UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Sašo Kodrič

Orodja za razbijanje substitucijske šifre

DIPLOMSKO DELO

VIŠJEŠOLSKI STROKOVNI ŠTUDIJ PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: prof. dr. Aleksandar Jurišić

Ljubljana, 2013

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00356/2012
Datum: 08.11.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **SAŠO KODRIČ**

Naslov: **ORODJA ZA RAZBIJANJE SUBSTITUCIJSKE ŠIFRE**
TOOLS FOR BREAKING A SUBSTITUTION CIPHER

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Delo naj predstavi matematične osnove najpomembnejših metod za razbijanje substitucijskih šifer, npr. na podlag statistike parov črk. Program bo naložil v pomnilnik dan tajnopis (šifrirano besedilo) in obsežnejši slovar. Iz njiju izračuna matriki frekvenc parov. Uporabi različne metode za izračun razdalje med dobljenima matrikama (Evklidska, Mathattnova in maksimalna razdalja) ter testira različne oblike reševanja problema. Glavni cilji so (a) odšifriranje poljubnega tajnopisa na podlagi pogostosti pojavljanja parov črk, (b) grafični vmesnik za pomoč pri kriptoanalizi in (c) podrobna analiza uspešnosti odšifriranja besedila.

Mentor:

prof. dr. Aleksandar Jurišić

Dekan:

prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Sašo Kodrič, z vpisno številko **63050147**, sem avtor diplomskega dela z naslovom:

Orodja za razbijanje substitucijske šifre

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Aleksandra Jurišića

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ Podpis avtorja:_____

Kazalo

Povzetek

Digitalna komunikacija postaja vse pomembnejši del modernega načina življenja. V nekaterih primerih je zaželeno, da ta komunikacija poteka zakrito, v nekaterih jo je potrebno razkriti. Poenostavljeno je kriptografija veda, ki preučuje tehnike za zakrivanje vsebine sporočila s šifriranjem in razkrivanje z dešifriranjem. Naše delo se posveti kriptoanalizi, bolj konkretno predstavi matematične osnove za razbijanje substitucijskih šifer na podlagi statistike parov črk. Program besedilo šifrira z naključno substitucijsko šifro. Izračuna dve matriki frekvenc parov na podlagi korpusa in šifriranega besedila. Sledi izvajanje permutacij tajne abecede in ponovno računanje razdalje med matrikama tajnopisa in korpusa. Postopek se ponavlja, dokler se njuna razdalja zmanjšuje. Kot rezultat dobimo ključ, s katerim je bilo besedilo zašifrirano.

Ključne besede:

kriptografija, kriptoanaliza, substitucijska šifra

Abstract

Digital communication is becoming an essential part of modern way of life. In some cases it is desired, that message stays secret, while in some other cases it is required to be revealed. Simplified, cryptography is a theory that studies techniques to enable secret communication with the use of encryption and the use of decryption to obtain back the original message. This thesis deals with cryptoanalysis, in particular, it explains mathematical background for breaking the substitution cipher based on statistics of pairs of letters. It presents our implementation of algorithms, which encrypt a clear text with the use of random substitution cipher, calculate the frequencies of pairs of letters in corpus and encrypted text and stores them in two matrices. It permutes the substitution cipher in matrix of the encrypted text and calculates the distance with corresponding matrix of corpus. The algorithm is running, until the distance is decreasing. As result we get substitution cipher, which was used to encrypt the clear text.

Key words:

cryptography, cryptoanalysis, substitution cipher

1. Uvod

Živimo v časih, ko postaja varnost vse bolj pomembna. Ob vsakodnevni komunikaciji preko nezaščitenih kanalov lahko za zakrivanje pomena vsebine sporočila uporabljamo različne načine šifriranja. Kadar nam je na voljo ključ, z njim sporočilo odšifriramo. Včasih pa je vsebino sporočila potrebno ugotoviti brez ključa. Takrat rečemo, da je šifro, s katero je bilo šifrirano sporočilo, potrebno razbiti.

Običajno se vsebino sporočila, ki je šifrirano s substitucijsko šifro, želi odkriti z zbiranjem podatkov o pogostosti pojavljanja črk v besedilu. Če je besedilo zadosti dolgo, lahko na tak način ugotovimo permutacijo uporabljenega ključa. Na voljo so namreč natančni statistični podatki za porazdelitev črk v besedilu. Šifro pa se lahko razbije tudi z upoštevanjem pogostosti pojavljanja parov črk, saj tako uporabimo še dodatne informacije o njihovi medsebojni povezanosti. Na tak način se razbijanja substitucijske šifre lotimo tudi v programu, ki ga opisujemo v tej diplomski nalogi.

V uvodnem poglavju se spoznamo s teoretičnimi osnovami uporabljenega znanja in pripomočkom za delo. Nato postavimo formalno definicijo problema, opišemo slovar in testno besedilo ter razložimo shemo postopka reševanja. Nadaljujemo s predstavitvijo glavnih gradnikov našega programa, razredov, ki ga sestavljajo, zraven pa je tudi natančen opis delovanja algoritma reševanja. Sledi opis različnih konfiguracij in načinov testiranja. Ob pomoči grafičnega prikaza interpretiramo in analiziramo rezultate. V sklepnem poglavju opišemo potek dela in pomembnejše ugotovitve.

2. Osnove

2.1. Kriptografija

Komunikacija nas povezuje, nam pomaga sodelovati, vendar nas tudi razdvaja. Dandanes se mnogo informacij hrani v digitalni obliki na računalnikih. Pomnilni mediji kot so trdi disk, SD kartica, USB ključ, itd., nam omogočajo številne prednosti, saj so kompaktni, hitro dostopni, hkrati pa omogočajo organiziran dostop do raznovrstnih podatkovnih struktur. Z razvojem telekomunikacij, računalniških omrežij in obdelovanjem informacij pa je postalo tudi precej lažje prestreči in modificirati digitalno informacijo, kot je bilo to možno pri njenem papirnatem predhodniku. Naloga varnosti računalniških sistemov in omrežij postaja vse bolj pomembna, saj živimo v dobi, ko so naše informacije vedno bolj izpostavljene internetnemu kriminalu. Med preventivnimi ukrepi, ki so na voljo danes, nudi največjo stopnjo varnosti kriptografija. To je veda o komunikaciji v prisotnosti aktivnega napadalca.

Osnovni namen kriptografije je omogočiti sporočevalcu in naslovniku, da se sporazumevata preko nezaščitenega kanala, tako da nasprotnik, ki želi izvedeti vsebino njunega pogovora, tega ne more razumeti. Ta zveza je lahko na primer telefonska linija ali računalniška mreža. Sporočilo, ki ga želi sporočevalec posredovati naslovniku, imenujemo čistopis in je lahko besedilo, številski podatki ali karkoli v poljubni strukturi. Sporočevalec s pomočjo vnaprej določenega ključa čistopis zašifrira in dobljeni tajnopis pošlje po kanalu. Nasprotnik, ki s pomočjo prisluškovanja vidi tajnopis, ne more določiti čistopisa, medtem ko naslovnik, ki pozna šifrirni ključ, lahko dešifrira tajnopis in rekonstruira čistopis. To lahko opišemo bolj formalno v matematičnem jeziku [5].

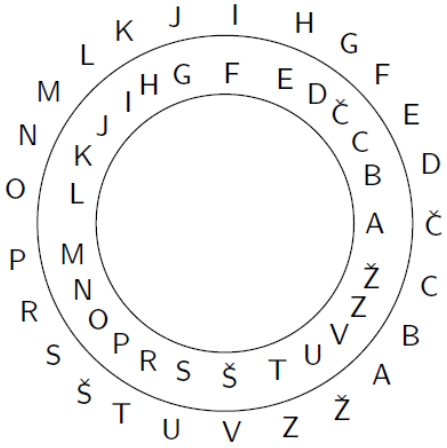
Simetrični kriptosistem je peterica $(\mathrm{P}, \mathrm{C}, \mathrm{K}, \mathcal{E}, \mathrm{D})$, za katero velja:

1. P je končna množica možnih čistopisov
2. C je končna množica možnih tajnopisov
3. K je končna množica možnih ključev
4. Za vsak ključ $\mathrm{K} \in \mathrm{K}$ se da učinkovito priti do šifrirnega postopka $e_{\mathrm{K}} \in \mathcal{E}$ in dešifrirnega postopka $d_{\mathrm{K}} \in \mathrm{D}$. Vsaka $e_{\mathrm{K}} : \mathrm{P} \to \mathrm{C}$ in $d_{\mathrm{K}}: \mathrm{C} \to \mathrm{P}$ sta taki funkciji, da velja $d_{\mathrm{K}}(e_{\mathrm{K}}(x)) = x$ za vsak čistopis $x \in \mathrm{P}$

Najpomembnejša je četrta lastnost. Pove nam, da če čistopis zašifriramo in dobljeni tajnopis dešifriramo, dobimo nazaj začetni čistopis [8].

Omenimo nekaj najbolj enostavnih kriptosistemov, ki pri šifriranju ohranijo vrstni red črk, kot je v čistopisu, črke same pa nadomestijo z drugimi črkami. Pri tem uporabljajo različne metode, ki določajo, katera črka se zamenja s katero. Govorimo o *substitucijskih* šifrah.

V *zamični* šifri se posamezna črka nadomesti z drugo črko, ki je določeno število mest naprej v abecedi. Velikost zamika je ključ šifre. Prvi je to obliko šifriranja opisal Julij Cezar v svojem delu *Galska vojna* (sam je uporabljal zamik za tri mesta).
Primer: "Cezar" → "Ehbčt"



Slika 1: Zamična šifra slovenske abecede s ključem 3

Afina šifra vsaki črki glede na njeno zaporedno mesto v abecedi priredi njen številčni ekvivalent. Njegovemu večkratniku prištejemo zamik in to vsoto delimo s številom vseh črk abecede.

$$e(x) = ax + b \mod 25, \text{ za neka izbrana } a, b \in \mathbb{Z}_{25}$$

Dobljeni ostanek določa s katero črko se nadomesti posamezna črka. Za $a = 1$ dobimo zamično šifro. Funkcija je injektivna, če in samo, če je $D(a, 25) = 1$.

Zgoraj omenjeni šifri pa sta pravzaprav posebna primera *substitucijske* šifre. Zanjo je značilno, da njen ključ določa naključna permutacija vseh črk abecede. Takšno šifriranje uporabljamo, analiziramo in razbijamo tudi v našem programu. Podrobnejši opis delovanja sledi v 4.1.2. podpoglavju.
Primer naključne permutacije:

A	B	C	Č	D	E	F	G	H	I	J	K	L	M	N	O	P	R	S	Š	T	U	V	Z	Ž
D	L	R	J	V	O	H	E	Z	Ž	Š	P	T	B	G	F	Č	N	M	U	S	K	A	C	I

Do sedaj omenjene *enostavne* substitucijske šifre temeljijo na uporabi posameznih črk, ki v čistopisu vedno zamenjajo isto črko, zato rečemo, da so *enoabecedne*. V kolikor izvedemo zamenjavo nad večjim številom črk (par, trojica, kombinacije, itn.) se imenujejo *večgrafske* substitucijske šifre. Če pa čistopis na različnih mestih šifriramo z več kot eno samo substitucijsko šifro, govorimo o *večabecedni* substitucijski šifri (npr. vsako 5. črko šifriramo z

eno substitucijsko šifro, črke, ki neposredno sledijo tem pa z drugo). Pri tovrstnem šifriranju so pogostosti črk v tajnopisu precej izenačene.

Iskanje novih izboljšav obstoječih kriptografskih mehanizmov in dokazov o varnosti se nadaljuje z veliko hitrostjo. Novi varnostni izdelki se ves čas razvijajo, da bi zadostili potrebo po varnosti informacijske družbe [7].

Omenimo nekaj osnovnih principov za študij varnosti nekega kriptosistema:

- Računska varnost
- Dokazljiva varnost
- Brezpogojna varnost

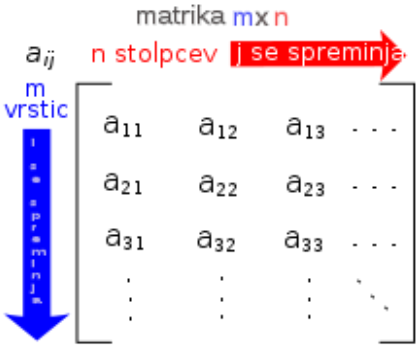
Kriptosistem je *računsko varen*, če tudi najboljši algoritem za njegovo razbitje potrebuje vsaj N operacij, kjer je N neko konkretno zelo veliko število. *Dokazljivo vare*n je, če lahko dokažemo, da se njegova varnost zreducira na varnost kriptosistema, ki je zasnovan na dobro preštudiranem problemu. Ne gre torej za absolutno varnost, temveč *relativno varnost*. Kriptosistem je *brezpogojno varen,* kadar ga napadalec ne more razbiti, tudi če ima na voljo neomejeno računsko moč [5].

