Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Automatizácia riešenia substitučných šifier

(Bakalárska práca)

Marcel Kucharík

9.2.1 Informatika

Vedúci: Mgr. Tomáš Kulich

Bratislava, 2009

Abstrakt

Autor:	Marcel Kucharík
Názov bakalárskej práce:	Automatizácia riešenia substitučných šifier
Škola:	Univerzita Komenského v Bratislave
Fakulta:	Fakulta matematiky, fyziky a informatiky
Katedra:	Katedra informatiky
Vedúci bakalárskej práce:	Mgr. Tomáš Kulich
Rozsah práce:	33 strán
Bratislava, jún 2009	

V tejto bakalárskej práci sa venujem riešeniu substitučných šifier s dodatočnou informáciu. Touto dodatočnou informáciou môže byť jazyk alebo štruktúra šifry. Poukazujem na nevýhody riešenia týchto šifier frekvenčnou analýzou a ponúkam inú, efektívnejšiu metódu riešenia. Túto som aj naprogramoval a s programom otestoval jej úspešnosť.

Kľúčové slová: substitučná šifra, riešenie šifier, frekvenčná analýza, kryptoanalýza.

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Poďakovanie

Chcem sa poďakovať v prvom rade môjmu bakalárskemu vedúcemu Mgr. Tomášovi Kulichovi, ktorý sa mi počas tvorby celej práce venoval a pomáhal mi svojím odborným prístupom. Ďalej vďaka patrí mojej priateľke Martine Hojčkovej za morálnu podporu a gramatickú korektúru. Slová mojej vďaky patria aj spolužiakom, ktorí mi vedeli poradiť pri rôznych problémoch.

Obsah

Zoznam tabuliek

Zoznam obrázkov

Úvod

Medzi najznámejšie šifrovacie algoritmy patrí jednoduchá substitučná šifra. Jej vznik je datovaný mnoho rokov pred nástupom počítačov a výpočtovej techniky. Aj preto už v dnešných dobách stráca svoje uplatnenie. Napriek tomu sa s jej obmenami môžeme stretnúť v symetrickom šifrovaní alebo šifrovacích hrách.

V mojej práci sa venujem automatizácií riešenia substitučných šifier. Ukážem, že riešenie frekvenčnou analýzou neposkytuje požadované výsledky a skonštruujem algoritmus, ktorým následne riešim tieto šifry s použitím výpočtovej techniky.

Bakalárska práca je rozdelená na 3 kapitoly.

V prvej kapitole sa zaoberám základnými pojmami súvisiacimi so substitučnou šifrou. Rozoberám jej nedostatky a ponúkam rôzne obmeny tejto šifry, ktoré čiastočne alebo úplne stierajú spomenuté nedostatky.

Druhá kapitola hovorí o samotnom riešení substitučnej šifry. Spomínam dve metódy na jej riešenie - frekvenčnú analýzu a algoritmus s použitím n-gramov, ktorý som aj implementoval v mojom programe.

Výsledkom tejto metódy sa venuje 3. kapitola, v ktorej som navyše navrhol jej možné zlepšenia.

Kapitola 1

Substitučná šifra

1.1 Šifrovanie a kryptológia

Kryptológia je vedný odbor zaoberajúci sa bezpečnostnými prvkami prenosu informácií. Až donedávna bolo jej jediným cieľom šifrovanie, teraz sa však pridávajú podružné odvetvia ako autorstvo, elektronické podpisy a pod.

Šifrovanie je proces transformácie normálnej informácie na nezrozumiteľný (zašifrovaný) text. Dešifrovanie je opakom - získavaním pôvodnej informácie zo zašifrovaného textu. Pôvodnú informáciu nazývame otvorený text a jeho zašifrovaný ekvivalent šifrový text. Šifrou teda môžeme nazvať dvojicu algoritmov - šifrovací a dešifrovací. Šifrovanie je navyše parametrizované dodatočnou vstupnou informáciou - kľúčom, ktorý je nezávislý na otvorenom texte. Tento kľúč je veľmi dôležitý, keďže šifrovacie algoritmy bez neho sú slabé - k tajnej informácií sa ľahko dostane aj nepovolaný subjekt. Uvažujeme totiž, že šifrovanie nie je založené na utajení šifrovacieho a dešifrovacieho algoritmu, ale kľúča.

Na základe použitia kľúča rozlišujeme 2 základné typy šifier - symetrické a asymetrické. Symetrické šifry používajú na šifrovanie a dešifrovanie ten istý kľúč, kým asymetrické používajú dva rôzne - verejný (public key) na

šifrovanie a súkromný (private key) na dešifrovanie. Symetrické šifry možno ďalej rozdeliť na blokové a prúdové. Pri blokových sa text rozdelí na časti (bloky) pevnej dĺžky. Každý tento blok sa potom šifruje aj dešifruje zvlášť. Pri prúdových sa pomocou kľúča nastaví počiatočný stav konečnostavovému deterministickému zariadeniu (KSDZ), ktoré poskytuje prúd bitov. Tento prúd bitov skombinovaný (operácia XOR) s otvoreným textom dáva šifrový text. Na dešifrovanie sa použije rovnaký kľúč, vygeneruje rovnaký prúd bitov, čím sme schopní spätne zrekonštruovať otvorený text. Týmto sme vymedzili pojmy, ktoré tvoria šifrovací systém.

Kryptológia sa delí na dve základné disciplíny: kryptografia a kryptoanalýza. Kým kryptografia sa zaoberá navrhovaním šifrovacích systémov podľa konkrétnych bezpečnostných požiadavok, kryptoanalýza skúma možnosti útokov na tieto systémy a ich bezpečnosť.

1.2 Zraniteľnosť šifrovacieho systému

Formálne takýto šifrovací systém môžeme zapísať ako dvojicu funkcií (algoritmov):

$E : P \times K \rightarrow C$

$D : C \times K \rightarrow P$

s vlastnosťou $\forall k \in K, \forall p \in P : D(E(p,k),k) = p$, kde množiny P, K, C sú konečné. P je množina všetkých otvorených textov, K množina všetkých kľúčov a C množina všetkých šifrových textov. Funkcia E reprezentuje šifrovanie a D reprezentuje dešifrovanie. Pri útoku na šifrový systém sa snažíme zistiť používaný kľúč alebo otvorený text k šifrovému textu.

Na základe znalostí útočníka o šifrovacom systéme rozlišujeme 4 základné typy útokov na šifrové systémy:

- COA (ciphertext only attack) - útok zo znalosťou len šifrového textu

- KPA (known plaintext attack) - útok zo znalosťou otvoreného textu (a samozrejme ekvivalentného šifrového textu)
- CPA (chosen plaintext attack) - útok s možnosťou voľby otvoreného textu
- CCA (chosen ciphertext attack) - útok s možnosťou voľby šifrového textu

Tieto útoky sú zoradené podľa sily vzostupne. Najčastejšie sa však stretávame práve s COA, keďže šifrový text sa pri prenose od odosielateľa k príjemcovi veľmi ľahko dostane do nepovolaných rúk. Pri tomto type útoku je definovaný pojem **absolútne bezpečný šifrový systém**. Ten sa opiera o predpoklady:

- fixná distribúcia pravdepodobností nad P a K
- konkrétny kľúč je zvolený len na jedno šifrovanie
- voľba kľúča je nezávislá na voľbe otvoreného textu

a vyžaduje nepodmienenú bezpečnosť systému.

Bezpečnosť systému delíme na výpočtovú a nepodmienenú. Pri výpočtovej predpokladáme, že útočník je síce schopný vytvoriť algoritmus, ktorým sa dostane od šifrového textu k otvorenému, ale má len obmedzenú výpočtovú silu, ktorá nepostačuje na zrealizovanie tohto algoritmu. Čas potrebný na zlomenie šifry výpočtovou silou je často úmerný mohutnosti množiny K kľúčov, keďže najjednoduchším útokom je prebratie celej množiny kľúčov a (pri COA) testovanie „zmysluplnosti“ takto získaného otvoreného textu. Preto pri návrhu šifrovacieho systému treba zobrať do úvahy aj požiadavku na čas, ktorý má trvať útočníkovi prelomiť tento systém a získať otvorený text.

