Univerzita Komenského, Bratislava
Fakulta Matemetiky, Fyziky a Informatiky

Automatické dešifrovanie Fleissnerovej mriežky

Bakalárska práca

2012 **Michal Hajdin**

Univerzita Komenského, Bratislava

Fakulta Matematiky, Fyziky a Informatiky

Automatické dešifrovanie Fleissnerovej mriežky

Bakalárska práca

Študijný program: Informatika
Študijný odbor: 2508 Informatika
Školiace pracovisko: Katedra informatiky
Školiteľ: RNDr. Michal Forišek, PhD.

Bratislava, 2012 **Michal Hajdin**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Michal Hajdin
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Automatické dešifrovanie Fleissnerovej mriežky

Cieľ: Cieľom práce je preskúmať možné postupy vedúce k automatickému dešifrovaniu jednoduchých "klasických" šifier. Hlavný dôraz by mal byť kladený na dva typy metód: slovníkové metódy a metódy využívajúce frekvencie krátkych podreťazcov. Nasledujúcim cieľom práce je niekoľko vybraných metód implementovať a porovnať ich účinnosť pre šifru Fleissnerova otočná mriežka.

Vedúci: RNDr. Michal Forišek, PhD.
Katedra: FMFI.KI - Katedra informatiky

Dátum zadania: 26.10.2011

Dátum schválenia: 26.10.2011 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestne prehlasujem, že bakalársku prácu som vypracoval samostatne s použitím uvedenej literatúry a pod dohľadom môjho vedúceho práce.

. . . . . .

Abstrakt

Autor: Michal Hajdin
Názov práce: Automatické dešifrovanie Fleissnerovej mriežky
Škola: Univerzita Komenského v Bratislave
Fakulta: Fakulta Matematiky, Fyziky a Informatiky
Katedra: Katedra informatiky
Vedúci bakalárskej práce: RNDr. Michal Forišek, PhD.
Jún 2012, Bratislava

Táto bakalárska práca sa venuje automatickému dešifrovaniu klasického transpozičného šifrovacieho systému Fleissnerova otočná mriežka. Poukazuje na rôzne spôsoby ohodnocovania dešifrovaného textu a to slovníkové metódy a metódy využívajúce frekvencie krátkych podreťazcov. Obidve metódy boli implementované pri dvoch prístupoch prehľadávania priestoru kľúčov (útok hrubou silou a horolezecký algoritmus). Na výslednom programe boli prevedené testy týkajúce sa časovej zložitosti a úspešnosti automatického dešifrovania.

Kľúčové slová: Fleissnerova otočná mriežka, šifrovanie, ohodnocovanie textu, slovník, n-gramy, BruteForce, Hill-climbing

Abstract

Author: Michal Hajdin
Thesis title: Automatic decryption of Fleissner grid
University: Comenius University, Bratislava
Faculty: Faculty of Mathematics, Physics and Informatics
Department: Department of Computer Science
Advisor: RNDr. Michal Forišek, PhD.
June 2012, Bratislava

The subject of the thesis is the automatic decryption of classic transposition encryption system known as the Fleissners rotary grid. The work is focused on various modes of evaluation of decrypted text such as lexical methods and methods using frequencies of short substrings. Both methods were implemented in two modes of key space search (Brute Force and Hill Climbing algorithm). The final program was tested to the time complexity and success rate of automatic decryption.

Keywords: Fleissner grid, cryptology, text, lexical tree, n-gramms, BruteForce, Hill-climbing

Obsah

Úvod **1**

1 Prehľad problematiky **2**

1.1 Základné pojmy 2

1.2 Rozdelenie klasických šifier 4

1.2.1 Transpozičná šifra 5

1.3 Fleissnerova otočná mriežka 6

2 Prístupy automatického dešifrovania **8**

2.1 BruteForce 8

2.2 Hill-climbing 11

2.3 Iné optimalizačné metódy 15

3 Ohonotenie textu **17**

3.1 Ohonotenie textu slovníkom 17

3.2 Ohonotenie textu n-gramami 21

4 Praktické výsledky **25**

4.1 Využité technológie 25

4.2 Testovanie 27

Záver **30**

Literatúra **31**

Zoznam obrázkov

1.1 Šifrovanie pomocou Fleissnerovej mriežky a jej rotácie. 7

2.1 Návrh zápisu mriežky. 9

2.2 Rotácie jednotlivých otvorov mriežky. 9

2.3 Orientačné časy dešifrovania Fleissnerovej mriežky. 10

2.4 Niektoré podobné mriežky 4×4. 12

2.5 Niektoré podobné mriežky 6×6. 13

3.1 Lexikografický strom. 18

3.2 Najčastejšie trigramy v slovenčine. 22

4.1 Screenshot výslednej aplikácie. 26

4.2 Tabuľka BruteForce testov. 28

4.3 Tabuľka Hill-climbing testov. 29

Úvod

Klasické šifrovacie postupy boli v minulosti využívané ako spôsob utajovania dôležitých informácií. V súčasnosti sa používajú len ako ilustračný príklad alebo len tak zo zábavy pri rôznych šifrovacích hrách. Touto prácou by som chcel tento spôsob utajovania informácií trochu priblížiť a samozrejme aj uľahčiť čítanie otvoreného textu bez poznania šifrovacieho kľúča a proces dešifrovania textu zautomatizovať. Tým pádom je potrebné preskúmať možné postupy vedúce k automatickému dešifrovaniu transpozičného systému Fleissnerova otočná mriežka. Automatický dešifrovací proces sa skladá z dvoch častí. Jednou je generovanie kľúčov a dešifrovanie a druhou je výber správneho textu z tých dešifrovaných. Na obidve časti existuje viacero prístupov, o ktorých si v tejto práci niečo povieme. Niektoré z nich boli vybrané na podrobnejšie preskúmanie a následnú implementáciu do aplikácie umožňujúcej využiť tieto poznatky v praxi. Bakalárska práca sa skladá zo štyroch kapitol.
V prvej kapitole sú vysvetlené pojmy používané v oblasti kryptológie, ďalej obsahuje rozdelenie šifrovacích systémov. Nasleduje podrobnejší prehľad transpozičných šifier a v závere je popísaný systém Fleissnerovej otočnej mriežky.

Druhá kapitola sa podrobnejšie zaoberá prístupmi automatického dešifrovania textu. Obsahuje podrobnejšie rozobrané prístupy BruteForce a Hill-climbing aplikované na otočnú mriežku a stručný popis ich implementácie. Nasleduje ešte prehľad iných algoritmov ako genetické a evolučné algoritmy a simulované žíhanie.

V tretej kapitole sa venujeme ohodnocovaniu textu. Preskúmame podrobnejšie slovníkovú a n-gramovú metódu a rozoberieme ich výhody prípadne nedostatky a tvorbu vlastnej štatistiky.

Záverečná kapitola obsahuje informácie o tvorbe výslednej aplikácie a použitých technológiách. Za tým nasledujú výsledky praktických testov.

Kapitola 1

Prehľad problematiky

V tejto kapitole sa oboznámime so základnými pojmami, s ktorými budeme pracovať, a ktoré sa týkajú problematiky kryptológie. Dozvieme sa o rozdelení šifrovacích systémov. Ďalej si priblížime transpozičné šifry a ich rôzne variácie a obmeny. V záverečnej časti kapitoly si povieme ako šifra Fleissnerova otočná mriežka funguje. Dozvieme sa, ako a kedy sa prvýkrát objavila na verejnosti, ale aj kde a za akých okolností bola používaná v praxi.

1.1 Základné pojmy

Potreba ochrany a utajovania informácií existuje pravdepodobne už od vzniku ľudstva. Informácia je cenná vec pre jej vlastníka a v rôznych situáciách mu zabezpečuje výhodu. Ako každú cennú vec, tak aj informácie je treba chrániť pred nepovolanými, prípadne ju vedieť utajiť. To by nebol až taký problém. Ten však nastáva, keď sa chceme o informáciu s niekým podeliť. Vtedy musí správa prejsť komunikačným kanálom, kde je náchylná na odcudzenie, prípadne jej modifikáciu. Na jej bezpečný prenos existujú rôzne metódy. Prvou je utajenie samotnej existencie správy v inej bezvýznamnej správe. Touto oblasťou sa zaoberá **steganografia**. Druhou je fyzicky zabezpečiť dostatočnú ochranu informácie (SBS a iné bezpečnostné služby). Treťou metódou je uvedenie informácie do stavu, keď napriek tomu, že ju nepriateľ získa, je pre neho nezrozumiteľná. Touto metódou sa zaoberá **kryptológia**. Časťou kryptológie riešiacou návrhy vhodných šifrovacích systémov s následným zabezpečením dôvernosti, integrity, pôvodu správy a jej autentizácie je **kryptografia**. Na druhej

strane je **kryptoanalýza**, ktorá rozoberá jednotlivé systémy a ich bezpečnosť a odolnosť z hľadiska narušiteľa. Snaží sa rozšifrovať informáciu a preniknúť do systému.

Šifrová abeceda označovaná aj Σ je množina znakov, z ktorých sa skladá otvorený a zašifrovaný text. V nasledujúcom texte pod pojmom abeceda budeme chápať klasickú 26 znakovú medzinárodnú abecedu bez diakritiky. **Otvorený text** (OT) je ľubovoľná správa v pôvodnom nezašifrovanom tvare, ktorú máme v pláne zašifrovať.

