Keszitette: Hegyesi Akos

Neptun kod: HSFPOJ

1. Feladat

Az alábbi függvényt használjuk egy eltolásos titkosítási algoritmusban (shift-ciper szerű). A bemeneti szöveg és a titkos szöveg is csak az angol ábécé kisbetűit tartalmazza. Legyen f a következőképpen definiálva:

Ha k páros, akkor f(k) = ((k+2)^3) % 107.

Ha k páratlan, akkor f(k) = ((k+3)^2) % 111.

Ezzel úgy titkosítunk, hogy a bemeneti szöveg k-adik karakterét (k = 0, 1,...) pontosan f(k)-val toljuk el. (A szokásos módon, ha az eltolás során 'z' után újra 'a' jön.)

Valósítsuk meg ezt a titkosítást, és titkosítsuk az alábbi sztringeket:

"cryptography"

"hungary"

"letsgobacktoschoolsoon"

Az alábbi szövegtöredékről azt tudjuk, hogy egy olyan bemeneti szöveg titkosítása során jött létre, amiben szerepel egy európai főváros (csupa kisbetűs) neve. Mi lehet ez a város, és a bemeneti szöveg melyik pozícióján található az első karaktere?

"....nmhhlzjvtslmnbfzdwgk..."

In [1]:
```
# karakterek kodolasa
def indexOfLetter(c):
    return ord(c) - 97
def letterWithIndex(i):
    return chr(i+97)
def shiftChar(c, shift):
    i = indexOfLetter(c)
    new = (i + shift) % 26
    return letterWithIndex(new)
```

In [2]:
```
def shiftChiperEnc(string):
    chiper = ""
    for i in range (len(string)):
        if i % 2 == 0:
            chiper += shiftChar(string[i], (i + ((i+2)^3) % 107))
        else :
            chiper += shiftChar(string[i], (i + ((i+3)^2) % 111))
    return chiper
```

In [3]:
```
m1 = "cryptography"
m2 = "hungary"
m3 = "letsgobacktoschoolsoon"
m4 = "sssssssssssssssssssssssssssss"

c1 = shiftChiperEnc(m1)
c2 = shiftChiperEnc(m2)
c3 = shiftChiperEnc(m3)
c4 = shiftChiperEnc(m4)

print(c1)
print(c2)
print(c3)
print(c4)
```

```
dyhwcdxgrmgv
ibwnjgp
mlczpdspthslrhotvyhbdi
tzbzbhjhjprprxzxzfhfhnpnpvxvxdfd
```

In [4]:
```
def shiftChiperDec(chiper, shift = 0):
    message = ""
    for i in range (len(chiper)):
        if i % 2 == 0:
            j = i + shift
            message += shiftChar(chiper[i], ((j) - ((i+2)^3) % 107) - 2*j)
        else :
            j = i + shift
            message += shiftChar(chiper[i], ((j) - ((i+3)^2) % 111) - 2*j)
    return message
```

In [5]:
```
m_c1 = shiftChiperDec(c1)
m_c2 = shiftChiperDec(c2)
m_c3 = shiftChiperDec(c3)
m_c4 = shiftChiperDec(c4, 0)

print(m_c1)
print(m_c2)
print(m_c3)
print(m_c4)
```

```
cryptography
hungary
letsgobacktoschoolsoon
sssssssssssssssssssssssssssss
```

