Mendelova univerzita v Brně
Provozně ekonomická fakulta

Dávkový automatický korektor pravopisu slov rozsáhlých textových souborů

Diplomová práce

Vedoucí práce:
doc. Ing. Jan Žižka, CSc.

Bc. Martin Péchal

Brno 2014

Rád bych poděkoval svému vedoucímu práce doc. Ing. Janu Žižkovi, CSc., za odborné vedení, podněty a připomínky, které mi poskytl v průběhu zpracování této práce. Dále bych chtěl poděkovat mé rodině, zejména mé matce, za podporu v průběhu mého studia.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Dávkový automatický korektor pravopisu rozsáhlých textových souborů** vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom/a, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 20. května 2014 _____

Abstract

Péchal, M., Fully automatic spell checker for large volume of text data. Master thesis. Brno 2014.

This thesis is about spell checking. The main goal is to design an automatic spell checker which can find misspellings in text and correct them without any additional user input. The spell checker must be also able to correct huge amount of text data because it will be used for pre-processing of text before some data mining methods will be applied.

Keywords

Spellcheck, text correction, data mining, vocabulary

Abstrakt

Péchal, M., Dávkový automatický korektor pravopisu slov rozsáhlých textových souborů. Diplomová práce. Brno, 2014.

Diplomová práce se zabývá problematikou korekce chyb v textu. V rámci práce je analyzována problematika vzniku a korekce chyb. Na základě dané analýzy je navržena aplikace, která dokáže provádět korekci textu automaticky bez toho, aby uživatel rozhodoval o tom, kterým slovem má být chyba nahrazena. Aplikace je navržena primárně pro korekci textu, předtím než z něj budou dolovány znalosti. Z toho vyplývá, že aplikace musí být schopna zpracovat velké objemy textových dat.

Klíčová slova

Korekce textu, oprava chyb, dolování znalostí, slovník

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod a cíl práce

1.1 Úvod

Dnešní doba s sebou nese značný nárůst objemu dat ve světě. Jednou z hlavních tažných sil tohoto nárůstu je internet, který dnes pokrývá velkou část populace. Jeho zlepšující se dostupnost, spolehlivost a v neposlední řadě také nárůst rychlosti přispívají k růstu počtu uživatelů internetu. Na rozšířenost internetu navazují i další technologie, jako jsou například chytré telefony apod., pomocí kterých můžeme produkovat další a další data. Je tedy zcela zřejmé, že se nárůst objemu dat ve světe jen tak nezastaví a zdá se, že se bude rychlost nárůstu spíše ještě zvyšovat (Cisco, 2013).

Objem dat už dosáhl takových rozměrů, že je často nelze zpracovávat ručně, a proto se také mění role člověka v procesu zpracovávání dat. Člověk spíše vytváří nástroje, které použije na zpracování dat a poté kontroluje výsledky. Objemy dat, které je potřeba zpracovávat dnes v některých případech dosahují dokonce takových rozměrů, že je problém je zpracovávat i automatizovaně. V poslední době se v souvislosti s touto problematikou objevuje pojem *Big data*, kterým je tato problematika označována. Tento pojem ale zachycuje širší pohled na problematiku dat a nezahrnuje pouze velké množství dat, ale i další faktory, jako například rychlost vzniku a různorodost dat (Russom, 2011, s. 6).

Jeden z typů dat, který tvoří velkou část objemu dat ve světě, jsou textová data. Textová data mají spoustu úskalí, která komplikují jejich zpracování, jako je například mnoho různých jazyků, problémy s kódováním apod. Informace v textové podobě také nemusí být úplně jasné. Pokud je například použito spojení „koupil jsem to“, tak není jasné, co bylo myšleno pod slovem „to“. Pokud se na to tedy podíváme z pohledu definice pojmu *Big data* zmíněného výše, tak je zřejmé, že textová data souvisí hlavně s faktorem různorodosti, která je zde evidentně velká.

S textovými daty souvisí problematika chyb v textu. Chyby mohou být způsobeny různými důvody, jako je například překlep, přeřeknutí při diktování či neznalost správného pravopisu slova. Snaha omezit tyto chyby vedla k tomu, že spousta textových editorů a programů obecně, přes které lze zadávat text, obsahuje součást na vyhledávání chyb, na ty poté upozorňuje uživatele. Pokročilejší implementace kontroly chyb se snaží rovnou chybu odstranit nebo alespoň nabídnout co nejlepší možnou opravu ke špatně napsanému slovu.

Situace se komplikuje v případě, že je potřeba provést korekci textu bez pomoci uživatele. Ten totiž lépe rozumí kontextu celého textu a dokáže lépe zvolit vhodné slovo. Nicméně pokud je nutné zkontrolovat a opravit větší množství textu, tak není ruční korekce vhodným přístupem (ve skutečnosti bude záviset na množství a požadované přesnosti). Příkladem je situace, kdy bude nutné pomocí nějaké metody dolování znalostí z dat získávat nějaké informace z většího množství textových dat. V takových případech je velikost zdrojových dat již tak rozsáhlá, že korekci v podstatě ani ručně nelze provést. Proto by bylo vhodné mít k dispozici ně-

jaký nástroj, který by dokázal korekci provést automaticky, i když třeba nebude výstup tak kvalitní, jako kdyby prováděl korekci uživatel.

1.2 Cíl práce

Cílem práce je navržení a následná implementace softwarového nástroje, který bude provádět korekci textu automaticky bez nutného významnějšího zásahu uživatele. Textové vstupy jsou předpokládány v prostém textovém formátu. Primární účel, pro který by měla být aplikace optimalizována, je korekce textu před procesem dolování znalostí. Důvodem je skutečnost, že díky opravám chyb klesne počet unikátních slov, neboli z pohledu dolování dat klesne počet dimenzí a tím pádem dojde i ke snížení výpočetní složitosti následného dolovacího procesu. Z daného zaměření vyplývá, že aplikace bude muset být schopna pracovat s velkými objemy textových dat (v řádech stovek MB, GB i více).

Dílčí částí práce bude příprava slovníků a jejich součástí, pomocí kterých bude prováděna korekce nalezených chyb. Slovníky by měly být pokud možno co nejvíce nezávislé na aplikaci. Díky tomu bude ponechána možnost jednoduchého rozšíření slovníků v budoucnu.

Na závěr bude provedeno zhodnocení dosažených výsledků na reálných datech z hlediska přesnosti korekce a následně i z hlediska paměťové a časové náročnosti. Jako reálná data budou použity recenze hotelů z webových stránek sloužících k rezervaci pokojů v daných hotelech. Mimo dané výsledky budou diskutovány i získané poznatky z dané problematiky.

2 Literární přehled

S chybami v textu se každý setkává v podstatě denně. Ačkoliv to nemusí být na první pohled zřejmé, tak chyby mohou mít značný vliv na náš život. Příkladem může být situace, kdy člověk udělá chybu ve svém životopise. Tato chyba může poté vést až k tomu, že nebude pozván na pohovor. Je tedy jasné, že důvody proč se této problematice podrobněji věnovat jsou nezpochybnitelné. Dané problematice se věnuje spousta autorů, přičemž základní přístupy byly navrženy již před půl stoletím. Jedním z prvních autorů, který se zabýval řešením chyb, byl Damerau (1964). Na jeho práci dále navázali například Kernighan et al. (1990) a postupně další autoři. V posledních letech vývoj není natolik rychlý, nicméně inovace jednotlivých přístupů dále probíhají. V rámci literárního přehledu bude tedy shrnut vývoj a aktuální stav.

Ještě předtím než bude problematika podrobněji rozebrána, je nutné zmínit, že korekci textu komplikuje celá řada skutečností. Jednou z hlavních je existence velkého množství jazyků. Každý jazyk totiž obsahuje různá specifika. Informace v literárním přehledu budou tedy vysvětlovány zejména v kontextu anglického jazyka, protože velká část vývoje dané problematiky byla definována právě na příkladech v anglickém jazyce. Zde je nutné zmínit, že tato skutečnost se nese ještě trochu dále. Spousta jazyků díky tomu nemá korekci textu vyřešenou na takové úrovni jako anglický jazyk (ačkoliv spoustu přístupů lze implementovat nezávisle na konkrétním jazyku).

2.1 Typy chyb

Chyby ve slovech lze rozdělit na dva typy, prvním typem jsou situace, kdy díky překlepu vznikne slovo, které vůbec neexistuje (Kukich, 1992). Příkladem takové chyby je situace, kdy je chybně napsáno „oběv“ a ve skutečnosti mělo být napsáno „objev“. Ve slově bylo mylně prohozeno „je“ za „ě“, přičemž tento konkrétní překlep je poměrně pravděpodobný vzhledem k tomu, že znaky ve kterých je chyba, znějí stejně.

Druhým typem chyby je situace, kdy je použito jiné slovo, které v daném jazyce normálně existuje. Tato situace nastává například při přeslechnutí diktovaného slova, jelikož se některá slova vyslovují podobně, ale jinak se píší a mají jiný význam. Situace s přeslechnutím je problémem zejména v anglickém jazyce. V českém jazyce dojde pravděpodobněji k tomuto typu chyby stejným způsobem jako v případě prvního typu a to překlepem.

2.1.1 Výskyt chyb

Ačkoliv by se to nemuselo na první pohled zdát, tak se chyby v textu vyskytují v poměrně velké míře, jak uvádí například Wang et al. (2003, s. 754). Při zadávání dotazů do vyhledávačů v anglickém jazyce dosahuje dokonce míra překlepů 26 %. Takto velké procento nicméně může být způsobeno tím, že lidé již počítají s tím, že

je opraví funkce „měli jste na mysli“ a neopravují tedy chyby i v případě, že si jich všimnou. U delšího textu psaného běžně na počítači bude chybovost značně nižší oproti příkladu uvedenému výše. Tuto problematiku dále podrobněji rozebírá Kukich (1992). V dané práci jsou zmíněny informace z různých zdrojů, jako jsou další vědecké práce a studie. Z daných informací vyplývá, že chybovost se v textu v anglickém jazyka pohybuje přibližně v rozsahu 1-3 %. Zde je nutné zmínit, že studie, které autor zmiňuje, byly prováděny nad relativně kvalitním zdrojem textu (vědecké článku, eseje studentů apod.). V praxi bude tedy velice záležet na konkrétním původu textu a lze předpokládat, že částečně i na konkrétním jazyce.

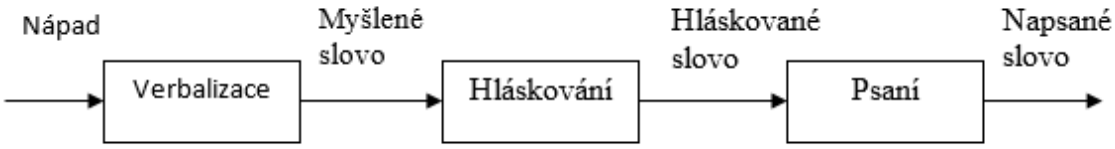
Na výskyt chyb se lze podívat i hlediska dvou typů chyb definovaných v předchozí kapitole. Kukich (1992) v této souvislosti uvádí opět různé práce, dle kterých se pohybuje rozmezí výskytu chyb ve slovech, která nejsou součástí jazyka (slovníku) okolo 60-70 %. To znamená, že 30-40 % chyb v textech jsou chyby ve slovech, která jsou běžnou součástí jazyka, ale byly pouze špatně použity. Zmíněné informace jsou opět zmiňovány v kontextu s anglickým jazykem.

Třetím pohledem na výskyt chyb jsou chyby, které vzniknou při získávání textu z obrazových dat. Kukich (1992) v této souvislosti zmiňuje, že i když by tyto techniky dokázaly rozpoznat znak s přesností 99 %, tak lze vypočítat, že bude jeden znak ze sta špatně, což při průměrné délce slova pět znaků (pro anglický jazyk) znamená, že každé dvacáté slovo bude špatně. Přesnost detekce slov bude tedy pouze 95 %.

V souvislosti s výskytem chyb je nutné si dále uvědomit, že velké množství textu může být v dnešní době již nějakým způsobem zkontrolováno předtím, než se dostanou do našich rukou. Lidé totiž dnes mohou využít značné množství nástrojů, které již korekci textu mají integrovanou.

2.1.2 Vznik chyb

Předtím, než bude rozebrána samotná detekce a korekce chyb, je dobré mít přehled o tom, jak vlastně typy chyb popsané v předchozí kapitole vznikají. Tuto znalost lze totiž později využít v návrhu procesu korekce chyb. Modely, které korekci řeší, totiž mohou být díky těmto znalostem přizpůsobeny konkrétní aplikaci (původu textových dat). Danou problematiku dobře ilustruje Obr. 1 z (Deorowicz a Ciura, 2005).



Obr. 1 Proces tvorby slova

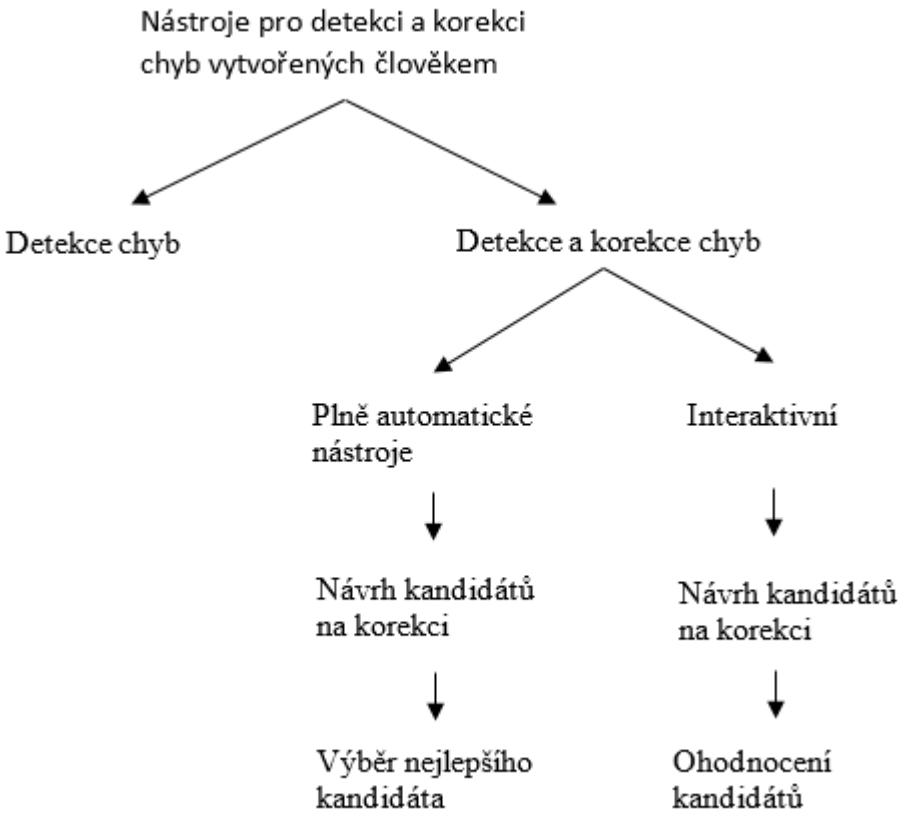
Na daném obrázku lze dobře ilustrovat, kdy nejčastěji vznikají jednotlivé typy chyb. Chyby druhého typu, kdy je špatné slovo normálně ve slovníku, vzniká nejčastěji v subprocesu verbalizace či hláskování. U verbalizace tento problém často pramení z neznalosti jazyka a nevědomosti toho, jak má slovo vypadat, když je použito v jiné části řeči či kontextu. Stejný typ chyby může vzniknout při hláskování slova, pokud má slovo podobnou výslovnost jako jiné, či se jedná o jinou podobně znějící variantu stejného slova. Příkladem může být slovo „mně“, kdy v závislosti na konkrétní situaci (pádu) může být správné hláskování „mě“ a v jiné „mně“. Chyby prvního typu, kdy slovo neexistuje ve slovníku, nastávají zejména v poslední části procesu, kdy dojde například k překlepnutí při psaní na klávesnici.

Kukich (1992) uvádí, že lze chyby v podstatě rozdělit na dvě třídy, kognitivní chyby a typografické. Pokud danou definici vztáhneme k Obr. 1, tak lze říci, že kognitivní chyby nastávají v prvních dvou částech procesu a typografické v poslední části (při psaní na klávesnici).

Proces na Obr. 1 zaznamenává tvorbu textu z pohledu člověka. Text ale může vznikat i odlišnými způsoby. Velmi podobnou cestou, jako je popsána výše je situace, kdy jeden člověk diktuje text druhému. V tomto případě, je prostor pro chybu větší, protože nad slovem postupně přemýšlí dva lidé. Jinak řečeno, proces vedoucí k napsanému slovu je delší a tím pádem je také větší prostor pro chyby. Za další situaci, která je ale značně odlišná od předchozích, je vznik chyby při převodu hlasu či obrazu do textu. V těchto případech chyby vznikají často kvůli nedokonalosti daných technologií (Deorowicz a Ciura, 2005). Konkrétní důvodem vzniku chyb je tedy nepřesnost daných technologií.

2.2 Typy přístupů

Předtím, než budou podrobněji rozebrány jednotlivé části procesu korekce chyb, je nutné zmínit, jaké typy korektorů textu existují. Implementace některých subprocesů korekce je totiž závislá na tom, jaký typ aplikace je vyvíjen a k čemu bude aplikace sloužit. Základní dva procesy, které definoval Kukich (1992) jsou detekce a korekce chyb. Některé softwarové nástroje korekci neřeší a nechávají ji pouze na uživateli. Situaci na tomto poli dále dobře ilustruje diagram níže (Verberne, 2002).



Obr. 2 Typy přístupů ke korekci textu

Z diagramu je zřejmé, že spousta přístupů implementuje pouze detekci chyb. Většinou se jedná o aplikace, které se touto problematikou přímo nezabývají, ale jsou v nich tvořena nějaká textová data (např. nástroje na programování), proto je implementována alespoň základní kontrola, zda jsou slova korektně napsaná. Tento typ lze považovat za interaktivní korekci až v momentě, kdy budou nabízeni kandidáti pro opravu. Nejsofistikovanější přístup vyžaduje automatická korekce, kde na rozdíl od interaktivní korekce musí být způsob hodnocení jednotlivých kandidátů podstatně přesnější, protože je automaticky volen nejlepší kandidát. U tohoto typu se tedy nabízí využití okolních slov, kterými je chyba obklopena (kontext). Daná problematika bude podrobněji rozebírána později.

Na typy nástrojů se lze podívat ještě z pohledu typů chyb. Většina nástrojů totiž řeší pouze chyby u slov, která nenajde ve slovníku. Důvodem je skutečnost, že pro detekci chyb, kdy je použito jiné slovo, ale je ve slovníku, je nutné použít velmi sofistikovaný přístup, který má hodně společného se samotnou automatickou korekcí textu. Daný přístup bude nastíněn později.

2.2.1 Implementace přístupů

V současné době implementuje korekci textu alespoň nějakým ze zmíněných přístupů celá řada aplikací. Korekce textu tedy není již implementována pouze v textových editorech (Word, OpenOffice), ale i v dalších typech aplikací. Jako jsou například webové prohlížeče, vývojová prostředí (tzv. IDE) či dokonce samotné webové stránky. V implementaci těchto přístupů je nutné zmínit, že velká část aplikací, co korekci řeší, využívá různé externí knihovny, které jsou připravené k řešení detekce nebo korekce. Příkladem je například GNU Aspell, starší Ispell či Hunspell (knihovna využívána v rámci OpenOffice, LibreOffice či v prohlížeči Mozilla Firefox).

Implementace korekce textu se vyskytují i na mobilních zařízeních jako jsou chytré telefony a tablety. Na těchto zařízeních dává implementace korekce velký smysl, protože vzhledem k typům a velikostem jejich klávesnic je pravděpodobnost vzniku chyb větší než na počítači. V souvislosti s mobilním zařízeními se lze vrátit zpětně ke kapitole, která se zabývala vznikem chyb. Malé rozměry zde komplikují psaní a tím pádem zvětšují pravděpodobnost vzniku chyby v poslední části procesu tvorby textu, která byla popsána na Obr. 1.

2.3 Detekce chyb

Jak již bylo naznačeno v předchozí části, tak korekci textu lze rozdělit na více částí. Nyní bude rozebírána první část a to detekce chyb. Detekce chyb spočívá v tom, že se postupně řetězce (slova) na vstupu vyhledávají ve slovníku (Kukich, 1992). Aby bylo možné identifikovat slova, která jsou napsána chybně, tak je nutné využít slovník slov daného jazyka. Tento slovník by měl být pokud možno co největší, aby nebyla označována slova, která jsou dobře, jako špatná. To by totiž mohlo vést naopak k tomu, že by byl počet chyb zvýšen. Daná slova by totiž byla nahrazena za jiná, která sice existují ve slovníku, nicméně nehodí se do aktuálního kontextu. Značnou nevýhodou tohoto přístupu je skutečnost, že je nutné udržovat slovník pokud možno co nejvíce aktuální a průběžně jej rozšiřovat o nově vznikající slova (Whitelaw et al., 2009).

I přesto, že se jedná o jednodušší část problematiky, zde mohou nastat problémy. Jedním z nich je právě velikost slovníku, kdy v případě velmi rozsáhlého slovníku může být výpočetně složitější vyhledávat požadovaná slova, nicméně díky výkonu současného HW a možnosti přizpůsobení struktury pro ukládání slovníku není tato část výraznějším problémem.

Ještě než budou rozebrány další potenciální problémy prvního přístupu, je nutné zmínit druhý možný přístup, který pracuje na odlišném principu s využitím n-gramů. Za n-gramy v tomto přístupu[1] považujeme sekvenci písmen o různých délkách. Unigram je jednopísmenná sekvence, bigram je sekvence dvou písmen

[1] Označení n-gramy se používá i v souvislostí s výskytem slovních spojení (např. dvě slova – bigram atd.)

a trigram třech. V různých jazycích se jednotlivé sekvence vyskytují s různou frekvencí, což lze využít při analýze chyb. Pokud se ve slově vyskytuje nějaký n-gram, který má malou frekvenci výskytu, tak dané slovo může být chyba. Výhodou tohoto řešení je, že není potřeba slovník, nicméně příprava statistických informací o n-gramech je naopak relativně náročná. V praxi se tedy pro korekci textu používá spíše první zmiňovaný způsob s vyhledáváním ve slovníku (Kukich, 1992).