Šifrovaný text (ŠT) je výsledkom procesu šifrovania. Je to pôvodná informácia v tvare, kedy aj po jej prípadnom odchytení nepriateľom si zachováva dôvernosť informácie. To však pri väčšine systémov platí len pri ideálnych podmienkach. **Šifrovací systém** je dvojica zobrazení/algoritmov (šifrovací (e_k)a dešifrovací(d_k)) závislých na tajnom parametri k.

$$e_k : \text{OT} \longrightarrow \text{ŠT}$$
$$d_k : \text{ŠT} \longrightarrow \text{OT}$$

Vyššie spomínané zobrazenia d_k a e_k nemusia byť nutne bijektívne. Musia však pre ľubovolný kľúč k a hocijaký vstupný text x spĺňať vlastnosť, že po dešifrovaní ŠT dostaneme text rovnaký, ako bol pôvodný OT

$$d_k(e_k(x)) = x \quad (x \in \Sigma^*)$$

Parameter k sa nazýva **šifrovací kľúč**. Kľúč k si volíme z množiny K všetkých prípustných kľúčov a je nezávislý na OT. **Šifrovanie** je proces transformácie OT na ŠT, ktoré je realizované ako postupnosť jednoduchých krokov (algoritmus).

Dešifrovanie je opačný proces k šifrovaniu, pri ktorom sa príjemca zašifrovanej správy snaží o zrekonštruovanie pôvodného OT a dostať ŠT do čitateľnej a zrozumiteľnej podoby. Býva prevedené buď tým istým algoritmom a kľúčom ako bolo šifrované (napr. šifrovací systém preklopenie textu odzadu), alebo len mierne modifikovaným algoritmom (napr. Caesarova šifra), alebo je rozdielny aj kľúč aj dešifrovací algoritmus. O príjemcovi predpokladáme, že má kľúč a pozná použitý šifrovací algoritmus, pripadne všetky potrebné informácie na úspešné dešifrovanie. Ak ŠT dostane neželaný príjemca/ narušiteľ/ kryptoanalytik, ktorý nemá všetky potrebné informácie, ide

už o lúštenie. Pri úspešnom vylúštení hovoríme, že šifra bola vylúštená. Pokiaľ sa to však týka celého šifrovacieho systému bez ohľadu na použitý kľúč, tak bola prelomená(čiastočne prelomená, ak to platí len za istých podmienok). Lúštenie ŠT môže prebiehať rôznymi spôsobmi v závislosti od dostupných informácií :

- Poznáme len jeden, prípadne niekoľko ŠT (Ciphertext-only attack).
- Poznáme ŠT a k nemu príslušný OT (Known-plaintext attack).
- Máme možnosť zašifrovať vlastné OT (Chosen-plaintext attack).
- Máme možnosť dešifrovať vlastné ŠT (Chosen-ciphertext attack).
- Máme možnosť zvoliť vlastný kľúč (Chosen-key attack).
- Máme možnosť získať informácie z iného zdroja (Side channel attack).

Tieto spôsoby sú zoradené z pohľadu kryptoanalytika od najťažšie prelomiteľného po najľahšie. Ďalšou dôležitou informáciou je jazyk, v ktorom bol OT napísaný a to, či poznáme použitý spôsob šifrovania.

1.2 Rozdelenie klasických šifier

Šifrovacie systémy sa vyvíjali postupne od dávnych čias. Postupne sa nachádzali ich slabiny a tie sa odstraňovali a vznikali stále nové a lepšie. V súčasnosti ich môžeme rozdeliť asi takto:

- Klasické
  - Substitučné
    - Monoalfabetické
    - Homofónne
    - Polygramové
    - Polyalfabetické
  - Transpozičné

- Symetrické
  - Prúdové
  - Blokové
- Asymetrické

1.2.1 Transpozičná šifra

Transpozičná šifra je založená na ponechaní znakov z pôvodného OT a len ich rôzne preusporiada v ŠT. Zachováva frekvenciu jednotlivých znakov OT a ŠT. Preto je použitie frekvenčnej analýzy priamo na šifru bezvýznamné a nepomôže nám pri lúštení tak, ako pri šifre substitučnej. Jedinú informáciu čo nám poskytne, je možnosť zistiť jazyk, v ktorom bol OT napísaný. Jednotlivé transpozície môžu závisieť len od použitého šifrovacieho algoritmu alebo sú závislé aj od kľúča. S tými najjednoduchšími sa väčšina stretla už v detstve na základných školách pri prvom kontakte s kryptografiou. Napríklad pri posielaní si tajných správ počas vyučovacej hodiny. S tými zložitejšími sa väčšina ľudí pravdepodobne nestretla.

Najjednoduchšou transpozičnou šifrou je asi text napísaný odzadu. Ďalšie možnosti transpozície bez kľúča sú napríklad vyberať písmená na striedačku zo začiatku a konca OT alebo napísať text do štvorcovej špirály a čítať po riadkoch/stĺpcoch. Existuje ešte veľké množstvo, líšiace sa v drobných alebo aj zložitejších obmenách.

Ďalšou šifrou je **Cardanova mriežka**, ktorá nie je úplne čistou transpozíciou. Ide o to, že sa podľa nejakej mriežky(kľúča) vyberú písmená na pozíciách otvorov v mriežke a tie tvoria OT, pričom ostatné nevyužité písmená majú vlastne úlohu klamačov. Preto neplatí klasické, že veľkosť OT = veľkosti ŠT, ale OT $<$ ŠT. Ak je napr. ŠT 100 stranová kniha a OT je text dĺžky rádovo 10 písmen, tak je táto šifra bez poznania kľúča nerozlúštiteľná. Tu však, ako pri väčšine jednoduchých, ale nerozlúštiteľných šifrách, nastáva problém v bezpečnej distribúcií kľúča. A aj to, aby sa tým istým kľúčom nešifrovalo viackrát za sebou, lebo tým by sa šance na úspešné vylúštenie rapídne zvýšili. Slabosť šifrovacieho systému na viacnásobné použitie kľúča je známa pod pojmom *Two-time pad* problém.

Nasledujúce zložitejšie transpozičné šifry sú napríklad napísanie OT do tabuľky pevnej šírky(kľúča) a potom čítanie po stĺpcoch. Pri tejto šifre je aj pre dlhšie OT problém v malej veľkosti priestoru kľúčov κ. Túto šifru používali už starom Grécku. Keď navinuli dlhý pás papiera/plátna na tyč istého polomeru a správu napísali pozdĺž tyče, po rozvinutí pásu dostali ŠT. Vylepšenie predchádzajúcej šifry môže spočívať v tom, že kľúč pozostáva z dvoch častí. Jednou je šírka tabuľky a druhou je permutácia čísel $1-n$, podľa ktorej n-tice riadkov spermutujeme presne podľa kľúča. Samozrejme sa to dá symetricky vymeniť a písať do stĺpcov a čítať po riadkoch.

Iné vylepšenie spočíva vo viacnásobnom použití daného šifrového systému,

$$\text{OT} \longrightarrow \text{ŠT1} \longrightarrow \text{ŠT}$$

čo však v konečnom dôsledku zase vedie len k inej(zložitejšej) transpozícií.

1.3 Fleissnerova otočná mriežka

Fleissnerova otočná mriežka je zdokonalením vyššie spomínanej Cardanovej mriežky s využitím otočenia štyroch symetrických častí vo štvorci. Toto otáčanie bolo prvýkrát použité koncom 18. storočia. Ako sa neskôr ukázalo, otočnú mriežku používali Hindenburg, Martens, de Presse, Kluber a iní ešte pred rakúskym plukovníkom Edouardom Fleissner von Wostrowitz(1825-1888). On ju však ako prvý podrobnejšie popísal vo svojej knihe *Handbuch der Kryptographie*, ktorá bola vydaná v roku 1881. Keďže sa tento šifrovací systém takto vďaka nemu spopularizoval, dostal po ňom aj pomenovanie. Pár rokov po vydaní knihy zakomponoval otočnú mriežku do svojej novely aj Jules Verne.

Počas prvej svetovej vojny bol istý čas tento šifrovací systém používaný aj v praxi nemeckou armádou. Jednotlivé mriežky mali dokonca pre rôzne veľkosti svoje mená: ANNA(5), BERTA(6), CLARA(7), DORA(8), EMIL(9), FRANZ(10). Vzhľadom na to, že takýmito mriežkami mohli šifrovať len krátke správy a pri takýchto veľkostiach aké mali, sa to dá aj manuálne dešifrovať v rozumnom čase, prešli po pár týždňoch na iné šifrovacie systémy.