Od zgoraj omenjenih kriptosistemov je najmanj varna zamična šifra. Omogoča le toliko možnih ključev kot je črk v abecedi (v slovenščini 25). Pri afini šifri število možnih ključev izračunamo tako, da število vseh števk, ki so manjše in tuje številu velikosti abecede, pomnožimo s številom velikosti abecede. Za slovenščino (20×25) omogoča 500 možnih ključev. Še bolj varna je substitucijska šifra, saj omogoča $25!$ možnih ključev. To je število večje kot 1.55×10^{25}. Zato postane iskanje s preverjanjem vseh možnih ključev ali z grobo silo (angl. *brute-force*) neuporabno. Lahko pa si pomagamo z drugimi metodami, ki jih bomo opisali v tej diplomski nalogi.

2.2. Matrike in metrike

Ob pojavljanju različnih oblik podatkov se izrazi potreba za njihov ustrezno formatiran zapis. Ker bomo reševali problem, ki vključuje pare črk, bomo za zapis frekvenc pojavljanja parov uporabili matrike. Razsežnosti sta enaki številu uporabljenih znakov. Matrike se uporabljajo na veliko različnih področjih, npr. v matematiki, statistiki, grafiki, računalništvu, gradbeništvu, itd. Uporabili smo jih tudi zato, ker je med njimi možno računati razdaljo.

Matrika velikosti $m \times n$ je pravokotna tabela mn števil, ki so razporejena v m vrstic in n stolpcev.



Slika 2: Matrika m vrstic in n stolpcev z elementi a_{ij} [10]

Matrike običajno označujemo z velikimi tiskanimi črkami $(A, B, C, \dots)$. Številom v matriki pravimo *elementi matrike*. Element a_{ij} se nahaja v i-ti vrstici in j-tem stolpcu matrike [9]. V naši diplomski nalogi bodo elementi matrike realna števila, ki predstavljajo pogostosti pojavljanja parov črk v besedilu. Prva črka para bo v matriki pogostosti pojavljanja parov črk predstavljala vrstico, druga pa stolpec. Stolpce ali vrstice v matriki smo med seboj tudi zamenjevali.

Permutacija pomeni z medsebojnimi zamenjavami preurejeno zaporedje končnega števila elementov (pri tem število elementov ostane enako). Ker lahko permutacija pomeni večje število zamenjav, v diplomskem delu pa smo uporabljali predvsem zamenjavo dveh stolpcev med seboj ali dveh vrstic med seboj, bomo rekli, da gre pri takšni zamenjavi za *transpozicijo*. To je posebna oblika permutacije, ki zamenja mesti samo dvema elementoma iz množice (v našem primeru samo dvema stolpcema ali samo dvema vrsticama). Ostali elementi ostanejo na svojih mestih.

Metrični prostor je neprazna množica M, opremljena s preslikavo $d: M \times M \to [0, \infty)$, ki ima naslednje lastnosti za vse $x, y, z \in M$:

- $d(x,y) \geq 0 \ \ in \ \ d(x,y) = 0$, natanko tedaj, ko je $x = y$ (nenegativnost)
- $d(x,y) = d(y,x)$ (simetrija)
- $d(x,y) \leq d(x,z) + \ d(z,y)$ (trikotniška neenakost)

Preslikavi d pravimo *metrika* ali *razdalja* [6]. V našem programu razdaljo računamo pri primerjanju matrik pogostosti pojavljanja parov črk. Iz matrike pravzaprav naredimo vektor. To storimo tako, da stolpce (lahko pa tudi vrstice) iz matrike zložimo enega za drugega in tako naredimo dolgo kačo, ki je v resnici vektor. Uporabimo dve različni razdalji, v enačbah katerih vzamemo, da je: $N = nm$.

Evklidska razdalja:

$$d(x,y) = \ ||\, x - y|| \ = \sqrt{\sum_{i=1}^{N} \left(x_i - y_i\right)^2}$$

Manhattanova razdalja:

$$d(x,y) = \ ||x - y|| = \sum_{i=1}^{N} |x_i - y_i|$$

2.3. Korpusno jezikoslovje

Korpus je obsežna zbirka besedil. Je jezik v resnični in sodobni podobi v elektronski obliki. Predstavlja reprezentativen vzorec za jezik, ki naj bi ga preučevali. Služi za opisovanje in raziskovanje jezika. Iz njega se izdela jezikovne vire. Uporablja se za izdelavo slovnice in drugih opisov jezikovnih struktur, lahko pa tudi kot slovar. Iz njega se razvijajo pripomočki za prevajanje in učenje jezika. Potrebujemo jih za raziskovanje vseh oblik jezikovnega vedenja. Te vključujejo pogostosti uporabe posameznih besed, opazovanje besed skupaj s sobesedilom in zanimivih sopojavitev besed. Na spletu obstaja mnogo profesionalnih korpusov, lahko pa ga z zbiranjem besedil naredimo tudi sami.

Postopek:

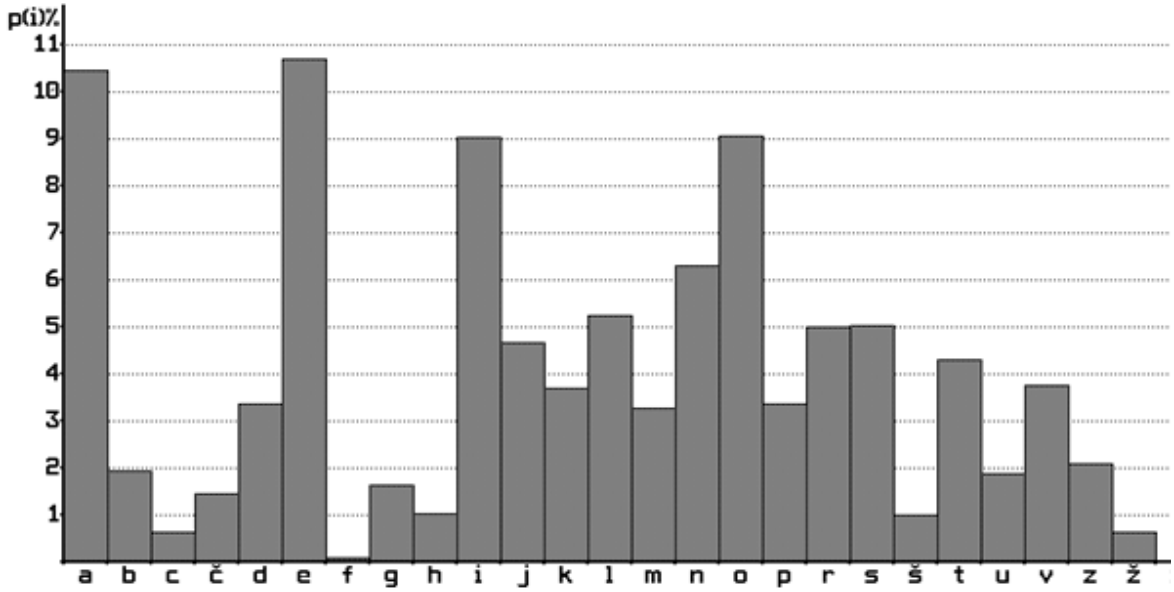
1. Izbira besedil: reprezentativnost, uravnoteženost, izvedljivost
2. Digitalni zajem: OCR, Word, HTML
3. Normalizacija besedil: enovit format
4. Označevanje: oblikoslovne oznake, osnovne oblike in skladnje
5. Distribucija: avtorske pravice, platforma

Glavna kriterija pri izbiri besedil sta dva. Prvi je reprezentativnost, ki pomeni, da korpus zajema "vse" besedilne zvrsti različnih starosti. Drugi je uravnoteženost, ki zahteva, da so velikosti vzorcev besedilnih zvrsti v sorazmerju z njihovo "pomembnostjo" za govorce jezika. Da se v praksi zadosti kriterijema, je potrebna metodičnost in razmislek pri odločitvah, katera besedila se v korpus vključijo.

Kakovosten korpus mora biti čim večji. Nacionalni korpusi so navadno veliki več 100 milijonov besed (slovenska korpusa sta FidaPLUS in Nova Beseda). Pomembno je, da je pravilno zapisan in označen. Mora biti enostaven in njegov računalniški zapis razumljiv. Korpusi morajo biti na računalniku zapisani tako, da bodo uporabni za uporabnike z različno računalniško/programsko opremo in za različne namene. Standardizacija zapisa pomeni večjo trajnost, uporabnost, razumljivost in preverljivost. Današnji standard pri zapisih korpusov je standard XML. Vsa besedila morajo biti shranjena v enakem kodnem naboru. Dober korpus je dokumentiran z bibliografskimi in drugimi podatki.

Tradicionalno je bilo zbiranje besedil za korpus dolgotrajen in drag proces, danes pa na spletu najdemo ogromno besedil iz raznovrstnih področij. Ob pravilnem uravnoteženju in reprezentativnosti je splet lahko uporaben kot vir za izgradnjo korpusov. Z avtomatskimi metodami selekcije, zajema in poenotenja formata medmrežnih strani korpusi dosegajo tudi milijardo besed. Pri zelo velikih vzorcih ponovno nastopijo omejitve, saj računalniške zmogljivosti otežujejo delo s takšno količino besed [2].

V okviru korpusnega jezikoslovja je rezultate preštevanja frekvenc črk za reprezentativni vzorec leposlovnih besedil objavil dr. Primož Jakopin v svoji doktorski disertaciji *Zgornja meja entropije pri leposlovnih besedilih v slovenskem jeziku*. Frekvenca črk je lastnost besedila, ki se pogosto uporablja pri kriptografiji. Jezik se spreminja, pa tudi vsak avtor piše nekoliko drugače, zato je običajno mogoča le statistična analiza. Odvisna je torej od jezika besedila, avtorja, opisane teme pa tudi časovnega obdobja. Frekvenca črk, dvojčkov, trojčkov in *n*-teric črk ali znakov lahko pokaže na značilnosti besedila ali ovrže avtorstvo neznanega besedila, ki pa mora biti dovolj dolgo. Frekvence črk pa so pomembne tudi v prenosni tehniki, ko skušamo zakodirati sporočilo tako, da zavzame čim manj prostora pri prenosu in tudi shranjevanju. Značilen primer je Morsejev kod, kjer se E kodira z enim znakom [4].



Slika 3: Frekvence črk v slovenskem jeziku

Na sliki 3 se vidi, da prevladujejo samoglasniki. Med najmanj pogostimi so šumniki. Najbolj pogoste črke so: { [E, (10,71%)], [A, (10,47%)], [O, (9,08%)], [I, (9,04)]}. Najmanj pogoste črke: { [F, (0,11%)], [Ž, (0,65)], [C, (0,66%)], [Š (1%)]. Sicer je najpogostejši znak presledek (17,03%).

2.4. Razvojno okolje

Za razvoj orodij smo uporabili okolje Eclipse Indigo, ki nam olajša izdelavo programske opreme. Orodje je izdano pod odprtokodno licenco Eclipse Public License. Primarno podpira razvoj za programski jezik Java, z raznimi dodatki, ki jih je možno dodatno namestiti, pa Eclipse podpira tudi Ada, C, C++, COBOL, Haskell, Perl, PHP, Python, R, Ruby, Scala, Clojure, Groovy in Scheme. Možno ga je namestiti na veliko različnih operacijskih sistemov, kar je še ena prednost pred ostalimi razvojnimi orodji [11].

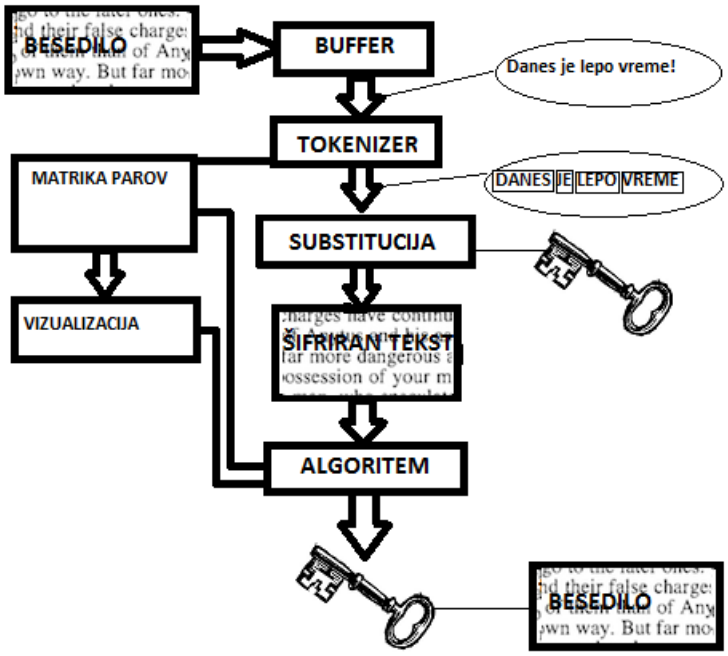
3. Priprava

Najprej se seznanimo s problemom. Za njegovo rešitev je bilo potrebno razširiti znanja s področij, opisanih v uvodnem poglavju. Teoretični podlagi je sledilo zbiranje besedil za sestavo slovarja, ki v programu predstavlja vhodno datoteko. Nato se je pričel razvoj posameznih razredov, potrebnih za reševanje problema v istem vrstnem redu, kot je to opisano v naslednjem poglavju.

