Kriptografija i sigurnost mreža

Zadaća 2

Mihael Petrinjak

In [17]:
```
import numpy as np
import pandas as pd
```

Zadatak 1

Uredimo podatke za obradu.

In [164]:
```
sifrat = '''YCGRC AAZHW YMPHQ EYEDG MMBAR SLITG ASROK KGWOX
ULDUH NKDTY QMVTX FZRPE CWYEV OGIJE TAOOG XPEKD
ZVRAB SMXGD DBHNK LDATN KKPHD HXUVS NQKTG WIITA
HJDYI IAWRX AUDLE SXDJ'''

sifrat = sifrat.replace(" ", "").replace("\n", "")
print(sifrat)
```

```
YCGRCAAZHWYMPHQEYEDGMMBARSLITGASROKKGWOXULDUHNKDTYQMVTXFZRPECWYEVOGIJETAOOGXPEKDZVRA
BSMXGDDBHNKLDATNKKPHDHXUVSNQKTGWIITAHJDYIIAWRXAUDLESXDJ
```

Provjerimo pojavljuju li se neki trinomi više puta u šifratu.

In [19]:
```
trinomi = {}
for i in range(len(sifrat) - 2) :
    trinom = sifrat[i:i+3]
    try : trinomi[trinom] += 1
    except KeyError : trinomi[trinom] = 1
```

In [20]:
```
{k:v for k,v in trinomi.items() if v>1}
```

Out[20]:
```
{'HNK': 2}
```

HNK se pojavljuje 2 puta. Tražimo udaljenost pozicija na kojima ppočinju.

In [21]:
```
for i in range(len(sifrat) - 2) :
    if sifrat[i:i+3] == "HNK" : print(i)
```

```
44
92
```

In [22]:
```
92 - 44
```

Out[22]:
```
48
```

Prema Kasiskijevom testu bismo trebali očekivati neki m koji dijeli 48.

Definiramo funkciju koja računa frekvenciju pojedinih slova u danom tekstu.

In [23]:
```
def frekvencije(x) :
    f = {}
    for c in x :
        try : f[c] += 1
        except KeyError : f[c] = 1
    return f
```

In [24]:
```
print(f := frekvencije(sifrat))
```

```
{'Y': 6, 'C': 3, 'G': 8, 'R': 6, 'A': 10, 'Z': 3, 'H': 7, 'W': 5, 'M': 5, 'P': 4,
'Q': 3, 'E': 7, 'D': 11, 'B': 3, 'S': 5, 'L': 4, 'I': 6, 'T': 7, 'O': 5, 'K': 8,
'X': 7, 'U': 4, 'N': 4, 'V': 4, 'F': 1, 'J': 3}
```

Definiramo funkciju koja računa *indeks koincidencije* u danom tekstu.

In [25]:
```
def Ic(x) :
    f = frekvencije(x)
    n = len(x)
    Sum = 0
    for c in f.keys() :
        Sum += f[c]*(f[c] - 1)
    Sum /= n*(n-1)
    return Sum
```

In [26]:
```
Ic(sifrat)
```

Out[26]:
```
0.03857783338546554
```

Vidimo da je $I_c(\text{sifrat}) \approx 0.038$ što je očekivano za nasumičan niz slova.
Istražimo sada ponašanje funkcije na podnizovima šifrata različitih duljina. Trebamo funkciju za podjelu šifrata na opisan način.

In [27]:
```
def podijeli(txt, m) :
    blokovi = []
    for start in range(0, m) :
        komad = ""
        for i in range(start, len(txt), m) :
            komad += txt[i]
        blokovi.append(komad)
    return blokovi
```

Za $m = 10$ na primjer dobivamo podnizove:

In [28]:
```
podijeli(sifrat, 10)
```

Out[28]:
```
['YYMAUQCTZDKNHA',
 'CMMSLMWAVBKQJU',
 'GPBRDVYORHPKDD',
 'RHAOUTEOANHTYL',
 'CQRKHXVGBKDGIE',
 'AESKNFOXSLHWIS',
 'AYLGKZGPMDXIAX',
 'ZEIWDRIEXAUIWD',
 'HDTOTPJKGTVTRJ',
 'WGGXYEEDDNSAX']
```

Iduća funkcija računa I_c na dobivenim blokovima.

