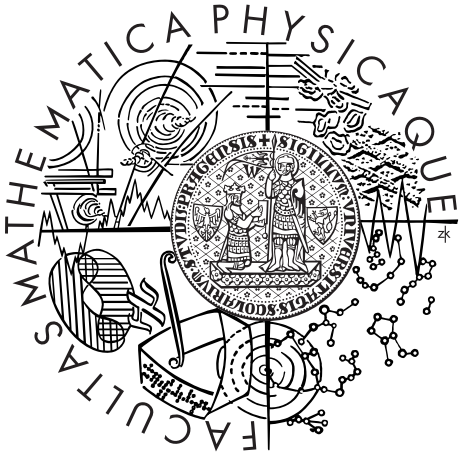
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Filip Hlásek

Automatický dešifrátor pro šifrovací hry

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: RNDr. David Mareček, Ph.D.
Studijní program: Informatika
Studijní obor: Obecná infromatika

Praha 2014

Chtěl bych poděkovat organizátorům šifrovacích her a jejich účastníkům. Díky úsilí věnovanému přípravě, resp. řešení šifer vznikl postupem času vzorek několika tisíc šifer, který se stal základem pro tuto práci. Ze všech šifrovacích her bych chtěl speciálně vyzdvihnout hry Bedna a Po trati, jejíchž organizátoři zveřejňují na svých stránkách řešení všech šifer v přehledném a strojově snadno rozpoznatelném formátu. Navíc doplňují historické ročníky svých her také o mapy, které umožnily lazení a trénování algoritmu za použití geografických souřadnic.

Jednomu členu organizačního týmu zmíněné šifrovací hry Po trati patří můj největší dík. Je jím vedoucí této práce RNDr. David Mareček, Ph.D., který přišel s nápadem na téma, neváhal se se mnou kdykoliv při tvorbě práce sejít a konzultoval se mnou vytrvale různé verze dešifrátoru. Děkuji mu tímto za trpělivost a profesionalitu, se kterou se své role zhostil. Během společných diskuzí mi předal spoustu svých zkušeností nejen z počítačové lingvistiky, ale také o zásadách psaní odborných textů.

Na závěr bych chtěl poděkovat těm, kteří mi nabídli svou pomoc s manuální evaluací výsledků. Byli jimi slečna Michaela Hubatová a již zmíněný RNDr. David Mareček, Ph.D.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Automatický dešifrátor pro šifrovací hry

Autor: Filip Hlásek

Katedra: Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: RNDr. David Mareček, Ph.D., Ústav formální a aplikované lingvistiky

Abstrakt: Práce se zaměřuje na substituční šifry používané při terénních šifrovacích hrách. Nejprve získáme vzorky jazyka používaného v souvisejících textech a prozkoumáme jeho zvláštnosti. Dále navrhneme jazykový model speciálně určený pro práci s omezeným množstvím dat. Poté prozkoumáme různé možnosti vyhledávání pravděpodobných řešení a předvedeme, jak je možné přímočarý algoritmus vylepšit na poměrně efektivní. Podstatnou součást celého projektu tvoří softwarové dílo, jímž je konzolová aplikace sloužící k luštění šifer. Ta je schopna kompletně vyřešit více než 15 % šifer, na kterých byla testována. Dalšího zlepšení je možné dosáhnout zadáním zeměpisných souřadnice místa, na kterém se uživatel nachází. Program pak bude hledat polohu další šifry pouze na nepříliš vzdálených místech. To umožní dešifrátoru prozkoumat více možností a docílit větší přesnosti.

Klíčová slova: dešifrátor, substituční šifra, jazykový model

Title: Deciphering tool for puzzlehunt games

Author: Filip Hlásek

Department: Institute of Formal and Applied Linguistics

Supervisor: RNDr. David Mareček, Ph.D., Institute of Formal and Applied Linguistics

Abstract: The thesis focuses on substitution ciphers used at puzzlehunt games. First, we collect samples of the language which is used in the texts, that we are interested in. Furthermore, we propose a language model specially designed for working with sparse data. After that, we will explore the ways of searching for probable solutions and we will present a straightforward algorithm. Then we will improve it and make it more efficient. A significant part of the project is a console application, which serves as a deciphering tool. It is able to solve more than 15 % out of the ciphers, on which it has been tested. The result can be improved if the user inputs his current geography location. In this case, the program will search just for nearby places. That will allow the deciphering tool to explore more options and achieve a better precision.

Keywords: deciphering, substitution cipher, language model

Obsah

Kapitola 1

Úvod

V posledních letech se na českém území začal rozmáhat fenomén *šifrovacích her*. Těchto soutěží se každoročně účastní několik tisíc hráčů a vyskytují se v nich obvykle velice originální šifry, při jejichž luštění je potřeba důvtipu, logického uvažování a někdy také encyklopedických znalostí. Není žádnou výjimkou, že i zkušený tým stráví řešením jediného takového problému třeba i několik hodin.

To nás přivedlo k myšlence, zda by nebylo možné některé šifry, které se na zmíněných hrách používají, luštit algoritmicky, nebo alespoň vytvořit nástroj, který by při luštění pomáhal.

Cílem práce rozhodně není nijak nabourat koncept šifrovacích her, neboť i samotný autor je jejich náruživým účastníkem, ale spíše prozkoumat současnou situaci a analyzovat, do jaké míry jsou použité šifry neprolomitelné. Pokud se ukáže, že k jejich řešení vůbec není potřeba komplikovaný postup a bohatě stačí nějaký počítačový program, organizátoři budou patrně při chystání dalších šifer opatrnější.

1.1 Základní pojmy

Přestože následující pojmy mají v běžném jazyce mnohem širší význam, pro účely tohoto textu je budeme používat v popsaném užším smyslu:

Šifrovací hra[1] (šifrovačka) je týmový závod, jehož součástí je luštění šifer. Organizátory předem vytyčená trať se skládá z několika stanovišť (obvykle mezi deseti a dvaceti). Na každém takovém stanovišti čeká jedna nebo více šifer, které musí tým vyřešit, aby se poté dozvěděl polohu dalšího stanoviště. Řešení obvykle obsahuje buď název místa, nebo popis cesty k následující šifře. Na českém území se každoročně koná přibližně deset šifrovacích her, kterých se pravidelně účastní několik tisíc hráčů. V celém textu se budeme zabývat výhradně českými šifrovacími hrami.

Abeceda je množina šestadvaceti písmen anglické abecedy $\{A, B, C, \ldots, Z\}$. Přestože jsou všechny texty česky, upouští se od písmen s diakritikou. Text i bez nich snadno pochopitelný a na anglické abecedě je jasné přirozené

[1] Přesněji **terénní šifrovací hra**. Její alternativou je *internetová šifrovací hra*, která je založena na podobném principu, ale místo polohy dalšího stanoviště je řešením šifry nějaké heslo. Cílem internetové hry je obvykle odhalit co nejvíce hesel.

uspořádání písmen, což je u některých šifer klíčová vlastnost. Někdy šifry obsahují také čísla, ale i v tomto ohledu problém zjednodušíme.

Symboly budeme značit malými písmeny anglické abecedy $\{a, b, c, \ldots, z\}$, ale na použitých znacích vůbec nezáleží. Důležité je pouze to, které z nich jsou stejné, a které naopak různé. Ve skutečných šifrovacích hrách se potom často jedná o nějaké obrázky, či grafémy.

Monoalfabetická substituční šifra je jeden z typů šifer používaných při šifrovacích hrách. Jednotlivá písmena abecedy jsou při použití tohoto kódování reprezentována (substituována) jinými symboly. Stejná písmena jsou vždy popsána stejnými symboly, zatímco různá písmena různými symboly. Samotný text je poté zašifrován přímočaře nahrazením písmen odpovídajícími symboly. K rozluštění takové šifry je potřeba nalézt systém, jakým je převod realizován. Často je poměrně náročné učinit klíčové pozorování, přijít s nějakou trikovou myšlenkou, nebo v použitých symbolech něco *„uvidět“*. Jelikož se práce zabývá výhradně tímto typem šifer, někdy budeme používat pouze pojem **šifra**.

Zadání (substituční) šifry je konečná posloupnost symbolů.

Převodní tabulka je bijektivní zobrazení *symbolů* na *abecedu*. Často se může jednat o použití známého kódování jako například Morseovy abecedy nebo Brailleova písma. Jindy jsou zase písmena abecedy zašifrovány pomocí ASCII hodnot, ale téměř vždy se jedná pouze o jeden z kroků, který je potřeba při luštění udělat.

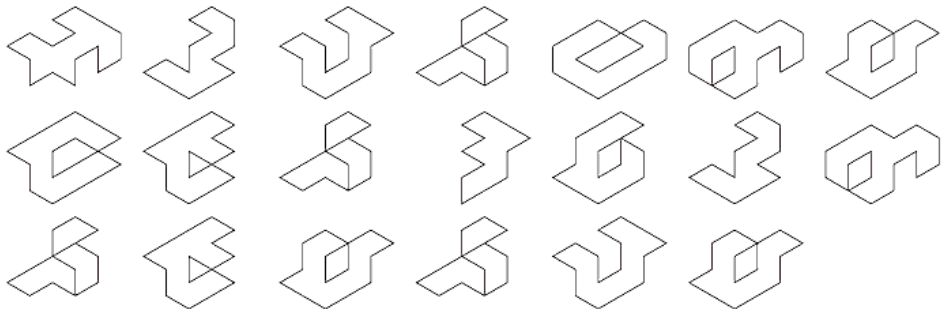
Řešení šifry (výsledek šifry) je posloupnost písmen abecedy, která vznikne tak, že na zadání šifry aplikujeme převodní tabulku. Jednotlivé symboly zadání rozšifrujeme pomocí převodní tabulky a získáme původní text, který nám organizátoři hry chtěli předat.

Správné řešení šifry je řešení očekávané organizátory šifrovací hry. Aby bylo řešení přímo čitelné, je potřeba ještě text rozdělit na slova. Tento druhý krok je ale většinou již snadné udělat, a dokonce i organizátoři očekávají, že výsledek bude pochopitelný i bez rozdělení na slova.

Dešifrátor (automatický dešifrátor) je počítačový program, jehož vstupem je zadání šifry a výstupem jsou možná řešení šifry.

1.2 Ukázka šifry

Pojmy, o kterých jsme hovořili v předchozí sekci, osvětlíme na následující šifře. Ta byla použita na *šifrovací hře* Bedna 2012.[2]



Obrázek 1.1: Zadání 13. šifry z šifrovací hry Bedna 2012

Zadání obsahuje několik obrazců, z nichž některé jsou stejné. Navíc jsou jednoznačně uspořádané a to nás vede k myšlence, že by se mohlo jednat o *monoalfabetickou substituční šifru*. Jednotlivé *symboly*[3] jsme již identifikovali a *zadání šifry* tak můžeme přepsat například následovně:

```
ywosbfuvhstiwfshusou
```

Jedno z možných *řešení* uvedené šfiry může být:

```
YWOSBFUVHSTIWFSHUSOU
```

Zřejmě se nejedná o *správné řešení* zkoumané šifry. Cesta k rozluštění je poměrně náročná a nakonec vede k jednoznačné identifikaci jednotlivých symbolů a k *převodní tabulce*:

```
b -> S, f -> T, h -> R, i -> B, o -> M, s -> E
t -> D, u -> I, v -> P, w -> A, y -> N
```

Této tabulce odpovídá následující *řešení*:

```
NAMESTIPREDBATERIEMI
```

Snadno již rozdělíme výsledný text na slova a zjistíme, že další stanoviště se nachází na *náměstí Před bateriemi*. Doplňme ještě, že uvedená převodní tabulka úplně neodpovídá popsané definici, protože nepopisuje, na co se zobrazí ostatní symboly. Ať to ale bude jakkoliv, řešení to neovlivní. Podobně budeme také v dalším textu připouštět tento volnější význam pojmu *převodní tabulka*.