3.1. Definicija problema

Predpostavimo, da imamo dano besedilo v nekem jeziku, za katerega vemo, da je šifrirano s substitucijsko šifro. Potrebno je ugotoviti s katero permutacijo substitucijske šifre je bilo dano besedilo šifrirano. Povezave med pari črk nudijo več informacij kot samo statistika pojavljanja posamezne črke. Z analizo porazdelitve parov se lotimo razbijanja šifre. Problem rešujemo z zmanjševanjem razdalje med dvema matrikama pogostosti pojavljanja parov črk.

3.2. Shema celotnega postopka



Slika 4: Shema postopka

Besedilo slovarja se v obliki tekstovne datoteke naloži v razred Buffer. Tokenizer besedilo razbije na posamezne besede, odstrani posebne znake (ločila, tuje črke, simboli,..) in jih shrani v seznam besed. Nato se izvede izbira podmnožice besedila, ki se jo vzame za testno besedilo. Nad celotnim seznamom besed testnega besedila se izvede Substitucija, s katero dobimo šifriran tekst. Nad tem Matrika parov izvede izračun matrike frekvenc parov črk. Algoritem nato izvaja transpozicije stolpcev in transpozicije pripadajočih vrstic med matrikama frekvenc slovarja in šifriranega besedila, dokler se njuna razdalja ne neha zmanjševati. Kot rezultat dobimo ključ, to je permutacijo, ki je bila uporabljena za šifriranje originalnega besedila. V kolikor je ključ enak tistemu, s katerim smo besedilo zašifrirali, smo uspešno razbili substitucijsko šifro.

3.3. Slovar in testno besedilo

Program za vhod dobi tekstovni datoteki, slovar in testno besedilo, ki vsebujeta različni količini teksta. Večji vzorec, slovar, šteje 3 milijone besed, ki sestavljajo različne tipe besedil, od starejšega in novejšega leposlovja, aktualnih zapisov na portalih z novicami, znanstvenih člankov, zapisov s spletnih dnevnikov slenga, komentarjev in poezije. Slovar je v programu pravzaprav velika zbirka besed in ne abecedna ureditev razloženih gesel.

Potrebno je bilo zbrati dovolj besedila, saj smo za potrebe slovarja potrebovali velik vzorec, da bi bilo moč izdelati matriko parov, ki bi dobro odražala lastnosti slovenskega oz. angleškega in drugih jezikov. Sprva je slovar nastal iz prosto dostopnih besedil, v največji meri iz leposlovja [13], spletnih novic, besedil seminarskih nalog in blogov. Z raziskovanjem sem ugotovil, da tovrstne obsežne zbirke besedil že obstajajo in da so besedila v njih vključena na podlagi uravnoteženosti in reprezentativnosti, dveh kriterijev, ki jim vsak korpus mora ustrezati. Kasneje smo pridobili pravico do uporabe dveh profesionalnih korpusov.

Za dopolnitev slovarja je dr. Tomaž Erjavec z odseka za inteligentne sisteme na Inštitutu Jožefa Stefana omogočil dostop do obsežnega referenčnega korpusa ccKRES [3]. Iz tega obsežnega besedila v slovenskem jeziku smo pridobili referenčno matriko frekvenc pojavljanja parov. Za angleški slovar je uporabljen korpus Corpora, ki je prosto dostopen na straneh Lancastarske univerze [1]. Korpus obsega 2 milijona besed različnih žanrov iz različnih obdobij in je uravnotežen ter reprezentativen.

Manjše, testno besedilo je naključno izbrano besedilo. Nad njim se izvede substitucijska šifra. Testno besedilo se lahko izbere tako, da se izbere naključen del slovarja, ali pa da se ročno nastavi na besedilo poljubne velikosti in vsebine. Pri začetnem obdobju testiranja je bilo potrebno ročno vnašati različne velikosti testnih besedil, nato pa se je začela uporabljati avtomatska možnost, da iz celotnega slovarja algoritem naključno izbere podmnožico besed, ki jo nato uporabimo kot testno besedilo. Velikost testnega besedila se spreminja glede na nastavitve testa.

4. Program

Vhodna datoteka, ki vsebuje velike količine besedila, se s pomočjo razreda *Buffer* naloži v bralno-pisalni pomnilnik. Razred *Tokenizer* besedilo pretvori v posamezne besede in odstrani odvečne znake. V takšni obliki lahko z razredom *Matrika parov* iz seznama besed izračunamo pogostosti pojavljanja parov črk, frekvenc, ki jih shranimo v matriko. Izračuna se dve matriki frekvenc. Prva je na podlagi slovarja in ji rečemo referenčna matrika frekvenc. Druga nastane na podlagi testnega besedila in ji rečemo matrika frekvenc besedila. Razred *Substitucija* izvede šifriranje besedila z naključno substitucijsko šifro. Z *Vizualizacijo* lahko izdelamo slike, ki prikazujejo porazdelitev frekvenc v matrikah. Z *algoritmi reševanja* nato izvajamo transpozicije stolpcev in pripadajočih vrstic nad matrikami, ki zmanjšujejo razdaljo med referenčno in matriko frekvenc besedila. Ko se razdalja več ne zmanjšuje, pridemo do končne rešitve, katere uspešnost tudi vrednotimo.

4.1. Obdelava besedila

4.1.1. Preoblikovanje

Razred Buffer prebere celotno datoteko z besedilom in njeno vsebino zapiše v objekt tipa *StringBuffer*. Na ta način se vsebina shrani v bralno-pisalni pomnilnik.

Razred Tokenizer poskrbi, da se vsebina razdeli na posamezne besede. Besede nato nastavi na same velike črke in jim, če je to potrebno, doda presledek pred prvo in za zadnjo črko. S tako dodanima presledkoma prva in zadnja črka dobita svoj par. Če presledka ne upoštevamo, se ne dodaja. Če se to zgodi, je odvisno od ročne nastavitve. Algoritem reševanja izvajamo samo s črkami in tudi z upoštevanjem presledka. To je pomembno, ker upoštevanje presledka vpliva na rezultate razbijanja šifre. Tokenizer ostale znake, ki niso črka ali presledek, shrani v znak *, ki pa se ne uporablja. Večinoma gre za posebne znake, tuje črke, matematične simbole in ločila. Vse besede se shranijo v seznamu *List<String>*.

4.1.2. Substitucija

Za šifriranje se uporablja enoabecedna naključna substitucija, ki vsaki črki iz množice abecede črk substitucijske šifre bijektivno priredi naključno črko iz abecede. Glavna naloga razreda Substitucija je generiranje naključne substitucijske šifre, s katero se nato šifrira testno besedilo. To stori tako, da iz tabele *tabela,* v kateri so shranjene vse možne črke in presledek,

preko seznama znakov *lstTab* vse elemente preslika v novo tabelo *tabSub*. Postopek je opisan na sliki 5.

```
public void genSubstitucijo()
{
    tabSub = new char[tabela.length];
    List<Character> lstTab = new LinkedList<Character>();
    for(char c : tabela)
    {
        lstTab.add(c);
    }
    int mesto = 0;
    while(lstTab.size() > 0)
    {
        int index = (int)Math.floor(lstTab.size()*Math.random());
        tabSub[mesto] = lstTab.get(index);
        lstTab.remove(index);
        mesto++;
    }
}
```

Slika 5: Metoda *genSubstitucijo*

Takšna določitev ključa substitucijske šifre se izvede samo prvič. Ko se izvajajo testi s konfiguracijami, ki obsegajo milijone različnih testov in toliko različnih substitucijskih šifer, postane bolj praktično, če se novo substitucijsko šifro naredi na tak način, da se izvede naključna permutacija stare in si zapomni kakšna je.

Možna je tudi substitucija enega para črk, kar je bilo uporabno v zgodnejših fazah testiranja pri opazovanju sprememb pri matrikah frekvenc s pomočjo vizualizacije. Učinek take substitucije je viden na sliki 11. Seveda pa je dodan izpis trenutne substitucijske šifre za preverjanje vmesnih rezultatov.

4.3.3. Matrike frekvenc parov

Razred *Matrika parov* je zadolžen za več nalog. Izračuna pogostosti pojavljanja parov in jih shrani v matriko. Vrstice v matriki predstavljajo prvo in stolpci drugo črko v paru. Vrednosti posameznega para so normirane s skupno frekvenco, ki je seštevek vseh frekvenc parov. Tako dobimo matriko pogostosti pojavljanja parov, ki ji rečemo kar matrika frekvenc. Računamo jo tako za slovar, kot testno in šifrirano testno besedilo. Tisto, ki jo izračunamo za slovar, imenujemo referenčna matrika frekvenc in se uporablja za računanje razdalje z izbrano matriko frekvenc testnega besedila. Vse besede iz besedila razdeli na pare znakov. Pare se dobi na naslednji način:

BESEDA → BE, ES, SE, ED, DA in ob upoštevanju presledka: ' 'B, BE, ES, SE, ED, DA, A' '

Izračunane matrike se da tudi izpisati. Izpis je omejen na 2 decimalni mesti natančnosti. Opaziti je, da se pari, sestavljeni iz iste črke (npr. AA, BB, CC, itd.) skoraj ne pojavljajo.

Manj pa se pojavljajo tudi pari, ki vsebujejo črke Q, X, W in Y, kar je pričakovano, saj niso črke slovenske abecede. 20 najpogostejših parov črk z odstotkom pojavljanja v slovenščini:

{ [JE, 2,49%], [NA, 1,62%], [NI, 1,59%], [PO, 1,48%], [RA, 1,45%], [PR, 1,43%], [ST, 1,41%], [IN,1,32%], [RE,1,32%], [KO,1,31%], [SE, 1,27%], [NE, 1,26%], [AL, 1,25%], [LA, 1,22%], [IL, 1,15%], [EN, 1,13%], [LI, 1,11%], [NO,1,06%], [AN, 101%], [KA, 1,01%], [RI, 0,98%], [TI, 0,98%], [OV, 0,98%], [TE, 0,98%], [VE, 0,97%], [DA, 0,96%], [EL, 0,93%], [ZA, 0,89%], [JA, 0,86%] }

Opazimo, da par JE odstopa kot daleč najbolj pogost par črk v slovenščini. Gre za nedovršno obliko glagola biti, ki pomeni izraz materialne ali duhovne navzočnosti v stvarnosti. Pogosto mu sledi prislovno določilo s širokim pomenskim obsegom. Večinoma se uporablja kot pomožni glagol z opisnim ali trpnim deležnikom [12]. Naslednja lastnost parov, ki jo opazimo, je prisotnost samoglasnikov v 18 od 20 najbolj pogostih parih (izjemi sta PR in ST). Večinoma se pojavljajo na drugem mestu v paru, ker je lastnost večine jezikov, da soglasniku večinoma sledi samoglasnik.

Metode za pridobitev frekvence parov delujejo tako, da ob vsaki pojavitvi enega para njegovemu seštevku prištejejo 1. Za normalizacijo potem seštevke posameznih parov delijo z vsoto vseh seštevkov. Tako dobimo normalizirane frekvence posameznih parov.

```
public void posodobiFrekvence(List<String> lstTokens)
{
    resetMatrix();
    for(String beseda : lstTokens)
    {
        posodobiBesedo(beseda);
    }
    if(normaliziraj)
        normalizirajMatriko();
}

private void posodobiBesedo(String beseda)
{
    for(int i = 0; i < beseda.length() - 1 ; i++)
    {
        CharSequence par = beseda.subSequence(i, i+2);
        posodobiPar(par);
    }
}

private void posodobiPar(CharSequence par)
{
    int vrstica = getIndex(par.charAt(0));
    int stolpec = getIndex(par.charAt(1));
    frekvencaParov[vrstica][stolpec]++;
    if(Math.min(vrstica, stolpec) != 0)
        sestevek++;
}
```

Slika 6: Metode za pridobitev frekvence para

Nastala matrika frekvenc parov ni simetrična, saj prve in druge črke v paru ne moremo kar tako zamenjati. Pimer: [JE, 2,49%] je veliko bolj pogost kot [EJ, 0,33%].

Matrika parov izračuna tudi moč celotnega stolpca ali vrstice. Moč je vsota vseh frekvenc v stolpcu ali vrstici. Kot pomemben parameter pri testiranju uporabljamo število, od katerega je odvisen vrstni red, ki ga bomo vzeli za izvajanje transpozicij v matrikah. To število nam pove za koliko najmočnejših stolpcev se računa spremembo razdalje ob njihovih transpozicijah.

MaxN = število najmočnejših stolpcev

Poišče tudi *Npar* najbolj pogostih parov znotraj matrike frekvenc parov, kar je vidno pri grafičnem prikazu, kjer ob dodatni označitvi lahko opazimo poudarjene vzorce.