Pri nepodmienenej bezpečnosti predpokladáme, že útočník má neobmedzenú výpočtovú silu.

1.3 Substitučné šifry

1.3.1 Jednoduchá substitučná šifra

Označme množinu znakov abecedy jazyka otvoreného textu ako A. Kľúčmi budú permutácie nad touto abecedou. Nech $P = C = A$. Šifrovacia a dešifrovacia funkcia sú definované ako:

$E(p, k) = k(p)$

$D(c, k) = k^{-1}(c)$

Takto je definovaná jednoduchá substitučná šifra pre každé písmeno abecedy. Pri šifrovaní textu aplikujeme E na jednotlivé znaky textu. Dešifrovanie prebieha analogicky, s inverznou permutáciou.

Pri jednoduchých substitučných šifrách má zmysel hovoriť iba o COA type útoku, pretože ostatné typy útokov by ju veľmi ľahko zlomili. Pri KPA útočník získava priamo časti permutácie (kľúča) z dvojíc písmen otvoreného a šifrového textu. Pri CPA a CCA je situácia ešte ľahšia - útočník zvolí ako vstup celú abecedu a dostane kľúč (alebo inverzný kľúč) ako výstup. Preto ďalej v práci nebudem rozoberať, o aký typ útoku sa jedná, ale uvažovať vždy COA útok. Bezpečnosť tejto šifry je iba výpočtová, keďže po vyskúšaní všetkých permutácií dostávame otvorený text ako jednu z možností. Mohutnosť množiny kľúčov je však $|A|!$, čo je už pri anglickom jazyku ($|A| = 26$, $|K| = 26! = 403291461126605635584000000 \approx 2^{88,38}$) priveľa na útok hrubou silou. Pri slovenskom a iných jazykoch je to kvôli väčšej základnej abecede ešte viac. To však stále nepostačuje na to aby bola šifra bezpečná.

1.3.2 Obmeny substitučných šifier

Bezpečnostný problém jednoduchých šifier tkvie hlavne v nerovnomernom rozložení početnosti písmen otvoreného textu. Funkcia E túto vlastnosť otvoreného textu prenáša aj na šifrový text. Takže je možné spraviť frekvenčnú

analýzu [Sta04] (bližšie v 2.1) a mapovanie početností písmen v šifrovom texte na ich ekvivalenty v texte otvorenom. Preto vznikli mnohé obmeny substitučných šifier pokúšajúce sa zrovnomerniť rozloženie početnosti písmen a tým sťažiť (alebo znemožniť) útoky. Spomeniem niektoré z nich:

Polygrafická substitučná šifra

Definujeme E a D nie na písmenách, ale na väčších blokoch (2, 3, a viac znakových) - zväčšíme tým mohutnosť množiny kľúčov a zároveň tým vyrobíme viac základných znakov pre permutácie. Tým značne sťažíme frekvenčnú analýzu - kým pri 26 písmenách anglickej abecedy bol znak e ďaleko najčastejší, teraz musíme uvažovať všetky kombinácie e a iného znaku, kde nám môžu vzniknúť rovnako pravdepodobné kombinácie. Početnosť takýchto blokov písmen je rovnomernejšia ako pri jednotlivých písmenách.

Homofónna substitučná šifra

Predstavuje priamočiarejší spôsob, ako zrovnomerniť rozloženie písmen. Pri tejto šifre je množina C väčšia ako P, pretože znak z P môže byť zašifrovaný do rôznych znakov. Samozrejme tak, aby spätná dešifrácia bola deterministická. Ak pre početnejšie znaky vyberieme viac znakov a menej početné budeme šifrovať len do jedného alebo dvoch, dokážeme spraviť frekvenčnú analýzu nepoužiteľnou.

Polyalfabetické substitučné šifry

Používajú viaceré permutácie pre jednotlivé znaky otvoreného textu podľa kľúča. Patrí sem napríklad známa Vignereho šifra, ktorá používa tabuľku písmen 26x26 - prvý riadok sú písmená abedecy, ďalšie riadky sú tieto isté písmená posunuté vždy o počet riadkov doľava. Kľúčom je nejaké slovo alebo fráza napr. DOM. Šifrovanie potom prebieha pre každé písmenko inou per-

mutáciou, pričom permutácie sa vyberajú ako riadky tabuľky podľa kľúča. Čiže v našom prípade by sa prvé písmenko zašifrovalo permutáciou D (4 riadok), 2. písmenko s O, 3. s M, 4. s D a tak ďalej. V praxi sa tieto kľúče skladajú z niekoľkých fráz. Táto šifra bola zlomená až v roku 1863, keď F. Kasiski publikoval metódu o zisťovaní dĺžky kľúča. So znalosťou dĺžky kľúča sa z tejto šifry stáva iba viacero jednoduchých substitučných šifier. Napriek tomu je tento druh šifry teoreticky ťažko napadnuteľný, ak sú použité viaceré permutácie a nie iba posúvanie, kľúč je náhodný a celkový text je dosť krátky. Hranicu zlomiteľnosti šifry sme znásobili (zo základom v jednoduchej substitučnej) počtom písmen kľúča.

Substitučné šifry v modernej kryptografii

Substitučné šifry spomenuté vyššie, a najmä tie staršie, už nemajú seriózne použitie. Nástupom počítačov a veľkým nárastom výpočtovej sily útočníka sa stali priveľmi zraniteľné. Napriek tomu substitučné šifry nezanikli, ale stále sa používajú ako súčasť väčších šifrovacích systémov. Moderné bitovo orientované blokové šifry sú vlastne systémy substitučných a permutačných šifier nad veľmi veľkou binárnou abecedou. Spomeniem napr. Data Encrypion Standard (DES), ktorý sa donedávna aktívne používal. Jeho 56-bitový kľúč však nestačil na distribuované siete počítačov. Nahradil ho Advanced Encryption Standard (AES) [inf01] používajúci 128, 192 a 256-bitové kľúče. Dnes je AES jedným z najpopulárnejších algoritmov používaných v symetrickom kryptovaní. Ďalšie známe symetrické algoritmy sú Triple DES (TDEA) a Skipjack [oST].

1.3.3 Návrat k jednoduchej substitučnej šifre

Pri procese lámania týchto substitučných šifier však vždy narazíme na riešenie nejakej jednoduchej substitučnej šifry, či už v poslednom kroku (po zistení

dĺžky kľúča polyalfabetickej šifry), alebo v niekoľkých krokoch pri lúštení systému šifier. Polygramická šifra sa dá tiež ľahko konvertovať na jednoduchú iba rozšírením abecedy. Na riešenie jednoduchej substitučnej šifry máme síce nástroj, ktorým je frekvenčná analýza, tá sa však ukazuje na tieto obdoby a, ako neskôr v 2.1.1 ukážem, aj na samotnú jednoduchú substitučnú šifru ako nepostačujúca. Aj preto som sa rozhodol napísať túto bakalársku prácu ako pokročilejší nástroj na riešenie takýchto šifier.

Kapitola 2

Riešenie substitučnej šifry

Ako sme už spomenuli, triviálne riešenie prehľadávaním kľúčov a overovaním „zmysluplnosti“ riešenia pri jednoduchých substitučných šifrách neprichádza v úvahu. Takéto riešenie by trvalo neúmerne dlho. Treba hľadať sofistikovanejší algoritmus, ktorý by z podoby šifrového textu vedel vylúčiť nesprávne kľúče, alebo by poskytoval návod, ako sa k výslednej permutácií dostať kratšou cestou. Takéto algoritmy však zväčša potrebujú ešte nejakú dodatočnú informáciu o tom, ako asi bude vyzerať otvorený text. Takáto dodatočná informácia môže byť napr. jazyk otvoreného textu alebo jeho štruktúra. V ďalšom texte budeme predpokladať, že poznáme jazyk otvoreného textu a teda môžeme postaviť algoritmy aj s uplatnením vlastností tohto jazyka.

