

```
def getFreqOrder(message):

    #- (letter, freq) 사전 만들기: { 'A': 999, ... }
    letter2freq = getLetterCount(message)
    #- (freq, letter) 사전 만들기: { 999: 'A', ... }
    freq2letter = {}
    for char in LETTERS:
        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])

    return ''.join(freqOrder)
```

message =
'Cryptography is the practice and study of techniques for secure communication in the presence of third parties (called adversaries). More generally, it is about constructing and analyzing protocols that overcome the influence of adversaries and which are related to various aspects in information security such as data confidentiality, data integrity, and authentication.'




freq_order_str = ETIAONRSCHDLMPUYFGBVWKQXZJ

# 알파벳 빈도를 이용한 키 예측

- 키 예측 전략
  - 암호문의 알파벳 빈도를 계산한다.
  - 알파벳의 빈도와 영문 빈도순을 비교한다.
  - 대응되는 문자가 암호화된 것으로 판단하여 암호키를 예측한다.

```
def Freq2Key(freq_order):
    temp_dict = {}
    i=0
    for char in freq_order:
        temp_dict[ETAOIN[i]] = char
        i += 1
    temp_list = list(temp_dict.items())
    #print('temp_list =', temp_list)

    temp_list.sort(key=getItemAtIndexZero)
    #print('temp_list =', temp_list)

    temp_key_list = []
    for item in temp_list:
        temp_key_list.append(item[1])

    return ''.join(temp_key_list)
```

print()를 활용하여 계산과정을 파악해보자!

**기본 정보**

```
ETAOIN  = 'ETAOINSHRDLCUMWFGYPBVKJXQZ'  # 영문 빈도 순서
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

**암호문의 빈도**

```
FREQ =  'ETIAONRSCHDLMPUYFGBVWKQXZJ'
```

**암호키 예측**

```
KEY =  'IVLHEYFSOQKDPNABZCRTMWUXGJ'
```

# 난수 생성하기

- 랜덤한 숫자 만들기
  - 의사난수 생성기: 초기값(seed)로부터 결정되는 난수를 생성하는 알고리즘
  - 특징 통계적으로 난수성이 우수한 출력이 얻어지나
    같은 초기값을 사용하면 똑같은 난수가 재연됨

```
import random

# seed 설정 (고정값)
random.seed(2020)

for i in range(10):
    print(random.randint(1,100))
```

반복 실행시
동일한 난수 출력

```
import random

# seed 설정 (현재 시각)
random.seed(time.time())

for i in range(10):
    print(random.randint(1,100))
```

매번 다른
초기값으로
설정되어 실행할
때마다 다른 난수가
출력

1,2,..., 100
중에서 랜덤하게 출력

# 난수를 이용한 무작위 섞기(shuffling)

- 난수를 활용한 예제들

```
#-- 알파벳으로 랜덤 길이 랜덤 메시지 만들기
Alphabet    = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'       # 알파벳
alphabet_msg = Alphabet * random.randint(5,10)  # 알파벳을 반복한 문자열
alphabet_list_msg = list(alphabet_msg)          # 문자열 --> 리스트
random.shuffle(alphabet_list_msg)               # 리스트 랜덤셔플
shuffled_msg = ''.join(alphabet_list_msg)       # 리스트 --> 문자열
print(shuffled_msg)                             # 랜덤 메시지 출력
print('Random Message: "%s..."' % (shuffled_msg[:50])) # 간결한 출력
```

난수를 이용하여 랜덤한
암호키를 생성해보자!

# 사전을 이용한 Subst Cipher 해독

* 교재에 있으나 강의 동영상에는 포함되지 않은 부분입니다.
* Python에 어느 정도 익숙하면 쉽게 이해할 수 있습니다.

# 단어의 패턴 분석

- 단어를 패턴으로 변환

```
def getWordPattern(word):
    # Returns a string of the pattern form of the given word.
    # e.g. '0.1.2.3.4.1.2.3.5.6' for 'DUSTBUSTER'
    word = word.upper()
    nextNum = 0
    letterNums = {}
    wordPattern = []

    for letter in word:
        if letter not in letterNums:
            letterNums[letter] = str(nextNum)
            nextNum += 1
        wordPattern.append(letterNums[letter])
    return '.'.join(wordPattern)
```