Horší situace ale nastává v případě, že text na vstupu je ve více jazycích. Pokud by byl použit pouze jeden slovník, tak budou daná slova z jiného jazyka označena jako špatná. Daný problém by šel teoreticky řešit použitím více slovníků, nicméně to přináší spoustu úskalí. Jedním z nich je například situace, kdy je slovo napsáno špatně, ale shodou okolností se nachází ve slovníku jiného jazyka, takže je označeno jako dobře napsané. Proto by bylo nutné implementovat funkčnost, která na základě okolí aktuálního slova rozpozná jazyk a použije podle toho daný slovník. V praxi ale není funkčnost rozpoznávání jazyka implementována, protože tento proces je poměrně komplikovaný. Existuje několik metod, které lze použít pro rozpoznávání jazyka, přičemž dané metody jsou postaveny na pravděpodobnostech výskytu n-gramů, markových modelech a dalších přístupech (Řehůřek, Kolkus, 2009). Funkčnost rozpoznávání jazyků je často suplována jinými přístupy, jako je přepínání slovníku na základě toho jaká klávesnice (jazyk) je aktuálně zvolena (MS Word).

Dalším ještě komplikovanějším problémem je detekce chyb ve slovech, která jsou napsána správným pravopisem, ale do věty ve skutečnosti nepatří. Jedná se tedy o druhý typ chyby popsané výše. V tomto případě nelze použít detekci na základě vyhledávání slova ve slovníků, ale je nutné přistoupit k mnohem sofistikovanějšímu přístupu, který vychází z přístupů ke korekci chyb. Z toho vyplývá, že detekce tohoto typu chyb není většinou implementována v případě, kdy je řešena pouze detekce chyb a uživatelé si je poté opravují sami. Problematika oprav tohoto typu chyb bude rozebírána později.

Jedním z důvodů proč různé aplikace, které nějakým způsobem řeší kontrolu či korekci textu stejným přístupem, je skutečnost, že dané aplikace používají stejné externí knihovny viz kapitola 2.2.1. Ty často řeší pouze některé části problematiky (například pouze detekci). Důvodem je zejména to, že pokročilejší funkce jsou značně náročné na implementaci.

2.4 Korekce chyb

Většina současných aplikací pro korekci je postavena pouze na detekci chyb a následné opravě chyb uživatelem. Za korekci v tomto případě lze považovat pouze část, kdy se generují slova jako možní kandidáti pro nahrazení špatného slova. Tyto aplikace tedy mohou být jednodušší a mohou využívat pouze základní přístupy ke korekci. Použití pokročilejších přístupů, které budou rozebírány dále, ale není vyloučeno.

Situace se komplikuje v případě, kdy uživatel nemůže do procesu korekce zasahovat ať už z jakéhokoliv důvodu (množství textu atd.). Zcela automatickou

korekci lze provádět s velkou přesností a automaticky víceméně pouze v situacích, kdy je korektor přizpůsoben nějakému konkrétnímu použití, kde je počet možných slov omezený (slovní zásoba). Zástupcem této kategorie je například korekce příkazů pro nějaký program. Dalším podstatně složitějším příkladem je situace, kdy se má korekce provádět automaticky nad velkým objemem dat, který už nelze zpracovávat ručně. Této části problematiky se primárně věnuje tato práce. Dalším typickým příkladem, kdy nelze provádět korekci ručně je převod mluveného slova do textu. V rámci daného procesu se používá korekce textu, aby byla zajištěna co nejpřesnější detekce toho, co člověk vyslovil (Kukich, 1992). Automatická korekce bývá také často součásti OCR procesu.

Dalším aspektem, který ovlivňuje přístup k opravě chyb je skutečnost, zda jsou opravována jednotlivá slova nebo zda je k dispozici i kontext (předchozí a následující slova po chybném slově). V případě, že by bylo nutno provádět i kontrolu slov, která jsou ve slovníku, ale byla pouze zaměněna za jiné slovo, tak je okolní kontext nutností. Jinak by nebyla k dispozici žádná informace, na základě které by se dali vygenerovat kandidáti na korekci. Kontext také velice napomáhá s volbou opravy v případě, kdy ke slovům, která nejsou ve slovníku, vygenerujeme víc než jednoho kandidáta.

Korekce se skládá z několika navazujících procesů. Nejprve se vytvoří seznam kandidátů, poté se daní kandidáti nějakým způsobem ohodnotí, vybere se nejvhodnější kandidát a nakonec se rozhodne, zda se oprava provede. Zde lze ilustrovat rozdíl mezi interaktivní korekcí a automatickými řešeními – proces generování kandidátů je víceméně stejný, hodnocení kandidátů ale nemusí být u interaktivní korekce natolik přesné, protože ještě prochází kontrolou od uživatele. Pokud je ale k dispozici řešení, které se používá pro automatickou korekci, tak je samozřejmě vhodné použít je pro hodnocení (řazení kandidátů) i u interaktivní korekce (pokud nejsme omezeni třeba výpočetním výkonem).

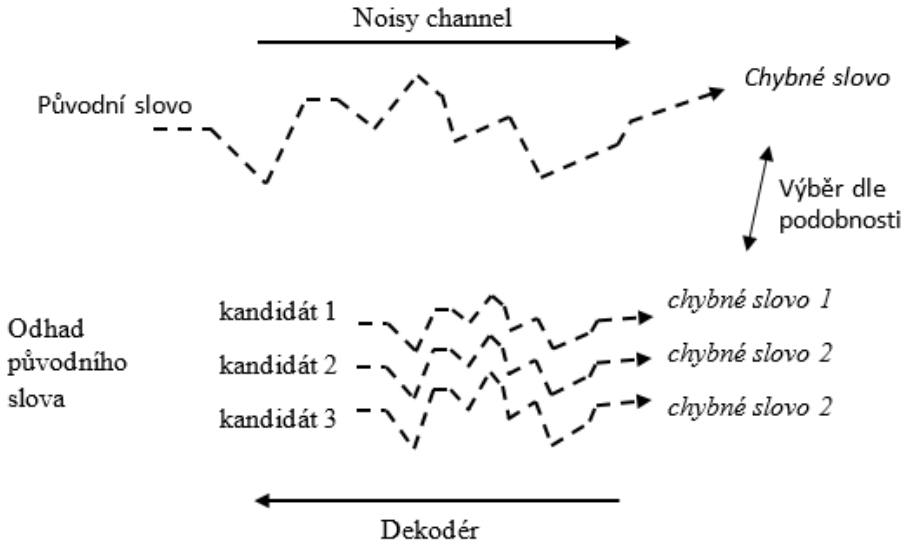
2.4.1 Typy korekce

Pod pojmem typ korekce se rozumí způsob, kterým je hodnocena vhodnost jednotlivých kandidátů. Korekci chyb lze rozdělit na dva základní typy (Kukich, 1992). Jedním typem je situace, kdy je prováděna korekce slova bez informace o jeho okolních slovech. Slova, která jsou vlevo a vpravo od vybraného slova (chyby) jsou označována jako kontext. Druhý typ korekce je opak prvního a okolní kontext využívá k tomu, aby byla oprava chyb co nejpřesnější. Za třetí typ korekce je někdy považována pouhá detekce chyby, nicméně z našeho pohledu se o korekci nejedná, protože cílem této práce je implementace automatického korektoru pro velké objemy textu, pouhá detekce tedy není řešením. Typy korekce tedy jdou ruku v ruce s typy aplikací, které byly popsány výše.

Pokud jsou typy korekce vztaženy k typům chyb popsaných v kapitole 2.1, tak je zřejmé, že chyby ve slovech, která jsou součásti jazyka (jsou ve slovníku) lze opravovat pouze druhým způsobem s využitím kontextu, protože není k dispozici žádná jiná informace o tom, jakým směrem by se korekce měla vydat.

2.5 Noisy channel model

Model přenosové cesty ovlivněné šumem (volně přeloženo) původně navrhl Shannon (2001) při matematické analýze komunikace. Model lze vysvětlit při pohledu z hlediska korekce textu následovně: chybné slovo je výsledkem poškození původního slova při přechodu přes komunikační kanál, který není dokonalý (obsahuje šum). Jurafsky (2009) popisuje šum jako formu substitucí a dalších změn v písmenech slova, které komplikují rozpoznání pravého slova. Autor podrobněji vysvětluje princip modelu na následujícím obrázku.



Obr. 3 Princip Noisy channel modelu

Jurafsky (2009) dále uvádí, že cílem je vytvořit model přenosové cesty. Tento model je poté využit tak, že se vezme každé slovo daného jazyka a nechá se projít daným modelem. Následně se poté vybere slovo, které je nejbližší chybnému slovu. *Noisy channel* model je postaven na pravděpodobnosti a je dán následujícím vztahem (1).

$$\hat{w} = \arg\max_{w \in V} P(w \mid x) \tag{1}$$

Znak $\hat{w}$ zastupuje slovo, která je nejlepším odhadem modelu na původní slovo. Hledáno je tedy slovo w ze slovníku V, které má maximální pravděpodobnost vzhledem ke slovu x s chybou, které je opravováno. Jelikož je problém pracovat s touto pravděpodobností, tak je dále aplikována tzv. Bayesova věta. Tuto problematiku podrobněji rozebírá (George, 1992).

$$\hat{w} = \arg\max_{w \in V} \frac{P(w \mid x)P(w)}{P(x)} \tag{2}$$

Rovnici nicméně lze dále zjednodušit tím, že je odstraněn jmenovatel. To lze provést díky tomu, že *P(x)* je konstanta. Výsledná pravděpodobnost bude tedy maximalizována jak v případě předchozí rovnice, tak i v případě následující zjednodušené verze (3).

$$\hat{w} = \arg\max_{w \in V} P(w \mid x)P(w) \tag{3}$$

Díky tomuto zjednodušení lze říci, že výsledná pravděpodobnost se rovná součinu *P(x|w),* což je tzv. chybový model a *P(w)* neboli apriorní model (Kernighan et al., 1991). V literatuře se dále objevují i jiná označení pro dané modely. Například Jurafsky (2009) užívá pro chybový model označení *channel model* a pro apriorní model uvádí název *language model* (jazykový model). V rámci této práce budou dále používány výrazy chybový model a jazykový model.

Je důležité si uvědomit, že pravděpodobnosti, které vyjdou z chybového modelu, a z jazykového modelu nelze přímo srovnávat, protože vlastně vypovídají o jiné skutečnosti. V praxi se tedy do vzorce doplňuje do rovnice parametr, pomocí kterého je upravována nesprávnost odhadů chybového modelu vzhledem k výskytu chyb v textu (Whitelaw et al., 2009).

$$\hat{w} = \arg\max_{w \in V} P(w \mid x)P(w)^{\lambda} \tag{4}$$

Whitelaw et al. (209) dále uvádí, že daný parametr λ lze natrénovat vzhledem k aktuálním datům a jeho hodnotu lze dále měnit vzhledem k tomu kolik kontextu je k dispozici. Pokud je například opravováno slovo na počátku či konci věty, tak lze chybovému modelu pomocí daného parametru nastavit větší váhu, protože jazykový model nemá k dispozici tolik kontextu.

2.5.1 Generování kandidátů

Prvním krokem při aplikaci přístupu, který byl definovaném výše, je generování kandidátů slov. Z nich bude později vybrán nejpravděpodobnější kandidát na základě vypočítané pravděpodobnosti. Kernighan et al. (1990) navrhl použít metodu, kterou definoval Damerau (1964). Tato metoda identifikuje čtyři základní operace: vložení znaku, nahrazení znaku, smazání znaku a prohození dvou sousedních znaků. Na základě těchto operací dále definuje tzv. *edit distance* (vzdálenost řetězců na). Pod tímto pojmem je myšlen minimální počet operací (ze zmiňovaných čtyř), který vede k tomu, že se z jednoho řetězce stane druhý.

Získané poznatky lze tedy využít při generování kandidátů, kdy se vezme slovo s chybou a pomocí možných operací definovaných výše jsou vygenerovány všechny možné varianty řetězců. To znamená, že se například vloží všechna možná písmena použité abecedy na každou pozici v textu. Dále se jednotlivá písmena

ve slově postupně vymažou, prohodí či nahradí. Takto nám vznikne množina řetězců, které jsou ve vzdálenosti jedna od původního řetězce. Z této množiny poté vybereme pouze slova, která existují ve slovníku, čímž se jednak značně sníží počet kandidátů na opravu, ale hlavně zůstanou pouze relevantní kandidáti. Následující tabulka demonstruje generování kandidátů nad anglickým slovníkem. Tento příklad byl zvolen zejména z toho důvodu, že jsou zde vidět všechny možné druhy operací, které vedou ke vzniku kandidáta na korekci ve vzdálenosti jedna od původního slova.

Tab. 1 Generování kandidátů pro korekci

Chyba	**Korekce**	**Špatný znak**	**Správný znak**	**Pozice**	**Typ operace**
acress	actress	-	t	2	mazání
acress	cress	a	-	0	vložení
acress	caress	ac	ca	0	prohození
acress	access	r	c	2	nahrazení
acress	across	e	o	3	nahrazení
acress	acres	s	-	4	vložení
acress	acres	s	-	5	vložení

Zdroj: Kernighan et al., 1990

Znak „-“ reprezentuje nulový (prázdný) znak. Vygenerovaní kandidáti jsou poté dále předáni k ohodnocení použitým modelem.

Z příkladu je zřejmé, že správné slovo bude pravděpodobně skutečně ve vzdálenosti jedné editace od slova s chybou, protože kandidáti jsou relevantní. Damerau (1964), který navrhl tuto problematiku, dále uvádí, že 80 % překlepů je ve vzdálenosti jedné operace od zamýšleného slova a téměř všechny chyby jsou do vzdálenosti dvě. Analýzou chyb se dále zabývali například Pollock a Zamora (1984), kteří uvádí, že chyby se vzdáleností jedna od správných řetězců (slov) dokonce tvoří 90-95 % chyb. Skutečnost nicméně může být rozdílná, vzhledem k tomu, že autoři prováděli analýzu nad rozdílnými daty. V praxi bude tedy záviset na tom, jaký text bude opravován.

Pokud by bylo nutné provádět korekci s co největší přesností, tak je možné použít metodiku generování kandidátů i opakovaně a získat tak i varianty, které jsou vzdáleny více. Při zvažování do jaké vzdálenosti generovat kandidáty by se mělo vzít v potaz, jaké množství textu bude zpracováváno a jaký je požadovaný výkon, protože s opakovaným spouštěním roste poměrně rychle výpočetní náročnost.

V souvislosti s generováním kandidátů či spíše se vzdáleností mezi řetězci (*edit distance*) je nutné zmínit, že existují i další přístupy založené na podobných principech a lze je tedy využít v rámci procesu korekce textu. Jedná se například o Hammingovu vzdálenost (Hamming, 1950), která je postavena na hledání minimálního počtu substitucí pro změnu jednoho slova na druhé. Danou vzdálenost lze vypočítat pouze na řetězcích stejné délky. Jako jednu z další technik lze zmínit hle-

dání nejdelší společné subsekvence, která slouží k hledání sekvencí stejných znaků (mezi nimi mohou být mezery). Danou techniku lze řešit několika různými algoritmy (Hirschberg, 1977).

2.5.2 Jazykový model

Jakmile jsou vygenerováni kandidáti na korekci, tak se přechází k jejich hodnocení. To vychází z modelu definovaného výše, který lze, jak již bylo řečeno, rozdělit na dvě části. Jednou z částí je tzv. language model, který ve zmíněné rovnici reprezentuje část *P(w)*. Tato pravděpodobnost zachycuje, jak často se dané slovo vyskytuje v daném jazyce v případě, že ignorujeme kontext slova. V případě, že se bere v úvahu kontext slova, pak pravděpodobnost reprezentuje, jak moc často se dané slovo vyskytuje v obklopení daných slov. Ohodnocení pravděpodobnosti demonstruje následující tabulka, která zachycuje ohodnocení kandidátů z Tab. 1.

Tab. 2 Ohodnocení pomocí modelu jazyka

c (kandidát)	frekvence c	P(c)
actress	1343	0.0000315
cress	0	0.000000014
caress	4	0.0000001
access	2280	0.000058
across	8436	0.00019
acres	2879	0.000065

Zdroj: Jurafsky (2009)

V tabulce je vidět, že momentálně je nejpravděpodobnější korekce slovem „across“. Je důležité zmínit, že konkrétní hodnoty vzešly z analýzy korpusu AP newswire z roku 1988, který obsahoval 44 miliónů slov. Kvůli zjednodušení nebyl v úvahu brán kontext slova (byla tedy použita pouze frekvence slov v daném korpusu). Pokud má tedy slovo frekvenci 8436, tak se přesně tolikrát vyskytovalo v daném korpusu. Výsledná pravděpodobnost se tedy podělí celkovým počtem slov daného korpusu. V praxi by se spíše použily frekvence spojení dvou, třech či více slov (tzv. n-gramy). Zde ale v rámci zjednodušení autoři ilustrují model na unigramech. N-gramy budou podrobněji popsány v jedné z následujících kapitol.

Dále si všimněme, že ačkoliv slovo „*cress*“ nemá žádný výskyt, tak jeho pravděpodobnost není rovna nule. Důvodem je skutečnost, že nemáme vždy k dispozici informaci o frekvenci každého slova (protože ta záleží na použitém korpusu). V tom případě se používají různé techniky (např. vyhlazování), které tomuto problému předcházejí. Daná problematika bude podrobněji rozebírána později.

2.5.3 Chybový model

Podstatně složitější situace je v případě evaluace druhé části modelu tzv. chybového modelu (*error model*) neboli *P(w|x)*. Tento model totiž musí nějakým způsobem modelovat aspekty vedoucí k tvorbě chyby v textu. Modelování chyb je poměrně

složité, protože to může záležet na celé řadě faktorů. Jurafsky (2009) uvádí, že mezi ně patří například rozložení klávesnice, skutečnost zda je uživatel pravák či levák a dále na spoustě dalších faktorů, mezi které patří i proces vzniku chyb popsaný v kapitole 2.1.2.

Kernighan et al. (1990) navrhl metodu, která využívá podobného přístupu, jako při generování kandidátů. U daných slov tedy sleduje jaká operace a jaký konkrétní znak byl zaměněn, smazán, vložen či přehozen. Zároveň také sleduje, jaký znak dané editaci předcházel (sleduje se tedy kontext, tentokrát ale na úrovni znaků). Na základě tohoto přístupu lze poměrně dobře vytvořit způsob k hodnocení kandidátů.

Hodnoty pro jednotlivé operace a znaky ukládal Kernighan et al. (1990) do matic záměn (*confusion matrices*). Celkem byly tedy vytvořeny čtyři matice:

- del[x,y] - reprezentuje, kolikrát byl řetězec „xy“ napsán jako „x“
- ins[x,y] - reprezentuje, kolikrát byl řetězec „x“ napsán jako „xy“
- sub[x,y] - reprezentuje, kolikrát bylo „x“ napsáno jako „y“
- trans[x,y] - reprezentuje, kolikrát bylo „xy“ napsáno jako „yx“

Jurafsky (2009) doplňuje, že tyto matice byly vytvořeny (pro operace mazání a vložení) se závislostí na předchozím znaku, nicméně lze je udělat i se závislostí na znaku následujícím. Autor ale dále nerozebírá, který z přístupů je výhodnější.

Následně bude proveden výpočet celkové pravděpodobnosti chybového modelu pomocí vzorců, které hodnoty z matic vztahují k velikosti korpusu, ze kterých byly matice generovány.

$$P(w \mid x) = \begin{cases} \dfrac{del[c_{p-1,}c_p]}{chars[c_{p-1,}c_p]} & \text{- mazání} \\ \dfrac{ins[c_{p-1,}t_p]}{chars[c_{p-1}]} & \text{- vložení znaku} \\ \dfrac{sub[t_{p,}c_p]}{chars[c_p]} & \text{- nahrazení znaku} \\ \dfrac{trn[c_p, c_{p+1}]}{chars[c_p, c_{p+1}]} & \text{- prohození znaků} \end{cases} \tag{5}$$

Následující tabulka zachycuje vypočtené pravděpodobnosti spolu s pravděpodobnostmi modelu jazyka a chybového modelu.

Tab. 3 Ohodnocení kandidátů

slovo c	frekvence(c)	p(c)	p(t|c)	p(c) p(t|c)	%
actress	1343	0.0000315	0.000117	$3.69 * 10^{-9}$	37
cress	0	0.000000014	0.00000144	$2.02 * 10^{-14}$	0
caress	4	0.0000001	0.00000164	$1.64 * 10^{-13}$	0
access	2280	0.000058	0.000000209	$1.21 * 10^{-11}$	0
across	8436	0.00019	0.0000093	$1.77 * 10^{-9}$	18
acres	2879	0.000065	0.0000321	$2.09 * 10^{-9}$	21
acres	2879	0.000065	0.0000342	$2.22 * 10^{-9}$	23

Zdroj: Jurafsky (2009)

Ačkoliv má největší pravděpodobnost, že chybné slovo bylo „*actress*“, tak se použije jako náhrada chyby slovo „*acres*“, protože je ve výsledcích dvakrát (byly totiž provedeny dvě různé operace mazání, postupně předposlední a poslední znak „s“). Z hlediska chybového modelu ale stačilo slovo „*acres*“ vyhodnotit pouze jednou, protože chybový model není závislý na kontextu.

2.5.4 Generování matic pro chybový model

Výše již bylo zmíněno, že se v rámci chybového modelu používají matice, které zachycují, jak moc často se daná operace s daným znakem provedla, aby bylo dosaženo správného slova. Jinak řečeno zachycují, jak často došlo k chybě v závislosti na chybové znaku a dle typu chyby případně i na předchozím znaku. Mimo tyto matice ještě ve výpočtu figurují frekvence jednotlivých znaků a frekvence sekvencí dvou znaků. Všechny tyto informace (matice a výskyty) jsou nutné k tomu, aby bylo možné použít tento model.

Jurafsky (2009) uvádí, že lze matice získat vygenerováním ze seznamu chyb a jejich korekcí. Tento přístup má ale nevýhodu v tom, že bude i tak nutné získat nějakým způsobem informace o frekvenci znaků a sekvencí znaků. To lze vygenerovat z nějakého korpusu v daném jazyce. Kernighan et al. (1990) nicméně původně navrhl způsob, který využívá korpus rovnou i ke generování matic. To je realizováno tak, že se matice nejprve inicializují na stejnou hodnotu a poté se spustí hledání chyb nad daným korpusem. Získané informace o chybách a korekcích se poté použijí pro aktualizaci matic a takto se spustí daná činnost opakovaně. Autor nicméně neuvádí, kolik iterací by se mělo provést. Druhé řešení má i přes to dvě výhody: nevyžaduje tolik ruční práce při přípravě seznamu známých chyb a korekcí a frekvence výskytů znaků a bigramů jsou více relevantní vzhledem k maticím.

