OŠ Vižmarje-Brod

Raziskovalna naloga

RAZBIJANJE SUBSTITUCIJSKE ŠIFRE S FREKVENČNO ANALIZO

Raziskovalno področje: RAČUNALNIŠTVO

Avtor: Taras Vuk, 7.r.

Mentorica: Petra Škofic Valjavec

Somentor: Miha Vuk

Ljubljana, 2022

KAZALO

1. POVZETEK

Šifriranje sporočil zadnje čase postaja vse pomembnejše. Veliko šifer (šifrirnih algoritmov), katere so bile včasih nezlomljive, se danes razbije čisto preprosto. Sam sem se lotil razbiti substitucijsko šifro s pomočjo frekvenčne analize. Vseh ključev je preveč, da bi pogledal vse, a se iz porazdelitve znakov s pametnim algoritmom da najti pravega.

Ključne besede: kriptografija, substitucijska šifra, frekvenčna analiza besedil

2. UVOD

Pred leti sem naredil program, ki iz besedila vzame pogostosti zaporedij črk in presledka ter z enakimi verjetnostmi generira naslednjo črko in nadalje celo besedilo. Generirano besedilo je bilo sicer berljivo, a ni imelo preveč smisla. Letos pa sem ugotovil, da bi lahko mogoče na podoben način razbijal substitucijsko šifro.

HIPOTEZA:

Substitucijsko šifro se da v manj kot 1 minuti razbiti tako dobro, da je dešifrirano besedilo mogoče razumeti.

3. TEORETIČNI DEL

3.1 Substitucijska šifra

Eno abecedna substitucijska šifra ali monoalfabetska zamenjalna šifra je šifra pri kateri se neka črka ali znak v odprtem besedilu vedno nadomesti z isto črko ali znakom.

Slika 1: Substitucija angleške abecede. (Vir: Wikipedija [2])

Po zgornjem ključu bi se na primer »raziskovalna naloga« zakodiralo v »kqmolagcqsfq fqsguq«.

Pred kodiranjem se vse črke spremeni v male, stavčna ločila pa se napišejo z besedo (»pika«, »vejica«,…) ali izpustijo.

3.1.1 Izbira ključa

Naključna izbira: Ključ je najbolje izbrati popolnoma naključno, kar zagotavlja največjo varnost.

```
Original: ABCČDEFGHIJKLMNOPRSŠTUVZŽ
   Ključ: UFLPŽDRASJM CŠONBVTEHZČKGI
```

Slika 2: Popolnoma naključno izbran ključ.

Izbira z geslom: Ključ lahko izberemo z geslom. Pod originalno abecedo najprej napišemo geslo (vsak znak samo enkrat) in nato vse preostale znake.

```
Original: ABCČDEFGHIJKLMNOPRSŠTUVZŽ
   Ključ: NOVAKJEZBCČDFGHILMPRSŠTUŽ (NOVAKJANEZ)
```

Slika 3: Ključ izbran z geslom »novakjanez«.

Izbira s Cezarjevo šifro: Cezarjeva šifra originalno abecedo le zamakne za dogovorjeno število mest.

```
Original: ABCČDEFGHIJKLMNOPQRŠSTUVWXYZŽ
   Ključ: GHIJKLMNOPQRSŠTUVWXYZŽABCČDEF (7)
```

