VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

# KONTROLA PRAVOPISU V ČESKÝCH TEXTECH

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

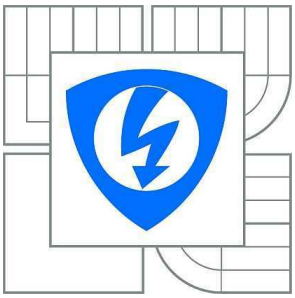
AUTOR PRÁCE
AUTHOR

Bc. STANISLAV BUREŠ

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

# KONTROLA PRAVOPISU V ČESKÝCH TEXTECH

SPELLING CHECK IN THE CZECH TEXTS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

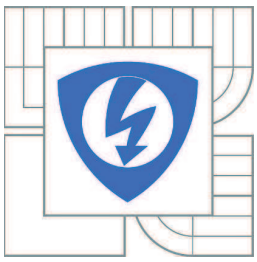
AUTOR PRÁCE — Bc. STANISLAV BUREŠ

AUTHOR

VEDOUCÍ PRÁCE — Ing. LUCIE FOJTOVÁ

SUPERVISOR

BRNO 2011



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

**Fakulta elektrotechniky a komunikačních technologií**

**Ústav telekomunikací**

# Diplomová práce

magisterský navazující studijní obor
**Telekomunikační a informační technika**

| | | | |
|---|---|---|---|
| ***Student:*** | Bc. Stanislav Bureš | ***ID:*** | 83889 |
| ***Ročník:*** | 2 | ***Akademický rok:*** | 2010/2011 |

**NÁZEV TÉMATU:**

## Kontrola pravopisu v českých textech

**POKYNY PRO VYPRACOVÁNÍ:**

Prostudujte a stručně popište problematiku kontroly pravopisu se zaměřením na česky psané texty. Na tomto základě navrhněte algoritmus, který umožní v psaných textech identifikovat a opravit pravopisné a jiné chyby (překlepy, chybějící diakritika atd.). Algoritmus implementujte v některém z programovacích jazyků (Java, C++) a ověřte jeho funkčnost na reálných datech.

**DOPORUČENÁ LITERATURA:**

[1] Mitton, R.: Ordering the suggestions of a spellchecker without using context. Natural Language Engineering, [online], Dostupné z www: https://eprints.bbk.ac.uk/782/1/mitton_pre-pub.pdf
[2] FISCHER, H. Soundex: Algorithm for phonetic search, Alternative to Soundex. Sound-ex. [online]. Dostupné z www:
http://www.sound-ex.com/alternative_zu_soundex.htm.

| | | | |
|---|---|---|---|
| ***Termín zadání:*** | 7.2.2011 | ***Termín odevzdání:*** | 26.5.2011 |

***Vedoucí práce:*** Ing. Lucie Fojtová

**prof. Ing. Kamil Vrba, CSc.**
*Předseda oborové rady*

**UPOZORNĚNÍ:**

ABSTRAKT

Diplomová práce se zabývá problematikou kontroly pravopisu v česky psaných textech. Dále práce obsahuje přehled nejpoužívanějších fonetických algoritmů včetně jejich vlastností a vybrané metrické metody sloužící k určení podobnosti dvou slov. Další část práce se zabývá implementací daných metod do kontroloru pravopisu a demonstrací efektivity jejich využití v českém textu. Poslední část práce se zabývá návrhem algoritmu, který na základě kontextu věty opravuje pravopisné chyby v textech.

Klíčová slova

Soundex, Metaphone, Daitch–Mokotoff, Levenshtein, Jazzy, Aspell, trigram

ABSTRACT

The Master‘s thesis deals with spell checking in the czech texts. It also contains an overview of the most used phonetic algorithms, including their properties and it deals with focus on metric methods, which are used to compare two words. The second part of this thesis deals with implementation of selected algorithms to the spell checker software and demonstration its spell - checking function in czech texts. The last part of this thesis deals about building context – sensitive algorithm, which is performs text correction.

Keywords

Soundex, Metaphone, Daitch–Mokotoff, Levenshtein, Jazzy, Aspell, trigram

BUREŠ, S. *Kontrola pravopisu v českých textech.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 46 s. Vedoucí semestrální práce Ing. Lucie Fojtová.

PROHLÁŠENÍ

Prohlašuji, že svou semestrální práci na téma „Kontrola pravopisu v českých textech“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této semestrální práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujícího autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ……..

……
**Bc. Stanislav Bureš**

PODĚKOVÁNÍ

Děkuji vedoucímu práce ing. Lucii Fojtové za velmi užitečnou metodickou pomoc a cenné rady při zpracování diplomové práce.

# Obsah

# Úvod

V dnešní době se rozmáhá písemná komunikace elektronickou formou (e-mail, chat,…). Komunikují tak mezi sebou lidé nebo společnosti. Při takové komunikaci však může docházet k překlepům v psaném textu nebo k neúmyslnému nedodržování pravopisu. Aby se těmto chybám předcházelo, tak existují programy pro kontrolu pravopisu, které daný text zkontrolují a označí v něm chybná slova, ke kterým nabídnou návrhy na opravu. Takové programy fungují buď samostatně, nebo jsou, jako ve většině případů, součástí jiných programů (internetový prohlížeč, textový editor, IM klient, …).

Obsahem této diplomové práce je náhled na problematiku kontroly pravopisu v českých textech doplněný o přehled nejpoužívanějších algoritmů a metod určených k tomuto účelu. Jde o nezbytný základ pro další vývoj práce směrem k vytvoření kontroloru pravopisu pro český jazyk, který bude automaticky, tj. bez zásahu uživatele, opravovat chyby v textech. Tento program bude následně možné integrovat do aplikací psaných v jazyce JAVA.

První část práce se zabývá nejčastěji používanými algoritmy pro kontrolu pravopisu, jejich historií, popisem funkce a možností jejich implementace v kontroloru pravopisu česky psaných textů.

Druhá část této práce se zabývá praktickou ukázkou funkce jednoho z algoritmů na testovacím programu pro opravu česky psaných textů. Jedná se o demonstrační program, který má za úkol předvést možnosti algoritmu pro využití v českém jazyce.

Třetí část této práce se zabývá návrhem a implementací algoritmu pro automatickou opravu textu. Je zde popsán návrh a funkce tohoto algoritmu včetně testů výkonnosti.

# 1 Pravopisné chyby

Pravopisné chyby se dnes vyskytují ve velké skupině textů. Jedná se nejen o soukromou korespondenci, ve které se komunikuje ve velké míře hovorovou češtinou, ale také se chyby vyskytují v obchodní poště, článcích na Internetu a dokonce i ve veřejných publikacích a dokumentech. Existuje několik druhů chyb. Kapitola 1.1 pojednává o klasifikaci pravopisných chyb. Kapitola 1.2 pojednává o nejčastěji se vyskytujících chybách.

## 1.1 Klasifikace pravopisných chyb

Pravopisné chyby se dělí hned na několik druhů. Tyto druhy jsou včetně příkladů popsány níže [6]:

1. pravopisné
2. lexikální
3. hláskoslovné
4. morfologické
5. syntaktické
6. ve slovosledu
7. v úzu (tj. nevhodné vyjadřování)/ stylistické
8. makarónský text
9. zcela rozvrácený text

ad 1) Pravopisné chyby – jedná se o chyby vzniklé buď:

- nevhodným psaním i/y nebo s/z – například: *spíval*, *vivinul*
- nesprávně psanou palatalizací – například: *děťi*, *skříňi*
- neutralizací znělosti na konci slova – například: *led*/*let*
- psaním velkých a malých písmen – například: *praha je naše hlavní město*
- chybou v interpunkci – například: *Obloha byla zatažená* [,] *ale nepršelo*.

ad2) Lexikální chyby

- existující, ale nevhodně užité slovo – například: *je to velmi hebce* (hebké)
- neexistující slovo – například: *Myslel jsem, že mě bude kritikovat*.

ad3) Hláskoslovné chyby

- jiná hláska (délka vokálů, palatály) projevující se chybnou diakritikou – například: *mesíc*, *údolí*
- metateze grafémů – například: *ponměnka*
- chybějící nebo přebývající grafém – například: *právě* [j]*ím olivy, jedi*[n]*ná*
- jiná chyba ve slově

ad4) Morfologické chyby – jedná se o chyby vzniklé buď:

- ve flexi – například: *Sedím v baru a popíjí*[m] *drink*.
- v derivační morfologii – například užití nevhodné předpony
- obecně -česká morfologie – například: *černýho, bílej*

ad5) Syntaktické chyby – jedná se o chyby buď:

- morfosyntaktické – například: Myslím *se* (si)*, že je dobrý nápad, naučit* [se] *na zkoušku*.
- ve shodě – například: *To byl ale špatná nápad*.
- ve vazbách – například: *Jdu ven s děti*.
- vynechané/nadbytečné slovo/slova narušující/rozvracející syntax – například: *Mám radost* [že] *je venku hezky*.
- jiné – například: *Jsem jdu se psem ven*.

ad6) Ve slovosledu

- negramatický slovosled – například: *Se divím, že to tak dopadlo*.
- gramatický, ale nevhodný slovosled – například: *Sklenice stojí taky na stole*. (Na stole stojí taky sklenice.)

ad7) V úzu

- nevhodné vyjadřování – například: *Vídeň je hlavní město z Rakouska*.
  *Pracuji v domě*. (Pracuji doma)
- stylistické / aktuální členění – například: *Na stole jsou propiska a papír*.

Ad8) Makarónský text

- kombinace vícejazyčných výrazů – například: *Check this článek*.

Ad9) Zcela zvrácený text

- věty bez smyslu – například: *Dnes ráno bylo pes přestal štěkat*.

## 1.2 Nejběžnější chyby

Chyby, se kterými se můžeme v textech setkat, se liší typem daného dokumentu, resp. média kde se text vyskytuje. Nejvíce druhů chyb se vyskytuje především v internetové komunikaci pomocí programů jako je ICQ, Skype nebo Facebook. Jde především o morfologické chyby a o chyby ve slovosledu. Tyto chyby jsou následovány lexikálními chybami, protože se zde často užívá zkratek (například *LOL*, *ROFL*, *OMG*), které pochází z anglických výrazů.

V obchodní komunikaci se můžeme setkat především s chybami jako je chybějící interpunkce, chybějící diakritika nebo makarónský text.

V internetových médiích se mohou vyskytovat různé počeštěné názvy, které nelze nalézt ve slovníku spisovných slov (lexikální chyby). Další častou chybou, vyskytující se v internetových médiích, je gramatický, ale nevhodný slovosled.

Další skupinou je komunikace s cizinci. Zde nejčastěji vzniká kombinace více chyb, jako je makarónský text, chyby v slovosledu, v úzu a lexikální chyby. Je to z důvodu jejich nedostatečné znalosti češtiny, kterou tak kombinují se svou rodnou řečí.

Poslední skupinou jsou lidé s dysgrafií, u kterých vznikají především pravopisné chyby v psaní i/y a s/z.

# 2 Algoritmy

## 2.1 Fonetické algoritmy

Fonetické algoritmy jsou algoritmy, které porovnávají slova na základě jejich znělosti. Většina těchto algoritmů byla vyvinuta pro anglický jazyk, a proto nemusí vykazovat správné výsledky, pokud se použijí pro jiný jazyk, než anglický.

### 2.1.1 Soundex algoritmus

Jedná se o nejznámější fonetický algoritmus, který porovnává slova podle znělosti. Často bývá nesprávně uváděn jako synonymum k fonetickému algoritmu. Cílem tohoto algoritmu je, aby homofony byly zakódovány do stejných reprezentací tak, aby mohly být rozeznány navzdory minimálním rozdílům ve slovech [5]. Soundex kóduje pouze souhlásky. Samohlásky vynechává, pokud se nejedná o první písmeno ve slově.

#### *2.1.1.1 Historie*

Soundex byl vyvinut Roberetem C. Russellem a Margaret K. Odellovou. Následně byl patentován v letech 1918 a 1922. Jeho varianta nazývaná Americký Soundex byla použita v roce 1930 pro zpětné sčítání lidu pro léta od 1890 po 1920. Do podvědomí se kód Soundexu dostal v 60. letech 20. století, kdy se stal jedním z hlavních témat měsíčníku *Journal of the Association for Computing Machinery* a déle když byl popsán v publikaci *The Art of Computer Programming* [7] Donalda Knuthse.

Nezávislá agentura NARA (National Archives and Records Administration) udržuje současné znění pravidel pro oficiální implementaci Soundexu použitého americkou vládou.