[2] `http://www.bedna.org/2012/sifry/13`

[3] Stejným obrazcům přiřadíme stejný *symbol*, ale nezáleží na tom, které použijeme. Ekvivalentně můžeme zadání přepsat například jako `abcdefghidjkbfdigdcg`.

1.3 Jazyk použitý ve správných řešeních šifer

Jazykem šifrovacích her je čeština, ašak značně specifická. Řešení šifer často netvoří kompletní věty, ale spíše heslovitě a zkratkovitě popisují cestu k dalšímu stanovišti. Samotný text je obvykle tak stručný, jak jen je možné se zachováním srozumitelnosti a někdy je velmi náročné interpretovat výsledek, dokonce i u správně rozluštěné šifry.

Jako ukázku typických řešení šifer uvádíme následující seznam. Jednotlivá řešení jsme pro snadnější čtení rozdělili na slova.

```
FUGNEROVO NAM JZ ROH
NADRAZI BRANIK
TRI SEST PET
TRI CTVRTE KILAKU S ODSUD
KRUHOVA LESNI SKOLKA CHATKA
ZENA VRTAKA
DOLNI RYBNIK VYCHOD ROH
KOSTEL ULICE ELISKY PREMYSLOVNY
POMNIK PALACKEHO VRCH
BOHACOVA HRISTE SEVER
INFO CEDULKA NA SOUTOKU
VRCHOL KOPCE BRABENAK
SRBSKO MOST POD ROZVODNOU
UL PETRSKOU AZ K TOALETAM
JEDLOVKA NA ROZC ZELENE A CYKLO
SKOLKA D MECHOLUPY
DALSI SIFRU VEM U STRAH KLASTERA P MARIE
```

Mohlo by se zdát, že každý organizátor používá své vlastní zkratky a tím pádem úplně jiný jazyk. Nemělo by potom smysl nijak analyzovat a snažit se popsat společný jazyk šifrovacích her. Na druhou stranu se organizátoři drží nějakých zásad a nepsaných pravidel[4]. Díky vzájemné komunikaci mezi různými šifrovačkami se zkoumaný jazyk stále více ustaluje, a proto má smysl se jím zabývat.

Jeden z důvodů, proč se organizátoři uchylují ke zkratkám a různým opisům, je, že chtějí tímto umělým zásahem do čistého jazyka znemožnit správný výsledek odhadnout. Pro ukázku uvažme zadání šifry, kde se první symbol shoduje s devátým a také sedmý s desátým. Zkušený luštitel uvedené šifry si snadno povšimne, že řešení může začínat poměrně běžným slovem `KŘIŽOVATKA`[5] a pomůže mu to k rozluštění zbytku šifry. Naopak zkratku `KŘIŽ` téměř nelze na začátku šifry odhadnout, neboť na čtyři různá písmena začíná mnoho dalších slov. Mezi nejčastější zkratky patří zkráceniny geografických názvů jako například `NÁM` (náměstí), `NÁDR` (nádraží), `POM` (pomník), `ROZC` (rozcestí / rozcestník), `ZAST` (zastávka), ...

[4] Jedním takovým psaným kodexem by mohla být kniha Almanach TMOU (Pelánek, 2008).

[5] Pro snazší čitelnost uvádíme v této sekci slova včetně diakritiky, přestože se v této podobě v řešeních šifer vyskytují.

Dále se ve správných řešeních šifer běžně vyskytují názvy ulic, ale i jména menších obcí, případně kostelů, pomníků, studánek, památných stromů, mostů, náměstí, soch, zastávek MHD, kopců, či jiných geograficky významných míst.

Typické jsou v rozluštěném textu také názvy světových stran a jejich zkratky jako `S`, `JZ`, `SEVEROVÝCHODNÍM SMĚREM`, atp. Trochu méně častou skupinou slov jsou barvy reprezentující turistické značky jako `KŘIZOVATKA ČERVENÉ A ŽLUTÉ`.

Posledním jevem, který v této kapitole zmíníme, je častý výskyt objektů, kde může být šifra fyzicky ukryta. Mezi taková slova patří například `HŘIŠTĚ`, `LAVIČKA`, `SOCHA`, `HRÁZ`, `KOSTEL`, `TOPOL`, `POMNÍK`, `LOUKA`, `SLOUP` nebo třeba `KAŠNA`.

1.4 Související práce

Dílem, které se zabývá podobnou problematikou, je bakalářská práce Klasické kryptografické metody (Žeravík, 2012). Autor zde zkoumá různé druhy šifer a mimo jiné i monoalfabetickou substituční šifru. Navrhuje také dešifrátor používající genetický algoritmus, ale nijak se nezaměřuje na šifrovací hry ani na žádný konkrétní jazyk. Tato práce by se měla zásadně odlišovat především tím, že se zaměřuje na extrémně krátké vzorky předem známého jazyka používaného v řešeních šifer z šifrovacích her.

Dalším souvisejícím projektem je mobilní aplikace Šifrovací pomůcky Absolutno[6]. Tento nástroj pomáhá řešiteli při luštění různých druhů šifer, ale nenabízí možnost automatického dešifrování bez asistence člověka.

Velmi teoretickou prací týkající se našeho tématu je článek *Decipherment Complexity in 1:1 Substitution Ciphers* (Nuhn a Ney, 2013). Autoři v něm diskutují algoritmickou složitost přesně našeho problému. Pro unigramový model (viz Kapitola 3) přicházejí s polynomiálním řešením, zatímco pro bigramový ukazují převod na problém obchodního cestujícího (TSP). Z toho plyne, že v plné obecnosti se jedná o NP-úplný problém.

1.5 Hlavní výzvy

Na první pohled se může zdát, že se jedná o poměrně jednoduchý problém, ale již při prvním pokusu o návrh řešení narazíme na několik překážek. Další se vyskytnou až v průběhu vývoje a některé se naplno projeví až při samotném testování.

Možných řešení je příliš mnoho.
Každá z 26! permutací písmen abecedy odpovídá jedné převodní tabulce. Většinou se v šifře nevyskytuje všech 26 různých symbolů a průměrná šifra[7] jich obsahuje pouze 11,7. Přesto když uvážíme podprůměrnou šifru obsahující 10 různých symbolů, přichází v úvahu celkem $26 \cdot 25 \cdots 17 = 19\,275\,223\,968\,000$ různých řešení. Kdybychom například dokázali analyzovat 10^9 řešení za vteřinu[8], celý výpočet by trval přes pět hodin. Navíc

[6] `https://play.google.com/store/apps/details?id=cz.absolutno.sifry`

[7] Uvedená statistika je vytvořená na základě trénovacích dat (viz Sekce 2.1.)

[8] Jedná se o velmi hrubý a spíše optimistický odhad, který se řádově může podobat výkonnosti běžně dostupných osobních počítačů.

u složitějších zadání roste potřebný čas exponenciálně. K úspěšnému vypořádání se s problémem je tedy nutné navrhnout jiný postup než zkoušení všech možných řešení a vybírání pouze těch slibných.

Frekvenční analýza není účinná.
Řešení monoalfabetické substituční šifry samo o sobě není příliš zajímavý problém. Arabové dokázali již ve středověku prolomit kód pomocí frekvenční analýzy textu (Singh, 2003). Tento postup je ale založen na velkém množství dat a pro krátký text, se kterým pracujeme, není možné metodu úspěšně přímo aplikovat.

Medián[7] délek šifer je pouze 16 písmen. Málokdy se v šifře vyskytne nějaký symbol více než dvakrát a rozdíl mezi písmeny, které se vyskytují takto řídce není nijak statisticky významný. Použitelnost frekvenční analýzy textu je z tohoto důvodu v našem případě značně omezená.

Je dostupné malé množství trénovacích dat.
Abychom mohli prozkoumat jazyk šifrovacích her, budeme potřebovat skutečné vzorky jazyka. Historicky proběhlo na českém území kolem 400 šifrovacích her. Některé vůbec neodpovídají konceptu stanovišť, který zkoumáme, jiné naopak nemají dostupné žádné webové stránky a i pokud by se z každé hry podařilo získat průměrně 15 šifer, jednalo by se pouze o vzorek několika tisíc velmi krátkých úseků textu. Je to samozřejmě to jediné, co je možné použít, ale značná limitovanost těchto dat bude mít rozhodně velký vliv na celý projekt.

Jazyk je velmi těžko analyzovatelný.
Jak jsme již nastínili v Sekci 1.3, jazyk šifrovacích her je zkratkovitý a heslovitý. Často navíc neodpovídá ani syntaktickým pravidlům češtiny. Bude velmi obtížné sestavit kompletní slovník všech možných pojmů, které se mohou ve výsledcích šifer vyskytnout.

Které řešení je lepší, když neznáme to správné?
I kdybychom měli dostatek výpočetního času na analýzu všech možných řešení, stále musíme rozhodnout, které z nich je lepší. Přirozeně bychom chtěli vybrat to s největší mírou správnosti, ale k její vyhodnocení bychom potřebovali mít správné řešení, které bohužel neznáme. Proto se musíme spokojit s nějakou aproximací správnosti a půjde o skloubení analýzy trénovacích dat s vhodným návrhem jazykového modelu. Volba této porovnávací funkce může velmi zásadně ovlivnit úspěšnost celé práce při hledání kvalitních řešení.

1.6 Struktura práce

Seznámili jsme se se základními pojmy a víme, jak může taková šifra vypadat. V předchozí sekci jsme blíže prozkoumali studovaný problém a nyní stručně popíšeme, v jakém pořadí budeme dále postupovat.

V kapitole *Sběr dat* rozebereme všechny druhy dat, které jsou potřeba k úspěšnému návrhu dešifrátoru, trénování jeho parametrů, a dokonce i k samotnému běhu.

Následující dvě kapitoly *Jazykový model* a *Algoritmus* popisují dvě nejdůležitější součásti dešifrátoru. Obě části jsou téměř nezávislé a drobné změny v jedné z nich nijak neovlivní tu druhou.

Celou práci zakončíme kapitolou *Dokumentace* a diskuzí nad *Výsledky*.

Kapitola 2

Sběr dat

V této kapitole popíšeme, jak získáme potřebná data pro následnou analýzu jazyka a návrh vhodného algoritmu. Vzorek správných řešení reálných šifer navíc využijeme také na trénování parametrů dešifrátoru a v neposlední řadě k závěrečnému vyhodnocení úspěšnosti vytvořeného dešifrátoru. Všechna data volíme výhradně z volně dostupných ke stažení na internetu.

Nejprve specifikujeme všechny kategorie dat, které budeme potřebovat. Pro každou z nich poté vybereme vhodný zdroj a dále navrhneme skript, který bude vybraná data automaticky stahovat. Někdy je navíc potřeba získané informace vhodně naformátovat a uspořádat tak, aby se s nimi poté dalo efektivně manipulovat.

2.1 Vzorek jazyka šifrovacích her

Jak jsme již nastínili, nejprve potřebujeme získat dostatečně velký vzorek jazyka, abychom jej mohli analyzovat a použít jeho specifik. Budeme využívat výhradně řešení šifer z reálných šifrovacích her. Mohli bychom si vymyslet vlastní slovní spojení, která by se podobala popisům dalšího stanoviště, ale ztratili bychom tak věrohodnost a mohli bychom přijít o důležité detaily jazyka. Navíc není možné použít taková data při testování.

