Gymnázium, Praha 6, Arabská 14

Šifrování a prolamování šifer

Ročníkový projekt

Vojtěch Franc, 1E

24. 04. 2021

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V …… dne …… podpis.....

1. Anotace

Cílem mého projektu bylo vytvořit program schopný dešifrovat jednoduché substituční šifry a celé to provést co nejuniverzálněji a pro uživatele co nejjednodušeji. V tomto dokumentu krátce představím základní šifry a ukážu jak je rozpoznat a jakým způsobem je luštit. Dále se budu zabývat frekvenční analýzou a způsoby jak ji zpřesnit přidáním analýzy bigramů a trigramů. Uvedu ukázku kódu, která porovnává bigramy ze zašifrovaného textu s bigramy z otevřeného textu. Část práce je věnována grafickému prostředí a tabulkám, které představovaly problém.

2. Obsah

1. Anotace 3
2. Obsah 4
3. Úvod 5
4. Zadání 5
5. Šifry 5
   5.1. Substituční šifry 6
      5.1.1. Caesarova šifra 6
      5.1.2. Jednoduchá substituční šifra 6
      5.1.3. Frekvenční analýza 6
      5.1.4. Homofonní šifra 7
      5.1.5. Polyalfabetická šifra 7
      5.1.6. Vigenèrova šifra 8
      5.1.7. Vernamova šifra 9
   5.2. Transpoziční šifry 9
6. Části programu 9
   6.1. Třída Abeceda 9
   6.2. Třída AnalyzaZnaku 10
   6.3. Třída Porovnavac 10
7. Analýza textu 10
8. Porovnávání bigramů 11
9. Práce s výsledky 12
   9.1. Tabulky 12
10. Uživatelské rozhraní 13
11. Nevyřešené problémy 14
   11.1. Aktualizace stavového řádku 14
   11.2. Kódování vstupních souborů 14
   11.3. Volání metody v jiné třídě bez její instance 14
12. Závěr 15
   12.1. Doporučení 15
13. Bibliografie 16
14. Seznam obrázků 17
15. Rejstřík 17

3. Úvod

Projekt se zabývá hledáním způsobu automatizované dešifrace jednoduché substituční šifry. Kdyby se povedl tento problém efektivně a univerzálně vyřešit, mohl by se využít k řešení složitějších šifer s transpozicí nebo Vigenèrovy šifry. Všechny šifry, které se v této práci vyskytují, se v různých formách a vylepšeních využívaly od starověkého Egypta do poloviny sedmdesátých let dvacátého století.

Já jsem se rozhodl substituci (záměnu znaků) řešit pomocí hledání vztahů mezi písmeny v textu. Nejprve jsem zkoušel luštit pomocí frekvenční analýzy a zkoumání vztahů mezi sousedícími písmeny, to se mi však moc neosvědčilo, a tak jsem kromě frekvenční analýzy využil hledání a porovnávání bigramů (skupina dvou znaků) a trigramů.

Informace o četnosti písmen a bigramech daného jazyka se program dozvídá ze vzorových textů, které do programu musí uživatel vložit. To mi přijde jako hlavní výhoda mého programu, jelikož je schopen řešit zašifrovaný text v jakémkoliv jazyce, u kterého se dá využít frekvenční analýza. Tato funkce nebývá u podobných programů běžná, uživatel tam dostane na výběr, ve kterém jazyce si myslí, že byl text napsán a program si potřebné údaje načte z databáze.

Tato funkce je užitečná především pokud máme k dispozici nezašifrované texty stejného autora, který zašifrovaný text vytvořil, a tudíž můžeme předpokládat, že se bude podobně vyjadřovat.

4. Zadání

Vytvořím program, který bude umožňovat šifrovat text pomocí několika typů šifer.

Program bude také nabízet možnost text dešifrovat podle statistiky znaků a návaznosti znaků na sebe, které si program zjistí tím, že si zpracuje několik dlouhých textů (knih) v jazyce šifry. Pokud by zbyl čas, bylo by možné rozšířit o další šifry.

5. Šifry

V této práci jsem se zabýval nejběžnějšími typy šifer, které se do nástupu moderního šifrování v sedmdesátých letech dvacátého století využívaly. Při výběru vhodného typu šifrování se kromě bezpečnosti přihlíželo k její obtížnosti a z toho vyplývající chybovosti při šifrování a dešifraci. Tyto úkony prováděli do dvacátého století lidé za pomocí pouze několika jednoduchých pomůcek. Nejbezpečnější by bylo kombinovat několik způsobů šifrování, to se ale kvůli vysoké časové náročnosti a chybovosti nevyužívalo.

5.1. Substituční šifry

Substituční šifra je druh šifry, kdy dochází k záměně znaků nebo skupin znaků. Text zašifrovaný použitím jednoduché substituční šifry je možné rozpoznat tím, že relativní četnost znaků zašifrovaného textu je podobná jako relativní četnost znaků otevřeného textu, jen znaky s podobnou četností se nebudou shodovat.