Npar = število najbolj pogostih parov

Razred vpelje tudi računanje evklidske in Manhattanove matematične razdalje. Razdalje se računajo tako, da se iz matrik primerja celico po celico.

```
public static double evklidskaRazdalja(double[][] razlika)
{
    double suma = 0;
    for(int i = 0; i < MAX_PAROV; i++)
    {
        for(int j = 0; j < MAX_PAROV; j++)
        {
            suma = suma + razlika[i][j] * razlika[i][j];
        }
    }
    return Math.sqrt(suma);
}

public static double manhattanRazdalja(double[][] razlika)
{
    double suma = 0;
    for(int i = 0; i < MAX_PAROV; i++)
    {
        for(int j = 0; j < MAX_PAROV; j++)
        {
            suma = suma + Math.abs(razlika [i][j]);
        }
    }
    return (suma);
}
```

Slika 7: Metodi za izračun razdalj

Med testiranji je potrebno izračunati mnogo matrik frekvenc parov. Zato ne ustvarjamo vsakič nove matrike, ampak se staro ponovno inicializira. Tako pohitrimo delovanje programa.

```
private void resetMatrix()
{
    sestevek = 0;
    for(int i = 0; i < MAX_PAROV; i++)
        for(int j = 0; j < MAX_PAROV; j++)
            frekvencaParov[i][j] = 0;
}
```

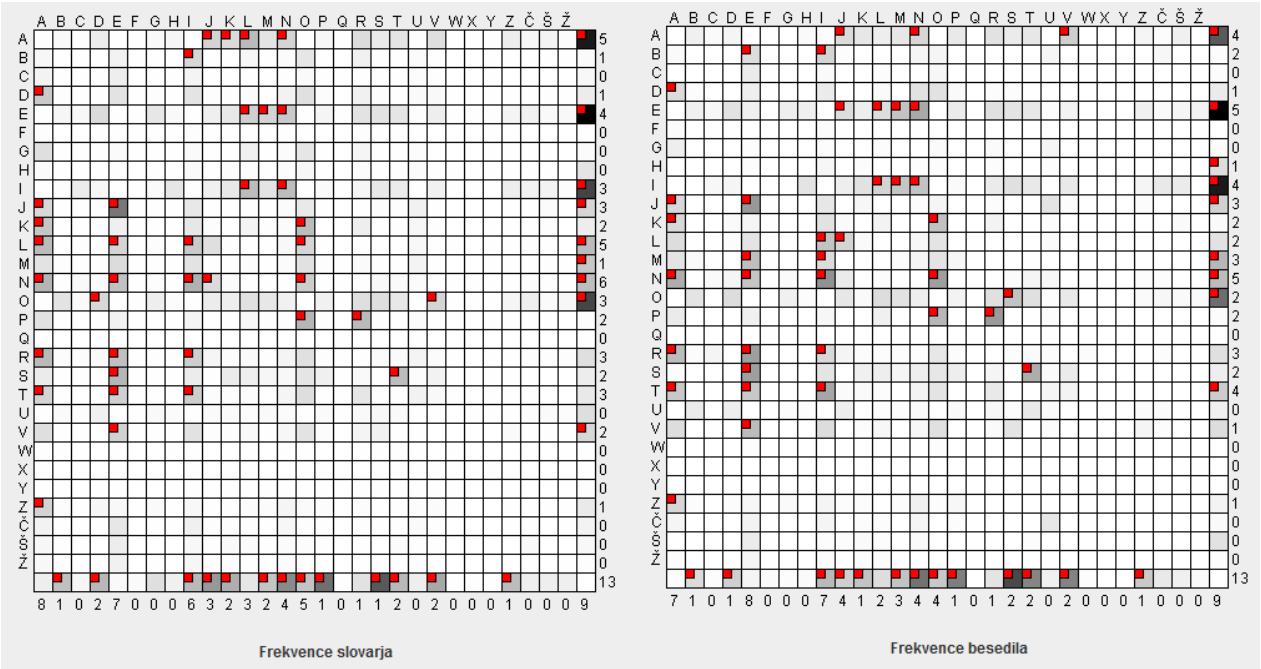
Slika 8: Metoda za *reset* matrike

4.2. Vizualizacija

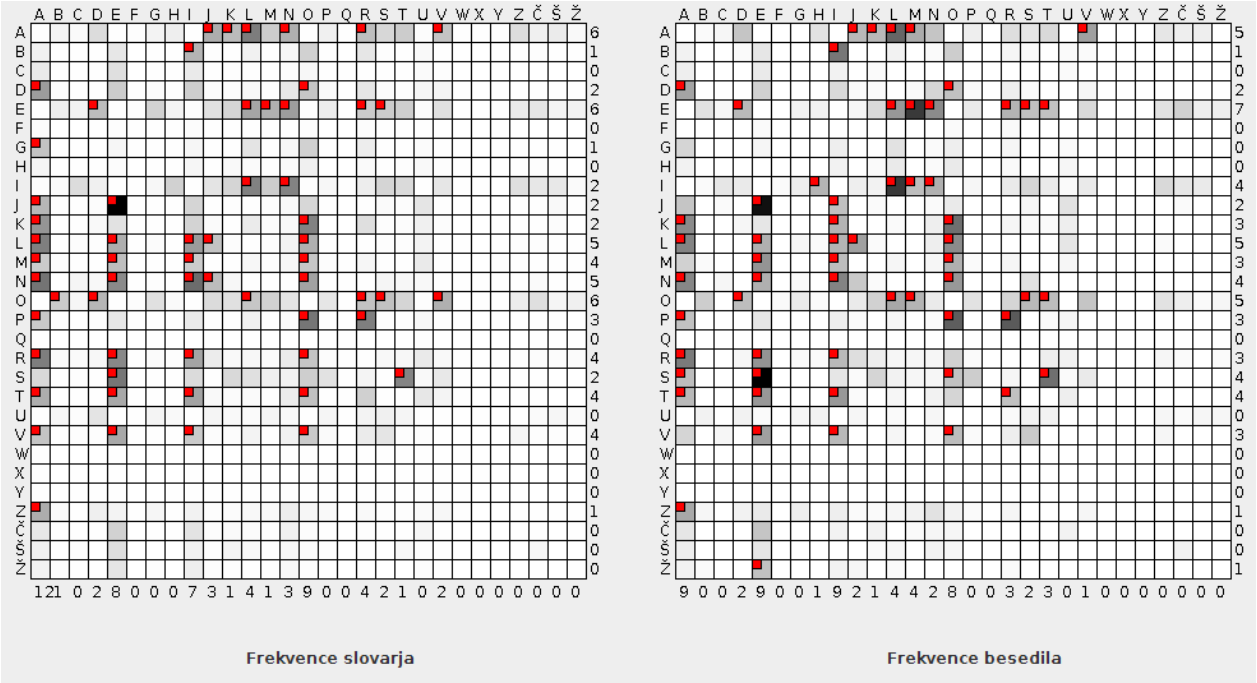
Razred Vizualizacija se uporablja za grafični prikaz matrik frekvenc parov in ne vpliva na sam program. Na spodnjih slikah so ob levi strani in na vrhu vrstic ter stolpcev matrik napisane črke tajne abecede. Ob vsaki vrstici na desni strani in na dnu vsakega stolpca je podano število najbolj pogostih parov *Npar* (ki smo ga vpeljali pri matrikah frekvenc parov). Njegovo velikost je potrebno ročno nastaviti v razredu in mora biti takšna, da se vidi večino najbolj pogostih parov. Razlike med frekvencami parov so tako majhne, da je priporočljiva velikost vsaj *Npar*=20, saj se jih toliko pojavlja v besedilu s pogostostjo vsaj 1%. Zgornja meja je *Npar*=70, saj ima toliko parov v besedilu vsaj 0,5% pogostost pojavljanja. Uporabno je za razvidnost vzorcev, ki poudarijo stolpce pri samoglasnikih in presledku. Frekvenca parov na grafičnem prikazu je izražena s stopnjevanjem barve.

- bela znotraj polja posameznega para pomeni zelo majhno frekvenco (povsem bela 0)
- sivina majhno frekvenco
- temnejša, skoraj črna, zelo veliko frekvenco (povsem črna ni nobena, za njo bi potrebovali vsoto vseh frekvenc)
- rdeči kvadratki v levi zgornji strani polja označujejo *Npar*=60

Predstavljene so matrike frekvenc parov slovenskega slovarja in besedila brez ter z upoštevanjem presledka. Substitucija enega para pomeni transpozicijo dveh stolpcev in transpozicijo pripadajočih dveh vrstic. Prikazana je tudi sprememba porazdelitev frekvenc parov v matriki besedila, ki se zgodi zaradi šifriranja.

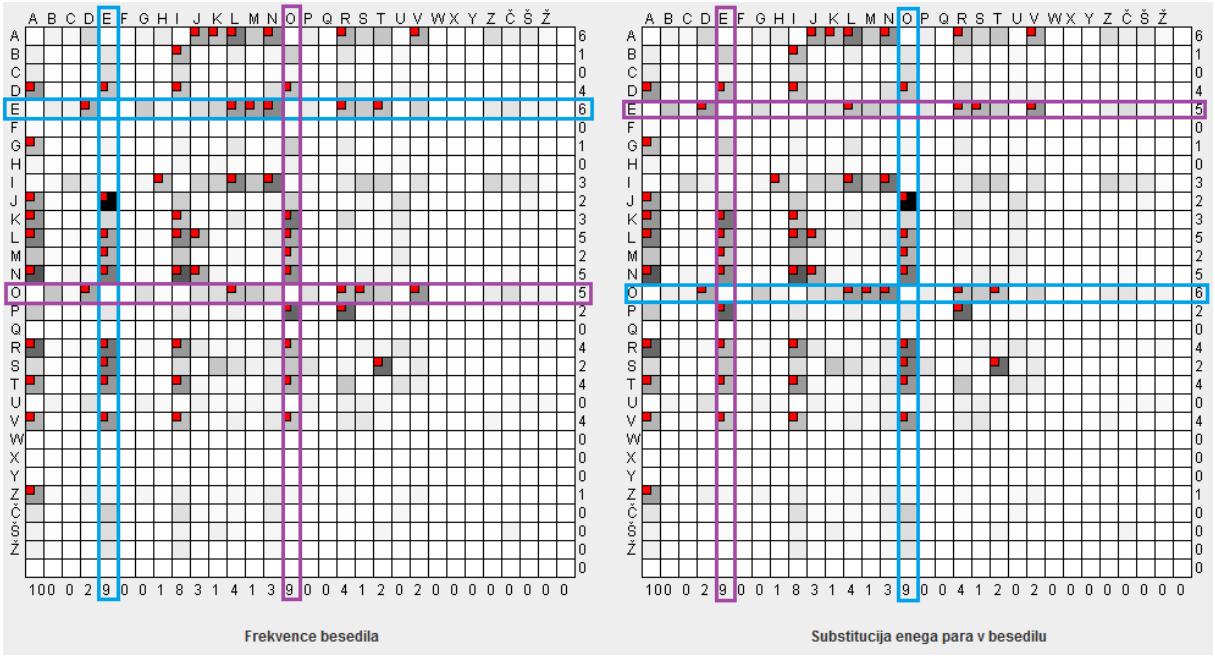


Slika 9: Frekvence slovenskega slovarja in besedila z upoštevanjem presledka



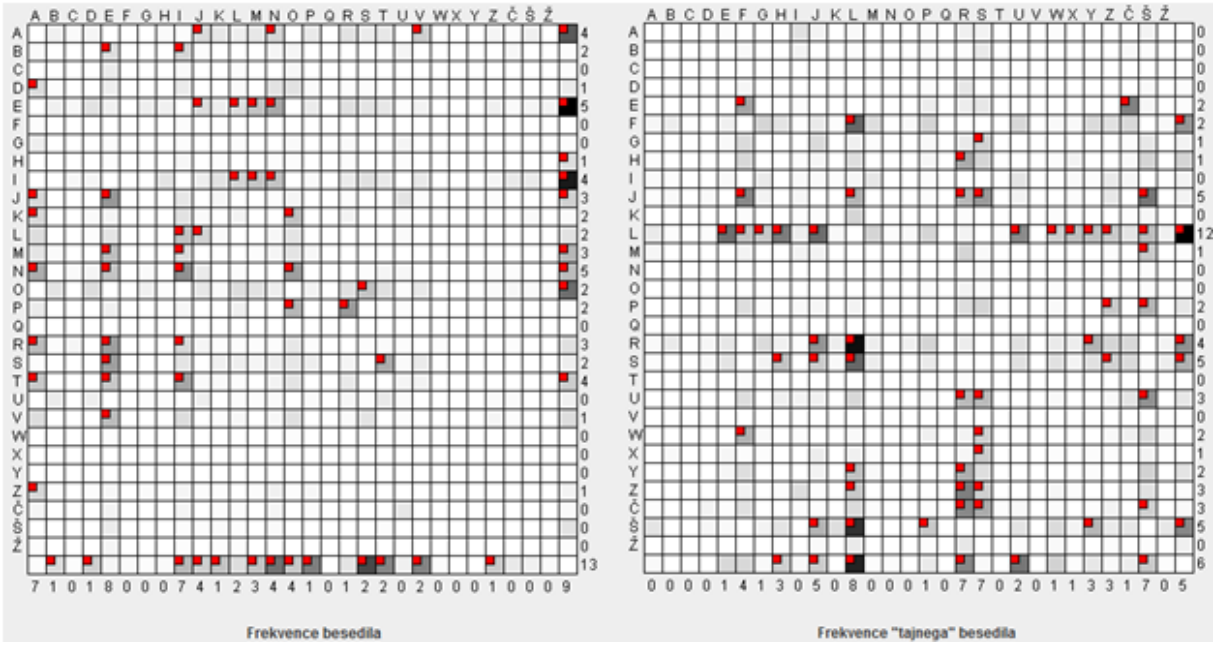
Slika 10: Frekvence slovenskega slovarja in besedila brez upoštevanja presledka

Na slikah 9 in 10 vidimo, da sta matriki frekvenc slovarja in besedila podobni, vendar nista popolnoma enaki (npr. odstopanje je na sliki 10 vidno pri stolpcih A, E, I, R, itd.). Matrika frekvenc besedila je nastala na podlagi 1000 slovenskih besed, matrika frekvenc slovarja pa na podlagi približno 3 milijonov besed. Razvidno je, da so stolpci (predstavljajo drugo črko v paru) samoglasnikov in presledka bolj poudarjeni.