Základom šifrovania je štvorcová tabuľka a rovnako veľká mriežka(kľúč) s vyrezanými otvormi. Šifrovanie prebieha tak, že sa priloží mriežka na tabuľku, v ktorej je klasicky napísaný OT a po jednom sa vypisujú znaky z voľných otvorov do ŠT. Po ich vypísaní sa mriežka otočí o 90 ° v dohodnutom smere. Ak bude ďalej v texte spomenutá rotácia, tak bude myslená vždy v smere hodinových ručičiek. Takéto otočenie vieme spraviť 3×. Po štvrtom otočení by sme dostali už pôvodnú mriežku. Každá mriežka má 4 polohy. Otvory nie sú robené ľubovoľne, ale tak, aby sa po vypísaní zo všetkých štyroch polôh mriežky použilo každé jej políčko práve raz. Zvolíme si prvé políčko a vyčiarkneme zvyšné tri, ktoré sú na pozíciach prvého políčka pri jeho rotácií. Následne pokračujeme výberom ešte neoznačeného políčka. Tento postup opakujem až po vyplnenie kompletnej mriežky. Šifrovanie môže vyzerať napríklad takto:

OT: TAJNEZASIFROVANE (dĺžka 16 znakov)

Mriežka:

T	A	J	N
E	Z	A	S
I	F	R	O
V	A	N	E

rotácia o 0 °

T	A	J	N
E	Z	A	S
I	F	R	O
V	A	N	E

rotácia o 90 °

T	A	J	N
E	Z	A	S
I	F	R	O
V	A	N	E

rotácia o 180 °

T	A	J	N
E	Z	A	S
I	F	R	O
V	A	N	E

rotácia o 270 °

Dostávame ŠT : ZOVN TAIA ANER JSFE

Obr. 1.1: Šifrovanie pomocou Fleissnerovej mriežky a jej rotácie.

Dešifrovanie funguje veľmi podobne s malými zmenami a vymeneným poradím. Najprv vpisujeme ŠT do okienok a následne čítame OT po riadkoch z tabuľky. Ak pri šifrovaní nemá OT toľko znakov ako nejaká mriežka políčok, zvyknú sa doplniť znakmi X. Čo však nie je ideálne z hľadiska bezpečnosti. Preto je rozumné navrhnúť vhodný text, ktorého dĺžka je rovná k^2, kde $k \in \mathbb{N}$. Toto už nabáda k zisťovaniu, aké mriežky existujú a následne vedie k myšlienke, aký veľký je priestor kľúčov K a ako je rozložený. Na to sa pozrieme podrobnejšie hneď v nasledujúcej kapitole.

Kapitola 2

Prístupy automatického dešifrovania

Dôvodov, prečo chceme poznať obsah zašifrovaného textu, existuje veľa. Jednu črtu však majú spoločnú. Informáciu chceme mať presnú a čo najskôr. Preto čas, za ktorý to chceme zistiť, by mal byť čo najkratší. Lúštenie môže prebiehať manuálne (pre väčšinu ľudí zdĺhavé), poloautomaticky(počítač očakáva interakciu a spoluprácu s užívateľom) alebo úplne automaticky. V tejto kapitole sa budeme zaoberať prístupmi automatického dešifrovania a uvedieme prehľad existujúcich prístupov. Popíšeme ich časovú zložitosť vzhľadom na Fleissnerovu otočnú mriežku a samotnú ideu jej implementácie. Ďalej sa pozrieme na ich objektívne výhody či nevýhody a následne sa ich pokúsime odstrániť.

2.1 BruteForce

BruteForce prístup, alebo po slovensky útok hrubou silou, je založený na tom, že postupne vyskúšame všetky existujúce kľúče z priestoru kľúčov K. Na to, či sa tento prístup oplatí, treba poznať veľkosť priestoru K. Kľúčmi sú v tomto prípade použité mriežky. Tie vieme popísať napríklad súradnicami otvorov [riadok,stĺpec] číslovaných od 0.

T	A	J	N
E	Z	A	S
I	F	R	O
V	A	N	E

mriežka: [1,1] [2,3] [3,0] [3,2]

Obr. 2.1: Návrh zápisu mriežky.

Každý otvor pokrýva presne 4 políčka na pozíciach symetrických voči stredu štvorcovej tabuľky. Riadky a stĺpce indexujeme od nuly, tak ako to je aj implementované v priloženej aplikácií. Otvor na pozícií [i,j] sa otáča v mriežke rozmeru n takto:

Obr. 2.2: Rotácie jednotlivých otvorov mriežky.

$$[i,j] \longrightarrow [j, n-i-1] \longrightarrow [n-i-1, n-j-1] \longrightarrow [n-j-1, i]$$

Každá mriežka veľkosti n má presne $\frac{n^2}{4}$ otvorov. Ak si všimneme ľavú hornú štvrtinu mriežky, zistíme, že po štyroch rotáciách sa vyplní celá mriežka. Pre mriežku veľkosti 4×4 sú to políčka [0,0] [0,1] [1,0] [1,1] zoradené najprv podľa riadku a potom podľa stĺpca. V prípade nepárnej mriežky je oblasťou nie štvorec, ale obdĺžnik, ktorý má väčšiu dĺžku ako výšku. Jeho ľavý horný roh sa nachádza v bode [0,0]. Všeobecne je to obdĺžnik daný protiľahlými bodmi A a C.

$$A = [0,0] \quad , \quad C = [\frac{N}{2} - 1 + (N \mod 2), \frac{N}{2} - 1].$$

Takýmto zápisom by sme veľa miesta neušetrili a ani by sme ešte nevedeli presne určiť mriežku. Preto namiesto súradníc políčka uvedieme číslo, po koľkej rotácii sa na danom políčku objaví otvor z mriežky. Obrázok rotácií mriežky je rovnaký ako Obr.1.1 v predchádzajúcej kapitole.

Bez rotovania mriežky je otvor len nad políčkom [1,1] $\Longrightarrow$ [1,1] = 0;

po 1.rotácií mriežky bude otvor len nad políčkom [0,0] $\Longrightarrow$ [0,0] = 1;

po 2.rotácií mriežky budú otvory nad políčkami [0,1] a [1,0] $\Longrightarrow$ [0,1] = [1,0] = 2;

po 3.rotácií nebude žiaden otvor nad oblasťou.

Mriežku teda môžeme popísať postupnosťou otvorov [0,0] [0,1] [1,0] [1,1]. Po dosadení získame zápis *1220*, ktorý je oveľa stručnejší. Má ešte jednu výhodu a tú aj využívame pri implementácií BruteForce algoritmu. Na každej pozícií z [0,0][0,1][1,0][1,1] nastávajú presne 4 možnosti, kde sa otvor nachádza. Z toho vyplýva, že všetkých možných mriežok rozmerov 4×4 je 4^4.

Tu nastáva ešte jeden problém, a to, čo so stredovým políčkom. Pri niektorých obmenách Fleissnerovej mriežky sa políčko obchádza. My však predpokladáme, že sa nevynecháva a dokonca si môžeme určiť, či bude isto zarátané v počiatočnej mriežke, alebo mu určíme, v ktorej rotácií sa zapíše do ŠT. Táto voľba má za následok pri BruteForce prístupe 4-násobné zväčšenie priestoru kľúčov, ako keby bola pevne v počiatočnej rotácií. Všeobecne je počet kľúčov pre n takýto:

$$4^{n^2/4} = 2^{n^2/2}$$

Vidíme, že priestor K rastie s veľkosťou mriežky rýchlejšie ako exponenciálne, keďže má v exponente štvorec veľkosti. To vyzerá z hľadiska bezpečnosti voči BruteForce útoku dobre. Už pre $n = 8$ je to 2^{32} rôznych mriežok a pre $n = 10$ je to 2^{50}. Podľa zdroja [OG07] sú to hranice možného úplného prehľadávania (v závislosti od dostupnosti výkonu). V mojej aplikácií zbehol BruteForce za čas podľa tabuľky testovaný len na 1-3 vstupoch a časy sú udávané len orientačne pre predstavu. Podrobnejšie testovanie a výsledky sú uvedené v 4 kapitole.

veľkosť mriežky	4	5	6	7	8
ohodnotenie slovníkom	15 ms	149 ms	9 s	12.7 min	45 hod
ohodnotenie N-gramami	8 ms	69 ms	4.8 s	6.8 min	25 hod
					predpokladaný čas

Obr. 2.3: Orientačné časy dešifrovania Fleissnerovej mriežky.

V priloženej aplikácií je tento prístup implementovaný nasledovne. Jeho hlavné časti sú generovanie všetkých mriežok a ich následné dešifrovanie a uloženie do dátovej štruktúry uchovávajúcej výsledky. Vzhľadom na obrovský počet takto vygenerovaných OT je ich kompletné prezeranie užívateľom nevhodné a neefektívne, keďže väčšina textov je aj tak nezrozumiteľná. Preto je vhodné zaviesť metriku na výber

vhodných textov, podľa ktorej ponúkne počítač používateľovi len zopár OT, ktoré sú najpravdepodobnejšie a ním očakávané. O ohodnocovacích funkciách bude viac napísané v kapitole 3 ,ktorá je im venovaná. Takto vyzerá implementácia zapísaná v pseudo kóde:

- Inicializácia
- $FOR \quad i = 1 \ldots L; \quad L = |K|$
  - $k_i \longleftarrow Zmen\_bazu(i, 4)$;
  - $OT_{k_i} \longleftarrow d_{k_i}(\text{ŠT})$;
  - $hodnota_{k_i} \longleftarrow ohodnot(OT_{k_i})$;
  - $riesenia \longleftarrow par(OT_{k_i}, hodnota_{k_i})$;
- $vypis(riesenia, 100\ najlepsich)$;

Pri generovaní všetkých kľúčov sa využili výhody zvoleného zápisu mriežky. Stačilo číslo poradia generovanej mriežky previesť do číselnej sústavy so základom štyri. V prípade menších čísiel sa doplnili na začiatok nuly, aby mal zápis mriežky požadovanú dĺžku.