#### *2.1.1.2 Pravidla kódování*

Soundex kód se skládá z jednoho písmene a tří číslic. Písmeno v kódu je prvním písmenem slova a v číslicích jsou zakódovány zbývající souhlásky, vyjma souhlásek h a w. Kódování slov probíhá v následujícím schématu:

1. První písmeno ve slově se ponechá a nebude zakódováno.
2. Ze slova se odstraní samohlásky a souhlásky h a w.
3. Samohlásky následující první písmeno se zakódují následovně [10]:
    - Jako 1 pro b, f, p, v
    - Jako 2 pro c, g, j, k, q, s, x, z
    - Jako 3 pro d, t
    - Jako 4 pro l
    - Jako 5 pro m, n
    - Jako 6 pro r

Dvě sousední samohlásky zakódované stejným číslem se kódují jako jedno číslo. Platí to i v případech, když jsou tyto samohlásky odděleny písmeny ‘h’ a ‘w’.

4. Kóduje se do dosažení tří číslic. Pokud není co kódovat, tak se doplní 0.

Podle výše uvedeného můžeme např. slovo *Mahagon* zakódovat jako M250.

*Postup:* Mahagon – Mhgn – Mgn – (*M* zůstává, *g* se zakóduje jako 2, *n* se zakóduje jako 5 a trojici znaků kódu doplníme nulou, aby byl kód úplný.)

#### 2.1.1.3 Soundex a kontrola pravopisu

Pro použití v kontrole pravopisu není Soundex vhodně vybaven, protože dvě odlišně znějící slova mohou být stejně zakódována (viz kap. 2.1.1.4). To je dáno tím, že jsou ve stejné skupině slova, která znějí podobně, nikoliv identicky. Tato vlastnost může být užitečná například pro rozpoznání jmen vyslovených v různých akcentech. Pro kontrolu pravopisu však Soundex vykazuje mnoho návrhů pro opravu nespisovného slova. Toto má na svědomí způsob kódování, který Soundex používá, kdy jsou slova kódována pouze do 4 znaků a tak dochází k ignorování konců dlouhých slov.

#### 2.1.1.4 Problém s homofony

Stejně jako mohou mít různě znějící slova stejný soundex kód, tak může dojít i k opačné situaci, kdy stejně znějící slova mají různý kód. Pro anglický jazyk se jedná například o slova *Thompson*, kdy se samohláska p nečte, a slovo *Thomson*. Tato dvě stejně znějící slova mají různé kódy (T512 pro Thompson a T525 pro slovo Thomson). Jedná se také o případ, kdy změna počátečního písmene ve slově nezmění jeho výslovnost, jako například slova Karr (K600) a Carr (C600).

Soundex je jedním z algoritmů, který je v základní podobě určen pro anglický jazyk. Ale ani pro tento jazyk není dostačující díky nedostačující znalosti pravidel anglického pravopisu a proto není příliš vhodný pro kontrolu pravopisu. Toto dalo vzniknout variantě náhradě, za Soundex zvanou Metaphone, která je popsána v kapitole 2.3.

### 2.1.2 Daitch – Mokotoff algoritmus

Jedná se o upravený Americký Soundex algoritmus zaměřený na lepší výsledky v hledání slovanských a židovských jmen s podobnou výslovností, ale s odlišným hláskováním. Je také známý pod názvy *Jewish soundex* (židovský soundex) nebo *Eastern Europe Soundex* (východoevropský soundex), ke kterým se však autoři nehlásí, protože tento algoritmus nemá nic společného s etnickými či geografickými skupinami.

#### 2.1.2.1 Historie

Daitch – Mokotoff algoritmus, známý také jako D – M algoritmus, vznikl v roce 1985 [2]. Byl vyvinut židovskými genealogy Garym Mokotoffem a Randym Daitchem. Autoři vytvořili seznam 28,000 obyvatel, kteří legálně změnili svá jména, když žili v letech 1921 – 1948 v Palestině. Mnoho z těchto jmen bylo židovských s germánským nebo slovanským příjmením. V seznamu se vyskytovaly i různé pravopisné varianty téhož příjmení a měl být použit Soundex algoritmus k jeho prohledávání. Avšak při použití Soundex algoritmu mnoho východoevropských jmen, která zněla stejně, mělo odlišný kód. Nejčastěji šlo o stejně znějící slova s písmeny *v* a *w*, například *Moskowitz* a *Moskovitz*.

První návrh modifikace Amerického soundexu byl uveřejněn v prvním čísl*e* *Avotaynu,* časopise o židovské genealogii. Článek nesl název *Proposal for a Jewish Soundex Code*. Tento článek se stal základem pro nový systém, jehož autorem je Randy Ditch. Systém vznikl rozšířením pravidel Amerického soundexu a byl publikován o rok později ve stejném časopise v článku *The Jewish Soundex: A Revised Format*. Tento systém se stal známým pod názvem Ditch – Mokotoff Soundex System, podle jeho autorů.

D – M Soundex se stal standardem ve všech židovských genealogických organizacích. Dále byl přijat sociální organizací HIAS (Hebrew Immigrant Aid Society) a je používán v muzeu holocaustu ve Washingtonu, DC a dále k prohledávání v Ellis Island databázi 24 miliónů emigrantů.

#### 2.1.2.2 Pravidla kódování

Hlavní vylepšení, které s sebou D – M Soundex System nese oproti Americkému Soundexu jsou:

- Informace je kódována v šesti znacích, oproti čtyřem.
- První znak slova je také zakódován, místo jeho ponechání.
- Dvě po sobě jdoucí písmena, která znějí stejně, jsou kódovány jednou číslicí.
- Když písmeno, nebo kombinace více písmen, může mít jiné znění, tak je každé písmeno kódováno do jedné číslice.

Kódovací pravidla pro D – M Soundex System jsou následující [4]:

1. Slova jsou kódována do 6 číslic podle tabulky pravidel.
2. Písmena A, E, I, O, U, Y a J se vždy kódují, pokud jsou prvním písmenem ve slově. V ostatních případech se ignorují, vyjma případu, kdy dvě z těchto písmen netvoří pár, který předchází samohlásce. Písmeno H se kóduje jen v případech, kdy jím začíná slovo, nebo kdy se ve slově vyskytuje před samohláskou. V ostatních případech je ignorováno.
3. Pokud lze ve slově pospojovat sousední písmena tak, aby na nich ležel větší přízvuk, potom se tak i zakódují. Pro přiklad slovo *Chernowitz* se místo *Chernowi-t-z* (496734) zakóduje jako *Chernowi-tz* (496740).
4. Pokud dvě sousedící písmena mají stejné kódovací číslo, potom jsou obě zakódována jako jedno písmeno. Výjimku tvoří kombinace písmen „MN“ a „NM“, kde jsou obě písmena kódována zvlášť, nikoliv dohromady jako jeden znak, jako například ve slově *Kleinman* které je kódováno jako 5866600 a nikoliv jako 586600.

5. Pokud je jméno složeno z více než jednoho slova, tak se slova spojí a jsou kódována jako jedno slovo. Například *Nowy Targ* je kódován jako *Nowytarg*.
6. Písmena C, J a kombinace písmen CH, CK a RZ představují problém, protože mohou znít v určitých situacích odlišně. A tak jim jsou přiřazeny vždy dvě různé možnosti zakódování a je vždy nutné obě možnosti prověřit.
7. Pokud kódované slovo obsahuje méně než 6 znaků k zakódování, potom se jako zbývající číslice doplní nula do počtu šesti znaků kódu.

Příklad kódování některých jmen D – H Soundexem a jeho srovnání s kódováním Amerického Soundexu je uveden v tabulce 2.1 [2].

| Příjmení | Americký Soundex | D–M Soundex |
|---|---|---|
| Peters | P362 | 739400, 734000 |
| Peterson | P362 | 739460, 734600 |
| Moskowitz | M232 | 645740 |
| Moskovitz | M213 | 645740 |
| Auerbach | A612 | 097500, 097400 |
| Uhrbach | U612 | 097500, 097400 |
| Jackson | J250 | 154600, 454600, 145460, 445460 |
| Jackson-Jackson | J252 | 154664, 454664, 145466, 445466, 154646, 454646, 145464, 445464 |

**Tab. 2.1: Příklad kódování některých jmen a srovnání s kódováním Amerického Soundexu**

#### *2.1.2.3 Kontrola pravopisu*

Ačkoliv se D – M Soundex System zaměřuje na slovanská jména, není vhodný pro kontrolu pravopisu, protože stejně jako Soundex vykazuje mnoho návrhů opravy k hledanému slovu. Avšak v kombinaci s některým z metrických algoritmů (viz kap. 2.2) je lepším kandidátem na českého kontrolora pravopisu, než je Soundex.

### 2.1.3 Metaphone a Double Metaphone

Jedná se o další z často používaných fonetických algoritmů. Obsahují více pravidel pro anglickou výslovnost, než Soundex. Double Metaphone je druhou generací Metaphone algoritmu. Tato verze dosahuje přesnějších výsledků, než jeho předchůdce. Celkově je tento algoritmus lepší, než Soundex, ačkoliv má několik nedostatků ve svých pravidlech.

#### 2.1.3.1 Historie

Metaphone algoritmus byl vyvinut Lawrencem Philipsem a zveřejněn v roce 1990 [5] jako odpověď na Soundex algoritmus.

Double Metaphone byl zveřejněn v roce 2000, jako druhá generace algoritmu Metaphone. Velkou výhodou je existence tohoto algoritmu i pro jiné jazyky, než pro anglický.

#### 2.1.3.2 Vlastnosti

Metaphone produkuje kódy o variabilní délce, na rozdíl od algoritmu Soundex. Double Metaphone produkuje dva kódy – primární a sekundární.

Metaphone kód využívá ke kódování 16 souhlásek – 0, B, F, H, J, K, L, M, N, P, R, S, T, W, X, Y, kde znak 0 představuje ‘th‘, 'X' představuje "sh" nebo "ch", a ostatní znaky reprezentují svoji výslovnost v anglickém jazyce. Kódování probíhá dle tabulky pravidel, která jsou uvedena níže [5]:

- Smazat dvě stejná sousední písmena, vyjma písmene C.
- Pokud slovo začíná na KN, GN, PN, AE nebo WR, tak se smaže první písmeno.
- Pokud slovo končí MB, pak se smaže B.
- C - > X v CIA a CH; C - > S v CI, CE a CY; jinak C - > K.
- D - > J v DGE, DGY a DGI; jinak D - > T.
- Smaže se G v GH a pokud není na konci nebo před samohláskou tak v GN, GND; G - > J před I, E, Y, pokud se nejedná o GG; jinak G - > K.
- Smaže se H po samohlásce, pokud nenásleduje samohláska.
- Smaže se K po C.
- P - > F v PH.
- Q - > K.
- S - > X v SH, SIO a SIA.
- T - > X v TIA, TIO; T - > 0 v TH; T se smaže v TCH.
- V - > F.
- Ve slovech začínajících na WH smazat H; pokud nenásleduje samohláska, pak smazat W.
- Pokud slovo začíná na X, pak X - > S, jinak X - > KS.
- Smaže se Y, pokud nenásleduje samohláska.
- Z - > S.
- Samohlásky se ponechají jen v případě, kdy jimi začíná slovo.
- V ostatních případech se písmena ponechají

Příklad:

MONICA - > MONIKA - > MNK

MONIQUE - > MONIKUE - > MNK

#### *2.1.3.3 Kontrola pravopisu*

Algoritmus dosahuje velmi dobrých výsledků v prohledávání seznamů, avšak stejně jako všechny fonetické algoritmy není vhodný pro kontrolu pravopisu, protože není vybaven korekcí překlepů, které se mohou vyskytovat v textech. Bez této kontroly nemůže žádný program na kontrolu pravopisu správně fungovat, protože se překlepy nachází téměř ve všech textech.

### 2.1.4 Další fonetické algoritmy

**NYSIIS** – New *York State Identification and Intelligence System* algoritmus – využívá kanonický indexový kód jako Soundex, ale na rozdíl od něj zachovává informaci o pozici samohlásek jejich přetransformováním na písmeno A. Jedná se o abecední algoritmus, který vrací čistý abecední kód.

**LIG algoritmus** – kombinuje 3 porovnávací metody – Guth, ISG a Levenshtein. Jedná se tedy o hybridní algoritmus kombinující fonetický a hláskový přístup. Pro měření podobnosti je použita pravděpodobnost. Jedná se o algoritmus s nejvíce přesnými odpověďmi. Existuje ve verzích LIG1, LIG2 a LIG3.

**ISG algoritmus** – *Index of Similarity Group* je hybridní algoritmus kombinující abecední a fonetický přístup. Porovnání podobnosti je založené na Guthově metodě.

## 2.2 Algoritmy podobnosti řetězců

Tyto algoritmy porovnávají dva textové řetězce a určují jejich podobnost. Dosahují velmi dobrých výsledků pro opravu překlepů v textech. Mezi nejčastěji používané metody pro toto porovnání patří výpočet Levenshteinovy vzdálenosti, která v kombinaci s dynamickým programováním představuje silný algoritmus pro tyto účely.