```
word_list = [ 'apple', 'connection', 'birthday', 'happy', 'mommy', 'cloud' ]
for word in word_list:
    print(getWordPattern(word))
```

```
     'apple'    0.1.1.2.3
'connection'    0.1.2.2.3.0.4.5.1.2
 'birthday'     0.1.2.3.4.5.6.7
   'happy'      0.1.2.2.3
   'mommy'      0.1.0.0.2
   'cloud'      0.1.2.3.4
```

# 사전의 패턴 분석

- 사전파일('dictionary.txt') 내 모든 단어의 패턴 분석

```
import pprint
allPatterns = {}

    fo = open('dictionary.txt')
    wordList = fo.read().split('\n')
    fo.close()

    for word in wordList:
        # Get the pattern for each string in wordList:
        pattern = getWordPattern(word)

        if pattern not in allPatterns:
            allPatterns[pattern] = [word]
        else:
            allPatterns[pattern].append(word)

    # This is code that writes code. The wordPatterns.py file contains
    # one very, very large assignment statement:
    fo = open('wordPatterns.py', 'w')
    fo.write('allPatterns = ')
    fo.write(pprint.pformat(allPatterns))
    fo.close()
```

사전

단어가 Enter('\n')로 구분된 사전파일을 읽는다.

새로운 패턴을 만들고 단어를 추가

이미 존재하는 패턴에 단어를 추가

```
allPatterns = {'0.0.1': ['EEL'],
 '0.0.1.2': ['EELS', 'OOZE'],
 '0.0.1.2.0': ['EERIE'],
 '0.0.1.2.3': ['AARON', 'LLOYD', 'OOZED'],
 '0.0.1.2.3.4': ['AARHUS', 'EERILY'],
 '0.0.1.2.3.4.5.5': ['EELGRASS'],
 '0.1.0': ['ADA',
           'BIB',
           'BOB',
           'DAD',
           'DID',
           'DUD',
           'EKE',
           'ERE',
           'EVE',
           'EWE',
           'EYE',
           'GAG',
           'GIG',
           'HUH',
           'NAN',
           'NON',
           'NUN',
           'PEP',
           'PIP',
           'POP',
           'PUP',
           'SUS',
           'TIT'],
 '0.1.0.0': ['DODD', 'LULL'],
 '0.1.0.0.1': ['MAMMA'],
 '0.1.0.0.1.2': ['MAMMAL', 'MAMMAS', 'PAPPAS', 'PEPPER'],
 '0.1.0.0.1.2.1.3': ['PEPPERED'],
 '0.1.0.0.1.2.3': ['BABBAGE',
                   'BIBBING',
                   'LILLIAN',
                   'MAMMALS',
```

# 암호문 단어의 패턴 분석

- 예: 암호문 단어의 패턴 분석
  - 암호문 단어: 'XYX' ➔ 패턴: 0.1.0
  - 사전 분석으로 얻는 패턴파일에서
    'ADA', 'BIB', 'BOB', 'DAD', …, 'TIT' 중 하나임
  - 패턴으로부터 X, Y의 가능한 후보를 만들 수 있음

```
{'A': [], 'B': [], 'C': [], 'D': [], 'E': [], 'F': [], 'G': [], 'H': [], 'I': [], 'J': [],
'K': [], 'L': [], 'M': [], 'N': [], 'O': [], 'P': [], 'Q': [], 'R': [], 'S': [], 'T': [],
'U': [], 'V': [], 'W': [],
 'X': ['A', 'B', 'D', 'E', 'G', 'H', 'N', 'P', 'S', 'T'],
'Y': ['D', 'I', 'O', 'A', 'U', 'K', 'R', 'V', 'W', 'Y', 'E'],
'Z': []}
```

