Kryptologia: Laboratorium 2

2 listopada 2022

1. Wstęp

W poniższym kodzie podjęliśmy się rozszyfrowania dowolnie wybranego tekstu w języku angielskim (1.txt) oraz polskim (2.txt). Tekst został zakodowany przez naszych kolegów według wybranego przez nich klucza.

[1]:
```
import os
import io
from random import shuffle
```

Aby rozszyfrować tekst zdefiniowaliśmy następujące funkcje lub klasy:

(1) funkcja formatująca tekst (opisana w poprzednim sprawozdaniu),

(2) funkcja obliczająca częstotliwość danej litery w podanym tekście,

(3) funkcja znajdująca pasującą częstotliwość danej litery w kryptażu do częstotliwości litery występującej w danym języku,

(4) klasa Kodowania Cezara (opisana w poprzednim sprawozdaniu).

[2]:
```
def formatText(remove_nums="T", fname="text.txt"):
    remove_sym="T"
    table_format="N"
    to_upper="N"
    print_file="N"
    replace_symbols="\"...\""
    replacements="\"
    f_in=fname\"
    f_out="text_out"␣
 ↪args=[remove_nums,remove_sym,table_format,to_upper,print_file
,replace_symbols,replacements]
    cmd="python ./readData.py \"%s\" \"%s\" "%(f_in,f_out)+" ".join(args)
    assert os.system(cmd)==0
    text=None
    with io.open(f_out,mode="r",encoding="utf-8") as f:
        text=f.read()
    return text
```

```
def compute_symbol_frequencies(string):
    syms=dict((x,0) for x in [chr(x) for x in range(ord('a'),ord('z')+1)])
    for s in string:
        if not s in syms:
            syms[s]=1
        else:
            syms[s]+=1
    output=dict((x,syms[x]/len(string)) for x in syms)
    return output

def match(freq,lang,delta=0.002):
    eps=0
    matched={}
    found=0
    while eps<1:
        if (found==len(freq)):
            break
        for symbol in freq:
            if symbol in matched:
                continue
            for s in lang:
                if abs(freq[symbol]-lang[s])<=eps:
                    found+=1
                    if symbol in matched:
                        matched[symbol].append(s)
                    else:
                        matched[symbol]=[s]
        eps+=delta
    return matched

class CaesarEncoding:
    def __init__(self,n,alphabet):
        self.n=n
        self.alphabet=[x for x in alphabet]
        self.alp=dict((alphabet[i],i) for i in range(len(alphabet)))

    def encode(self,symbol):
        return self.alphabet[(self.alp[symbol]+self.n)%len(self.alphabet)]

    def decode(self,symbol):
        return self.alphabet[(self.alp[symbol]-self.n)%len(self.alphabet)]
```

```
    def decodeString(self,string):
        return "".join(map(self.decode,string))

    def encodeString(self,string):
        return "".join(map(self.encode,string))
```

Następnie na podstawie wcześniejszej analizy częstotliwości wyszczególniśmy częstotliwości występowania wszystkich liter w języku angielskim. Nasze wyniki można porównać z ogólnodostępnymi danymi.

[3]:
```
frequencyEnglish = {
     't': 0.09081222484527532,
     'o': 0.07726025649205641,
     's': 0.06220251387736872,
     'h': 0.06718560581892427,
     'e': 0.12309704587507178,
     'r': 0.05673451158042493,
     'l': 0.03888215402284183,
     'c': 0.02468576532890959,
     'k': 0.008281758438078223,
     'm': 0.02813756141134435,
     'i': 0.06909334524341224,
     'a': 0.0821540228418299,
     'w': 0.026261723983921393,
     'y': 0.02170611880303707,
     'n': 0.06671345626236203,
     'v': 0.01023416065845722,
     'd': 0.042423275697058636,
     'u': 0.030440885599438524,
     'p': 0.015734064952466025,
     'f': 0.021208447648822817,
     'x': 0.0012696994831876474,
     'b': 0.014387800676322338,
     'g': 0.01817775792764627,
     'j': 0.00135925074969693,
     'q': 0.0011101894978625662,
     'z': 0.0004466279589102276
}
```

2. Dekryptaż

Przechodzimy do dekryptażu pliku 1.txt.

W pierwszym kroku formatujemy tekst oraz dokonujemy analizy częstotliwości. Następnie za pomocą funkcji `match` dopasowujemy litery z zakodowanej wiadomości do liter z angielskiego alfabetu.