Další ale značně komplikovanější přístup ke generování matic definují Ristad a Yianilos (1998).

2.5.5 Chyby ve slovech obsažených ve slovníku

Výše popsaný přístup ke korekci pomocí *noisy channel* modelu je popsán z pohledu chyb ve slovech, která nejsou ve slovníku (jazyce). Na problém korekce textu se ale lze podívat i z druhého pohledu a to opravy chyb, které ve slovníku jsou (druhý typ

chyb popsaný v kapitole 2.1). Je zřejmé, že v tomto případě nelze realizovat detekci klasickým přístupem s vyhledáváním ve slovníku. Jednotlivá slova ve větě tedy musí být vyhodnocována sofistikovanějším přístupem. Pro řešení tohoto problému lze opět použít více přístupů. Kukich (1992) uvádí, že k opravě těchto chyb lze použít přístupy ze dvou skupin. Jedna skupina je postavena na omezeních pro zpracování přirozeného jazyka (*NLP constrains*). Princip metod, které vychází z daného přístupu je zjednodušeně řečeno postaven na tom, že jsou definována nějaká omezení na skladbu věty a tato omezení se kontrolují. V této práci není tento přístup podrobněji rozebírán, protože výše zmíněný model je postaven na druhém, tzv. statistickém přístupu. Tyto přístupy využívají pravděpodobnosti n-gramů pro stanovení správné korekce (a v tomto případě i pro detekci).

Jurafsky (2009) uvádí, že generování kandidátů v rámci korekce všech slov funguje podobně, jako generování popsané v kapitole 2.5.1. Na rozdíl od daného přístupu je v tomto případě mezi kandidáty zahrnuto i samotné slovo, které se snažíme opravit. Toto slovo totiž může být správně (nejsme si jisti, že je v něm chyba). Mimo to se také generují kandidáti, kteří znějí podobně jako dané slovo.

Jurafsky (2009) poté uvádí, že oprava spočívá v tom, že se vypočítají pravděpodobnosti jednotlivých slov ve větě a následně se vyberou ta slova, která maximalizují celkovou pravděpodobnost. Pro výpočet pravděpodobnosti jednoho slova je nutné vzít v potaz jeho kontext. Přístup je tedy víceméně stejný jako přístup pro opravu chyb ve slovech, která nebyla nalezena ve slovníku. Jediným rozdílem je nutnost použití jazykového modelu, který využívá kontext. Pravděpodobnost výskytu jednotlivých slov sama o sobě v tomto případě tedy nestačí. Dále je nutné upravit i chybový model, ten musí totiž zachycovat i pravděpodobnost s jakou k chybě nedošlo, aby byly minimalizovány opravy správného slova na špatné slovo. Při korekci se poté vybírá věta, ve které bylo nahrazeno (opraveno) pouze jedno slovo. To přispívá k odfiltrování špatných oprav.

Z výše zmíněného postupu pro opravu chyb ve slovech, která jsou součástí slovníku, je jasné, že se jedná o značně náročný proces a to jak po stránce samotného přístupu, tak i z hlediska výpočetního času. Brill a Moore (2000) navíc uvádějí, že pro opravu pomocí daného statistického přístupu je nutné mít k dispozici velice kvalitní (rozsáhlý) zdroj informací o entitách používaných ke stanovení vhodné korekce. Pod entitami jsou zde myšleny zejména n-gramy. Pokud by byl zdroj informací nekvalitní či méně rozsáhlý, tak může tento typ korekce vést naopak k tomu, že bude počet chyb zvýšen.

2.6 Slovníky a korpusy

Pod slovníkem je v kontextu této práce myšlen seznam slov, která se vyskytují v daném jazyce včetně jejich všech možných tvarů. Je nutné zmínit, že slovník neobsahuje veškerá možná slova v daném jazyce, protože nová slova vznikají tak rychle, že nelze slovník držet stále aktualizovaný.

Korpus je zjednodušeně řečeno rozsáhlý soubor textů v elektronické podobě (Cvrček, 2013). Autor dále uvádí, že korpus zobrazuje jazykové jevy v jejich přiro-

zeném kontextu, což umožnuje provádění jazykového výzkumu. Hlavní předností korpusu je tedy schopnost vypovídat o frekvencích různých jevů.

Na internetu je dostupná celá řada korpusů, bohužel většina z nich je buď placená, nebo je jejich použití omezeno licenčními podmínkami. Příkladem je Linguistic Data Consortium[2], které skrze své stránky nabízí celou řadu korpusů (zejména v anglickém jazyce). Bohužel i zde je nutné zaplatit členský poplatek, aby bylo možné stáhnout si požadovaný korpus. V praxi je tedy nutné použít nějaký volně dostupný korpus. Tyto korpusy nicméně často nejsou tak kvalitní či rozsáhlé.

Přesnost metod pro korekci textu je značně závislá na slovnících a korpusech. Informace v nich obsažené, či v případě korpusů z nich získané, jsou využívány k realizaci korekce. Slovníky a korpusy mají v procesu korekce textu rozdílnou roli. Slovníky se primárně používají v rámci detekce chyb, kdy se kontroluje, zda jsou slova na vstupu obsažena ve slovníku. Naproti tomu korpusy se používají ke generování dodatečných informací nebo samotných slovníků. Pod generováním informací jsou myšleny informace, které jsou dále využívány v různých aplikacích v rámci zpracování přirozeného jazyka. Konkrétním příkladem jsou frekvence různých řetězců (n-gramů). Problematika n-gramů je podrobněji popsána v následující kapitole. Použití slovníků a korpusů lze dále vztáhnout k jednotlivým typům přístupů, které byly popsány dříve. Pokud je cílem aplikace provádět pouze korekci textu, tak není nutné mít k dispozici korpus (maximálně pro vygenerování slovníku), respektive není nutné generovat informace pro část korektoru.

2.6.1 N-gramy

V rámci *noisy channel* modelu definovaném výše byla využívána pravděpodobnost výskytu unigramů, což je v podstatě pravděpodobnost jednotlivých slov. Model tuto informaci využíval k tomu, aby mohl ohodnotit a zvolit nejvhodnějšího kandidáta na korekci. Pokud se tedy nějaké slovo vyskytovalo v trénovacích datech častěji, tak mělo větší pravděpodobnost výskytu. Tím pádem byl kandidát ohodnocen lépe. Ve skutečnosti se ale pouze na pravděpodobnosti výskytu daného slova nedá natolik stavět. To platí obzvlášť v případě plně automatické korekce nebo v případě, že se jedná o druhý typ chyb (špatně použitá slova, která se ale vyskytují ve slovníku). V těchto případech se jako podklad pro jazykový model používají n-gramy vyššího stupně[3].

Jurafsky (2009) uvádí, že intuice na pozadí n-gramů vychází z toho, že je potřeba vypočítat pravděpodobnost slova *w* pomocí jeho historie *h*, neboli *P(w|h).* Příkladem může být například situace, kdy je prováděn výpočet pravděpodobnosti slova „auto“. Pravděpodobnost *P(auto|nehodu způsobilo rychle jedoucí auto)* lze tedy spočítat tak, že se vezme počet výskytů řetězce „nehodu způsobilo rychle jedoucí auto“ a počet výskytů řetězce „nehodu způsobilo rychle jedoucí“ a dané počty se vydělí. Jurafsky (2009) dále zmiňuje, že danou pravděpodobnost bude pro-

[2] Stránky daného konsorcia jsou na adrese https://www.ldc.upenn.edu

[3] Jako n-gramy jsou někdy označovány i sekvence n-znaků, v tomto kontextu jsou ale myšleny slova

blém spočítat i kdyby byl jako korpus použit celý web, protože dané výskyty v konkrétní příkladu uvedeném výše nebudou natolik velké, aby dobře vypovídaly o skutečné pravděpodobnosti.

Jelikož tedy nelze přistupovat k výpočtu pravděpodobností touto cestou, tak Jurafsky (2009) dále zmiňuje, že lze použít tzv. Markovův předpoklad (*Markov assumption*), který vychází z toho, že lze předpovědět pravděpodobnost výskytu slova bez toho, aby byl zkoumán větší počet předchozí slov (Fosler-Lussier, 1998). Z tohoto předpokladu tedy vychází následující jednotky:

- bigram (pohled o jedno slovo zpět)
- trigram (pohled o dvě slova zpět)
- n-gram (pohled o n-1 slov zpět).

Základní rovnice pro výpočet pravděpodobnosti n-gramu je tedy následující (6):

$$P(w_i \mid w_1^{i-1}) \approx P(w_i \mid w_{i-N+1}^{i-1}) \tag{6}$$

Autor dále vzorec rozšiřuje pro výpočet pravděpodobnosti bigramů. Následující vzorec už je zjednodušen a neobsahuje ve jmenovateli sumu všech bigramů, které začínali slovem w_{i-1}, protože tato skutečnost je zachycena v počtu unigramů z pohledu slova w_{i-1}.

$$P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})} \tag{7}$$

Tento způsob výpočtu se nazývá *Maximum likelihood estimate* (dále MLE), protože maximalizuje pravděpodobnost výskytu n-gramu vzhledem ke zdrojovému korpusu. Daný přístup dobře ilustruje příklad na malém korpusu. Uvažujme tedy následující věty, které tvoří korpus (<s> a </s> značí začátek a konec věty, respektive kontextu).

```
<s>Moje auto je červené</s>
<s>Tvoje auto je modré</s>
<s>Vidím modré auto</s>
```

Následující výpočty demonstrují výpočet pravděpodobnosti vybraných bigramů dle definovaného korpusu (vět) výše:

$$P(je \mid auto) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})} = \frac{2}{3} \qquad P(červené \mid je) = \frac{1}{2} \qquad P(Moje \mid < s >) = \frac{1}{3}$$

Jeden z důvodů, proč nemusí být n-gramy vnímány, jako vhodná technika k využití pro zpracování textu je zdání, že nezachycují jazyk z gramatického hlediska. Ve skutečnosti ale informaci o gramatice do určité míry obsahují, což lze dobře demonstrovat na následujícím příkladu, podobnému tomu co uvádí Jurafsky (2009). Bude-li tedy uvažována např. věta „Byl jsem tam nakoupit suroviny“, tak

při výpočtu pravděpodobností jednotlivých bigramů ve větě lze dojít k těmto výsledkům.

$$P(jsem|byl) = 0.30 \qquad P(suroviny|nakoupit) = 0.0015$$

Dané pravděpodobnosti jsou dány pouze jako příklad jak by výpočet mohl dopadnout. Důležité tedy nejsou konkrétní hodnoty, nýbrž rozdíl v řádech výsledné pravděpodobnosti. Slova, která se v rámci jazyka používají často (např. jsem) totiž hrají výraznou roli v gramatice. Na hodnotách je tedy zřejmé, že bigram obsahující tato slova má řádově větší pravděpodobnost oproti slovům, která nejsou moc závislá na okolních slovech a která mohou být použita v různých kontextech.

2.6.2 Vyhlazování

V předchozí kapitole byly definovány n-gramy. Jejich použití v praxi je ale komplikovanější, než by se mohlo na první pohled zdát. Korpusy, z kterých jsou generovány, mají totiž omezenou velikost. Často tedy nastává situace, že se některé n-gramy nevyskytují v korpusu ani v jedné instanci. To nicméně neznamená, že se v rámci daného jazyka n-gramy chybějící v korpusu nevyskytují. Tento problém byl nastíněn již v kapitole 2.6.1, nyní tedy bude představen podrobněji a doplněn o možné přístupy k jeho řešení.

Situaci lze nejlépe demonstrovat příkladem, je-li tedy uvažována věta „Moje avto má růžovou barvu“ s chybou ve slově „auto“, tak lze identifikovat pravý kontext chyby s kandidátem „auto“ jako „auto má růžovou“. Jelikož se ve skutečnosti moc často nevyskytuje auto v růžové barvě, tak je možné, že se daný trigram nevyskytuje ani v korpusu na kterém je model trénován, to ale neznamená, že dané větné spojení nemůže nastat. Ve větším korpusu by se dané spojení vyskytovat mohlo. Pokud by tedy byla tato chyba detekována, tak by byla pravděpodobnost daného trigramu rovna nule. Předpokládá se výpočet pomocí metodiky MLE, která byla popsána v předchozí kapitole 2.6.1.

Aby se tomuto problému předešlo, tak se používají techniky, které provádí tzv. vyhlazování (*smoothing*). Tyto techniky přiřazují pravděpodobnost i entitám, které se v korpusu vůbec nevyskytují. Žádní kandidáti na korekci by tedy neměli být vynechání z úvahy. Korekce by tedy měla být kvalitnější. Konkrétních metod, které tento problém řeší, existuje celá řada. Zmínit lze například metody *Laplace*, *Good-Turing* či *Kneyser-Ney smoothing* (Chen a Goodman, 1996).

Za základní přístup je považována metoda *Laplace smoothing*, která je také nazývána *add-one smoothing* (přidání jedné). Jurafsky (2009) popisuje dané vyhlazování následovně. Nejprve definuje vzorec, který používá MLE a následně jej rozšíří:

$$P(w_i) = \frac{c_i}{N} \tag{8}$$

Kde c_i je počet výskytů daného slova a N je počet všech výskytů slov v rámci slovníku. Následně je do vzorce doplněno tzv. přidání jedné.

$$P_{Laplace}(w_i) = \frac{c_i + 1}{N + V} \tag{9}$$

Zde je vidět, že ke každému výskytu slova *c* byla připočten jeden výskyt. To se tedy musí zohlednit i ve jmenovateli a tak se zde přičítá *V*, což je počet unikátních slov ve slovníku (protože je jednička přičtena ke každému slovu ze slovníku). Jurafsky (2009) dále zmiňuje, že se lze na danou situaci podívat i z jiného pohledu, kdy dochází ke snižování pravděpodobnosti u slov, které mají nenulový výskyt, a naopak zvyšování pravděpodobnosti u těch, co doteď měli nulový výskyt.

Vzorce popsané výše jsou pro určeny pro unigramy, v případě použití s bigramy, je nutné vzorec dále upravit:

$$P^*_{Laplace}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i) + 1}{C(w_{i-1}) + V} \tag{10}$$

Jurafsky (2009) demonstruje využití tohoto přístupu na testovacím korpusu, který obsahuje 1446 unikátních slov a celkem 9332 vět. Autor vybral několik slov a vytvořil z nich tabulku, která obsahuje počty výskytu jednotlivých bigramů v korpusu. V převzaté tabulce níže jsou již počty všech výskytů inkrementovány o jedničku, jak definuje tento model vyhlazování.

Tab. 4 Výskyt bigramů po přidání jednoho výskytu

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Zdroj: Jurafsky (2009)

Aby bylo možné provést výpočet pravděpodobností pomocí definovaných vzorců, je nutné uvést ještě výskyty daných unigramů v testovacím korpusu.

Tab. 5 Výskyty unigramů

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Zdroj: Jurafsky (2009)

Následně mohou být vypočítány pravděpodobnosti bigramů dle vzorce (9).

Tab. 6 Vypočtené pravděpodobnosti bigramů

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Zdroj: Jurafsky (2009)

V tabulce jsou červenou barvou vyznačeny hodnoty, které nyní mají díky vyhlazování alespoň nějakou pravděpodobnost. Důležité je také zmínit, že hodnoty pravděpodobností nebyly rozloženy mezi jednotlivé bigramy, které měli předtím nulovou pravděpodobnost rovnoměrně, protože ve vzorci pro normalizaci figuruje i hodnota unigramů (jmenovatel). To je výhoda, protože takto dané pravděpodobnosti přesněji reprezentuji skutečnost.

Aby bylo možné provést zhodnocení přínosu této metodiky vyhlazování, je nutné provést přepočet na výskyty po provedení vyhlazování. Díky tomu bude možné porovnat výskyty před vyhlazováním a po vyhlazování. Pro zpětný výpočet výskytů lze použít vzorec (11), který také uvádí Jurafsky (2009).

$$c^*(w_{i-1}w_i) = \frac{[C(w_{i-1}w_i)+1]\times C(w_{n-1})}{C(w_{i-1})+V} \tag{11}$$

Následně je pomocí daného vzorce vypočítáno rozložení výskytu po vyhlazení.

Tab. 7 Zpětně rekonstruované výskyty po vyhlazení

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Zdroj: Jurafsky (2009)

Z tabulky je zřejmé, že tento typ vyhlazování má značný vliv na výskyty slov, která měli před vyhlazením značně vyšší výskyty. Počet výskytů pro bigram „want to“ se

dokonce zmenšil z 608 před vyhlazením na 238 po vyhlazení. Na základě toho lze tedy dojít k závěru, že daná metodika není moc vhodná, protože provádí poměrně velké změny u hodnot s původním vyšším výskytem. V praxi lze tento dopad mírně zmírnit tím, že se nebude přičítat jednička k jednotlivým výskytům bigramů, ale číslo o něco menší. I tak je ale vhodné spíše použít jinou metodiku řešící tento problém.

Jako alternativní metodika může být zvoleno *Good-Turing* vyhlazování. Jurafsky (2009) uvádí, že tato metoda je postavena stejně jako některé další metody na myšlence, která pro stanovení pravděpodobnosti využívá entity (n-gramy), které byly zpozorovány v trénovacích datech pouze jednou. Jinak řečeno, frekvence singletonů (entity co se vyskytují pouze jednou) se využijí pro výpočet frekvence n-gramů, které nebyly zpozorovány ani jednou. Tato metodika je již o něco komplikovanější než metodika přidání jedné, která je zmíněna výše. Z tohoto důvodu tedy nebude dále rozebírána a budou místo toho představeny alternativní metody k vyhlazování. Metodu *Good-Turing* případně podrobněji rozebírá Gale (1995). Dalším vhodným zdrojem zabývajícím se vyhlazováním a analýzou vhodnosti jednotlivých metod je práce, kterou vytvořili Chen a Goodman (1996).

2.6.3 Alternativy k vyhlazování

Vyhlazování (*smoothing*) není jedinou cestou, jak řešit problém s nulovým výskytem n-gramů v trénovacích datech (korpusu). Jurafsky (2009) uvádí, že lze použít i znalosti z nižších n-gramů. Jednou z metodik, která je na tomto přístupu postavena, je interpolace. Tato metodika bere v potaz všechny informace (o n-gramech nižšího stupně) a kombinuje je dohromady tak, aby lépe vypovídaly o skutečné pravděpodobnosti výskytu daného n-gramu. Pro interpolaci trigramů lze tedy použít následující vzorec.

$$\hat{P}(w_i \mid w_{i-1}w_{i-2}) = \lambda_1 P(w_i \mid w_{i-2}w_{i-1}) + \lambda_2 P(w_i \mid w_{i-1}) + \lambda_3 P(w_i) \qquad (12)$$

Hodnoty λ jsou váhy a určují, jak moc se lze na danou část modelu spoléhat. V případě, že by například data ohledně bigramů byla hodně kvalitní, tak je dobré mít danou váhu nastavenu tak, aby tuto část modelu posílila.

$$\sum_i \lambda_i = 1 \qquad (13)$$

Součet jednotlivých vah dle vzorce (13) ale musí být roven jedné. Tuto metodu lze dále vylepšit pomocí závislosti vah na kontextu (okolních slovech).

Další alternativní metodou je tzv. couvání (*backoff*). Princip této metody je zřejmý již z názvu. Metoda vždy používá nejvyšší dostupný n-gram a v případě, že daný n-gram není nalezen, tak se použije n-gram nižšího stupně. Konkrétní metody postavené na tomto přístupu jsou např. *Katz smoothing* (Chen a Goodman, 1996). Součástí metodik couvání je často i vyhlazování popsané v předchozí kapitole.

2.7 Alternativní přístupy ke korekci

Výše popsané řešení detekce a korekce textu pomocí *Noisy channel* modelu není jedinou cestou jak řešit daný problém. Existují další techniky, přičemž některé z nich používají odlišný způsob korekce a jiné dále rozšiřují daný model a zlepšují jeho vlastnosti a přesnost. Techniky lze dále rozdělit na techniky postavené na statistickém přístupu ke korekci (*noisy channel*), techniky postavené na znalosti jazyka a techniky pracující na základě podobnosti klíčů.

2.7.1 Techniky založené na podobnosti klíčů

Techniky založené na tomto přístupu k opravě chyb jsou postaveny na mapování řetězců do klíčů. Dvě velmi podobná slova by poté měla mít tyto klíče stejné, či velmi podobné. Pokud je tedy vypočítán klíč nějakého řetězce, tak daný klíč v podstatě ukazuje na slova, která jsou kandidáty na korekci. Výhodou této techniky je skutečnost, že se nemusí procházet celý slovník a tím pádem je daná technika velice rychlá (Kukich, 1992). Autor dále uvádí zástupce dané techniky tzv. SOUNDEX, ten byl navržen již roku 1918. Pro tvorbu klíčů byla využívána následující pravidla dle následující tabulky.

Tab. 8 Tabulka kódování pro Soundex

Znak	Kód
A,E,I,O,U,H,W,Y	0
B,F,P,V	1
C,G,J,K,Q,S,X,Z	2
D,T	3
L	4
M,N	5
R	6

Zdroj: Kukich (1992)

Na základě daných klíčů lze poté sestrojit kód nějakého řetězce, např. „the“ bude mít kód 300 a „that“ bude mít 3003. Z daných kódů je tedy zřejmé, že jsou značně podobné.

Techniky založené na tomto přístupu byly sice rozvíjeny dále (např. technika SPEEDCOP, která využívala více klíčů pro zvětšení přesnosti), nicméně v rámci literatury nejsou moc často zmiňovány. V poslední době se již techniky postavené na tomto přístupu pravděpodobně nevyvíjí z toho důvodu, že jejich výhoda v malé výpočetní náročnosti nehraje již takovou roli. Jsou tedy zastoupeny novějšími technikami, jako *noisy channel* model.

2.7.2 Vylepšení Noisy channel modelu

Noisy channel model je velice často zmiňován v různé literatuře zabývající se problematikou korekce, proto je jasné, že se spousta autorů snažila daný model dále

vylepšit. První výraznější vylepšení modelu navrhli Brill a Moore (2000). Jejich vylepšení spočívá v tom, že chybový model neuvažuje jednotlivé editace po jednom znaku, ale i editace sekvencí řetězců. V úvahu se také bere pozice chyb v rámci slov (začátek, prostředek, konec). Trénování tohoto modelu je poté realizováno na párech chyb a jejich korekcí. Autoři uvádí, že na testovacích datech dokonce dosáhli zlepšení přesnosti až o 50 %. Tento model navíc lépe řeší opravu chyb způsobenou jejich podobnou výslovností. Podobnost výslovností totiž často není skryta pouze v jednom znaku, ale ve více znacích.