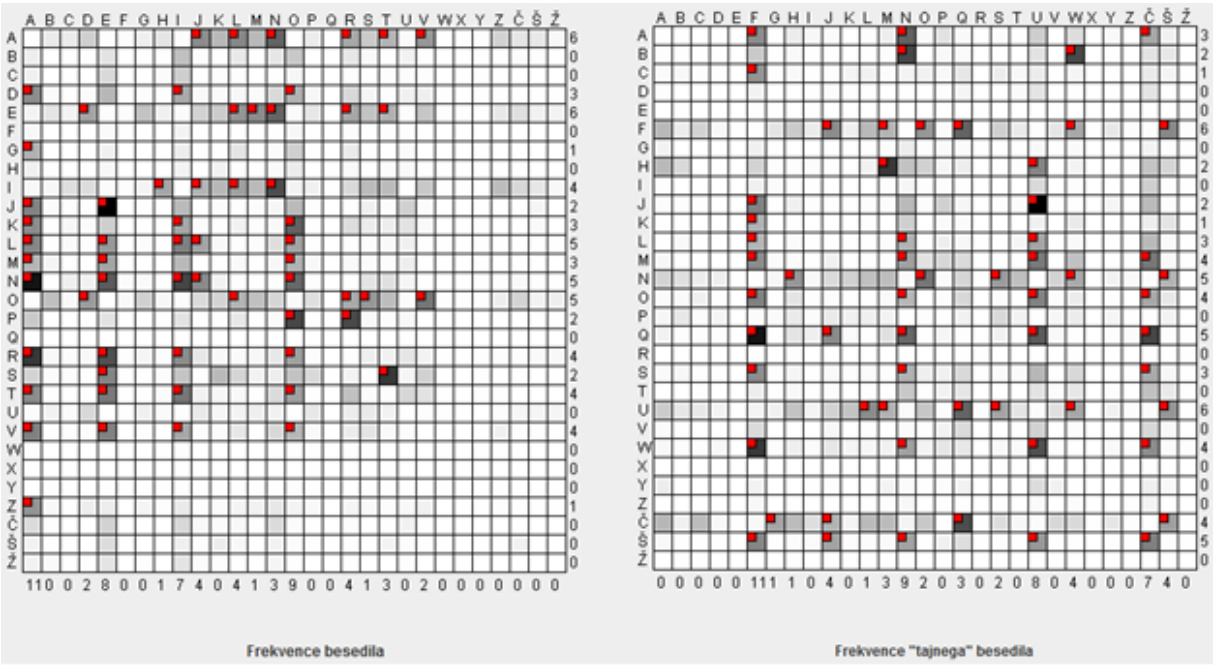


Slika 11: Substitucija enega para v besedilu

Na sliki 11 opazimo, da substitucija enega para črk povzroči zamenjavo dveh stolpcev in dveh pripadajočih vrstic (zamenjata se stolpca, ki pripadata O in E ter vrstici, ki pripadata O in E).



Slika 12: Frekvence besedila z upoštevanjem presledka po šifriranju



Slika 13: Frekvence besedila brez upoštevanja presledka po šifriranju

Na slikah 12 in 13 je prikazana sprememba porazdelitve frekvenc besedila po šifriranju. Leva stran slik prikazuje porazdelitev frekvenc v originalnem besedilu, desna pa porazdelitev frekvenc po šifriranju. Ta sprememba je seveda vsakič drugačna. Število *Npar* v posamezni vrstici ali stolpcu se lahko tudi spremeni, celotna moč vrstice ali stolpca pa se ohranja.

4.3. Algoritem reševanja

Že pri vizualizaciji se opazi, da so bolj kot vrstice poudarjeni stolpci. To se zgodi, ker je lastnost jezika, da soglasniku sledi samoglasnik in prav stolpci s samoglasniki in presledkom so najbolj izrazite. Zato se tudi v šifriranem besedilu opazijo izraziti stolpci. Odločimo se, da bomo izkoristili to lastnost in bo algoritem slonel na zaporedju transpozicij stolpcev ter vrstic v matriki frekvenc šifriranega besedila, ki bo odvisno od moči stolpcev v referenčni matriki (frekvence pridobljene na podlagi slovarja). Ob izvedbi ene transpozicije dveh stolpcev, se zraven nujno izvede tudi transpozicija pripadajočih vrstic (saj v paru črka lahko nastopa tako na prvem, kot na drugem mestu). Sledi kratek opis algoritma reševanja.

Opis iteracije reševanja:

Zapomnimo si vrstni red najmočnejših stolpcev v referenčni matriki ter zaporedno mesto, kjer se nahajajo (katero drugo črko v paru predstavljajo).

Primer: { [A, 1], [E, 5], [O, 15], [I, 9],.. itn.)

Nato poiščemo v matriki frekvenc šifriranega besedila *MaxN* najmočnejših stolpcev (predstavljajo naključno drugo črko para in so na naključnem mestu).

Primer: {[S, 19], [F, 4], [Š, 14], [Ž, 2],.. itn.}

Zamenjamo stolpec, ki je v matriki šifriranega besedila na 1. mestu s stolpcem S. Ker transpozicija dveh stolpcev nujno pomeni tudi transpozicijo dveh pripadajočih vrstic, se izvede še transpozicija vrstice črke, ki jo predstavlja stolpec na 1. mestu, in vrstice S. Izračunamo razdaljo med referenčno matriko in novonastalo matriko šifriranega besedila. Razdaljo si zapomnimo, vnovič pa izvedemo obe transpoziciji, tako da se matrika šifriranega besedila vrne v svoje prvotno stanje.

Sedaj zamenjamo stolpec, ki je v matriki šifriranega besedila na 1. mestu s stolpcem F. Izvede se še transpozicija pripadajočih vrstic. Znova izračunamo razdaljo med matrikama frekvenc in si jo zapomnimo. Matriko vrnemo v prvotno stanje.

To izvedemo za vsakega izmed preostalih *MaxN* stolpcev (če je *MaxN*=30, potem za vsakega izmed možnih stolpcev). Ko končamo, pogledamo kateri je najbolj zmanjšal razdaljo med matrikama. Izvedemo transpozicijo tega stolpca in stolpca na 1. mestu in transpozicijo njunih pripadajočih vrstic. Shranimo spremenjeno matriko šifriranega besedila in zapišemo transpozicijo v tabelo ključa, ki predstavlja našo rešitev.

Poiščemo novih *MaxN* stolpcev v matriki šifriranega besedila (tokrat brez tistega, ki je sedaj na 1. mestu). Izvedemo transpozicijo vsakega izmed njih s stolpcem na 5. mestu (zraven izvedemo tudi transpozicijo pripadajočih vrstic) in izračunamo razdaljo. Znova se odločimo

za tistega, ki je najbolj zmanjšal razdaljo med matrikama, in izvedemo transpozicijo tega stolpca in stolpca na 5. mestu ter transpozicijo njunih pripadajočih vrstic. Izvedeno zamenjavo si zapišemo v tabelo ključa.

Poiščemo novih *MaxN* stolpcev v matriki šifriranega besedila (brez tistih dveh, ki sta na 1. in 5. mestu). Izvedemo transpozicije s stolpcem na 15. mestu. Izberemo najboljšega in ga prestavimo na 15. mesto, izvede se še transpozicija pripadajočih vrstic. Zopet si zapomnimo transpozicijo in jo shranimo v tabelo ključa.

Nato postopek nadaljujemo, dokler ne pridemo čez vse stolpce referenčne matrike. Če matrika vsebuje 30 znakov, to pomeni 30 korakov (brez upoštevanja presledka jih je 29). Kadar je *MaxN* manjši od števila vseh stolpcev (če je 30, potem vedno izvaja transpozicije preostalih stolpcev), pri korakih, ki so manjši od *MaxN* izvedemo le toliko transpozicij, kolikor je še nepregledanih stolpcev. Ko nam preostaneta samo še dva stolpca, naredimo obe transpoziciji (ter zraven pripadajočih vrstic) in postavimo ugodnejšega na pravo mesto. Ko pridemo do zadnjega koraka, bi stolpec (ki predstavlja statistično najmanj pogosto drugo črko v paru) moral biti na pravem mestu (saj smo to mesto določili že v prejšnjem koraku).

To je ena iteracija algoritma. Vedno se izvede najmanj toliko iteracij, dokler se razdalja več ne zmanjšuje. Povprečno število takih iteracij je 4. Ko se iteracije več ne izvajajo, bi morali biti vsi stolpci in vse vrstice v matriki šifriranega besedila na pravem mestu. Rešitev bi morala biti prava permutacija substitucijske šifre, ki je bila uporabljena za šifriranje besedila.

Naša matrika je velikosti 30×30. Torej obstaja $30!$ ($2{,}6 * 10^{32} \cong 2^{108}$) možnih permutacij ključa. To pa je precej preveč, da bi šifro razbijali z brute-force napadom. Danes se smatra 40 bitne simetrične ključe (2^{40}) za premalo varne pred tovrstnimi napadi, saj jih je možno razbiti že z močjo osebnega računalnika. Kot ustrezno zaščito se smatra 80 bitne ključe. Najboljšo zaščito med simetričnimi ključi pa nudijo 256 bitni ključi, ki so varni tudi pred kvantnimi računalniki.

Z izvajanjem transpozicije stolpcev in vrstic v skladu s spreminjanjem razdalje pridemo do zmanjšanega števila vseh preverjenih ključev, saj ne pregledamo vseh možnih permutacij ključa. Ker *MaxN* ni poljubno veliko število, ampak je odvisno od velikosti abecede (katere velikost se tudi ne spreminja), obstaja torej zgornja meja časa izvajanja programa, ki je nikoli ne presežemo.

4.3.1. Zmanjševanje razdalje

Vsi doslej opisani razredi so povezani, sam algoritem razbijanja šifre pa uporablja štiri razrede. Najprej je bil razvit razred *TestSlovarBesedilo*. Algoritem v njem je sprva deloval tako, da je vzel samo en najmočnejši stolpec v matriki šifriranega besedila in ga premaknil na mesto, kjer se je nahajal najmočnejši stolpec v matriki slovarja (obenem se je izvedla tudi

pripadajoča transpozicija vrstice) ter izračunal razdaljo. Bil je nezanesljiv, vendar mu je v nekaterih primerih uspelo dobiti dobre rezultate (npr. pri dovolj velikih besedilih, kjer matrike frekvenc šifriranega besedila ne odstopajo veliko od referenčne). Večjo uspešnost dosežemo, če ga poženemo večkrat, saj se razdalja po več iteracijah manjša, po določenem številu pa se več ne spreminja ali se celo poveča. To je postal kriterij o prenehanju izvajanja iteracij algoritma. Potrebna je bila ročna nastavitev podanih datotek slovarja in besedila. Narejen je bil podroben izpis, ki je bil uporabljen za iskanje napak v algoritmu. Izpis je vključeval:

- število besed v vzorcu
- končno razdaljo med referenčno in šifrirano matriko frekvenc parov
- število potrebnih iteracij, da se razdalja ni več zmanjševala
- razliko razdalje med matriko šifriranega besedila in matriko razbitega besedila
- permutaciji ključa rešitve in uporabljene substitucijske šifre
- število različnih črk med šiframa
- razdalje nad matrikama slovarja in testnega besedila in med njima
- tabelo seštevka stolpcev slovarja, testnega in šifriranega besedila

```
Inicializacija...Št besed v vzorcu: 168
Resitev[0]: 0.48188684094517364 St. iteracij: 4 Je uspesno? = 0.0
**Permutacija premešanega besedila
[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, Č, Š, Ž,  ]
[G, L, Z, A, O, Y, D,  , H, T, J, R, M, N, F, P, Q, Č, C, X, U, E, W, K, S, B, V, Š, Ž, I]
**Permutacija rešitve
[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, Č, Š, Ž,  ]
[G, L, Z, A, O, Y, D,  , H, T, J, R, M, N, F, P, Q, Č, C, X, U, E, W, K, S, B, V, Š, Ž, I]
Št različnih: 0
Resitev[1]: 0.48188684094517364 St. iteracij: 5 Je uspesno? = 0.0
**Permutacija premešanega besedila
[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, Č, Š, Ž,  ]
[U, F, O, L, D, E, B, W, I, N, K, Č, M, J, C, R, A, H, Y, G, Š, V, T, X,  , Z, S, P, Q, Ž]
**Permutacija rešitve
[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, Č, Š, Ž,  ]
[U, F, O, L, D, E, B, Y, I, N, K, Č, M, J, C, R, A, H, X, G, Š, V, T, Q,  , Z, S, P, W, Ž]
Št različnih: 4
Resitev[2]: 0.48188684094517364 St. iteracij: 4 Je uspesno? = 0.0
**Permutacija premešanega besedila
[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, Č, Š, Ž,  ]
[S, T, A, R, Č, X, Š, E, I, N, K, Y, U, H, O, J, Q, W, C, V, M, F, G, B,  , Z, D, P, Ž, L]
**Permutacija rešitve
[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, Č, Š, Ž,  ]
[S, T, A, R, Č, X, Š, E, I, N, K, W, U, H, O, J, Q, Y, C, V, M, F, G, B,  , Z, D, P, Ž, L]
Št različnih: 2
Končana
Razlike: [evklidska,manhattan,maximum]
        besediloInSlovar:0.03921128860578591,0.48188684094517364,0.010218020959464533
        besediloInSifriranTekst:0.1303946986742096,1.6571428571428535,0.02857142857142857
        slovarInSifriranTekst:0.1241627152370387,1.6402001861738904,0.030237562710453355
Tabela seštevka stolpcev
SLOV:  A 0.0866975813115028 B 0.015431160475088617 C 0.0075064840520108035 D 0.028506733955306694 E 0.08654042563058256
BESE:  A 0.07472527472527471 B 0.016483516483516484 C 0.005494505494505494 D 0.023076923076923078 E 0.09780219780219777
SIFR:  A 0.04945054945054945 B 0.020879120879120878 C 0.07472527472527474 D 0.03076923076923077 E 0.013186813186813187
Permutacija šifriranega besedila
[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, Č, Š, Ž,  ]
[S, T, A, R, Č, X, Š, E, I, N, K, Y, U, H, O, J, Q, W, C, V, M, F, G, B,  , Z, D, P, Ž, L]
```

Slika 14: Izpis razreda *TestSlovarBesedilo*

4.3.2. Izbira kandidatov

Naslednji razvit razred je *SolveCandidat*, katerega postopek reševanja je opisan v psevdo kodi.

Algoritem SolveCandidat:

> Vhod: Matriki frekvenc šifriranega besedila in slovarja.
> Izhod: Tabela substitucijske šifre.

1. V tabeli pregledanih stolpcev vse vrednosti nastavimo na *false* (kar pomeni, da nismo pregledali še nobenega stolpca).
2. Poiščemo najmočnejši nepregledan stolpec v matriki frekvenc slovarja in si zapomnimo njegov indeks kot *idx*.
3. V matriki frekvenc šifriranega besedila izberemo *MaxN* najmočnejših nepregledanih stolpcev.
4. Izberemo en stolpec iz točke 3 in izvedemo transpozicijo s stolpcem na isto ležečem mestu *idx* v matriki frekvenc šifriranega besedila. Obenem se izvede tudi transpozicija pripadajočih vrstic.
5. Izračunamo razdaljo med novo nastalo matriko frekvenc šifriranega besedila in vhodno matriko frekvenc slovarja. To razdaljo in njeni transpoziciji iz točke 4 si zapomnimo.
6. Novo nastalo matriko frekvenc šifriranega besedila popravimo nazaj tako, kot da transpozicij iz točke 4 ni bilo.
7. Skočimo v točko 4 za vsak stolpec iz točke 3.
8. V matriki šifriranega besedila izvedemo transpoziciji, ki sta najbolj zmanjšali razdaljo iz točke 5.
9. V tabeli pregledanih stolpcev označimo stolpec *idx* kot že pregledan.
10. Vračamo se v korak 2 dokler ne pregledamo vseh stolpcev.
11. Shranimo razdaljo med matrikama frekvenc šifriranega besedila in slovarja.
12. Primerjamo zadnji dve shranjeni razdalji med seboj. Če se razdalja zmanjšuje, skočimo v točko 1.
13. Končana je substitucijska tabela ključa, ki je sestavljena iz transpozicij, izvedenih v točki 8.

Za razliko od algoritma v TestSlovarBesedilo se pri izbiri transpozicij začne upoštevati spremembe razdalje za *MaxN* kandidatov. Ta izboljšava bistveno dvigne uspešnost rezultatov. Kandidati za zamenjavo se shranjujejo v strukturi *stackedIndex*, kjer so zapisane njihove vrednosti in indeksi. Po velikosti se jih uredi s pomočjo sklada. Katere kandidate smo že

pregledali, si zapomnimo s pomočjo tabele *tabelaPregledanih*. Omogočeno je tudi delovanje, ko je zaporedje transpozicij v matriki šifriranega besedila odvisno od moči vrstic.

```
private int poišči Najbljižjega(List<Integer> lstMaxN, int sKaterim)
{
    if(lstMaxN.size() == 0)
        throw new UnsupportedOperationException("To stanje algoritma ni dovoljeno!");

    double najRazdalja = -1;
    int idx  = -1;
    for(Integer i : lstMaxN)
    {
        double raz = racunajRazdaljomed(i, sKaterim);
        if(idx == -1)
        {
            najRazdalja = raz;
            idx = i;
        }
        else if(najRazdalja > raz )
        {
            idx = i;
            najRazdalja = raz;
        }
    }
    return idx;
}

private double racunajRazdaljomed(int indexSifStolpca, int indexSlovarStolpca)
{
    double raz;
    matrikaSifriranegaBesedila.permutacija(indexSifStolpca, indexSlovarStolpca);
    raz = matrikaSifriranegaBesedila.razdalja(matrikaSlovarja, funkcijaRazdalje);
    matrikaSifriranegaBesedila.permutacija(indexSifStolpca, indexSlovarStolpca);
    return raz;
}
```

Slika 15: Metodi *poišči Najbližjega* in *racunajRazdaljomed*

Na sliki 15 je prikazano, kako za izbrano število *MaxN* najmočnejših stolpcev preverimo, transpozicija katerega stolpca in pripadajoče vrstice najbolj zmanjša razdaljo. Iz metode *racunajRazdaljomed* je razvidno, da se po vsakem izračunu razdalje matriko vrne v stanje pred izračunom. Šele, ko gremo čez vseh *MaxN* stolpcev, izvemo, kateri je najbolj ugoden in vrnemo njegov indeks *idx*. Izbrano transpozicijo zapišemo.

```
koncnaSubstitucija.narediZamenjavo(idxSlovar, idxSif);
```

Slika 16: Permutacija končne rešitve

V tabeli *koncnaSubstitucija* je zapisana permutacija substitucijske šifre, ki na koncu predstavlja rešitev našega problema. Metoda *narediZamenjavo* izvede transpozicijo črke v matriki šifriranega besedila na mesto, ki ustreza tistemu iz referenčne matrike frekvenc parov (kot argumenta prejme *idxSlovar* in *idxSif*, nato pa vrne izbrana stolpca in pripadajoči vrstici nazaj na prejšnje mesto.

4.3.3. Nastavitve parametrov

Ko je algoritem začel vračati uspešne rezultate, je nastala potreba po testiranju z različnimi parametri. V razredu *IzpisStatistike* so metode za izbiranje testnega besedila. S podano spremenljivko *numBesed* določimo, koliko besed bo v testnem besedilu. Nato izberemo naključno podmnožico soležnih besed moči *numBesed* iz slovarja. Podmnožica se ob vsaki iteraciji testa ponovno izbere. Metoda izbire podmnožice je predstavljena na sliki 17.

numBesed = število besed, uporabljenih za testno besedilo

```
public void generirajNaključnoBesedilo(int numBesed)
{
        int index = (int)Math.max(Math.random()*lstBesedeBesedila.size() - numBesed, 0);
        lstNoveBesede = new ArrayList<>();
        for(int i = index; i < index + numBesed; i++)
        {
                lstNoveBesede.add(lstBesedeBesedila.get(i));
        }
        matBesedilo = new MatrikaParov();
        matBesedilo.posodobiFrekvence(lstNoveBesede);

}
```

Slika 17: Metoda za generiranje naključne podmnožice besedila

Dodana je bila tudi možnost hkratnega testiranja z različnimi števili *MaxN* najmočnejših stolpcev v konfiguraciji. Za obsežna testiranja se lahko izbere število iteracij enega testa z določeno konfiguracijo. Obenem je bilo možno izpisovati posamezne korake algoritma za učinkovitejše iskanje napak v programu.

```
Slovar: C:\Users\PC\workspace\diplomas\slovar-sl.txt, Kombajn:C:\Users\PC\workspace\diplomas\kombajn
Razlika med slovarjem in testnim besedilom: E=5.456767919855653E-4
Razlika med slovarjem in testnim besedilom: M=0.007102169200510103
Konfiguracija testa:
MaxNZaRazdalji = [5, 6, 7, 8, 20, 30]
ŠTEVILO_ITERACIJ_ENEGA_TESTA = 10
TEKSTOVNE_PODMNOŽICE = [500]
ŠT_ITERACIJ_GENERACIJE_PODMNOŽICE = 200
FUNKCIJA za računanje zmote: kazenZaZmoto
**Test algoritma nad 500 besedami
#NumBesed|MaxN|razdalja|povprečen uspeh %
500|5|evklidska|81,787779
500|5|manhattan|87,899134
500|6|evklidska|83,100662
500|6|manhattan|89,779613
500|7|evklidska|82,289955
500|7|manhattan|89,325914
500|8|evklidska|82,358128
500|8|manhattan|89,798522
500|20|evklidska|82,249227
500|20|manhattan|90,107317
500|30|evklidska|81,786155
500|30|manhattan|89,878034
```

Slika 38: Izpis rezultatov testa razreda *IzpisStatistike*

4.3.4. Vrednotenje uspešnosti

Razred *Normalizacija* je postavil pravila za ugotavljanje uspešnosti razbijanja šifre. Pri uspešnosti upošteva pogostost pojavljanja napačno ugotovljene črke v splošnem. To pogostost si zapomni s pomočjo strukture *HashMap*. V njej črka ali presledek predstavljata *key*, medtem ko je *value* frekvenca.

Uspešnost razbijanja šifre se računa v metodi *kazenZaZmoto*. Izvede se primerjava tabele substitucijske šifre *SubUporabljena*, ki je bila uporabljena za šifriranje testnega besedila in *subRešitev*, ki jo je vrnil algoritem, in pogleda, pri katerih črkah se razlikujeta. Računanje kazni je obteženo glede na pogostost pojavljanja črke v slovenskem jeziku. Vsoto kazni se odšteje od uspešnosti rešitve. Tako dobimo procent vseh pravilno ugotovljenih črk in presledka v šifriranem besedilu.

```
private static double kazenZaZmoto(Substitucija subRešitev, Substitucija subUporabljena)
{
    double sestevekKazni = 0;
    for(int i = 0; i < Substitucija.tabela.length; i++)
    {
        char znak = Substitucija.tabela[i];
        if(subRešitev.substitute( subUporabljena.substitute(znak) ) != znak)
            sestevekKazni += tabelaKazni[i];
    }
    if(sestevekKazni > 1)
        return 1;
    return sestevekKazni;
}
```

Slika 49: Metoda računanja kazni

Na začetku so vse kazni v tabeli enake 0. Ob vsaki napačno ugotovljeni črki se v spremenljivko *seštevekKazni* prišteje vrednost, ki ustreza pojavljanju črke v splošnem. Seveda ena napačno ugotovljena črka pomeni dve napaki. Ko se sešteje vse kazni, se uspehu reševanja, ki je zapisan s spremenljivko *uspehr* in je na začetku 1, odšteje *seštevekKazni*. Tako dobimo odstotno uspešnost razbijanja šifre. Celoten uspeh se izračuna tako, da se sešteje vse rezultate in jih deli s številom vseh testov.

Ker imamo na voljo permutacijo abecede, s katero smo besedilo šifrirali, lahko s primerjavo enostavno preverimo uspešnost naše rešitve. V kolikor pa je ne bi imeli, bi si besedilo s črkami, ki bi bile zamenjane v skladu z rešitvijo, izpisali in skušali prepoznati besede, ki bi nam razkrile pravilno uporabljene črke. Če bi program pravilno ugotovil dovolj črk, bi nam to lahko uspelo za vse in nato bi preverili, kakšna je bila uspešnost razbijanja.

5. Testiranje

Načrtovanje obsega, natančnosti, časa in različnih faz izvajanja je bilo potrebno določiti s spremembo nastavitev pred zagonom programa. Testirali smo različna testna besedila z različnimi nastavitvami. Skozi razvoj algoritma smo sproti spoznavali okvirje v katerih se izvajajo testi. Nekaj, kar je na začetku veljalo za uspeh, je bilo kasneje razočaranje. Mnogokrat smo testirali zato, da se je odkrilo, kje se skriva napaka v algoritmu. Večinoma smo si pomagali z izpisom na mestih, ki se zdijo verjetna. Dodajale so se možnosti različnih nastavitev in tako se je omogočilo obširnejše testiranje z bolj natančnimi rezultati. V končnem testiranju smo operirali s poenotenim naborom možnosti.