Na druhou stranu se neomezujeme pouze na monoalfabetické substituční šifry, neboť jazyk použitý v jiných typech šifer je v zásadě stejný. Přesto jsme značně limitováni celkovým počtem šifer, které kdy na českém území byly řešeny. V předchozí kapitole jsme odhadli celkový počet takových šifer na několik tisíc.

V tomto ohledu máme jen velmi úzký výběr mezi dostupnými zdroji a použijeme v šifrovací komunitě uznávaný kalendář šifrovacích her[1]. Z tohoto seznamu má 42 her své webové stránky, ale pouze 20 z nich (viz Tabulka 2.1) je terénních a na stránkách zveřejňuje správná řešení jednotlivých šifer.

Stránky každé z těchto her jsme prozkoumali a vytvořili regulární výraz, který ze staženého HTML vybere správný výsledek dané šifry. Takto jsme ze stránek šifrovacích her získali dohromady 961 správných řešení šifer. U některých z nich jsme měli výsledek již v textové podobě u jiných byl výsledek reprezentován obrázkem, nebo pouze zapsaný v textu tak, že nebyl možný strojově vyhledat. Zbylé šifry jsme tedy ručně prošli a řešení opsali.

[1] `http://kalendar.chlyftym.cz`

Šifrovací hra	Použité ročníky	Webové stránky
Bedna	2002 – 2012	`bedna.org`
Bota	2010 – 2012	`bota.chytry.cz`
Budějovické želvování	2012	`sifrovackacb.wz.cz`
Hradecká sova	2005 – 2012	`sova.osjak.cz`
iNula	1. – 6. ročník	`inula.sifrovacky.cz`
Krutá (zá)krůta	1. – 6. ročník	`krutazakruta.sifrovacky.cz`
Leonardo	1. – 3. ročník	`leonardo.ph7.cz`
Lost in Hvozd	2012	`lostinhvozd.wz.cz`
Matrix	2005 – 2012	`velkyvuz.cz/matrix`
NaPALMně	2012	`napalmne.cz`
Noc tapürů	2010 – 2012	`noctapuru.cz`
Palapeli	2013	`palapeli.wz.cz`
Paralaxa	2011 – 2012	`paralaxa.chim.cz`
Plzeňská šifrovačka	1. ročník	`plzenska-sifrovacka.webnode.cz`
Po škole	2009 – 2012	`poskole.podrate.cz`
Po trati	2012 – 2013	`potrati.cz`
Rozjezd	2012	`rozjezd.cz`
Sešlost	2009 – 2011	`seslost.cz`
Studna	2003, 2005 – 2011	`studna.zizkov.org`
TMOU	2004 – 2012	`tmou.cz`

Tabulka 2.1: Seznam šifrovacích her použitých k získání řešení šifer.

Na závěr všechna získaná řešení zašifrujeme použitím náhodné permutace abecedy a získáme věrohodná zadání šifer. Všechny šifry přibližně rovnoměrně rozdělíme do následujících tří skupin:

Trénovací data (`training`) – pomocí těchto řešení šifer navrhneme algoritmus a analyzujeme jazyk. Můžeme je využít v plném rozsahu a také ručně hledat závislosti, které by mohli být při výpočtu užitečné.

Vývojová data (`development`) – tato skupina je taktéž určena k vývoji, ale již pouze k testování, že jsme údaje napočítané na první skupině využili správně. Dále řešení použijeme k ladění parametrů dešifrátoru (Sekce 6.3).

Evaluační data (`evaluation`) – nejsou během vývoje nijak využívána a slouží výhradně k závěrečnému vyhodnocení.

2.2 Slovník

Naprostá většina slov vyskytujících se ve správných řešeních šifer pochází z češtiny. Jako základ našeho slovníku volíme veřejně dostupný seznam českých slov na stránce `www.oficialni.cz/slova/`.

Abychom slovník doplnili o různé zkratky a hovorová slova, která se vyskytují v popisech cest, provedeme analýzu sedmnácti popisů cest dostupných na internetu (obvykle popis typu *„Jak se k nám dostanete?“*). Z těchto popisů do našeho slovníku přidáme ta slova, která se v nich vyskytují nejčastěji.

Další slova, která se v řešeních šifer vyskytují, jsou různá vlastní jména, ať se už jedná o názvy obcí, ulic, nebo například kostelů či památníků. O získávaní tohoto typu údajů blíže hovoří následující pasáž.

2.3 Geografické informace

Názvy (pojmenované entity), které řešení šifer obsahují, jsou výhradně takové, které se dají najít na běžné mapě. Abychom tuto skupinu slov pokryli, využijeme skutečnou mapu. Nejrozšířenější volně dostupnou mapou v elektronickém formátu je nyní ta, kterou poskytuje OpenStreetMap[2].

Jedná se o velice podrobné mapy, což vyžaduje netriviální extrakci těch informací, které jsou důležité pro naše účely. Vybereme-li z dostupných dat pouze názvy ulic, obcí a zastávek hromadné dopravy, získáme seznam rozumné velikosti (přibližně 92 MB). Společně s názvy zjistíme z map také geografické souřadnice jednotlivých míst. U rozměrnějších objektů jako jsou ulice bereme v úvahu pro jednoduchost pouze průměr souřadnic, kterými je objekt popsán.

Tabulka 2.2 vyjadřuje počty extrahovaných názvů zeměpisných míst z jednotlivých skupin.

Typ zeměpisného místa	**Počet objektů**
ulice	2 454 802
obec	10 391
autobusová zastávka	9 369
železniční stanice	3 020
tramvajová zastávka	1 440
hospodářský objekt	1 090
celkem	2 480 112

Tabulka 2.2: Počty použitých geografických názvů.

2.4 Rozdělení řešení na jednotlivá slova

Vzhledem k dalšímu postupu je nezbytné rozdělit řešení šifer na jednotlivá slova. K tomu není možné využít žádný automatický nástroj, neboť nemáme zdaleka kompletní slovník. Navíc poté můžeme využít rozdělených řešení k nalezení často používaných zkratek či jiných chybějících slov ve slovníku. Ručním dělením na slova sice mohou vzniknout nepřesnosti a chyby způsobené lidským faktorem, ale pro další vývoj by se nemělo jednat o nic zásadního.

Analyzujeme-li rozdělená řešení, zjistíme, že jen u 49 % všech šifer v trénovacích a vývojových datech máme všechna slova v nich obsažena v našem slovníku. Jelikož hledáme pouze taková řešení, která se skládají ze slov ve slovníku, značně tím klesá naděje na úspěšnost při řešení odpovídajících šifer.

[2] `http://download.geofabrik.de/europe/czech-republic.html`

Kapitola 3

Jazykový model

Značně specifický jazyk šifrovacích her, jak je popsán v Kapitole 1 je poměrně náročné popsat nějakým modelem. Vzhledem k velmi malému množství trénovacích dat se pokusíme použít jeden z nejjednodušších a nejpoužívanějších přístupů, takzvaný Katzův back-off model (Katz, 1987) (Manning a Schütze, 1991).

Tento model předpokládá, že již známe všechna slova zkoumaného jazyka. Seznamu všech slov jazyka šifrovacích her budeme říkat *slovník*. Cílem této kapitoly je co nejlépe vystihnout a popsat syntaktickou stránku jazyka pomocí nějakých konkrétních číselných hodnot.

3.1 Slovní třídy

Zmíněný Katzův model pracuje obvykle s jednotlivými slovy, ale to je vzhledem k velmi malému množství trénovacích dat zcela nevyhovující. Přímou aplikací bychom dospěli k závěru, že většina slov ze slovníku se v jazyce vůbec nevyskytuje. Proto shlukneme související slova do *slovních tříd* tak, aby se slova z každé třídy vyskytovala v rozumné míře v trénovacích datech.

Vzhledem k zeměpisnému zaměření jazyka je přirozené určit následujících pět slovních tříd:

ulice – seznam názvů ulic v lokalitě, kde šifrovací hra probíhá (např. v Praze)

obec – seznam názvů obcí poblíž lokality, kde šifrovací hra probíhá

místo – často vyskytující se místa, kde může být následující šifra fyzicky ukryta

směr – seznam nejpoužívanějších světových stran a jejich zkratek

barva – seznam barev užívaných na turistických značkách

Další skupinu slovních tříd získáme analýzou různých popisů cest. Z těchto popisů vybereme taková slova, která se v nich vyskytují nejčastěji. Jsou to tedy česká slova, která mají ovšem zvýšenou pravděpodobnost výskytu ve správných řešeních šifer:

běžné sloveso – často vyskytující se slovesa, př. `HLEDEJTE`, `JDĚTE`, `ZAHNĚTE`

časté slovo – slova získaná analýzou popisů cest, př. `MINUT`, `ROVNĚ`, `PŘIBLIŽNĚ`

slovo ulice – obsahuje slovo `ULICE` a jemu podobná slova, př. `CESTA`

běžné spojení – často používaná slovní spojení v řešeních šifer, př. `DALŠÍ ŠIFRA`

Poslední skupinou jsou ty slovní třídy, které nemají na první pohled k šifrovacím hrám žádný zvláštní vztah. Přesto se tato slova samozřejmě v řešeních šifer často vyskytují a musíme je tedy zahrnout. Obecně by tyto slovní třídy měly obsahovat všechna ostatní slova a ideální by bylo rozdělení na slovní druhy. Bohužel použitý slovník popsaný v Sekci 2.2 nenabízí takovou možnost a vyčleníme alespoň ty slovní třídy, které se dají snadno vyjmenovat ručně:

písmeno – seznam všech písmen použité anglické abecedy

spojka – všechny české spojky

předložka – všechny české předložky

ostatní – slova, která jsme nezařadili do jiné slovní třídy

3.2 Použité pojmy a značení

Ve zbytku kapitoly budeme potřebovat přesnější značení:

Řetězec je konečná posloupnost písmen abecedy. Nechť s je řetězec, potom $|s|$ nazýváme **délkou řetězce**, která je rovna počtu písmen v s. Pro každé $i \in \{1, 2, \ldots, |s|\}$ značíme i-té písmeno řetězce s symbolem s_i.

Slovní třída je množina řetězců. Označme množinu všech slovních tříd, se kterými pracujeme, symbolem $\mathscr{C}$.

Slovo je dvojice (s, T) taková, že $s \in T \in \mathscr{C}$. Nechť w je slovo. Prvnímu prvku dvojice tvořící w budeme říkat řetězec[1] slova w a značit $\mathcal{R}(w)$, zatímco druhému prvku slovní třída slova w a značit $\mathcal{T}(w)$.

Věta je konečná posloupnost slov. Nebude-li to matoucí, budeme někdy pro jednoduchost uvažovat větu jako zřetězení řetězců jejích slov.

Slovník je množina všech slov.

n**-gram** je uspořádaná n-tice *slovních tříd*. Budeme jej značit $[T_1, T_2, \ldots, T_n]$, kde $T_1, T_2, \ldots, T_n \in \mathscr{C}$. i-tý člen n-gramu G budeme značit G_i.

[1] Dva stejné řetězce ještě nemusí nutně tvořit stejné slovo, pokud jsou obsažené v různých slovních třídách. Např. řetězec `CERVENA` může tvořit jak slovo (`CERVENA`, `barva`), tak slovo (`CERVENA`, `ulice`).