Další vylepšení daného modelu navrhli Toutanova a Moore (2002). Jejich práce je zaměřena hlavně na další odstranění chyb ve slovech s podobnou výslovností. V rámci daného modelu používají stejný chybový model jako v předchozí práci, dále ho ale rozšiřují o další samostatnou část, která hledí na chyby z hlediska výslovnosti. V rámci této části je implementovaná technika, která převádí znaky na jejich výslovnost. Tato část se musí natrénovat samostatně (lze ji ale trénovat ze stejných dat). Pro trénování lze využít různé algoritmy strojového učení.

2.7.3 Využití webu pro korekci

Whitelaw et al. (2009) navrhli novou metodu korekce textu, která využívá textová data dostupná na internetu. Výhodou dané metody je skutečnost, že nevyžaduje ručně připravený slovník správných slov jazyka. Místo toho jsou slova získávána z webových stránek (autoři uvádí, že byla zpracována víc jak jedna miliarda stránek). Filtrování toho, zda jsou slova součástí jazyka, poté probíhá tak, že jsou použita pouze slova, jejichž počet výskytů překračuje stanovenou hranicí. Takto získaná slova poté tvoří slovník. Daná metodika poté dále využívá *noisy channel* model s vylepšeným chybový modelem (Brill a Moore, 2000). Na daný model dále navazuje další část, která rozhoduje, zda má být korekce provedena. Tato část je označována jako klasifikátor.

3 Návrh řešení

Aplikace bude postavena na *noisy channel* modelu, který byl představen v literárním přehledu. Bude tedy obsahovat chybový a jazykový model. Generování kandidátů bude využívat vzdálenost řetězců.

Korekce se bude zabývat pouze opravou chyb ve slovech, která nebudou nalezena ve slovníku. Důvodů, proč neprovádět kontrolu (a opravu) u slov, která jsou součástí slovníku, je několik. Prvním z nich je zaměření vyvíjené aplikace, ta má totiž sloužit k opravě velkého množství textových dat. Předpokládané množství je natolik velké, že by doba korekce takového textu byla velmi dlouhá. Dalším důvodem je skutečnost, že kontrola a případná oprava slov, která jsou ve slovníku, vyžaduje značně kvalitní a velmi rozsáhlé informace (n-gramy) pro daný jazyk (Brill a Moore, 2000). Získání těchto informací v požadovaném rozsahu je bohužel dnes stále problematické.

Kandidáti budou generováni pouze ve vzdálenosti jedné editace od původního slova. Důvodem je opět snaha minimalizovat výpočetní náročnost daného přístupu. Chybový model bude poté postaven na původním přístupu, který navrhl (Kernighan et al., 1990). Jazykový model bude využívat informace z dostupných n-gramů ve slovnících.

Celé řešení bude fungovat v režimu *case-insensitive* (velikost písmen bude ignorována). To znamená, že nebude prováděna oprava malých a velkých písmen. Důvodů je opět více, prvním z nich je výpočetní náročnost (snaha o minimalizaci slov ve slovníku a snaha o snížení počtu kandidátů). Dalším důvodem, který byl zmíněn již v souvislosti s opravou slov, která jsou ve slovníku, je skutečnost, že by bylo nutné použít kvalitnější data (slovníky, n-gramy).

Z informací výše je tedy zřejmé, že řešení obsahuje poměrně dost kompromisů. Z toho důvodu by mělo být řešení realizováno tak, aby mezi jednotlivými komponentami (např. chybovým modelem a jazykový modelem) nevznikaly přímé závislosti. Díky tomu bude možné danou aplikaci v budoucnu dále rozšířit (např. o jiný chybový model) bez zásahu do ostatních funkcionalit. V rámci řešení tohoto problému bude použit přístup *dependency injection*[4] (Fowler, 2004).

Implementace bude dále rozdělena na dvě části. Nejprve bude implementována knihovna s funkcionalitou korektoru a následně až aplikace, která bude danou knihovnu využívat. Tento způsob řešení vychází z toho, že aplikace, které řeší korekci textu, většinou využívají také nějakou knihovnu třetí strany. Díky tomu bude možné využít implementovanou knihovnu i v rámci jiných aplikací. Další výhodou tohoto přístupu je skutečnost, že kód korektoru nebude přímo provázán s aplikační logikou.

[4] Dependecy injection je zjednodušeně řečeno způsob předávání závislostí, jehož cílem je odstranění pevných vazeb v rámci kódu (mezi třídami)

3.1 Nástroje pro realizaci

Pro realizaci knihovny a poté i aplikace řešící korekci textu bude použit programovací jazyk C#. K volbě daného jazyka vedla celá řada skutečností. První z nich je skutečnost, že daný jazyk spolu s .NET frameworkem značně zjednodušuje implementaci některých funkcionalit, než kdyby byl použit například jazyk C++. Výhodou jsou tedy více abstrahované funkce pro práci s vlákny, soubory apod. (Microsoft, 2014). Aplikace by díky tomuto přístupu měla být rychleji realizovatelná. Na první pohled totiž nemusí být zřejmé, že korektor textu bude mít poměrně rozsáhlou základnu kódu. Základní funkčnost korektoru lze sice implementovat poměrně rychle (Norvig, 2007), nicméně situace se dále komplikuje při optimalizacích kvůli výkonu, zvětšování přesnosti, získávání informací pro korekci a univerzalitě směrem ke zpracovávaným jazykům. Dalším faktorem hovořícím pro volbu daného jazyka je skutečnost, že s ním autor práce má již větší zkušenosti.

V předchozí kapitole bylo zmíněno, že bude nejprve implementována knihovna a později na ní bude postavena daná aplikace. Realizace za pomocí jazyka C#, vývojového prostředí Visual Studio a platformy Windows obecně, nám nabízí další možnosti jak knihovnu využít. S danou platformou sice nelze dosáhnout takové univerzálnosti[5] jako v případě realizace pomocí jiných jazyků (např. C++), nicméně i tak lze knihovnu v budoucnu uvolnit například skrze distribuční systém NuGet. Ten je integrován přímo ve zmiňovaném vývojovém prostředí a lze prostřednictvím něho umožnit použití dané knihovny ostatním vývojářům (vyvíjejícím pro danou platformu).

3.2 Metodika vývoje aplikace

Vývoj aplikace bude realizován pomocí prototypů. Jednotlivé části celého procesu budou postupně implementovány jako jednotlivé prototypy. Pod těmito částmi je myšleno zpracování vstupu, detekce chyb, korekce chyb a uložení výstupu. Dané prototypy budou poté tvořit funkční korektor, nicméně budou mít omezenou funkčnost. Z daných prototypů se poté bude vycházet při tvorbě finálních částí. Výhodou daného přístupu by měla být skutečnost, že bude po implementaci základních prototypů zřejmé, které části budou vyžadovat nejvíce práce a kde může být nejvíce problémů.

V rámci vývoje bude použita v některých situacích technika *test driven development*, neboli vývoj řízený testy (Gunderloy, 2007). Při analýze problematiky bylo identifikováno několik funkcionalit, které generují velké množství výstupů (kombinací). Dané výstupy nejdou zpravidla kontrolovat ručně. Z těchto funkčností lze jmenovat například generování řetězců v určité vzdálenosti od původního řetězce (popsáno v kapitole 2.5.1). V této situaci totiž může v závislosti na daném slově vzniknout velké množství různých variant řetězců. Člověk tedy není schopen

[5] Na systémy postavené na unixu by bylo teoreticky možné knihovnu naportovat pomocí frameworku Mono (http://www.mono-project.com)

v takovém případě provést efektivně a spolehlivě kontrolu, zda byly vygenerovány všechny varianty správně či ne. Pro danou funkčnost by tedy měl být nejprve napsán test, který bude obsahovat nějaká testovací slova spolu se seznamem možných variant. Následně lze implementovat danou funkcionalitu. Ta může být poté považována za hotovou až v momentě, kdy bude procházet předem vytvořenými testy.

Jako další příklad, kdy bude vhodné použít daný přístup, lze uvést načítání slovníků. Slovníky se totiž mohou skládat ze dvou souborů, jeden obsahuje základy slov a druhý pravidla definující možné předpony a přípony. Daná pravidla jsou zejména u slovníku českého jazyka natolik rozmanitá, že ruční kontrola funkčnosti je téměř nemožná.

Jako poslední příklad lze uvést kontrolu zpracování vstupu. Tím je myšleno, zda jsou v textu korektně odhalena samotná slova vzhledem ke znakům v jejich okolí. Dané zpracování se navíc může lišit jazyk od jazyka.

V neposlední řadě je nutné zmínit, že dané testy lze dobře zužitkovat v kombinaci s tvorbou prototypů. Pokud je například u jedné verze prototypu identifikováno nějaké slabé místo (chyba), tak lze zařadit kontrolu daného problému do nějakého testu. Tím pádem při další iteraci (verzi prototypu) bude automaticky kontrolováno, zda nová verze problém odstranila.

4 Implementace

Následující kapitoly se budou postupně zabývat jednotlivými částmi procesu korekce chyb. Zmíněny budou i některé oblasti, které na dané části úzce navazují. Jednotlivé části budou popsány v podobném pořadí, jako ve kterém byla knihovna (aplikace) postupně implementována. Je tedy nutné zmínit, že některé kapitoly nemusí zcela zachycovat finální stav. Ten bude dokreslen až v posledních kapitolách, které doplní již zmíněné části o přístupy k optimalizaci kritických míst a o specifika, která musela být implementována kvůli kompatibilitě s různými jazyky. Důvodem, proč je přistupováno k popisu touto cestou, je snaha zachytit reálný postup tvorby aplikace.

4.1 Zpracování vstupu

První částí procesu korekce textu je zpracování vstupního textu. Na první pohled se může zdát, že tato část problematiky není natolik komplikovaná, skutečnost je nicméně odlišná, protože i zde je nutné řešit různé problémy.

Prvním problémem je skutečnost, že textové soubory mohou být uloženy v celé řadě různých kódování. Jelikož je aplikace určena pro hromadné zpracování velkého množství dat, tak je možné, že tato data budou na vstupu ve formě několika desítek, stovek, či dokonce ještě více souborů. Z tohoto důvodu je jasné, že nelze zadávat (kontrolovat) formát každého souboru ručně. Na nastavení jednoho kódování pro všechny soubory se také nelze spoléhat. Kvůli těmto skutečnostem byl tedy implementován následující postup detekce kódování. První krokem je detekce z BOM[6] hlaviček. Jelikož tyto hlavičky nejsou často k dispozici, tak následuje detekce pomocí heuristiky. Ta zjednodušeně řečeno hledá byty, které se vyskytují či nevyskytují u konkrétních kódování. Pokud ani tato metoda nepomůže, tak je použita knihovna EncodingTools (Zeumer, 2007), která zjednodušeně řečeno analyzuje soubory více do detailu a tím pádem je určení kódování spolehlivější (nicméně i výpočetně náročnější).

Aby bylo poté možné provádět korekci, je nutné získat ze vstupu posloupnost slov. Zde je zřejmý další problém, kterým jsou znaky, jež nejsou součástí slova. Jedná se o znaky, které slouží k organizaci textu, tedy tečka, čárka, vykřičník, otazník a další, které mají rozdílnou roli (interpunkce). Aby tedy mohlo být provedeno vyhledání slova ve slovníku při detekci chyb, je nutné mít slovo očištěno od těchto znaků. Mimo tyto znaky navíc dále komplikují situaci různé zkratky, čísla, časy, atd. Tato druhá problematická skupina je při korekci přeskakována, proto bylo nutné implementovat přístupy, které tuto skupinu řetězců detekují. Pro řešení těchto problémů se nabízí regulární výrazy. V rámci implementace knihovny lze uvést několik příkladů konkrétního použití regulárních výrazů.

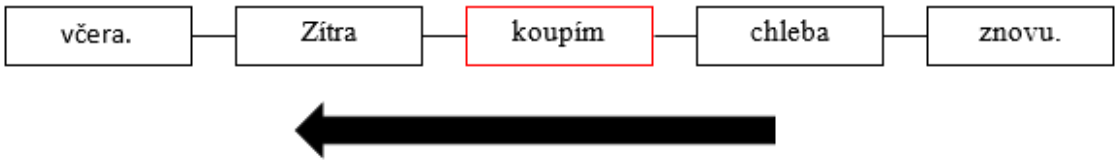
[6] Byte Order Mark – hlavičky souboru, které nesou informaci o kódování daného souboru

Tab. 9 Příklady použitých regulárních výrazů

`(\W*)([a-z-]*'*[a-z]+)(\W*)`	detekce slova a jeho okolí
`([a-z\d]+)`	detekce, zda slovo obsahuje číslo
`([A-Z]{2,})`	detekce, zda se jedná o zkratku
`([-]+)`	detekce oddělovače

První z regulárních výrazů slouží k tomu, aby ze vstupního řetězce (jedno slovo se znaky v bezprostředním okolí) vytáhl dané slovo a jeho kontext (okolí). V případě, že bude navržena nějaká korekce, tak se nahradí špatné slovo za nové a zpětně doplní okolní znaky, které byly vyextrahovány pomocí daného výrazu. Další regulární výrazy poté slouží k identifikaci, zda se má slovo přeskakovat nebo ne. Je důležité upozornit, že regulární výrazy může být nutné přizpůsobit konkrétním jazykům. Příkladem je apostrof uvedený v prvním regulárním výrazu (ten je určen pro anglický jazyk).

Dalším z problémů komplikujících zpracování vstupu je samotná velikost vstupu. Jelikož je vyvíjená aplikace určena pro zpracování velkého množství textu, tak nelze vstupní text načítat celý do paměti. I kdyby bylo zpracování prováděno postupně po souborech, tak by to mohl být problém. Jeden vstupní soubor totiž může mít velikost větší jak 500 Mb. Navíc část, která provádí korekci textu je také poměrně paměťově náročná, což v kombinaci s načítáním celých souborů do paměti může značně zvyšovat celkovou paměťovou náročnost. Z tohoto důvodu je zpracování realizováno postupným čtením vstupu. Vstup je čten po znacích, ze kterých je průběžně sestavován nepřerušovaný řetězec. Čtení po znacích je realizováno z důvodu, že řádky mohou mít velkou délku. Jakmile je na vstupu detekována mezera, tak je daný segment zpracován jako jedno slovo (s využitím zmíněných regulárních výrazů). Dané slovo je poté přidáno do fronty, ve které je uloženo posledních pět[7] takto detekovaných slov, respektive řetězců a aktuální řetězec je vyprázdněn. Při přidání dalšího slova se fronta posune a slovo, které bylo přidáno jako první, je zapomenuto.



Obr. 4 Zpracování vstupního textu pomocí fronty

Fronta je ilustrována na Obr. 4, uprostřed je vždy slovo, které je aktuálně zpracováváno (vyhledáváno ve slovníku). V případě, že je dané slovo špatně, tak se vytvo-

[7] Velikost fronty lze měnit v závislosti na tom, kolik informací potřebuje jazykový model pro korekci

ří nový objekt reprezentující danou chybu, který obsahuje i okolní slova dle stavu okna v daném momentu.

Při zpracování jednotlivých řetězců dále probíhá detekce začátku a konce věty (kontextu). To je realizováno tak, že jsou detekovány znaky: tečka, otazník a vykřičník. Informace o konci kontextu je poté ukládána do objektů ve frontě spolu s daným slovem. Informaci lze tedy později využít při korekci chyby.

4.2 Detekce chyb

Detekce chyb funguje tak, že se průběžně kontroluje střed fronty popsané v předchozím kroku. Následně se tedy vezme prostřední okno a získané slovo se převede na malá písmena. Poté se vyhledá ve slovníku. V případě, že není slovo ve slovníku nalezeno, tak je vytvořena instance třídy, které slouží pro zachycení chyb. Tato instance obsahuje mimo slova, ve kterém je chyba, i stav fronty a informace o okolních znacích původního slova (a také informace o velikosti znaků). Je nutné zmínit, že tato detekce neprobíhá v případě, kdy slovo obsahuje nějaké speciální znaky (např. dvojtečka), čísla, či bylo vyhodnoceno jako zkratka.

Proces detekce dále komplikuje skutečnost, že slovníky obsahují velký počet slov. V případě, že by se u každého slova sekvenčně procházela všechna možná slova ve slovníku a porovnávala by se, tak by byla časová náročnost algoritmu lineární. To je problém zejména z hlediska zamýšleného použití vyvíjené aplikace. Ta má totiž provádět korekci nad velkým počtem vstupů. Tento problém je tedy nutné nějakým způsobem řešit. Optimalizace vyhledávání ve slovníku bude podrobněji rozebírána později.

4.3 Korekce chyb

Korekce textu je nejsložitější částí celého procesu. Daná skutečnost je zřejmá již z literárního přehledu, kde byla dané problematice věnována největší část. Implementace bude tedy vycházet z informací zmíněných v literárním přehledu a také z definovaného návrhu aplikace.

Třída, která bude řešit samotnou korekci, bude postavena tak, aby tvořila určité pouzdro pro chybový a jazykový model. Chybový model bude rovnou obsahovat i generování kandidátů, protože informace o generování lze využít i pro výpočet pravděpodobností chybového modelu. V případě, že by byl později implementován vylepšený chybový model (Brill a Moore, 2000), tak by musel obsahovat také generování kandidátů. Možnost přizpůsobení chybového modelu je ale i tak ponechána. Výše bylo zmíněno, že zde budou popsány jednotlivé bloky procesu korekce z pohledu problematických míst. V rámci chybového modelu lze jako dané místo identifikovat skutečnost, že je nutné mít k dispozici informace, které daný model využívá ke stanovení pravděpodobností editací vedoucích k pozorovaným chybám. Tyto informace bohužel není úplně jednoduché získat. Navíc se liší v závislosti na použitém jazyce. Jelikož se tedy jedná o poměrně rozsáhlou problematiku, tak jí bude později věnována zvláštní kapitola.

Podobná situace se opakuje i u jazykového modelu. Ten využívá informace o výskytech n-gramů v rámci daného jazyka. Je tedy nutné získat informace o výskytech n-gramů, což bude opět popsáno v samostatné kapitole. Nyní tedy bude podrobněji rozebrán pouze způsob práce s n-gramy. V rámci literárního přehledu byla věnována značná část prostoru situaci, kdy není k dispozici informace o existenci požadovaného n-gramu (tedy jeho výskyt je roven nule). V rámci realizované implementace bylo použito jedno ze základních řešení daného problému. Ke každému výskytu byl přidán jeden výskyt. Důvodem pro implementaci dané metodiky byla snaha o minimalizaci výpočetní náročnosti. Pokud by byly totiž použity sofistikovanější techniky na základě interpolace či couvání, tak by bylo nutné vyhledávat postupně ve všech dostupných n-gramech. Zde se tedy nabízí otázka, jak moc velké kompromisy lze dělat za účelem dosažení vyššího výkonu. Tato otázka bude později dále diskutována.

4.4 Výstup procesu korekce

Poslední částí procesu korekce textu je samotné provedení navrhovaných oprav. Tato část se velmi podobá části, která řeší zpracování vstupu. Pro provedení oprav se totiž musí načíst původní text, provést jeho oprava a text následně opět uložit. Je tedy zřejmé, že tato část sdílí velkou část potenciálních komplikací se zpracováním vstupu. Vstup totiž opět nelze načíst do paměti celý, provést jeho opravu a uložit. Je tedy nutné načítat text postupně, následně jej opravovat a ukládat opět zpět. V rámci této operace je nutné aplikovat vhodný přístup ke hledání chyb v původním textu a k jejich nahrazení. Díky tomu, že je v rámci korekce (jazykového modelu) využívána informace o kontextu, tak může být pro jednotlivé výskyty daného slova navržena jiná korekce. To vede k tomu, že nelze použít princip najít a nahradit. Proces tedy musí pracovat i s pozicemi slov. Ty jsou poté využity ke správnému nahrazení chyby.

Dalším problémem je velikost písmen. V návrhu bylo zmíněno, že implementace nebude hledět na velikost písmen. Velikost písmen bude tedy řešena tak, že v případě, že původní chybové slovo začínalo na velké písmeno, tak i nové (opravené) slovo bude upraveno tak, aby začínalo také na velké písmeno.

4.5 Slovníky a korpusy

Aby bylo možné realizovat korekci textu, tak je nutné připravit data, která jsou potřeba pro jednotlivé části procesu. První částí je slovník, který se využívá ve dvou částech procesu. První z nich je kontrola, zda je ve slově chyba. Druhé využití je v rámci filtrování kandidátů, kdy jsou vybírána pouze slova, která jsou ve slovníku. Bohužel v dnešní době není na internetu stále k dispozici tolik slovníků, které by se daly použít. Většinu slovníků lze totiž pouze procházet přes webové rozhraní. Nejsou tedy uvolněny ke stažení. Použití některých dalších slovníků je poté naopak omezeno licenčními podmínkami. Pro český jazyk je tedy v rámci programu použit slovník, který využívá program Aspell (slovník je distribuován s licencí GNU GPL).

Pro anglický jazyk je použit slovník, který je využíván korektorem Hunspell. Daný slovník má uvedenu stejnou licenci.

Další část, která řeší korekci chyb, potřebuje pro svou funkci další typy informací. První skupinou jsou informace pro chybový model. Tyto informace se týkají frekvence výskytu znaků a frekvence různých řetězců (o dvou znacích) v rámci daného jazyka. Druhou částí jsou informace o nejčastějších záměnách znaků, které vedou k chybě. Tyto informace nicméně nebyly na internetu nalezeny. Část z nich (záměny znaků) uvádí Kernighan et al. (1990) ve své práci, i tak je ale nutné je doplnit o frekvence výskytů znaků apod. Tyto informace je tedy nutné nějak získat. Přesně k tomuto lze využít korpusy. V rámci implementace knihovny byla tedy implementována skupina tříd, které lze využít k vygenerování požadovaných informací. Implementace tedy předpokládá vstupní texty korpusu v podobě textových souborů.