```
allPatterns = {'0.0.1': ['EEL'],
 '0.0.1.2': ['EELS', 'OOZE'],
 '0.0.1.2.0': ['EERIE'],
 '0.0.1.2.3': ['AARON', 'LLOYD', 'OOZED'],
 '0.0.1.2.3.4': ['AARHUS', 'EERILY'],
 '0.0.1.2.3.4.5.5': ['EELGRASS'],
 '0.1.0': ['ADA',
           'BIB',
           'BOB',
           'DAD',
           'DID',
           'DUD',
           'EKE',
           'ERE',
           'EVE',
           'EWE',
           'EYE',
           'GAG',
           'GIG',
           'HUH',
           'NAN',
           'NON',
           'NUN',
           'PEP',
           'PIP',
           'POP',
           'PUP',
           'SUS',
           'TIT'],
 '0.1.0.0': ['DODD', 'LULL'],
 '0.1.0.0.1': ['MAMMA'],
 '0.1.0.0.1.2': ['MAMMAL', 'MAMMAS', 'PAPPAS', 'PEPPER'],
 '0.1.0.0.1.2.1.3': ['PEPPERED'],
 '0.1.0.0.1.2.3': ['BABBAGE',
                   'BIBBING',
                   'LILLIAN',
                   'MAMMALS',
```

암호문 단어 'XYX'
패턴 0.1.0

# 암호문 단어의 패턴 분석

```
import random, re, copy   # re: regular expression
import wordPatterns, makeWordPatterns  # 교재의 라이브러리
import SubstCipher_lib

nonLettersOrSpacePattern = re.compile('[^A-Z\s]')
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

#--- 분석용 Mapping 리스트 초기화
def getBlankMapping():
    return { 'A':[], 'B':[], 'C':[], 'D':[], 'E':[], 'F':[],
             'G':[], 'H':[], 'I':[], 'J':[], 'K':[], 'L':[], 'M':[],
             'N':[], 'O':[], 'P':[], 'Q':[], 'R':[], 'S':[],
             'T':[], 'U':[], 'V':[], 'W':[], 'X':[], 'Y':[], 'Z':[]}

#--- 암호문의 단어 cipher_word의 후보단어 candidate의 정보를 Mapping에 반영
#--- 예: Mapping (빈 리스트), cipher_word='XTTPY', candidate='APPLE'
#        Mapping = { 'X': ['A'], 'T': ['P'], 'P': ['L'], 'Y':['E'], ...}
def addLettersToMapping(Mapping, cipher_word, candidate):
    for i in range(len(cipher_word)):
        if candidate[i] not in Mapping[cipher_word[i]]:
            Mapping[cipher_word[i]].append(candidate[i])
```

```
word_map = getBlankMapping()
word = 'XYXXYZ'
addLettersToMapping(word_map, word, 'MAMMAL')
print(word_map)
```

```
word_map = {'A': [], 'B': [], 'C': [], 'D': [], 'E': [], 'F': [], 'G': [], 'H': [], 'I': [],
'J': [], 'K': [], 'L': [], 'M': [], 'N': [], 'O': [], 'P': [], 'Q': [], 'R': [], 'S': [], 'T': [],
'U': [], 'V': [], 'W': [], 'X': ['M'], 'Y': ['A'], 'Z': ['L']}
```

```
word_candidate = ['MAMMAL', 'MAMMAS', 'PAPPAS', 'PEPPER']
for candidate in word_candidate:
    addLettersToMapping(word_map, word, candidate)
print(word_map)
```

```
word_map = {'A': [], 'B': [], 'C': [], 'D': [], 'E': [], 'F': [], 'G': [], 'H': [], 'I': [], 'J': [],
'K': [], 'L': [], 'M': [], 'N': [], 'O': [], 'P': [], 'Q': [], 'R': [], 'S': [], 'T': [], 'U': [], 'V': [],
'W': [], 'X': ['M', 'P'], 'Y': ['A', 'E'], 'Z': ['L', 'S', 'R']}
```

# 후보 알파벳의 교집합