3.3 n-gramová analýza dat

Dále budeme analyzovat vzorky správných řešení šifer. Předpokládejme, že všechna tato řešení jsou již rozdělena na slova. Bohužel při rozdělování může nastat případ, kdy není jasné, ze které slovní třídy řetězec pochází, případně nepatří vůbec do žádné. V takovém případě vezmeme v úvahu všechny možnosti slovních tříd, do kterých může slovo patřit.

Uvažujme nyní přirozené číslo n_{max}, o jehož optimální volbě pohovoříme v Kapitole 6.

Pro každý n-gram $G = [G_1, G_2, \ldots, G_n]$ (kde $1 \leq n \leq n_{max}$) spočteme kolikrát se v *trénovacích datech* vyskytuje n-tice po sobě jdoucích slov $[w_1, w_2, \ldots, w_n]$ taková, že pro každé i (kde $1 \leq i \leq n$) platí $\mathcal{T}(w_i) = G_i$. Například pro bigram [předložka, ulice] vyhodnotíme, kolika názvům ulic předchází nějaká předložka. Tomuto počtu budeme říkat **(absolutní) četnost n-gramu** G a značit ji $\mathcal{C}(G)$.

Abychom mohli spočtené hodnoty přímo použít při modelování jazyka, přidáme speciální slovo na začátek všech šifer (`START`) a jiné speciální slovo na konec všech šifer (`STOP`). Každé z těchto nově přidaných slov se nachází v samostatné slovní třídě. Celá tato analýza spěje k tomu, že budeme umět precizně vyjádřit například velmi nízkou pravděpodobnost předložky na konci zprávy, neboť bigram [předložka, `STOP`] nejspíše nalezneme v trénovacích datech jen zřídkakdy a bude mít tedy malou četnost.

3.4 Pravděpodobnost následujícího slova

Předpokládejme, že již známe část textu (větu) ve zkoumaném jazyce a zajímá nás, jak bude asi text pokračovat. Pokud zpráva začíná

```
DALŠÍ ŠIFRU NALEZNETE NA ...
```

nejspíše další slovo nebude ani `PROTOŽE` ani `HLEDEJTE`, ale může jím být například `KONCI`, nebo `HŘIŠTI`.

Takovou vlastnost jazyka můžeme nyní již poměrně snadno vystihnout pomocí dříve spočtených četností n-gramů. Vzhledem k velmi malému počtu trénovacích dat provedeme několik zjednodušení a většina konstant původního Katzova modelu bude mít triviální hodnoty.

Nechť $[T_1, T_2, \ldots, T_{L-1}]$ je $(L-1)$-gram odpovídající prvním $L-1$ slovům textu. Dále nechť K je přirozené číslo[2] představující hranici na minimální četnost n-gramu (Katz, 1987).

Pravděpodobnost třídy T_L následujícího slova definujeme předpisem

$$P(T_L|T_1, T_2, \ldots, T_{L-1}) = \frac{\mathcal{C}([T_{L-n+1}, T_{L-n+2}, \ldots, T_{L-1}, T_L])}{\sum_{T \in \mathscr{C}} \mathcal{C}([T_{L-n+1}, T_{L-n+2}, \ldots, T_{L-1}, T\ ])},$$

kde n je největší přirozené číslo menší než n_{max} a maximálně L takové, že jmenovatel uvedeného zlomku je větší nebo roven konstantě K.

Uvedené omezení na hodnotu n jinými slovy říká, že se nejprve snažíme použít (n_{max})-gramy, neboť by měly být nejpřesnější. Pokud je ale jejich četnost příliš malá, omezíme se pouze na $(n_{max}-1)$-gramy a takto pokračujeme dále, dokud četnost n-gramů není alespoň K (odtud název back-off).

[2] K je voleno tak, aby bylo menší než četnost každého unigramu. Použili jsme hodnotu 15.

3.5 Cena věty

Pro jednoduchost dále předpokládejme, že všechna slova z jedné slovní třídy jsou stejně pravděpodobná. Shlukli jsme je do tříd, abychom mohli odhadnout pravděpodobnost také pro slova, o kterých nemáme žádné informace z trénovacích dat. Použijeme pro ně pravděpodobnost jim podobných či příbuzných slov. U některých slovních tříd jako jsou *barva* nebo *směr* toto zjednodušení odpovídá velice přesně realitě, zatímco v jiných třídách (např. *předložka* nebo *spojka*) se pravděpodobnosti jednotlivých slov zásadně liší. Bohužel nemáme dostatečné množství trénovacích dat, abychom takové rozdíly mohli rozlišit, a proto uvažujeme uvedené zjednodušení pro všechny třídy.

Pravděpodobnost slova w_L , které následuje po skupině slov $[w_1, w_2, \ldots, w_{L-1}]$ pak definujeme následovně:

$$p\left(w_L|w_1, w_2, \ldots, w_{L-1}\right) = \frac{P(\mathcal{T}\left(w_L\right)|\mathcal{T}\left(w_1\right), \mathcal{T}\left(w_2\right), \ldots, \mathcal{T}\left(w_{L-1}\right))}{|\mathcal{T}\left(w_L\right)|}$$

Pravděpodobnost věty $[w_1,\ w_2,\ \ldots,\ w_L]$ je rovna

$$\mathscr{P}\left([w_1, w_2, \ldots, w_L]\right) = \prod_{i=1}^{L} p\left(w_i|w_1, \ldots, w_{i-1}\right).$$

Uvažme dále přirozený logaritmus pravděpodobnosti věty:

$$\begin{aligned}
&\log\left(\mathscr{P}\left([w_1, w_2, \ldots, w_L]\right)\right) = \\
&\quad = \log\left(\prod_{i=1}^{L} p\left(w_i|w_1, \ldots, w_{i-1}\right)\right) \\
&\quad = \sum_{i=1}^{L} \log\left(p\left(w_i|w_1, \ldots, w_{i-1}\right)\right) \\
&\quad = \sum_{i=1}^{L} \log\left(P\left(\mathcal{T}\left(w_i\right)|\mathcal{T}\left(w_1\right), \ldots, \mathcal{T}\left(w_{i-1}\right)\right)\right) - \log\left(|\mathcal{T}\left(w_i\right)|\right)
\end{aligned}$$

Dále budeme hledat takové řešení, které odpovídá větě s co největší pravděpodobností. Snažíme se tedy maximalizovat logaritmus pravděpodobnosti věty. Bohužel mohutnosti slovních tříd nejsou relevantní hodnoty, neboť většina slov ze slovníku se v trénovacích datech vůbec nevyskytuje. Uvažujme dále pro zjednodušení, že všechny slovní třídy mají stejnou velikost[3] $|T|$. Přirozená volba této konstanty je průměrný počet prvků slovní třídy a v Sekci 6.3 ukážeme, že její optimální hodnota se od průměru[4] skutečně o mnoho neliší. Pravděpodobnost třídy dále doplníme o zmíněnou *„průměrnou“* velikost slovních tříd a výsledný model poté bude upřednostňovat texty obsahující méně slov.

[3] To pro naše slovní třídy neplatí, ale u menších tříd se jejich velikost příliš neliší a pro větší třídy máme moc málo trénovacích dat k tomu, abychom určili jejich *„efektivní“* velikost.

[4] Naše slovní třídy obsahují průměrně 205 174 řetězců.

Cena slova w_L ve větě $[w_1, w_2, \ldots, w_L]$ je rovna

$$\text{cena}\left(w_L | w_1, \ldots, w_{L-1}\right) = \mathscr{L} \cdot \log\left(P(\mathcal{T}\left(w_L\right) | \mathcal{T}\left(w_1\right), \ldots, \mathcal{T}\left(w_{L-1}\right))\right) + \mathscr{D}.$$

Záporná konstanta $\mathscr{L}$ je zde kvůli jednodušší manipulaci s čísly. Její přesná hodnota není podstatná, ale volili jsme $\mathscr{L} = -20$, což umožňuje ceny zaokrouhlit s malou relativní chybou a zacházet s nimi jako s přirozenými čísly.

$\mathscr{D}$ je konstanta odpovídající hodnotě $-\mathscr{L} \cdot \log\left(|T|\right)$. Je vždy nezáporná a nulová právě tehdy, když uvážíme slovní třídy jednotkové velikosti. O její volbě pohovoříme blíže v Sekci 6.3.

Cena věty $[w_1, w_2, \ldots, w_L]$ je rovna

$$\text{Cena}\left([w_1, w_2, \ldots, w_L]\right) = \sum_{i=1}^{L} \text{cena}\left(w_i | w_1, \ldots, w_{i-1}\right).$$

Budeme uvažovat, že levnější věty jsou pravděpodobnější.

Pozn. Hodnota $\mathscr{D}$ toho v aktuálním smyslu již nemá mnoho společného s původním významem. Místo velikosti slovních tříd vyjadřuje penalizaci za každé další slovo použité ve větě. Bylo by tedy přirozenější uvažovat cenu slova bez hodnoty $\mathscr{D}$ a cenu věty ekvivalentně definovat jako součet cen slov zvětšený o součin $L \cdot \mathscr{D}$. Pro implementační účely a pro návrh algoritmu je ovšem jednodušší pracovat čistě se součtem cen. Proto se budeme držet původního návrhu a rozdělíme celkovou penalizaci rovnoměrně jednotlivým slovům, přestože to může být matoucí a neintuitivní.

Zdaleka se nejedná o ideální model, ale jeho velkou výhodou je jednoduchost a předvídatelnost. Navíc je zbytek práce téměř nezávislý na tom, jaký model zvolíme a bylo by ho proto možné samostatně vylepšovat.

3.6 Využití znalosti aktuální polohy

Mnohem lepších výsledků bychom mohli dosáhnout, pokud by dešifrátor využíval informaci o aktuální poloze a předpokládali bychom, že následující šifra nebude v příliš velké vzdálenosti. Vzhledem k tomu, že obvykle je k přesunům mezi stanovišti dovolena pouze chůze, jedná se o poměrně rozumný předpoklad. Dodatečná informace o stávající pozici značně omezí seznam geografických míst, která se mohou ve výsledku nacházet, a může výrazně zpřesnit řešení.

Z vhodně naformátované mapy vybereme pouze ty pojmenované entity, které nejsou vzdušnou čarou vzdálenější než zadaná konstanta popsaná v Sekci 5.4.

Znalost aktuální polohy by měla značně zvýšit přesnost jazykového modelu a eliminovat falešné výsledky navádějící například do desítky kilometrů vzdáleného města.

Kapitola 4

Algoritmus

Předcházející kapitola jednoznačně popisuje, jak budeme odhadovat pravděpodobnost jednotlivých řešení. Dosud jsme ale vůbec nezmínili, jak řešení procházet a efektivně hledat ta kvalitnější. Možných přístupů je mnoho a i při tvorbě této práce jsme jich několik zvážili a některé zkusili implementovat. Nakonec se ale ukazuje jako nejvhodnější jeden z nejpřímočařejších algoritmů, který ve zbytku kapitoly podrobněji rozebereme.

4.1 Základní myšlenka

Program načte zadání šifry a snaží se určit, které slovo by mohlo být první ve správném výsledku. Za tímto účelem vyzkouší všechny možnosti ze slovníku. Pro každou možnost zkusí pokračovat dále určením druhého slova. Zde bude nejspíše méně možností, neboť některá písmena jsou již přesně určena zvolením prvního slova. Takto algoritmus pokračuje a postupně prochází *částečná řešení*, dokud nenalezne nějaké řešení.

Částečné řešení dané šifry je věta tvořená L písmeny taková, že existuje převodní tabulka, která překládá prvních L symbolů zadání šifry na větu.