2.2 Hill-climbing

Ako sme si v predchádzajúcej kapitole ukázali, BruteForce metóda dáva síce presné výsledky, ale horšie je na tom s časom, za ktorý tie výsledky dodá. Vzhľadom na to, že čas je vzácny, chceli by sme algoritmus, ktorý to zvládne rýchlejšie. Možno aj za cenu menej presných výsledkov. Dešifrovacia metóda Hill-climbing bola inšpirovaná horolezeckým výstupom na vrchol hory. Predstavme si všetky možné dešifrované texty pre daný ŠT a ich ohodnotenia. Predpokladáme, že maximum prislúcha nášmu hľadanému textu a kľúču. Pre podobné kľúče predpokladáme aj podobné ohodnotenie. Teda, ak by sme si všetky kľúče usporiadali podľa podobnosti, dostali by sme funkciu, v ktorej chceme nájsť globálne maximum. Daná funkcia sa však nemusí dať predstaviť v klasickom priestore, ale až v hyperpriestore obsahujúcom aj lokálne maximá. Koľko tých maxím bude, závisí nielen od danej šifry a jej priestoru kľúčov, ale aj od ohodnocovacej funkcie.

Ak výsledné rozloženie ohodnotení tvorí akoby "*jeden kopec*", tak je jasné, že na vrchol sa dostaneme, ak budeme stále stúpať. Začneme ľubovolným kľúčom a budeme v ňom robiť zmeny vedúce k lepšiemu ohodnoteniu. Vďaka tomu nemusíme prehľadať celý priestor K, ale len jeho časť, čo nám zvyčajne ušetrí mnoho času. Avšak rozloženie nie je takmer nikdy také ideálne. Zväčša to vyzerá nie ako kopec, ale ako "*pohorie*". Rozloženie obsahuje väčší počet lokálnych maxím (*kopcov*). Vtedy, ak začneme od náhodného kľúča a budeme pokračovať zmenami k lepším a lepším ohodnoteniam, dostaneme sa do maxima. Nemusí to byť globálne, ale iba lokálne maximum. Práve kvôli tomu je vhodné viacnásobné opakovanie tohto postupu. Z nájdených lokálnych maxím vyberieme to najväčšie, alebo ponúkneme užívateľovi viacej riešení zoradených od tých najpravdepodobnejších.

Princíp Hill-climbing algoritmu je už jasný, ale ešte sme nedefinovali, čo sú podobné mriežky. Všetky mriežky, ktoré sa líšia iba v jednom otvore, považujeme za podobné. Ako príklad použijeme už známu mriežku *1220* na text TAJNEZASIFROVANE, dostaneme ZOVNTAIAANERJSFE. Podobné mriežky vzniknuté rotáciou jej otvoru prislúchajúceho prvej cifre zapíšeme *2220*, *3220*, *0220*. Z poslednej cifry získame *1221*, *1222*, *1223*. Dešifrované OT sú v tabuľke 2.4. Z kľúča *1223* by sme si už vedeli domyslieť správny OT a manuálne vyskúšať a upraviť mriežku na správnu. V tomto prípade by nám určite pomohlo aj nie úplne presné, ale zato rýchle riešenie. Z mriežky *2220* by sme si mohli vymyslieť asi hocičo a určite nám takýto výsledok nepomôže. Aj napriek tomu, že sa tiež líši len jednou zmenou.

ŠT: ZOVNTAIAANERJSFE dešifrovaný mriežkami:

Mriežka	dešifrovaný ŠT	Δ
1220	TAJNEZASIFROVANE	0
2220	anjseztfaeroiavn	1
3220	jasznotfaeevrina	1
0220	zajtnoasifeveanr	1
1221	tajnearsizfovane	1
1222	tajnersfaizovane	1
1223	tajneszfariovane	1

Obr. 2.4: Niektoré podobné mriežky 4×4.

Pri kľúčoch Fleissnerovej otočnej mriežky spôsobí niekedy aj malá zmena veľký rozdiel vo výslednom texte. Pri nasledujúcom príklade Obr. 2.5 pre rozmery 6×6 je to ešte markantnejšie. Niektoré OT sú nečitateľné, ale na základe niektorých je doklepnutie správnej mriežky chvíľková záležitosť. V závislosti, ako dlho necháme túto metódu pracovať, dostaneme aj výsledky. Aj keď stále pôsobí faktor náhodnosti.

Pre väčšie mriežky, pre ktoré je táto metóda primárne určená, je plne automatická úspešnosť otázkou náhody a času. Pravdepodobnosť úspechu však nie je veľmi veľká. Ak pripustíme poloautomatické dešifrovanie a následná dolaďovanie kryptoanalytikom, úspešnosť výrazne vzrastie .

OT:LASTOVICKYPOCASLETUCHYTAJUMUSKYAHMYZ
ŠT:OOCHTAUMZCKSUMKYHYLSPCALEJAATVIYTYUS

Mriežka	dešifrovaný ŠT	Δ
232311221	LASTOVICKYPOCASLETUCHYTAJUMUSKYAHMYZ	0
232311220	lastovickypocacsltuhyetajumuskyahmyz	1
232311222	lastovickypocalteyscuhtajumuskyahmyz	1
232301221	lastoviocykcpcsaltuhtyaueumsjkyahmyz	1
232321221	lastovipcytocakleyschutajumsukyahmyz	1
232331221	lastoviyctocpckalyshtuauejusmkyahmyz	1
032311221	oaltocvksiscpcualymhttauemkyuysjhzya	1
132311221	caltosvksipocauleymchttajukyuymahzys	1
332311221	atlvooickyscpcsaltuhtyauemmuskjayzhy	1

Obr. 2.5: Niektoré podobné mriežky 6×6.

Je vhodné mať funkciu, ktorá zistí nakoľko sú dva šifrovacie kľúče podobné. Túto funkciu označíme Δ. Vyskytla sa už aj v tabuľkách 2.4 a 2.5. Výsledok bude súčet rozdielov číslic na rovnakých pozíciách. Rozdiel sa bude ale počítať na jednu aj druhú stranu ako počet rotácii. Je jedno, či v smere alebo protismere hodinových ručičiek. To zabezpečí funkcia $g(x)$.

$$g(x) = x \quad , \quad x \in \{0, 1, 2\} \qquad g(3) = 1$$
$$\Delta(mriezka1, mriezka2) = \sum_{i=1}^{|mriezka1|} g(abs(mriezka1 - mriezka2))$$

V praxi sa horolezecký algoritmus skladá z niekoľkých cyklov. Jeho najdôležitejší článok pozostáva z dvoch častí. Prvá vytvára istý počet náhodných mriežok rovný zvolenej konštante $C_{nahodnych}$. V implementácií som zvolil vyššiu konštantu $C_{nahodnych} = 1000$. Vygenerovanými kľúčmi dešifrujeme a ohodnotíme ŠT a následne zoradíme. V druhej časti vyberieme niekoľko najlepšie ohodnotených C_{naj} mriežok. V nich následne spravíme všetky elementárne zmeny, teda vytvoríme všetky mriežky, ktoré majú $\Delta = 1$. Túto druhú časť iterujeme pokiaľ sa nedostaneme do stavu lokálneho maxima a kedy už žiadne zmeny v mriežkach nevedú k zlepšeniu. Obe časti opakujeme niekoľkokrát v závislosti od časového limitu, ktorý máme k dispozícií a úspešnosti, ktorú chceme dosiahnuť. Keďže priestor kľúčov Fleissnerovej otočnej mriežky je veľmi členitý, iterovaná časť sa veľmi rýchlo dostane do lokálneho maxima a prezrie pri svojom behu len relatívne málo mriežok. Preto pre úspešné dešifrovanie potrebujeme, aby náhodne vytvorená mriežka bola dostatočne blízko k tej správnej mriežke. To sa dosiahne zvýšením konštanty $C_{nahodnych}$.

- Inicializácia
- $WHILE \quad casbehu \leq Limit$
  - $FOR \quad i = 1 \ldots C_{nahodnych}$
    - $k_i \longleftarrow randomKey$;
    - $OT_{k_i} \longleftarrow d_{k_i}(\text{ŠT})$;
    - $hodnota_{k_i} \longleftarrow ohodnot(OT_{k_i})$;
    - $riesenia_{rand} \longleftarrow par(OT_{k_i}, hodnota_{k_i})$;
  - $WHILE \quad ohodnoteniesazlepsuje$
    - $FOR \quad i = 1 \ldots C_{naj}$
      - $k \longleftarrow riesenia_{rand}[i]$;
      - generuj $\forall$ mriežky pre k s $\Delta = 1$;
      - $desifruj$ a $ohodnot$ $\forall$ generované mriežky ;
      - vlož $\forall$ hodnoty do $riesenia_{help}$;
    - $riesenia_{rand}.add(riesenia_{help})$;
  - $riesenia.add(riesenia_{rand})$;
- $vypis(riesenia,\ 100\ najlepsich)$;

Ohľadom časovej zložitosti nemá veľký zmysel rozprávať, keďže je algoritmus obmedzený časom. Generovanie náhodných mriežok je v konštantnom čase. Vytvorenie mriežok s $\Delta = 1$ je lineárne závislé od dĺžky kľúča. Jediný problém môže byť pri while-cykle, ktorý iteruje, kým sa ohodnotenia zlepšujú. Dôkaz časovej zložitosti tejto konvergencie je príliš náročný. Avšak tento počet počas testovania neprevýšil nikdy 20 iterácií.