[4]:
```
text1=formatText(fname="1.txt")
freq1=compute_symbol_frequencies(text1)
matched1=match(freq1,frequencyEnglish)
matched1Exact = dict((x,matched1[x][0]) for x in matched1 if␣
 ↪len(matched1[x])==1)
matched1Exact
```

[4]:
```
{'d': 'o',
 'h': 'e',
 'j': 'c',
 'k': 'i',
 'l': 'n',
 'q': 'r',
 'r': 'o',
 'y': 'b',
 'x': 'u',
 'o': 'r',
 'u': 'd',
 'w': 't'}
```

Znaleźliśmy 12 par liter, które mogą być ze sobą zamienione.

[5]:
```
text1partDec="".join(map(lambda s: matched1Exact[s] if s in matched1Exact␣
 ↪else "<"+s+">", text1))
text1partDec[:100]
```

[5]:
```
'ore<p>odrnrc<z>iercdecod<v>o<p><v>o<z>o<n>e<i>do<p>tdou<e>re<g><g>deo<p><v>ie<i
>our<g>in<p><v>er<i>t'
```

Ponieważ nie wszystkie litery znalazły swój dokładny odpowiednik jeśli chodzi o częstotliwość, odkodowana wiadomość nie jest jeszcze zrozumiała.
Zatem z otrzymanych wyników analizy częstotliwości srawdziliśmy jakie klucze mogły zostać użyte przez naszych kolegów. Spośród nich wybraliśmy ten najbardziej prawdopobny.

[6]:
```
keySuggestions = [abs(ord(x)-ord(matched1Exact[x])) for x in␣
 ↪matched1Exact]
mostProbableKey = max(set(keySuggestions), key=keySuggestions.count)
mostProbableKey
```

[6]:
```
3
```

[7]:
```
alphabet=[chr(x) for x in range(ord('a'),ord('z')+1)]
c = CaesarEncoding(mostProbableKey,alphabet)
c.decodeString(text1)[:100]
```

[7]:
```
'onemorningwhengregorsamsawokefromtroubleddreamshefoundhimselftransformedinhisbe
dintoahorriblеverminh'
```

Najczęściej występujący klucz okazał się słuszny w naszym przypadku.
Dokładnie takie same kroki podjęliśmy w przypadku dekryptażu tekstu w języku polskim.
Zgodnie z poleceniem spróbowaliśmy także odszyfrować tekst za pomocą metody siłowej.

[12]:
```
def forceDecode(text,alphabet):
    return [CaesarEncoding(i,alphabet).decodeString(text[:25]) for i in␣
 ↪range(1,len(alphabet))]
```

Aby tego dokonać wyświetliliśmy tekst odszyfrowany za pomocą każdego z 26-ciu kluczy.

[13]:
```
print(forceDecode(text1,alphabet))
##sample number 3 looks good
##=>
key = 3
```

```
['qpgoqtpkpiyjgpitgiqtucouc', 'pofnpsojohxifohsfhpstbntb',
'onemorningwhengregorsamsa', 'nmdlnqmhmfvgdmfqdfnqrzlrz',
'mlckmplgleufclepcempqykqy', 'lkbjlokfkdtebkdobdlopxjpx',
'kjaiknjejcsdajcnacknowiow', 'jizhjmidibrczibmzbjmnvhnv',
'ihygilhchaqbyhalyailmugmu', 'hgxfhkgbgzpaxgzkxzhkltflt',
'gfwegjfafyozwfyjwygjkseks', 'fevdfiezexnyvexivxfijrdjr',
'educehdydwmxudwhuwehiqciq', 'dctbdgcxcvlwtcvgtvdghpbhp',
'cbsacfbwbukvsbufsucfgoago', 'barzbeavatjuratertbefnzfn',
'azqyadzuzsitqzsdqsademyem', 'zypxzcytyrhspyrcprzcdlxdl',
'yxowybxsxqgroxqboqybckwck', 'xwnvxawrwpfqnwpanpxabjvbj',
'wvmuwzvqvoepmvozmowzaiuai', 'vultvyupundolunylnvyzhtzh',
'utksuxtotmcnktmxkmuxygsyg', 'tsjrtwsnslbmjslwjltwxfrxf',
'sriqsvrmrkalirkviksvweqwe']
```

3. Wnioski

Metoda siłowa jest efektywna jedynie wtedy, gdy znamy język zaszyfrowanej wiadomości. W przeciwnym przypadku nie byliśmy w stanie rozpoznać klucza tj. wiadomości, która "wygląda sensownie". Używając natomiast analizy częstotliwości jesteśmy w stanie rozszyfrować wiadomość bez znajomości danego języka, oczywiście pod warunkiem, że mamy możliwość przetłumaczenia rozszyfrowanej wiadomości.