5.1.1. Caesarova šifra

Nejjednodušší substituční šifra je Caesarova šifra, kdy dochází k posunu pořadí písmen v abecedě. Bude-li šifrovací abeceda posunuta o 1, pak písmeno „A“ se bude šifrovat jako „B“, „Z“ jako „A“ atd. Tuto šifru není příliš obtížné prolomit, jelikož počet možných kombinací zašifrování je roven délce původní abecedy mínus jedna.

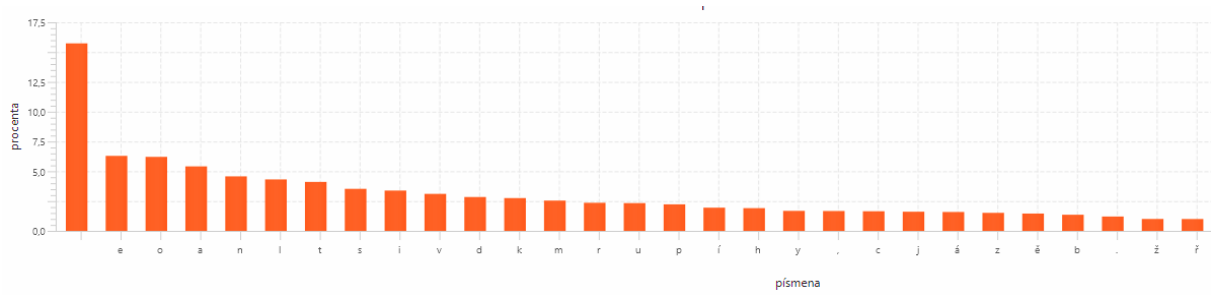
5.1.2. Jednoduchá substituční šifra

Jelikož nebyla Caesarova šifra příliš bezpečná, v minulosti se spíše využívala jednoduchá substituční šifra, kdy mohlo být každé písmeno zaměněno za jakékoliv jiné. Původní abeceda se tak náhodně promíchala a vytvořila šifrovací abecedu. Začátek šifrovací abecedy mohlo tvořit pro zjednodušení a snadné zapamatování klíčové slovo.

Tento způsob šifrování se dlouhou dobu hojně využíval. Počet možných kombinací je faktoriál délky abecedy, a proto se dlouho mělo za to, že jej není možné rozluštit.

5.1.3. Frekvenční analýza

Pověst neprolomitelné šifry jednoduché substituční šifře nezůstala navěky. V 9. století si vzdělaný arabský filosof všiml (pravděpodobně při přepisování knih), že některá písmena se v textu vyskytují častěji než jiná a že by se této skutečnosti dalo využít k dešifraci zpráv.



Obrázek 1 Četnost písmen v češtině (podle mé analýzy)

Tento způsob dešifrace se nazývá frekvenční analýza a díky ní je možné přiřadit k určitým znakům šifrovací abecedy písmena původní abecedy, doplnit jej do textu a zbylé znaky doluštit. Není to však jediný způsob dešifrace textu zašifrovaného jednoduchou substitucí.

K dešifraci lze použít také metodu dosazování předpokládaných slov, kdy se pokoušíme v zašifrovaném textu hledat shluky písmen, které by mohly představovat předpokládané slovo (předpokládané slovo by mělo obsahovat alespoň jedno písmeno více než jednou). Např. bychom si mohli myslet, že část zašifrovaného textu „gorroec“ představuje předpokládané slovo „villiam“.

Výsledek frekvenční analýzy zašifrovaného textu jsem v tomto projektu porovnával s výsledkem frekvenční analýzy předlohy (textů napsaných ve stejném jazyce jako zašifrovaný text). Ke každému znaku ze zašifrovaného textu jsem hledal nejvhodnější znak z předlohy tím, že jsem od sebe odečetl relativní četnosti obou znaků, které jsem umocnil na druhou. Nejvhodnějšímu znaku vyšel nejmenší výsledek.

Frekvenční analýzu je možné použít jen tehdy, je-li text dostatečně dlouhý. Ale i dlouhý text není zárukou, že bude frekvenční analýza dobře fungovat. Nejčastěji vyskytujícím se písmenem v angličtině je „e“. Přesto autor Ernest Vincent Wright dokázal napsat 260 stránkový román, ve kterém se toto písmeno ani jednou nevyskytuje.

5.1.4. **Homofonní šifra**

Na první pohled odolná vůči frekvenční analýze se jeví homofonní šifra, kdy je možné nejčastěji vyskytující se písmena otevřené abecedy zaměnit za několik symbolů (většinou se využívají dvojciferná čísla), aby frekvence jednotlivých symbolů v textu byla stejná.

I tak je možné homofonní šifru prolomit. Začneme vyhledáním poměrně vzácně vyskytujícího písmena, u kterého si můžeme být jisti, že bude v zašifrovaném textu reprezentováno jen jedním symbolem. V angličtině se pro tento účel využívá písmeno „q“, po kterém vždy následuje písmeno u, které bývá zastoupeno třemi symboly. Při luštění tohoto typu šifry se využívá vztahů mezi písmeny (1).