2.1 Frekvenčná analýza

Frekvenčná analýza sa ukázala ako veľmi užitočný nástroj na lámanie šifier alebo aspoň získavanie informácií o množstve šifier. Bez pomoci ľudského prístupu a improvizácie nedosahuje však zďaleka výborné výsledky. To značne znižuje možnosť zautomatizovania tohto prístupu.

2.1.1 Nevýhody frekvenčnej analýzy

Jednou z nevýhod frekvenčnej analýzy je však fakt, že využíva len početnosť znakov abecedy. Ak si je toho vedomá šifrujúca osoba, môže upraviť správu tak, aby sa frekvenčnou analýzou dosiahli len nič nehovoriace výsledky. Napr. **e** je najpočetnejšie písmenko anglickej abecedy, ale boli napísané celé knihy bez jeho použitia, len s písaním synoným alebo opisných vyjadrení neobsahujúcich toto písmenko.

Znak	%	Znak	%	Znak	%	Znak	%
A	10,88	R	4,74	U	3,13	G	0,21
O	9,39	V	4,63	P	3,03	F	0,20
E	8,49	L	4,41	Z	3,00	X	0,00
S	6,17	K	3,99	Y	2,70	Q	0,00
N	5,99	D	3,79	H	2,51	W	0,00
I	5,79	M	3,61	J	2,19		
T	5,76	C	3,58	B	1,81		

Tabuľka 2.1: Frekvencia výskytu znakov v slovenskom jazyku

Ďalšou a podstatnejšou nevýhodou je potreba veľmi dlhého šifrového textu. Kratšie texty totiž nemusia mať distribúciu písmen takú, ako by sme na základe štatistických dát o texte predpokladali. Napr.:

Predpokladajme, že otvorený text je zo slovenského jazyka a písmená majú pravdepodobnosť výskytu ako ukazuje tabuľka 2.1.1. Vezmime dvojicu znakov c_1 a c_2 s pravdepodobnosťami výskytu p_1 a p_2, kde $p_1 > p_2$. Bude nás zaujímať, aký dlhý text potrebujeme, aby sme s 95%-nou pravdepodobnosťou len na základe frekvenčnej analýzy odlíšili tieto znaky. Predpokladáme, že výskyt znakov môžeme aproximovať normálnym rozdelením. Zaveďme náhodnú premennú $X = P_1 - P_2$, kde P_1 a P_2 sú náhodné premenné zodpovedajúce výskytu znakov c_1 a c_2. Táto má strednú hodnotu

$E(X) = p_1 - p_2$. 95%-ná pravdepodobnosť približne zodpovedá dvojnásobku štandardnej odchýlky, čiže nás zaujíma n - počet znakov, pre ktoré táto náhodná premenná nenadobudne zápornú hodnotu s touto pravdepodobnosťou. Matematicky:

$$E(X) - 2 * \sigma(X) > 0 \tag{2.1}$$

Pričom výpočet štandardnej odchýlky sa riadi vzťahom:

$$\sigma(X) = \sqrt{D(X)} = \sqrt{D(P_1 - P_2)} \tag{2.2}$$

Odtiaľ, použitím vlastností disperzie:

$$\sigma(X) = \sqrt{D(P_1) + D(P_2)} = \sqrt{\frac{p_1 * (1 - p_1)}{n} + \frac{p_2 * (1 - p_2)}{n}} \tag{2.3}$$

Kombináciou 2.1 a 2.3 dostávame:

$$p_1 - p_2 - 2 * \sqrt{\frac{p_1 * (1 - p_1) + p_2 * (1 - p_2)}{n}} > 0 \tag{2.4}$$

Z čoho po úprave dostaneme n - počet znakov textu:

$$n > 4 * \frac{p_1 * (1 - p_1) + p_2 * (1 - p_2)}{(p_1 - p_2)^2} \tag{2.5}$$

To napríklad pre spoluhlásky **K**, **L**, $p_1 = p_K = 0,0399$, $p_2 = p_L = 0,0441$ vychádza $n \approx 18246$ znakov a pre **A** a **E** $n \approx 1223$. Preto riešenie kratších textov založené na frekvenčnej analýze potrebuje veľké množstvo improvizácie, ako napr. dopĺňanie slov, použitie digramov a pod. Tieto skutočnosti výrazne znižujú možnosť zautomatizovať riešenie založené len na frekvenčnej analýze. Naprogramovanie takéhoto riešenia by vyžadovalo nadľudské úsilie s otáznymi výsledkami.

2.2 Riešenie využitím vnútorných vlastností jazyka

Teraz opíšem riešenie využívajúce väčšiu informáciu o jazyku, ktoré som aj implementoval v mojom programe. Riešenie je založené na postupnostiach za sebou nasledujúcich písmen známych aj ako N-gramy.

2.2.1 N-gramový model jazyka

Nech $s = (s_0, s_1, \ldots, s_l)$ je reťazec dĺžky l. Ďalej nech $P(s)$ je pravdepodobnosť výskytu reťazca s v danej zbierke údajov. Nech n je dĺžka n-gramu v jazykovom modeli. Nech platí:

$$P(s) = \prod_{i=n}^{l} P(p_i | p_{i-n+1}, \ldots, p_{i-2}, p_{i-1}) \tag{2.6}$$

Takto definovaný n-gramový model jazyka môžeme pre lepšie pochopenie trochu preformulovať:

$$P(s) = \prod_{i=1}^{l-n+1} P\left(r_i | r_i = (s_i, s_{i+1}, \ldots, s_{i+n-1})\right) \tag{2.7}$$

Týmto preformulovaním sme vyrobili n-gramový model jazyka odvodený z podreťazcov (r_i) dĺžky n. Pravdepodobnosť výskytu reťazcov dĺžky n (n-gramov) som odvodil zo štatistík získaných z tzv. „trénovacieho textu“. Bližšie v 3.1.

Takýto model nám vlastne poskytuje ohodnotenie textu číslom (pravdepodobnosťou) podľa jeho príbuznosti s daným „trénovacím textom“. Špeciálne, pre $n = 1$ je takéto ohodnotenie totožné s prirodzeným ohodnotením frekvenčnou analýzou.

2.2.2 Hľadanie správnej permutácie

Nech c je šifrový text a p jemu zodpovedajúci otvorený text. Zadefinujme si množinu

$S = \{k(c)|\forall k \in K\}$. Permutáciu k nazveme *generujúcou* pre prvok $k(c)$. Určite $p \in S$, pretože množina K permutácií je uzavretá - čiže

$\forall k \in K : \exists k^{-1} \in K : \forall c : c = k^{-1}(k(c))$.

Prvky tejto množiny ohodnotíme pomocou n-gramového modelu. Tieto prvky sú vlastne reťazce textu (možné podoby otvoreného textu), ktoré ohodnocujeme pomocou funkcie P definovanej v 2.7. V mojom programe som funkciu P čiastočne pozmenil. Kým pôvodná pri výskyte gramu, ktorý sa nenachádzal v „tréningovom texte“, vrátila automaticky 0, ja som dal takýmto gramom ohodnotenie, ako by sa nachádzali v texte 0,1-krát. Čiže napr. pre text `DOMAQ` by pôvodná funkcia (pre 4-gramy) vrátila $P(\texttt{DOMAQ}) = P(\texttt{DOMA}) * P(\texttt{OMAQ}) = 0$, keď uvažujem, že gram `OMAQ` sa nenachádza v tréningovom texte. Moja funkcia vracia kladnú (ale veľmi malú hodnotu) pre takéto texty. Týmto kompenzujem možný výskyt nepravdepodobných gramov (napríklad mená alebo cudzie slová) v otvorenom texte.

Vzniknutý priestor sa snažíme najefektívnejšie prehľadávať a hľadať v ňom maximá pre spomínané ohodnotenie, keďže práve maximá sú najpravdepodobnejšie podoby otvoreného textu. Prehľadávanie všetkých prvkov je zaručene neefektívne, lebo $|S| = |K| \approx 2^{88,38}$.