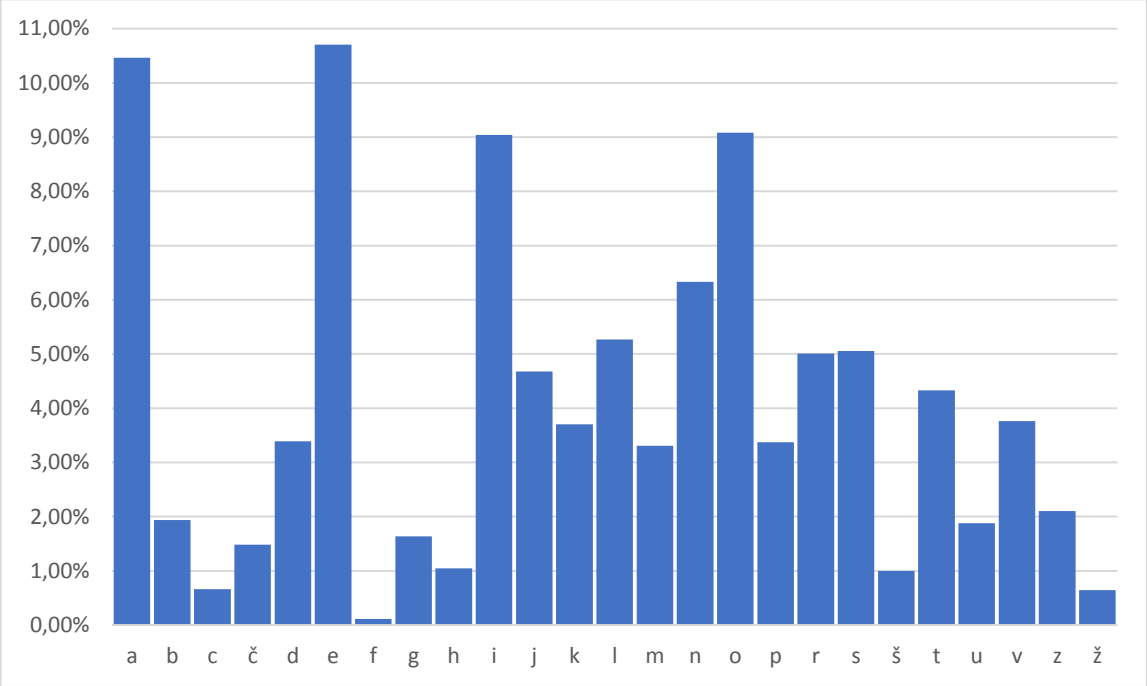
Slika 4: Ključ izbran s Cezarjevo šifro zamika za 7 mest.

3.1.2 Varnost

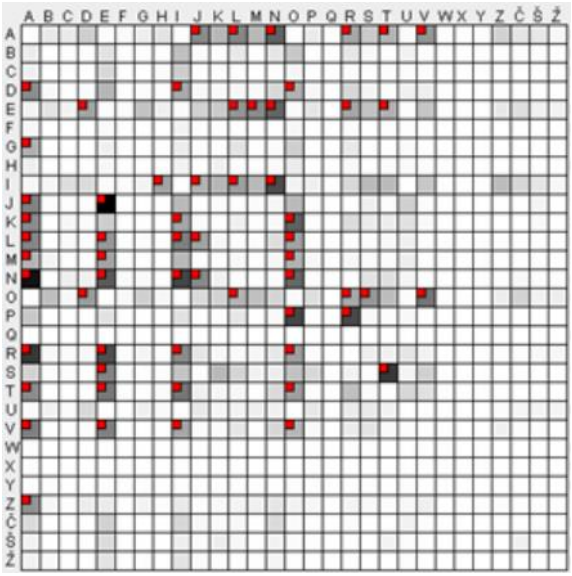
Eno abecedne substitucijske šifre se ne uporablja več. Zaradi njene lastnosti, da se vsak znak zamenja vedno z istim, lahko s štetjem pogostosti skupkov črk hitrost razbijanja močno pohitrimo. Vseeno pa obstaja kar 26! (ob upoštevanju presledka) možnih ključev, kar je približno 2^{88}. To pa je tudi za najmočnejše računalnike preveč, da bi poskusili vse.

3.2 Frekvenčna analiza besedila

Pri frekvenčni analizi besedila ugotavljamo pogostosti znakov ali črk ter njihovih parov, trojk,...



Slika 5: Pogostosti črk v slovenščini. (Vir: P. Jakopin, doktorska disertacija)



Slika 6: Pogostosti parov črk v slovenščini (levo prva črka, zgoraj druga črka). (Vir: S. Kodrič, diplomsko delo)

Iz frekvenc skupkov črk se da določiti jezik ter celo avtorja besedila in besedilne značilnosti. To je seveda možno le ob dovolj dolgih besedilih.

3.2.1 Merjenje razdalj med matrikami

Ko izračunamo frekvence, jih zapišemo v matriko. Med dvema matrikama lahko potem izračunamo razdaljo, ki nam pove razliko med matrikama. Ta predstavlja podobnost pogostosti parov črk.

Evklidska razdalja:

$$\sqrt{\sum_{i=1}^{N}(x_i - y_i)^2}$$

Manhattanova razdalja:

$$\sum_{i=1}^{N}|x_i - y_i|$$

Slika 7: Evklidska in Manhattanska razdalja med vektorjema x in y. Za matrike je podobno. (Vir: S. Kodrič, diplomsko delo)

4. RAZISKOVALNI DEL – PROGRAM

Po teoriji sem se lotil pisanja programa. Napisan je v C#.