2.3 Iné optimalizačné metódy

Jedným z ďalších optimalizáčných algoritmov je **Simulované žíhanie**. Je veľmi podobné Hill-climbing metóde. Nápad je odpozorovaný z fyzikálnych javov. Metóda sa snaží napodobniť pomalé chladenie roztoku a docieliť stav, kedy bude mať najnižšiu energiu tak, aby sa nezasekla len v lokálnom minime. Na základe hodnotení kľúčov vypočíta teplotu sústavy a jej celkovú energiu. Algoritmus postupne po vygenerovaní nových kľúčov prepočíta celkovú energiu systému. Ak je menšia ako bola pôvodná, tak zmení stav na energeticky menší. V opačnom prípade sa s istou pravdepodobnosťou, závislou od rozdielu energií a globálnej teploty, dostane do stavu s vyššou energiou.

Medzi optimalizačné metódy patria aj **Genetické algoritmy**. Sú inšpirované princípom evolúcie živých organizmov, kde každý jedinec má svoje jedinečné vlastnosti. A jeho schopnosť prežiť a reprodukovať závisí od jeho vlastností. Teda jedinci so zlými alebo nevhodnými vlastnosťami postupne vymierajú a naopak jedinci s dobrými vlastnosťami sa medzi sebou krížia. Tieto pozorovania sú namapované na vhodný problém. V genetickom algoritme sa objavujú javy ako *kríženie*, *mutácia* a *výber*. Kríženie je jav, kde z dvoch rodičov vzniká nový jedinec, majúci časť svojich vlastností od jedného a zvyšok vlastností od druhého rodiča. Mutácia je jav, kedy sa jemná informácia zakódovaných vlastností zmení. Vďaka mutáciám nedochádza ku stagnácií populácie v lokálnych maximách. Nemala by sa prejavovať často, ale len zriedka.

Celý genetický algoritmus vyzerá nasledovne. Vytvorí sa náhodná počiatočná populácia. Následne nastáva opakujúci sa proces výberu jedincov na kríženie. Výber je na základe pravdepodobnosti danou ohodnotením jedincov. Pri vzniku nového jedinca pravdepodobne nastane mutácia. Takto vznikne nová generácia. Tento proces sa opakuje. Má tendenciu konvergovať k populácii, ktorú tvoria najlepší jedinci.

Výhody tejto metódy spočívajú v tom, že nepotrebujeme vedieť ako sa správa ohodnocovacia funkcia na danom priestore prípustných mriežok. Pozitívom je, že sa vyhýba skĺznutiu do lokálnych maxím. Nevýhodou je často oveľa zložitejšia implementácia a správne zvolenie ohodnocovacej funkcie, prípadne vhodná kombinácia viacerých funkcií.

Kapitola 3

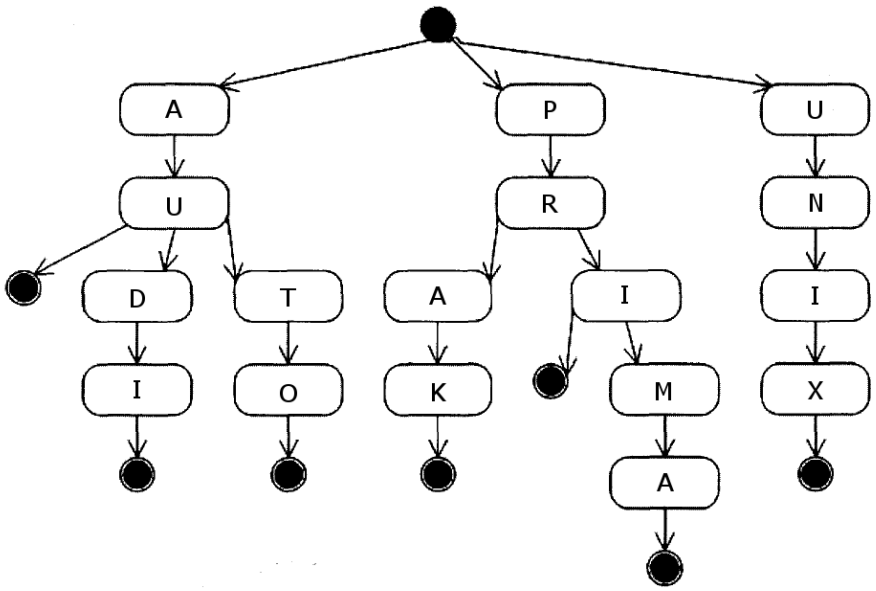
Ohonotenie textu

Z predchádzajúcej kapitoly už vieme, ako generovať OT. Pri BruteForce prístupe, ako aj ostatných prístupoch, sme si vysvetlili, prečo je dobré mať zavedenú porovnávaciu metriku. Pri automatickom dešifrovaní je to jedna z najdôležitejších súčastí celého programu. Veď aj keby sme ŠT dešifrovali všetkými mriežkami v nulovom čase, tak nájsť správny OT medzi ostatnými by nám trvalo veľmi dlho. Okrem toho, ako už bolo spomenuté, sa niektoré algoritmy opierajú počas svojho behu o výsledky tohto ohodnotenia. Na jeho základe sa následne rozhodujú, ktorým smerom sa vydať ďalej. Poďme sa teda pozrieť na jednotlivé idey ohodnocovania, na ich dobré, prípadne zlé vlastnosti. Umožní nám to v jednotlivých situáciach vybrať najvhodnejšiu ohodnocovaciu funkciu, ktorá nám dá čo najpresnejšie a najsprávnejšie výsledky.

3.1 Ohonotenie textu slovníkom

Slovníková metóda ohodnocovania vychádza z veľmi jednoduchého nápadu. Pozrieme sa na OT a následne porovnávame, ktoré slová z daného slovníka sa nachádzajú v texte. Efektívnejšie je to však naopak. Overíme, či sa slová z OT nachádzajú v slovníku. Slovník nám reprezentuje zjednodušený model jazyka. Nesie v sebe len informáciu o slovách v danom jazyku, prípadne jeho modifikácie, ktoré obsahujú aj frekvencie daných slov. Avšak chýbajú v ňom informácie ohľadom jednotlivých vzťahov medzi slovami. Túto informáciu zachytáva podstatne lepšie n-gramový model spomenutý v nasledujúcej podkapitole.

Presnosť slovníkovej metódy závisí hlavne od veľkosti a podrobnosti slovníka. Čím viac slov slovník obsahuje, tým je pravdepodobnosť, že sa v OT vyskytne slovo z jazyka, chýbajúce v našom slovníku, menšia. O to viac je ohodnocovacia funkcia presná. Prehľadávanie vo veľkom slovníku je ale časovo náročná záležitosť. Preto je vhodné naň použiť efektívnu dátovú štruktúru. Jednou z vhodných štruktúr je napríklad **prefixový strom** (nazývaný aj lexikografický). Je to n-árny strom, ktorý má v mojej implementácií pre každý vrchol najviac 26 synov. Ich počet závisí od abecedy, nad akou je strom postavený. Cesta z koreňa do vrcholu V a dĺžky l v sebe nesie informáciu prvých l znakov zo slova, nachádzajúceho sa v danom slovníku. Keďže ide o prefix (predponu) slova, tak podľa toho dostala táto štruktúra aj pomenovanie. Hĺbka prefixového stromu je taká, aké je najdlhšie slovo v slovníku. V bežnom slovenskom texte sú slová priemernej dĺžky 5 znakov. Vyhľadávanie v takomto strome je preto veľmi rýchle. V podstate je pre slovníky väčšiny jazykov v konštantnom čase. Túto zložitosť môžeme zapísať ako $O(n)$, kde n je dĺžka najdlhšieho slova. Ak je treba tento čas v najhoršom prípade ešte skrátiť, je možnosť príliš dlhé slová zo slovníka vyhodiť. Pozrime sa, ako taký prefixový strom vyzerá pre slovník obsahujúci slová { au, audi, auto. prak, pri, prima, unix}



Obr. 3.1: Lexikografický strom.

Keď už máme takýto slovník predpokladaného jazyka OT vytvorený, môžeme pomocou neho naplniť prefixový strom. K tomuto stromu pridáme funkciu, ktorá nám povie či sa dotazované slovo v našej štruktúre nachádza. Za pomoci takejto funkcie už môžeme uvažovať o nejakom ohodnotení OT. Je tu veľa možností a každá má svoje

pre a proti. Základom bude jeden cyklus, ktorý prebehne každý znak v OT a bude z tej pozície hľadať najdlhšie slovo. Teraz nastáva otázka, čo urobiť, keď to najdlhšie slovo nájdeme. Na základe nájdeného slova priradíme OT prislúchajúce body a buď pokračujeme v základnom cykle, teda druhým písmenom z najdlhšieho slova v predchádzajúcom cykle, alebo preskočíme až tesne za to nájdené slovo.

Algoritmus, ktorý preskočí až na koniec nájdeného slova patrí medzi takzvané pažravé **Greedy algoritmy**. Jeho výhodou je, že v základnom cykle na základe slov preskakuje pomerne veľa a preto je rýchlejší ako ten s úplným prehľadávaním textu. Pokiaľ vieme o OT, že sa v ňom nachádzajú dlhé slová, je použitie greedy algoritmu relatívne výhodné. Najväčšie problémy mu robia texty s krátkymi slovami. Napríklad nájdené slovo bude dlhšie ako by malo byť a zasiahne aj do nasledujúceho, a preto po preskočení preskočí aj začiatok nasledujúceho slova. A správne sa nastaví až pri treťom slove.