In [6]:
```
# a keresett varosnev a 26 hosszu periodus miatt az alabbi esetek kozott f
chiperText = "nmhhlzjvtslmnbfzdwgk"
for k in range (26):
    print (shiftChiperDec(chiperText, k))
```

```
mfyacksgcvmpowyuwjrx
lexzbjrfbulonvxtviqw
kdwyaiqeatknmuwsuhpv
jcvxzhpdzsjmltvrtgou
ibuwygocyrilksuqsfnt
hatvxfnbxqhkjrtprems
gzsuwemawpgjiqsoqdlr
fyrtvdlzvofihprnpckq
exqsuckyunehgoqmobjp
dwprtbjxtmdgfnplnaio
cvoqsaiwslcfemokmzhn
bunprzhvrkbedlnjlygm
atmoqyguqjadckmikxfl
zslnpxftpizcbjlhjwek
yrkmowesohybaikgivdj
xqjlnvdrngxazhjfhuci
wpikmucqmfwzygiegtbh
vohjltbplevyxfhdfsag
ungiksaokduxwegcerzf
tmfhjrznjctwvdfbdqye
slegiqymibsvuceacpxd
rkdfhpxlharutbdzbowc
qjcegowkgzqtsacyanvb
pibdfnvjfypsrzbxzmua
ohacemuiexorqyawyltz
ngzbdlthdwnqpxzvxksy
```

2.Feladat

Egy bemeneti szövegről a következőket tudjuk:

Az angol ábécé nagybetűiből áll (A-Z)

'H' után mindig 'U' jön.

Van valami olyan 5-betűs sztring, ami csupa egyforma betűből áll, és több helyen is előfordul a szövegben.

Egy Vigenère-szerű módszerrel van titkosítva, ahol az eltolási mennyiségek (a kulcs) egy L hosszú listában vannak megadva. (Pl. ha a kulcs [1, 2, 3], akkor "AAAAAAA" képe "BCDBCDB", vagy ha [1, 2, 3, 4, 5] a kulcs, akkor „BBBBBKRR" képe „CDEFGLTU".)

Elfogtunk egy titkos szöveget, melyről annyit tudunk, hogy vagy 5 vagy 7 hosszúságú a kulcs. Mi a kulcs, ha a titkos szöveg az alábbi (sortörések törlendőek)?

UEAPTEGKMVQRKMVEGVBYCGKMVEGLOVWLUJVEGKMFQKKM

GKMVCYJOONICWPLEMRVEGKMAEGKMVEGKMVUSSBWEGKM

MVEGKWVEGKMOTTQVITAGMVEGKMLGXKRSWZKMVEGKMVE

EGKITOZXISMLVMVEGKUWAGKMVELXRPJHQJHBQDOXUGKRG

KMVEGKMVEGVMVEGKQZBGKMVEHNVEYWTUHMJYNSNQKM

In [7]:
```
# karakterek kodolasa -> fuggvenyek ujradefinialasa
def indexOfLetter(c):
    return ord(c) - 65
def letterWithIndex(i):
    return chr(i+65)
def shiftChar(c, shift):
    i = indexOfLetter(c)
    new = (i + shift) % 26
    return letterWithIndex(new)
```

In [8]:
```
import math

# Ismetlodesek kiszurese
def repetitionsInString(s, length = 3):
    n = len(s)
    ret = [(i,j) for i in range(0, n-length) for j in range(i+1, n-length+1
    return ret

# Kulcshosszak keresese
def guessedKeyLength(ciphertext, length = 3):
    differences = [j-i for (i,j) in repetitionsInString(ciphertext, length
    return gcd(differences)

# Sage nelkul sajat fuggveny a kulcshosszak kereseshez
def keyLengths(ciphertext, length = 3):
    differences = [j-i for (i,j) in repetitionsInString(ciphertext, length
    return differences

# Kulcshosszak LNKO-ja
def gcdOfKeys(array):
    if len(array) > 0:
        b=array[0]
        for i in range (0, len(array)):
            s=math.gcd(b,array[i])
            b=s
        return b
```

In [9]:
```
# Vigenere titkosítás -> orai kod
def vigenere(s, shiftList):
    ret = ""
    # Current position in the shiftList
    pos = 0
    n = len(shiftList)
    for c in s:
        new = shiftChar(c, shiftList[pos])
        ret += new
        # Add one, restart if list is over
        pos = (pos + 1) % n
    return ret
```

In [10]:
```
# chiper text
chiper = "UEAPTEGKMVQRKMVEGVBYCGKMVEGLOVWLUJVEGKMFQKKMVEGTDNUGKMVEGKMVEMWMV
```