Druhou částí korektoru je jazykový model, přičemž i ten potřebuje informace. Konkrétně se jedná o n-gramy. Aplikace využívá pro svou práci unigramy, bigramy a trigramy. Tyto informace je ale opět problém získat. Pro český jazyk byly na internetu nalezeny pouze frekvence výskytů unigramů (Český národní korpus, 2010). Pro anglický jazyk bylo nalezeno zdrojů již více, nicméně většina z nich je placená nebo vyžaduje placené členství (daná skutečnost byla již popsána v kapitole 2.6). Jako řešení bylo tedy nakonec zvoleno vygenerování daných informací na základě volně dostupných korpusů. Nicméně aby bylo možné jednoduše přidat alespoň základní podporu nějakého dalšího jazyka, byl jazykový model implementován tak, že v případě, kdy nejsou k dispozici informace o vyšším stupni n-gramů, tak je použit nižší stupeň.

Pro vygenerování chybějících informací pro chybový a jazykový model byly tedy v rámci této práce použity volně dostupné korpusy na stránce HC Corpora (Christensen, 2010). Dané korpusy nejsou natolik rozsáhlé, nicméně jako základní zdroj informací pro korekcí je lze použít.

4.5.1 Balení slovníků

Z předchozí kapitoly je zřejmé, že příprava slovníků pro korektor je poměrně problematická. Pod slovníkem je v tomto kontextu myšlen souhrn všech informací, které aplikace (knihovna) vyžaduje k provedení korekce (n-gramy, frekvence znaků, slova jazyka, …). Připravené slovníky pro anglický a český jazyk tedy nejsou tak kvalitní jak by mohly být, když by byly použity kvalitnější korpusy. Aby tedy bylo možné v budoucnu slovníky aktualizovat či přidávat, je nutné vytvořit nějaký způsob pro jejich distribuci (aktualizaci).

Implementovaná knihovna tedy řeší daný problém tak, že je vytvořen zip soubor, ve kterém jsou uloženy všechny potřebné soubory s informacemi. Mimo tyto soubory je součástí také XML soubor, ve kterém jsou uloženy informace o slovníku. Daný soubor obsahuje informace o jazyku, pro který je slovník určen a dále o souborech s frekvencemi n-gramů, znaků, atd. Implementace je inspirována balením rozšíření do OpenOffice. Ukázkový XML soubor pro anglický slovník je přiložen jako příloha.

Balení dále podporuje načítání slovníku se slovy daného jazyka ze souborů ve formátu, který využívá OpenOffice, Aspell a další aplikace. Daný přístup provádí určitým způsobem kompresi. Princip je postaven na tom, že se vezme slovo, z něj se vybere základ slova a z daných předpon a přípon se sestaví seznam pravidel. Poté je u každého základu slova definován seznam klíčů, což jsou odkazy na pravidla, která lze aplikovat na daný základ. Díky tomu, že lze přiřadit k jednomu základu více pravidel a tedy i jedno pravidlo k více slovům, dochází ke kompresi celého slovníku (Atkinson, 2004).

4.5.2 Nástroj na tvorbu korpusu

Situace, kdy není k dispozici moc kvalitních korpusů, které by byly přístupné zdarma a jejich licenční politika umožňovala jejich použití, vedla k tomu, že byl implementován nástroj pro tvorbu vlastního korpusu z webu. Výstup z daného nástroje sice nebyl nakonec použit při tvorbě slovníků, nicméně získané poznatky stojí za zmínění.

Daný nástroj fungoval tak, že mu byly předány URL adresy stránek, které obsahovaly kvalitní text. Daný nástroj poté odesílal požadavky na tyto stránky a stahoval kód těchto stránek spolu se všemi odkazy na dané stránce. Dané odkazy byly poté kontrolovány, zda nevedou na jinou doménu a v případě, že nevedou, tak byly uloženy do fronty k dalšímu zpracování. Kód daných stránek poté procházel očištěním o HTML značky a také postupem, který detekoval části stránek, které se opakují (např. menu, patička, atd.), aby dané části nezkreslovaly frekvence sledovaných entit. Tento postup byl konkrétně postaven na tom, že bylo při jedné dávce staženo stránek více a následně byla porovnána získaná data. Části stránek, které se vyskytovaly v nadměrné míře, byly poté přeskočeny. Finální výstup tedy ukládal frekvenci jednotlivých slov, nicméně systém by bylo možné upravit i pro sledování spojení dvou a více slov (vyšších n-gramů obecně). Z frekvence daných slov lze poté také vypočítat frekvenci znaků a další potřebné informace, které jsou používány v rámci chybového modelu. Data byla do databáze ukládána tak, aby z nich nešlo rekonstruovat původní obsah (byla částečně filtrována).

Aplikace byla realizována pomocí jazyka PHP a využívala MySQL databázi. Provozována byla na virtuálním serveru. Sběr dat poté probíhal tak, že každou minutu byl pomocí plánovače úloh (CRON) spuštěn daný skript, ten poté na základě nastavené logiky zpracoval několik stránek. Tato činnost tedy poté probíhala stále dokola.

Za čtyři dny běhu byla daná aplikace schopna zpracovat celkem skoro pět tisíc unikátních stránek. Dále detekovala více jak 160 000 slov z celkem 4 190 000 zpracovaných řetězců. Průměrný výskyt byl tedy 25 výskytů na unikátní slovo (velikost písmen byla brána v potaz). Pokud vezmeme v úvahu skutečnost, že daný systém měl poměrně dlouhé pauzy mezi požadavky, tak lze dojít k závěru, že si lze vybudovat pomocí této metodiky poměrně jednoduše vcelku rozsáhlý korpus. Podobný způsob nicméně není žádnou novinkou a byl pravděpodobně využit i pro tvorbu jiných korpusů. Dokonce i společnost Google pravděpodobně využila podobnou metodiku, když tvořila jejich Google 1-T N-gram Corpus.

4.6 Optimalizace

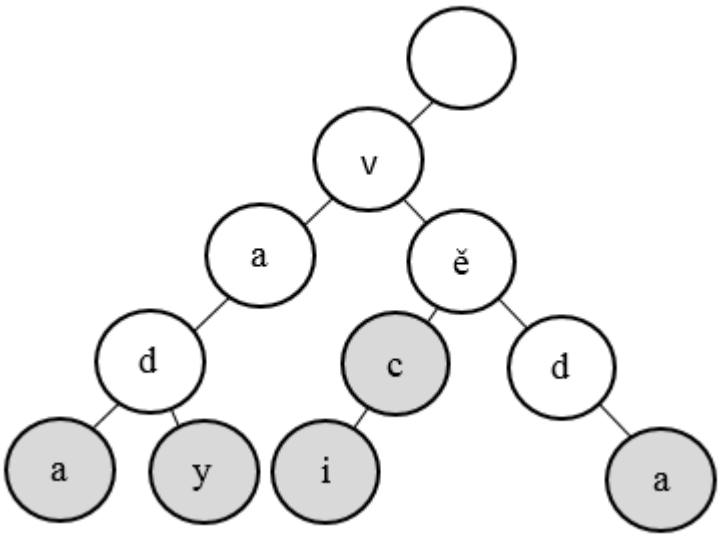
Po implementaci základní funkčnosti, přišla na řadu optimalizace kritických částí. Optimalizace byla odkládána zejména z důvodu toho, aby nebylo věnováno zbytečné úsilí optimalizaci nesprávné části. Jakmile byla dokončena základní funkcionalita aplikace (prototyp), tak se přešlo k profilování, pomocí kterého byla identifikována místa, jejichž optimalizací šlo nejvíce získat. Tato snaha o minimalizaci zbytečné optimalizace bývá nazývána pojmem *premature optimization*, který lze volně přeložit předčasná či nedospělá optimalizace (Knuth, 1974). V souvislosti s tím je nutné upozornit na skutečnost, že člověk může mít poměrně dobrou představu o problematických místech i bez profilování, nicméně díky profilování bude představa proměněna v realitu.

V rámci implementace knihovny bylo realizováno profilování aplikace pomocí vestavěného nástroje ve Visual Studiu. Na základě výstupů bylo identifikováno několik funkcí, ve kterých se při běhu aplikace stráví převážné množství času. Tyto funkce byly následně rozděleny do třech skupin dle jejich účelu. Prvními dvěma velmi podobnými skupinami je vyhledávání ve slovníku (při detekci chyb a filtraci kandidátů) a vyhledávání v n-gramech (pro získání informací o jejich výskytech). Poslední skupina funkcí, která tvořila velkou část výpočetní doby, obsahuje funkce pro načítání celého balíku slovníku (n-gramy, frekvence znaků apod.). Tuto skupinu se nicméně nevyplatí moc optimalizovat, protože výpočetní doba není závislá na délce vstupu. Následující kapitola se tedy bude zabývat pouze optimalizací prvních dvou skupin.

4.6.1 Vyhledávání ve slovníku

Již před začátkem implementace se předpokládalo, že bude nutné optimalizovat vyhledávání ve slovníku a také v n-gramech. Tato skutečnost byla potvrzena profilováním aplikace, díky kterému bylo zjištěno, že majoritní část výpočetního času byla strávena právě vyhledáváním v daných datových strukturách. Daný problém lze velmi dobře ilustrovat na příkladu kontroly jednoho slova. Použitý anglický slovník obsahuje okolo 150 000 slov (český ještě mnohem více). Pro vyhledání jednoho slova bylo tedy nutné projít sekvenčně celý seznam, dokud nebylo slovo nalezeno. Pokud navíc nebylo nalezeno, tak následující část, která generuje kandidáty (běžně i 50, počet závisí na délce slova) musela dané kandidáty vyhledávat znovu ve slovníku. Dané filtrování kandidátů tedy prodlužovalo výpočetní dobu ještě více. To ale stále není konec, protože kandidáti se poté musí ohodnotit, což je realizováno pomocí vyhledávání v n-gramech. Z těchto informací je tedy zřejmé, že je vyhledávání v daných datových strukturách kritickou částí z hlediska celého programu. Popsané skutečnost byly potvrzeny profilováním aplikace.

Vyhledávání je tedy nutné optimalizovat. Jako nejvhodnější způsob se jeví implementace tzv. „trie“ struktury[8], kterou původně navrhl Fredkin (1960). Konkrétní způsob implementace dané struktury je zachycen na obrázku níže.



Obr. 5 Příklad uložení slov v indexovaném slovníku

Daný obrázek obsahuje ukázku části struktury, která obsahuje slova: věda, věc, věci, vada, vady. Šedé políčko značí, že se jedná o konec slova. Pokud bude tedy vyhledáván řetězec „vad“, tak nebude nalezen, protože dané políčko s posledním písmenem „d“ není šedé (tedy označeno jako koncové). Z obrázku je dále zřejmé, že dochází určitým způsobem ke kompresi slovníku. Na uložení slova „věc“ a „věci“ stačí jen čtyři pole (+ informace v poli, že se jedná o koncový znak). Je nicméně nutné zmínit, že místo stejně nemusí být natolik ušetřeno, protože je nutné ukládat ještě ukazatele na jednotlivé objekty.

Z hlediska výkonu dochází díky dané struktuře ke značnému zrychlení. Před implementací optimalizace byla detekce chyb testována na několika vzorcích dat a vzorek textu o velikostí 16 kB trvalo zkontrolovat průměrně 1400ms. Pokud by se tedy vyšlo z daných testovacích korekcí, tak by 1 MB textu trvalo opravit přibližně 90 sekund. Jedná se tedy o velmi dlouhou dobu vzhledem k tomu, že pro korekci jednoho slova se dále generují kandidáti pomocí zmiňované vzdálenosti řetězců (kandidátů může být běžně 50, jak již bylo zmíněno). Doba běhu byla ale díky implementaci struktury popsané výše značně zkrácena. Detekce chyb na původním vzorku o velikosti 16 kB trvala po optimalizaci totiž pouhé 3 ms. V případě, že je daná hodnota přepočítána pro text o velikosti 1 MB, tak vychází

[8] Trie struktura je někdy také nazývána prefix tree (Germann, 2009)

požadovaná doba na přibližných 192 ms. To je v porovnání s předchozí dobou běhu, která byla 90 sekund, zrychlení o více jak 99 %.[9]

Jelikož se daný přístup osvědčil, tak byl dále implementován i pro vyhledávání v n-gramech. Zde byla potřeba optimalizace ještě více zřejmá vzhledem k tomu, že n-gramy obsahují (v závislosti na jejich stupni) ještě mnohem více záznamů (v řádech miliónů) než slovníky. Sekvenční vyhledávání zde tedy nelze vůbec aplikovat. Přístup popsaný výše byl tedy modifikován tak, že se ukládají první slova z n-gramů stejným způsobem a na ně dále navazují další podstromy realizované také stejným způsobem. Úplně na konci je místo informace o tom, zda byl daný n-gram nalezen, uložen počet výskytů daných n-gramů. V případě, že tento počet není nastaven, tak je považován n-gram za nenalezený.

Je nutné zmínit, že danou strukturu je možné ještě dále optimalizovat. Problematiku dále rozebírá Germann et al. (2009), který vztahuje využití dané struktury k práci s jazykovými modely. Autor podrobněji rozebírá další možnosti komprese stromu či propojení s „couváním“, které bylo zmiňováno v kapitole 2.6.3. Jelikož slovníky, které využívá naše aplikace, nejsou natolik rozsáhlé (větší nemáme k dispozici), tak tyto další optimalizace nejsou prozatím dále implementovány.

4.6.2 Paralelizace

Problematiku optimalizace aplikace z hlediska výkonu (výpočetní doby) lze uchopit více způsoby. V momentě, kdy jsou odladěna místa, která byla nejvíce kritická, lze dále pokračovat optimalizací míst, která už na celkovou dobu výpočtu nemají takový vliv. Dále lze ladit použité prostředky, které nabízí programovací jazyk (například nahradit dynamické kolekce za pole, která mohou realizovat stejnou funkci za menší výpočetní čas). Těmito optimalizacemi ale zpravidla už nelze tolik získat, proto se nabízí použití paralelizace.

Předtím tím, než bude paralelizace nějaký způsobem implementována, je nutné zanalyzovat, zda lze paralelizaci vůbec použít. Jinak řečeno, zda lze algoritmus paralelizovat. K tomu je nutné zhodnotit, jak moc velkou část běhu aplikace tvoří dané části, které lze vykonávat paralelně. Ke zhodnocení lze poté využít tzv. Amdahlův zákon (Amdahl, 1967). Danou problematikou se postupně zabývali další autoři. V rámci této práce ale stačí zmínit vzorec (14), který vyplývá z původní práce.

$$S = \frac{1}{r_s + \frac{r_p}{n}} \tag{14}$$

Ve vzorci (14) značí r_p podíl programu, který lze paralelizovat a r_s podíl programu, který nelze paralelizovat ($r_p + r_s = 1$). Počet procesorů je poté skryt v n. Hlavním dů-

[9] Dané hodnoty jsou vypočítány pro slovník, který obsahoval 160 000 slov

vodem, proč je daný zákon zmiňován, je jeho výpovědní hodnota o tom, kdy se vyplatí paralelizací zabývat vzhledem k velikost podílu jednotlivých částí. Pokud lze například paralelizovat 50 % běhu algoritmu, tak dojde dle vzorce (14) ke zrychlení o 33 %. V případě, že může být paralelizováno 80 %, tak je zrychlení rovno 66 %. Z daných informací je tedy zřejmé, že pro to, aby se vyplatilo řešit paralelizaci, je nutné mít dostatečně velký podíl části, kterou lze paralelizovat.

Paralelizaci lze tedy v případě korekce textu aplikovat. Důvodem je skutečnost, že je korekce textu nezávislá na konkrétních stavech. Jinak řečeno, korekce různých částí textů jsou na sobě nezávislé. Paralelizovat tedy lze téměř veškerý běh aplikace. Jako příklad lze uvést situaci, kdy je prováděna korekce dvou textových souborů zároveň. Paralelizovat nejdou v podstatě pouze části, které řeší načítání slovníku a generování výsledků (shrnujících informací).

Samotná implementace paralelního přístupu je ve skutečnosti poměrně komplikovaná. Korekce dvou různých textů může běžet paralelně vedle sebe, nicméně otázka je, jak získat tyto dva různé texty. Pokud je na vstupu více souborů, tak lze rozdělit tyto soubory mezi vlákna a problém je vyřešen. V případě, že je k dispozici pouze jeden soubor, tak se situace komplikuje. Teoreticky je možné soubor načíst a rozdělit mezi vlákna. Ve skutečnosti ale nelze tento postup takto použít, protože by bylo nutné načítat celý soubor do paměti. Implementovaný způsob korekce totiž funguje tak, že zpracovává vstup jako proud dat. Postupně načítá část textu, následně provádí korekci a ihned text ukládá. V případě, že by tedy mělo být rozděleno zpracování jednoho souboru mezi dvě vlákna, tak by musela být implementována nějaká složitá logika. Je tedy zřejmé, že je nutné zvolit pro implementaci paralelního zpracování vhodný přístup (strategii).

V implementované knihovně (aplikaci) byla paralelizace implementována dvěma způsoby. První způsobem je paralelizace na úrovni souborů. Před spuštěním procesu korekce se analyzují soubory na vstupu a rozdělí se na skupiny. Dané skupiny jsou poté opravovány zároveň (paralelně). Počet skupin je určen na základě počtu dostupných jader procesoru. Druhý způsob využívá paralelizaci v průběhu opravy jednoho souboru. Tento způsob je realizován tak, že je soubor zpracováván po dávkách. Na vstupu se identifikují chyby ve vstupním textu a ukládají se do dočasné fronty. Jakmile tato fronta dosáhne určité velikosti, tak je vytvořeno nové vlákno, které danou frontu opravuje. Mezi tím co dané vlákno opravuje chyby, se dále provádí kontrola vstupu. Jakmile je naplněna nová fronta chyb, tak se počká na opravu předchozích chyb (dokončení vlákna s opravami). Opravy předchozích chyb se zapíší na výstup a celý proces se opakuje znovu, dokud není celý soubor zpracován.

Z výše zmíněných informací je jasné, že druhý způsob implementace paralelizace není zdaleka tak efektivní, jak by mohl být. Důvodů, proč není řešen sofistikovaněji, je několik. Prvním důvodem je skutečnost, že vstup musí být zpracováván jako proud dat (již zmíněno výše). Druhým důvodem je skutečnost, že první způsob paralelizace bude dostatečný ve většině případů. Text, který bude zpracováván, totiž bude pravděpodobně často rozdělen do více souborů. Lze se domnívat, že rozdělení bude realizováno z toho důvodu, aby šlo s textem pracovat i nějakým

způsobem ručně. Větší soubory mají totiž problém textové editory otevřít. Dalším důvodem může být také situace, že texty mohou být rozděleny do nějakých skupin (např. kvůli dolování znalostí).

V následující části bude provedeno zhodnocení dané problematiky z pohledu obou zmíněných způsobů. Pro testování byly použity dva zdroje dat. Prvním z nich jsou recenze hotelů ze stránky *booking.com*, které jsou pro tento příklad uloženy v jednom souboru. Jako druhý zdroj dat byly použity diskuze z 20 newsgroups, které jsou uloženy ve více souborech (Mitchell, 1999). Konkrétní specifika daných dat zde nejsou podrobněji rozebírána, protože nyní není řešena přesnost korekce. Analýza je prováděna pouze z pohledu anglického jazyka. V českém jazyce by byly výsledky podobné. Uváděné hodnoty jsou zprůměrovány z více měření, protože jednotlivé běhy mohou být ovlivněny aktuálním vytížením procesoru. Je nutné také zmínit, že daná měření nezahrnují části na zpracování přehledu o provedených korekcích a samotné načítání slovníku.

Tab. 10 Přínosy paralelizace

	Booking.com	20 newsgroups
Původní doba	1863 s	1222 s
Par. zpracování souborů	1890 s	704 s
	101,5 %	**57,6 %**
Dávkové zpracování	1469 s	615 s
	78,9 %	**50,4%**

Procentuální hodnoty zachycují celkovou dobu běhu korekce vzhledem k původní době. Z výsledků je zřejmé, že vstupní formát (jeden či více souborů) má vliv na zisk z hlediska rychlosti programu. V případě zpracování jednoho souboru dokonce došlo k navýšení doby potřebné ke korekci. Naopak v situaci, kdy bylo na vstupu více souborů, je vidět významný přínos paralelizace. Pokud jsou navíc aplikovány oba přístupy současně, tak dojde téměř k dvojnásobnému zrychlení, pokud je na vstupu více souborů. To je v podstatě ideální výsledek vzhledem k tomu, že počítač, na kterém byl test prováděn, obsahuje procesor s dvěma jádry (čtyři logická jádra díky technologii Hyperthreading[10]).

Z výše zmíněných informací je tedy zřejmé, že se paralelizací vyplatí zabývat. Ze získaných zkušeností také vyplývá, že lze použít celou řadu přístupů k implementaci paralelního zpracování. Volba vhodného přístupu nicméně není vždy úplně jednoduchá. To je zřejmé i z této práce, kdy je přínos závislý na formátu vstupních dat.

[10] Díky dané technologii lze zjednodušeně řečeno lépe využít jádra procesoru. Problematiku podrobněji rozebírá Marr et al. (2002)

4.7 Jazyková specifika

Již z přehledu literatury je zřejmé, že hlavní vývoj v korekci textu probíhá na metodikách pro anglický jazyk. Bylo také naznačeno, že u jiných jazyků se vyskytují různá specifika. Z daných informací tedy vyplývá, že je nutné aplikaci přizpůsobit tak, aby si byla schopna poradit s více jazyky. Následující kapitoly se tedy zabývají danou problematikou.

V souvislostí s podporou více jazyků je vhodné zmínit roli definované vývojové metodiky pomocí prototypů. Daná metodika měla totiž značné přínosy pro integraci podpory více jazyků. První prototypy jednotlivých částí byly připraveny pouze pro opravu textu v anglickém jazyce. Na daných prototypech byly poté identifikovány části, které bude nutné upravit kvůli kompatibilitě s jinými jazyky. Jako konkrétní příklad lze uvést zpracování vstupního textu. V anglickém jazyce se může v rámci slova vyskytovat apostrof, naproti tomu slova v českém jazyce apostrof obsahovat nemohou. Z prototypu bylo tedy zřejmé, že regulární výrazy, které se používají pro zpracování vstupu, se budou muset přizpůsobovat dle jazyka textu. To ve své podstatě předznamenalo oddělení daných regulárních výrazů od samotné funkcionality (jsou tedy načítány ze slovníku).

Problematika podpory více jazyků bude v následujících kapitolách rozebírána zejména z pohledu českého jazyka.