In [29]:
```
def indeksi(blokovi) :
    i = []
    for blok in blokovi :
        i.append(Ic(blok))
    return i
```

In [30]:
```
for m in range(1, 12):
    print(m, end=" : ")
    for indeks in indeksi(podijeli(sifrat, m)) :
        print(f"{indeks:.4f}", end=", ")
    print()
```

```
1 : 0.0386,
2 : 0.0435, 0.0388,
3 : 0.0389, 0.0367, 0.0464,
4 : 0.0387, 0.0319, 0.0521, 0.0357,
5 : 0.0265, 0.0344, 0.0503, 0.0635, 0.0456,
6 : 0.0616, 0.0514, 0.0356, 0.0593, 0.0672, 0.0791,
7 : 0.0474, 0.0263, 0.0263, 0.0421, 0.0263, 0.0368, 0.0468,
8 : 0.0458, 0.0261, 0.0784, 0.0368, 0.0368, 0.0368, 0.0515, 0.0515,
9 : 0.0333, 0.0250, 0.0583, 0.0417, 0.0381, 0.0476, 0.0381, 0.0095, 0.0381,
10 : 0.0220, 0.0330, 0.0549, 0.0440, 0.0220, 0.0330, 0.0330, 0.0659, 0.0769, 0.0513,
11 : 0.0256, 0.0513, 0.1026, 0.0256, 0.0385, 0.0641, 0.0256, 0.0000, 0.0152, 0.0758,
0.0303,
```

Za $m = 6$ indeksi koincidencije su blizu vrijednosti 0.064 što je očekivano za tekst pisan na hrvatskom jeziku. Možemo pretpostaviti da je **6 duljina ključne riječi**. To je u skladu s Kasiskijevim testom. Dakle dobivano ovakve blokove:

In [31]:
```
(blokovi := podijeli(sifrat, 6))
```

Out[31]:
```
['YAPDRAGDTXCGOKBDDPVGHADJ',
 'CZHGSSWUYFWIODSBAHSWJWL',
 'GHQMLROHQZYJGZMHTDNIDRE',
 'RWEMIOXNMREEXVXNNHQIYXS',
 'CYYBTKUKVPVTPRGKKXKTIAX',
 'AMEAGKLDTEOAEADLKUTAIUD']
```

Pišemo funkcije za šifriranje teksta Cezarovom šifrom.

In [32]:
```
def pomak(c, k) :
    k = (k + 26) % 26
    return chr(65 + (ord(c) + k - 65) % 26)

def pomakni(txt, k) :
    rez = ""
    for c in txt :
        rez += pomak(c, k)
    return rez
```

Prva metoda (neuspjeh)

Treba nam funkcija za računanje međusobnih indeksa koincidencije dva bloka.

In [40]:
```
def MIc(z1, z2) :
    f1 = frekvencije(z1)
    f2 = frekvencije(z2)
    n1 = len(z1)
    n2 = len(z2)
    Sum = 0
    for c in "ABCDEFGHIJKLMNOPQRSTUVWXYZ" :
        try : Sum += f1[c]*f2[c]
        except KeyError : pass
    Sum /= n1 * n2
    return Sum
```

Funkcija `mics` uzima dva indeksa blokova i računa vektor međusobnih indeksa koincidencije za te blokove za svaki od mogućih pomaka drugog bloka.

In [56]:
```
def mics(i, j) :
    rez = []
    for k in range(0, -26, -1) :
        mic = MIc(blokovi[i], pomakni(blokovi[j], k))
        if mic < 0.044 : mic = 0
        rez.append(mic)
    return rez
```

Funkcija `Dist` uzima indeks bloka i i generira tablicu čiji stupci za j t.d. $6 > j > i$ sadrže vrijednosti funkcije `mics(i,j)` ako su veće od 0.44.

In [57]:
```
def Dist(i) :
    df = pd.DataFrame({k : mics(i,k) for k in range(i+1,6)} )
    m = df > 0.044
    return df.where(m)
```

Pokušajmo sada izvući odnose k_i-ova iz ključa. Gledamo stupce u kojima je većina vrijednosti značajno manja od, a jedna ili dvije blizu 0.064.