```
#--- 두 개의 Mapping의 교집합을 계산
#    주의: 어느 한쪽의 Mapping에만 candidate가 있는 경우는 그것을 사용함
#          공통 후보가 있으면 교집합으로 후보를 업데이트
def intersectMapping(mapA, mapB):
    intersectedMapping = getBlankMapping()
    for letter in LETTERS:
        if mapA[letter] == []:
            intersectedMapping[letter] = copy.deepcopy(mapB[letter])
        elif mapB[letter] == []:
            intersectedMapping[letter] = copy.deepcopy(mapA[letter])
        else:
            for mapped_letter in mapA[letter]:
                if mapped_letter in mapB[letter]:
                    intersectedMapping[letter].append(mapped_letter)

    return intersectedMapping
```

후보 알파벳이 다수인 문자는 교집합으로

```
map_A = { 'A': ['C', 'D', 'E'], 'B': [],        'C': ['B', 'A'], 'D': ['E'], 'E': ['C','D'], … }
map_B = { 'A': ['D', 'E'],      'B': ['B', 'E'], 'C': ['A'],      'D': [],    'E': ['C'],     … }

map_C = intersectMapping(map_A, map_B)
print(map_C)
```

후보 알파벳이 한쪽에만 있으면 모두 후보로 유지

map_C =
{ **'A': ['D', 'E'], 'B': ['B', 'E'], 'C': ['A'], 'D': ['E'], 'E': ['C'],** 'F': [], 'G': [], 'H': [], 'I': [], 'J': [], 'K': [], 'L': [], 'M': [], 'N': [], 'O': [], 'P': [], 'Q': [], 'R': [], 'S': [], 'T': [], 'U': [], 'V': [], 'W': [], 'X': [], 'Y': [], 'Z': []}

# 후보 알파벳의 정리

```
#--- 후보가 하나만 남아 확정된 문자들을 정리
def removeSolvedfromMapping(Mapping):
    loop_flag = True
    while loop_flag:
        loop_flag = False
        solved_letters = [] # 중간계산에 필요한 리스트 (출력에 반영안됨)
        for cipher_letter in LETTERS:
            #- 한개의 후보만 남은 경우 => 확정
            if len(Mapping[cipher_letter]) == 1:
                solved_letters.append(Mapping[cipher_letter][0])

        # 남은 문자의 후보에서 이미 확정된 것을 제외
        for cipher_letter in LETTERS:
            for s in solved_letters:  # 이미 확정된 문자에 대하여
                # 다른 문자에서 확정된 문자가 후보에 있으면 제외
                if len(Mapping[cipher_letter]) != 1 and s in Mapping[cipher_letter]:
                    Mapping[cipher_letter].remove(s)
                    if len(Mapping[cipher_letter]) == 1: # 이제 후보가 한개만 남으면
                        loop_flag = True  # while 반복에서 후보 확정으로 이어짐

    # 확정된 문자들은 후보가 한개만 남아 있게 된다.
    return Mapping
```

대응되는 후보 알파벳이 한개이면 확정

확정된 알파벳은 다른 알파벳의 후보에서 삭제

확정된 알파벳 삭제후 후보가 한 개만 남으면

# 암호문으로부터 후보키 맵을 만든다

```
#--- 사전에 저장된 패턴을 이용하여 후보키를 만든
def hackSubst(message):
    intersectedMap = getBlankMapping()
    #- message를 대문자로 바꾸고, 영문자만 남기고, 단어로 나누어 리스트 만들기
    cipherword_list = nonLettersOrSpacePattern.sub('', message.upper()).split()

    # message의 각 단어에 대하여
    for cipher_word in cipherword_list:
        candidate_map = getBlankMapping()
        wordPattern = makeWordPatterns.getWordPattern(cipher_word) # 단어의 패턴을 만들기
        if wordPattern not in wordPatterns.allPatterns: # 영어단어 패턴에 속하지 않으면
            continue # 사전에서 발견되지 않는 단어면 통과

        # 사전에 있는 단어이면 해당 패턴으로 후보를 업데이트 함
        for candidate in wordPatterns.allPatterns[wordPattern]:
            addLettersToMapping(candidate_map, cipher_word, candidate)
        intersectedMap = intersectMapping(intersectedMap, candidate_map)


    return removeSolvedfromMapping(intersectedMap)
```

암호문을 단어 단위로 나누어 리스트로 만든다.
(특수문자, 숫자는 삭제한다)

각 단어에 대하여 패턴을 분석하여
후보 알파벳 정보를 만든다.

기존 후보 알파벳 정보와 합친다.

# 후보키 맵을 이용한 복호화