4.7.1 Diakritika

Diakritika tedy vede k tomu, že je původní abeceda rozšířena o další znaky. Pokud je vztažena tato skutečnost k chybovému modelu, tak je jasné, že informace[11], které chybový model používá pro opravu, bude nutné generovat dynamicky v závislosti na jazyce. Je tedy zřejmé, že implementované funkcionality pro generování daných informací musí mít k dispozici informace o jazyku, pro který budou generovány. Pod pojmem informace je zde myšlen zejména seznam jednotlivých znaků abecedy. Jeho nutnost lze ilustrovat na zpracování vstupu z korpusu. V rámci textu korpusu se totiž mohou vyskytovat různé znaky, jejichž výskyty jsou irelevantní, protože chybový model s nimi nepracuje (např. závorky, speciální znaky, …). Aby byla zaručena univerzalita, je nutné poskytnout těmto generátorům informaci o abecedě dle konkrétního jazyka. Zde tedy přichází na řadu balení slovníku. V případě, že je vytvářen nový slovník, tak je nutné nejprve vytvořit základní XML soubor, který ho popisuje (viz kapitola 4.5.1). Tento soubor tedy obsahuje mimo jiné informace i danou abecedu. Tím pádem mohou implementované funkcionality čerpat potřebné informace z balení slovníku. V případě zmiňovaného zpracování vstupu lze tedy využít informace (z balení slovníku) k filtraci znaků na vstupu generátorů při zpracování korpusu.

Hlavním problémem při práci s diakritikou není nicméně zmiňovaná příprava dat pro chybový model, ale skutečnost, že text často nemusí diakritiku obsahovat

[11] Za informace jsou zde považovány frekvence znaků, četnost záměn znaků, …

vůbec. Tím pádem, když je prováděno generování kandidátů na korekci, tak jsou generováni pouze kandidáti ve vzdálenosti jedna od původního slova. Pokud by tedy chybělo u nějakého slova více diakritických znamének, tak nebude slovo opraveno. Nyní tedy nastává otázka jak danou situaci řešit. Jednou z možností je rozšířit vzdálenost generování kandidátů na dvě editace. Díky tomu bude chyba ve slově, ve kterém chybí dvě diakritická znaménka opravena. I tak ale stále může nastat situace, kdy bude detekováno slovo, kde chybí diakritických znamének ještě více (např. ve slově „rozšířit"). V daných případech je už tento přístup značně nevhodný, protože s počtem kandidátů roste významně výpočetní náročnost. Z tohoto důvodu je nutné přijít s nějakou cestou, která bude schopna doplnit diakritiku pro celé slovo, pokud možno s největší efektivitou.

Nabízí se tedy jiná cesta a to generování všech možných variant slova pouze vzhledem k diakritickým znaménkům a jejich následná filtrace dle slovníku. Pro generování se budou opět používat informace o znacích z balení slovníku. Konvenční přístup, kdy by byly vygenerovány všechny možné varianty, také není úplně efektivní. Počet kandidátů je totiž stále poměrně velký. Situaci lze ilustrovat na slově „večeře". Na místě druhého znaku totiž může být mimo krátké písmeno „e", také dlouhé nebo může být s háčkem. To stejné platí pro další znaky. Z toho tedy vyplývá, že pokud by měly být generovány všechny varianty, tak jich bude nepřeberné množství. Tento problém nicméně lze řešit dále. K řešení lze totiž využít strukturu popsanou v kapitole 4.6.1. Kandidáty lze generovat tak, že se postupně berou znaky z původního slova a zkouší se nahrazovat za možné znaky s diakritikou. Následně se zkontroluje, zda v dané struktuře existuje cesta pro daný řetězec, pokud ano, tak se pokračuje s generováním nad množinou řetězců, pro které cesta existuje dále. Vygenerované řetězce, pro které cesta neexistuje, jsou poté rovnou zapomenuty. Na konci procesu je provedena kontrola, zda se vygenerované řetězce nachází ve slovníku. Tento postup vede k tomu, že je počet generovaných kandidátů minimalizován. Je tedy zřejmé, že tento postup obsahuje nejméně kroků a tím pádem byl implementován.

I přesto, že byl nalezen vhodný způsob k doplnění diakritiky, tak stále zůstává jeden zádrhel. Ten spočívá v tom, že je nutné tento způsob opravy slova nějak propojit s *noisy channel* modelem, který vyvíjená knihovna (aplikace) využívá. Problém konkrétně spočívá v tom, jak dát dohromady chybový model a definovaný přístup pro přidání diakritiky. Implementace byla nakonec realizována tak, že je nejprve provedena kontrola (doplnění) diakritiky. Pokud byl nalezen nějaký kandidát, tak je chybový model přeskočen. V případě, že je kandidátů více, tak je k volbě vhodného kandidáta použit jazykový model. V praxi tato situace ale nastává minimálně, protože většinou je vygenerován pouze jeden kandidát. Lze tedy říci, že část, co řeší diakritiku, je považována za chybový model, přičemž ten původní je z procesu zcela vynechán.

Daný přístup byl poté otestován při korekci slovníku HC Corpora (Christensen, 2010). Na základě toho bylo zjištěno, že část pro opravu diakritiky realizovala opravu 7,7 % nalezených chyb. Z toho 6,4 % oprav bylo realizováno rovnou, protože byl vygenerován pouze jeden kandidát. To znamená, že pouze v 1,3 % případů

byl použit dodatečně jazykový model pro jejich ohodnocení. Tento experiment dokazuje, že zvolená posloupnost aplikace přístupů ke korekci slova je použitelná (původní model stále řeší majoritní část chyb). Zde je nutné zmínit, že daná měření jsou značně závislá na kontrolovaném textu a na použitých slovnících.

4.7.2 Vzdálené závislosti mezi slovy

Jednou z problematických částí zvoleného přístupu jsou situace, kdy slovo ve kterém je chyba, je závislé na slově, které je v jiné části věty nebo se nachází v jedné z předchozích vět. Daná slova tedy mají mezi sebou tak velkou vzdálenost, že jejich vztah nemohou zachytit n-gramy v jazykovém modelu. Tato skutečnost je částečně v rozporu s předpokladem definovaným Markovem, který byl podrobněji zmiňován již v kapitole 2.6.1.

Situaci lze dobře demonstrovat na příkladu. Věta „Nakoupil jsem dostatek jídla na celý týden.“ může mít chybu ve slově „nakoupil“, přičemž chyba může být například přidání znaku na konec slova (tedy „nakoupilp“). V daném případě budou vygenerování kandidáti jako: nakoupil, nakoupila, nakoupilo, nakoupili. Z kandidátů je zřejmé, že jejich koncovka závisí na podmětu. Ten ale vychází z kontextu celé diskuze (z jedné z předchozích vět). Korekce pomocí chybového a jazykového modelu tím pádem bude mít problém identifikovat, kterou variantu použít. Jazykový model je totiž závislý na korpusu, ze kterého byly informace generovány. Tím pádem bude vyhodnocovat lépe varianty, které měly v rámci daného korpusu větší výskyt. Stejná situace se opakuje i u chybového modelu, ním využívané informace jsou také generovány z korpusu.

Zde je nutné zmínit, že tento problém se týká zejména českého jazyka (a podobných jazyků). V angličtině nejsou formy slov totiž natolik závislé na slovech ve větší vzdálenosti. Model využívaný pro korekci je tedy více vhodný pro anglický jazyk. Tato skutečnost je dána tím, že vývoj daných metodik, jak již bylo zmíněno, probíhá zejména na anglickém jazyce.

5 Vyhodnocení řešení

Následující kapitoly se budou zabývat zhodnocením implementovaného řešení z pohledu přesnosti korekce a výkonnosti řešení. V rámci těchto kapitol budou opět zmíněny problémy, které bylo nutno řešit v souvislosti s danými částmi.

5.1 Testovací data

Pro vyhodnocení řešení je nutné použít vhodná data. V následujících kapitolách budou pro potřeby vyhodnocení použity recenze hotelů, které pochází z portálu *booking.com*, který slouží k rezervaci pokojů v hotelech (Dařena, 2010). Jedná se tedy o reálná data, na jejichž korekci může být vyvíjená aplikace použita i v praxi. Daná data jsou také vhodná k tomu, aby z nich byly dolovány znalosti pomocí metodik k tomu určených. Recenzí je tím pádem velké množství, což je v souladu se zadáním aplikace, která má být schopna zpracovat velké objemy.

Recenze jsou ve formátu prostého textu a jsou rozděleny do souborů podle jejich jazyků. V souvislosti s jazykem je nutné zmínit, že rozdělení není zcela dokonalé, a tak se mohou v některých souborech vyskytovat i texty v jiném jazyce. Jedna recenze se skládá obvykle z několika vět. Recenze jsou v daných souborech uloženy tak, že jeden řádek obsahuje jednu recenzi. V rámci vyhodnocení řešení budou používány zejména recenze v anglickém jazyce. Soubor s recenzemi v anglickém jazyce obsahuje přibližně 1 930 000 recenzí. Celková velikost daného souboru je přibližně 250 Mb. Jak již bylo řečeno, tak většina experimentů bude prováděna na textu v anglickém jazyce. Součástí bude i zhodnocení přesnosti korekce pro český jazyk. Soubor s recenzemi hotelů v českém jazyce obsahuje pouze 17 000 recenzí, ale i tak je jejich počet dostatečný k realizaci základního zhodnocení.

5.2 Přesnost korekce

Předtím než může být provedeno zhodnocení přesnosti korekce, je nutné zvolit vhodný přístup k analýze výsledků. Jelikož je aplikace určena primárně pro kontrolu rozsáhlých textů, tak nelze provést kontrolu výsledků ručně (kvůli objemu dat). Množství detekovaných potenciálních chyb bude totiž velmi vysoké. Jako alternativní varianta se nabízí použití testovacích dat či pouze výběr podmnožiny z původních dat, nicméně v daných případech by nemusel být pohled na výsledky objektivní. Typy chyb a četnost chybovosti totiž může do značné míry záviset na tom, kdo text tvořil (jakou měl slovní zásobu, znalost problematiky, …). V rámci analýzy přesnosti korekce budou použita reálná data (recenze hotelů), která byla popsána v předchozí kapitole. Tato kapitola se bude zabývat zhodnocením přesnosti pouze na recenzích v anglickém jazyce. Z těchto dat bude tedy vybrán vzorek chyb, který bude analyzován pomocí ruční kontroly.

Konkrétní výběr vzorku a jeho analýza bude poté probíhat tak, že se provede korekce celého textu se zapnutým podrobným generováním informací o prováděˇ-

ných korekcích. Ty budou poté uloženy do souboru. Z tohoto souboru bude následně vybrán náhodně vzorek chyb (náhodný výběr je realizován pomocí skriptu). Jedna chyba ve vzorku bude obsahovat informace o původním slově, jeho okolí a případné opravě. Analýza bude poté prováděna pomocí vlastního nástroje, implementovaného jako součást aplikace. Tento nástroj funguje tak, že zobrazí jeden záznam (chybu) z daného vzorku (včetně okolních slov) a uživatel poté zařadí chybu do správné třídy. Zařazením chyby je vynuceno zobrazení další položky ze vzorku. Použití daného nástroje by tedy mělo urychlit zpracování vzorku a tím pádem umožnit analýzu většího vzorku dat. Podnětem pro přípravu vlastního nástroje byla skutečnost, že je nutné zařazovat výstup do více tříd (skupin) než dvou základních (správná či špatná korekce). V následující části jsou zmiňované třídy podrobněji definovány. Z tabulek, které budou zachycovat konkrétní výsledky, na ně bude později odkazováno.

1. Slovo, které nebylo opraveno a ani nemělo být opraveno (tedy nemělo být ani detekováno, protože je správně)
2. Slovo, které je špatně, ale nebylo opraveno
3. Korektně provedená korekce
4. Chybně provedená korekce
5. Oprava správného slova (slovo pouze chybí ve slovníku)
6. Oprava správného slova na špatné, protože dané slovo je z jiného jazyka nebo se jedná o název

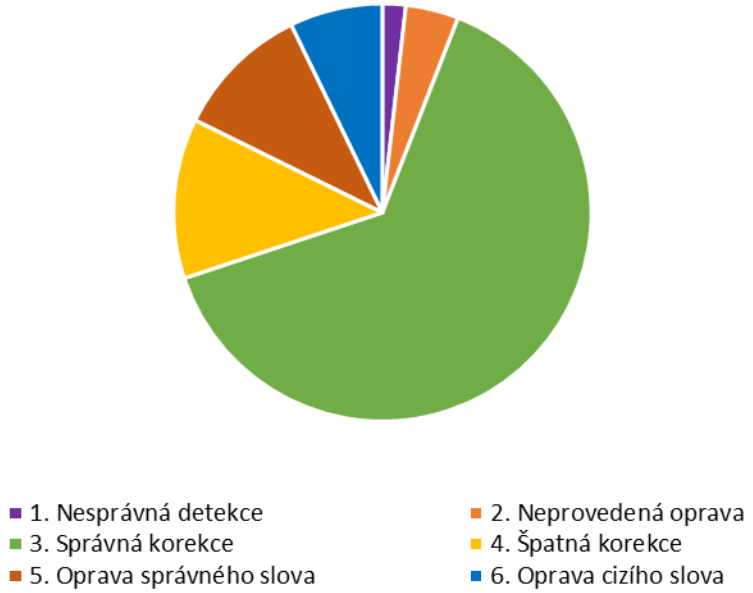
V následující tabulce jsou tedy zobrazeny konkrétní výsledky analýzy vzorku 1000 náhodně vybraných chyb z korekce recenzí hotelů.

Tab. 11 Výsledky analýzy vzorku reálných dat

1. Nesprávná detekce	18	1,8 %
2. Neprovedená oprava	41	4,1 %
3. Správná korekce	640	64 %
4. Špatná korekce	124	12,4 %
5. Oprava správného slova	105	10,5 %
6. Oprava cizího slova	72	7,2 %

Z daných výsledků je zřejmé, že aplikace byla schopna opravit přibližně dvě třetiny chyb. Tento výsledek není vůbec špatný. Danou skutečnost lze prezentovat na podílech ostatních tříd. Třída chyb, které byly opraveny na špatné slovo v důsledku toho, že původní slovo není obsaženo ve slovníku, tvoří více jak 10 %. Kdyby tedy měla aplikace k dispozici kvalitnější (rozsáhlejší slovník), tak by ke korekci (detekci) těchto slov nedošlo. Díky tomu by byl výstup korekce ještě kvalitnější. Kvalitnější slovník by dále znamenal, že by při korekci byla zvážena i další slova. Díky tomu by třídy chyb, kdy nebyla provedena oprava a kdy byla provedena špatná korekce, tvořily opět o něco menší podíl. Poslední skutečnost, která může zvýšit

přesnost korekce, je úprava generování kandidátů, tak aby se generovaly až do vzdálenosti dvě. Autor si při kontrole výsledků všiml skutečnosti, že až třetina špatných korekcí byla způsobena tím, že správné slovo bylo ve vzdálenosti více než jedné editace. Je nutné zmínit, že zahrnutí daného slova mezi kandidáty na korekci ještě neznamená, že bude vybráno, takže skutečný nárůst přesnosti nemusí být až tak velký.



Obr. 6 Graficky znázorněné podíly operací na anglickém jazyce

Výsledky lze dále přepočítat tak, aby se na ně dalo podívat pouze z pohledu správné a špatné korekce. Nesprávná detekce v daném případě bude zcela vynechána, protože text nebyl změněn. Ostatní korekce, které vedly ke špatnému slovu nebo špatné slovo ponechaly, se poté sečtou. V daném případě bude podíl správné korekce tvořit 65,1 % a tím pádem podíl špatné korekce 34,9 %.

Tab. 12 Příklady nalezených chyb

Správný tvar slova	Nalezené chybné tvary slova
accommodation	accomomdation, accopmmodation, accomomodation, accommodtion, accommodaltion, accommodiation, accommoodation, accommocdation, occommodation, …
breakfast	mreakfast, breakfest, brekfast, breakfeast, breackfast, brakfast, beakfast, breafast, breakast, breakfst, breakfas, ...
restaurant	reataurant, restauarant, restaruant, restraurant, restarants, restauant, restautant, rstaurant, restaurnt, …

Informace zmíněné v předchozím odstavci lze shrnout tak, že jsou dvě cesty pro navýšení kvality (přesnosti korekce). První cestou je použití lepších podkladů

(slovníky). Druhou cestou je navýšení počtu možných editací pro získání kandidátů. Je tedy zřejmé, že původní rozhodnutí o tom, že pro generování kandidátů lze využít pouze jednu editaci (problematika byla rozebírána v kapitolách 2.5.1 a 4.3) je do určité míry omezující. Jedna editace byla původně upřednostněna kvůli rychlosti procesu korekce. Problematika kompromisů mezi přesností a výkonem bude diskutována později.

V předchozích částech nebyla podrobněji rozebírána přesnost detekce chyb, je tedy nutné zmínit důvody, které k tomu vedly. Prvním a hlavním důvodem je skutečnost, že přesnost detekce je závislá zejména na použitém slovníku. Pokud bude slovník obsahovat více slov, tak bude detekováno méně chyb a naopak. Implementovaným algoritmem nelze detekci výrazným způsobem ovlivnit (ačkoliv nějaké možnosti, jako detekce názvů, zkratek, apod. zde jsou). Druhým důvodem je skutečnost, že zhodnocení přesnosti lze provádět velice obtížně, protože výskyt chyb závisí na konkrétním původu textu. Různí autoři mohou dělat různé typy chyb, jak již bylo naznačeno dříve. Posledním důvodem je samotné měření, na základě kterého byla prováděna analýza v předchozí části. Při něm bylo zjištěno, že detekce identifikovala celkem více než 800 000 potenciálních chyb. Hodnota je tak velká, že lze předpokládat, že takové množství chyb v textu ani není (některá slova byla totiž identifikována jako chybná, ale nebyla opravena).

5.2.1 Korekce textu v českém jazyce

Tato kapitola se bude věnovat stejné problematice jako předchozí kapitola, s tím rozdílem, že zde bude analyzována dosažená přesnost korekce na textu v českém jazyce. Pro testování budou opět použity recenze hotelů, které byly již popsány dříve. Postup pro zhodnocení přesnosti bude také stejný jako v předchozí kapitole.

Tab. 13 Výsledky analýzy vzorku reálných dat na českých recenzích

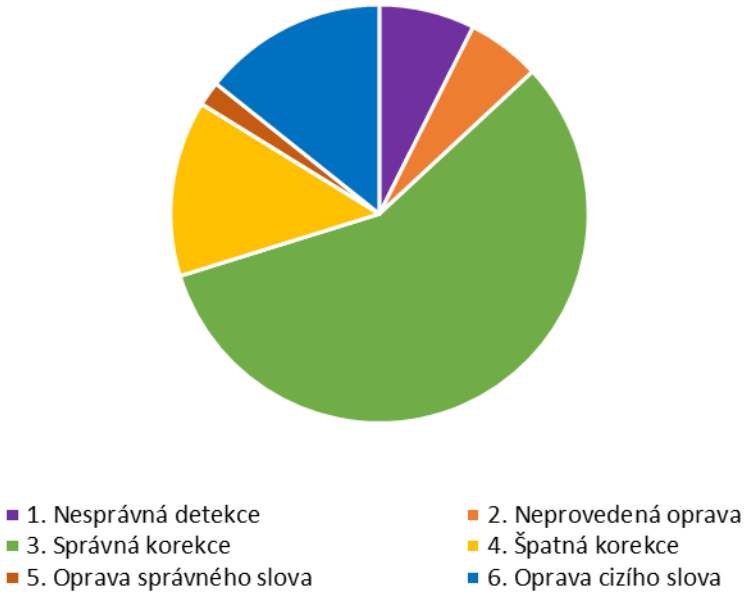
1. Nesprávná detekce	74	7,4 %
2. Neprovedená oprava	57	5,7 %
3. Správná korekce	571	57,1 %
4. Špatná korekce	136	13,6 %
5. Oprava správného slova	19	1,9 %
6. Oprava cizího slova	143	14,3 %

Ze získaných hodnot je zřejmé, že korekce je u českého jazyka méně přesná. Ve skutečnosti by navíc byla situace ještě horší, protože podíl správné korekce značně navyšuje speciální část modelu pro korekci, která má na starost diakritická znaménka. Autor si totiž při vyhodnocování výsledků všiml, že velkou část chyb tvořila slova, ve kterých diakritická znaménka zcela chyběla.

Tab. 14 Příklady nalezených chyb v recenzích v českém jazyce

Správný tvar slova	Nalezené chybné tvary slova
personál	presonál, peronál, prsonál, personáli, perosnál, pesrsonál, …
snídaně	snidaně, snídane, snídně, sníddaně, snídadě, sídaně, …
výborná	vyborna, ýborná, výbrná, výbroná, výborna
parkování	parkovní, pakrování, parkovění, parkovani

Naproti tomu podíly nesprávné detekce a opravy cizího slova jsou zde značně větší. Důvodem je skutečnost, že některé recenze byly napsány ve slovenském jazyce. Ten je natolik podobný českému, že velká část slov ve slovenském jazyce byla opravena na slova v českém jazyce.



Obr. 7 Graficky znázorněné výsledky korekce textu v českém jazyce

Na rozdíl od korekce textu v anglickém jazyce, je u českého jazyka rozdíl v hodnocení úspěšnosti detekce chyb. Autor si při kontrole totiž všiml, že se v textu vyskytuje jeden typ chyb, který nebyl vůbec detekován. Jedná se o tvrdá a měkká písmena „i“ a „y“ na konci slov. U daných slov jsou většinou možné oba tvary, což vede k tomu, že jsou daná slova označena jako správná (oba tvary jsou totiž ve slovníku). Tato problematika byla již nastíněna v kapitole 4.7.2, ve které bylo na danou situaci pohlíženo spíše z pohledu výběru vhodného kandidáta na korekci. Ve skutečnosti se ale jedná o stejný problém.