In [189]:
```
Dist(0)
```

Out[189]:

	1	2	3	4	5
0	NaN	0.045290	NaN	0.048913	0.068841
1	NaN	0.045290	0.045290	NaN	0.045290
2	NaN	NaN	NaN	NaN	NaN
3	0.050725	0.047101	NaN	NaN	0.048913
4	NaN	0.048913	NaN	0.050725	NaN
5	0.045290	NaN	NaN	NaN	0.050725
6	NaN	0.052536	NaN	NaN	NaN
7	NaN	NaN	0.045290	0.059783	NaN
8	0.045290	NaN	NaN	NaN	NaN
9	NaN	0.045290	NaN	NaN	NaN
10	NaN	0.050725	0.047101	NaN	NaN
11	NaN	NaN	NaN	NaN	0.052536
12	0.045290	NaN	NaN	NaN	NaN
13	NaN	NaN	0.045290	NaN	NaN
14	NaN	0.048913	NaN	NaN	NaN
15	0.052536	NaN	0.047101	NaN	NaN
16	0.045290	NaN	NaN	NaN	NaN
17	NaN	NaN	0.045290	0.056159	NaN
18	0.047101	NaN	NaN	0.045290	NaN
19	0.057971	NaN	NaN	NaN	NaN
20	NaN	NaN	0.059783	NaN	0.054348
21	NaN	NaN	NaN	0.063406	NaN
22	0.048913	NaN	NaN	NaN	NaN
23	NaN	NaN	0.045290	NaN	0.081522
24	NaN	NaN	0.048913	NaN	NaN
25	0.048913	NaN	NaN	NaN	NaN

$$k_0 = k_3 - 20 \,\mathrm{i}\, k_0 = k_1 - 19$$

In [190]:
```
Dist(1)
```

Out[190]:

	2	3	4	5
0	NaN	NaN	NaN	NaN
1	NaN	0.051040	0.066163	0.047259
2	NaN	NaN	NaN	0.045369
3	NaN	NaN	0.047259	NaN
4	NaN	NaN	NaN	0.064272
5	NaN	0.058601	NaN	NaN
6	NaN	NaN	NaN	NaN
7	0.056711	NaN	NaN	NaN
8	NaN	NaN	NaN	0.077505
9	NaN	NaN	NaN	NaN
10	NaN	NaN	0.047259	NaN
11	0.062382	NaN	NaN	NaN
12	NaN	0.064272	0.047259	0.064272
13	NaN	NaN	NaN	NaN
14	NaN	NaN	0.073724	NaN
15	0.049149	NaN	NaN	NaN
16	NaN	0.066163	0.045369	NaN
17	NaN	NaN	NaN	NaN
18	NaN	NaN	0.064272	0.049149
19	NaN	NaN	NaN	0.047259
20	NaN	NaN	NaN	NaN
21	NaN	0.056711	NaN	NaN
22	NaN	NaN	NaN	NaN
23	NaN	NaN	0.058601	0.049149
24	NaN	NaN	NaN	NaN
25	NaN	0.049149	0.056711	NaN

$k_1 = k_2 - 7$ ili $k_1 = k_2 - 11$

In [194]:
```
Dist(2)
```

Out[194]:

	3	4	5
0	NaN	NaN	NaN
1	0.054820	NaN	0.047259
2	NaN	NaN	NaN
3	NaN	0.062382	NaN
4	NaN	NaN	NaN
5	0.058601	NaN	NaN
6	0.058601	NaN	NaN
7	NaN	0.060491	NaN
8	NaN	NaN	NaN
9	0.047259	NaN	NaN
10	0.049149	NaN	NaN
11	NaN	0.052930	NaN
12	NaN	0.056711	0.047259
13	NaN	NaN	0.060491
14	0.052930	NaN	NaN
15	NaN	NaN	NaN
16	NaN	0.045369	NaN
17	NaN	0.047259	NaN
18	NaN	NaN	NaN
19	NaN	NaN	0.045369
20	NaN	0.058601	0.052930
21	NaN	NaN	NaN
22	NaN	NaN	0.047259
23	NaN	NaN	0.062382
24	NaN	NaN	NaN
25	NaN	NaN	NaN