4.1 Vhodni podatki in začetne spremenljivke

Program za vhodne podatke dobi *slovar* (testno besedilo), *tajnopis* in *abeceda*, ki je abeceda iskanega jezika (dodal sem še presledek). Za slovar sem uporabil besedila iz Digitalne knjižnice Slovenije [5].

```
static string slovar = File.ReadAllText(@"C:\Users\Taras\Downloads\slovar.txt");
static string tajnopis = File.ReadAllText(@"C:\Users\Taras\Downloads\tajnopis.txt");

const string abeceda = " abcčdefghijklmnoprsštuvzž";

static StringBuilder ugibanKljuč = new StringBuilder(abeceda);
```

Slika 8: Vhodni podatki programa.

Program ob tem uvede tudi spremenljivko *ugibanKljuč*, ki predstavlja s kakšnim ključem program trenutno misli, da je bilo zakodirano besedilo. Njegova začetna vrednost je kar *abeceda*.

4.2 Frekvenčna analiza parov znakov

Ko imamo *slovar* in *tajnopis*, lahko naredimo frekvenčno analizo parov črk in dobimo matriki *frekvenceS* in *frekvenceT*.

```
const double[,] frekvenceS = frekvenceParov(slovar);
const double[,] frekvenceT = frekvenceParov(tajnopis)
```

Slika 9: Zapis frekvence slovarja in tajnopisa.

Da program izračuna frekvence parov znakov, besedilo najprej besedilo pretvori v seznam številk. Vsak znak, ki je v nizu *abeceda* zapiše z mestom v njem. Znak za naslednjo vrstico razume kot presledek. Nato se sprehodi po seznamu in za vsak znak poveča *frekvence*[*besediloByte*[i], *besediloByte*[i+1]] za 1/(število zaznanih znakov besedila).

```
public static double[,] frekvenceParov(string besedilo)
{
    double[,] frekvence = new double[abeceda.Length, abeceda.Length];

    besedilo = besedilo.ToLower();

    int len = 0;
    byte[] besediloByte = new byte[besedilo.Length];
    for (int i = 0; i < besedilo.Length; i++)
    {
        int c = abeceda.IndexOf(besedilo[i]);

        if (besedilo[i] == '\n')
            c = abeceda.IndexOf(' ');

        if (c < 0)
            continue;

        besediloByte[len] = (byte)c;
        len++;
    }

    for (int i = 1; i < len; i++)
        frekvence[besediloByte[i - 1], besediloByte[i]] += 1.0 / len;

    return frekvence;
}
```

Slika 10: Podprogram za frekvenčno analizo parov znakov.

4.3 Razporeditev parov znakov po pomembnosti

Pare znakov tajnopisa bomo razporedili po pomembnosti (0 je najmanj pomemben *abeceda.Length* * *abeceda.Length* pa najbolj), da se bomo pri njihovem kasnejšem premikanju najprej prilagodili pomembnejšim. Postopek deluje tako, da vzame vsak par in se z njim premika po seznamu navzgor dokler ne pride do pomembnejšega kot je sam in ga vpiše. Ta metoda se imenuje urejanje z ustavljanjem. Pomembnost se računa po formuli *frekvenceT*[x, y] * 1.2 + *frekvenceT*[y, x].

```
double[,] pomParov = new double[abeceda.Length * abeceda.Length, 3];
for (int i = 0; i < abeceda.Length * abeceda.Length; i++)
    pomParov[i, 0] = pomParov[i, 1] = pomParov[i, 2] = -1;

for (int x = 0; x < abeceda.Length; x++)
    for (int y = 0; y < abeceda.Length; y++)
        for (int i = 0; i < abeceda.Length * abeceda.Length - 1; i++)
        {
            if (frekvenceT[x, y] * 1.2 + frekvenceT[y, x] <= pomParov[i + 1, 0])
            {
                pomParov[i, 1] = x;
                pomParov[i, 2] = y;
                pomParov[i, 0] = frekvenceT[x, y] * 1.2 + frekvenceT[y, x];
                break;
            }
            else
            {
                pomParov[i, 0] = pomParov[i + 1, 0];
                pomParov[i, 1] = pomParov[i + 1, 1];
                pomParov[i, 2] = pomParov[i + 1, 2];
                if (i == abeceda.Length * abeceda.Length - 2)
                {
                    pomParov[i + 1, 1] = x;
                    pomParov[i + 1, 2] = y;
                    pomParov[i + 1, 0] = frekvenceT[x, y] * 1.2 + frekvenceT[y, x];
                }
            }
        }
```

Slika 11: Program za razvrščanje parov znakov po pomembnosti.