In [11]:
```
# 5, illetve 7 hosszu ismetlodesek keresese -> a gyakori ismetlodest fogom
rep_5 = repetitionsInString(chiper, 5)
rep_7 = repetitionsInString(chiper, 7)
```

In [12]:
```
# Kulcshossz megallapitasa

array_of_differences = keyLengths(chiper, 5)
gcd_of_keys = gcdOfKeys(array_of_differences)

print("length of the key: ", gcd_of_keys)
```

```
length of the key:  5
```

In [13]:
```
# Tudjuk, hogy van egy 5 hosszu listam, amely tartalmaz [x0, x1, x2, x3, x
# Tudjuk tovabba,
#   hogy a szovegben H-t mindig U koveti,
#   es van olyan 5 betus string, ami ugyan olyan betuket tartalmaz

print(chiper[5:10])
print(chiper[55:60])
```

```
EGKMV
EGKMV
```

In [14]:
```
# Az ismetlodo szovegreszletekben, ha feltetelezzuk, hogy ezek egy 5 ugyan
# abban az esetben a kulcsok kulonbseget mar meg tudjuk allapitani

# pl.: Tfh, hogy az az ismetlodo szovegreszlet az EGKMV, es ez a DDDDD szo
# igy kezi szamitassal az alabbi eltolasokat allapitottuk meg DDDDD -> L =
# ezt felhasznalva az L=[0, -2, -6, -8, -17] tombbel egy brute force megol
L = [-1, -3, -7, -9, -18]
L = [0, 2, 6, 8, 17]
indexes = []

# Brute force modszerrel vegig probalom mind a 26 lehetseges eltolast, es

for i in range (26):
    L = [0-i, -2-i, -6-i, -8-i, -17-i]
    text = vigenere(chiper, L)
    for j in range(len(text)):
        if text[j] == 'H' and (j + 1 < len(text)) and  text[j+1]  == 'U':
            indexes.append(i)
#             print(text)
#    print()

print('Potencialis eltolasok: ', indexes)
```

```
Potencialis eltolasok:  [7, 7, 8, 11, 11, 11, 12, 13, 13, 15, 17, 20, 23]
```

In [15]:
```
# Brute force megoldas utan csak nehany lehetseges eset maradt, amit meg k
# A vizsgalat uta arra a megallapitasra jutottam, hogy a TTTTT szoveg volt
# Az eredeti szoveg pedig a kovetkezo:

L = L = [0 - 11, -2-11, -6-11, -8-11, -17-11]
original_message = vigenere(chiper, L)

print('Original message:\n')
print(original_message)
```

```
Original message:

JRJWRTTTTTFETTTTTEIWRTTTTTTUVTLYDQTTTTTDFXTTTTTCKLJTTTTTTTTTTTZFTTTTTHUTTTTT
RLSVMCVLDNARVYTTTTTTYTTTTTTTTTTTJFBIUTTTTTDYIVBTTTTTVXTTTTTGMDAQBVDQJWTTTTTDT
TTTTMIGZCGINPTTTTTTJVKTYQLMTTTTTTTTTTYRSCLNETTTTTTTTTTLLOGULXLKTTTTTPRDMGPQ
BYETTTTTBUPTTTTTYGYNYUZQFQDMVVJTTYEBTTTTTTTTTTZHUTTTTTVTTTTTERTTTTTTTTTETT
TTTXXQTTTTTUWCCNJCBFBWHUQCDTTTTTTVBDTTTTTVT
```

3.Feladat

Emlékeztető: egy n elemű halmaznak binomial(n, k) darab k elemű része van (n alatt a k)

Építsünk megkülönböztetőt a következő véletlen generátorhoz. (Megkülönböztető bemenete két véletlennek szánt sztring, mely csak 0 és 1 karaktert tartalmaz, egyikül igazi véletlen, másikat az alábbi módszerrel állítottuk elő. A kimenet az, hogy megtippeljük, melyik melyik.