### 2.2.1 Dynamické programování

Dynamické programování [3] řeší rozdělení problému na dílčí, jednoduše řešitelné podproblémy. Řešení těchto podproblémů se poté využijí k finálnímu řešení daného problému. Je využíváno mnoha algoritmy.

Jedním z jeho hlavních úkolů je nalezení nejkratší cesty mezi dvěma body v tabulce, nebo v síti, kde každá buňka, resp. bod má danou hodnotu označovanou jako cena nebo váha. Úkolem je potom dojít z bodu A do bodu B za nejnižší cenu.

Jeden z případů jeho využití je uvedený v kapitole 2.2.2, kde je využit k výpočtu editační vzdálenosti dvou řetězců.

### 2.2.2 Levenshteinova vzdálenost

Jindy nazývaná také jako *editační vzdálenost*, je vzdálenost dvou řetězců/slov, definovaná jako počet změn ze zdrojového řetězce na cílový řetězec [7]. Jinými slovy je rovna počtu změn ve slově A, které je nutné provést, aby se z něj stalo slovo B. Patří mezi metrické metody. Pro příklad ze slova *tráva* vznikne slovo *krysa* ve třech krocích, protože je nutné v prvním slově změnit 3 znaky, aby vzniklo slovo druhé. Vzdálenost mezi těmito slovy je tedy rovna 3.

Př.: ***Tráva*** *->* *Kráva* *->* *Krása* *->* ***Krysa***

*1. krok – změna T –> K*
*2. krok – změna V –> S*
*3. krok – změna Á –> Y*

Levenshteinova vzdálenost nabízí pro editaci 3 operace:

- Substituce – nahrazení znaku ve zdrojovém řetězci za znak v cílovém
- Vložení
- Smazání

Každá z těchto operací má tzv. cenu. Cena operace substituce je rovna 0, pokud se jedná o totožné znaky, nebo 1, pokud se jedná o znaky odlišné. Cena operace smazání a operace vložení je 1. Celková vzdálenost dvou řetězců je potom nejnižší cena přeměny jednoho řetězce na druhý.

Nejběžnějším způsobem výpočtu Levenshteinovy vzdálenosti je použití dynamického programování. To probíhá tak, že se nejprve vytvoří matice o $m + 1$ x $n + 1$ prvcích, kde $m$ a $n$ jsou délky porovnávaných řetězců. Nultý řádek a nultý sloupec v této matici se vyplní posloupností čísel $1…m$, resp. $1…n$.

Hodnota každého prvku v matici se potom vypočítá z jeho okolí podle následujícího schématu:

| *Hodnota v diagonále*<br>*X* | *Hodnota nad buňkou*<br>*Z* |
|---|---|
| *Hodnota vlevo od buňky*<br>*Y* | **Výběr minimální hodnoty z okolí + přičtení ceny operace, tedy:**<br>**min (**<br>**X + cena substituce;**<br>**Y + cena vložení;**<br>**Z + cena smazání)** |

**Tab. 2.2: Výpočet prvku v matici**

Jak naznačuje tabulka 2.1, tak každá operace znamená pohyb v matici jiným směrem. Jedná se o pohyb ve směru dolů po diagonále u operace substituce. U operace smazání jde o přímý pohyb dolů a u operace vložení jde o pohyb vodorovný ve směru doprava. Ke každému pohybu se následně přičte cena dané operace a vzniká tím konečná hodnota dané buňky.

K výpočtu této metody na PC se využívá dynamické programování. To koresponduje s nutností projít tabulku, jejíž buňky obsahují různé číselné hodnoty, za nejnižší cenu, tzn. vybrat cestu takovou, aby vedla přes buňky s nejnižší hodnotou a zároveň aby těch buněk bylo co nejméně.

Kód pro výpočet této metody za použití dynamického programování (viz výpis 2.1) a výsledek tohoto výpočtu na dvou testovacích slovech je uveden na další straně této práce.

Výpis 2.1: Pseudokód metody výpočtu editační vzdálenosti [8]

```
if s[i] = int LevenshteinDistance(char s[1..m], char t[1..n])
 {
   // d je matice o m+1 řádcích a n+1 sloupcích
   declare int d[0..m, 0..n]

   for i from 0 to m
     d[i, 0] := i // operace smazání
   for j from 0 to n
     d[0, j] := j // operace vložení

   for j from 1 to n
   {
     for i from 1 to m
     {
t[j] then
         d[i, j] := d[i-1, j-1]
       else
         d[i, j] := minimum
                    (
                      d[i-1, j] + 1,  // smazání
                      d[i, j-1] + 1,  // vložení
                      d[i-1, j-1] + 1 // substituce
                    )
     }
   }

   return d[m,n]
 }
```

Výše uvedený kód porovnává řetězec *s* o délce *m* s řetězcem *t* o délce *n*. Výsledkem je matice koeficientů *d*, ve které je na pozici *d[m + 1, n + 1]* hodnota korespondující s Levenshteinovou vzdáleností dvou porovnávaných řetězců. Tato matice, spočítaná pro výše uvedená slova *tráva* a *krysa*, je znázorněna na obrázku 2.1.

| | | K | R | Y | S | A |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |
| **T** | 1 | 1 | 2 | 3 | 4 | 5 |
| **R** | 2 | 2 | 1 | 2 | 3 | 4 |
| **Á** | 3 | 3 | 2 | 2 | 3 | 4 |
| **V** | 4 | 4 | 3 | 3 | 3 | 4 |
| **A** | 5 | 5 | 4 | 4 | 4 | 3 |

**Obr. 2.1: Matice editační vzdálenosti dvou slov**

Zde je vidět, že je na pozici *d[m + 1, n + 1]* hodnota *3*. To souhlasí s výše uvedenými předpoklady. Šedivé buňky zde označují cestu k nejnižší ceně. V tomto případě se vždy jednalo o substituci, proto šlo ve všech případech o pohyb po diagonále.

### 2.2.3 Damerau–Levenshteinova vzdálenost

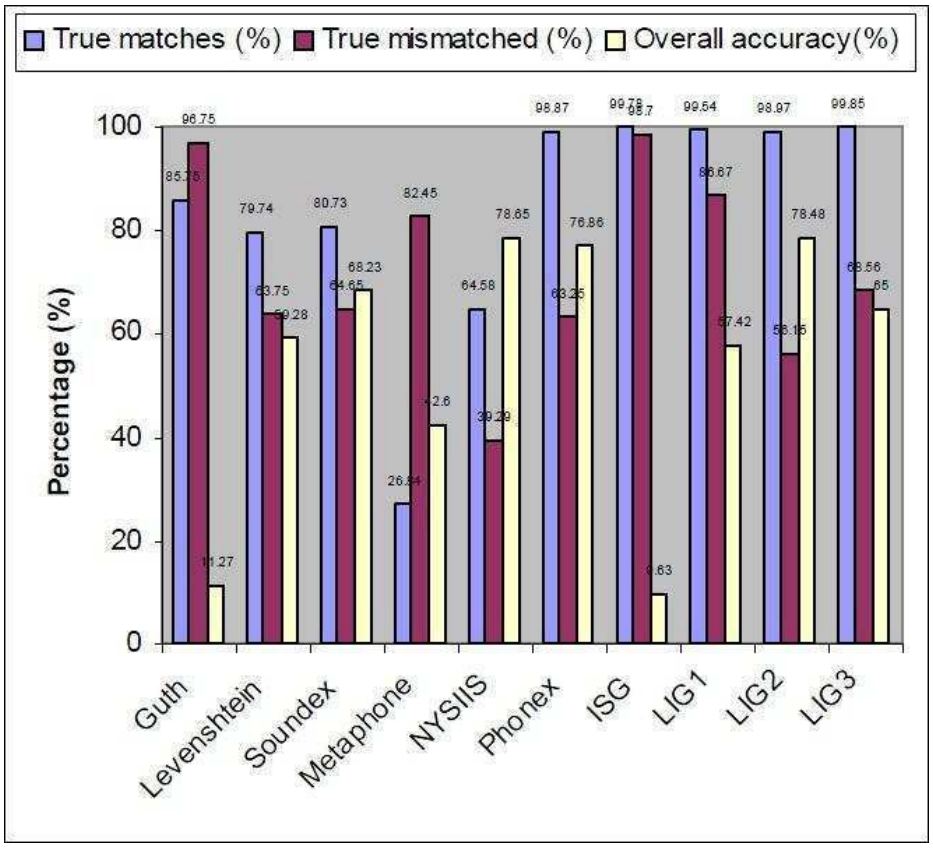
Damerau- **Levenshtein**ova vzdálenost je metrická metoda, která vychází z Levenshteinovy vzdálenosti popsané v kapitole 2.2.2. Hlavním rozdílem od Levenshteinovy vzdálenosti je přidání operace transpozice. Princip funkce je jinak u obou metod stejný. Tato metoda je dnes využívána častěji, než Levenshteinova vzdálenost. Používá se například k porovnávání řetězců DNA nebo k odhalování podvodů.

### 2.2.4 Kontrola pravopisu

Jak již bylo zmíněno, tento algoritmus je vhodným kandidátem pro použití v kontrole překlepů. Pro kontrolu pravopisu je ale poněkud omezen, avšak vhodným základním stavebním kamenem aplikace pro kontrolu pravopisu, která by využívala kombinace porovnávacího a fonetického algoritmu.

## 2.3 Porovnání algoritmů

Následující graf ukazuje výsledky porovnání 10 různých algoritmů pro detekci chyb v textech. Toto porovnání bylo provedeno v [9] Charkkitem Snaem na testovací databázi 11,369 amerických příjmení [9].




Obr. 2.2: Porovnání 10 algoritmů

Z grafu na obrázku 2.2 je patrné, že nejúspěšnější algoritmy pro porovnání jmen jsou NYIIS, LIG2 a Phonex. Algoritmy NYIIS a Phonex jsou specializované na americká jména, proto není tento výsledek, vzhledem k testovacím datům, překvapivý. Uspokojivých výsledků dosáhl algoritmus LIG ve všech svých verzích. Jeho hybridní koncepce (viz podkapitola 2.1.4) ho předurčuje k univerzálnímu použití, nejen pro porovnání jmen.

# 3 API pro kontrolu pravopisu

V současné době existuje několik API (rozhraní pro programování aplikací) pro kontrolu pravopisu. Jsou určené pro různé programovací jazyky a mají různé licence. Např. společnosti jako Google a Yahoo mají své API určené pro implementaci do webových stránek uživatelů. Některé API však umožňují tvorbu samostatných aplikací. Nejznámější z těchto API jsou Aspell a Jazzy. První z nich je určený pro jazyk C, druhý je pro jazyk Java.

## 3.1 Aspell

Jedná se o API (http://aspell.net/) pro kontrolu pravopisu s volnou licencí. Podporuje mnoho světových jazyků, včetně českého. Knihovny tohoto rozhraní jsou určeny pro jazyk C a existuje také ve formě samostatného programu. Jedná se o rozšíření původně unixového kontroloru pravopisu Ispell, který využívá ke kontrole pravopisu metrickou metodu Damerau–Levenshteinovu vzdálenost, která se od Levenshteinovy vzdálenosti liší především rozšířením operací o transpozici znaků. Využívá kombinace fonetického (Metaphone) a metrického algoritmu (Ispell). Tím dosahuje výborných výsledků v kontrole pravopisu.

## 3.2 Jazzy

API Jazzy (http://sourceforge.net/projects/jazzy/) je postavené na API Aspell a je určené pro jazyk Java. Stejně jako Aspell, využívá jako fonetickou metodu Metaphone, ale má zde i možnost definice vlastní fonetické metody. Jako metrickou metodu využívá metodu založenou na metodě výpočtu Levenshteinovy vzdálenosti. Ta se od ní odlišuje přidáním operace prohození sousedících slov a změnu znakové sady písmena. Cena operací je konfigurovatelná pomocí tabulkového zápisu kódu. Díky tomu lze aplikace postavené na API Jazzy snadno nakonfigurovat pro ostatní jazyky, než je jazyk anglický.

### 3.2.1 Funkce

Algoritmus, se kterým Jazzy pracuje, je založený na Aspell algoritmu. Pokud se v kontrolovaném textu objeví slovo, které se neshoduje s žádným slovem ve slovníku, potom je označeno jako nespisovné a jsou provedeny následující kroky k dohledání navrhovaných slov k opravě:

1. **Nalezení foneticky nejbližších slov k chybnému slovu** – Hledají se slova se stejným fonetickým kódem, u kterých editační (Levenshteinova) vzdálenost nepřesahuje stanovenou hodnotu.
2. **Nalezení foneticky nejbližších slov odvozených od chybného slova** – Hledají se slova, která jsou o jednu editační operaci od chybného slova. Z těchto slov se vyberou slova se stejným fonetickým kódem, jaký má chybné slovo. Tato slova nesmí přesahovat stanovenou editační vzdálenost.
3. **Nejlepší odhad** – pokud nejsou nalezeny žádné návrhy hledaného slova, tak se dohledají slova se stejným fonetickým kódem a s nejmenší editační vzdáleností.