4.4 Premikanje parov znakov

Zdaj, ko imamo vse pripravljeno, lahko začnemo iskati ključ. Sprehodimo se čez seznam *pomParov* (zdaj od najbolj do najmanj pomembnega) in vsak par znakov zamenjamo s takim parom, da sta si po zamenjavi *frekvenceT* in *frekvenceS* najbolj podobni. Parov, ki imajo prvo in drugo črko enako ne menjamo z pari, ki imajo prvo in drugo črko različno in obratno. V resnici sploh ne spreminjamo *frekvenceT*, ampak le *ugibanKljuč*, ki nam pove kako gledati na *frekvenceT*.

```
for (int i = abeceda.Length * abeceda.Length - 1; i > 0; i--)
{
    int maxX = 0, maxY = 0;
    double minRaz = double.MaxValue;
    double raz;

    for (int x = 0; x < abeceda.Length; x++)
        for (int y = 0; y < abeceda.Length; y++)
        {
            if (x == y && pomParov[i, 1] != pomParov[i, 2])
                continue;
            if (x != y && pomParov[i, 1] == pomParov[i, 2])
                continue;

            raz = učinkovitostPremika((int)pomParov[i, 1], (int)pomParov[i, 2], x, y);

            if (minRaz > raz)
            {
                maxX = x;
                maxY = y;
                minRaz = raz;
            }
        }

    char a = ugibanKljuč[(int)pomParov[i, 1]];
    char b = ugibanKljuč[(int)pomParov[i, 2]];

    ugibanKljuč[(int)pomParov[i, 1]] = ugibanKljuč[maxX];
    ugibanKljuč[(int)pomParov[i, 2]] = ugibanKljuč[maxY];
```

```
    ugibanKljuč[maxX] = a;
    ugibanKljuč[maxY] = b;
 }
```

Slika 12: Program za premikanje parov znakov.

4.4.1 Merjenje učinkovitosti premika

Do sedaj sem pokazal, katere pare se zamenjuje in s katerimi, nisem pa še pokazal na kakšen način se meri učinkovitost premika para. Za računanje učinkovitosti zamenjave para [*črka1*, *črka2*] s parom [*mesto1*, *mesto2*] v *ugibanKljuč* preprosto seštejemo:

- Manhattansko razdaljo med matrikama *frekvenceS* in frekvence*T*, ki jo preberemo z novo nastalim ključem,

- absolutno razliko med pogostostjo črke *mesto1* v slovarju in tajnopisu ter pogostostjo črke *mesto2* v slovarju in tajnopisu krat parameter imenovan *pomČrke*,

- absolutno razliko med pogostostjo para [*mesto1*, *mesto2*] v slovarju in tajnopisu ter pogostostjo para [*mesto2*, *mesto1*] v slovarju in tajnopisu krat parameter imenovan *pomPara.*

O pomenu in optimalnih vrednostih parametrov *pomPara* in *pomČrke*, ki sta v spodnji kodi nastavljena na 1, bom govoril pri rezultatih.

```
public static double učinkovitostPremika(int črka1, int črka2, int mesto1, int mesto2)
{                                          // vsi parametri so indeksi v ugibanKljuč
double oddaljenost = 0;
StringBuilder novaAbeceda = new StringBuilder(ugibanKljuč.ToString());

char a = novaAbeceda[črka1];
novaAbeceda[črka1] = novaAbeceda[mesto1];
novaAbeceda[mesto1] = a;

a = novaAbeceda[črka2];
novaAbeceda[črka2] = novaAbeceda[mesto2];
novaAbeceda[mesto2] = a;

string nAbeceda = novaAbeceda.ToString();
```

```
double pomPara = 1;
double pomČrke = 1;

double a1 = 0; double a2 = 0; double b1 = 0; double b2 = 0;for (int x = 0; x <
abeceda.Length; x++)
    for (int y = 0; y < abeceda.Length; y++)
    {
        double razlika = Math.Abs(frekvenceS[x, y] -
frekvenceT[nAbeceda.IndexOf(abeceda[x]), novaAbeceda.ToString().IndexOf(abeceda[y])]);

        if (mesto2 == novaAbeceda.ToString().IndexOf(abeceda[y]))
        {
            a1 += frekvenceS[x, y];
            a2 += frekvenceT[nAbeceda.IndexOf(abeceda[x]),
       novaAbeceda.ToString().IndexOf(abeceda[y])];

            if (mesto1 == nAbeceda.IndexOf(abeceda[x]))
                razlika *= pomPara;
        }

        if (mesto1 == nAbeceda.IndexOf(abeceda[y]))
        {
            b1 += frekvenceS[x, y];
            b2 += frekvenceT[nAbeceda.IndexOf(abeceda[x]),
                    nAbeceda.IndexOf(abeceda[y])];

            if (m2 == nAbeceda.IndexOf(abeceda[x]))
                razlika *= pomPara;
        }

        oddaljenost += razlika;
    }

return oddaljenost + (Math.Abs(a1 - a2) + Math.Abs(b1 - b2)) * pomČrke;
}
```