V češtině bychom mohli považovat za nepravděpodobné, že po písmenech „k“ a „r“ bude následovat „i“.

5.1.5. **Polyalfabetická šifra**

I když je homofonní šifra obtížně luštitelná, používá pouze jednu šifrovací abecedu, kterou když rozluštíme, dokážeme dešifrovat celý text. Všechny šifry využívající pouze jednu šifrovací abecedu se nazývají **monoalfabetické šifry**.

Polyalfabetická šifra využívá několik šifrovacích abeced. Například Albertiho šifra střídá dvě šifrovací abecedy. Použitím tohoto druhu šifrování se výrazně zvýší bezpečnost, ale ztíží se proces šifrování a dešifrace.

5.1.6. **Vigenèrova šifra**

Nejznámější polyalfabetickou šifrou se stala Vigenèrova šifra, která používá stejný počet šifrovacích abeced jako je délka dané abecedy. Každá šifrovací abeceda je pak posunuta o jeden znak. Pomůckou k šifrování a dešifraci se stal Vigenèrův čtverec (viz Obrázek 2 Vigenèrův čtverec).

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

Obrázek 2 Vigenèrův čtverec

První sloupec tvoří písmena otevřené abecedy a první řádek slouží k výběru šifrovací abecedy pod sebou. Šifruje se a dešifruje pomocí klíčového slova.

Kdybychom chtěli zašifrovat zprávu „sejdemesezitra“ za pomoci klíčového slova „topol“, první písmeno „s“ bychom šifrovali šifrovací abecedou pod písmenem „t“, tedy jako „l“. Druhé písmeno „e“ bychom šifrovali šifrovací abecedou pod písmenem „o“ jako „s“ atd. Jako výsledek by nám vyšla zpráva „lsyrpfshskbhgo“.

K šifrování a dešifraci Vigenèrovy šifry pomocí klíče doporučuji použít web crypto.interactive-maths.com/vigenegravere-cipher.

Tento způsob šifrování se jeví jako nejbezpečnější ze všech ostatních způsobů, které jsem zmínil. Přesto není vždy nerozluštitelný. Pokud si zvolíme klíčové slovo dlouhé 5 znaků a v textu se často opakují některá slova (např. „ale“ se bude šifrovat jen pěti způsoby), můžeme vypozorovat hlouček opakujících se znaků a z nich se pokusit vyvodit délku klíče. Když známe délku klíče, budeme moci použít frekvenční analýzu na každou šifrovací abecedu a pokusit se tak uhodnout klíčové slovo.

5.1.7. **Vernamova šifra**

Jediná modifikace Vigenèrovy šifry, která je opravdu nerozluštitelná, je Vernamova šifra, kdy je klíčové slovo stejně dlouhé, jako zpráva samotná. To je ale také jedna z nevýhod, kvůli kterým se tato šifra příliš nepoužívala a nepoužívá. Dále je tu stále nutnost předat příjemci dlouhé klíčové slovo, což moderní asymetrické šifrování nevyžaduje.

5.2. Transpoziční šifry

Princip transpozičních šifer spočívá v systematickém proházení znaků v textu. Jako příklad transpoziční šifry lze použít text psaný opačně nebo text, kde se čte každý druhý znak. Text zašifrovaný transpozicí se dá jednoduše rozpoznat tím, že výsledek jeho frekvenční analýzy je velmi podobný frekvenční analýze otevřeného textu. K dešifraci je zapotřebí velká databáze slov a probíhá hledáním přesmyček.

6. Části programu

Celý program se skládá z několika tříd uvedených níže. Hlavní třída, která obsahuje instance většiny tříd je FXMLDocumentController, kde jsou instance veškerých předem vytvořených grafických objektů a metody volané stisknutím tlačítek.

Seznam tříd programu:

- **Abeceda** – práce s otevřenou a šifrovací abecedou
- **AnalyzaZnaku –** podrobná analýza textu
- **Porovnavac –** slouží k porovnávání dvou objektů typu AnalyzaZnaku
- **Znaky -** data z instance této třídy slouží pro TableView
- **FXMLDocumentController –** ovládá grafické prostředí, reaguje na kliknutí, obsauje instance výše uvedených tříd
- **PráceŠifrování –** obsahuje metodu main.

6.1. Třída Abeceda

Třída Abeceda umožňuje získat abecedu ze souborů v určité složce a přidat ke každému zjištěnému znaku abecedy znak, který by jej měl nahradit (Ke znaku z šifrovací abecedy přiřadí znak z otevřené abecedy, nebo naopak). Tato třída obsahuje jednu podtřídu Znak, která má dvě proměnné typu char (původní znak a znak, na který by se měl změnit). V metodách getNahled a ulozit se informace o znacích převádí z ArrayListu obsahujícího instance třídy Znak na HashMap, kde původní znak slouží jako klíč. Vlastní objekt typu HashMap lze vložit jako parametr těchto metod a tyto metody budou pracovat rovnou s ním.