Pokud by měly být zmíněny části, které lze vylepšit, aby bylo dosaženo lepšího výsledku, tak se nabízí lepší propojení modelu na doplnění diakritiky a chybového modelu. V rámci kontroly chyb byla totiž identifikována situace, kterou nebyl schopný implementovaný přístup vyřešit. Jedná se o situaci, kdy celé slovo bylo napsáno bez diakritiky a do toho v něm byla chyba. Aplikace není tuto chybu

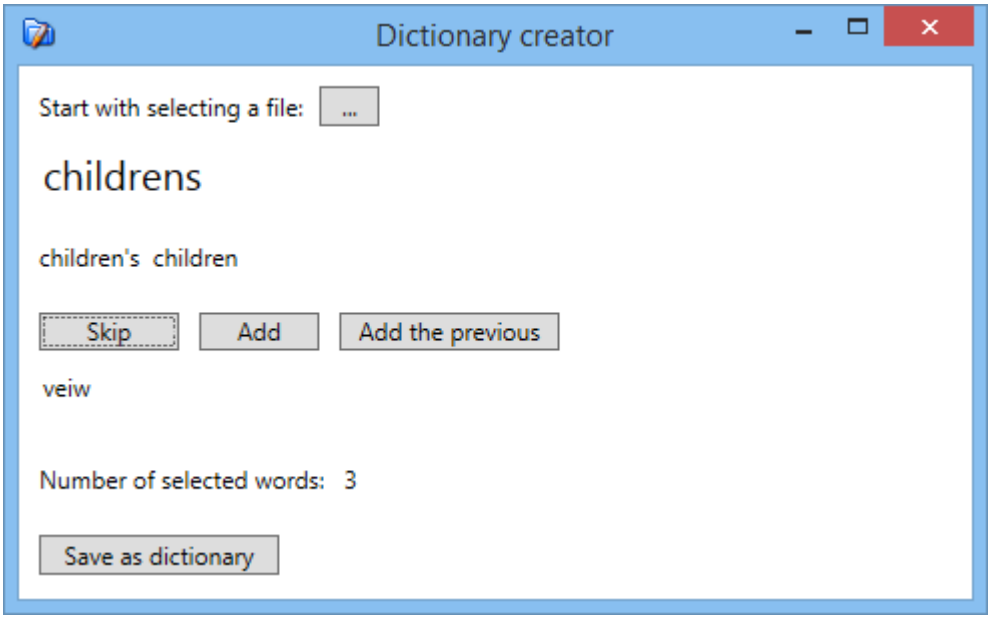
schopna opravit, protože může buď přidat pouze diakritiku, nebo opravit překlep. Částečným řešením je generování kandidátů až do vzdálenosti dvou editací, nicméně toto řešení stejně nedokáže opravit situace, kdy chybí více diakritických znamének.

5.2.2 Vliv pomocného slovníku

V rámci aplikace byla implementována funkcionalita, která umožňuje použití vlastního doplňkového slovníku. Tuto možnost lze využít v situaci, kdy je nutné provádět korekci nějakých specifických textů. Jelikož je aplikace určena primárně pro korekci textů, ze kterých budou poté dolovány znalosti, tak je zřejmé, že dané texty se mohou týkat různých oblastí. V rámci daných oblastí se mohou vyskytovat například různá označení, zkratky a obecně cizí výrazy, které běžně nejsou ve slovníku. Přesně v těchto situacích se vysloveně nabízí použití doplňkového slovníku.

Danou funkcionalitu lze ale využít dále. Textová data, která aplikace zpracovává, lze díky jejich velikostem vlastně považovat za korpus. Daná skutečnost je zřejmá již ze zmiňovaných recenzí hotelů, na kterých bylo provedeno vyhodnocení přesnosti implementovaného algoritmu v předchozí kapitole. Nabízí se tedy otázka jak daný korpus využít. Odpověď vychází z přístupu, který je popsán v kapitole 2.7.3. V dané kapitole je nastíněn přístup, který využívá frekvence výskytů slov na internetu k tomu, aby mohl algoritmus stanovit, zda je slovo špatně či ne. Podobný přístup v omezeném způsobu lze aplikovat i zde. Jedním z výstupů, který informuje uživatele aplikace o provedených korekcích, je seznam slov a toho jak často se vyskytují. Jelikož je seznam seřazen dle výskytů, tak se od něj lze dobře odrazit při tvorbě vlastního doplňujícího slovníku. Člověk tím pádem může slova, která se nejčastěji vyskytují, projít a vybrat z nich ty, které jsou správně, ale byly špatně opraveny. Z daných slov může být poté vytvořen doplňující slovník, který lze použít při opakovaném provedení korekce. Díky tomuto přístupu by tím pádem měl být výstup kvalitnější a neměla by být mylně opravena slova, která jsou správně (chybí v původním slovníku). Omezením, které bylo zmiňováno v souvislosti s načítáním výskytů slov z internetu, je zde nutnost ručního výběru správných slov. Korpus je totiž sice poměrně velký, ale velikost bohužel není dostatečná na to, aby se na frekvence dalo plně spoléhat.

Vytvoření doplňujícího slovníku je dosti pracné, pokud bude danou činnost člověk provádět ručně. Důvodem je skutečnost, že výstup obsahující frekvence slov je ve formátu CSV (data oddělená středníkem). Pro uživatele je tím pádem kontrola souboru poměrně náročná, protože se musí soustředit vždy na jeden řádek a vnímat dané souvislosti (soubor obsahuje i informace o tom, na jaká slova bylo slovo případně opraveno). Zejména z tohoto důvodu byla implementována doplňující funkcionalita, která slouží k vytvoření slovníku. Rozhraní aplikace zachycuje Obr. 8. Na prvním řádku je aktuálně zpracovávané slovo, poté následují slova, na která bylo zpracovávané slovo opraveno. Po kliknutí na některé z tlačítek pro přeskočení slova či přidání slova do slovníku je poté automaticky zobrazeno další slovo.



Obr. 8 Aplikace na tvorbu slovníku

Vliv doplňujícího slovníku byl zjištěn pomocí experimentu. Autor vytvořil nový doplňující slovník pomocí výstupu korekce recenzí hotelů v anglickém jazyce z předchozí kapitoly. Vytvoření daného slovníku trvalo přibližně půl hodiny a obsahovalo 30 slov. Čas je zmíněn, protože soubor s frekvencemi slov je velký a tím pádem ho nelze zpracovat celý. Kvůli dané skutečnosti je soubor seřazen dle výskytů tak, aby uživatel nejprve zpracoval ta slova, která se vyskytují nejčastěji.

Konkrétní vliv daného slovníku je tedy následující. V původním vstupu bylo detekováno celkem 820 000 slov, která korekce považovala za chybná (některá z nich ale neopravila). Po tom, co byl použit doplňující slovník, klesl tento počet na 772 000 slov. Z toho je tedy zřejmé, že daný doplňující slovník měl obrovský vliv na kvalitu korekce. Užitečnost doplňujícího slovníku potvrzuje také fakt, že slovník měl pouze 30 slov. Pouhých těchto pár slov stačilo k tomu, aby byl redukován počet detekcí o téměř 6 %. Je nutné zmínit, že podíl je takto vysoký, pravděpodobně kvůli skutečnosti, že se většina textu věnovala jedné konkrétní problematice (hodnocení hotelů).

Pokud budou výsledky vztaženy k výsledkům v Tab. 11, tak lze říci, že díky danému slovníku dojde ke zmenšení podílu situací, kdy je správné slovo opraveno na špatné. Na samotnou přesnost korekce nebude mít daný slovník významnější vliv.

5.2.3 Detekce cizího jazyka

V rámci kontrolovaných textů se mohou často vyskytovat různá slova či dokonce celé věty, které jsou v jiném jazyce. Z tohoto důvodu byla implementována funkcionalita, která má předcházet tomu, aby byla tato slova opravována na jiná a nevznikal tak nepříjemný mix dvou jazyků, který nedává smysl.

Tento problém lze řešit celou řadou cest. Některé jsou více sofistikované a některé méně. V rámci implementace aplikace byl problém řešen zcela jiným způsobem. Ten spočívá v tom, že kontroluje, kolik slov bylo detekováno jako chyba v aktuálním okně. Pod oknem zde myslíme jedno kontrolované slovo a jeho kontext. V aktuálním nastavení má aplikace nastavenu velikost kontextu na dvě slova. To znamená, že vlevo a vpravo od kontrolovaného slova jsou v okně uložena dvě slova navíc. Samotná funkčnost je postavena na tom, že v případě, že jsou čtyři z pěti slov v daném okně špatně, tak je daná část textu považována za cizí jazyk. Druhou částí je pak podmínka, která vyžaduje, aby slova v bezprostředním okolí kontrolovaného slova byla správně. Mohlo by se zdát, že to povede k tomu, že mohou být některé chyby ignorovány a tím pádem bude kvalita korekce menší. Ve skutečnosti by tato situace ale neměla být problémem. Pokud budou špatně čtyři slova z pěti, tak by stejně nebylo možné provést kvalitně korekci, protože nejsou k dispozici kvalitní informace o okolí.

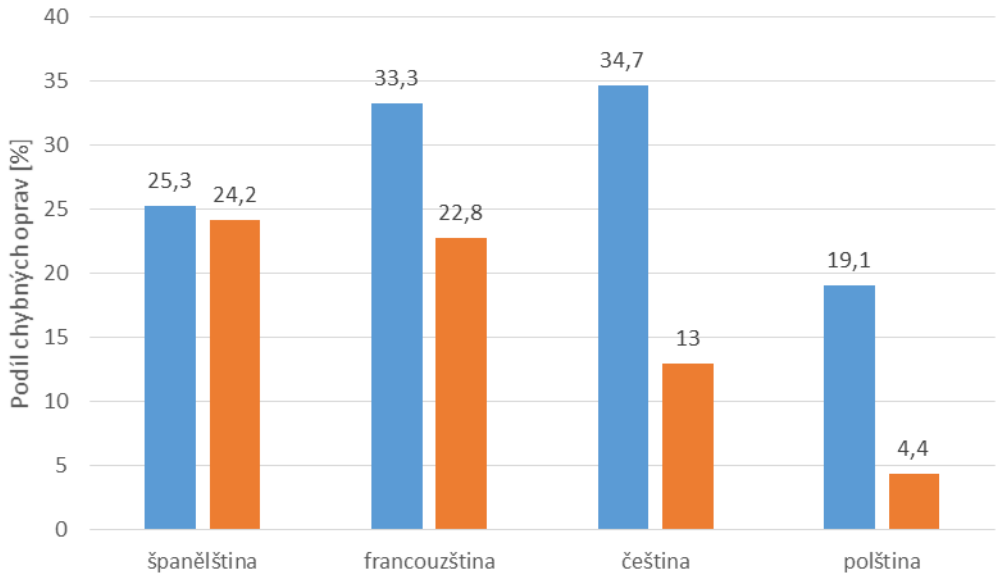
Vyhodnocení daného přístupu bylo realizováno opět na recenzích hotelů (v anglickém jazyce), s tím rozdílem, že zde byly ručně vybrány části, které jsou v jiném jazyce než zbytek textu. Následně byl vytvořen soubor, který obsahoval recenze v jazyce, ve kterém byla prováděna kontrola. Ty byly poté proloženy recenzemi v jiných jazycích. Recenze v jiných jazycích byly ve španělštině, češtině, ruštině, arabštině, francouzštině a polštině. Samotné vyhodnocení následně probíhalo tak, že byly ručně kontrolovány detekované chyby a jejich případné korekce. Vzorek tím pádem musel být poměrně malý (z každého jazyka bylo použitou pouze 300 slov). Je tedy jasné, že dané vyhodnocení slouží pouze pro základní zhodnocení daného přístupu. Aby bylo možné provést zhodnocení podrobněji, tak by bylo nutné připravit velké množství testovacího textu. Důvodem je skutečnost, že funkčnost dané metodiky bude do značné míry záviset na podobnosti jednotlivých jazyků. Daná problematika je natolik rozsáhlá, že by jí mohla být věnována celá práce.

Samotné hodnocení bylo prováděno srovnáním detekovaných chyb se vstupním textem. Text byl poté analyzován z hlediska slov a řazen do třech tříd (skupin). První třídu tvoří chybně detekovaná slova, která ale nebyla opravena. Druhou skupinou jsou chybně detekovaná a opravená slova a třetí skupinou jsou slova, ve kterých nebyla chyba detekována. Následující tabulka zachycuje podíly daných tříd před tím, než byla funkčnost aktivována a po aktivaci. Hodnoty jsou přepočteny na procenta vzhledem k celkovému počtu slov jiného jazyka.

Tab. 15 Výsledky korekce částí textu v cizích jazycích

	Chybná detekce		Chybná oprava	
	před	po	před	po
španělština [%]	14,1	12,8	25,3	24,2
francouzština [%]	15,8	10,5	33,3	22,8
čeština [%]	17,3	6,5	34,7	13
polština [%]	48,5	16,2	19,1	4,4
ruština [%]	0	0	0	0
arabština [%]	0	0	0	0

Při pohledu na výsledky je nutné si uvědomit, že chybné detekce slov nejsou problém, protože nedojde k opravě správného slova v cizím jazyce na špatné dle aktuálně použitého slovníku (jazyka) pro kontrolu. Dané hodnoty jsou zde zmíněny pouze informativně. Na následujícím obrázku jsou znázorněny pouze chybné opravy a jejich stav před a po aplikaci navržené metodiky (oranžovou barvou je znázorněn stav po aplikaci metodiky). Ze znázornění byly vynechány jazyky s jinou abecedou, protože hodnoty oprav jsou u nich rovny nule.



Obr. 9 Znázornění vlivu zvolené metodiky na opravy cizích slov

Pokud se podíváme na hodnoty chybných oprav, tak je zřejmé, že nějakého pokroku bylo dosaženo. Pokrok ale není u některých jazyků moc výrazný (např. španělština). Z toho lze usoudit, že navržený přístup není dokonalý. V aplikaci bude ponechán, protože situace, kdy bude text obsahovat více částí v různých jazycích, by měly nastávat minimálně.

Z výsledků je dále zřejmá skutečnost, že výstup je závislý na konkrétních jazycích, ve kterých jsou jednotlivé části textu. Některý jazyk může mít slova velice po-

dobná angličtině a jiný ne. Problém s detekcí nenastává v případech, kdy se v rámci textu vyskytují jazyky, které používají jinou abecedu (např. ruština či arabština). Naopak některé jazyky mohou mít slova relativně podobná. Z pohledu tohoto experimentu se jedná zejména o francouzštinu a španělštinu. Je nutné zmínit, že dané výsledky jsou velice závislé i na algoritmu korekce. V případě, že by korektor generoval kandidáty na korekci ve vzdálenosti více než jedné editace, tak se lze domnívat, že by bylo chybných oprav více (oprav slov v cizím jazyce).

5.2.4 Pohled na korekci z hlediska dolování znalostí

V cíli práce bylo zmíněno, že implementovaná aplikace bude primárně sloužit pro korekci textu před tím, než z něj budou dolovány znalosti. Z toho důvodu je nutné se podívat na přínosy korekce i z tohoto pohledu.

Existuje celá řada přístupů a algoritmů, které lze pro dolování znalostí použít. Jednou skupinou algoritmů, které jsou pro daný úkol používány, jsou rozhodovací stromy. Například lze uvést algoritmus C4.5 (bývá také nazýván J48). Horní mez výpočetní náročnosti daného algoritmu je dána vzorcem níže (15). Ze vzorce vyplývá, že je daný algoritmus závislý na počtu unikátních slov *n* kvadraticky (Quinlan, 1993). Počet dokumentů je značen *m* (počet dokumentů je u experimentu s recenzemi hotelů roven počtu recenzí).

$$O(m * n^2) \tag{15}$$

Počet unikátních slov v textu tím pádem vede k rychlému nárůstu výpočetní doby nutné k provedení dolování znalostí. Cílem korekce textu je tedy snížení počtu unikátních slov. Ke snížení by mělo dojít kvůli tomu, že špatné varianty slova budou opraveny na jednu správnou variantu slova (v ideálním případě).

Zhodnocení korekce (z hlediska dolování znalostí) bylo tím pádem provedeno pomocí výpočtu unikátních slov (dimenzí) nad textem před korekcí a po korekci. Experiment byl prováděn na recenzích hotelů v anglickém i českém jazyce. Jedná se o stejná testovací data jako v předchozích kapitolách. Identifikace slov (řetězců) probíhala tak, že za slovo byl považován každý neprázdný řetězec, který byl oddělený mezerou. Znaky, které sloužily pro organizaci textu, byly odstraněny (interpunkce). Velikost písmen byla poté sjednocena tak, aby byla všechna písmena malá.

Tab. 16 Analýza počtu unikátních slov

	Před korekcí		Po korekci	
	Počet	Podíl	Počet	Podíl
anglický jazyk	238 269	100 %	171 499	71,9 %
český jazyk	28 547	100 %	25 806	90,0 %

Ze získaných výsledků je zřejmé, že korekce měla významný vliv na počet unikátních slov. Lze tedy říci, že by výpočetní doba měla z hlediska algoritmu J48 kles-

nout. Mimo ušetření výpočetní doby by ale měl být i samotný výstup daného procesu kvalitnější. Snížení počtu dimenzí u českého jazyka není tak výrazné ze dvou důvodů. Prvním důvodem je skutečnost, že zpracovávaný počet recenzí v českém jazyce byl značně menší. Druhým důvodem je větší počet slov, která se v daném jazyce vyskytují.

5.3 Výkonnost řešení

Výkonnosti implementované aplikace byla věnována již poměrně velká část práce v souvislosti s paralelizací. Z dané části si lze udělat poměrně dobrou představu o tom, čeho je aplikace schopna. Zde bude na problematiku pohlíženo z jiného úhlu. V rámci předchozích částí byly zjištěny nějaké poznatky. Z těch poté vychází nějaká doporučení. Aplikace doporučení ale může mít vliv na výkonnost implementovaného řešení. Příkladem dané situace je navýšení slov ve slovníku, kdy větší počet slov může znamenat vyšší výpočetní dobu k vyhledání slova. Této konkrétní situaci bude věnována následující kapitola.

5.3.1 Vliv velikosti slovníku na výpočetní dobu

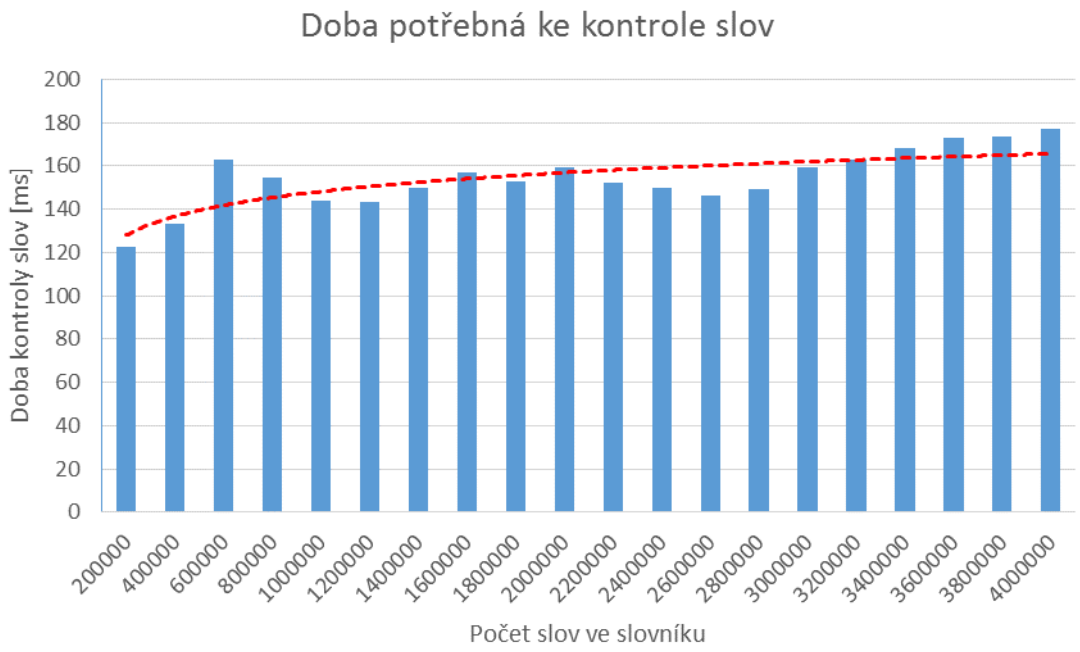
V rámci této práce bylo již několikrát zmíněno, že kvalita korekce do značné míry závisí na počtu slov v použitém slovníku. Ačkoliv by se mohlo zdát, že velikost slovníku nemůže rapidněji narůst, tak opak je pravdou. Na danou situaci se lze totiž podívat z jiného pohledu, kterým je přidání podpory nového jazyka. Pokud bude mít nějaký jazyk komplexnější strukturu (více možností koncovek slov, diakritika atd.), tak může velikost slovníku být signifikantně větší. Danou situaci dobře ilustruje srovnání českého a anglického jazyka. Anglický jazyk neobsahuje diakritiku a počet možných koncovek slov je značně menší. Naproti tomu v českém jazyce může mít jedno slovo celou řadu možných forem (skloňování, …).

Kvůli výše zmíněným důvodům bude proveden experiment, jehož cílem je zjistit jaký vliv má velikost slovníku na výpočetní dobu potřebnou k identifikaci chyb v textu. Daný experiment bude prováděn na slovech v českém jazyce. Český jazyk byl zvolen kvůli dvěma důvodům. Prvním z nich je skutečnost, že díky diakritice je zde větší počet možných znaků. To znamená, že v navržené struktuře pro reprezentaci slovníku (kapitola 4.6.1) může být v jedné úrovni větší počet položek (znaků). Díky tomu, že je nutné procházet více znaků, by tím pádem měl být výsledek experimentu blíže nejhoršímu možnému případu. Anglický jazyk naopak diakritiku neobsahuje, z čehož vyplývá, že počet možných znaků (abeceda) je menší. Díky tomu by mělo být vyhledávání při stejném počtu slov o něco rychlejší. Druhým důvodem je skutečnost, že na českém jazyce lze díky většímu počtu slov situaci lépe otestovat.

Samotná realizace experimentu probíhala tak, že z českého slovníku, který využívá program Aspell (Atkinson, 2004), bylo vygenerováno několik menších slovníků. Dané slovníky obsahovaly různý počet slov. Z každého slovníku bylo poté vybráno náhodně 30 000 slov. Tato slova byla poté použita spolu s 20 000 slovy

vygenerovanými z anglického slovníku k testování. Slova z anglického slovníku suplovala skupinu řetězců, které nelze v českém slovníku najít.