Slika 13: Podprogram za merjenje učinkovitosti premika.

5. REZULTATI IN TESTIRANJE PARAMETROV

5.1 Definicija uspešnosti

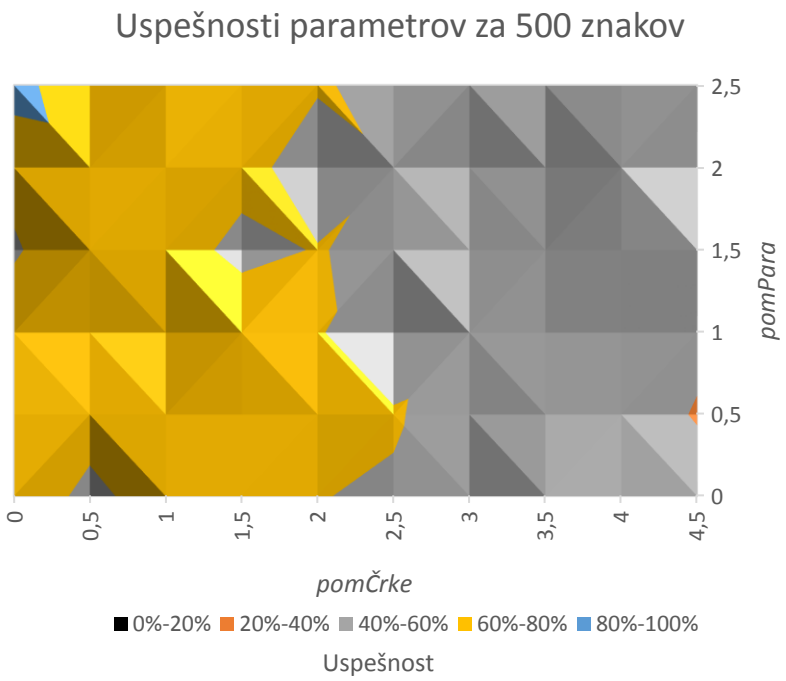
Če sem želel testirati parametre, sem moral na začetku definirati uspešnost. Zamislil sem si jo tako, da se za vsak pravilno ugotovljen znak ključa doda toliko procentov kot je bil ta znak pogost. Za mejo zadostnega razbitja, da se besedilo razume, pa sem si zastavil 85%.

```
prešeg nžšt za mnade matematžge fžzžge aštroiome ži računianižgarje
presek list za mlade matematike fizike astronome in računalnikarje
```

Slika 14: Besedilo po 85% in po 100% razbitja.