Jako ukázku uvažme následující zadání:[1]

`lavickauhvezdarny`

Částečnými řešeními této šifry jsou například věty: [`LAVICKA`], [`LAVICKA`, `U`, `HVEZDARNY`], nebo [`DALSI`]. Na druhou stranu věta [`DALSI`, `SIFRA`] ani [`KRIZOVATKA`] nejsou částečnými řešeními[2] uvedené šifry.

Protože námi užívaný slovník obsahuje všechna jednotlivá písmena abecedy, můžeme částečné řešení ekvivalentně popsat tím, že ho lze doplnit slovy ze slovníku, aby vzniklo nějaké řešení dané šifry.

Slovo S **rozšiřuje** částečné řešení R dané šifry, pokud po přidání slova S za větu R vznikne opět částečné řešení šifry.

[1] Symboly v tomto případě volíme pro názornost tak, aby odpovídaly správnému řešení šifry. Obyčejně to ale neplatí a stejné zadání by mohlo být popsáno např. `uwzidvwptzebcwryq`.

[2] V prvním příkladu se shoduje čtvrté písmeno s šestým, což odporuje zadání, zatímco v druhém se liší druhé písmeno od sedmého, což zadání šifry naopak vyžaduje.

Jedná se tedy v podstatě o backtracing stavovým prostorem všech možných částečných řešení. Takový postup má zřejmě exponenciální časovou složitost vzhledem k délce zadaného textu. Na první pohled to vypadá jako zásadní úskalí, ale přesto ukážeme, jak tuto velice jednoduchou myšlenku vylepšit a zdokonalit. Bude se jednat o kombinaci vhodné heuristiky, postupného prořezávání možností a efektivní implementace souvisejících primitiv.

První vylepšení přináší následující pozorování. Pokud je cena částečného řešení větší nebo rovna ceně dosud nejlepšího nalezeného řešení, nemá smysl v prohledávání pokračovat. Snadno nahlédneme, že ceny všech slov jsou vždy kladné, a proto by cena dokončeného řešení byla větší než dosud nejlepšího nalezeného. Uvedené pozorování pomáhá značně prořezat neperspektivní větve prohledávání.

4.2 Vyhledávání rozšiřujícího slova

K úspěšné a efektivní implementaci musíme nejprve navrhnout řešení následujícího algoritmicky zajímavého problému:

Problém. *Uvažujme částečné řešení R nějaké šifry. Nalezněte všechna slova, kterými je možné R rozšířit, tj. po rozšíření R vznikne částečné řešení té samé šifry.*

Přímočarým řešením problému by bylo vyzkoušet všechna slova ze slovníku a o každém z nich rozhodnout, zda vznikne částečné řešení oné šifry (tedy zda je nové slovo v souladu s převodní tabulkou odpovídající částečnému řešení). Krátké částečné řešení lze obvykle rozšířit většinou slov ze slovníku, ale když se výpočet dostane k delšímu částečnému řešení, pak se popsaný neefektivní přístup stává úzkým hrdlem a kritickým bodem celého výpočtu. Přestože slov, kterými je možné částečné řešení rozšířit, je často pouze několik stovek, zkoušíme zbytečně miliony slov z celého slovníku.

Efektivnější řešení problému budeme ilustrovat na příkladu. Uvažujme následující zadání šifry:

```
fosjavpuajwgopuswzefw
```

Jedním z jejich částečných řešení je:

```
KONEC ULICE
```

To odpovídá částečně vyplněné převodní tabulce:

```
a -> C, f -> K, j -> E, o -> O,
p -> L, s -> N, u -> I, v -> U
```

Z toho plyne, že při rozšiřování částečného řešení musíme pokračovat slovem tvaru (rozšiřující slovo může být i kratší):

```
wgOLINwzeKw
```

Pokud si symboly přeznačíme malými písmeny anglické abecedy v přirozeném pořadí, dostaneme jednoznačný tvar bez ohledu na dosud použité symboly:

`abOLINacdKa`

Jedním z možných rozšíření zkoumaného řešení je například slovo `APOLINARSKA`.

Normalizované pokračování částečného řešení: Uvažme částečné řešení R dané šifry. Označme počet písmen v R jako L. Nejprve prvních L symbolů ze zadání šifry nahradíme v celém zadání odpovídajícím překladem z převodní tabulky příslušící R. Zbylé nepřeložené symboly postupně přeznačíme na $a, b, c, \ldots$, podle prvního výskytu v upraveném zadání. Když na závěr ze vzniklé posloupnosti vypustíme prvních L symbolů, dostáváme normalizované pokračování R.

Dále zapišme všechna slova ve slovníku všemi možnými způsoby tak, aby se stačilo podívat na normalizované pokračování částečného řešení a abychom okamžitě věděli, kterými slovy lze částečné řešení rozšířit.

Například slovem `KROK` můžeme rozšířit jedině to částečné řešení, jehož normalizované pokračování začíná jednou z následujících osmi možností:

```
KROK
KRaK
KaOK
KabK
aROa
aRba
abOa
abca
```

Podobně můžeme ke každému slovu ze slovníku vygenerovat všechny *varianty*, kterých je

$$2^{\text{počet různých písmen v daném slově}}.$$

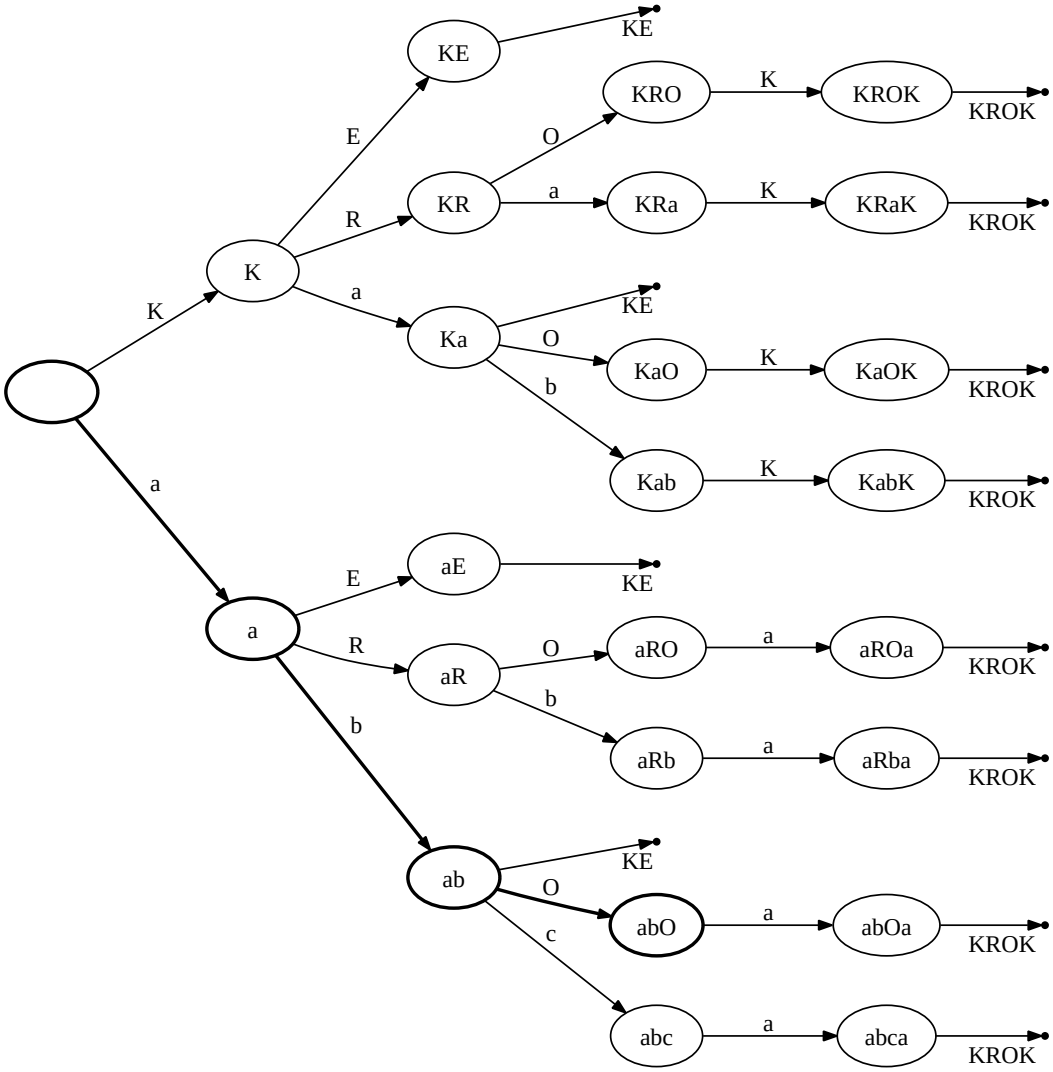
Varianta slova S délky L je posloupnost symbolů a písmen abecedy délky L, pro kterou existuje šifra a její částečné řešení R takové, že normalizované pokračování R začíná přesně členy této posloupnosti.

Variant každého slova by sice mohlo mít slovo teoreticky až 2^{26}, ale slova ze slovníku obsahují průměrně pouze 7,7 různých písmen.

Dále uložíme všechny varianty všech slov do struktury známé pod pojmem trie. Obrázek 4.1 znázorňuje trii vytvořenou ze všech variant slov `KROK` a `KE`. Dále je na něm vyznačená cesta, kterou by algoritmus prošel při hledání rozšiřujících slov k již diskutovanému normalizovanému pokračování `abOLINacdKa`. Pro tento omezený slovník obsahující pouze dvě slova vyhledávání skončí ve vrcholu `abO`, zatímco normálně bychom dále pokračovali do vrcholu `abOL`. Vybudování struktury trie vyžaduje lineární čas vzhledem k souhrnné délce všech vkládaných řetězců a je tedy velice efektivní. Navíc si u každého vrcholu trie budeme udržovat seznam slov, které jsou tímto vrcholem reprezentovány.

Předpokládejme, že máme částečné řešení a zajímalo by nás nyní, jakými slovy ho lze rozšířit. Budeme přímočaře postupovat v dosud nerozluštěné části

zadání, používat známé překlady a neznámé symboly postupně indexovat malými písmeny anglické abecedy, čímž vlastně dostaneme normalizované pokračování částečného řešení. Spolu s tím se budeme pohybovat jedním ukazatelem po trii. Po každém přečteném symbolu se můžeme posunout také v trii do příslušného vrcholu. Všechna slova, která jsou u aktuálního vrcholu, mohou teoreticky rozšiřovat zkoumané částečné řešení. Průchod končí pokud se dostaneme na konec normalizovaného pokračování částečného řešení, nebo když z aktuálního vrcholu v trii nevede požadovaná hrana.



Obrázek 4.1: Trie obsahující všechny varianty slov KROK a KE

Tímto způsobem značně omezíme množinu kandidátů na rozšiřující slovo. Navíc se jedná o efektivní operaci, která spotřebuje pouze lineární množství času vzhledem k maximální délce z vrácených slov a k jejich počtu.

Ukazuje se, že krátká slova nejsou problém, ale dlouhá slova s mnoha různými písmeny mohou mít hodně variant a spotřebují neúměrně mnoho paměti. Je tedy potřeba nastavit nějakou hranici pro počet písmen ve slově, dokdy jsme ochotni generovat všechny varianty. Naštěstí pro dlouhá slova se již mnohem lépe zapojí efekt, kdy jsou ve slově alespoň dvě stejná písmena a pasují skutečně pouze ta slova, která vykazují velkou shodu. Pro dlouhá slova tedy uvažujeme pouze va-

riantu složenou výhradně ze symbolů. Velmi jednoduchá modifikace popsaného algoritmu si poradí i s tímto případem, pouze omezuje množinu kandidátů o něco méně.