$k_2 = k_5 - 13$ ili $k_2 = k_5 - 20$ ili $k_2 = k_5 - 23$

In [192]:
```
Dist(3)
```

Out[192]:

	4	5
0	NaN	NaN
1	NaN	NaN
2	0.058601	NaN
3	0.045369	0.060491
4	NaN	NaN
5	NaN	NaN
6	0.062382	0.058601
7	NaN	0.060491
8	NaN	NaN
9	NaN	NaN
10	NaN	NaN
11	0.056711	NaN
12	NaN	NaN
13	0.062382	0.069943
14	NaN	0.047259
15	0.047259	NaN
16	NaN	0.045369
17	NaN	NaN
18	NaN	NaN
19	0.047259	NaN
20	NaN	NaN
21	NaN	NaN
22	0.045369	0.075614
23	0.054820	NaN
24	0.058601	NaN
25	NaN	NaN

In [196]:
```
Dist(4)
```

Out[196]:

	5
0	0.047259
1	NaN
2	0.049149
3	NaN
4	NaN
5	NaN
6	NaN
7	NaN
8	NaN
9	0.056711
10	0.054820
11	0.052930
12	NaN
13	NaN
14	NaN
15	NaN
16	0.062382
17	NaN
18	NaN
19	0.052930
20	0.047259
21	NaN
22	NaN
23	NaN
24	NaN
25	NaN

$k_4 = k_5 - 9$ ili $k_4 = k_5 - 16$

Sve pretpostavke na istom mjestu:

$k_0 = k_3 - 20$ i $k_0 = k_1 - 19$
$k_1 = k_2 - 7$ ili $k_1 = k_2 - 11$
$k_2 = k_5 - 13$ ili $k_2 = k_5 - 20$ ili $k_2 = k_5 - 23$
$k_4 = k_5 - 9$ ili $k_4 = k_5 - 16$

Bez brige o k_4 pišemo neke generatore ključa.

In [66]:
```
def fkljuc1(k) :
    Key = [k, k+19, k+19+7, k+20, 0, k+19+13]
    for i in range(6) : Key[i] %= 26
    return Key

def fkljuc2(k) :
    Key = [k, k+19, k+19+11, k+20, 0, k+19+20]
    for i in range(6) : Key[i] %= 26
    return Key

def fkljuc3(k) :
    Key = [k, k+19, k+19+11, k+20, 0, k+19+23]
    for i in range(6) : Key[i] %= 26
    return Key

def prevedi(kljuc) :
    rez = ""
    for v in kljuc :
        rez += pomak("A", v)
    return rez
```

Ne dobivamo ništa smisleno niti za jedan od slučajeva. Na primjer za jedan od mogućih ključeva:

In [166]:
```
for k in range(0, 26) :
    rez = ""
    kljuc = fkljuc1(k)
    for i in range(len(sifrat)) :
        bindeks = i % 6
        rez += pomak(sifrat[i], kljuc[bindeks])
        if bindeks == 5 : rez += " "
    print(f"{prevedi(kljuc)} : {rez[:60]}")
```

```
ATAUAG : YVGLCG ASHQYS PAQYYK DZMGBG RLLCTM ALRIKQ GPORUR DNHHKJ TRQG
BUBVAH : ZWHMCH BTIRYT QBRZYL EANHBH SMMDTN BMSJKR HQPSUS EOIIKK USRH
CVCWAI : AXINCI CUJSYU RCSAYM FBOIBI TNNETO CNTKKS IRQTUT FPJJKL VTSI
DWDXAJ : BYJOCJ DVKTYV SDTBYN GCPJBJ UOOFTP DOULKT JSRUUU GQKKKM WUTJ
EXEYAK : CZKPCK EWLUYW TEUCYO HDQKBK VPPGTQ EPVMKU KTSVUV HRLLKN XVUK
FYFZAL : DALQCL FXMVYX UFVDYP IERLBL WQQHTR FQWNKV LUTWUW ISMMKO YWVL
GZGAAM : EBMRCM GYNWYY VGWEYQ JFSMBM XRRITS GRXOKW MVUXUX JTNNKP ZXWM
HAHBAN : FCNSCN HZOXYZ WHXFYR KGTNBN YSSJTT HSYPKX NWVYUY KUOOKQ AYXN
IBICAO : GDOTCO IAPYYA XIYGYS LHUOBO ZTTKTU ITZQKY OXWZUZ LVPPKR BZYO
JCJDAP : HEPUCP JBQZYB YJZHYT MIVPBP AUULTV JUARKZ PYXAUA MWQQKS CAZP
KDKEAQ : IFQVCQ KCRAYC ZKAIYU NJWQBQ BVVMTW KVBSKA QZYBUB NXRRKT DBAQ
LELFAR : JGRWCR LDSBYD ALBJYV OKXRBR CWWNTX LWCTKB RAZCUC OYSSKU ECBR
MFMGAS : KHSXCS METCYE BMCKYW PLYSBS DXXOTY MXDUKC SBADUD PZTTKV FDCS
NGNHAT : LITYCT NFUDYF CNDLYX QMZTBT EYYPTZ NYEVKD TCBEUE QAUUKW GEDT
OHOIAU : MJUZCU OGVEYG DOEMYY RNAUBU FZZQTA OZFWKE UDCFUF RBVVKX HFEU
PIPJAV : NKVACV PHWFYH EPFNYZ SOBVBV GAARTB PAGXKF VEDGUG SCWWKY IGFV
QJQKAW : OLWBCW QIXGYI FQGOYA TPCWBW HBBSTC QBHYKG WFEHUH TDXXKZ JHGW
RKRLAX : PMXCCX RJYHYJ GRHPYB UQDXBX ICCTTD RCIZKH XGFIUI UEYYKA KIHX
SLSMAY : QNYDCY SKZIYK HSIQYC VREYBY JDDUTE SDJAKI YHGJUJ VFZZKB LJIY
TMTNAZ : ROZECZ TLAJYL ITJRYD WSFZBZ KEEVTF TEKBKJ ZIHKUK WGAAKC MKJZ
UNUOAA : SPAFCA UMBKYM JUKSYE XTGABA LFFWTG UFLCKK AJILUL XHBBKD NLKA
VOVPAB : TQBGCB VNCLYN KVLTYF YUHBBB MGGXTH VGMDKL BKJMUM YICCKE OMLB
WPWQAC : URCHCC WODMYO LWMUYG ZVICBC NHHYTI WHNEKM CLKNUN ZJDDKF PNMC
XQXRAD : VSDICD XPENYP MXNVYH AWJDBD OIIZTJ XIOFKN DMLOUO AKEEKG QOND
YRYSAE : WTEJCE YQFOYQ NYOWYI BXKEBE PJJATK YJPGKO ENMPUP BLFFKH RPOE
ZSZTAF : XUFKCF ZRGPYR OZPXYJ CYLFBF QKKBTL ZKQHKP FONQUQ CMGGKI SQPF
```

Druga metoda (uspjeh)

Trebaju nam podaci o frekvencijama slova u tekstovima na hrvatskom jeziku.

In [103]:
```
P = {
    "A":115,
    "I":98,
    "O":90,
    "E":84,
    "N":66,
    "S":56,
    "R":54,
    "J":51,
    "T":48,
    "U":43,
    "D":37,
    "K":36,
    "V":35,
    "L":33,
    "M":31,
    "P":29,
    "C":28,
    "Z":23,
    "G":16,
    "B":15,
    "H":8,
    "F":3
}

# pretvaramo u postotke
for k, v in P.items() :
    P[k] = v / 1000
```

Sljedeća funkcija `MIc` je zapravo $\mathrm{MI}_c(x, \cdot)$ gdje je x neki -tipičan tekst na hrvatskom jeziku.

In [84]:
```
def MIc(z) :
    f = frekvencije(z)
    n = len(z)
    Sum = 0
    for c in "ABCDEFGHIJKLMNOPQRSTUVXYZ" :
        try : Sum += P[c]*f[c]/n
        except KeyError : pass
    return Sum
```

Pogledajmo kako se ponaša na primjer na nultom bloku.