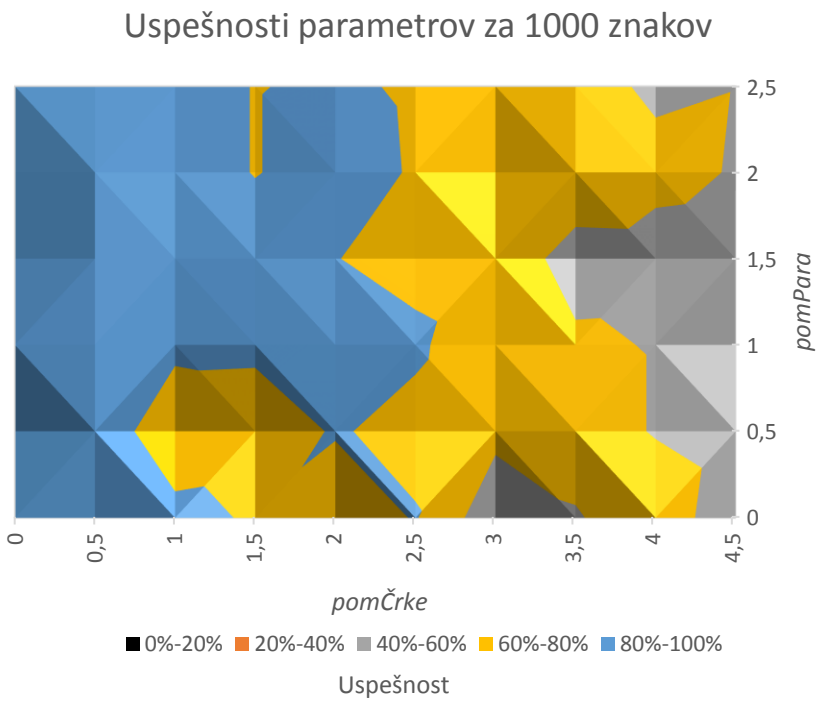
5.2 Testiranje parametrov

Ko je program začel dobro delovati, se je pojavila potreba po testiranju in izbiri optimalnih parametrov. Dva izmed njih sta *pomPara* in *pomČrke*, ki vplivata na merjenje učinkovitosti premika. Testiral sem z *pomPara* 0 do 2,5 ter *pomČrke* 0 do 4,5 s korakom 0,5 in v vseh možnih kombinacijah. Tajnopis za testiranje sem dobil tako, da sem vsak znak čistopisa zamenjal z znakom, ki je na enakem mestu v substitucijski abecedi kot originalni znak v originalni abecedi.



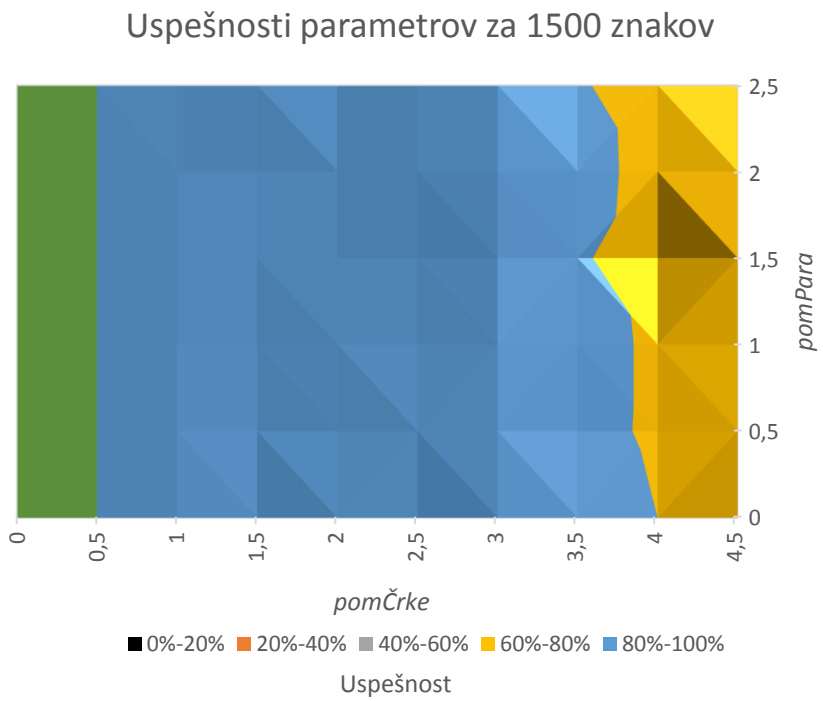
Slika 15: Uspešnosti parametrov za 500 znakov.

Pri testiranju s 500 znaki sta se najbolje izkazala parametra *pomČrke* = 0 in *pomPara* = 2,5 z 88% uspešnostjo.