4.3 Pořadí zkoušených slov

Uvažujme nyní částečné řešení nějaké šifry a seznam všech slov, které jej rozšiřují. Ideálně bychom chtěli nejprve pokračovat tím slovem, které nejspíše povede k nejlepšímu řešení. Pro průběh celého výpočtu může být zcela zásadní, kterým slovem se rozhodneme pokračovat.

Jelikož pro každé slovo již známe podmíněnou pravděpodobnost, se kterou se na zkoumaném místě bude nacházet, zdá se být přirozené uspořádat slova sestupně podle této pravděpodobnosti. Když ale budeme vybírat velice krátká slova, budeme jich na vyplnění celého řešení potřebovat mnoho a přestože jednotlivá slova mají poměrně vysokou pravděpodobnost, může jejich součin být ve výsledku mnohem menší než při použití jednoho dlouhého slova s trochu nižší pravděpodobností.

Nabízí se možnost slova uspořádat sestupně podle jejich délek. To bychom ale zase vůbec nebrali v úvahu jejich pravděpodobnost a preferovali bychom velmi nepravděpodobné slovo například s patnácti písmeny před mnohem pravděpodobnějším slovem obsahujícím jen čtrnáct písmen. Délka slova v tomto příkladu zřejmě není zásadní a velký rozdíl pravděpodobností může být mnohem důležitější.

Všechno zatím směřuje k tomu, že optimální heuristika bude někde mezi těmito extrémy. Bude potřeba nějak vhodně zkombinovat délku slova a pravděpodobnost, že jím bude řešení pokračovat. Dokonce i naše experimenty ukazovaly, že kombinace těchto hodnot, kterou jsme intuitivně vyzkoušeli, dávala mnohem lepší výsledky než předcházející dvě možnosti.

Zkusme nyní formálněji dospět ke stejnému výsledku, který jsme zatím jen intuitivně odhadli a experimentálně ověřili. Uvažujme větu $[w_1, w_2, \ldots, w_S]$. Pro každé $i \in \{1, 2, \ldots, S\}$ označme nyní $p_i = p\left(w_i | w_1, w_2, \ldots w_{i-1}\right)$ a $l_i = |\mathcal{R}\left(w_i\right)|$ (počet písmen ve slově w_i).

Lemma. *Rozdělíme-li pro každé $i \in \{1, 2, \ldots, S\}$ slovo w_i na jednotlivá písmena a těm všem přiřadíme pravděpodobnosti $\sqrt[l_i]{p_i}$, potom pravděpodobnost původní věty bude shodná s pravděpodobností nově vytvořené věty.*

Uvedené lemma dokážeme snadno tím, že spočteme pravděpodobnost věty jako součin pravděpodobností jednotlivých slov a pravděpodobnosti písmen i-tého slova po umocnění na l_i-tou dávají opět pravděpodobnost celého slova.

Dále si všimněme, že každé kompletní řešení zadané šifry má stejný počet písmen. Když částečné řešení rozšiřujeme novým slovem, díky zmíněnému lemmatu se nabízí, že bychom měli vybrat takové slovo, které svým písmenům dává co největší pravděpodobnost a maximalizuje tedy hodnotu

$$\sqrt[\text{délka(slovo)}]{\text{pravděpodobnost(slovo)}}.$$

Jelikož je funkce logaritmus rostoucí, chceme maximalizovat také hodnotu

$$\log\left(\sqrt[\text{délka(slovo)}]{\text{pravděpodobnost(slovo)}}\right) = \frac{\log\left(\text{pravděpodobnost(slovo)}\right)}{\text{délka(slovo)}}.$$

Uchýlíme-li se ke stejnému zjednodušení jako v Sekci 3.5, dostáváme, že se snažíme minimalizovat

$$\frac{\text{cena(slovo)}}{\text{délka(slovo)}}.$$

Podařilo se nám vhodně zkombinovat požadavek na délku a pravděpodobnost slova a dospět k této heuristice posloupností logických úvah. Jedná se sice stále pouze o hladovou heuristiku, neboť vybráním takového slova, které maximalizuje pravděpodobnost svých písmen, se můžeme snadno dostat do stavu, kdy nebude možné pokračovat žádným výhodným slovem.

4.4 Postupné prořezávání možností

Obzvláště na dlouhých šifrách dosud popsaný algoritmus málokdy stihne projít všechny možnosti a často dokonce pouze zvolí první dvě slova a zbytek času tráví neperspektivním hledáním optimálního řešení pro tento podproblém.

Čas vyhrazený k výpočtu rozdělíme na k kroků, abychom ho využili efektivněji. Rozdělení časového intervalu není rovnoměrné, ale pozdější kroky mají vyhrazený větší úsek. Délka úseků roste lineárně s druhou mocninou čísla kroku.

V prvním kroku necháme algoritmus prohledávat částečná řešení bez omezení. V kroku $i > 1$ postupuje algoritmus normálně pro částečná řešení obsahující méně než $k - i + 1$ slov. Pro ostatní částečná řešení uvažuje vždy jen první slovo ze seznamu rozšiřujících slov. Procházený stavový prostor se tedy větví jen na prvních $k - i + 1$ úrovních rekurze a poté je již lineární. Využitím tohoto postupu se vyhneme *„zaseknutí“* algoritmu ve větvi výpočtu, kde stráví všechen vymezený čas.

4.5 Dvě fáze výpočtu

Čím nižší cenu má dosud nejlevnější nalezené řešení, tím rychleji algoritmus pokračuje. Neztrácí poté čas procházením neperspektivních větví výpočtu, o kterých již ví, že nepovedou k levnějšímu řešení. Z tohoto důvodu na začátku výpočtu rychle prozkoumáme mnoho různorodých řešení, což nám dá lepší odhad na hledanou cenu.

Rozdělíme průběh výpočtu na dvě fáze:

1. fáze prochází všechna slova, kterými může zpráva začínat. Pro každé z nich algoritmus pokračuje hladově a vždy vezme první rozšiřující slovo podle heuristiky popsané v Sekci 4.3.

2. fáze je totožná s dosud popisovaným algoritmem, ale začíná s již předpočítanou množinou nalezených řešení z 1. fáze.

První fáze výpočtu je poměrně rychlá a má zaručenou lineární časovou složitost vzhledem k délce zadání. Není potřeba pro ni stanovit žádný časový limit, neboť i pro extrémně dlouhé šifry trvá její běh nejvýše několik minut.

Naopak druhá fáze může být až exponenciální a musíme ji tedy časově omezit. V Sekci 6.3 hovoříme o tom, jaké limity jsme volili.

Kapitola 5

Dokumentace

5.1 Kompilace programu

K přeložení automatického dešifrátoru slouží přiložený `Makefile`.
Bližší technické infromace jsou dostupné v souboru `README`.

Program	Verze	Komentář
`gcc`	$\geq$ `4.2.1`	
`sh`	$\geq$ `3.2`	potřeba k zašifrování trénovacích dat
`python2`	$\geq$ `2.7.6`	nutné k používání dalších nástrojů (viz 5.5)

Tabulka 5.1: Seznam potřebných programů

5.2 Uživatelská dokumentace

Po spuštění program přečte jediný řádek ze standardního vstupu. Dále z něj vypustí všechny znaky kromě malých a velkých písmen anglické abecedy. Ta jsou poté převedena na malá písmena a chápána jako zadání šifry. Program nijak nezohledňuje použité symboly a využívá tento vstup pouze pro informaci o tom, které symboly jsou stejné, a které různé.

Dešifrátor má dva nepovinné argumenty: zeměpisnou šířku a zeměpisnou délku. Zadávají se jako desetinná čísla a pokud nejsou zadány přesně dva argumenty, nebo když jsou v zadány chybném formátu, ignorují se. Popsané zeměpisné souřadnice vyjadřují aktuální polohu řešitele a omezují vyhledávání řešení pouze na okruh několika kilometrů od zadaného místa. Taková znalost může zásadně zrychlit celý výpočet a umožnit aplikaci dospět k mnohem přesnějším výsledkům.

Výstupem programu je deset[1] řešení zadané šifry, o kterých program usoudí, že jsou nejpravděpodobnější. Společně s každým řešením dešifrátor vypíše jeho celkovou cenu.

Obrázek 5.1 ukazuje jak vypadá výstup dešifrátoru na příkladu z Kapitoly 1.

Jelikož je správné řešení jediný název ulice, není překvapivé, že je toto řešení nalezeno jako první, ale přesto je potěšující, že alespoň nějaká reálná substituční šifra lze vytvořeným programem rozluštit. Čísla vedle jednotlivých řešení jsou ceny výsledných vět.

[1] Případně nějaká jiná konstanta nastavená v konfiguračním souboru.

```
>>> ./decipher
ywosbfuvhstiwfshusou

Reseni:
        43 - NAMESTIPREDBATERIEMI
       168 - LIBOCKA HVOZDIKOVA OBA
       174 - LIBOCKE HVOZDIKOVE OBE
       201 - BRANKE MLYN UTRENYM NAM
       202 - JINAK MOHLA TRI MALO ANO
       202 - JINAK MOHLA ZDI MALO ANO
       204 - LIBOCKA UVOZ SIKOVA OBA
       204 - LIBOCKA UVOZ MIKOVA OBA
       205 - LIBOCKA DVOJNIKOVA OBA
       207 - HLINKY SAUNE MLYNU SNIS
```

Obrázek 5.1: Ukázka vstupu a výstupu dešifrátoru

Další ukázka se nachází na Obrázku 5.2, kde jsou zadány geografické souřadnice šifry, která se nacházela u *Mostu inteligence*. Z této šifry vyšla poloha dalšího stanoviště *Nádraží Braník*, ale to program bez zadání souřadnic aktuálního stanoviště nenalezne, neboť takto krátkému zadání odpovídá mnoho jiných míst.

```
>>> ./decipher 50.024512 14.392334
hdkzdsctzdhcy

Reseni:
        56 - NADRAZI BRANIK
       133 - VEDLE KAPLE VAM
       133 - VEDLE KAPLE VAS
       138 - NADHAJEM HANET
       138 - NADHAJEM HANEL
       140 - VEDLE NAHLE VAS
       140 - VEDLE TAHLE VAS
       140 - VEDLE TAHLE VAM
       140 - VEDLE NAHLE VAM
       148 - NADRAZIM RANIT
```

Obrázek 5.2: Ukázka použití dešifrátoru se zadáním zeměpisných souřadnic

5.3 Programátorská dokumentace

Jako programovací jazyk jsme vzhledem k očekávané výpočetní náročnosti zvolili C++. Poskytuje všechny potřebné nástroje a navíc disponuje dostatečnou výkonností.

Implementace dešifrátoru přímočaře využívá jazykový model a vykonává algoritmus navržený v předchozích kapitolách.

Jazykový model společně s převodní tabulkou a správou slovníku jsou implementovány jako samostatná knihovna[2]. Celou knihovnu poté využívá vyhledávací algoritmus, který do hloubky prohledává stavový prostor částečných řešení a zajišťuje prořezávání neperspektivních možností.

Většina abstraktních objektů je implementována jako samostatná třída, která poskytuje všechnu potřebnou funkcionalitu. Jedná se o třídy jako `Wordlist` (slovník), `ConversionTable` (převodní tabulka), `Solution` (řešení), či `Option` (rozšiřující slovo částečného řešení).

Všechny zdrojové kódy jsou detailněji dokumentovány použitím komentářů.