Definujme si na tomto priestore **susednosť** takto: dva prvky množiny S sú susedné práve vtedy a len vtedy, ak ich generujúce permutácie k sa líšia práve jednou transpozíciou.

V takto definovanom priestore cestovanie po susedných rastúcich ohodnoteniach končí v lokálnych maximách. Jedným z týchto lokálnych maxím je aj hľadané globálne maximum. Označme g globálne maximum a L množinu lokálnych maxím. Platí $g \in L$ a $\forall s \in S : P(g) \geq P(s)$, kde $P(x)$ je ohodnotenie pomocou n-gramového modelu.

Ďalej predpokladáme, že lokálnych maxím nie je veľa a najväčšiu pravdepodobnosť máme skončiť prehľadávanie v najvyššie hodnotených maximách. Tento predpoklad nebude v skutočnosti presný, ale z vlastností jazyka a zo stavby n-gramov vyplývajú podobné výsledky. Dole popísaný algoritmus, založený na tomto predpoklade v majoritnej väčšine funguje, takže môžeme považovať predpoklad za pre naše účely postačujúci.

2.2.3 Algoritmus

Algoritmus pozostáva z niekoľkých krokov:

1. inicializácia a načítanie „tréningových textov“
2. n-násobné opakovanie:
   (a) vygenerovanie náhodnej permutácie k
   (b) prehľadanie všetkých susedov $k(c)$
      i. ak existuje k_s také, že $k_s(c)$ je susedný s $k(c)$ a $P(k_s(c)) > P(k(c))$, tak $k := k_s$ a goto 2(b)
      ii. ak neexistuje - zapísanie výsledku
3. výpis riešení

Algoritmus nájde pre každé opakovanie cyklu jedno lokálne maximum. Samozrejme, nie je zaručené, že tieto lokálne extrémy nebudú rovnaké. A taktiež nepoznáme hodnotu $|L|$ ani $P(g)$, aby sme vedeli identifikovať nájdené globálne maximum. Z toho dôvodu považujem hodnotu n za vstup a očakávam, že s väčším n sa dosiahnu štatisticky lepšie výsledky (samozrejme, po nejakú hranicu). Túto hodnotu budem v ďalšom texte nazývať „počet skúsených permutácií“.

V práci som použil tento algoritmus s malými zmenami.
Prvá zmena: po nájdení vhodnej susednej permutácie neukončujem cyklus,

ale si len zapamätám, že mám prehľadávať ďalej a cyklus dokončím. Týmto sa nezameriavam zo začiatku len na jednu časť abecedy, ale uskutočňujem výmeny znakov v permutáciách rovnomerne po celej abecede.
Druhou zmenou je zmena funkcie P pri neskorších testoch. Touto zmenou sa snažím nájsť lepšie riešenie ako s použitím iba n-gramov.

Tento prístup výrazne uľahčuje hľadanie správnych riešení.

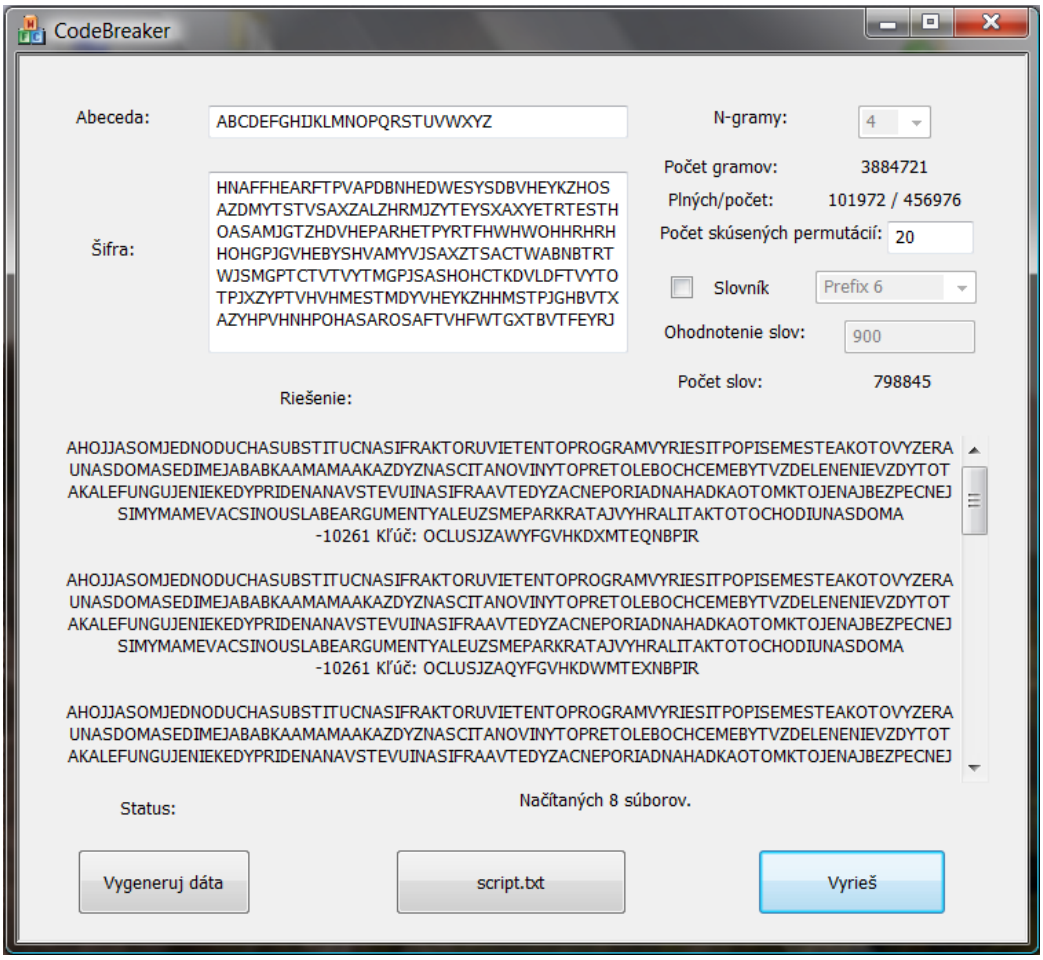
2.3 Moja implementácia

V tejto časti opíšem program, ktorý som vytvoril a ktorým som vypočítal všetky výsledky spomenuté v kapitole 3. Program som vytvoril v jazyku C++ a vývojovom prostredí Microsoft Visual Studio 2005. Použil som MFC knižnice pre GUI objekty. Logicky je rozdelený na dve časti - dátovú a používateľské rozhranie. Na načítavanie zo súboru s diakritikou som použil triedu CStdioFileEx od Davida Pritcharda stiahnutú zo stránky `www.codeproject.com`. Pre spustenie programu na počítači bez Visual Studia 2005 je potrebné mať nainštalovaný Visual Studio 2005 Redistributable SP1.

2.3.1 Používateľské rozhranie

Úlohou tejto práce nebolo vytvoriť zložitý program na riešenie šifier, ale efektívny a jednoduchý na používanie. Preto používateľské rozhranie je veľmi jednoduché a nebudem ho podrobne popisovať. Skladá sa z nastavovacích prvkov a 3 tlačítok - **Vygeneruj dáta**, **script.txt** a **Vyrieš**. Pre nastavenia používam rovnaké názvoslovie ako v bakalárskej práci, takže by mali byť zrozumiteľné.

Tlačítko **Vygeneruj dáta** zoberie adresár `inputs` z umiestnenia, kde sa nachádza program a načíta z neho všetky súbory podľa nastavení.



Obr. 2.1: Ukážka užívateľského rozhrania

Tlačítko **script.txt** spustí skriptový súbor `script.txt` tiež z umiestnenia, kde sa nachádza program a vykoná všetky príkazy v ňom.

Tlačítko **Vyrieš** vyreši zadanú jednoduchú substitučnú šifru a najlepších 20 riešení vypíše na obrazovku. Ku každému riešeniu vypíše pravdepodobný otvorený text, ohodnotenie tohto textu podľa funkcie P a použitý kľúč.

Všetky tlačítka vydajú po skončení práce krátky zvukový signál.