Nejdůležitější metody:

- **nacti** – přečte soubory a zaznamená použitou abecedu
- **serad** – abecedu setřídí podle indexu daného charu
- **getNahled** – vrátí 1 000 slov dlouhý náhled souboru se zaměněnými znaky
- **ulozit** – zamění znaky ze všech souborů v dané složce a u každého zaměněného souboru dostane uživatel vybrat, kam by jej chtěl uložit a pod jakým jménem.

6.2. Třída AnalyzaZnaku

Tato třída má na starosti podrobnou analýzu textů v dané složce. Zjišťuje četnost jednotlivých znaků, nejčastěji nadcházející a předcházející znaky jednotlivých znaků, četnost bigramů a trigramů. Obsahuje dvě stěžejní metody: **analyzuj** a **serad**. Informace o jednotlivých znacích ukládá třída do instancí podtřídy AnalyzaZnak v ArrayListu. Podtřída AnalyzujZnak je potomek třídy Znak, jejíž nejdůležitější proměnné jsou: char znak a long počet. AnalyzujZnak obsahuje navíc dva ArrayListy obsahující instance třídy Znak, kam se ukládají informace o předcházejících znacích a znacích nadcházejících.

6.3. Třída Porovnavac

Třída Porovnavac má za úkol porovnat dvě instance třídy AnalyzaZnaku (analýzu otevřeného textu a analýzu zašifrovaného textu). Mimo porovnávání se ale také stará o vykreslování dešifrační tabulky, kde může uživatel upravovat šifrovací abecedu. Porovnavac obsahuje několik podtříd zjednodušujících porovnávání. Podtřída NejmensiPodleBigramu obsahuje dva ArrayListy, první uchovává id písmen šifrované abecedy a druhý obsahuje instance podtřídy ZnakyPredloha, která uchovává ve dvou ArrayLiistech id znaků z referenčního textu a počet případů, kdy program vyhodnotil, že daný znak z šifrovací abecedy zastupuje znak z otevřené abecedy.

7. Analýza textu

O analýzu textu se v projektu stará třída AnalyzaZnaku, jejíž konstruktor přijímá jako parametr název složky, ve které jsou texty k analýze uložené. O analýzu samotnou se stará metoda analyzuj, která si soubory po písmencích projde a ukládá záznam o počtu jednotlivých písmen, bigramů a trigramů. Tento projekt pracuje pouze se soubory zakódovanými kódováním UTF-8. Pokud metoda čte otevřený text, zbaví ho mezer a netisknutelných znaků, jelikož mezery se do zašifrovaného textu většinou nezařazují a netisknutelné znaky působí jen problémy. Dále program udělá z každého velkého písmena písmeno malé, jelikož velkých písmen není v textu velký počet a nedá se na ně efektivně frekvenční analýza použít.

Třída si uchová v ArrayListu záznam o každém písmenu v použité abecedě spolu s počtem výskytu daného písmena. Metoda AnalyzaZnaku navíc ke každému znaku vytvoří další dva seznamy, které obsahují záznamy o předcházejících znacích a nadcházejících znacích.

8. Porovnávání bigramů

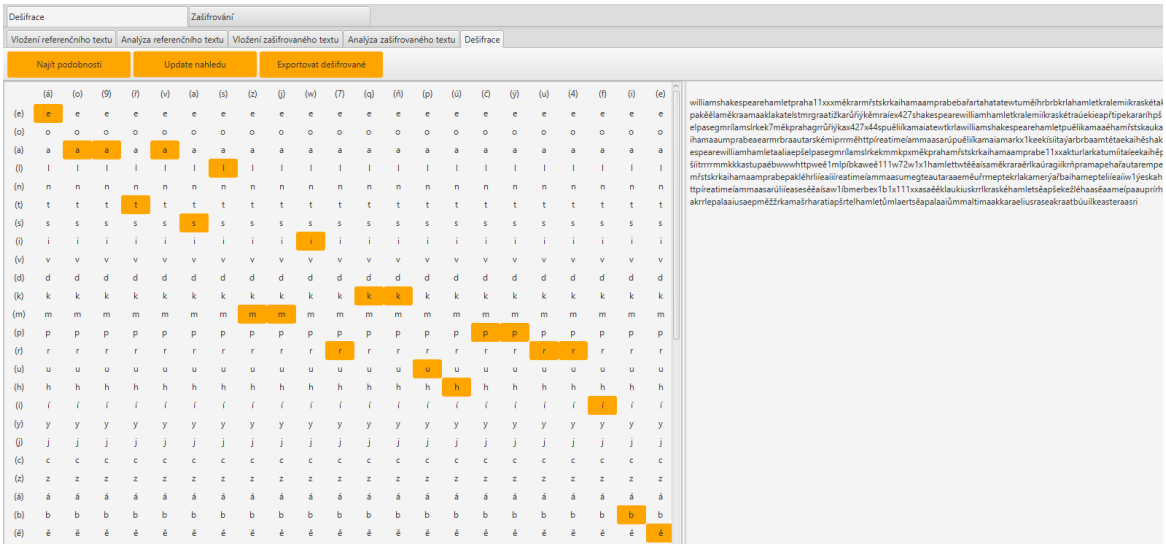
Program zjištěné bigramy ukládá do ArrayListu ve třídě AnalyzaZnaku, ty posléze metoda serad seřadí podle četnosti výskytu. Třída Porovnavac dostane jako parametr analýzu referenčních textů a analýzu textu zašifrovaného. Kód níže se stará o nalezení vhodného bigramu z textu referenčního ke každému bigramu ze zašifrovaného textu. K tomu program stanoví ke každému bigramu relativní četnosti, které porovnává tím, že je od sebe odečte a rozdíl umocní na druhou. Nejvhodnější dvojici vyjde nejmenší číslo.