5.1. Konfiguracije testa

Eno konfiguracijo testa sestavljajo nastavljive konstante:

- `boolean ENAKOPRAVNO_RACUNANJE_ZMOTE`
- `boolean PRESLEDEK`
- `int ŠTEVILO_ITERACIJ_ENEGA_TESTA`
- `int[] MaxNZaRazdalji`
- `int[] TEKSTOVNE_PODMNOŽICE`
- `int ŠT_ITERACIJ_GENERACIJE_PODMNOŽICE`

`ENAKOPRAVNO_RACUNANJE_ZMOTE` pomeni, da se bo upoštevala uspešnost s pravično kaznijo sorazmerno glede na pogostost pojavljanja napačno ugotovljene črke v slovenskem jeziku. Na sliki 3 se vidi, da je prisotnost nekaterih črk manjša in zato morda lahko zanemarljiva, zato je na voljo možnost, da se napačno ugotovljenih črk, ki se pojavljajo v manj kot 0,016%, ne upošteva. Odločimo se samo za prvo, pravičnejšo in realnejšo oceno uspešnosti.

`PRESLEDEK` je najbolj pogost znak v besedilih. Z njegovim upoštevanjem dobimo več informacij o sosednosti polj v matrikah. Testi so se izvajali z upoštevanjem presledka in tudi brez njega.

`ŠTEVILO_ITERACIJ_ENEGA_TESTA` določa, kolikokrat se bo izvedel test z določeno konfiguracijo. Ta konstanta lahko zelo poveča časovno zahtevnost izvajanja programa, vendar je zaradi želje po čim obsežnejšem testiranju visoka. Za končno testiranje se odločimo za 1000 iteracij.

`MaxNZaRazdalji` je število *MaxN* najmočnejših stolpcev, ki jih bomo uporabili za računanje sprememb razdalje. Skozi testiranja se ugotovi, da so manj kot 3 stolpci premalo za uspešno delovanje algoritma. Pri $MaxN \geq 5$ pa se rezultati ne razlikujejo več kot za pol odstotka. Zato bomo za vrednosti poleg mejnih primerov vzeli še vmesna števila:

$$MaxN = \{ 1, 2, 3, 5, 8, 13, 21, 30 \}$$

Števili 1 in 30 sta mejna primera. 1 je samo en *MaxN*, 30 pa vsi razpoložljivi stolpci (25 črk slovenske abecede, presledek, X, W, Y in Q). Ob neupoštevanju presledka *MaxN* znaša 29.

`TEKSTOVNE_PODMNOŽICE` določi, iz koliko besed bo sestavljeno testno besedilo. To je zelo pomemben pogoj, ki determinira uspešnost delovanja našega programa. Na majhnih vzorcih je pričakovati slabe rezultate, medtem ko na velikih zadovoljiv uspeh. Povprečna dolžina besede je 4,55 črke [4]. Odločimo se, da bomo vzeli naslednje primere števila besed: { 30, 60, 150, 300, 600}

- dolžina SMS sporočila (120 znakov, 30 besed)
- dolžina dveh SMS sporočil (60 besed)
- en odstavek (150 besed)
- dva odstavka (300 besed)
- 1 tipkana stran (600 besed)

`ŠT_ITERACIJ_GENERACIJE_PODMNOŽICE` določi, koliko različnih podmnožic z istim številom *numBesed* se bo ustvarilo. Za obsežno testiranje se odločimo za 1000 iteracij.

Vsak test z eno konfiguracijo se izvede za evklidsko in Manhattanovo razdaljo. Izračun pokaže, da se na eno testiranje izvede 80 milijonov različnih testov. Skupaj je bilo opravljenih več kot milijardo različnih testov. Rezultate izpisa se shranjuje v tekstovno datoteko.

6. Rezultati

Med testiranji smo dobili večje količine rezultatov z različnimi konfiguracijami. Potrebno je bilo ugotoviti, kateri nastavljivi parametri pripeljejo do najboljših rezultatov. Postaviti je bilo treba merilo uspešnosti za zadovoljiv in preslab rezultat ter se odločiti, kateri rezultati so primerni za grafični prikaz. V analizi smo predstavili, kako parametri vplivajo na uspešnost razbijanja šifre.

6.1. Merilo uspešnosti

Predstavljene meritve pokažejo zmogljivosti algoritma v odvisnosti od konfiguracij testiranja. Permutacijo substitucijske šifre, ki se je uporabila za šifriranje, se primerja s tisto, ki jo vrne algoritem. Tako se preverjajo rešitve. Merilo za uspešnosti delovanja je odstotna uspešnost ugotovljenih črk besedila. Kot zadosten rezultat smatramo vsaj 85% pravilno ugotovljenih črk, saj bi se tako besedilo intuitivno dalo brati.

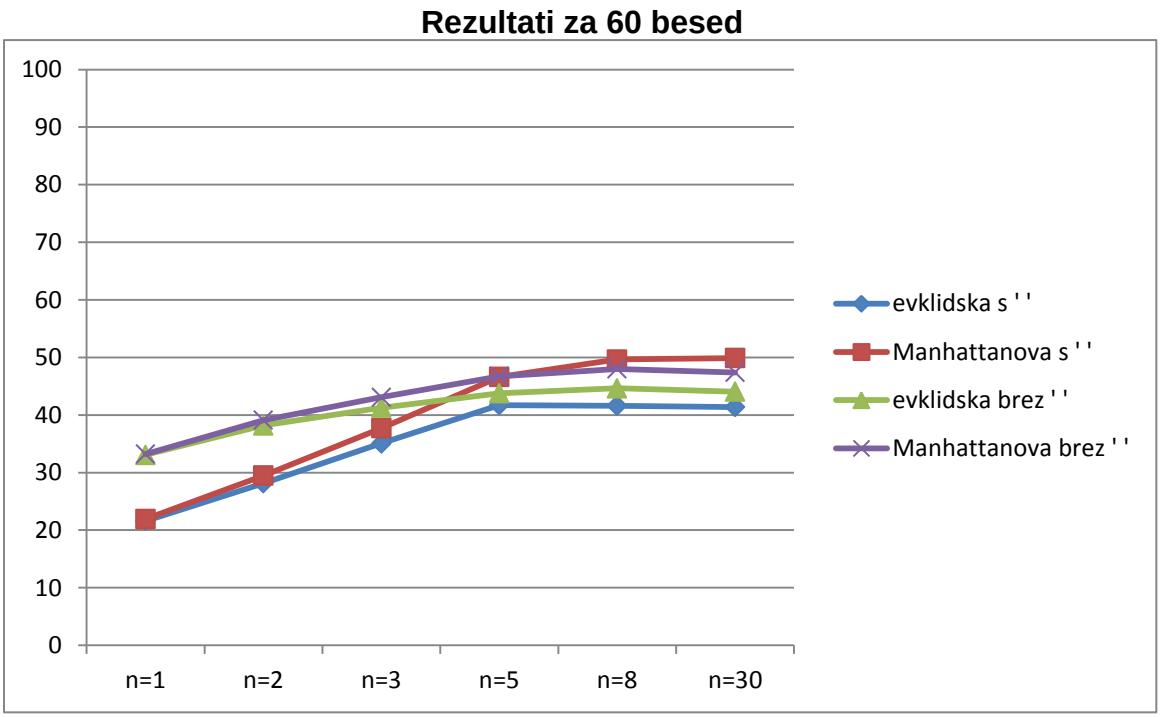
6.2. Grafični prikaz

Zaradi večje preglednosti se odločimo za grafični prikaz rezultatov. Predstavljeno je testiranje z evklidsko in Manhattanovo razdaljo za slovenski jezik z in brez upoštevanja presledka. Prikazani so testi za 30, 60, 150, 300 in 600 besed. Na grafu nam Y-os pove odstotno uspešnost razbijanja šifre, medtem ko X-os pove število *MaxN*. Pri grafu, kjer je predstavljenih več rezultatov, nam Y-os pove odstotno uspešnost razbijanja šifre, X-os pa število *numBesed*. Ker so rezultati večinoma najboljši za *MaxN* = 8 in se za večje *MaxN* spreminjajo relativno malo, smo vzeli:

$$MaxN = \{ 1, 2, 3, 5, 8, 30 \}$$

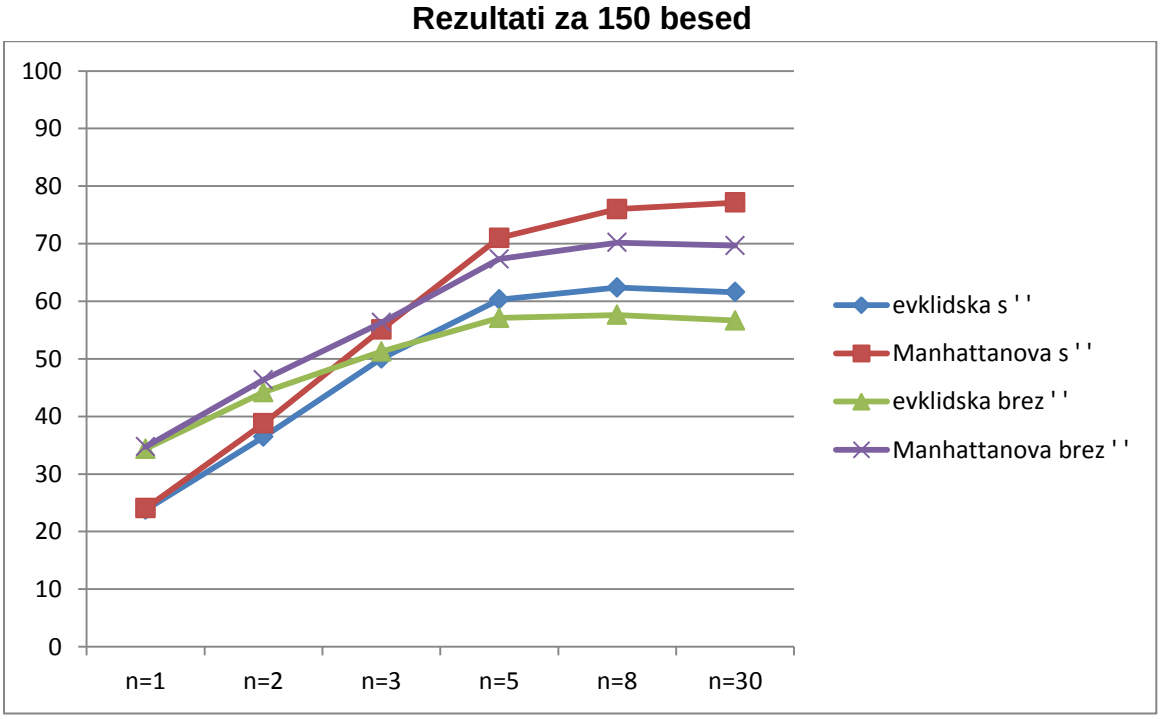
Različni grafi:

- za različne *MaxN* so predstavljeni testi za evklidsko in Manhattanovo razdaljo za slovenski jezik z in brez upoštevanja presledka
- za več *numBesed* so predstavljeni testi za oba načina računanja razdalj z in brez presledka za slovenski jezik, ki vsebujejo več različnih rezultatov (upoštevani samo rezultati za najboljši *MaxN*)



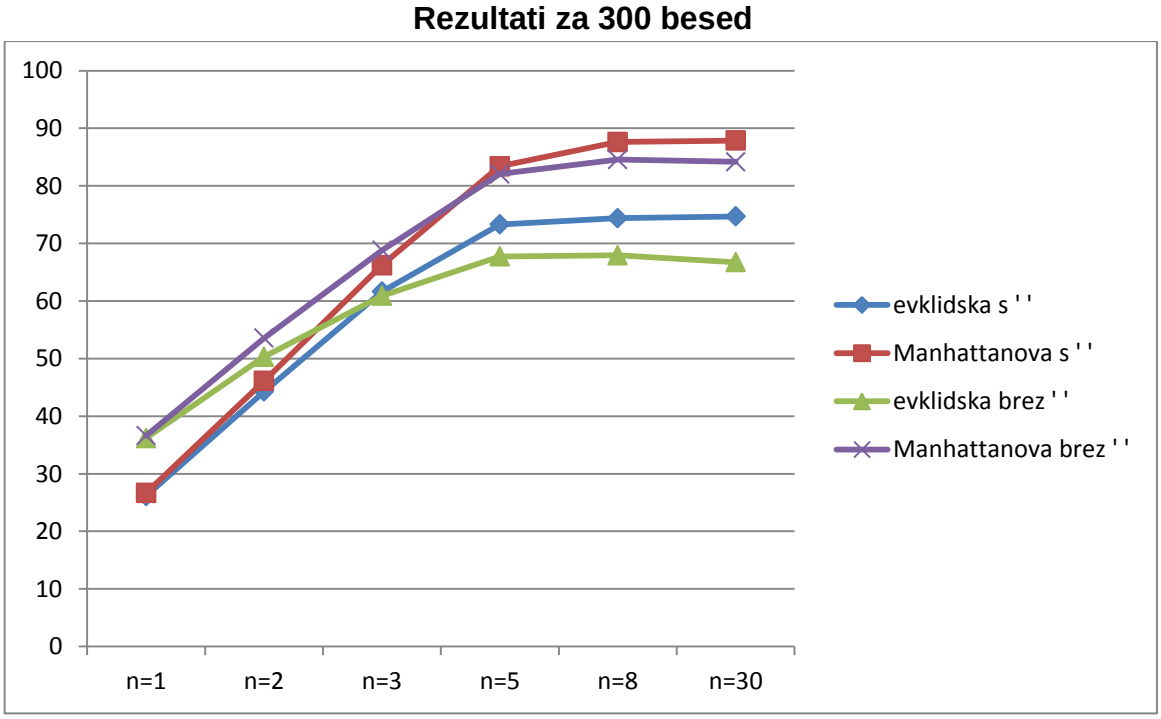
Slika 20: Graf uspešnosti za 60 slovenskih besed

Na sliki 20 se vidi, da je uspešnost razbijanja šifre ob Manhattanovi razdalji 50% z upoštevanjem presledka in 48% brez, kar ni zadovoljiv rezultat. Za evklidsko razdaljo je rezultat uspešnosti 41% z in 44% brez upoštevanja presledka. Algoritem z večjim številom *MaxN* dosega boljše rezultate.