4. **Setřídění** – navrhovaná slova se setřídí pouze podle editační vzdálenosti.

Díky kombinaci fonetické a porovnávací metody dosahuje API Jazzy dobrých výsledků v prohledávání navrhovaných slov, což je dobrý předpoklad pro aplikace pro kontrolu pravopisu.

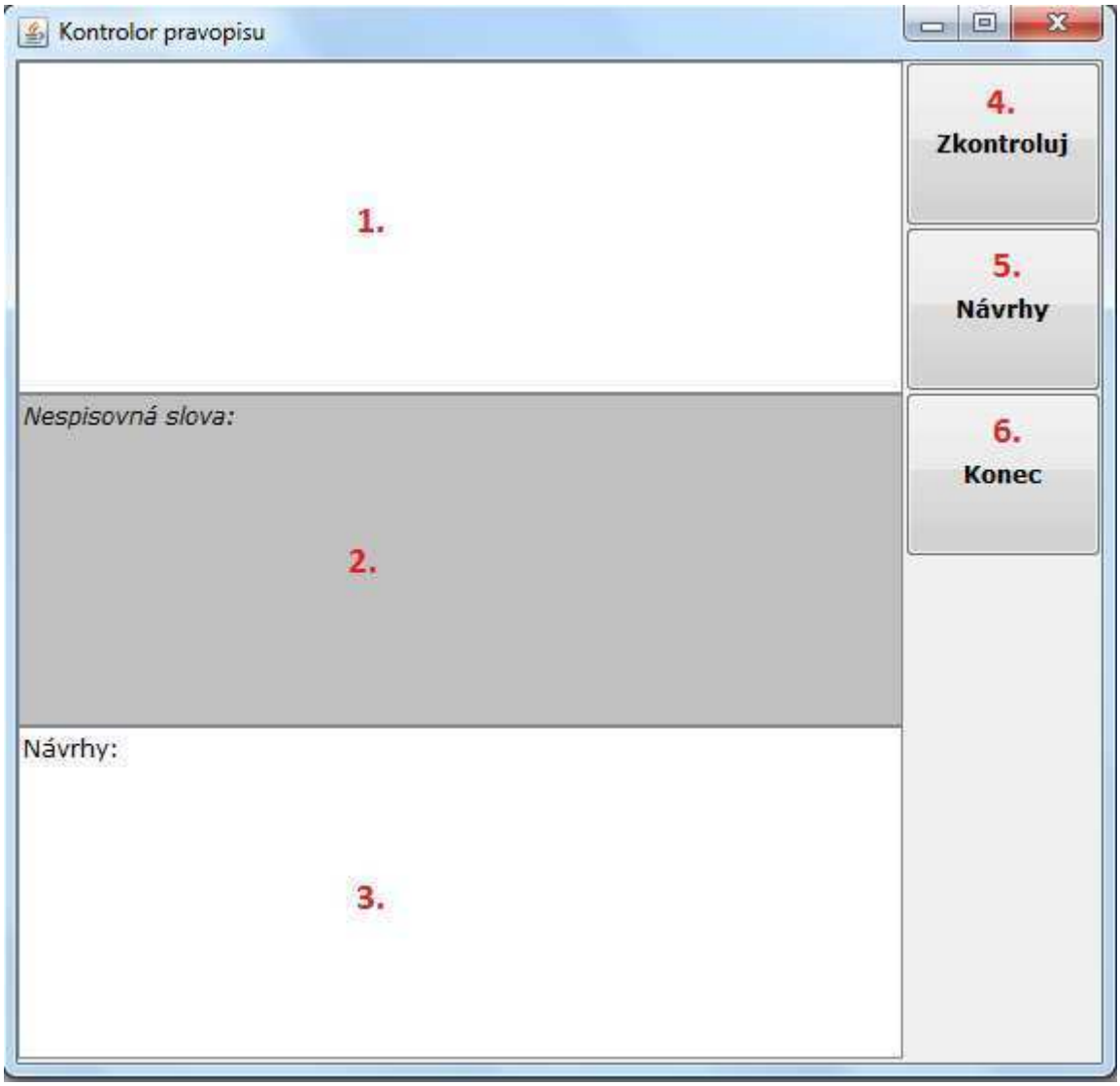
Pro kontrolu v českém jazyce se používá kombinace buď fonetického souboru pro anglický jazyk spolu se slovníkem českých slov, pro porovnání distanční (editační) vzdálenosti, nebo se fonetický soubor úplně vynechá a počítá se pouze distanční vzdálenost.

# 4 Testovací uživatelské rozhraní

Pro testovací účely bylo vytvořeno jednoduché uživatelské rozhraní. Toto rozhraní využívá API Jazzy a je tedy vytvořeno v programovacím jazyce Java.

## 4.1 Popis

Protože rozhraní slouží k testování a ukázkám, obsahuje jednoduchá ovládací tlačítka. Rozhraní je zobrazeno na obr. 4.1.




Obr. 4.1: Uživatelské rozhraní

Obsahuje 3 textová pole:

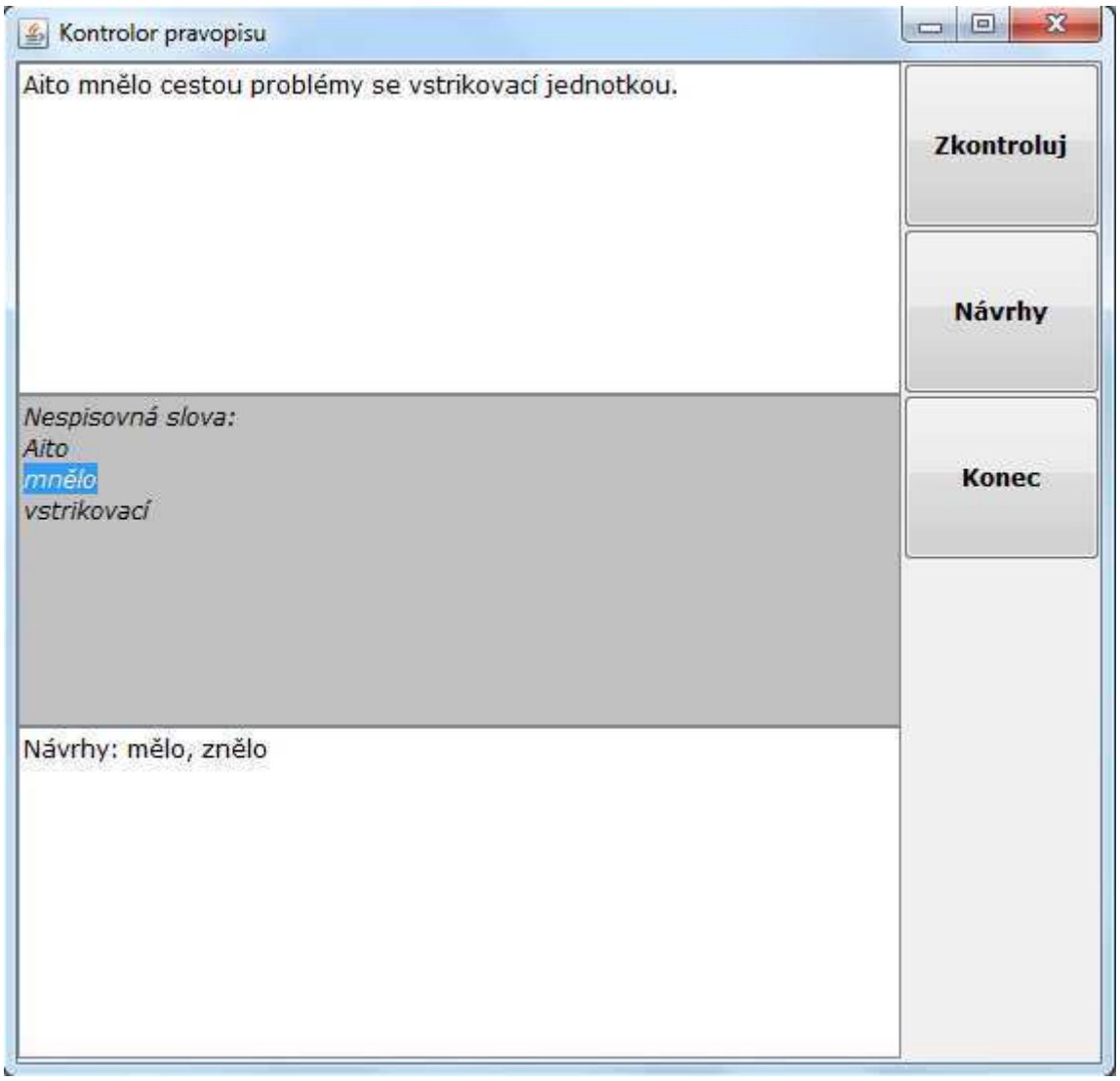
1. Pole kontrolovaného textu
2. Pole, kde jsou vypisována nespisovná slova
3. Pole, kde jsou zobrazeny návrhy na opravu chybného slova

Dále jsou zde 3 tlačítka:

4. Tlačítko „Zkontroluj“ zkontroluje vložený text a vyhledá v něm nespisovná slova. Ta jsou potom vypsána do pole, označené číslem 2.
5. Tlačítko „Návrhy“ má prozatím funkci vypsání návrhu oprav pro nespisovné slovo.
6. Tlačítko „Konec“ ukončí program.

## 4.2 Princip funkce

Funkce programu je znázorněna na obr. 4.2. Následuje její popis:




Obr. 4.2: Funkce programu

1. Text obsažený v textovém poli 1 projde po stisknutí tlačítka „Zkontroluj“ kontrolou pravopisu, kdy se slovo po slovu porovnává se slovy ve slovníku.
2. Slova, která nejsou ve slovníku nalezena, se vypíší do textového pole 2.
3. Při označení příslušného slova a použitím tlačítka „Návrhy“ dojde do textového pole 3 k výpisu návrhů na opravu příslušného slova.

## 4.3 Výstupy

Pro kontrolu pravopisu byl použit následující text [1], do kterého byly úmyslně zavedeny chyby:

*„Mlha je průzracná, ale těžko se skrze ni rozzná barva a obrysy predmětů. Vsecko se zdá být něčim jiným, ney je. Jedeš a náhle vidíš vpředu, těsně vdle cesty, jakoby siuetu mnicha, nehybnou, čekající a cosi džící v rukou. Není to snad lupič? Postava se přibližuje, roste, teď ji bryčka mýjí a vy poznáváte, že to není člověk, ale osamnělý keř nebo veliký balvan. Stejně nehybné, kohosi očekávající postavi stojí na pahorcích, schovávají se za mohylami, vyhlížejí ze stepní trávy a všechny vypadají jako lidé a jsou podezrelé.“*

Text obsahuje 14 chyb a překlepů. Následující tabulka Tab. 4.1 obsahuje nalezená slova vyhodnocená algoritmem jako chybná a návrhy na jejich opravu.

| Chybné slovo | Návrhy opravy |
|---|---|
| průzracná | průzračná |
| těžko | těžko |
| rozzná | rozená, rozezná |
| predmětů | předmětů |
| Vsecko | vřecko, všecko |
| něčim | něčem, něčím |
| ney | ne, neu, new, než, key, ned, net, nes, nez, neb, nep, nej |
| vdle | vedle, vidle, vde, dle, vele, vile, vole, vále, víle, vůle, mdle |
| siuetu | siluetu |
| džící | držící, lžící |
| mýjí | mají, mojí, myjí, míjí, mýlí |
| osamnělý | osamělý |
| postavi | postav, postava, postavu, postavy, postaví, postavě, postavit, postavil |
| podezrelé | podezřelé |

Tab. 4.1: Chybná slova a návrhy na jejich opravu

Jak je vidět z Tab. 4.1, tak algoritmus odhalil všechny chyby v textu. Zde hraje velkou roli použitý slovník, který algoritmus prochází. Pokud má slovník dostatečné množství slov a jejich tvarů, potom je kontrola přesnější. Jakékoliv slovo, které není ve slovníku nalezeno, je označeno jako nespisovné a hledá se jemu podobné slovo (viz. kapitola 3.2.1). Spisovné návrhy oprav jsou ke každému nespisovnému slovu zobrazeny v pravém sloupci tabulky.