In [100]:
```
for h in range(0, -26, -1) :
    mic = MIc(pomakni(blokovi[0], h))
    print(f"{h} : {mic}")
```

```
0 : 0.04170833333333334
-1 : 0.03575
-2 : 0.037625000000000006
-3 : 0.04733333333333333
-4 : 0.024874999999999998
-5 : 0.025541666666666667
-6 : 0.045125
-7 : 0.03691666666666667
-8 : 0.029625000000000002
-9 : 0.03875
-10 : 0.034416666666666665
-11 : 0.041708333333333326
-12 : 0.04004166666666666
-13 : 0.03508333333333333
-14 : 0.03854166666666667
-15 : 0.06133333333333332
-16 : 0.03558333333333333
-17 : 0.034125
-18 : 0.043708333333333335
-19 : 0.04
-20 : 0.03620833333333333
-21 : 0.04525
-22 : 0.035875
-23 : 0.037625
-24 : 0.03991666666666667
-25 : 0.036333333333333336
```

Vrijednosti izgledaju puno pravilnije. Većinom su manje od 0.044 i jedna je oko 0.064. Izračunajmo sada za svaki blok maksimalnu vrijednost indeksa koincidencije i za koji pomak se dobiva.

In [92]:
```
for bindex in range(6) :
    mics = {}
    for h in range(0, -26, -1) :
        mics[h] = MIc(pomakni(blokovi[bindex], h))
    M = max(mics.values())
    h = [k for k, v in mics.items() if v == M][0]

    print(f"blok = {bindex} : M = {M}, za h = {h}")
```

```
blok = 0 : M = 0.06133333333333332, za h = -15
blok = 1 : M = 0.06208695652173914, za h = -14
blok = 2 : M = 0.05939130434782609, za h = -25
blok = 3 : M = 0.06665217391304348, za h = -4
blok = 4 : M = 0.058347826086956524, za h = -6
blok = 5 : M = 0.06491304347826087, za h = 0
```

Dakle numerička reprezentacija ključa je $(15, 14, 25, 4, 6, 0)$.

In [104]:
```
prevedi(kljuc := (15,14,25,4,6,0))
```

Out[104]:
```
'POZEGA'
```

Ključna riječ je POZEGA.
Dešifrirajmo sada tekst.

In [108]:
```
tekst = ""
for i in range(len(sifrat)) :
    bindex = i % 6
    tekst += pomakni(sifrat[i], -kljuc[bindex])
print(tekst)
```

```
JOHNWALLISSMATRASEOSNIVACEMENGLESKEKRIPTOLOGIJEDEKRIPTIRANJENIZAPORUKANAZAHTJEVPARLA
MENTADONIJELOMUJEKATEDRUGEOMETRIJENASVEUCILISTUUOXFORDU
```

Otvoreni tekst je:
JOHN WALLIS SMATRA SE OSNIVAČEM ENGLESKE KRIPTOLOGIJE. DEKRIPTIRANJE NIZA PORUKA NA ZAHTJEV PARLAMENTA DONIJELO MU JE KATEDRU GEOMETRIJE NASVEUČILIŠTU U OXFORDU

Zadatak 2

Ubacujemo podatke u varijable.

In [16]:
```
otvoreni = "GIOVANNI SORO"
kljuc = "CRYPTANALYSIS"
```

Uređujemo podatke tako da ih možemo koristiti u algoritmu.

In [17]:
```
otvoreni = otvoreni.replace(" ", "")
blokovi = []
for i in range(0, len(otvoreni), 2) :
    blokovi.append(otvoreni[i] + otvoreni[i+1])

blokovi
```

Out[17]:
```
['GI', 'OV', 'AN', 'NI', 'SO', 'RO']
```

In [19]:
```
koristeni = set()
key = ""
for c in kljuc + "ABCDEFGHIJKLMNOPQRSTUVXYZ" :
    if not c in koristeni :
        key += c
        koristeni.add(c)

key
```

Out[19]:
```
'CRYPTANLSIBDEFGHJKMOQUVXZ'
```

Konstruiramo matricu.