2.3.2 Dátová časť

Dátová časť programu zahŕňa všetky kroky samotného riešenia šifry od načítania vstupov po výpis do súboru. Skladá sa z 3 tried - CData, CPerm a CSlovnik.

CData

Táto trieda zaobaľuje celý proces riešenia šifry. Jej inštanciu vytváram len raz, a to pri štarte programu. V krátkosti popíšem jej hlavné metódy približne v poradí spracovania.

```
void SetupLetters(CString alphabet);
```

Nastaví pracovnú abecedu na *alphabet* a spraví mapovanie na túto abecedu. Takže keď v pracovnej abecede nie je napr. **ä**, bude sa namiesto neho ukladať **A**. Mapovanie je napevno v zdrojovom kóde, zahŕňa však všetky písmenká slovenskej, nemeckej a anglickej abecedy.

```
int LoadFromFile(CString file, int ngrams);
int LoadFromDir(CString dir, int ngrams);
```

Zo súboru *file* alebo cesty so súborovým filtrom *dir* načíta n-gramy, konkrétne pre $n = ngrams$, ak už predtým neboli načítavané gramy s inou dĺžkou. Zároveň načítava slová do stromovej štruktúry. Nájdené súbory by mali byť v kódovaní UTF-8.

```
CString Solve(CString str, int many, int slovnik);
```

Reprezentuje algoritmus popísaný v 2.2.3. Vyrieši šifru *str*, s *many* skúsenými permutáciami. Hodnota *slovnik* predstavuje hodnotu slova, ktorá sa posúva do `Language`.

```
double Language(CString str, CPerm *perm, int *wrds, int value);
double GramProbability(int index);
```

Programová realizácie funkcie P. Funkcia `GramProbability` vracia pravdepodobnosť výskytu určitého gramu s indexom *index*. Používa ju funkcia `Language`, ktorá dostáva na vstupe šifrový text - *str*, permutáciu - *perm* a v prípade, že chceme zohľadňovať aj slová, dostáva navyše hodnotu slova vo *value* a vie vrátiť počet nájdených slov (prefixov) do *wrds*. Návratová hodnota je logaritmus pravdepodobnosti textu vynásobený 1000 a znormovaný na počet písmen šifry.

Nachádza sa tu ešte pár pomocných metód, ktoré nebudem spomínať.

CSlovnik

Reprezentuje prvok stromovej štruktúry, v ktorej sú uložené slová. Pamätá si svojich nasledovníkov a slovo, ktoré reprezentuje, ako aj počet týchto slov.

CPerm

Zaobaľuje permutáciu a poskytuje metódy na efektívnu prácu s ňou a textom. Je vnútorne závislá na pracovnej abecede iba počtom písmen, čiže metódy na prácu so šifrou majú aj ďalší parameter, ktorým je táto pracovná abeceda. Jej hlavné metódy sú:

```
void Generate();
```

Vygeneruje náhodnú permutáciu.

```
void swap(int a, int b);
```

Vymení si hodnoty s indexmi *a* a *b*. Stane sa z nej týmto iná, susedná permutácia k pôvodnej.

```
CString GetString(CString str, CString active_l);
```

Vráti text *str* zašifrovaný touto permutáciou. Pracovná abeceda je *active_l*.

Kapitola 3

Výsledky

Táto kapitola je zameraná na zhrnutie všetkých výsledkov, ktoré sa mi podarilo s programom dosiahnuť. V závere spomeniem aj možné zlepšenia. Všetky výsledky spomenuté tu a niektoré ďalšie sú prehľadne spracované v súbore `výsledky.xls`, ktorý sa nachádza v prílohách. V tomto súbore sú aj všetky tabuľky k uvedeným grafom. V prílohách sú tiež výstupné súbory môjho programu, z ktorých sú tieto výsledky generované.

3.1 Vstupy a formátovanie

Vstupné texty som vytvoril z mnohých slovenských elektronických kníh, hlavne zo stránok `www.palmknihy.cz`, `www.klasici.sk` a Zlatého fondu SME (`www.zlatyfond.sme.sk`). Takto získaný korpus som potom rozdelil na 2 časti - „tréningový text“ a text, z ktorého som generoval náhodné šifry. Tým pádom som zaistil nezávislosť korpusu na otvorenom texte. Môj tréningový text mal približne 25 MB. Súbor, z ktorého som vyberal náhodné šifry mal asi 1,5 MB.

Ako šifrujúcu permutáciu som zvolil identitu, keďže takto sa mi najjednoduchšie generovali šifry a na výsledky to nemalo vplyv - algoritmus aj

tak volí náhodné permutácie.

Pracovnú abecedu som používal `ABCDEFGHIJKLMNOPQRSTUVWXYZ`, čiže základnú anglickú abecedu. Tým pádom som ignoroval výhodu slovenského jazyka vo väčšom počte písmen a unikátnosti použitia niektorých písmenok ako **ä**, **ô** a pod. Tiež písmenká Q, W, X a vo väčšine prípadov aj F a G boli navyše (keďže texty, s ktorými som pracoval mali dĺžku do 500 znakov a stredná hodnota výskytu týchto písmen je $(E('G') \approx E('F') = n * p_F = n * 0,002)$ menšia ako jedna pre texty kratšie ako 500 znakov), ale táto skutočnosť nijako neovplyvnila výsledky. Znak pre medzeru tiež nie je v základnej abecede, takže medzery som bral ako neznámy znak a preskakoval ich spolu s inou interpunkciou. Dôvod za touto neveľmi vhodne zvolenou abecedou bola istá všeobecnosť výsledkov riešenia. V e-mailovej, SMS a postupom času aj v mnohých iných spôsoboch komunikácie sa prestávajú požívať mäkčene, dĺžne a niekedy aj medzery. Tiež šifry bez medzier sú ťažšie na prelomenie, keďže nemôžeme použiť mapovanie slov, resp. slovných druhov podľa dĺžky slova, a preto sa častejšie používajú. Niekedy sa dokonca do šifier pre prehľadnosť vkladajú „falošné“ medzery po 5 znakoch a šifry sa píšu len s použitím veľkých písmen. Z tohto a podobných dôvodov ako vyššie som sa vyhol aj rozlišovaniu medzi veľkými a malými písmenami a všetky znaky som prevádzal na veľké.

Samozrejme, na vytváranie slovníka som medzery (a interpunkciu) využíval ako oddeľovače slov, ale inak platili rovnaké podmienky aj pre slová.

Počet zapamätaných najlepších riešení som vo všetkých testoch nastavil na 20. Ak bolo medzi týmito dvadsiatimi riešeniami správne riešenie (podotýkam, že to nemusela byť identita, kvôli mnohým chýbajúcim znakom), test som vyhodnotil ako úspešný s nula nesprávne určenými znakmi. Inak som vybral riešenie s najmenej nesprávnymi znakmi a toto som ďalej spracovával. Týmto som získal podklady na 2 grafy - graf percentuálnej správnosti riešenia a graf percentuálnej zhody nájdeného textu s otvoreným textom.

Tento prístup mi pomohol vysporiadať sa s riešeniami, ktoré boli vyriešené správne okrem jednej zlej transpozície - výskyt týchto znakov bol značne nízky (väčšinou len 1 znak na celý text) a obyčajne to boli práve najmenej časté znaky `F` a `G`. Kým v prvom grafe sa síce toto „skoro“ správne riešenie ukáže ako nesprávne, v druhom sa jeho percentálna úspešnosť prakticky vyrovná správnemu riešeniu. Pri výsledkoch uvádzam zvyčajne oba tieto grafy.