Očakávané rozdelenie : bod overitelny v rovine $(10^2 + 6^2 = 136)$
Greedy rozdelenie : bodov eritelny v rovine $(5^2 + 6^2 = 61)$

skóre je podľa $n^2, n \geq 4$

Úplné prehľadávanie má zase problém so slovami, ktoré v sebe obsahujú ďalšie slová. Vtedy dostáva jedno slovo body aj za svoje podslová a to značne skreslí výsledné ohodnotenie. Napríklad pri skladaných slovách alebo slovách s viacerými predponami a príponami. Na druhej strane však eliminuje problém, že by mohlo nejaké slovo preskočiť, čo sa pri greedy môže stať.

Očakávané rozdelenie : myslienka $(9^2 = 81)$
úplné prehľadanie: myslienka $(6^2 + 9^2 = 117)$

skóre je podľa $n^2, n \geq 4$

Ešte sme nedoriešili, ako priradiť skóre, keď už máme nájdené slová z jazyka v OT. Ako najrozumnejšie sa ukázalo, ak k celkovému skóre prirátame dĺžku slova alebo nejakú hodnotu vypočítanú na základe tejto dĺžky. Ak je dĺžka slova n, potom prichádzajú do úvahy jednoduché funkcie, ako n, n^2, n^3, $n \log n$.

Lineárna funkcia pre greedy nemá veľký význam. Jasné to bude hneď z príkladu porovnania textov $AAAA$ a očakávaného $AHOJ$. Slovo AA v jazyku nie je, ale slovo A je ako samostatná spojka. Základný cyklus preto zbehne $4\times$ a zakaždým nájde slovo dĺžky 1 a teda výsledné skóre bude 4. Pri texte $AHOJ$ zbehne len raz, ale výsledné skóre bude tak isto 4. Pravdepodobnosť výskytu dlhého slova je v dešifrovanom texte pri transpozičnej šifre výrazne menšia ako krátkeho, preto by sme to mali zobrať do úvahy. Funkcia $n \log n$ sa od lineárnej veľmi nelíši vzhľadom na to, že priemerné n je 5. Za pomoci kvadratickej alebo kubickej ohodnocovacej formulky sa vieme vyhnúť problémom nastávajúcim pri lineárnej funkcii. Avšak objavia sa iné problémy. Ak sa v niektorom z eventuálnych OT vyskytne text, ktorý nie je zmysluplný a ani v ňom nie sú zmysluplné slová až na jedno-dve náhodné, ktoré majú nadpriemernú dĺžku, môže pridelené skóre prevýšiť skóre očakávaného textu. So zvyšujúcou sa mocninou v ohodnocovacej funkcii sa táto anomália objavuje častejšie.

ja by som sa to naucil keby sa to dalo
$(2^2 + 2^2 + 3^2 + 2^2 + 2^2 + 6^2 + 4^2 + 2^2 + 2^2 + 4^2 + = 101)$
po nejakej transpozícií napríklad :
aaabb analytickemu djloooossty $(12^2 = 144)$

skóre je podľa n^2

Čas potrebný na ohodnotenie textu pomocou slovníka a úplného prehľadania vieme ohraničiť nasledovne. Ak n je dĺžka textu, c je priemerná a m maximálna dĺžka slova v jazyku, potom v najhoršom prípade je čas potrebný na ohodnotenie rovný $O(n * m^2)$. V priemernom prípade je to ale len $O(n * c^2)$. Pri greedy algoritme vzhľadom na skoky podľa dĺžky slov je to priemerne c-krát menej, teda len $O(nc)$.

Pre implementáciu ohodnocovacej funkcie vo výslednej aplikácií som zvolil kvadratickú funkciu spojenú s greedy algoritmom. Pažravý algoritmus som si vybral kvôli rýchlosti a hlavne predpokladu, že OT začína od začiatku zmysluplným slovom. Ďalej som zaviedol podmienku, že skóre započítavam len za podslová, ktoré sú aspoň tak dlhé ako nejaká konštanta. V mojom prípade som zvolil konštantu $k = 3$. Vylepšenie ohodnocovacej funkcie pomocou slovníka by som ešte videl v použití dynamického programovania. A to tak, že by sme pre OT našli rôzne pokrytia slovami zo slovníka a ohodnotili daný OT podľa najväčšieho pokrytia. Ak by takých pokrytí bolo viac,

tak vyberieme to s najdlhšími slovami. Nájdenie optimálneho pokrytia je časovo náročnejšie ako obyčajné ohodnotenie. Vzhľadom na porovnanie rýchlosti slovníkového ohodnotenia a n-gramového ohodnotenia popísaného v nasledujúcej sekcii som od implementácie optimálneho pokrytia upustil. Namiesto toho je upravené výsledné skóre a to tak, že je prenásobené percentuálnym pokrytím pri vybraných podslovách z OT pažravým spôsobom. Implementácia by vyzerala nasledovne:

- Inicializácia
- $FOR \quad i = 1 \ldots |OT|$;
  - $s \longleftarrow najdlhsieslovozOTodpozicie(i)$;
  - $IF\ |s| \geq 3$
    - $pokrytie \longleftarrow pokrytie + |s|$;
    - $skore \longleftarrow skore + |s|^2$;
  - $i \longleftarrow i + |s|$;
- $skore \longleftarrow skore * pokrytie/|s|$;

3.2 Ohonotenie textu n-gramami

Iný prístup ako slovník je použitie n-gramového modelu. Ten vychádza z vlastností jazyka, v ktorom bol OT napísaný. Na jednoduché určenie jazyka sa často používa klasická frekvenčná analýza. Tá vychádza z poznatkov o tom, ktorý znak sa s akou pravdepodobnosťou nachádza v bežnom texte daného jazyka. Tieto pravdepodobnosti následne porovnávame s hodnoteným textom. Čím viac sa podobajú, tým skôr je text napísaný v tom jazyku. Klasická frekvenčná analýza je pri dlhších textoch relatívne účinná, avšak pre Fleissnerovu otočnú mriežku nepostačujúca. Problém je práve v tom, že našou šifrou sa šifrujú prevažne krátke texty v priemere od 20 do 150 znakov, čo značne sťažuje situáciu.

N-gramový model túto frekvenčnú analýzu vylepšil. Nepozerá sa na pravdepodobnosť výskytu jednotlivých znakov, ale rovno n-tíc za sebou idúcich znakov. Teda vyššie spomenutá frekvenčná analýza je vlastne 1-gramový model. Ak použijeme pravdepodobnosti výskytu dvojíc, nazývame model **bigramový** a pre trojice **trigramový**.

Tieto modely sú výhodné v tom, že nesú v sebe istú informáciu o danom jazyku. Čím model pracuje s väčšími n-ticami, tým v sebe nesie presnejšie informácie o danom jazyku. Vlastne sa na to môžme pozerať ako na zvláštny špeciálny slovník. Pozrime sa na veľkosť tohto "slovníka":

$$n = \text{dĺžka } n\text{-gramu}, \qquad L = \text{počet znakov v abecede } \Sigma$$
$$\text{počet } n\text{-gramov} = L^n$$

Pre $n \in \{2, 3, 4\}$ je to rozumná veľkosť, ak je potrebná aj rýchlosť ohodnotenia. Ja som sa rozhodol v aplikácií využiť len trigramy. Výhody oproti klasickému slovníku sú jasné už vo veľkosti. Navyše vieme aj pravdepodobnosti výskytu, čo v slovníku nebolo. Taký slovník, čo by obsahoval aj frekvenciu jednotlivých slov by sa nedal tak jednoducho ani vytvoriť. Veď každý štýl písania používa iné slová, a už len určiť, v akom pomere je v bežnom jazyku napríklad odborný, publicistický alebo umelecký štýl by bol problém. V trigramovom modeli to až tak nevadí, lebo to sa až tak výrazne neprejaví na výslednej štatistike, aj keď to v sebe tú informáciu mať bude .

Ako vidíme, úspešnosť či už slovníka, alebo n-gramového modelu závisí od vzorky, na akom bol vytvorený. Aj tu platí, čím presnejšia štatistika, tým lepšie výsledky. Na to bolo potrebné zohnať veľké množstvo textov. Za pomoci kamarátov som zozbieral niekoľko desiatok skrípt z rôznych vedných odborov, kníh a románov. Boli tam teda zastúpené nielen odborné texty, ale aj beletria. Pridal som k tomu aj pár klasických básnických diel od slovenských autorov(povinné čítanie) a publicistických článkov.Vzhľadom na veľkosť básní a článkov to asi príliš nezaváţilo. Výsledkom bolo skoro 5Mb čistého textu. Tie potom prešli úpravou ako je odstránenie medzier a diakritiky, vzhľadom na dohodu, že OT sa pred šifrovaním prevedie na znaky medzinárodnej abecedy bez medzier. Na vopred upravený text bol spustený skript a následne bola štatistika trigramov slovenského jazyka hotová. Najprv som myslel, že textu bude málo, avšak už prvé pokusné skúšky ukázali, že je to postačujúce množstvo.

OVA, STA, OST, PRE, OVE, EHO, ANI, EST,
YCH, TOR, NIE, KTO, PRI, VED, ALE

Obr. 3.2: Najčastejšie trigramy v slovenčine.