5.4 Konfigurace dešifrátoru

Konfigurace dešifrátoru je uložena v hlavičkovém souboru `config.h`. Při každé jeho změně je potřeba program znovu přeložit. Jednotlivá nastavení jsou v něm uložena přímo jako konstanty jazyka C++. Vysvětlíme význam některých konfiguračních parametrů.

- `MAX_PRINT_SOLUTIONS` vyjadřuje kolik nejlepších nalezených řešení se má vypisovat
- `PROGRESS` je logická hodnota, která rozhoduje o tom, zda se mají vypisovat ladící informace o aktuálním stavu výpočtu
- `MAX_DISTANCE` značí největší vzdušnou vzdálenost, do které může být následující stanoviště od toho aktuálního
- `EARTH_RADIUS` je poloměr Země, který se použije při výpočtu vzdušné vzdálenosti ze zeměpisných souřadnic

5.5 Další dostupné nástroje

Kromě hlavní aplikace na luštění šifer vzniklo v souvislosti s prací ještě několik drobnějších nástrojů, které se nacházejí ve složce `script/`. Podrobnější informace o jejich použití jsou obsaženy vždy na začátku příslušeného zdrojového kódu.

- `code.cpp` je program sloužící k zašifrování zadaného textu náhodnou převodní tabulkou. Každá převodní tabulka může být vybrána se stjnou pravděpodobností.
- `address.py` je skript sloužící k vyhledávání názvů obcí a ulic v jednotlivých správních územích jako jsou kraje a okresy.
- `divide.cpp` rozděluje všechna stažená řešení na skupinu trénovacích, vývojových a evaluačních dat.
- `download_solutions.py` pomáhá stahovat řešení ze stránek šifrovacích her.

[2]Zdrojový kód se nachází ve složce `src/lib`.

- `download_wordlist.py` stahuje seznam všech dostupných českých slov.
- `gen_costs.py` provádí na zadané skupině dat analýzu n-gramů a generuje v závislosti na spočtených četnostech pravidla pro výpočet cen slov.
- `places.py` normalizuje nalezené zeměpisné názvy a pokouší se skloňovat názvy ulic podle jejich nejčastějších koncovek.
- `score.py` pomáhá vyhodnocovat úspěšnost dešifrátoru. Spustí program na všechna data ze zvolené skupiny a vyhodnotí jeho úspěšnost. Zároveň v přehledném formátu porovnává správná řešení šifer s těmi, které vrátil program.

Kapitola 6

Výsledky

V této kapitole nabízíme čtenáři srovnání různých konfigurací dešifrátoru a jejich úspěšností. Uvádíme také parametry všech experimentů a naměřené hodnoty. Na závěr vyhodnotíme výsledný program na evaluačních datech a slovně shrneme dosažené výsledky.

6.1 Hodnocení úspěšnosti dešifrátoru

Cílem celého projektu je navržení takového dešifrátoru, který bude schopen nalézt správné řešení u co největšího počtu šifer. Navrhneme mechanizmus, jak mezi dvěma dešifrátory rozhodnout, který z nich je úspěšnější.

Mírou správnosti (správnost) nazveme takovou funkci $\mathcal{Q}$, která pro každou šifru se správným řešením q a nějakým dalším řešením p splňuje $\mathcal{Q}(p, q) \in [0, 1]$. Řešení, které považujeme za naprosto přijatelné a vykazuje velkou podobnost se správným řešením, bude mít míru správnosti rovnou jedné, zatímco špatné řešení má správnost nulovou.

Úspěšnost dešifrátoru na dané množině šifer je průměrná správnost řešení, které dešifrátor vrátí pro každou ze šifer z této množiny (vrátí-li pro nějakou šifru řešení více, uvažujeme pouze to s nejvyšší mírou správnosti).

Při vyhodnocování úspěšnosti různých konfigurací dešifrátoru používáme následující míry správnosti:

1. **Binární** míra správnosti přiřadí správnému řešení správost rovnou jedné a všem ostatním nulovou.

$$\mathcal{Q}_{bin}(p, q) = \begin{cases} 1, & \text{pokud } p = q \\ 0, & \text{pokud } p \neq q \end{cases}$$

2. **Relativní shoda řešení** je míra správnosti, která zohledňuje, jak velká část zprávy byla správně rozluštěna. Správnost řešení p spočteme použitím této míry jako počet pozic, na kterých se písmeno v p shoduje s písmenem ve správném řešení q, dělený počtem písmen q.

$$\mathcal{Q}_{sol}(p, q) = \frac{|\{i \; ; p_i = q_i, \; i \in \{1, \ldots, |q|\}\}|}{|q|}$$

3. **Relativní shoda převodní tabulky** je míra správnosti vyhodnocující, kolik písmen odpovídající převodní tabulky bylo správně přiřazeno. Přesněji se jedná o počet různých symbolů, vyskytujících se v zadání šifry, které převodní tabulka odpovídající správnému řešení q zobrazuje stejně jako převodní tabulka odpovídající zkoumanému řešení p, dělený počtem různých symbolů v zadání.

$$\mathcal{Q}_{conv}(p,q) = \frac{|\{q_i \ ; \ p_i = q_i, \ i \in \{1,\ldots,|q|\}\}|}{|\{q_i \ ; \qquad\qquad i \in \{1,\ldots,|q|\}\}|}$$

4. **Manuální evaluace** je míra správnosti, která pomáhá odhadnout využitelnost dešifrátoru v reálných podmínkách. Necháme vybraného člověka pro každé řešení určit, kde by hledal následující stanoviště.

$$\mathcal{Q}_{real}(p,q) = \begin{cases} 1, & \text{pokud člověk po přečtení } p \text{ pochopí, že má jít na } q \\ 0, & \text{v opačném případě} \end{cases}$$

6.2 Parametry algoritmu

`N_MAX` je hodnota konstanty n_{max} popsané v Sekci 3.3.

- `1` – uvažujeme pouze unigramy
- `2` – uvažujeme unigramy a bigramy
- `3` – unigramy, bigramy a trigramy
- `INF` – bez omezení na n_{max}, což ale v praxi znamená, že použijeme nejvíce 4-gramy, neboť žádný 5-gram nemá v trénovacích datech dostatečnou četnost

`COST_DIFF` značí hodnotu konstanty $\mathscr{D}$ popsané v Sekci 3.5. Připomeňme, že $|T|$ značí velikost jedné slovní třídy.

- `0` – uvažujeme slovní třídy jednotkové velikosti
- `50` – odpovídá $|T| \doteq 12$
- `100` – odpovídá $|T| \doteq 148$
- `150` – odpovídá $|T| \doteq 1808$
- `200` – odpovídá $|T| \doteq 22\,026$
- `250` – odpovídá $|T| \doteq 268\,337$
- `300` – odpovídá $|T| \doteq 3\,269\,017$

`PHASE` určuje, které fáze popsané v 4.5 se mají vykonávat.

- `1` – pouze 1. fáze
- `2` – pouze 2. fáze
- `12` – vykonají se obě fáze

`ALG` specifikuje pořadí procházení částečných řešení.

`DFS` – do hloubky, nejprve se algoritmus *„zanoří“* a poté *„backtrackuje“*

`BFS` – do šířky, za použití prioritní fronty algoritmus prochází částečná řešení podle jejich ceny od nejlevnějších po dražší; při vyčerpání dostupné paměti se automaticky přepne na průchod do hloubky

`MAX_LEN_GEN_VARIANTS` omezuje počet písmen ve slově, pro které se budou generovat všechny varianty (viz 4.2). Je žádoucí, aby tato hodnota byla co nejvyšší, aby vybírání možných rozšiřujících slov bylo co nejefektivnější. S rostoucí hodnotou tohoto parametru rychle roste potřebná operační paměť pro uložení celé struktury trie.

`8` – spotřebuje přibližně 840 MB operační paměti

`7` – spotřebuje přibližně 380 MB operační paměti

`ORDER` udává pořadí, ve kterém se zkoušejí rozšiřující slova.

`CL` – v pořadí s rostoucí hodnotou $\frac{\text{cena(slovo)}}{\text{délka(slovo)}}$

`C` – v pořadí s rostoucí cenou slov

`L` – v pořadí s klesající délkou slov

`MAX_TIME` vyjadřuje maximální dobu běhu druhé fáze výpočtu v sekundách.

`60` – zřejmě bude dávat nižší úspěšnost, ale je potřebné k tomu, abychom byli schopni spustit testy na dostatečné množství parametrů; na jedné sadě 320 vstupů trvá výpočet přibližně 5 hodin

`300` – v praxi mnohem realističtější podmínky, ale při testování na celé sadě vstupů zabere výpočet 20 hodin

`GEOGRAPHY` je parametr dešifrátoru, který rozhoduje o využití zeměpisných souřadnic aktuálního stanoviště.

`Y` – souřadnice se použijí

`N` – souřadnice se nepoužijí

6.3 Trénování parametrů

Všechny konfigurace našeho algoritmu testujeme na trénovacích a vývojových datech. Jelikož je dešifrátor navržený pomocí první sady vzorků, očekáváme na druhé sadě menší úspěšnost. Jedná se ale o mnohem relevantnější výsledek, kterým se budeme řídit, neboť mnohem lépe simuluje průběh řešení nových neznámých šifer.

K hodnocení úspěšnosti využijeme první tři metriky popsané v Sekci 6.1. Ukazuje se, že míry správnosti $\mathcal{Q}_{sol}$ a $\mathcal{Q}_{conv}$ dávají na reálných datech velmi podobné výsledky ($\mathcal{Q}_{sol}$ je obvykle o trochu vyšší). Navíc se obě hodnoty pozoruhodně přesně blíží $\sqrt{\mathcal{Q}_{bin}}$. Dalším důležitým aspektem je, že průměrnou míru správnosti $\mathcal{Q}_{sol}$, případně $\mathcal{Q}_{conv}$, lze velmi těžko interpretovat, zatímco průměrná míra správnosti $\mathcal{Q}_{bin}$ vyjadřuje jednoduše procento správně rozluštěných šifer. Z uvedených

důvodů prezentujeme u všech výsledků pouze úspěšnost spočtenou za použití binární metriky. Detailní výsledky všech možných konfigurací, za použití všech měr správnosti můžete nalézt ve složce `results/`.

Abychom mohli různé konfigurace parametrů algoritmu snadno srovnávat, určili jsme počáteční nastavení a poté testovali různé konfigurace jalo jeho jednoduché změny. Vždy změníme právě jeden parametr a zjistíme úspěšnost modifikovaného dešifrátoru. Poté vrátíme aplikaci do původního stavu a experiment opakujeme s jiným parametrem. Nakonec navrhneme pro finální algoritmus takovou konfiguraci parametrů, která bude nejúspěšnější na vývojových datech.

Tabulky 6.1 a 6.2 obsahují výsledky experimentů. Ve sloupci *Konfigurace* je vždy definován pouze ten parametr, který se liší od počáteční konfigurace dešifrátoru.