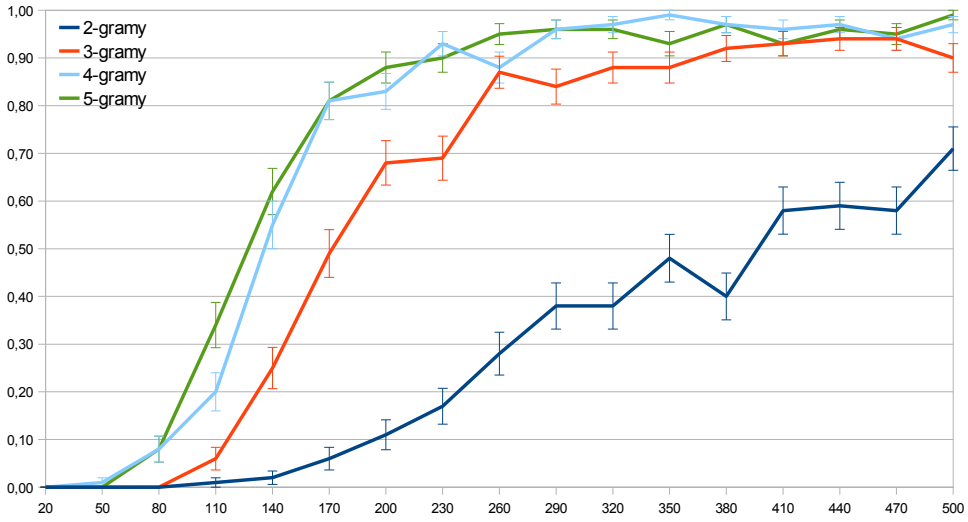
3.2 Výsledky pre rozdielne n-gramy

Prvý test mal za úlohy vyskúšať ako sa správajú výsledky pre rôzne n-gramy. Výsledky som spracoval pre $n = 2, 3, 4$ a 5, keďže pre uchovanie všetkých gramov dĺžky n treba $4*26^n$ bytov operačnej pamäte pri používanej základnej abecede. Pre $n = 5$ je to cca. 45 MB, ale pri $n = 6$ to vychádza už nad 1 GB. Taktiež spraviť výsledky pre $n = 1$ považujem za nepotrebné, keďže tieto sú ohraničené zhora možnosťami frekvenčnej analýzy a teda zďaleka nemôžu dosahovať výsledky ostatných n-gramov.

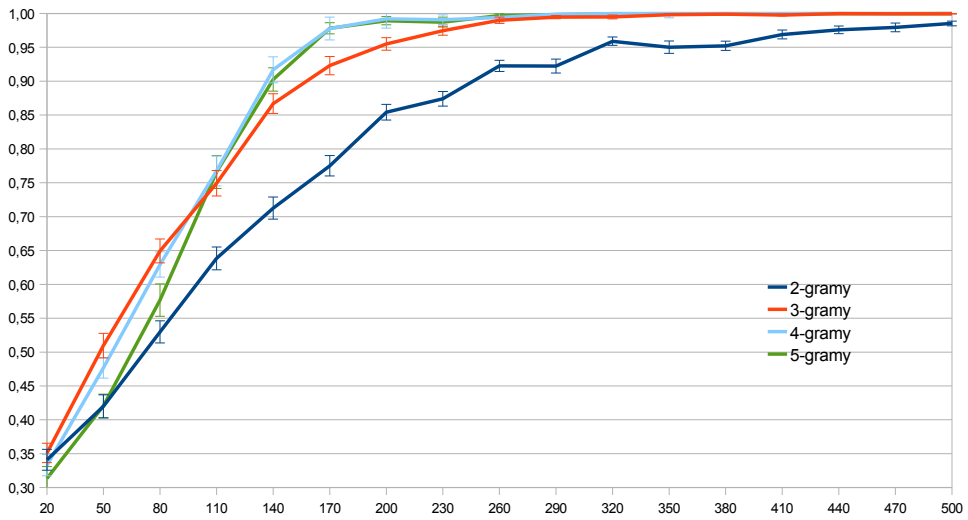
Výsledky som generoval pre texty dĺžky od 20 do 500 s krokom 30. Pre každý krok som vygeneroval 100 náhodných vstupov a tie som nechal riešiť. Týmto som získal vcelku hodnovernú presnosť riešenia. Výsledky som generoval pre 50 skúsených permutácií. Prehľadné zhrnutie týchto výsledkov ponúkajú grafy 3.1 a 3.2. V grafoch je možné vidieť aj strednú odchýlku pre každý nameraný výsledok.

Teoreticky by mali pre vyššie čísla n vychádzať vyššie hodnoty správnych riešení. V grafoch je však možné vidieť, že už 5-gramy zaznamenali len minimálne zlepšenie. Preto som s väčšími ako 5-gramami nepracoval.

Zaujímavým výsledkom je tiež fakt, že pre malé texty (do 110 znakov) sú úspešnejšie v počte dosadených správnych písmen (graf 3.2) menšie gramy, konkrétne 3-gramy. 2-gramy sa ukazujú ako neporovnateľne horšie od ostatných. Túto skutočnosť spôsobuje veľký význam jedného gramu v takomto



Obr. 3.1: Graf celkovej úspešnosti riešenia pre rôzne gramy



Obr. 3.2: Graf percentuálnej úspešnosti znakov pre rôzne gramy

texte.

Ďalej budem uvažovať 4-gramy a snažiť sa tieto ďalej zlepšiť, resp. porovnať pre rôzne vstupné parametre.

3.3 Výsledky pre rozdielny počet skúsených permutácií

V tomto teste som sledoval správanie úspešnosti výsledkov pre rozdielny počet skúsených permutácií. Cieľom bolo určiť optimálnu hodnotu, pre ktorú program nájde dobré výsledky, ale nezaberie veľa času. Tento test som robil na 4-gramoch a pre počty skúsených permutácií $n = 1, 10, 20, 30, 40, 50$ a 60. Okrem toho je tento test totožný s testom z 3.2. Z výsledných grafov 3.3 a 3.4 je možné nahliadnuť, že po hodnote 30 už úspešnosť riešenia signifikantne nerastie. Pre prehľadnosť v týchto grafoch neudávam štandardné odchylky.

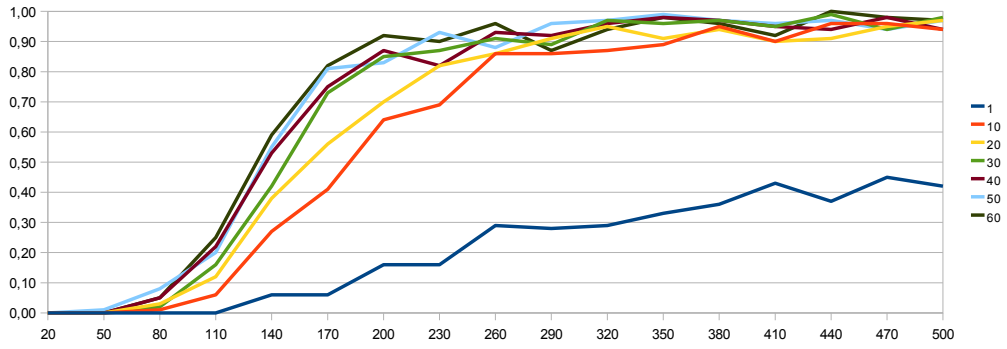
3.4 Slovník

Jedným z logicky možných zlepšení je aplikovanie slovníka. Niektoré výsledné texty totiž mali tendenciu vyzerať ako správny slovenský text (korektne sa striedali samohlásky a spoluhlásky, text bol čitateľný a často sa vyskytujúce znaky - a, o, e, s, p, a pod. tu boli hojne zastúpené, často dokonca aj na správnych miestach), ale po prečítaní tohoto textu sme našli len veľmi málo zmysluplných slov. Napríklad:

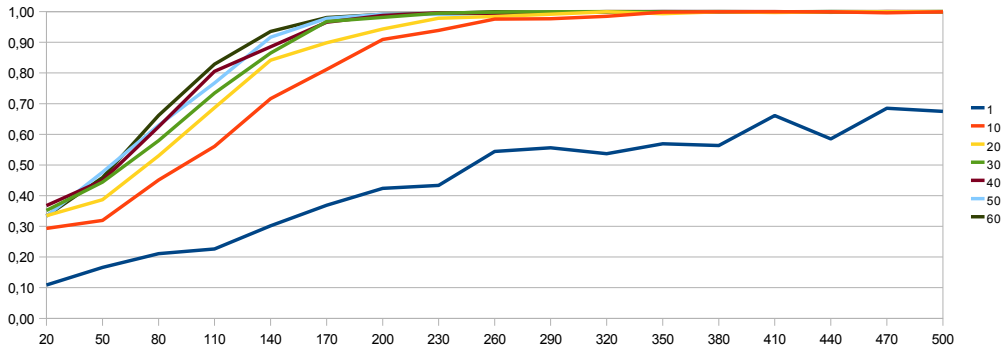
Nesprávne interpretovaný text:

```
KADVOCHCOJDUKSOTZYCHSENPUJUCTVOIRUMEJVECCEJMYSUJTE
IMNEDENMULEJSADOLAVOMIBOJTAZTURAMIBOVALKEACTRPEOTK
UCEACHSALOLDANOSZNESNDYTECHUZVOLPRESTIPENAUMLANLI
```

Správny text (s doplnenými medzerami pre zvýšenie čitateľnosti):



Obr. 3.3: Graf celkovej úspešnosti riešenia pre rôzny počet skúsených permutácií (4-gramy)



Obr. 3.4: Graf percentuálnej úspešnosti znakov pre rôzny počet skúsených permutácií (4-gramy)

```
VAZNE CHCEM ZO VSETKYCH SIL POMOCT NEUROBIM NIC CIM BY SOM TI
UBLIZIL BOJIM SA ZE JA NEBUDEM TA KTORA BUDE NAJVIAC TRPIET
V OCIACH SA JEJ ZALESKLI SLZY TICHO K NEJ PRISTUPIL A OBJAL JU
```

Pripomeniem, že tento text mal písmená `A, C, H, P, R ,S ,T ,Y` na správnom mieste.

3.4.1 Použitie slovníka

Možnosti zakomponovania slovníka do programu by sme vedeli rozdeliť na základe miesta v programe na:

1. rozšírenie funkcie P
2. prerátanie jednotlivých získaných výsledkov každého zbehnutia cyklu

a na základe hľadaného prvku:

1. hľadanie celých slov dlhších ako x
2. hľadanie prefixov dĺžky x
3. hľadanie najčastejších slov jazyka (alebo najčastejších prefixov)

a mnohé iné. Týchto možností je veľa a skoro všetky niekoľkonásobne zvyšujú časovú náročnosť tohto program. Kvôli tomu som bol nútený vybrať si jednu z možností a venovať sa tej.

Ako najrozumnejšia mi prišla možnosť rozšírenie funkcie P, keďže táto je zodpovedná za hodnotenie výsledkov a hľadanie prefixov dĺžky 6.

Prerátavanie jednotlivých výsledkov by v mojom prípade neprinieslo želané výsledky, lebo jednotlivých výsledkov je porovnateľne veľa s vypisovanými výsledkami, čiže by sa iba dosiahlo iné poradie výsledkov. Takisto hľadanie slov rovnako dlhých a kratších ako je dĺžka gramu je zbytočná - túto informáciu nám poskytuje sám gram. Hľadanie celých slov som zavrhol

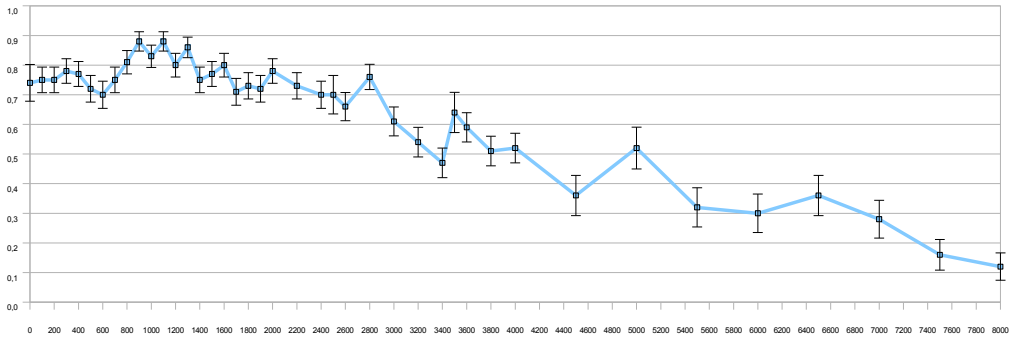
kvôli skloňovaniu. Za dĺžku hľadaného prefixu som nakoniec vybral 6. Slová s dĺžkou 6 a viac sa ešte v slovenčine bežne vyskytujú, ale už je to o dosť viac ako dĺžka n-gramu a teda je táto informácia schopná programu napovedať niečo o povahe textu.

3.4.2 Ohodnotenie slova

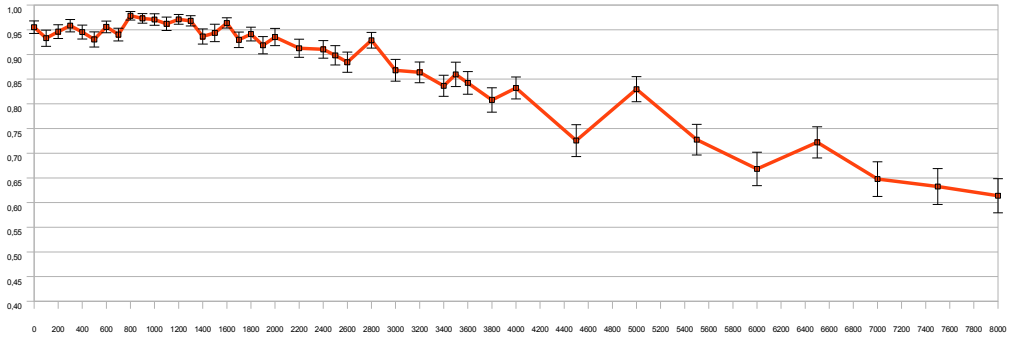
Implementáciu funkcie P som zmenil nasledovne:

$$P_n(s) = P_s(s) + k * Pre_6(s) \tag{3.1}$$

kde P_s je pôvodná funkcia P, P_n je nová a $Pre_6(s)$ je počet prefixov dĺžky 6 v texte s, pričom som hľadal len neprekrývajúce sa prefixy. Paramter ohodnotenia slov je k. Optimálnu veľkosť tohoto parametra som určil experimentálne. Porovnal som výsledky pre texty s pevnou dĺžkou 200 a pre rôzne hodnoty paramentra k. Začal som s rozptylom 0 - 20000, krokom 500 a 50 generovanými šiframi. Z tohto som postupne robil priblíženia na hľadaný interval, posledné priblíženie malo krok 100 a 100 generovaných šifier. Výsledky tohoto porovnania sú zhrnuté v grafoch 3.5 a 3.6. Z týchto grafov vidno, že so stúpajúcou hodnotou k najprv rastie úspešnosť riešenia, ale tá potom okolo hodnoty 1500 začne klesať. To je spôsobené nadmerným ohodnotením slov. Pri prehľadávaní takto ohodnoteného priestoru sa totiž môže stať, že program nájde permutáciu, ktorou vznikne síce nezmyselný text, ale náhodou má niekoľko zmysluplných prefixov. Tieto prefixy však zanikajú pri prechode na inú - zmysluplnejšiu permutáciu. Týmto sa stane táto v praxi horšia permutácia lepšie ohodnotenou. Z výsledkov vidno, že optimálna hodnota parametra k je niekde medzi hodnotami 800 a 1300. Ja som pre ďalšie testovanie zobral hodnotu 900 ako najlepšie riešenie.