Dále program hledá indexy jednotlivých dvojic znaků, které se zastupují, aby jim přidal výskyt (viz kód níže). Na konci program ke každému znaku ze šifry přiřadí znak z referenčních textů, který měl výskytů nejvíce. Toto provede program jak u bigramů, tak i u trigramů a jednotlivých znaků. Výsledkem jsou tři mnohdy různé návrhy, jak by mohlo dané písmeno být zašifrované. Nejlepší výsledky při testování vykazovala analýza samotných znaků, ale pokud se bude analýza bigramů a trigramů společně shodovat, budou více relevantní a výsledek se převezme od nich.

```
for (int i = sifra.bigram.size()-1; i >= 0; i--) {
     double relativniCetnostSifra = sifra.bigram.get(i).getPocet() * (100.0 / celkovyPocetBigramSifra);
     int indexNejpodobnejsiPredlohy = -1;
     double nejvetsiPodobnost = Double.MAX_VALUE;
// hlednání nejvíce hodícího se bigramu z referenčního textu
     for (int j = 0; j < predloha.bigram.size(); j++) {
          double relativniCetnostPredloha = predloha.bigram.get(j).getPocet() * (100.0 / celkovyPocetBigramPredloha);
          double nepodobnost = Math.pow(relativniCetnostSifra - relativniCetnostPredloha, 2);
          if (nepodobnost < nejvetsiPodobnost) {
               indexNejpodobnejsiPredlohy = j;
               nejvetsiPodobnost = nepodobnost;
          }
     }

     if (indexNejpodobnejsiPredlohy != -1) {
// rozdělení bigramů na jednotlivé znaky
          char bigramSifra1 = sifra.bigram.get(i).getZnak().charAt(0);
          char bigramSifra2 = sifra.bigram.get(i).getZnak().charAt(1);

          char bigramPredloha1 = predloha.bigram.get(indexNejpodobnejsiPredlohy).getZnak().charAt(0);
          char bigramPredloha2 = predloha.bigram.get(indexNejpodobnejsiPredlohy).getZnak().charAt(1);

          int idSifra1 = -1;
          int idSifra2 = -1;

// nalezení id znaků z bigramu ze šifrovací abecedy
          for (int j = 0; j < sifra.znaky.size(); j++) {
               if (sifra.znaky.get(j).getZnak() == bigramSifra1) {
                    idSifra1 = j;
               }
               if (sifra.znaky.get(j).getZnak() == bigramSifra2) {
                    idSifra2 = j;
               }
          }
// nalezení id znaků z bigramu z předlohy
          for (int j = 0; j < predloha.znaky.size(); j++) {
               if (predloha.znaky.get(j).getZnak() == bigramPredloha1) {
                    if(idSifra1 != -1){
// zaznamenání nálezu
                         hodiciSe.get(idSifra1).add(j);
                    }
               }
               if (predloha.znaky.get(j).getZnak() == bigramPredloha2) {
                    if(idSifra2 != -1){
// zaznamenání nálezu
                         hodiciSe.get(idSifra2).add(j);
                    }
               }
          }
     }
}
```

9. Práce s výsledky

Když program určí, jak by mohly být texty zašifrovány, zobrazí uživateli dešifrovací tabulku (viz Obrázek 3 Tabulka dešifrace). V prvním řádku jsou znaky šifrovací abecedy a pod nimi jsou znaky otevřené abecedy. Znak ve sloupci, na který se znak v prvním řádku dešifruje, je zvýrazněn oranžovým pozadím a kliknutím myši na jiný znak jej lze změnit. Aby uživatel věděl, jak se zašifrovaný text bude podle stávajícího nastavení dešifrovat, má k dispozici v pravé části okna náhled, který se aktualizuje kliknutím na tlačítko „Update náhledu“. O vyplnění této tabulky a aktualizaci po kliknutí na jiný znak, na který se bude dešifrovat, se stará třída Porovanvac.