`INITIAL` – je počáteční konfigurace parametrů

`N_MAX`	=	`2`
`COST_DIFF`	=	`0`
`PHASE`	=	`12`
`ALG`	=	`DFS`
`MAX_LEN_GEN_VARIANTS`	=	`8`
`ORDER`	=	`CL`
`MAX_TIME`	=	`60`
`GEOGRAPHY`	=	`N`

Konfigurace	**Trénovací data**	**Vývojová data**
`N_MAX = 1`	13,1 %	13,2 %
`N_MAX = 3`	13,1 %	13,2 %
`N_MAX = INF`	13,8 %	13,2 %
`COST_DIFF = 50`	14,7 %	14,9 %
`COST_DIFF = 100`	15,3 %	15,9 %
`COST_DIFF = 150`	15,0 %	16,3 %
`COST_DIFF = 200`	14,7 %	16,9 %
`COST_DIFF = 250`	14,4 %	17,3 %
`COST_DIFF = 300`	14,7 %	15,9 %
`PHASE = 1`	12,5 %	9,2 %
`PHASE = 2`	13,4 %	12,9 %
`ALG = BFS`	14,4 %	13,2 %
`MAX_LEN_GEN_VARIANTS = 7`	13,4 %	13,2 %
`ORDER = C`	11,6 %	7,5 %
`ORDER = L`	13,1 %	14,2 %
`MAX_TIME = 300`	14,7 %	13,9 %
`INITIAL`	**14,1** %	**13,6** %

Tabulka 6.1: Úspěšnosti různých konfigurací dešifrátoru za použití míry $\mathcal{Q}_{bin}$

Naměřené výsledky ukazují, že pro zkoumaná data je dešifrátor nejúspěšnější s hodnotou $n_{max} = 2$. Rozdíly v úspěšnsosti jsou u tohoto parametru velmi malé.

Konfigurace	Trénovací data	Vývojová data
GEOGRAPHY = Y	26,5 %	20,0 %
INITIAL	17,6 %	14,3 %

Tabulka 6.2: Testování přínosu zeměpisných souřadnic na menší části dat

Velmi překvapivý je výsledek trénování parametru COST_DIFF. Oproti našim odhadům se jako optimální ukazuje hodnota 250, která odpovídá průměrné velikosti slovní třídy 268 337. To poměrně přesně koreponduje se skutečnou velikosti slovních tříd. Upřesněme, že jazykový model při trénování nemohl nijak přímo zohlednit skutečnou velikost slovních tříd, neboť byl trénován pouze na 7 659 slovech z trénovací sady vzorků.

Dále za povšimnutí stojí výsledek konfigurace ORDER = L. Ta je na vývojových datech úspěšnější než konfigurace ORDER = CL navržená v Sekci 4.3. Úspěšnost uvedené heuristiky na pořadí zkoušených slov silně závisí na definici ceny a tedy na parametru COST_DIFF. Ukazuje se, že hodnota COST_DIFF = 0 nebyla volena vhodně a lepší výsledky dosahují konfigurace s vyšší hodnotou parametru. To nejspíše zásadně ovlivní úspěšnost heuristiky a z tohoto důvodu budeme dále pokračovat s konfigurací ORDER = CL, která je logicky opodstatněnější. Mohli bychom na vývojových datech trénovat různé kombinace parametrů, ale to již přesahuje rámec této práce.

Na základě uvedených výsledků[1] navrhujeme následující konfiguraci parametrů dešifrátoru jako finální:

```
FINAL:
    N_MAX                 =  2
    COST_DIFF             =  250
    PHASE                 =  12
    ALG                   =  DFS
    MAX_LEN_GEN_VARIANTS  =  8
    ORDER                 =  CL
    MAX_TIME              =  300
    GEOGRAPHY             =  N
```

6.4 Vyhodnocení na evaluačních datech

Na závěr otestujeme výslednou konfiguraci dešifrátoru na evaluačních datech. Jelikož jsme je dosud nikdy nepoužili, nehrozí na nich *„přetrénování“* parametrů a jedná se proto o naprosto relevantní výsledky, jaké můžeme očekávat u jakékoliv nové šifry.

Tabulka 6.3 shrnuje úspěšnost finální konfigurace dešifrátoru na evaluačních datech pro všechny míry správnosti kromě manuální evaluace. Odhadnout řešení u všech 300 šifer podle výstupu dešifrátoru by vyžadovalo příliš mnoho času a úsilí pomocníků.

[1] Nezohledňujeme parametr GEOGRAPHY, protože ho lze testovat pouze na menší části dat a vyhodnotíme obě jeho varianty i na evaluačních datech.

Konfigurace	$\mathcal{Q}_{bin}$	$\mathcal{Q}_{sol}$	$\mathcal{Q}_{conv}$
`FINAL`	16,1 %	44,8 %	41,0 %

Tabulka 6.3: Závěrečné vyhodnocení úspěšnosti dešifrátoru na evaluačních datech

Konfigurace	$\mathcal{Q}_{bin}$	$\mathcal{Q}_{sol}$	$\mathcal{Q}_{conv}$	$\mathcal{Q}_{real}$
`GEOGRAPHY = Y` (konf. `FINAL`)	13,3 %	46,2 %	42,6 %	13,3%; 23,3% ($\star$)
`FINAL`	13,3 %	49,2 %	44,9 %	

($\star$) Vyhodnoceno díky pomoci dvou lidí. První nemá s šifrovacími hrami téměř žádnou zkušenost, zatímco druhý je zkušený hráč. Celkem tento vzorek obsahuje pouze 30 šifer. První pomocník by hledal další stanoviště správně ve čtyřech případech a druhý v sedmi.

Tabulka 6.4: Vyhodnocení přínosu zeměpisných souřadnic na menší části dat

Tabulka 6.4 srovnává úspěšnost dešifrátoru `FINAL` při použití a bez použití zeměpisných souřadnic předcházející šifry. Geografické umístění známe pouze u 30-ti šifer z této skupiny, a proto bylo možné použít také manuální evaluaci. Dva nezávislí pomocníci se pokusili pro každý výstup dešifrátoru odhadnout, kde se bude nacházet následující stanoviště. Vzorek dat je to velmi malý a velmi náchylný na drobné výkyvy.

6.5 Zhodnocení výsledků

Podařilo se nám navrhnout aplikaci, která poměrně spolehlivě vyřeší 15 % všech substitučních šifer ze šifrovacích her. Některé šifry navíc dešifrátor téměř vyřeší a zmýlí se pouze v několika málo písmenech.

Nejúspěšnější je program na šifrách, jejichž řešení se skládá zhruba se dvou nebo tří slov. U delších šifer trvá našemu algoritmu výpočet obvykle příliš dlouho. Takové dlouhé šifry nejsou hlavním předmětem našeho zájmu, neboť na ně lze účinně aplikovat frekvenční analýzu, a proto se organizátoři zadávání takových substitučních šifer obvykle vyhýbají.

Naopak velmi krátké šifry, které obsahují pouze několik písmen a jejich řešení je jednoslovné, nesou málo informace a správně vypadajících řešení je příliš mnoho k tomu, aby bylo možné to zamýšlené uhodnout.

Zatímco na trénovacích a vývojových datech se přidání zeměpisných souřadnic ukázalo přínosné, na evaluačních datech nepřineslo žádné zlepšení. Přisuzujeme to malému množství testovacích dat. Zajímavé je, že bez použití zeměpisných souřadnic rozluštil dešifrátor čtyři šifry správně a s použitím dvě z těch správných řešení vůbec nenašel. Správné řešení v jedné ze zmíněných šifer bylo

```
HLEDEJTE U KOSTELA V LAZNICH
```

a dešifrátor, který neznal aktuální polohu, využil při hledání řešení názvů ulic `U KOSTELA` a `V LAZNICH`. První z ulic se ale nenachází poblíž umístění aktuální šifry, a proto ji dešifrátor `GEOGRAPHY = Y` nerozluštil. Podobným způsobem může dešifrátor občas dosáhnout lepší úspěšnosti, pokud nepoužije informaci o aktuální poloze.

Na evaluačních datech byla úspěšnost dešifrátru mírně menší než na trénovacích a vývojových datech. Takový výsledek je naprosto očekávaný, protože para-

metry dešifrátoru jsou voleny tak, aby luštil co nejlépe šifry z vývojových dat. Z výsledků manuální evaluace nevyplývá přesná hodnota úspěšnosti dešifrátoru (ta se hrubým odhadem bude pohybovat kolem 18 %), ale přesto přináší některé zajímavé informace. Například u jedné ze šifer, kterou dešifrátor rozluštil a správné řešení vypsal mezi prvními deseti, ani jeden z pomocníků nerozpoznal z výstupu polohu dalšího stanoviště. Na druhou stranu jeden z pomocníků odhadl správné řešení ze všech ostatních výstupů, které nebyly úplně správně, ale jejich relativní shoda řešení přesahovala 75 %. Taková řešení se liší pouze v několika málo písmenech a dají se obvykle poměrně snadno lidskou intuicí rozeznat. Manuální evaluaci považujeme za velmi důležitou, protože nejlépe odpovídá reálné situaci.

Kapitola 7

Závěr

Navržením a implementací programu se nám podařilo ukázat, že v nezanedbatelné části substitučních šifer používaných na šifrovacích hrách lze řešení poměrně přesně uhodnout. Na druhou stranu stále většina takových šifer zůstává odolných i při použití velké výpočetní síly a jazykového modelu, který se na ně přímo zaměřuje.

Kdyby se podařilo implementovat dešifrátor jako mobilní aplikaci pro chytré telefony, mohli bychom využít GPS navigaci (která je dnes již poměrně běžnou součástí těchto telefonů) pro určení aktuální polohy a dosáhnout přesnějších výsledků.

Zcela zásadní problém je ovšem v návrhu algoritmu, který nyní očekává velké množství výpočetní kapacity a dostupné operační paměti. Pravděpodobně by tak v neúnosné míře vzrostl potřebný čas k nalezení alespoň nějakého rozumného řešení. Dalším důležitým faktorem je poměrně malá kapacita baterií takových telefonů, které by pro aplikaci byly vhodné. Nejspíše by došla baterie mnohem dříve než bychom použitím programu nalezli řešení. Určitě by ale bylo možné takovou aplikaci navrhnout a přizpůsobit speciálně potřebám mobilních zařízení.

Druhou možností jak zpřístupnit dešifrátor účastníkům her je vytvořit webovou aplikaci snadno dostupnou přes telefon s připojením k internetu. Velmi jednoduchou verzi této služby můžete najít na stránce

`http://atrey.karlin.mff.cuni.cz/~filip/decipher/`.

Povedlo se nám analyzovat jazyk šifrovacích her a navrhnout účinný model, který ho popisuje. Dalším výrazným počinem bylo navržení algoritmu, který poměrně efektivně prohledává stavový prostor možných řešení zadané šifry. Obzvláště zajímavá je heuristika na pořadí zkoušených slov, jejíž účinnost je opravdu překvapivá.

Kromě jazykového modelu se nám podařilo také úspěšně využít toho, že šifrovací hry obvykle probíhají v terénu a výsledky šifer odkazují na jiná místa poblíž. Při znalosti aktuálních zeměpisných souřadnic může uživatel aplikace dosáhnout mnohem přesnějších výsledků.

Produkt této práce rozhodně není dokonalý a vytvořený program by bylo možné dále vylepšovat, ale základní cíle byly naplněny.

Seznam použité literatury

Katz, S. M. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, pages 400–401.

Manning, C. a Schütze, H. (1991). *Foundations of Statistical Natural Language Processing*. MIT Press.

Nuhn, M. a Ney, H. (2013). Decipherment complexity in 1:1 substitution ciphers. proceedings of the 51st annual meeting of the association for computational linguistics. *Sofia, Bulgaria, Association for Computational Linguistics*, pages 615–621.

Pelánek, R. (2008). *Almanach TMOU*. Vydání první. Instuktoři Brno.

Singh, S. (2003). *Kniha kódů a šifer*. DOKOŘÁN. ISBN 80-86569-18-7.

Žeravík, J. (2012). Klasické kryptografické metody.

Seznam obrázků

Seznam tabulek