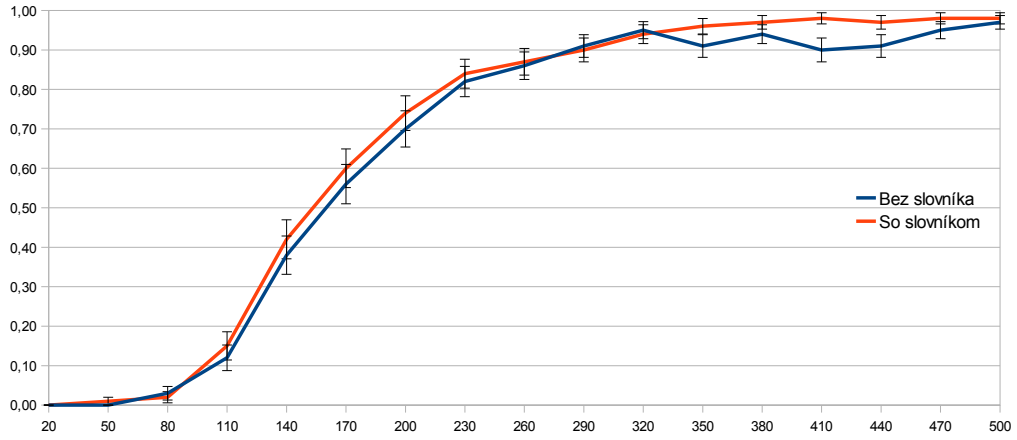
Obr. 3.5: Graf celkovej úspešnosti riešenia šifier s dĺžkou 200 a hľadaným prefixom dĺžky 6



Obr. 3.6: Graf percentuálnej úspešnosti znakov riešenia šifier s dĺžkou 200 a hľadaným prefixom dĺžky 6

3.4.3 Výsledky pre použitie slovníka

Pre túto metódu som spravil rovnaký test ako v 3.2, no iba s 20 skúsenými permutáciami. Dôvodom bola časová náročnosť tohoto testu. Výsledky som zhrnul do grafu 3.7, kde uvádzam aj referenčnú krivku pre riešenie bez použitia slovníka. Z tohto grafu vidno, že toto zlepšenie neprinieslo oveľa lepšie výsledky. To je spôsobené tým, že samotné 4-gramy už z veľkej časti informáciu o slovách v sebe obsahujú. Podarilo sa však čiastočne odstrániť problém zamieňania menej častých znakov. V grafe sa to prejaví ako vyhladenie a stály rast krivky. Pre tento výsledok uvádzam len graf celkovej úspešnosti.



Obr. 3.7: Graf celkovej úspešnosti riešenia pre 4-gramy s použitím slovníka

3.5 Časová zložitosť

Časovú zložitosť implementovaného programu by sme mohli rozpísať na dve čiastky - časovú zložitosť načítavania a inicializácie a časovú zložitosť samotného riešenia.

Časová zložitosť načítavania je priamo úmerná dĺžke „tréningového textu“. Načítanie korpusu do 10 MB pritom trvá do jednej minúty na terajšom štandardnom počítači.

Časová zložitosť riešenia je priamo úmerná dĺžke šifry a počte skúsených permutácií a pri implementácií slovníka aj priamo úmerná priemernej dĺžke slova v prirodzenom jazyku. Na štandardnom počítači sa pri 20 skúsených permutáciách spracuje asi 3000 znakov za minútu, pri použití slovníka asi 400. Pri jednej šifre je to teda zanedbateľný čas, no pri testovaní štatistickej správnosti som bol touto časovou zložitosťou obmedzovaný. Napríklad dostať výsledky pre graf 3.7 trvalo viac ako 20 hodín neustálej práce počítača.

3.6 Možné zlepšenia

Program mal problémy hlavne s kratšími textami, ktoré obsahovali nejaké netradičné slová (napríklad mená alebo cudzie slová).

Priestor definovaný v 2.2.2 by sa dal prehľadávať lepšie pomocou algoritmu simulovaného žíhania. Otázne je, či by za rovnaký čas dosiahol lepšie výsledky. Tento algoritmus totiž trvá dlhšie ako algoritmus použitý v tejto práci, ale s väčšou pravdepodobnosťou nájde globálne maximum na jedno spustenie. Neschopnosť nájsť riešenie na prvýkrát som pri mojom algoritme kompenzoval väčším počtom skúsených permutácií.

Ďalšie zlepšenie vidím vo zvýšení dĺžky základného gramu a tým aj vo väčšom základnom korpuse. Zvýšenie dĺžky gramu totiž so sebou prináša aj nároky na zväčšenie základného korpusu. Množstvo zmysluplných gramov rastie exponenciálne, čiže aj potrebný korpus by mal rásť exponenciálne od veľkosti gramu. Nedodržaním požadovanej veľkosti korpusu by vychádzali výsledky pre niektoré väčšie gramy horšie ako pre menšie gramy. Pri väčších gramoch sa potýkame ešte aj s pamäťovým problémom, ktorý by sa však dal odstrániť ukladaním gramov do inej štruktúry. V spomínanom programe

boli n-gramy uložené ako jedno pole. Toto pole bolo (hlavne u 4-gramov a 5-gramov) z väčšej časti prázdne, keďže väčšina n-gramov bola nezmyselná. Vytvorením stromovej štruktúry gramov by sa dala táto dĺžka zvýšiť. Taktiež by sa so stromovou štruktúrov dali vytvárať nielen gramy s pevnou dĺžkou, ale aj s pohyblivou. Napríklad často používané slovenské frázy by mali dlhšie (a lepšie hodnotené gramy), kým menej vyskytujúce sa gramy by mali malú dĺžku (napr. dlhé - `NACHADZA SA V`, krátke - `VRB`). Vytvorenie takejto štruktúry by bolo však podstatne náročnejšie a nemáme nijaké záruky, že výsledky budú naozaj lepšie.

Zlepšenie za pomoci slovníka má tiež mnohé obmeny, ktoré môžu viesť k výraznejšiemu zlepšeniu, ako v mojej implementácií. N-gramy sa však ukázali ako silný nástroj, ktorý už infomáciu o slovách v sebe časti obsahuje, takže tieto zlepšenia musíme brať s rezervou.

Záver

V tejto práci som sa venoval substitučným šifrám a ich riešeniu. Využitím n-gramov a slovníka som vystaval program riešiaci tieto šifry automaticky len za pomoci korpusu jazyka, ktorým je napísaný otvorený text šifry. Tento program som popísal, otestoval jeho výsledky na šifrových textoch v slovenskom jazyku a tieto spracoval do viacerých tabuliek a grafov. Navrhol som aj možné zlepšenia tejto metódy.

V našich dobách sa ešte stále môžeme stretnúť s jednoduchými substitučnými šiframi, napríklad na rôznych šifrovacích hrách a spomenutých medzikrokoch kryptoanalýzy zložitejšieho systému šifier. Niektoré historické substitučné šifry tiež ešte stále neboli vyriešené. Tento program zlomil substitučné šifry s pomerne krátkym šifrovým textom za veľmi krátku dobu. Tým poukazuje na bezpečnostný problém substitučnej šifry dokonca aj pri malých textoch a krátkej požadovanej výpočtovej bezpečnosti.

Prínos tejto práce vidím aj v poskytnutí nástroja na riešenie takýchto šifier, či už v spomínaných šifrovacích hrách alebo ako pomoc v zložitejších problémoch.

Dodatok

K práci prikladám aj CD s napáleným programom a zdrojovými kódmi. Súčasťou CD sú tiež súbory s výsledkami spomínané v kapitole 3 a balík Visual Studio 2005 Redistributable SP1 potrebný na spustenie tohoto programu.

Literatúra

[inf01] Federal information.
ADVANCED ENCRYPTION STANDARD (AES). Processing Standard Publication 197, Nov. 2001.

[oST] National Institute of Standards and Technology.
Security technology / Cryptografic Toolkit.
`http://csrc.nist.gov/groups/ST/toolkit/index.html` .

[Sta04] Martin Stanek. *Základy kryptológie*. 2004.
`http://www.dcs.fmph.uniba.sk/ stanek/crypto/main2.pdf` .

[Žub08] Tomáš Žubrietovský. *Automatická kryptoanalýza súčtu dvoch otvorených textov*. *Bakalárska práca*, 2008.

[Wika] Wikipedia. *N-gram*.
`http://en.wikipedia.org/wiki/N-gram` .

[Wikb] Wikipedia. *Substitučná šifra*.
`http://en.wikipedia.org/wiki/Substitution_cipher` .