A seed egy n pozitív egész szám. A kimenet hossza L bit.

Minden M-re a range(n, n+L) tartományból megszámoljuk, hány olyan részhalmaza van egy M-elemű halmaznak, melynek elemszáma osztható 4-gyel, a képlet

H = binomial(M, 0) + binomial(M, 4)+ ...

A kimenetre írunk egy 0 bitet, ha if H < 2^(n-2), különben egy 1-est.

In [16]:
```
import random
import scipy.special

# veletlen sorozat
def makeRandomArray (n):
    array = ""
    for i in range(n):
        rand = random.randint(0,1)
        array += str(rand)
    return array

# megszamoljuk, hogy hany olyan reszhalmaza van egy M-elemu halmaznak, mely
def createArrayOfBinoms(n, L, array_of_M):
    H = []
    for i in range(n, n+L):
        k = 0
        number = 0
        while k * 4 <= i:
            tmp = scipy.special.binom(i, k*4)
            number += tmp
            k = k + 1
        H.append(number)
        array_of_M.append(i)
    return H

# random szamok generalasa a reszhalmazok alapjan
def generatePseudoRandomArray(H, n, arrays):
    array = ""
    for i in range(len(H)):
        if H[i] < (2**(arrays[i]-2)):
            array += "0"
        else:
            array += "1"
    return array
```

In [17]:
```
L = 100
seed = random.randint(0,L)
array_of_M = []

rand_array_1 = makeRandomArray(L)

H = createArrayOfBinoms(seed, L, array_of_M)
print(seed)
rand_array_2 = generatePseudoRandomArray(H, seed, array_of_M)
```

```
61
```

In [18]:
```
# random sorozatok kiirasa
# lathato, hogy az altalunk generalt random sorozat sokkal 'blokk' szerubb
# lathato tovabba, hogy nagy valoszinuseg szerint az altalunk generalt sor

print(rand_array_1)
print(rand_array_2)
print()
print('Random sorozatban levo 0-k szama: ', rand_array_1.count('0'))
print('Random sorozatban levo 1-k szama: ', rand_array_1.count('1'))
print()
print('Altalunk generalt pszeudo-random sorozatban levo 0-k szama: ', rand
print('Altalunk generalt pszeudo-random sorozatban levo 1-k szama: ', rand


counter = 0
for i in range (1000):
    L = 100
    seed = random.randint(0,L)
    array_of_M = []
    rand_array_1 = makeRandomArray(L)
    H = createArrayOfBinoms(seed, L, array_of_M)
    rand_array_2 = generatePseudoRandomArray(H, seed, array_of_M)

    if rand_array_2.count('0') < rand_array_2.count('1') and (rand_array_1
        counter += 1

print('Counter: ', counter)
```

```
0100010000110111001100100010001111100010001000010000100101001001100111101001
1100000000001100100000111
0111100000111000011110000011100001111100001110110111011110011101111111010111
0010001000111100010010011

Random sorozatban levo 0-k szama:  62
Random sorozatban levo 1-k szama:  38

Altalunk generalt pszeudo-random sorozatban levo 0-k szama:  46
Altalunk generalt pszeudo-random sorozatban levo 1-k szama:  54
Counter:  559
```

In [19]:
```
# 1 es 100 kozott a 4 elemu reszhalmazok szama:
for i in range (100):
    k = 0
    number = 0
    while k*4 < 100:
        number += scipy.special.binom(i, k*4)
        k += 1
#    print(number)
```

In [20]:
```
# Az altalam, valoszinuleg felreertelmezes miatt rosszul megirt pszeudoran
# hogy a kapott reszhalmazok szama egy olyan nagy szam, hogy emiatt 1-es b

# Igy konnyen belathatjuk, hogy abban az esetben, ha tobb 1-es van az egyi
# valamint onnan is felismerheto az altalunk keszitett sorozat, hogy a sor
```

4.Feladat

Egy hash-függvényt valósítunk meg, mely csupa angol nagybetűből álló szövegekhez rendel számszerű hash értéket.