Teraz, keď už máme pripravenú štatistiku trigramov cieľového jazyka, môžeme prejsť na porovnávanie s naším eventuálnym OT. Pôvodná idea bola porovnávať všetky pravdepodobnosti trigramov z jazyka s pravdepodobnostou trigramu v OT.

$$x_i = \text{porovnávaný trigram}$$
$$J^n_{x_i} = \text{pravdepodobnosť výskytu } x_i \text{ v } n\text{-gramovom modeli jazyka.}$$
$$T^n_{x_i} = \text{pravdepodobnosť výskytu } x_i \text{ v } n\text{-gramovom modeli OT.}$$

Pri takomto modeli spočítame odchýlku pravdepodobnosti trigramov v texte a v jazyku. Aby sa odchýlky na jednu alebo druhú stranu navzájom nevynulovali, zarátame ich absolútne hodnoty. Celková odchýlka OT od štatistického jazyka je potom daná vzťahom, kde suma prechádza cez všetky trigramy jazyka :

$$\delta = \sum_{i=0}^{L^n} |J^n_{x_i} - T^n_{x_i}|$$

To je pri väčšine textov príliš veľa sčítancov. Jednak vzhľadom na dĺžky ŠT a OT, a na druhej strane má veľká časť trigramov príspevok do jazyka takmer nulový. Napríklad skoro všetky trigramy obsahujúce písmeno Q alebo W majú v Slovenskom jazyku zanedbateľný príspevok. Preto som v mojej aplikácií trochu upravil funkciu δ. Namiesto sumovania cez trigramy jazyka budeme sčítavať cez trojice znakov OT. Tých je najviac $|OT| - n$, avšak niektoré sa môžu opakovať a tým pádom ich bude menej. Preto pred zistením hodnoty δ prejdeme všetky trigramy, zapamätáme si ich množinovo a k nim pridáme prislúchajúce pravdepodobnosti výskytu. Funkcia δ dostane nasledovný tvar:

$$\delta = \sum_{i=0}^{|OT|-n} |J^n_{x_i} - T^n_{x_i}|$$

Napriek takto zjednodušenému modelu funkcie δ mala funkcia na prvých skúšobných testoch dobrú úspešnosť. Bola zároveň o niečo rýchlejšia ako slovník. Podrobnejšie výsledky testovania aj s porovnaním sú uvedené v nasledujúcej kapitole. V druhom cykle implementovanej funkcie δ máme podmienku, či sa daný trigram v jazyku nachádza. V prípade, že nie, pripočítame +1 odchýlku ako penalizáciu. Funkcia δ nám všetky OT zoradí v opačnom poradí ako slovníková a preto by bolo vhodné na záver funkcie otočiť. Potom budeme môcť ukladať OT spolu s ich ohodnotením do rovnakej štruktúry ako pri slovníku bez nutnosti zmeny.

- Inicializácia
- $FOR \quad i = 1 \ldots (|OT| - 2)$;
  - $x_i \longleftarrow OT.substring(i, 3)$;
  - $nastavT_{x_i}$;
- $FOR \quad i = 1 \ldots |T|$;
  - $IF \ J_{x_i} \neq 0$
    - $\delta \longleftarrow \delta + abs(J_{x_i} - T_{x_i})$;
  - inak
    - $\delta \longleftarrow \delta + 1$
- $return \ \dfrac{1}{1+\delta}$;

Časová zložitosť takto implementovaného n-gramového modelu je lineárna. Prvý cyklus zbehne v čase $O(n)$. Druhý tak isto v $O(n)$ za predpokladu, že hodnoty premenných J_{x_i} a T_{x_i} získame v konštantnom čase. To nám zaručí dátová štruktúra využívajúca hešovanie - hešovacia mapa. Prvky štruktúry budú indexované jednotlivými trigramami. Celková časová zložitosť je zase len $O(n)$.

Kapitola 4

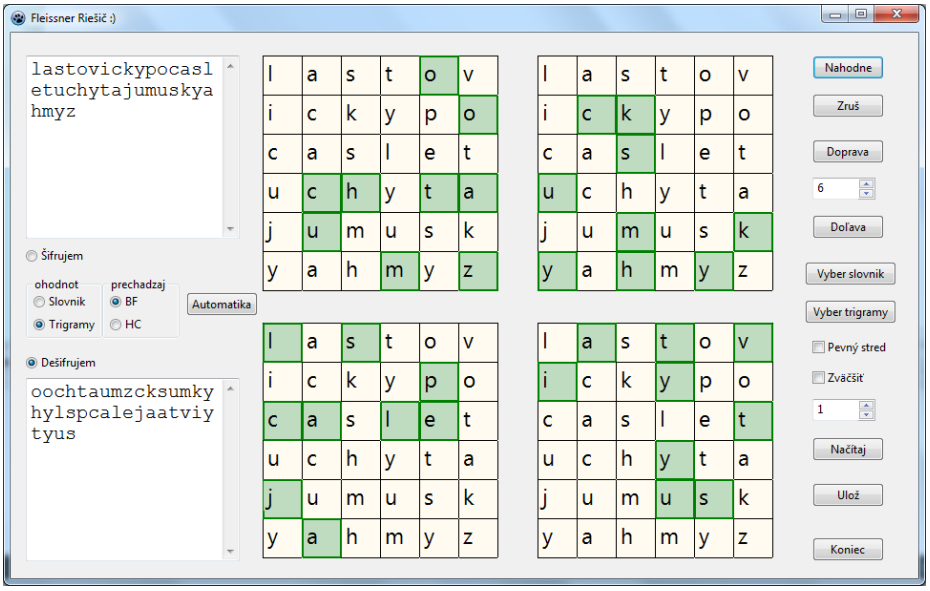
Praktické výsledky

V tejto kapitole sa pozrieme na riešenie problému Fleissnerovej mriežky z praktického hľadiska. Priblížime si, ako funguje výsledná aplikácia, ktorá zahŕňa nadobudnuté poznatky o tomto šifrovacom systéme. Ukážeme si úspešnosť jednotlivých prístupov na väčšom množstve dát a ešte raz zhrnieme ich výhody a nevýhody. V závere poukážeme na vlastnosti vstupov, s ktorými boli problémy a pokúsime sa navrhnúť prípadné možné vylepšenia.

4.1 Využité technológie

Jedným z cieľov tejto práce bolo vytvoriť aplikáciu, ktorá by umožňovala jednoduchú a prehľadnú prácu s týmto transpozičným šifrovacím systémom a zároveň poskytla nástroj na prípadné manuálne dešifrovanie. Preto bolo potrebné zistiť, čo k tomu užívateľ potrebuje a čo naopak je úplne zbytočné. Preto som sa rozhodol zopár mriežok rozmerov 4×4 a 6×6 dešifrovať len tak na papieri. Dal som túto úlohu aj niektorým kamarátom. Väčšina to však vzdala z dôvodu prácneho dešifrovania pri rotácií mriežky a ešte zložitejším úpravám pri zmene otvorov. Tí, čo sa to mu venovali trochu dlhšie, mali pár zhodných postrehov. Chceli by vidieť zmenu dešifrovaného textu hneď po zmene mriežky a zároveň vidieť, ktoré znaky sa vyberajú pri jednotlivej rotácií.

Na základe týchto pripomienok vzniklo nasledovné rozloženie aplikácie Obr. 4.1. Okno obsahuje pole s OT a ŠT a zobrazenie vyplnenej mriežky a jej otvorov pri všetkých štyroch rotáciách. V prípade nevyplnenia kompletnej mriežky upozorní na



Obr. 4.1: Screenshot výslednej aplikácie.

to užívateľa aj farebne. Ďalej umožňuje náhodné generovanie, ukladanie a načítavanie vlastných mriežok. Obsahuje možnosti automatického dešifrovania s voľbou metódy prechádzania priestoru kľúčov a ohodnocovacej funkcie. Obsahuje slovenský slovník a frekvencie trigramov v slovenskom jazyku a má aj možnosť tieto slovníky zmeniť na iné. Napríklad pri riešení cudzojazyčných ŠT alebo pri inej dodatočnej informácií ohľadom OT a jeho štruktúry.

Aplikácia je zložená z dvoch častí, zobrazovacej a výpočtovej. Na zobrazovaciu časť bolo použité open-source vývojové prostredie Lazarus. Prostredie je založené na jazyku FreePascal. Je veľmi podobné prostrediu Delphi, ktoré je oveľa známejšie. Výhodou je bohaté programovacie prostredie, umožňujúce vytváranie grafických aplikácií a použiteľný editor formulárov. Jedným z dôvodov výberu bola aj znalosť tohto prostredia. Výpočtová časť potrebuje využívať rôzne dátové štruktúry kvôli efektívnosti. Žiaľ, dostať sa k týmto štruktúram vo FreePascale, je takmer nemožné a vytvárať si vlastné by asi nebol ten najlepší nápad. Preto padlo rozhodnutie skombinovať to s iným jazykom, ktorý tieto štruktúry ponúka. Tak pribudol medzi použité technológie aj jazyk Java v prostredí Eclips. Automatické dešifrovanie a ohodnocovanie je teda naprogramované v Jave a je použiteľné aj ako samostatný modul. Komunikácia medzi výpočtovou a zobrazovacou časťou je riešená jednoducho pomocou súboru. Vďaka tomu je oveľa jednoduchšie použitie samostatného výpočtového modulu. Hotová aplikácia a aj kompletné zdrojové kódy jej častí sa nachádzajú v CD prílohe.

4.2 Testovanie

Na testovanie boli použité náhodne napísané texty vhodných dĺžok bez medzier a diakritiky uložené v súboroch podľa veľkosti. Tieto súbory, ako aj kompletné výstupy testovania, sú priložené na CD prílohe. Testovalo sa pomocou výpočtového modulu s miernymi úpravami.