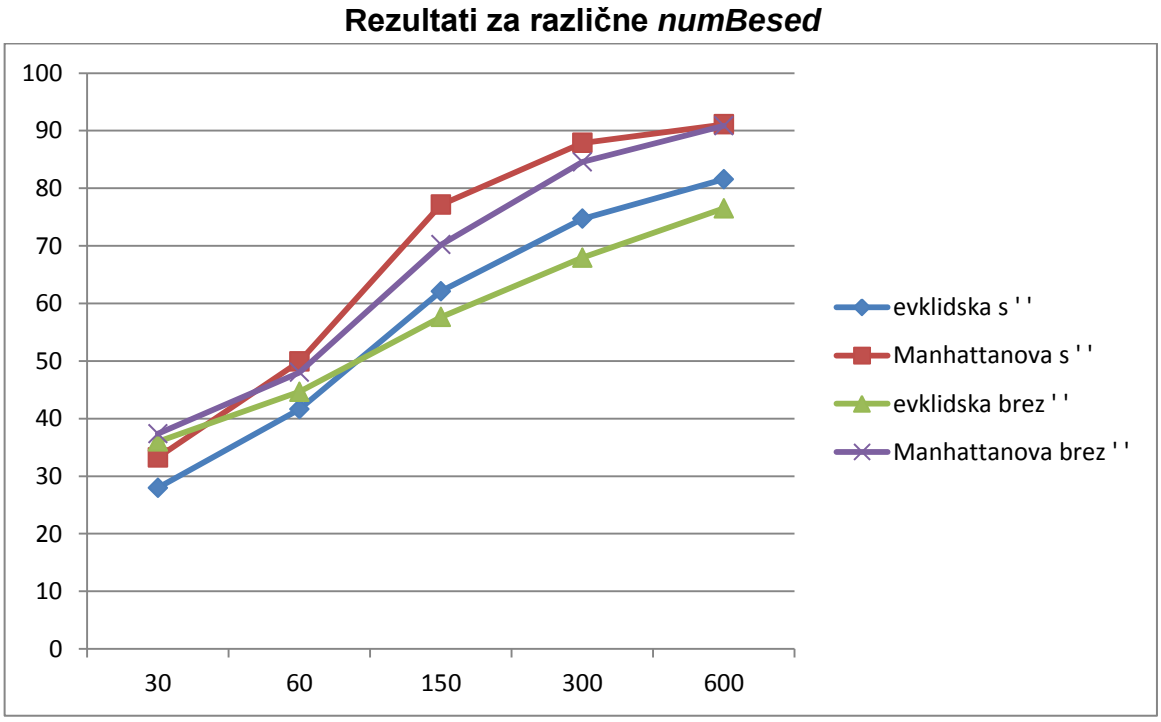
Slika 25: Graf uspešnosti za 150 slovenskih besed

Na sliki 21 se vidi, da algoritem doseže 77% uspešnost razbijanja šifre za Manhattanovo in 61% za evklidsko razdaljo z upoštevanjem presledka. Uspešnost brez upoštevanja presledka je 70% za Manhattanovo in 57% za evklidsko razdaljo.



Slika 6: Graf uspešnosti za 300 slovenskih besed

Na sliki 22 se vidi, da algoritem z upoštevanjem presledka doseže zadovoljivo 88% uspešnost pri Manhattanovi razdalji, medtem ko pri evklidski razdalji doseže 69%. Brez upoštevanja presledka doseže 84% uspešnost ob Manhattanovi in 67% ob evklidski razdalji.



Slika 7: Graf uspešnosti za različne *numBesed*

Na sliki 23 se vidi, kako uspešnost s povečevanjem števila *numBesed* narašča. Največji preskok se zgodi s 60 na 150 besed. Najboljše se odreže Manhattanova razdalja z upoštevanjem presledka, zanimivo pa je, da po 600 besedah doseže primerljive rezultate tudi brez upoštevanja presledka.

6.3. Analiza rezultatov

Opaziti je, da se Manhattanova razdalja skoraj vedno odreže bolje kot evklidska, čeprav je ta razlika pri manjšem številu *MaxN* neizrazita. Razlika med njima se ob neupoštevanju presledka še poveča. Zadovoljive rezultate 88% uspešnosti dosežemo za Manhattanovo razdaljo ob upoštevanju presledka pri 300 besedah. Zanimivo je, da pri *numBesed* manjšem od 60, dosežemo večjo uspešnost, če presledka ne upoštevamo. To se zgodi zato, ker ob manjših vzorcih prihaja do večjih razlik med matrikama frekvenc šifiranega besedila in slovarja, kazen za napačno ugotovljen presledek pa je velika (odbije se 17,03%).

Do napak pri razbijanju šifre pride, ko se izračuni razdalj ob transpozicijah zelo malo razlikujejo. V takem primeru lahko izberemo tistega kandidata, ki v trenutnem koraku najbolj zmanjša razdaljo med matrikama, vendar se v prihodnjih korakih izkaže, da je bil izbran napačen. Mesta, kjer je najugodnejši kandidat napačen, imenujemo lokalni minimumi.

Uspeh je najbolj odvisen od spremenljivke *numBesed*, ki vpliva na velikost besedila. Matrike frekvenc pri manjših besedilih dobijo premalo podatkov, da bi bile primerne za zanesljive rezultate. Pri večjih besedilih pridemo do popolnih rešitev.

Število *MaxN* je odločilno vplivalo na kakovost rešitev. Uspešnost se glede na velikost *MaxN* viša algoritemsko, dokler ne doseže svoje najboljše vrednosti, nato stagnira. Pri večjih testnih besedilih dosegajo dobre rezultate tudi uporabe manjšega števila stolpcev *MaxN*, kar logično sledi, saj so vzorci v matrikah močneje izraženi. To drži tudi za testiranje algoritma brez upoštevanja presledka. Od *MaxN* [5,30] dobimo primerljive rezultate, ki se razlikujejo največ za pol procenta. Za *MaxN* [1,3] dobimo občutno slabše rezultate. Zanimivo je tudi, da največkrat dosežemo najboljše rezultate za *MaxN*=8.

Presledek precej vpliva na uspešnost rešitev med 60 in 600 besed, saj kot najbolj zastopan znak v besedilih dobi v matriki najmočneje zastopana stolpec in vrstico, kar pomaga pri računanju razdalje.

Naš algoritem sloni na zaporedju transpozicij nad matriko šifriranega besedila, ki je odvisno od moči stolpcev. Možno pa je tudi razbijanje šifre, kjer vrstni red izvajanja transpozicij določa moč vrstic. Pri manjšem številu besed se dobijo slabši rezultati kot z zaporedjem ob upoštevanju moči stolpcev (npr. pri 150 besedah je uspešnost 69% za Manhattanovo razdaljo z upoštevanjem presledka, pri 600 besedah z istimi nastavitvami pa 88%, kar je približno 3% slabše kot pri zaporedju transpozicij na podlagi moči stolpcev). Ob večjem številu besed so rezultati primerljivi.

7. Sklep

V diplomskem delu opišemo orodja za razbijanje substitucijske šifre. Najprej se posvetimo kriptografiji, predstavimo matematične osnove matrik in metrik ter opišemo glavne lastnosti korpusov in razvojnega okolja, ki smo ga uporabili za izdelavo programa. Izpostavimo definicijo in prikažemo ter razložimo shemo celotnega postopka. Na kratko opišemo še vhodni datoteki v program, slovar in testno besedilo in kako ju pridobimo.

Nato predstavimo vse razrede, ki sestavljajo orodja, s katerimi si pomagamo razbijati substitucijsko šifro. Buffer in Tokenizer poskrbita, da vhodni datoteki vstopata v računalnik v pravi obliki, ki je pripravljena na nadaljnje obdelovanje. S substitucijsko šifro testno besedilo zašifriramo in si zapomnimo permutacijo, ki je bila uporabljena kot ključ. Z razredom Matrika parov lahko izračunamo normalizirane pogostosti pojavljanja parov črk. Tisto matriko frekvenc parov, ki jo pridobimo iz slovarja, imenujemo referenčna matrika frekvenc in je vedno ista. Nad testnim besedilom pa vsakič znova izračunamo novo, ki jo primerjamo z referenčno. Na podlagi števila najmočnejših stolpcev v matriki frekvenc testnega besedila izvajamo zaporedje transpozicij na primerljivo izbrano mesto v referenčni matriki frekvenc. Sproti računamo, kako se spreminja razdalja in naredimo tisto transpozicijo, ki razdaljo med matrikama najbolj zmanjša. Ta postopek ponavljamo, dokler ne izvedemo vseh transpozicij stolpcev in vrstic ter se razdalja več ne zmanjšuje. Tako dobimo permutacijo substitucijske šifre ali ključ, ki nam služi kot končni rezultat. Dobljeni ključ primerjamo s tistim, ki smo ga uporabili za šifriranje testnega besedila in izračunamo uspešnost razbijanja glede na obteženo pojavljanje pravilno ugotovljenih črk.

Izvedemo obsežna testiranja z različnimi konfiguracijami in ugotavljamo, kje so meje vrednosti, pri katerih algoritem še dobro deluje. Rezultate predstavimo z grafičnim prikazom z različnimi konfiguracijami in na mestih, kjer dosežejo zadovoljivo uspešnost. Možen je tudi grafični prikaz matrik parov, ki pomaga pri odločitvi za način delovanja algoritma.

Rezultate analiziramo in izpostavimo, kateri so najpomembnejši parametri v konfiguracijah ter na kaj vplivajo. Za najpomembnejšega izmed parametrov v konfiguraciji testiranja se izkaže število *numBesed*, ki določa velikost testnega besedila. Pri številu besed dveh odstavkov dosežemo zadovoljivo raven uspešnosti. Na krajših zašifriranih besedilih dosežemo slabše rezultate, vendar bi si z orodji, predstavljenimi v tej diplomski nalogi, lahko pomagali pri razbijanju šifer. Na rezultate vpliva tudi število *MaxN*, ki obenem določa vrstni red izvajanja transpozicij v matriki šifriranega besedila. Najboljše rezultate dosežemo z Manhattanovo razdaljo ob prisotnosti presledka v besedilu. Z dopolnitvijo črk, ki so značilne za nek jezik (tako kot č, ž, š za slovenščino), je lahko algoritem uporaben tudi za razbijanje šifer v drugih jezikih.

Kazalo slik

Literatura in viri

[1] P. Baker, "Using Corpora in Discourse Analysis", Continuum, 2006

[2] T. Erjavec, "Korpusno jezikoslovje in jezikovne tehnologije", Zbornik osme konference jezikovne tehnologije, 2012, Dostopno na: http://nl.ijs.si/et/teach/, 11.9.2012

[3] T. Erjavec, N. Logar Berginc, "Referenčni korpusi slovenskega jezika (cc)Gigafida in (cc)KRES", 2012, Dostopno na: http://nl.ijs.si/isjt12/, 4.6.2012

[4] P. Jakopin, "Zgornja meja entropije pri leposlovnih besedilih v slovenskem jeziku", doktorska disertacija na Fakulteti za elektrotehniko, Univerza v Ljubljani, 1999

[5] A. Jurišić," Kriptografija in teorija kodiranja", CD-ROM, Matematična knjižnica FMF in IFM, 2007

[6] B. Magajna, "Linearna algebra, metrični prostor in funkcije več spremenljivk", DMFA-založništvo, 2011

[7] A. Menezes, P. van Oorschot in S. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1997

[8] D. Stinson, "Cryptography- Theory and Practice", CRC Press, 3rd edition, 2005

[9] J. Žerovnik, "Matematika 2", Fakulteta za strojništvo, 2009

[10] Wikipedia, "Matrika", Dostopno na: http://sl.wikipedia.org/wiki/Matrika, 16.10.2012

[11] "Eclipse", Dostopno na: http://www.eclipse.org/org/, 24.9.2012

[12] "Slovar slovenskega knjižnega jezika", Inštitut za slovenski jezik ZRC SAZU, spletna izdaja, 2000, Dostopno na: bos.zrc-sazu.si/sskj.html, 6.1.2013

[13] " Zbirka slovenskih leposlovnih besedil", Dostopno na: http://lit.ijs.si/leposl.html, 30.6.2012