A módszer leírásához használjunk minden betűhöz egy számértéket a következő p függvény szerint: p('A') = 0, p('B') = 1, ..., p('Z') = 25.

Ezután a bemeneti szövegeket átalakítjuk polinommá a következő szabály szerint:

Összeadjuk az s szöveg összes k pozíciójára a p(s[k]) * x^k tagokat. A kapott polinom legyen f(s)

Például a hárombetűs „XCF" szöveg polinomja 23 + 2x + 5x^2.

Ezután az f(x) polinomot elosztjuk maradékosan a g(x) = x^12 + 5x^7 + 2 polinommal. A maradék legyen r(x). Végül a számszerű hash-érték legyen: r(73).

A feladat ennek a hash-függvénynek a megvalósítása, illetve a következő „törés" implementálása:

Bemenet két szöveg, S és T. A cél az, hogy írjunk néhány karaktert T végére úgy, hogy a az így kapott T' szöveg hashe ugyanaz legyen, mint S-é.

In [21]:
```
import numpy as np
import sympy
from sympy import *
# seged fuggveny a karakterbol szamra kepzeshez
def letterToNumber (c):
    idx = -1
    letters = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O'
    for i in range (len(letters)):
        if letters[i] == c:
            idx = i
    return idx

# polinom keszito fuggveny
def polinomyalMaker (string):
    array = []
    for i in range (len(string)):
        array.append(letterToNumber(string[i]))
    array.reverse()
    f = np.poly1d(array)
    return f

# g(x) polinom elkeszitese ket fele modon is
g = [12,0,0,0,0,5,0,0,0,0,0,0,2]

def createGPolinomialWithPoly1d (g):
    return np.poly1d(g)

def createGPolinomialWithArray (g):
    return np.array(g)

# polinom osztas elvegzese
def createPolinomyalDivision (f,g):
    division = np.polydiv(f,g)
    return division

# r(x) polinom generalasa
def createRemainderPolynomial (f,g):
    division = np.polydiv(f,g)
    return division[1]

# polinom szamitasa egy adott szammal -> HASH szamitasa
def evaluatePolynomial (r, x):
    return np.polyval(r, x)
```

In [22]:
```
# peldaban szereplo polinom megvalositasa ellenorzesre
f = polinomyalMaker(['X', 'C', 'F'])
print(f)
```

```
   2
5 x + 2 x + 23
```

In [23]:
```
g = createGPolinomialWithPoly1d(g)
print('g(x) polinom: \n', g)
print()

division = createPolinomyalDivision(f,g)
r = createRemainderPolynomial(f,g)
print('Maradek a polinomialis osztasbol: \n', r)
```

```
g(x) polinom:
     12     7
12 x  + 5 x + 2

Maradek a polinomialis osztasbol:
    2
5 x + 2 x + 23
```

In [24]:
```
# szamszeru hash ertek szamitasa
p = evaluatePolynomial(r, 73)
print(p)
```

```
26814.0
```

In [25]:
```
# MEGOLDAS
# adott S es T string
# a cel, hogy irjunk nehany karaktert ugy T vegere, hogy a kapott T' hash-
S = "KORTEFA"
T = "ALMAFA"

f1 = polinomyalMaker(S)
f2 = polinomyalMaker(T)

g = createGPolinomialWithPoly1d(g)

division_1 = createPolinomyalDivision(f1,g)
division_2 = createPolinomyalDivision(f2,g)

r_1 = createRemainderPolynomial(f1,g)
r_2 = createRemainderPolynomial(f2,g)

p_1 = evaluatePolynomial(r_1, 73)
p_2 = evaluatePolynomial(r_2, 73)

# hash ertekek
print('S hash erteke: ', p_1)
print('T hash erteke: ', p_2)
```

```
S hash erteke:  10486433877.0
T hash erteke:  142055956.0
```

In [ ]:
```

```