In [31]:
```
def show(M) :
    for r in range(0,5) :
        for c in range(0,5) :
            print(matrica[r][c], end=" ")
        print()
```

In [37]:
```
matrica = np.chararray((5,5), unicode=True)
i = 0
for r in range(0,5) :
    for c in range(0,5) :
        matrica[r][c] = key[i]
        i += 1

show(matrica)
```

```
C R Y P T
A N L S I
B D E F G
H J K M O
Q U V X Z
```

Šifrirajmo blokove teksta. Funkcija `polozaj` vraća koordinate slova u matrici --- (redak, stupac).

In [45]:
```
def polozaj(M, c) :
    mask = M.find(c)
    for i in range(0, 5) :
        for j in range(0, 5) :
            if mask[i][j] == 0 : return (i, j)
```

Ovisno o kojem od tri slučaja se radi, šifriramo blokove od dva slova.

In [52]:
```
def sifriraj(c, d) :
    cr, cc = polozaj(matrica, c)
    dr, dc = polozaj(matrica, d)
    # isti redak
    if cr == dr :
        return (
            matrica[cr][(cc + 1) % 5],
            matrica[cr][(dc + 1) % 5]
        )
    # isti stupac
    elif cc == dc :
        return (
            matrica[(cr + 1) % 5][cc],
            matrica[(dr + 1) % 5][cc]
        )
    # pravokutnik
    else :
        return (
            matrica[cr][dc],
            matrica[dr][cc]
        )
```

In [57]:
```
sifrat = ""

for c, d in blokovi:
    ce, de = sifriraj(c, d)
    sifrat += ce + de

print(sifrat)
```

```
OGKZNLLAIMTJ
```

Dakle šifrirani tekst je **OGKZNLLAIMTJ**

Zadatak 3

In [118]:
```
otvoreni = "VERNAM"
sifrat = "PJKZYU"

o_blokovi = []
s_blokovi = []
for i in range(0,6,2) :
    o_blokovi.append(otvoreni[i:i+2])
    s_blokovi.append(sifrat[i:i+2])

print(o_blokovi)
print(s_blokovi)
```

```
['VE', 'RN', 'AM']
['PJ', 'KZ', 'YU']
```

Reprezentirajmo numeričkim ekvivalentima blokove tekstova.

In [119]:
```
co_blokovi = []
cs_blokovi = []

for c,d in o_blokovi :
    co_blokovi.append((ord(c)-65, ord(d)-65))
for c,d in s_blokovi :
    cs_blokovi.append((ord(c)-65, ord(d)-65))

print(co_blokovi)
print(cs_blokovi)
```

```
[(21, 4), (17, 13), (0, 12)]
[(15, 9), (10, 25), (24, 20)]
```

In [130]:
```
X = np.matrix(co_blokovi[0:2])
Y = np.matrix(cs_blokovi[0:2])
print(X)
print()
print(Y)
```

```
[[21  4]
 [17 13]]

[[15  9]
 [10 25]]
```

Provjerimo ima li matrica X inverz. Determinanta od X je:

In [134]:
```
int(np.linalg.det(X)) % 26
```

Out[134]:
```
23
```

Tražimo inverz od 23 modulo 26.

In [137]:
```
[n for n in range(1, 27) if (23 * n) % 26 == 1]
```

Out[137]:
```
[17]
```

Računamo inverz od X : `Xi` .

In [147]:
```
Xi = 17 * np.matrix([
    [13, -4],
    [-17, 21]
])
Xi %= 26
print(Xi)
```

```
[[13 10]
 [23 19]]
```

Provjera.

In [150]:
```
print((Xi * X) % 26)
```

```
[[1 0]
 [0 1]]
```

In [152]:
```
K = (Xi * Y) % 26
print(K)
```

```
[[ 9  3]
 [15  6]]
```

Provjerimo preslikavaju li se blokovi korektno. Očekujemo parove:

In [154]:
```
print(cs_blokovi)
```

```
[(15, 9), (10, 25), (24, 20)]
```

In [158]:
```
for par in co_blokovi :
    print((np.matrix(par) * K) % 26)
```

```
[[15  9]]
[[10 25]]
[[24 20]]
```

Dakle, $K = \begin{pmatrix} 9 & 3 \\ 15 & 6 \end{pmatrix}$