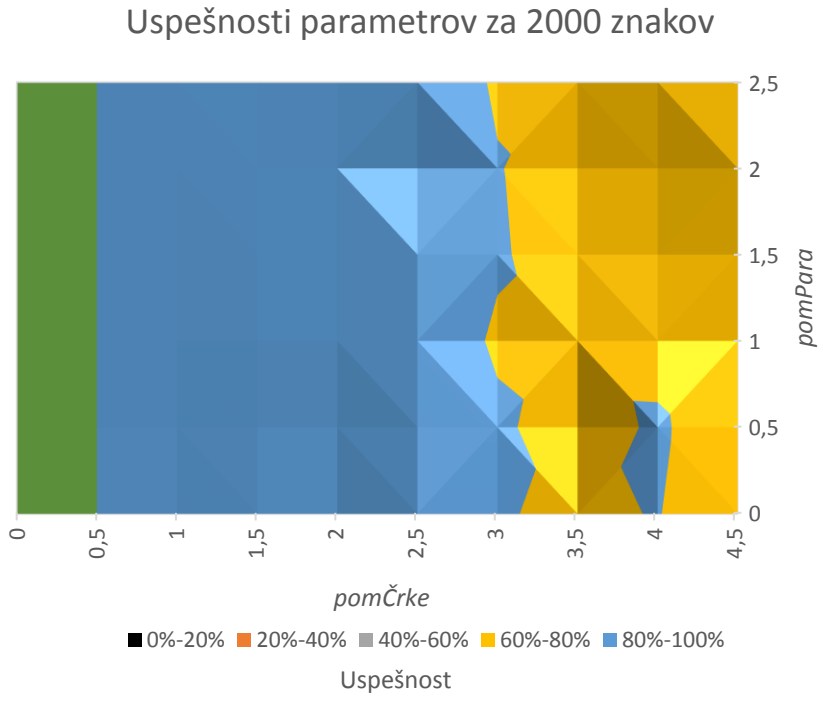
Slika 16: Uspešnosti za 1000 znakov.

Pri testiranju s 1000 znaki sta se najbolje izkazala parametra *pomČrke* = 0,5 in *pomPara* = 1,5 s 96% uspešnostjo.



Slika 17: Uspešnosti za 1500 znakov.

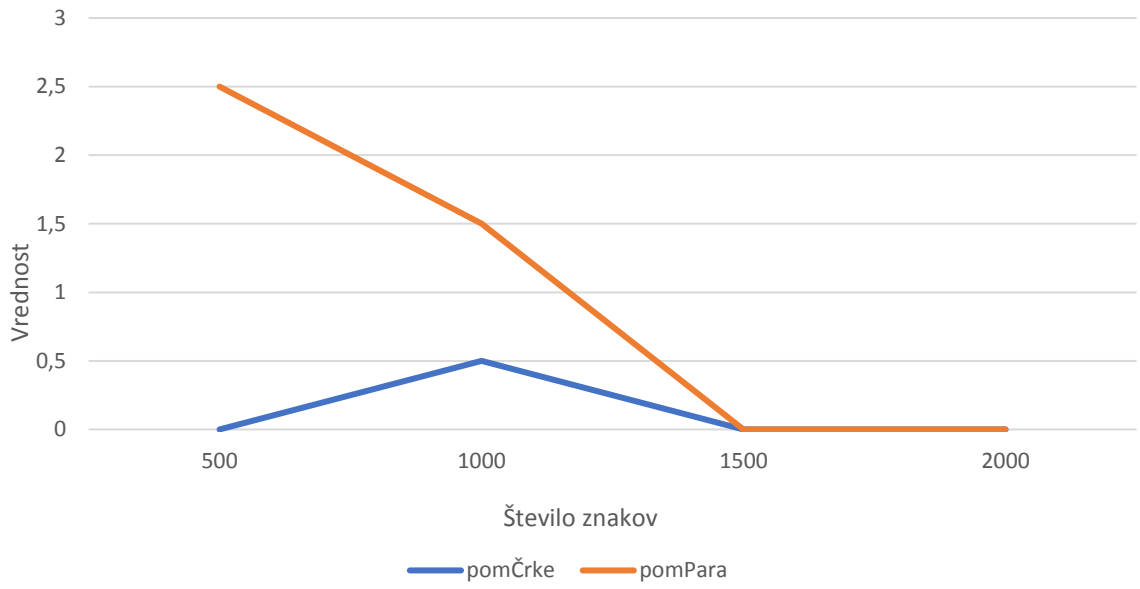
Pri testiranju s 1500 znaki sta se najbolje izkazala parametra *pomČrke* = 0 in celoten razpon *pomPara* s 100% uspešnostjo (označeno z zeleno).



Slika 18: Uspešnosti za 2000 znakov.