Obrázek 3 Tabulka dešifrace

9.1. Tabulky

Jako nejlepší způsob prezentace výsledku frekvenční analýzy znaků se jevilo použití TbaleView, což má spoustu výhod, např. umožňuje seřadit svůj obsah podle posloupnosti v jednotlivých sloupcích. Bohužel to má i několik nevýhod. Tou největší byla pro mě v tomto projektu náročnost vkládání dat, kdy data musela být v jedné třídě a muselo být předem nastaveno, jaký sloupec bude zobrazovat jaká data. V ukázce níže přiřazujeme sloupci tabulky proměnnou znak z třídy Znaky (2).

```
tabulkaZnak.setCellValueFactory(new PropertyValueFactory<Znaky, String>("znak"));
// znak je název proměnné ve třídě Znaky, tabulkaZnak je název sloupce tabulky
```

Poté je možné do tabulky vkládat data, v našem případě budeme přidávat jeden řádek.

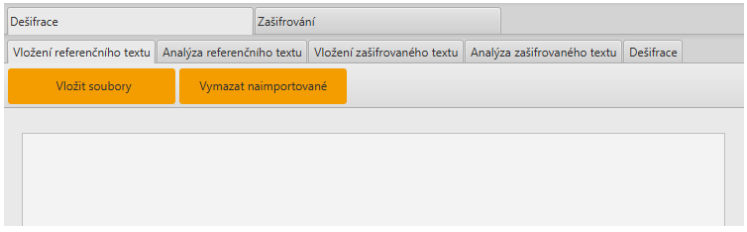
```
tabulka.getItems().add(new Znaky(proměnnáTypuString));
```

To má ale několik nevýhod, např. se mi nedařilo přidávat sloupce dynamicky za chodu aplikace. Jako řešení mi bylo doporučeno nepoužívat TableView a vytvořit si vlastní tabulku pomocí GirdPane (3), což fungovalo a problém vyřešilo. Vykreslování tlačítek do GirdPane a metodu, kterou tlačítko po stisknutí volá, můžete vidět v ukázce kódu níže (4).

```
// vykreslí vnitřek tabulky
public void vykresli() throws IOException {
     for (int i = 0; i < sifra.znaky.size(); i++) {
          for (int j = 0; j < predloha.znaky.size(); j++) {
               Button bt = new Button(String.valueOf(predloha.znaky.get(j).znak));
               bt.setMinWidth(40);
               bt.setStyle("-fx-background-color: none; -fx-border-width: 0;-fx-cursor: hand;");
               bt.setId(i + "," + j);
               bt.setOnAction(this::updateTabulkaPodobnosti);
               if (nejmensiId[i] == j) {
                    bt.setStyle("-fx-background-color: orange;");
               }
               tabulkaDesifrace.add(bt, i + 1, j + 1);
          }
     }
}

// reaguje na kliknutí na tlačítko
void updateTabulkaPodobnosti(ActionEvent event) {
     System.out.println("Stisknuto");
// zjistí id tlačítka, které metodu volalo (v id tlačítka je jeho poloha v tabulce)
     final Node source = (Node) event.getSource();
     String id = source.getId();
     int x = Integer.parseInt(id.split(",")[0]);
     int y = Integer.parseInt(id.split(",")[1]);
// zneaktivnění původního vybraného tlačítka
     tabulkaDesifrace.getChildren().get((sifra.znaky.size() + predloha.znaky.size()) + nejmensiId[x] + ((x) *
     predloha.znaky.size())).setStyle("-fx-background-color: none; -fx-border-width: 2;-fx-cursor: hand;-fx-border-radius: 0;");
// zaktivnění nově vybraného tlačítka
     tabulkaDesifrace.getChildren().get((sifra.znaky.size() + predloha.znaky.size()) + y + ((x) *
     predloha.znaky.size())).setStyle("-fx-background-color: orange;");
     nejmensiId[x] = y;
     System.out.println(x);
}
```

10. Uživatelské rozhraní

K tvorbě grafického prostředí pro tuto aplikaci jsem vybral FXML, jelikož využití externího grafického editoru Scene Builder usnadní spoustu práce. Jednotlivé funkce programu jsou rozčleněny do karet (viz obrázek). Karty jsou uspořádány tak, že by uživatel měl postupovat



Obrázek 4 Uživatelské rozhraní

nejprve zleva doprava (od importu podkladů k exportu). Pod těmito kartami je nástrojová lišta, kde jsou umístěna tlačítka, která vykonávají různé akce (také chronologicky seřazena zleva doprava). Zbylý obsah s výjimkou karty Dešifrace žádné akce nevyvolává.

11. Nevyřešené problémy

11.1. Aktualizace stavového řádku

Nejvhodnější způsob indikace zaneprázdněnosti programu složitějším úkolem mi přišel stavový řádek, který by uživatele informoval o probíhajícím procesu popř. s jeho průběhem. To se mi však nepodařilo, jelikož text ve stavovém řádku se aktualizoval vždy až po dokončení všeho, co obsahovala metoda vyvolaná stisknutím tlačítka. Při hledání příčiny tohoto jevu jsem zjistil, že celá aplikace běží pouze na jednom jádru, které nedokáže obsluhovat analýzu znaků a vykreslování grafiky najednou.