U slov *ney* a *vdle* najdeme mnoho návrhů pro opravu. To je dáno stavbou daných slov a jejich podobností ostatním slovům. Nicméně u obou těchto slov se mezi navrhovanými slovy vždy nachází to správné.

Originální text bez chyb vypadá takto [1]:

> *„Mlha je průzračná, ale těžko se skrze ni rozezná barva a obrysy předmětů. Všecko se zdá být něčím jiným, než je. Jedeš a náhle vidíš vpředu, těsně vedle cesty, jakoby siluetu mnicha, nehybnou, čekající a cosi držící v rukou. Není to snad lupič? Postava se přibližuje, roste, teď ji bryčka míjí a vy poznáváte, že to není člověk, ale osamělý keř nebo veliký balvan. Stejně nehybné, kohosi očekávající postavy stojí na pahorcích, schovávají se za mohylami, vyhlížejí ze stepní trávy a všechny vypadají jako lidé a jsou podezřelé.“*

## 4.4 Funkce automatické opravy

Tato aplikace neobsahuje funkci automatické opravy textu. Slouží pouze jako demonstrátor možností API Jazzy.

Funkce automatické opravy vyžaduje použití speciálního algoritmu. Takový algoritmus může pracovat tak, že na základě kontextu věty vybere z navrhovaných slov to správné a nahradí jím nespisovné slovo. Jednou z cest jak toho dosáhnout je aplikace algoritmu, který pracuje se statistickými údaji. Tento algoritmus je dále popsán v kapitole 5. V části tohoto algoritmu je využito API Jazzy.

# 5 Aplikace pro opravu pravopisu

Aplikace je napsaná v programovacím jazyce Java a slouží pro kontrolu a automatickou opravu textu. Nepředpokládá se u ní zásah uživatele do opravy. Tato aplikace má tvořit jeden ze stupňů aplikace pro detekci emocí z textů.

Jazyk Java byl vybrán pro snadnou přenositelnost aplikací napsaných v tomto jazyce mezi platformami (Windows, Linux, Mac OS, Solaris) bez nutnosti kompilace kódu pro každou platformu zvlášť. Dalším důvodem volby tohoto jazyku je snadná integrace API Jazzy, které je v tomto jazyce napsané.

Vstupem aplikace je neopravený a nekontrolovaný text a jejím výstupem má být gramaticky správný text, aby se tak usnadnila práce následujícím stupňům hlavní aplikace, jejíž součástí má tato aplikace být. Na výstupu není požadována interpunkce, proto jí výstupní text neobsahuje (vyjma konců vět), což aplikaci mírně zjednodušilo.

Pro účely automatické opravy textu byl vyvinut algoritmus, který pracuje se statistickými daty. Ten je blíže popsán v kapitole 5.1. Tato kapitola dále obsahuje popis podpůrných aplikací a alternativ použitých komponent.

Vzhledem k náročnosti některých kroků algoritmu klade aplikace vyšší nároky na HW PC, na kterém je spouštěna. Pojednání o dané problematice je blíže popsáno v kapitole 5.2.

V kapitole 5.3, je popsaná metoda testování výkonnosti aplikace. Dále tato kapitola obsahuje výsledky těchto testů a výstupy.

## 5.1 Algoritmus

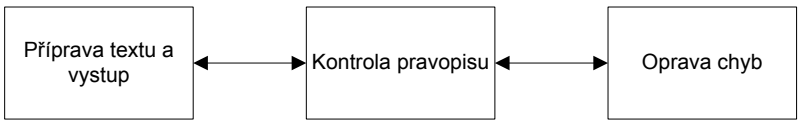
Hlavní myšlenkou při vývoji tohoto algoritmu bylo, aby byl schopen rozpoznat chybu v textu nejen na základě pravopisu, ale také na základě kontextu dané věty a aby tuto chybu dokázal následně opravit bez zásahu uživatele.

Jako vhodné řešení se nabízelo využít shluků slov z vět a získat tak statisticky nejčastěji vyskytovaná slova kolem slova zkoumaného. Pro tento úkol byly vybrány jedinečné trojice z vět, takzvané trigramy. Důvodem výběru právě trigramů bylo, že digramy, což jsou dvojice slov, neposkytují dostatečné informace z okolí slova. Naproti tomu n – gramy, kde $n > 3$ jsou již zbytečně veliké. Trigramy jsou kompaktní a přitom poskytují požadovaná data.

Algoritmus se skládá ze tří hlavních stupňů. Jejich výčet je:

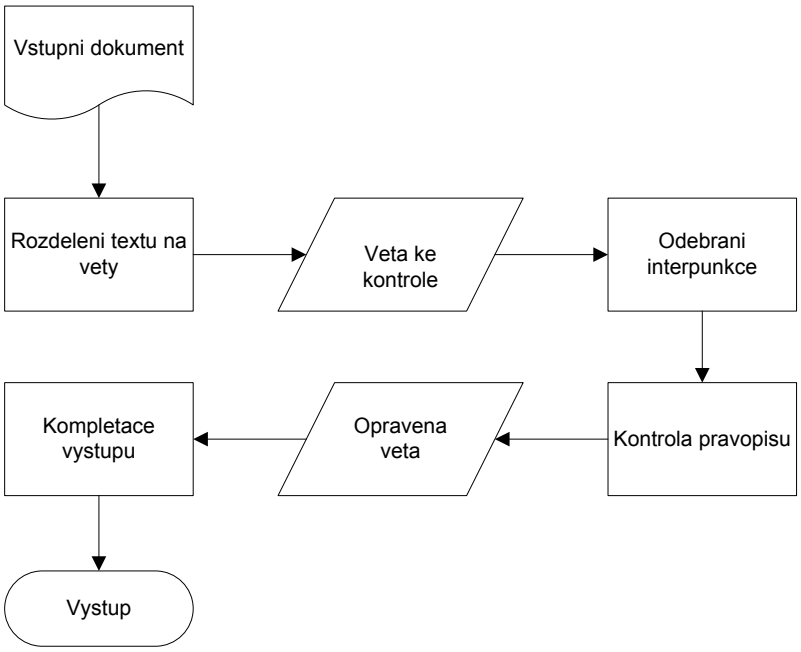
1. Zpracování vstupního textu
2. Kontrola pravopisu
3. Oprava chyb

Tyto stupně spolu spolupracují, jak ukazuje celkové základní schéma, které je znázorněno níže na obr. 5.1.




Obr. 5.1: Základní schéma hlavních stupňů

Schéma prvního stupně je znázorněno na obr. 5.2. V tomto stupni probíhá načtení vstupního textu a jeho rozdělení na jednotlivé věty. Tyto věty se následně zbaví interpunkce a předají se dále druhému stupni „Kontrola pravopisu“, který je ve schématu na obr. 5.2 reprezentovaný stejnojmenným blokem.

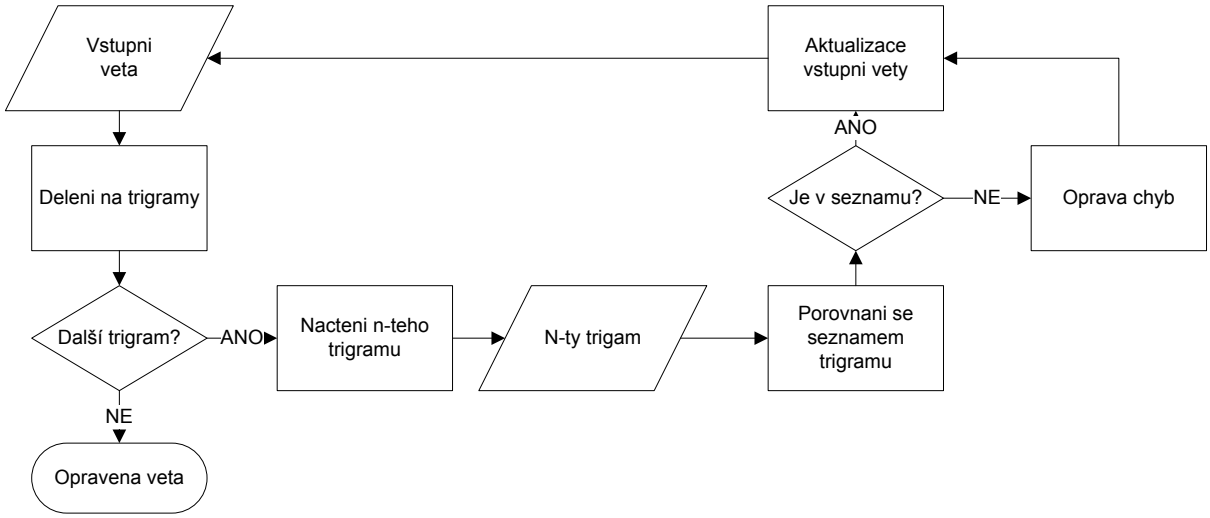



Obr. 5.2: Princip funkce prvního stupně

Výstupem toho stupně je zkontrolovaná a opravená věta, která je následně předána výstupnímu bloku, kde je přidána do výstupního textu. Výstupem tohoto bloku opravený text. Princip funkce je znázorněn na obr. 5.2.

Jak patrné z obr. 5.2, tak u opravené věty neprobíhá obnova interpunkce. Tato obnova, vzhledem k určení aplikace, není vyžadována, proto není implementovaná.

Dalším hlavním stupněm algoritmu je stupeň „Kontrola pravopisu“. Jeho vstupem je kontrolovaná věta. Tato věta se postupně dělí na trigramy, což jsou trojice sousedících slov, a tyto trigramy se jednotlivě porovnávají se seznamem trigramů. Pokud jsou v seznamu nalezeny, považuje se trigram za korektní. V opačném případě je trigram předán stupni „Oprava chyb“, který je ve schématu na obr. 5.3 reprezentovaný stejnojmenným blokem. Tento stupeň navrátí zkontrolovaný a opravený trigram a ten je následně použit jako náhrada původního, chybného, trigramu. Princip funkce tohoto stupně je znázorněn na obr. 5.3.

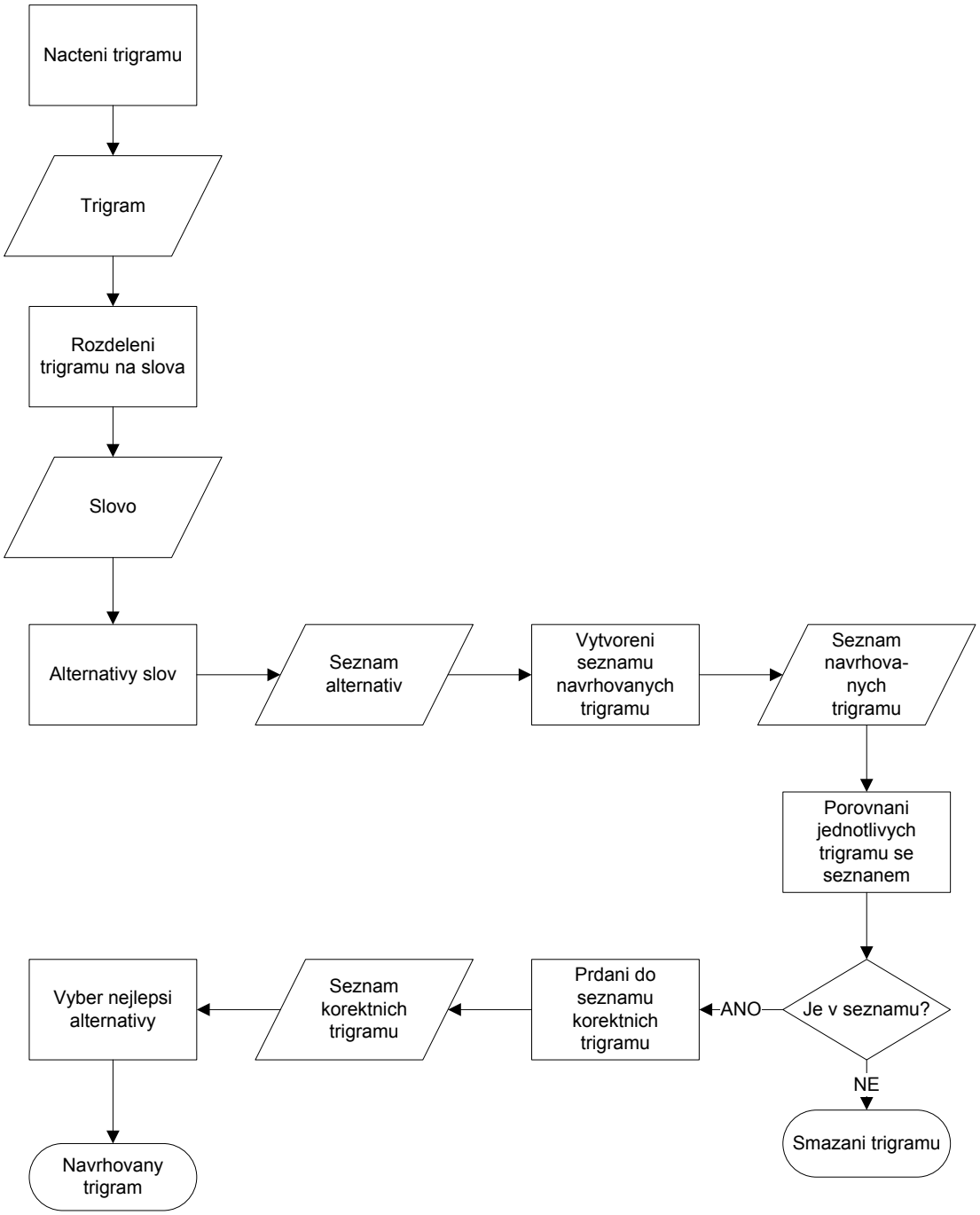



**Obr. 5.3: Princip funkce druhého stupně**

Důležitou součástí tohoto stupně je blok „Aktualizace vstupní věty“, protože průběžně aktualizuje vstupní větu o opravené trigramy. Z takto aktualizované věty se následně vytváří další trigram. Pokud by se vstupní věta průběžně neaktualizovala a vytvořily se z ní trigramy přímo, tak by na výstupu vzniklá věta pozbývala v kontextu smyslu. Celkový počet trigramů *n* ve větě je $n = p - 2$ kde *p* je rovno počtu slov v dané větě. Například věta: „Naším největším úspěchem bylo druhé místo“ obsahuje 6 slov a tím pádem 4 trigramy *(naším největším úspěchem*, *největším úspěchem bylo*, *úspěchem bylo druhé*, *bylo druhé místo*). Pokud došlo ke kontrole všech trigramů, tak je proces ukončen a na výstup je předána opravená věta.