Najprv bola testovaná metóda BruteForce na rovnakých vstupoch pri oboch ohodnocovacích funkciách. Vzhľadom na rýchlo rastúce trvanie prehľadávania všetkých kľúčov sa testovali mriežky so stranou $n = \{4, 5, 6, 7\}$. Priebeh testu vyzerá nasledovne. Postupne prejdeme všetky texty z prislúchajúceho vstupného súboru. Pre každý si vyberieme náhodnú mriežku a ňou tento text zašifrujeme. Následne sa tento text nechá dešifrovať BruteForce metódou so slovníkovým ohodnotením a hneď za tým s n-gramovým ohodnotením. Potom nasleduje ohodnotenie a zapísanie výsledkov do súboru. Podstatné informácie sú, na koľkej pozícií vo výslednom rebríčku textov je ten pôvodný OT, čas dešifrovania a najlepšie ohodnotený text. Keďže nechceme prezerať celý zoznam vygenerovaných textov, aplikácia zobrazuje len prvých 100 najlepších. Preto považujeme text za dobre vyriešený, ak sa medzi ponúknutými nachádza. Pokiaľ nie je v prvej stovke, tak už je jedno, na akej je pozícií, nepovažujeme to za automaticky dešifrované. V tom prípade nás zaujíma nie úplne presné, ale blízke riešenie z ponúknutých textov. Tu využijeme funkciu Δ spomenutú v závere podkapitoly 2.2. Do výstupného súboru zapíšeme prvý text s najnižšou odchýlkou Δ od pôvodného.

V nasledujúcej tabuľke 4.2 sú zaznamenané výsledky BruteForce metódy. V stĺpci pozícia je uvedené priemerné miesto, na ktorom sa nahádzal správny OT pri výstupe zo všetkých úspešných. V kolónke Δ je uvedená priemerná najlepšia odchýlka od správnej mriežky pri OT považovaných za automaticky nedešifrované. Všetky výpočty bežali na $2,1$GHz 2-jadrovom procesore okrem BruteForce pre mriežky 7×7. Tie boli riešené na kamarátovom $3,2$GHz 4-jadre, kde to trvalo len $3,3$ minúty namiesto siedmich. Keďže čas potrebný na BrutForce jednej mriežky8×8 je 256-krát dlhší, čo by bolo viac ako 12 hodín, tento test sme vynechali.

n	úspešnosť	pozícia	Δ	čas
4 (trigram)	99/100 (99%)	3	1	3 ms
4 (slovnik)	98/100 (98%)	7	1	2 ms
5 (trigram)	96/100 (96%)	6,9	2	53 ms
5 (slovnik)	88/100 (88%)	6,9	1,5	48 ms
6 (trigram)	68/75 (90,1%)	7,5	3	4,6 s
6 (slovnik)	60/75 (80%)	12	2,1	4 s
7 (trigram)	26/32 (81,3%)	3,2	5,3	3,4 min
7 (slovnik)	21/32 (65,6%)	17,4	1,6	3,3 min

Obr. 4.2: Tabuľka BruteForce testov.

Z dosiahnutých výsledkov je vidieť, že lepšiu úspešnosť dosiahlo trigramové ohodnotenie. Zaostávanie slovníka je spôsobené voľbou rýchlejšieho, ale menej presného greedy algoritmu. Ak porovnáme odchýlku Δ od presného riešenia pri tých OT, tak zistíme, že slovník bol oveľa bližšie. Napríklad pri mriežke 7×7 program vyhodnotil jeden zo vstupných textov (*akrobatnabicyklipredvadzalsaltapriamopredpublikom*) ako neúspešne dešifrovaný, pričom ponúkol aj text *akrobatnabicyklidrpavadzelsalarptiamopredpublikom*, ktorý si užívateľ po pohľade správne domyslí a rýchlo upraví pomocou aplikácie na korektný tvar. Väčšina riešení pri slovníku, aj keď mali veľkú odchýlku Δ, obsahovali správne slová. Na ich základe je manuálne dokončenie pre skúsenejšieho kryptológa relatívne jednoduchou záležitosťou. Na druhej strane pri riešeniach trigramovou metódou bolo oveľa menej textov obsahujúcich časti slov zo správneho textu.

Testovanie horolezeckého algoritmu vzhľadom na požiadavku byť rýchlejší oproti BruteForce prebiehalo na mriežkach väčších ako 6×6. Okrem rozmeru bol dôležitý aj čas behu algoritmu, ktorý sa volil ešte pred spustením. Výsledok prekvapil, lebo úplne automatický výsledok bol iba jeden, a to pri 7×7 trigramovom ohodnotení. Tabuľka vyzerá podobne ako predchádzajúca, len neobsahuje úspešnosť a priemernú úspešnú pozíciu.

n	Δ trigram	Δ slovník	čas
7	5,6	4,9	1 s
7	4,2	2,9	30 s
8	8,5	8,5	1 s
8	8,6	7,6	10 s
8	6,8	5,8	10 min
9	11,2	10,2	30 s
9	9,8	9,6	10 min
10	16,4	15,4	30 min

Obr. 4.3: Tabuľka Hill-climbing testov.

Tu sa problém rozumného automatického ohodnotenia prejavil ešte výraznejšie, keďže by sme potrebovali ohodnotiť funkciou, ktorou ohodnocujeme. Metrika Δ podľa testov rastie s pribúdajúcim n, čo nám príliš nevadí, lebo rastie aj počet otvorov mriežky. Pre $n = 10$ má 25 otvorov a pri $n = 9$ má len 20. Lepšie vlastnosti slovníka sa tu výrazne prejavili. Napriek podobným štatistikám je pre poloautomatické dešifrovanie oveľa výhodnejšie slovníkové ohodnotenie. Vzhľadom na povahu horolezeckého algoritmu zistenú až počas testovania, by bolo vhodnejšie použiť pomalšie (úplné) slovníkové prehľadávanie namiesto pažravého. Pri prezeraní najlepšie hodnotených výsledkov sa objavili také, ktoré mali správny začiatok a koniec správy, alebo také, čo mali správny zase stred. Aj na základe toho si vieme vytvoriť obraz, čo zašifrovaný text ukrýva a dokončiť to manuálne.

Záver

V tejto práci sme sa venovali šifrovaciemu systému Fleissnerova otočná mriežka. Popísali sme ako také šifrovanie prebieha. Snažili sme sa vytvoriť aplikáciu, ktorá toto šifrovanie pohodlne umožňuje a zároveň ho aj čiastočne vizualizuje aj pre ľudí nezaoberajúcich sa touto problematikou hlbšie. To sa nám aj podarilo.

Následne sme sa zaoberali priestorom všetkých kľúčov a automatickému dešifrovaniu. V kapitole 2 sme popísali metódy prístupu automatického dešifrovania a aj ich implementáciu a časovú zložitosť behu. Išlo o metódy BruteForce a Hill-climbing. Okrem toho sme popísali princíp genetických algoritmov.

V 3 kapitole sme popísali rôzne spôsoby ohodnocovania textu dvomi hlavnými metódami: slovníkom a metódou využívajúcou frekvencie krátkych podreťazcov.

Jednotlivé spomenuté metódy a prístupy boli implementované a následne testované. Testovanie prekvapilo. Pri frekvenciách trigramov sme predpokladali, že budú robiť problém príliš krátke texty (16 a 25), ale to sa vôbec nepotvrdilo a pri BruteForce prístupe mali frekvencie veľmi dobré výsledky. Slovníkové výsledky ani nesklamali, ale ani neprekvapili. Testy Hill-climbing prístupu potvrdili, že pojem susednosť mriežok je veľmi ťažko popísateľný a uchopiteľný problém. Zistili sme, že vhodnejšou ohodnocovacou funkciou pre tento prístup je slovníková funkcia úplného prehľadávania.

Do budúcna by bolo vhodné pouvažovať nad použitím a implementáciou genetického algoritmu a upraviť aplikáciu pre lepšiu interakciu s používateľom. Ten by mohol včas obmedziť prehľadávaný priestor a zefektívniť hľadanie. Vhodné pre rýchlosť dešifrovania by bolo aj pouvažovať nad použitím viacvláknového programovania.

Všeobecne vidím prínos tejto práce v poukázaní na to, že riešenie problému nezávisí len od voľby správneho algoritmu, ale aj od miery poznania problému. Konkrétne v poukázaní na menej známy šifrovací problém, ktorý je veľmi zaujímavý svojimi vlastnosťami.

Literatúra

[OG07] P. Zajac O. Grošek, M. Vojvoda. *Klasické šifry.* STU, 2007.

[pas12] pascal.input. N-gram - wikipedia, `http://pascal.input.sk/index.php/27.Prednaska/5tema`, Máj 2012.

[RJT12] Ph.D. RNDr. Jaroslav Teda. Genetické algoritmy a jejich aplikace, `http://programujte.com/clanek/2005072601-geneticke-algoritmy-\a-jejich-aplikace-v-praxi/`, apríl 2012.

[Von06] P. Vondruška. *Kryptologie, šifrování a tajná písma.* ALBATROS, 2006.

[Wik12a] Wiki. Grille - wikipedia, `http://en.wikipedia.org/wiki/Grille_%28cryptography%29`, Apríl 2012.

[Wik12b] Wiki. N-gram - wikipedia, `http://sk.wikipedia.org/wiki/N-gram`, Apríl 2012.