```
def decryptWithMapping(ciphertext, letterMap):

    key = ['x']*len(LETTERS)

    for cipher_char in LETTERS:

        if len(letterMap[cipher_char]) == 1:
            key_index = LETTERS.find(letterMap[cipher_char][0])
            key[key_index] = cipher_char
        else:
            ciphertext = ciphertext.replace(cipher_char.lower(), '_')
            ciphertext = ciphertext.replace(cipher_char.upper(), '_')

    key = ''.join(key)

    return SubstCipher_lib.subst_decrypt(key, ciphertext)
```

key = 26개의 'x'를 갖는 리스트
(타이핑을 덜하기 위한 것일 뿐)

후보키 맵의 각 문자에 대하여

문자에 대응되는 후보키가 한 개이면 확정

문자에 대응되는 후보키가 여러 개이면
문자를 '_'로 바꾸어 복호화에 반영되지
않도록 한다.

확정된 문자가 아닌 것은
'x'를 그대로 둔다.

# 복호화 예제

- 복호화 전략
  - 암호문 분석으로 얻는 후보키 맵으로부터 암호키를 만든다.
  - 암호키로 암호문을 복호화한다.

```
cipher_file_name = 'c0401-1.txt'
decrypt_file_name = 'dict_attack0401-1.txt'
print('Ciphertext File Name = ', cipher_file_name)
print('Decrypted File Name = ', decrypt_file_name)

ciphertext = SubstCipher_lib.ReadFile(cipher_file_name)

letterMapping = hackSubst(ciphertext)
print('Mapping: ', letterMapping)

decrypt_data = decryptWithMapping(ciphertext, letterMapping)

SubstCipher_lib.WriteFile(decrypt_file_name, decrypt_data)
```

알파벳 후보 맵

Mapping: {'A': ['D'], 'B': ['E'], 'C': ['F'], 'D': ['G'], 'E': ['H'], 'F': ['L'], 'G': ['M'], 'H': ['N'], 'I': ['I'], 'J': [], 'K': ['K'], 'L': ['O'], 'M': ['P'], 'N': ['T'], 'O': ['U'], 'P': ['V'], 'Q': ['Q'], 'R': ['R'], 'S': ['S'], 'T': ['W'], 'U': ['X'], 'V': ['A'], 'W': ['B'], 'X': ['C'], 'Y': ['Y'], 'Z': ['Z']}

복호화한 결과
(해결하지 못한 '_'가 보임)

dict_attack_0401.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말

```
Presiden_ Donald _rump finally leveled _i_h America abou_ _h
Even _i_h blanke_ na_ion_ide adop_ion of s_ringen_ mi_iga_io
I_ is in _he na_ure of _he presidency, _ha_ _he commander-in
George _. Bush had _o narra_e _he horror of _he 9/11 a__acks
Bu_ no presiden_ for many decades has had _o level _i_h his
Fac_-checking _rump&#39;s mara_hon coronavirus briefing
Fac_-checking _rump's mara_hon coronavirus briefing
I_ _as no_ _he firs_ _ime _ha_ adminis_ra_ion exper_s modele
_he s_ark spec_acle of a presiden_, especially one _ho spen_
"I _an
_rump's
```

plaintext_0401.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말

```
President Donald Trump finally leveled with America about th
Even with blanket nationwide adoption of stringent mitigatio
It is in the nature of the presidency, that the commander-in
George W. Bush had to narrate the horror of the 9/11 attacks
But no president for many decades has had to level with his
Fact-checking Trump&#39;s marathon coronavirus briefing
Fact-checking Trump's marathon coronavirus briefing
It was not the first time that administration experts modele
The stark spectacle of a president, especially one who spent
"I want every American to be prepared for the hard days that
Trump's briefing mostly lacked the elements of self-congratu
```

평문