Posledním funkčním stupněm algoritmu je stupeň „Oprava chyb“, jehož vstupem je trigram, který nebyl nalezen předchozím stupněm v hlavním seznamu trigramů. Tento trigram je následně rozdělen na jednotlivá slova. Pro každé slovo se pomocí slovníku vyhledá jeho $x \leq 10$ alternativ. Číslo *10* zde vyjadřuje počet slov, která se vyberou ze seznamu navrhovaných alternativ/oprav pro dané slovo. Jeho hodnotu lze tedy chápat jako hloubku, do které se algoritmus dívá při sestavování navrhovaných trigramů na opravu textu. Tato hodnota byla zvolena na základě testů jako vyhovující. Pokud opravované slovo nemá deset alternativ, tak se použije pouze jejich dostupný počet. Vzniknou tak 3 seznamy po deseti slovech. Pokud je ke slovu nabídnuto méně, než 10 alternativ, potom se seznam doplní o kopie poslední nabízené alternativy k danému slovu. Z takto vytvořených seznamů se kombinací slov „každý s každým“ vytvoří v bloku „Vytvoření seznamu navrhovaných trigramů“ seznam navrhovaných trigramů. Tento seznam se následně prochází po jednotlivých trigramech a

kontroluje se, zda daný trigram je obsažený v hlavním seznamu trigramů, či nikoliv. Pokud ano, je trigram přidán do zvláštního seznamu, který obsahuje pouze ty trigramy z navrhovaných, které jsou obsaženy zároveň v hlavním seznamu trigramů, čili ty gramaticky správné. Ke každému z těchto trigramů je přiděleno číslo reprezentující hodnotu, která koresponduje s četností výskytu daného trigramu v hlavním seznamu. Podle této hodnoty je potom vybrán trigram s nejvyšší četností výskytu a je navrhnut pro opravu. Může se však stát, že ani seznam navrhovaných trigramů neobsahuje trigram, který je obsažen v hlavním seznamu trigramů. V tom případě tento blok vrací prázdnou hodnotu *NULL* a trigram, který byl původně detekován jako chybný, je ponechán neopraven. Princip funkce tohoto bloku je znázorněn na obr. 5.4.




**Obr. 5.4: Princip funkce třetího stupně**

Samostatná část „Alternativy slov“ je realizována pomocí API Jazzy (viz kapitola 3.2). Tato část navrací ke každému slovu seznam jeho možných alternativ. Toto rozhraní je navrženo především pro anglický jazyk. Jak ukazují testy v kapitole 5.3, tak je tato skutečnost původcem problému v případě detekce některých chyb. Částečně je toto eliminováno použitím trigramů a využitím deseti alternativ navrhovaných oprav. Kdy jsou do tvorby navrhovaných trigramů zahrnuty i ty návrhy opravy slov, které uvedené rozhraní považuje za méně výhodné (mají větší distanční (editační) vzdálenost od kontrolovaného slova).

Tato část (stupeň) algoritmu je početně nejnáročnější. Může za to především část „Alternativy slov“, kde probíhá vyhledávání navrhovaných slov a sestavování navrhovaných trigramů z těchto slov.

### 5.1.1 Vytvoření hlavního seznamu trigramů a jeho struktura

Hlavní seznam vytvářen z externího souboru, který je aplikací načten při jejím spuštění do paměti. Tento soubor obsahuje trigramy vytvořené ze vstupních dat. Je tak umožněno soubor jednoduchým způsobem průběžně aktualizovat o nová data.

Pro vytvoření hlavního seznamu je třeba použít co největší množství reálných a zároveň gramaticky správných textů – vstupních dat. Předejde se tak řídkosti dat, která může ve výsledku způsobit nesprávnou opravu textu, protože bude v seznamu málo trigramů a nebude podle čeho kontrolovat text. Texty by měly být použity z oboru, ze kterého bude pocházet opravovaný text (e – mail, zpravodajství, internetové komunikátory jako například ICQ, …).

Z těchto textů se vytvoří trigramy. U trigramů, které pochází ze začátku věty, se první znak převede na malé písmeno. Dále se upravené trigramy uloží do samostatného souboru. Takto je možné tento soubor aktualizovat o další trigramy z nových textů, kdy se nové trigramy přidají na konec souboru. Struktura aplikace umožňuje takto vypadající soubor, kdy trigramy nejsou seřazeny podle abecedy, použít pro tvorbu hlavního seznamu trigramů.

Výpis 5.1: Výpis ze souboru trigramů a jejich prezentace v databázi aplikace

| *Výpis* | *Prezentace v databázi* | |
|---|---|---|
| . | . | |
| . | . | |
| . | . | |
| *vždy nutno přihlédnout* | *vždy nutno přihlédnout* | *3* |
| *nutno přihlédnout ke* | *nutno přihlédnout ke* | *2* |
| *přihlédnout ke složitosti* | *přihlédnout ke složitosti* | *1* |
| . | . | |
| . | . | |
| . | . | |
| *brožury hostýnských vrchů* | *brožury hostýnských vrchů* | *5* |
| *hostýnských vrchů nebo* | *hostýnských vrchů nebo* | *3* |
| *vrchů nebo bývá* | *vrchů nebo bývá* | *1* |
| *nebo bývá k* | *nebo bývá k* | *2* |
| . | . | |
| . | . | |
| . | . | |

Při spuštění aplikace dojde k načtení souboru obsahujícího trigramy do lokální databáze. Každý trigram je v této databázi uložen pouze jednou a je k němu přidána číselná hodnota vyjadřující počet jeho výskytů ve vstupních textech. Výpis ze souboru trigramů, který je použit v aplikaci, je uveden níže (viz Výpis 5.1).

Aplikace, o níž tato diplomová práce pojednává, pracuje se seznamem vytvořeným ze vstupních dat, která zahrnují 4600 článků ze serveru www.novinky.cz. Články byly převzaty z tematicky různých rubrik (zpravodajství, sport, ekonomika). V tabulce 5.1 je uveden statistický přehled dat získaných z uvedených textů.

| Počet článků | Počet slov celkem | Počet trigramů |
|---|---|---|
| 4600 | 1,657,876 | 1,179,790 |

Tab. 5.1

*Počet slov celkem …… celkový počet slov (tj. vč. opakujících se)*
*Počet trigramů …….. počet trigramů bez opakování (tj. každý trigram z tohoto počtu je jedinečný)*

Soubor s trigramy se vytváří separátně od algoritmu v samostatné aplikaci (též napsané v jazyce Java). Aplikace funguje výše uvedeným způsobem, kdy jejím vstupem je korektní text a jejím výstupem je soubor s trigramy, které jsou řazeny pod sebe.

V důsledku načtení databáze při spuštění programu, trvá toto spuštění delší dobu (více viz kapitola 5.2).

### 5.1.2 Opravný proces

Jak už bylo uvedeno výše, tak každá věta je zbavena interpunkce a následně je první znak věty převeden na malé písmeno. Po této proceduře následuje postupné dělení věty o *n* slovech na *n – 2* trigramů. Vždy se vytvoří jeden trigram a ten projde kontrolou pravopisu, kdy se porovnává s hlavním seznamem trigramů. Pokud je trigram v seznamu nalezen, tak pokračuje kontrola dalšího trigramu věty. Pokud nalezen není, tak následuje jeho náhrada za opravený trigram, nebo jeho případné ponechání, pokud se vhodná náhrada nenajde. Tímto způsobem se kontrolují všechny trigramy věty. Po kontrole trigramu se věta vždy aktualizuje před kontrolou dalšího trigramu v pořadí. Před přidáním věty na výstup je její první znak převeden na velké písmeno a přidána interpunkce na její konec.

Pro příklad souvětí *„Dnes je venku krásně počasí, ale já musím sedet doma a psát diplomovou práci“,* do kterého byly zavedeny chyby, se postupně rozdělí na 12 trigramů a zpracuje následovně (v každém lichém bodě je aktualizovaná věta a v každém sudém je kontrolovaný trigram):

1. *dnes je venku krásně počasí ale já musím sedet doma a psát diplomovou práci*
2. *dnes je venku* – správně, trigram je ponechán
3. *dnes je venku krásně počasí ale já musím sedet doma a psát diplomovou práci*
4. *je venku krásně* – správně, trigram je ponechán
5. *dnes je venku krásně počasí ale já musím sedet doma a psát diplomovou práci*
6. *venku krásně počasí* – **špatně**, nahrazeno trigramem „*venku krásné počasí*“
7. *dnes je venku krásné počasí ale já musím sedet doma a psát diplomovou práci*
8. *krásné počasí ale* – správně, trigram je ponechán
9. *dnes je venku krásné počasí ale já musím sedet doma a psát diplomovou práci*
10. *počasí ale já* – správně, trigram je ponechán

11. *dnes je venku krásné počasí ale já musím sedet doma a psát diplomovou práci*
12. *ale já musím* – správně, trigram je ponechán
13. *dnes je venku krásné počasí ale já musím sedet doma a psát diplomovou práci*
14. *já musím sedet* – **špatně**, nahrazeno trigramem „*já musím sedět*“
15. *dnes je venku krásné počasí ale já musím sedět doma a psát diplomovou práci*
16. *musím sedět doma* – správně, trigram je ponechán
17. *dnes je venku krásné počasí ale já musím sedět doma a psát diplomovou práci*
18. *sedět doma a* – správně, trigram je ponechán
19. *dnes je venku krásné počasí ale já musím sedět doma a psát diplomovou práci*
20. *doma a psát* – správně, trigram je ponechán
21. *dnes je venku krásné počasí ale já musím sedět doma a psát diplomovou práci*
22. *a psát diplomovou* – správně, trigram je ponechán
23. *dnes je venku krásné počasí ale já musím sedět doma a psát diplomovou práci*
24. *psát diplomovou práci* – správně, trigram je ponechán
25. *dnes je venku krásné počasí ale já musím sedět doma a psát diplomovou práci*
26. *Dnes je venku krásné počasí ale já musím sedět doma a psát diplomovou práci.*

V kroku č. 6 je vidět, že nalezený trigram obsahuje nestandardní spojení slov „*krásně počasí*“ a je následně nahrazen trigramem, který obsahuje standardní spojení „*krásné počasí*“. V kroku č. 14 je jako nesprávný detekován trigram „*já musím sedet*“ a je nahrazen trigramem „*já musím sedět*“.

Výše uvedený proces je proveden u všech vět vstupního textu a věta je v předposledním kroku (zde krok č. 25) předána na výstup aplikace ke kompletaci výstupního textu. V posledním kroku (zde krok č. 26) dochází k převedení prvního znaku věty na velké písmeno a k přidání tečky za větu.

Takto opravená věta je přidána k výstupu aplikace. Po kontrole věch vět z opravovaného textu je zkompletovaný výstupní text předán na výstup aplikace a uložen jako řetězec, který slouží ke zpracovaní dalším stupněm v serverové aplikaci.