Jako nejjednodušší řešení se jevil kód, který měl jádro pozastavit a dát tím tak prostor pro vykreslování grafiky, žádné z vyzkoušených řešení z internetu mi ale nefungovalo. Jako další možné řešení by bylo aktualizaci textu v objektu Label provádět na jiném jádře, které by běželo nezávisle na tom hlavním. Zjistil jsem, že komunikaci mezi jádry a celé této problematice příliš nerozumím. Tento problém však nijak neomezuje funkce aplikace, není pro tento projekt stěžejní a nebrání jeho používání.

11.2. Kódování vstupních souborů

Další problém se vyskytl při čtení textového souboru, jelikož referenční texty používaly různá kódování. U klasické abecedy bez diakritiky nebyly žádné rozdíly, ale znaky s diakritikou byly pokaždé kódovány jinak. Problém nešel vyřešit jednoduše a uživatel si tak musí všechny soubory, které chce, aby s nimi program pracoval, uložit s kódováním UTF-8, které je nejrozšířenější. Nejjednodušší způsob, jak změnit kódování souboru, je použít předinstalovaný program Notepad na operačním systému Windows, který kódování dokáže většinou rozpoznat.

11.3. Volání metody v jiné třídě bez její instance

Bylo by logické, kdyby se uživateli po ruční úpravě šifrovací abecedy zaktualizoval náhled dešifrovaného textu. To se mi ale nepodařilo, jelikož tato tabulka je převážně spravována třídou Porovnavac, která obsahuje také metodu reagující na změnu vyvolanou kliknutím na možnost změny šifrované abecedy. O náhled se zas stará třída Abeceda, jejíž instance je stejně jako v případě třídy Porovnavac ve třídě FXMLDocumentController.

Problém bych řešil tím, že bych vytvořil metodu ve třídě FXMLDocumentController, která by získala náhled dešifrovaného textu a nechala by ho zobrazit. Metoda ve třídě Porovnavac by se pak k ní musela nějakým způsobem dostat a volat ji. Třída FXMLDocumentController by podle mého měla být dostupná odkudkoliv, ale nenašel jsem fungující způsob, jak se k ní dostat. Problém jsem vyřešil neelegantně, uživatel pro aktualizaci náhledu musí navíc stisknout tlačítko.

12. Závěr

Po přečtení knihy Kniha šifer a kódů (1) jsem měl naplánováno, že k vyřešení jednoduché substituční šifry postačí k použitelnému výsledku jednoduchá frekvenční analýza znaků, kterou by uživatel jen lehce opravil. Plánoval jsem se zaměřit na řešení Vigenèrovy šifry, kde by program zkoušel navyšovat délku klíče a uživatel by kontroloval, jestli tím nevzniká náznak otevřeného textu. To se mi ale nepodařilo, jelikož výsledky frekvenční analýzy nebyly dostatečné ani v případě, kdy se program pokoušel dešifrovat celou knihu. Zjistil jsem, že výsledky frekvenční analýzy jsou nekonzistentní a mohou se drasticky lišit i když budeme porovnávat frekvenční analýzu dvou dlouhých knih. Proto jsem se pokusil výsledky zpřesnit přidáním analýzy bigramů a trigramů, čímž jsem se připravil o možnost využít tuto analýzu pro řešení Vigenèrovy šifry. I tak analýza nedosahuje uspokojivých výsledků a uživatel musí na základě vygenerovaných podkladů a náhledu dešifrace šifru luštit z větší části sám.

Už při výběru tohoto téma jsem věděl, že to nebude jednoduché a bude hrozit, že projekt nebude správně fungovat. Nejvíce se však člověk naučí, vytyčí-li si obtížné cíle, které se pokusí alespoň z části dosáhnout. To se mi se mi povedlo, dozvěděl jsem se spoustu o fungování šifer a naučil jsem se používat spoustu grafických prvků v Javě.

12.1. Doporučení

Mně se způsob dešifrace substituční šifry pomocí automatizované frekvenční analýzy neosvědčil. Myslím si, že by ale mohl fungovat, kdyby program pracoval i se slovy, kterými by výsledky frekvenční analýzy opravoval. Další možný způsob řešení substitučních šifer, který jsem nezkoušel do projektu implementovat, spočívá v dosazování předpokládaných slov, což by mohlo dosahovat větší úspěšnosti. Program, který tímto způsobem luští jednoduchou substituci, naleznete na webových stránkách http://www.musilek.eu/michal/sifry-lusteni-slovo.html. Pokud by se někdo chtěl ve své budoucí práci věnovat této tématice, doporučil bych mu řešení transpozičních šifer pomocí hledání přesmyček.

13. Bibliografie

1. **Singh, Simon.** *Kniha kódů a šifer.* [překl.] Petr Koubský a Dita Eckhardtová. Praha : Argo; Dokořán, 2003. str. 382. ISBN80-86569-18-7; 80-7203-499-5.