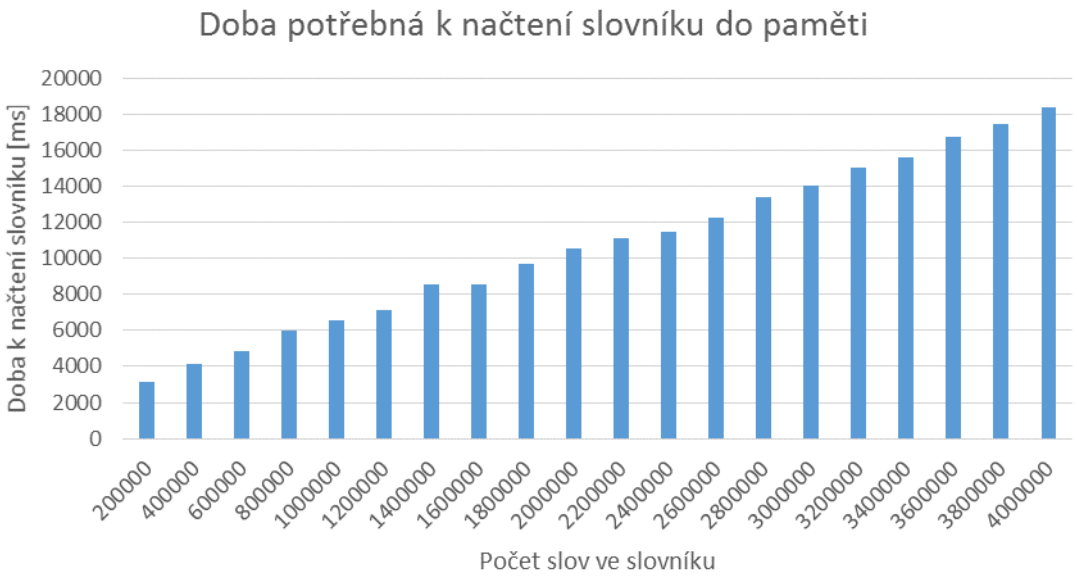
Výsledky byly následně získány tak, že každý vygenerovaný slovník byl použit ke kontrole, zda jsou slova z testovací množiny obsažena ve slovníku. Měření s jednotlivými slovníky bylo opakováno vždy třikrát, aby byl minimalizován vliv aktuálního vytížení počítače. Výsledné hodnoty z každého měření byly následně zprůměrovány.



Obr. 10 Závislost doby nutné k vyhledání slov na velikosti slovníku

Z předchozího obrázku je zřejmé, že doba nutná pro vyhledání slova ve slovníku narůstá minimálně ve srovnání s nárůstem počtu slov. Na použitých testovacích datech lze považovat danou závislost za logaritmickou (viz spojnice trendu). Výkyvy od dané spojnice jsou pravděpodobně způsobeny tím, že testovací data nebyla úplně totožná pro každé měření. Na výsledky má pravděpodobně také vliv aktuální vytížení počítače (použití mezipamětí atd.).

Druhou části daného experimentu bylo zhodnocení vlivu počtu slov ve slovníku na dobu nutnou k jeho načtení do paměti. Výsledné časy jsou vidět na Obr. 11. Z grafu je zřejmé, že nárůst doby nutné k načtení slovníku je lineární. Je nutné zmínit, že daná doba pravděpodobně nezávisí výrazněji na použité struktuře pro ukládání slovníku. Nárůst potřebného času je spíše způsoben samotným načtením souboru z disku.



Obr. 11 Doba potřebná k načtení slovníku do paměti

Získané výsledky lze vztáhnout i k další komponentě korektoru, která má na starost samotnou korekci. V rámci této části jsou využívány n-gramy, ve kterých je stejně jako u slovníku nutné rychle vyhledávat. Jelikož byla pro jejich uložení v paměti použita stejná struktura, tak se lze domnívat, že i zde má počet n-gramů podobný vliv na výpočetní dobu jako v případě slovníku.

6 Diskuze

Z práce vyplývá, že problematika korekce textu je mnohem rozsáhlejší, než by se na první pohled mohlo zdát. Na samotnou problematiku korekce navíc navazuje celá řada dalších oblastí, o kterých je nutné mít alespoň základní přehled, aby bylo možné vytvořit kvalitní řešení daného problému. V rámci této části budou tím pádem diskutovány některé aspekty, které sice většinou byly v rámci práce nastíněny, ale nebyl jim věnován takový prostor.

První diskutovanou částí je podpora více jazyků. Implementované řešení je schopné provést korekci textu v různých jazycích, pokud bude mít k dispozici adekvátní slovníky. Ve skutečnosti je ale podpora do určité míry limitovaná. Ačkoliv se autor práce snažil o co největší univerzalitu řešení, tak byl limitován skutečností, že pro přidání podpory jiného jazyka je nutná alespoň jeho základní znalost. Jako příklad skutečnosti limitující podporu dalších jazyků lze uvést diakritiku. Kdyby nebyla implementována další komponenta, která je schopna doplnit diakritiku ke všem znakům slova, které ji postrádají, tak by byla korekce textu v českém jazyce méně kvalitní. Pokud by tedy jiný jazyk používal nějaký speciální přístup, který aplikace není schopna nyní řešit, tak by nebylo možné provést korekci tak kvalitně či v některých příkladech vůbec. Konkrétním příkladem je například čínština, která neobsahuje mezery mezi slovy. Text v jazycích, které jsou podobné češtině a angličtině, by měl být schopen korektor opravovat. Univerzalita z pohledu těchto jazyků je dána použitým přístupem ke korekci. Přístup je totiž postaven na statistickém pohledu na daná data (texty). Opačným přístupem by bylo poté využití znalostí daného jazyka a řešení jeho závislostí na přímo. Autor se na základě získaných znalostí z této práce ale domnívá, že daný přístup by byl značně náročnější na implementaci. Je nutné také zmínit, že by vyžadoval větší znalost konkrétního jazyka z pohledu řešitele. Přístup by také vedl k tomu, že by bylo řešení více závislé na konkrétním jazyce.

V souvislosti s podporou více jazyků je nutné zmínit, že je stále problém získání dostatečných informací pro korektor. Co je myšleno pod těmito informacemi, bylo rozebíráno již dříve, nicméně zde lze pro rekapitulaci uvést nejdůležitější typy, kterými jsou frekvence znaků, slova a n-gramů. Aby bylo možné provést korekci co nejlépe, tak je potřeba mít k dispozici dané informace co nejkvalitnější. Zde se ale nabízí otázka, co je vlastně myšleno pojmem „nejkvalitnější“. Problém formulace daného pojmu lze uvést na příkladu, kdy má korektor k dispozici velice kvalitní slovník, který je použit na korekci nějakého specifického textu. Tento text může být natolik specifický, že ani velmi rozsáhlý slovník nemusí tato slova obsahovat. Jako další situaci lze uvést korekci textu, který pochází ze sociálních sítí. Vzhledem k tomu, že dané sítě často používá mladá generace a nesou se přes ně různé trendy, tak může docházet k velkému výskytu slangových výrazů či zkratek. Nastává tedy otázka co dělat s takovými výrazy respektive s případnými chybami v nich. Mají se opravovat na normální výrazy nebo se mají opravovat na slangové výrazy? Lze se tedy domnívat, že se mohou slovníky v budoucnu nějakým způsobem fragmentovat. Ze zmíněných informací tedy vyplývá, že jakkoliv bude korekce postupně vy-

lepšována, tak stejně pravděpodobně nelze podchytit kvůli rozsáhlosti dané problematiky všechny možné situace.

Dalším poznatkem získaným při realizaci korektoru byla skutečnost, že použitý jazyk má vliv na výpočetní náročnost korekce. Daný problém je zřejmý už z toho, kolik průměrně obsahují slov slovníky anglického a českého jazyka. Slovník českého jazyka, který využívá implementovaná aplikace, obsahuje okolo 4 000 000 slov. Naproti tomu slovník anglického jazyka neobsahuje ani 300 000 slov. Situaci navíc dále komplikuje diakritika a další různé závislosti v českém jazyce. Některé z těchto závislostí (shoda přísudku s podmětem) navíc nelze zvoleným přístupem v podstatě vůbec řešit. Autor této práce se domnívá, že skutečnost, že angličtinu lze lépe zpracovávat pomocí počítače, může být v určité paralele s její rozšířeností ve světě.

Při implementaci korektoru několikrát nastala situace, kdy nutné bylo z nějakého důvodu zvážit vliv implementované funkcionality na celkovou výpočetní dobu nutnou pro korekci textu. Důvodem byl fakt, že jedním z požadavků na řešení byla schopnost korekce velkého množství textu. Kdyby tedy nebylo řešení optimalizováno pro daný problém, tak by nemuselo být v praxi moc použitelné, protože doba potřebná pro korekci by byla moc dlouhá. Jako příklad konkrétního kompromisu lze uvést generování kandidátů. V daném případě byl zvolen přístup, kdy budou kandidáti generováni pouze do vzdálenosti jedné editace. Tento přístup má v konečném důsledku vliv na výslednou přesnost korekce. Ukazuje, že ve skutečnosti by bylo vhodnější zvýšit možnou vzdálenost až na dvě editace a ponechat uživateli možnost nastavit si požadovanou přesnost. Jako další kompromis lze uvést implementaci korekce slov, která jsou ve slovníku, ale mohou být špatně použita. V daném případě by musela být kontrolována všechna slova ve větě, což by zvyšovalo razantně výpočetní náročnost. Proti implementaci dané funkcionality dále hovořila i skutečnost, že pro korekci daných chyb je nutné mít k dispozici velice kvalitní statistické informace o n-gramech (informace by navíc byly potřebné pro každý jazyk). V případě tohoto druhého příkladu je tedy zřejmé, že rozhodnutí neimplementovat daný přístup bylo správné.

Na problematiku korekce textu a kompromisů kvůli výkonu lze dále navázat optimalizací výkonu aplikace pomocí paralelizace. Již v samotné práci byla daná problematika rozebírána a bylo zmiňováno, že lze korekci textu velice dobře paralelizovat. Daná skutečnost je tedy částečně v rozporu s problémem zmíněným v předchozím odstavci. Zde se totiž nabízí otázka, proč neprovádět kontrolu všech slov, když lze docílit značného zrychlení pomocí paralelizace a minimalizovat tak dopad rozšířené kontroly na výpočetní dobu. Implementované řešení je limitováno dostupnými statistickými informacemi, které je nutné mít pro korekci. Jinak lze ale říci, že kontrolu všech slov lze právě díky paralelizaci poměrně dobře realizovat i nad velkým objemem dat. Korekce textu by tím pádem neměla být v budoucnu, i kdyby byl objem textových dat ve světě stále větší, problémem. Na základě získaných zkušeností je ale nutné zmínit, že efektivní implementace paralelizace je poměrně komplikovaná.

Z výše zmíněných informací vyplývá, že implementovanou aplikaci lze dále rozšířit. Rozšíření lze rozdělit do několika skupin. První skupinou je podpora nových jazyků. Pokud je nový jazyk podobný již implementovaným jazykům a nemá zvláštní specifika, tak jej lze přidat pouze pomocí vytvoření nového balíku slovníku. Pokud ale obsahuje jazyk nějaká specifika, jako například zmiňovaná čínština, tak bude nutné provést zásah přímo do aplikace. Aplikaci lze dále také rozšířit o nějaké techniky, které by byly schopny řešit některé problémy, které aplikace momentálně není schopna řešit (shoda přísudku s podmětem). Druhou skupinou možných rozšíření je zkvalitnění současných slovníků (větší počet n-gramů apod.). Třetí skupinou jsou dílčí vylepšení aplikace, ze kterých lze uvést lepší segmentaci věty do slov. Nyní je například tečka považována za konec věty (kontextu), ve skutečnosti ale může být součástí nějaké zkratky, což povede k špatnému rozdělení věty.

V souvislosti s řešenou problematikou korekce textu je nutné zmínit, že získané poznatky nabyté během realizace této práce, poskytují autorovi určitý náhled do problematiky zpracování přirozeného jazyka. Některé přístupy jsou totiž využívány v různých dalších technologiích. Jedná se například o přístup k práci s jazykem za pomocí statistických informací. Problematika zpracování přirozeného jazyka se dále zabývá například převodem mluveného slova do textu. Daná oblast navíc v poslední době zažívá renesanci, která je způsobena vznikem hlasových asistentů v chytrých telefonech. Jako konkrétní příklady lze uvést Siri v iPhone či Cortanu na platformě Windows Phone. Autor se navíc na základě nabytých znalostí domnívá, že korekce textu hraje ve hlasovém ovládání velkou roli, protože je pravděpodobně využívána pro dodatečnou korekci textu syntetizovaného z hlasového vstupu.

Na závěr diskuze je nutné zmínit skutečnost, že korekce textu může mít i svou stinnou stránku. Autor práce se domnívá, že korekce textu může vést k určité deformaci člověka. Ten si na ni totiž může natolik zvyknout, že již nebude schopen rozeznat slova s chybou od správných slov. Jako příklad dané situace lze uvést zadávání dotazů do vyhledávačů. Pokud je vyhledávaný výraz zadán špatně, tak to vyhledávač rozpozná a v některých situacích dokonce rovnou zobrazí výsledky pro správný výraz. Autor práce na sobě zpozoroval, že v některých případech si uvědomí, že zadal výraz špatně, nicméně neopraví jej, protože počítá s tím, že to za něj udělá vyhledávač.

7 Závěr

V rámci práce byl v souladu s navrženými cíli implementován nástroj, který dokáže provádět korekci textu bez toho, aby uživatel musel vybírat vhodná slova ze seznamu kandidátů na korekci. Implementace nástroje je realizována na základě informací zmíněných v literárním přehledu.

Daný nástroj také splňuje požadavek na to, aby byl schopný provádět korekci velmi rozsáhlých textových souborů (v řádů stovek MB, GB). Zde je nutné zmínit, že právě velikost vstupů vedla k tomu, že implementovaný nástroj musí být schopný automatické korekce. Kvůli rozsáhlosti vstupů byla také implementovaná funkcionalita, která uloží informace o provedených korekcích do souborů tak, aby bylo možné vždy provedenou korekci zhodnotit.

Dílčí částí byla příprava slovníků pro korekci textu v českém a anglickém jazyce. Aplikace musela být navržena tak, aby byla na použitém slovníku co nejméně závislá. Slovník neobsahuje pouze možná slova, ale i další informace o frekvencích výskytů slov, slovních spojení a znaků v rámci daného jazyka. Požadavek na to, aby byla aplikace schopna korekce textu v českém a anglickém jazyce, byl tím pádem splněn, navíc byla díky navrženému přístupu ponechána možnost přidání dalších jazyků později.

Nakonec bylo provedeno několik experimentů, které prověřují schopnosti implementovaného nástroje na korekci textu na reálných datech v českém i anglickém jazyce. Dané experimenty dokazují, že zvolený přístup dokáže provést korekci nalezených chyb s poměrně velkou přesností. V rámci experimentů bylo také prokázáno, že je vhodné použít automatický korektor v rámci procesu přípravy textu, předtím než z něj budou dolovány znalosti.

Autor se na základě výše zmíněných informací domnívá, že nástroj splňuje všechny stanovené požadavky a lze jej použít v praxi.

8 Literatura

AMDAHL, Gene M. Validity of the single processor approach to achieving large scale computing capabilities. In: *Proceedings of the April 18-20, 1967, spring joint computer conference.* ACM, 1967. p. 483-485.

ATKINSON, Kevin. *GNU Aspell* [online]. 2004 [cit. 2014-05-03]. Dostupné z: http://aspell.net/

CISCO. *Cisco Visual Networking Index: Forecast and Methodology*, 2012–2017. 2013. Dostupné z: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf

CVRČEK, Václav. Korpus. *Manuál práce s ČNK* [online]. 2013 [cit. 2014-05-03]. Dostupné z: http://wiki.korpus.cz/doku.php/pojmy:korpus

Český národní korpus: Srovnávací frekvenční seznamy. Ústav Českého národního korpusu FF UK, Praha 2010. Dostupné z WWW: http://ucnk.ff.cuni.cz/srovnani10.php

CHRISTENSEN, Hans. *HC Corpora* [online]. 2010, 2013 [cit. 2014-05-03]. Dostupné z: http://www.corpora.heliohost.org/

DAŘENA, František. Soubor recenzí uživatelů ubytovacích služeb poskytovaných on-line prostřednictvím *booking.com*, http://www.booking.com. ÚI PEF Mendelova universita v Brně, 2010.

BRILL, Eric; MOORE, Robert C. An improved error model for noisy channel spelling correction. In: *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics.* Association for Computational Linguistics, 2000. p. 286-293.

DAMERAU, Fred J. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 1964, 7.3: 171-176.

DEOROWICZ, Sebastian; CIURA, Marcin G. Correcting spelling errors by modelling their causes. *International journal of applied mathematics and computer science*, 2005, 15: 275-285.

FOSLER-LUSSIER, Eric. Markov Models and Hidden Markov Models: A Brief Tutorial. *International Computer Science Institute Technical Report TR-98-041*, 1998.

FOWLER, Martin. Inversion of control containers and the dependency injection pattern. 2004.

FREDKIN, Edward. Trie memory. *Communications of the ACM*, 1960, 3.9: 490-499.

GALE, William; SAMPSON, Geoffrey. Good-Turing smoothing without tears. *Journal of Quantitative Linguistics*, 1995, 2.3: 217-237.

GEORGE E.P. BOX, George E.P.George C. *Bayesian inference in statistical analysis*. Wiley classics library ed. New York: Wiley, 1992. ISBN 04-715-7428-7.

GERMANN, Ulrich; JOANIS, Eric; LARKIN, Samuel. Tightly packed tries: How to fit large models into memory, and make them load fast, too. In: *Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing*. Association for Computational Linguistics, 2009. p. 31-39.

GUNDERLOY, Mike. *Z kodéra vývojářem: nástroje a techniky pro opravdové programátory*. Vyd. 1. Překlad Miroslav Müller. Brno: Computer Press, 2007, 287 s. ISBN 978-80-251-1517-6.

HAMMING, Richard W. Error detecting and error correcting codes. *Bell System technical journal*, 1950, 29.2: 147-160.

HIRSCHBERG, Daniel S. Algorithms for the longest common subsequence problem. *Journal of the ACM (JACM),* 1977, 24.4: 664-675.

CHEN, Stanley F.; GOODMAN, Joshua. An empirical study of smoothing techniques for language modeling. In: *Proceedings of the 34th annual meeting on Association for Computational Linguistics.* Association for Computational Linguistics, 1996. p. 310-318.

JURAFSKY, Daniel. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition.* 2nd ed. Upper Saddle River: Prentice Hall, 2009, 1024 s. ISBN 01-350-4196-1.

KERNIGHAN, Mark D.; CHURCH, Kenneth W.; GALE, William A. A spelling correction program based on a noisy channel model. In: *Proceedings of the 13th conference on Computational linguistics-Volume 2*. Association for Computational Linguistics, 1990. p. 205-210.

KNUTH, Donald E. Structured Programming with go to Statements. *ACM Computing Surveys (CSUR),* 1974, 6.4: 261-301.

KUKICH, Karen. Techniques for automatically correcting words in text. *ACM Computing Surveys (CSUR)*, 1992, 24.4: 377-439.

MARR, Deborah T., et al. Hyper-Threading Technology Architecture and Microarchitecture. *Intel Technology Journal*, 2002, 6.1.

MICROSOFT. NET Framework 4.5. *Microsoft developer network* [online]. 2014 Dostupné z: http://msdn.microsoft.com/cs-cz/library/w0x726c2%28v=vs.110%29.aspx

MITCHELL, Tom. Twenty Newsgroups Data Set. *UCI Machine Learning Repository* [online]. 1999. Dostupné z: http://archive.ics.uci.edu/ml/datasets/Twenty+Newsgroups

NORVIG, Peter. How to write a spelling corrector. *Peter Norvig* [online]. 2007 [cit. 2014-05-02]. Dostupné z: http://norvig.com/spell-correct.html

POLLOCK, Joseph J.; ZAMORA, Antonio. Automatic spelling correction in scientific and scholarly text. *Communications of the ACM*, 1984, 27.4: 358-368.

QUINLAN, John Ross. *C4. 5: programs for machine learning*. Morgan kaufmann, 1993.

RISTAD, Eric Sven; YIANILOS, Peter N. Learning string-edit distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 1998, 20.5: 522-532.

RUSSOM, Philip. Big data analytics. *TDWI Best Practices Report, Fourth Quarter*, 2011.

ŘEHŮŘEK, Radim; KOLKUS, Milan. Language identification on the web: Extending the dictionary method. In: *Computational Linguistics and Intelligent Text Processing*. Springer Berlin Heidelberg, 2009. p. 357-368.

SHANNON, Claude Elwood. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 2001, 5.1: 3-55.

TOUTANOVA, Kristina; MOORE, Robert C. Pronunciation modeling for improved spelling correction. In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics.* Association for Computational Linguistics, 2002. p. 144-151.

VERBERNE, Suzan. Context-sensitive spell checking based on word trigram probabilities., University of Nijmegen, 2002.

WANG, Peiling; BERRY, Michael W.; YANG, Yiheng. Mining longitudinal Web queries: Trends and patterns. *Journal of the American Society for Information Science and Technology*, 2003, 54.8: 743-758.

WHITELAW, Casey, et al. Using the web for language independent spellchecking and autocorrection. In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*. Association for Computational Linguistics, 2009. p. 890-899.

ZEUMER, Carsten. Detect Encoding for In- and Outgoing Text. CODEPROJECT. *CodeProject* [online]. 2007 [cit. 2014-05-03]. Dostupné z: http://www.codeproject.com/Articles/17201/Detect-Encoding-for-In-and-Outgoing-Text

Přílohy

A XML popis balení slovníku

```
<?xml version="1.0" encoding="UTF-8"?>
<Dictionary locale="cs_CZ" version="1.0"
alphabet="abcdefghijklmnopqrstuvwxyzáéíóúýčďěňřšťžů"
allowedSpecialChars="" wordBoundaryRegex="[a-záéíóúýčďěňřšťžů]+"
accentPairs="a-á,e-é,i-í,o-ó,u-ú,y-ý,c-č,d-ď,e-ě,n-ň,r-ř,s-š,t-
ť,z-ž,u-ů">
      <File type="Dictionary">
            cs_CZ.dic
      </File>
      <File type="Affix">
            cs_CZ.aff
      </File>
      <File type="OneCharFrequences">
            oneCharFreq.txt
      </File>
      <File type="TwoCharFrequences">
            twoCharFreq.txt
      </File>
      <File type="DeletionsMatrix">
            del.txt
      </File>
      <File type="InsertionsMatrix">
            ins.txt
      </File>
      <File type="TranspositionsMatrix">
            trn.txt
      </File>
      <File type="SubstitutionsMatrix">
            sub.txt
      </File>
      <File type="UnigramFrequences">
            unigrams.txt
      </File>
</Dictionary>
```