## 5.2 Provozní nároky

### 5.2.1 Rozbor

Algoritmus, který aplikace používá ke kontrole a opravě textu, využívá pro svou činnost statistických údajů v datech. Tato data se vytvářejí z předpřipraveného souboru při spuštění aplikace. Díky tomu dochází ke zdržení spuštění samotného algoritmu.

Soubor obsahující trigramy má v popisované aplikaci velikost 30 MB. Při této velikosti trvá načtení aplikace na sestavě s procesorem Intel Core2Duo 1,66 GHz, 2 GB RAM, OS Windows 7 64bit 21 sekund. Aplikace si v tomto případě vyžádá 500 MB operační paměti. Při velikosti souboru 176 MB spuštění aplikace na stejné sestavě trvá 63 sekund. Nároky na operační paměť zůstaly stejné z toho důvodu, že větší soubor byl vytvořen duplikováním obsahu původního, menšího souboru. Tím pádem lokální databáze opět obsahuje 1,179,790 trigramů, jenom číslo značící počet jejich výskytu ve vstupních datech bude u každého trigramu větší. U souboru o velikosti 325 KB, který obsahuje 14564 trigramů, je na uvedené sestavě doba spuštění aplikace 5 sekund. Aplikace si s tímto souborem vyžádala 276 MB operační paměti.

Zde je vidět, že doba spuštění závisí na velikosti souboru vstupních dat a nároky na operační paměť závisí na počtu jedinečných trigramů, které soubor obsahuje.

V prvním případě by se doba spuštění dala zkrátit úpravou souboru vstupních dat v předprocesoru, jehož výstupem by byl v souboru uložený seznam trigramů včetně počtu jejich výskytů. Díky tomu, že by tento soubor obsahoval každý trigram pouze jednou, by se zmenšila jeho velikost a tím by se zkrátilo načítání aplikace. Aplikace byla testována se souborem uvedeného formátu a po drobné úpravě dokáže s tímto souborem pracovat. Pro vytvoření takového soboru byla vytvořena samostatná utilita.

V druhém případě je zřejmé, že s použitím souboru, který bude obsahovat více jedinečných trigramů, porostou i nároky na velikost operační paměti. Zde by bylo při velkém objemu dat výhodné využít propojení aplikace s databází typu SQL. Integrace tohoto prvku by vyžadovala větší zásah do zdrojového kódu aplikace.

### 5.2.2 Závěr

Aplikaci lze bez většího zásahu do jejího zdrojového kódu upravit tak, aby se zkrátila doba jejího spuštění. Avšak protože je určena pro běh na serveru, není tato úprava potřeba. Její současné verze je výhodná i z hlediska jednoduché úpravy souboru vstupních dat prostým přidáním dalších trigramů do něj.

Co se týká nároků na operační paměť, tak lze předpokládat, že ani při trojnásobku počtu trigramů nároky nepřekročí velikost 1 GB. I přes to je integrace SQL databáze do této aplikace perspektivním krokem.

## 5.3 Výstupy a výkon

Tato podkapitola obsahuje výsledky testu funkce algoritmu na různých datech. V části 5.3.1 se testování provádělo na upravených vstupních datech. Ve druhé části 5.3.2 se testovalo na datech, která nepochází z množiny vstupních dat.

### 5.3.1 Texty ze vstupních dat

Texty, které byly použity k testování, pochází z dat, pomocí kterých byl vytvořen seznam hlavních trigramů. Jedná se o články ze serveru www.novinky.cz. Do těchto článků byly vneseny různé pravopisné chyby (vynechaná diakritika, záměna i/y a s/z). V tabulce 5.2 je uveden celkový přehled statistických údajů jednotlivých článků včetně úspěšnosti detekce a opravy chyb aplikací. Z tabulky je patrné, že u deseti testovaných článků kolísala úspěšnost algoritmu v rozmezí 70 – 100%, což na celou testovanou množinu nakonec znamená úspěšnost téměř 89%. Vzhledem k tomu, že se jedná o data odvozená ze vstupních dat, je tato hodnota poměrně nízká. Velkou zásluhu na tom má použité rozhraní Jazzy, sloužící pro návrh alternativ slov, ze kterých se dále vytváří náhrada chybných trigramů. Toto rozhraní počítá s distanční vzdáleností slov. Díky jeho přednostnímu určení pro anglický jazyk i přes výsledky uvedené v kapitole 3.3 některá slova chybně opraví. V takovém případě je správně navrhované slovo až na posledních pozicích v návrzích, nebo mezi navrhovanými opravami nefiguruje vůbec. Jedná se především o slova, která obsahují znaky „*č, š, ř, ž, ě, ď* a *ť*“, které jsou typické pro českou abecedu a nefigurují v abecedě anglické.

| Číslo článku | Počet slov | Počet chyb | Počet detekovaných chyb | Úspěšnost [%] |
|---|---|---|---|---|
| 1 | 112 | 10 | 10 | 100,00 |
| 2 | 179 | 7 | 7 | 100,00 |
| 3 | 140 | 10 | 10 | 100,00 |
| 4 | 190 | 8 | 7 | 87,50 |
| 5 | 177 | 10 | 8 | 80,00 |
| 6 | 148 | 8 | 8 | 100,00 |
| 7 | 143 | 8 | 8 | 100,00 |
| 8 | 172 | 9 | 7 | 77,78 |
| 9 | 86 | 10 | 8 | 80,00 |
| 10 | 143 | 10 | 7 | 70,00 |
| **CELKEM** | **1490** | **90** | **80** | **88,89** |

Tab. 5.2

Ve výpisu 5.2 je uveden příklad opravovaného textu, který je následovaný neupravenou ukázkou výstupu aplikace po opravě tohoto textu (viz Výpis 5.3). Do opraveného textu bylo zavedeno 9 pravopisných chyb. Jejich výčet je uveden v tabulce 5.3.

Výpis 5.2: Opravovaný text (originál zdroj www.novinky.cz)

*Svet se bojí nestability v Egyptě, Írán mluví o islámském probuzeni. Zatímco běžní Egypťané slaví v ulicích páteční odstoupení prezidenta Husního Mubaraka, světové mocnosti i okolní státy jsou opatrnější a připomínají, že zemi čeká ještě dlouha cesta. Americky prezident Barack Obama, pro jehož zemi je Egypt klíčovým spojencem v regionu, vyzval Egypťany, abi v zemi byla nastolena skutečná demokracie, zároveň však připoměl, že mnoho otázek zatím zůstává bez odpovědí. Spojené státy se stejně jako sousední Izrael obávají zejména růstu vlivu islamistů. Nejmenovaný činnitel americké vlády řekl televizní stanici CNN, že v Egyptě nyní bude následovat "nepředvídatelná další kapitola". Egypt je první arabská země, která v roce 1979 uzavřela mirovou dohodu s Izraelem a má s Tel Avivem relativně dobré vztahy. Nyní v Tel Avivu a Washingtonu panují obavy, že nový egyptský režim bude k Izraeli rezervovanější. "Je důležité, aby příští vláda mírovou dohodu ctila," konstatoval mluvčí Bílého domu Robert Gibbs. Íránský prezident Mahmúd Ahmadínežád hovořil v souvislosti s egyptskými protesty o "islámském probuzení" a připomněl i revoluci ve sve zemi, která v roce 1979 dostala k moci šíitské duchovní.*

| slovo správně | zavedená chyba | druh chyby | oprava |
|---|---|---|---|
| svět | svet | vynechání diakritiky | svět |
| dlouhá | dlouha | vynechání diakritiky | dlouhá |
| aby | abi | pravopisná chyba | aby |
| připomněl | připoměl | pravopisná chyba | připomněl |
| činitel | činnitel | pravopisná chyba | činitel |
| mírovou | mirovou | vynechání diakritiky | mírovou |
| své | sve | vynechání diakritiky | své |

Tab. 5.3: Přehled zavedených chyb a jejich oprav

Výpis 5.3: Text na výstupu aplikace

*Svět se bojí nestability v Egyptě Írán mluví o islámském probuzeni. Zatímco běžní Egypťané slaví v ulicích páteční odstoupení prezidenta Husního Mubaraka světové mocnosti i okolní státy jsou opatrnější a připomínají že zemi čeká ještě dlouhá cesta. Americky prezident Barack Obama pro jehož zemi je Egypt klíčovým spojencem v regionu vyzval Egypťany aby v zemi byla nastolena skutečná demokracie zároveň však připomněl že mnoho otázek zatím zůstává bez odpovědí. Spojené státy se stejně jako sousední Izrael obávají zejména růstu vlivu islamistů. Nejmenovaný činitel americké vlády řekl televizní stanici CNN že v Egyptě nyní bude následovat "nepředvídatelná další kapitola". Egypt je první arabská země která v roce 1979 uzavřela mírovou dohodu s Izraelem a má s Tel Avivem relativně dobré vztahy. Nyní v Tel Avivu a Washingtonu panují obavy že nový egyptský režim bude k Izraeli rezervovanější. "Je důležité aby příští vláda mírovou dohodu ctila" konstatoval mluvčí Bílého domu Robert Gibbs. Íránský prezident Mahmúd Ahmadínežád hovořil v souvislosti s egyptskými protesty o "islámském probuzení" a připomněl i revoluci ve své zemi která v roce 1979 dostala k moci šíitské duchovní.*

Jak je vidět z výstupního textu, tak tento text neobsahuje interpunkci. Ta však na výstupu není vyžadována. Aby výstupní soubor obsahoval i interpunkci, je třeba provést určité úpravy v aplikaci. Text odpovídá článku č. 2 v tabulce 5.2. Byly zde objeveny a opraveny všechny zavedené chyby, čili v tomto případě se jedná o stoprocentní úspěšnost algoritmu. Je to dáno především díky tomu, že originální text, ze kterého opravovaný text vychází, byl použit pro vytvoření hlavního seznamu trigramů.

### 5.3.2 Ostatní texty

Kapitola 5.3.1 pojednává o výkonnosti aplikace, pokud ji aplikujeme na texty, jejichž gramaticky správné předlohy byly použity pro vytvoření hlavního seznamu trigramů, který aplikace používá pro detekci chyb. V této kapitole jsou obsaženy výsledky testů pro text, který nebyl použit k vytvoření hlavního slovníku trigramů.

| Chybné slovo | Oprava | Poznámka |
|---|---|---|
| průzracná | průzracná | špatně |
| těžko | těžko | správně, ale doplněn špatný trigram |
| rozzná | rozzná | špatně |
| predmětů | predmětů | špatně |
| vsecko | vsecko | špatně |
| něčim | něčím | správně, ale doplněn špatný trigram |
| ney | než | správně, ale doplněn špatný trigram |
| vdle | vdle | špatně |
| siuetu | siuetu | špatně |
| džící | džící | špatně |
| mýjí | mají | špatně – v textu je slovo *míjí* + doplněn špatný trigram |
| osamnělý | osamnělý | špatně |
| postavi | postavi | špatně |
| podezrelé | podezrelé | špatně |

Tab. 5.4: Přehled chybných slov a jejich oprav

K testování byl použit text z kapitoly 4.3. Tabulka 5.4 obsahuje výčet chybných slov a jejich opravu. V této tabulce je u každé opravy uvedena poznámka. Pokud je v této poznámce uvedeno „*doplněn špatný trigram*“ znamená to, že i když je dané slovo opraveno správně, tak jen za tu cenu, že byl do textu dosazen nesprávný trigram. Tím se text z hlediska kontextu stává nesmyslným a oprava pozbývá smyslu. Úspěšnost takové opravy je potom 0%.

Záměrně byl k tomuto testu použit text, který tematicky nezapadá do okruhu témat, ze kterého jsou vstupní data, která byla použita pro vytvoření hlavního seznamu trigramů. Pokud se trigramy z toho to textu doplní do tohoto seznamu, tak se účinnost opravy zvýší na 100% (otestováno).

Pro další test byla použita část článku ze serveru www.novinky.cz, který není součástí vstupních dat pro tvorbu hlavního seznamu trigramů. Bylo do něj zavedeno 7 chyb. Tato část článku je uvedena ve výpisu 5.4. Výpis těchto chyb spolu s jejich opravou je zobrazen v tabulce 5.4. Ve výpisu 5.5 je potom zobrazen opravený článek v neupravené podobě, v jaké je prezentovaný na výstupu aplikace. Při pohledu na tabulku 5.4 je vidět, že se správně opravily pouze 2 chyby ze 7i, což odpovídá úspěšnosti 28,5%. Takto nízká úspěšnosti bylo dosaženo především z důvodu řídkosti vstupních dat. Je tedy zřejmé, že pro korektní opravu je třeba použít rozsáhlejší seznam trigramů, což vyžaduje desetitisíce až milióny vstupních textů.