2. **Chrispie, Uluk Biy a multiplayer1080.** How to populate a TableView that is defined in an fxml file that is designed in JavaFx Scene Builder. *stack overflow.* [Online] 7 2012. [Citace: 22. 4 2021.] https://stackoverflow.com/questions/11180884/how-to-populate-a-tableview-that-is-defined-in-an-fxml-file-that-is-designed-in.

3. **Elias Elias, a další.** javafx GridPane retrieve specific Cell content. *stack overflow.* [Online] 1 2016. [Citace: 20. 4 2021.] https://stackoverflow.com/questions/20655024/javafx-gridpane-retrieve-specific-cell-content.

4. **knut5, Babak Hashemi a Joe1962.** Getting id from a ActionEvent in JavaFX. *stack overflow.* [Online] 9 2017. [Citace: 20. 4 2021.] https://stackoverflow.com/questions/45976471/getting-id-from-a-actionevent-in-javafx.

5. **Clark, Daniel Rodriguez.** Vigenère Cipher. *Crypto Corner.* [Online] [Citace: 25. květen 2021.] https://crypto.interactive-maths.com/vigenegravere-cipher.html.

6. **Irina Fedortsova.** Drag-and-Drop Feature in JavaFX Applications. *docs.oracle.* [Online] [Citace: 20. 4 2021.] https://docs.oracle.com/javafx/2/drag_drop/jfxpub-drag_drop.htm.

7. **Gupta, Lokesh.** How to iterate through ArrayList of objects in Java. *howtodoinjava.* [Online] [Citace: 20. 4 2021.] https://howtodoinjava.com/java/collections/arraylist/iterate-through-objects/.

8. **Novák, Michal.** Frekvenční analýza. *Wikisofta.* [Online] 30. 4 2021. https://wikisofia.cz/w/index.php?title=Frekven%C4%8Dn%C3%AD_anal%C3%BDza&oldid=54000.

9. **Electric Coffee, James_D a Mansueli.** stack overflow. *How do I open the JavaFX FileChooser from a controller class?* [Online] 9 2015. [Citace: 17. 4 2021.] https://stackoverflow.com/questions/25491732/how-do-i-open-the-javafx-filechooser-from-a-controller-class.

10. **vijayk, a další.** How to copy file from one location to another location? *stack overflow.* [Online] 6 2013. [Citace: 18. 4 2021.] https://stackoverflow.com/questions/16433915/how-to-copy-file-from-one-location-to-another-location.

11. **baeldung.** How to Find an Element in a List with Java. *baeldung.* [Online] 20. 4 2021. https://www.baeldung.com/find-list-element-java.

12. **Blog, 1bestCsharp.** Java - How To Drag And Drop Image In Java Netbeans. *1BestCsharp blog.* [Online] [Citace: 22. 4 2021.] https://1bestcsharp.blogspot.com/2015/02/java-how-to-drag-and-drop-image-in-java.html.

13. **Gupta, Lokesh.** ArrayList sort() – Sort list of objects by field. *HowToDoInJava.* [Online] [Citace: 25. 4 2021.] https://howtodoinjava.com/java/collections/arraylist/arraylist-sort-objects-by-field/.

14. **Kasyap, Krishna.** How to open multiple files using a File Chooser in JavaFX? *tutorialspoint.* [Online] 18. 5 2020. https://www.tutorialspoint.com/how-to-open-multiple-files-using-a-file-chooser-in-javafx.

15. **neznámý.** How to implement ActionEvent using method reference in JavaFX? *tutorialspoint.* [Online] 19. 1 2020. https://www.tutorialspoint.com/how-to-implement-actionevent-using-method-reference-in-javafx.

16. —. Java Delete Files. *w3schools.* [Online] [Citace: 10. 4 2021.] https://www.w3schools.com/java/java_files_delete.asp.

14. Seznam obrázků

Obrázek 1 Četnost písmen v češtině (podle mé analýzy) 6
Obrázek 2 Vigenèrův čtverec 8
Obrázek 3 Tabulka dešifrace 12
Obrázek 4 Uživatelské rozhraní 13

15. Rejstřík

A

Albertiho šifra, 7

asymetrická šifra, 8

B

bigram, 5

C

Caesarova šifra, 6

Č

četnost výskytu, 10

E

Ernest Vincent Wright, 7

F

Frekvenční analýza, 6

H

Homofonní šifra, 7

K

klíčové slovo, 6, 8

M

monoalfabetická šifra, 7

N

netisknutelné znaky, 10

P

Polyalfabetická šifra, 7

Porovnávání bigramů, 10

R

relativní četnost, 11

S

stavový řádek, 13

substituční šifra, 6

Š

šifrovací abeceda, 6, 7

T

TbaleView, 12

Transpoziční šifry, 9

trigram, 10

U

UTF-8, 10, 13

V

Vernamova šifra, 8

Vigenèrova šifra, 7