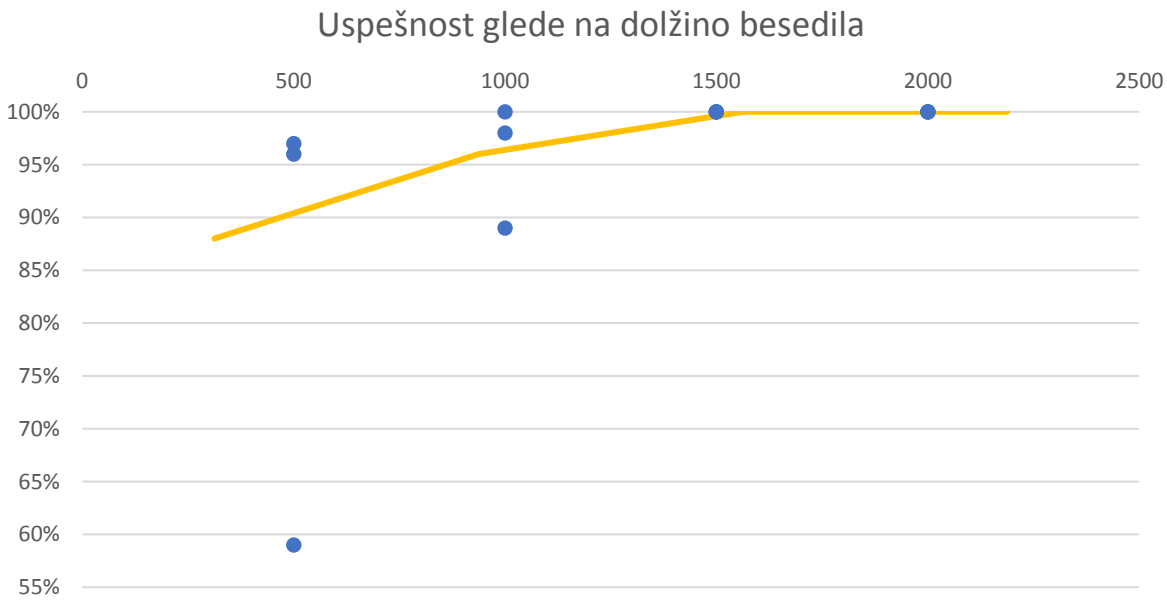
Pri testiranju z 2000 znaki sta se prav tako najbolje izkazala parametra *pomČrke* = 0 in celoten razpon *pomPara* s 100% uspešnostjo.

Optimalne nastavitve parametrov glede na dolžino besedila so prikazane na spodnjem grafu.



Slika 19: Nastavitve pomČrke in pomPara glede na število znakov.

Uspešnost za različne tajnopise in povprečno uspešnost glede na dolžino pa prikazuje spodnji graf. Z večanjem števila znakov tajnopisa uspešnost raste, saj se manjše razlike v besedilih manj poznajo, zato sta si frekvenčni matriki slovarja in dešifriranega tajnopisa bolj podobni.



Slika 20: Povprečna in posamična uspešnost glede na dolžno besedila.

6. ZAKLJUČEK

V raziskovalni nalogi smo spoznali substitucijsko šifro, izdelavo ključa zanjo in frekvenčno analizo besedila. Predstavili smo program za razbijanje substitucijske šifre in test uspešnosti glede na parametre ter kakšna je njihova najboljša nastavitev glede na dolžino besedila. Uspešnost pri najboljših nastavitvah je nad dolžino 1500 znakov zelo visoka, celo do 100% in je večinoma višja, kot jo je imel Sašo Kodrič v svoji diplomski nalogi [3], ki sem jo vzel za primerjavo. Ko pa so besedila krajša, uspešnost pričakovano pade nekje do 88%, kar je zelo blizu njegovi. Pri še krajših besedilih pa bi bila verjetno njegova boljša.

Hipotezo, ki sem si jo zastavil, sem potrdil, saj program večinoma v pol minute uspešno razbije zašifrirano sporočilo.

Viri

[1] Wikipedija, »Zamenjalna šifra«, 2021 Na povezavi: https://sl.wikipedia.org/wiki/Zamenjalna_%C5%A1ifra

[2] Wikipedija, »Substitution cipher«, 2022 Na povezavi: https://en.wikipedia.org/wiki/Substitution_cipher

[3] S. Kodrič, »Orodja za razbijanje substitucijske šifre«, 2012 Tudi na povezavi: http://eprints.fri.uni-lj.si/1976/1/Kodri%C4%8D_S-1.pdf

[4] MediaWiki, »Caesar / Substitution Cipher«, 2019 Na povezavi: https://www.trccompsci.online/mediawiki/index.php/Caesar_/_Substitution_Cipher

[5] Digitalna knjižnica Slovenije, Na povezavi: https://www.dlib.si

Kazalo slik