Výpis 5.4: Opravovaný text (originál zdroj www.novinky.cz)

*Novela zákona o veřejném pojistění, která umožňuje zavedení nadstandartní péče, se nelíbí ani některým koaličním partnerům. Projednávala se v pátek na zdravotním víboru Sněmovny a předseda výboru Boris Šťastný (ODS) uvedl, že předloha by mohla narazit u Ústavního soudu. Výbor proto jednání o novele přerušil do příštího pátku a očekává zmněny v návrhu. Poslankyně Soňa Marková (KSČM) se domnívá, že „bude vytvořeno dvojí zdravotnictví: pro ty, kteří na to mají, a pro ty, který na to nemají“. Části poslanců se také nelíbý vyší poplatek za den v nemocnici, který má stoupnout ze 60 na 100 korun.*

| slovo správně | zavedená chyba | druh chyby | oprava |
|---|---|---|---|
| pojištění | pojistění | vynechání diakritiky | pojistění |
| nadstandardní | nadstandartní | pravopisná chyba | nadstandartní |
| výboru | víboru | pravopisná chyba | víboru |
| změny | zmněny | pravopisná chyba | zmněny |
| kteří | který | nevhodné slovo | který |
| nelíbí | nelíbý | pravopisná chyba | nelíbí |
| vyšší | vyší | pravopisná chyba | vyšší |

Tab. 5.4: Výpis chybných slov a jejich opravy

Další problém z důsledku řídkosti dat nastane ve změně kontextu opravovaného textu na výstupu, jak ukazuje výpis 5.5. V tomto výpisu jsou žlutě označeny trigramy, které jsou dosazeny algoritmem aplikace. Samy o sobě jsou gramaticky správně, avšak do kontextu vět v článku se nehodí. Zde by opět pomohla větší databáze trigramů, ze které algoritmus vybírá vhodné kandidáty pro opravu chyb.

Výpis 5.5: Text na výstupu aplikace

*Novela zákona o veřejném pojistění která umožňuje zavedení nadstandartní péče se nelíbí ani některým koaličním partnerům. Projednávala se v pátek na zdravotním víboru Sněmovny i předseda výboru Boris Šťastný (ODS) uvedl že předloha by mohl narazit u Ústavního soudu. Výbor proto jednání o novele přerušil do příštího pátku a očekává zmněny v návrhu. Poslankyně Soňa Marková (KSČM) se domnívá že „bude vytvořeno dvojí zdravotnictví: pro ty kteří na to mají i pro to který na to nemají". Část poslanci se také nelíbí vyšší poplatek za den v nemocnice která má stoupnout ze 60 na 100 korun.*

Možná řešení tohoto problému jsou uvedena dále v kapitole 5.3.3. Při pohledu na výsledky lze říci, že aplikace se současnou databází není schopna uspokojivě opravit jiné texty, než texty vycházející ze vstupních dat.

### 5.3.3 Shrnutí výsledků

V kapitolách 5.3.1 a 5.3.2 byly provedeny testy výkonnosti algoritmu, se kterým pracuje aplikace.

V kapitole 5.3.1 byly k testování použity texty, které vycházely ze vstupních dat. I tak nebylo u všech 10i článků dosaženo očekávané úspěšnosti 100%. Je to především dáno funkcí bloku „Alternativy slov" v třetím stupni algoritmu (viz kapitola 5.1), jehož základ tvoří rozhraní pro generování navrhovaných slov pro opravu. Toto rozhraní u některých českých slov nenavrátí správný návrh opravy. To vede ke vzniku nesprávných trigramů, ze kterých algoritmus následně vybírá nejvhodnějšího kandidáta pro provedení opravy. Tím je dosaženo nesprávné opravy textu dosazením nevhodného trigramu, nebo ponecháním původní chyby bez opravy. Výše popsanému by se dalo předejít vhodnou konfigurací editačních vzdáleností, se kterými rozhraní počítá. Například distanční vzdálenost znaků *r* a *ř* (a analogicky i dalších podobných dvojic) je díky původnímu určení pro anglický jazyk větší, než například vzdálenost znaků *r* a *t.*

V kapitole 5.3.2 byly k testování použity texty, které nevyházely ze vstupních dat.

První z textů pocházel z odlišné oblasti, než vstupní data. Výsledek opravy tohoto textu naplnil očekávání, kdy se neočekávala dobrá úspěšnost algoritmu. Výstupní text byl plný chyb a v některých svých částech měl pozměněný kontext vět. Výsledek je dán odlišným tematickým okruhem, než ze kterého pochází opravovaný článek.

Druhý z textů pocházel ze stejné oblasti, jako vstupní data. Ale konečná úspěšnost opravy byla pouhých 28,5%. Za hlavní důvod této nízké úspěšnosti algoritmu se dá považovat řídkost vstupních dat, kdy bylo použito nedostatečné množství článků z jednotlivých oborů. Řešením tohoto nedostatku je zvětšení množiny vstupních dat. V takovém případě by ale u tematicky rozdílných textů mohly vznikat nesmyslné opravy, kdy by mohlo docházet ke změnám kontextu vět, protože se při výběru trigramu pro opravu textu vybírá ten z navrhovaných, který má nejvíce výskytů ve vstupních datech. Takový trigram může pocházet právě z tematicky odlišného článku. Z tohoto důvodu by bylo vhodné vytvořit různé databáze trigramu, kde by každá databáze byla tvořena trigramy z dat užšího tematického okruhu (internetová konverzace, obchodní e-mail, články o politice) a z těchto databází by se následně volilo. Tím by logicky vzrostla i úspěšnost opravy u prvního z textů.

# 6 Závěr

Jedním z cílů práce byl náhled na problematiku kontroly pravopisu v českých textech. Jsou zde popsány hlavní algoritmy, které se celosvětově používají pro kontrolu pravopisu. Jedná se především o fonetické algoritmy Soundex a Metaphone (resp. Double Metaphone). Ty jsou ale především určené pro anglický jazyk. Pro použití v českém jazyce je potřeba počítat s tím, že díky v nich obsaženým pravidlům pro anglickou výslovnost nemusí vždy vykazovat uspokojivé výsledky.

Aby fonetické algoritmy mohly vykazovat dobré výsledky v kontrole pravopisu, je nutné zbavit text překlepů. K tomuto účelu se používají algoritmy podobnosti řetězců, využívající metrických metod. Jednou z nejpoužívanějších metrických metod je výpočet Levenhsteinovy vzdálenosti. Mnoho metod je z této metody odvozených.

Z výše uvedeného je patrné, že ideální kontrolor pravopisu by měl obsahovat kombinaci fonetického a metrického algoritmu. Předešlé splňuje API Jazzy určené pro Javu, na kterém je postavený ukázkový program. Jak je vidět z výsledků, tak bylo pro český jazyk dosaženo uspokojivých hodnot. Toto prostředí bylo vybráno jako vhodný prostředek pro základ aplikace, která na základě kontextu věty sama detekuje a opraví chyby v textech.

Pro tuto aplikaci by vyvinut algoritmus, který na základě statistických údajů opravuje chyby v textech. Tento algoritmus pracuje na základě metody zpracování trigramů a jejich porovnání s databází. Díky této metodě ve výsledku nezaniká kontext věty a aplikace dokáže opravit širokou škálu chyb, jako například chyby vzniklé překlepem, chyby v psaní i/y a s/z, psaní velkých písmen a schodu přísudku s podmětem.

Jak ale ukázaly testy, tak je tento algoritmus silně závislý na objemu vstupních dat a dále na tematickém okruhu, ze kterého tato data pocházejí. Proto je pro jeho správnou funkci nutné vytvořit velké a tematicky odlišné databáze vstupních dat a mezi těmito databázemi vybírat podle tematického okruhu, ze kterého pochází opravovaný text. Tím se dosáhne větší úspěšnosti u oprav textů. Bez tohoto kroku nebude algoritmus účinný.

Aplikace je teprve v začátcích vývoje, jehož dalším stupněm by mohlo být propojení s SQL databází a implementace rozhraní pro kontrolu pravopisu, které je určené přímo pro český jazyk. Po těchto krocích by teoreticky měla stoupnout výkonnost celé aplikace o několik procent.

# Literatura

[1] **ČECHOV, A. P.** *Srdce nechodí samo.* Praha : Odeon, 1968.

[2] Daitch–Mokotoff Soundex. *Wikipedia, the free encyclopedia.* [Online] http://en.wikipedia.org/wiki/Daitch%E2%80%93Mokotoff_Soundex.

[3] Dynamic programming. *Wikipedia, the free encyclopedia.* [Online] http://en.wikipedia.org/wiki/Dynamic_programming.

[4] **EDMOND, J.** Soundex coding. *JewishGen.* [Online] http://www.jewishgen.org/infofiles/soundex.html.

[5] **FISCHER, H.** Soundex: Algorithm for phonetic search, Alternative to Soundex. *Sound-ex.* [Online] [Citace: 5. Listopad 2010.] http://www.sound-ex.com/alternative_zu_soundex.htm.

[6] **HNÁTKOVÁ, Milena**; **JELÍNEK, Tomáš**;. *Klasifikace a typologie chyb ve vstupních textech a koncepce značkování chybných textů* [online]. Liberec : Technická univerzita v Liberci, 2009 [cit. 2011-04-24]. Dostupné z WWW: <http://www.c2j.cz/attachments/075_Hnatkova-Jelinek-Petkevic_Chyby.pdf>

[7] **KNUTH, Donald E.** *The art of computer programming. Volume 3, Sorting and searching.* Reading, Massachusetts : Addison - Wesley, 1998.

[8] Levenshtein distance. *Wikipedia, the free encyclopedia.* [Online] http://en.wikipedia.org/wiki/Levenshtein_distance.

[9] **REANEY, P. H. a WILSON, R. M.** *A Dictionary of English Surnames.* Oxford : Oxford University Press, 1997.

[10] **SNAE, Ch.** A Comparison and Analysis of Name Matching Algorithms. [Online] http://www.waset.org/pwaset/v19/v19-47.pdf.

[11] Soundex Indexing. *National archives.* [Online] 30. Květen 2007. [Citace: 1. Listopad 2010.] http://www.archives.gov/publications/general-info-leaflets/55-census.html.

# Seznam použitých zkratek

API – *Aplication Programming Interface* – rozhraní pro programování aplikací
ICQ – foneticky *I seek you* – software pro instant messaging
IM – *Instant messaging* – internetová služba, která umožňuje posílání zpráv a souborů mezi uživateli
PC – *Personal komputer* – osobní počítač

# Přílohy

## Příloha 1 – Výpis zdrojového kódu hlavního (prvního) stupně algoritmu

soubor: TextLoader.java

```
package cz.stan.spellChecker;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.StringTokenizer;

import javax.swing.JFileChooser;

public class TextLoader {
	private String output = "";
	private String PATH;
	private String corrSen;
	public static boolean fileFlag;

	final static JFileChooser fc = new JFileChooser();

	public TextLoader() throws IOException {
		loadFile();
		getSentences();		// volání metody pro zpracování vstupu/výstupu aplikace
	}

	public void getSentences() throws IOException {
		if (fileFlag == true) {
			BufferedReader in = new BufferedReader(new InputStreamReader(
					new FileInputStream(PATH)));
			String line;
			String text = "";
			while ((line = in.readLine()) != null) {
				text += line;		// načtení vstupního textu
			}

			//rozdělení vstupního texu na věty a dobrání interpunkce
			StringTokenizer st = new StringTokenizer(text, ".");
			while (st.hasMoreTokens()) {
				String sent = st.nextToken();
				sent = sent.replace(".", "");
				sent = sent.replace("-", "");
				sent = sent.replace(",", "");
				// Změna prvního znaku na malé písmeno
				if(!sent.equals(" ")){
					String sentence;
					if (sent.substring(0,1).equals(" ")) {
						sentence = sent.substring(1, 2).toLowerCase() +
sent.substring(2);
					}else{
```

```
                                        sentence = sent.substring(0, 1).toLowerCase() +
sent.substring(1);
                                }
                                //Vvolání metody pro opravu věty
                                TrigramLoader tl = new TrigramLoader(sentence);
                                // Načtení opravené věty
                                corrSen = tl.getCorrectedSentence();
                        }

                        // Obnovení velkého počátečního písmena věty
                        String out;
                        if (!corrSen.isEmpty()) {
                                if(corrSen.substring(0,1).equals(" ")){
                                        out = corrSen.substring(1,2).toUpperCase() +
corrSen.substring(2);
                                }else
                                {
                                        out = corrSen.substring(0,1).toUpperCase() +
corrSen.substring(1);
                                }
                                // Kompletace výstupu
                                output += out + ". ";
                        }


                }
                System.out.println(output);
        }
        else
        {
                System.out.println("No file was selected.");
        }
    }
    // Metoda na otevření souboru
    private void loadFile() {
        fc.showOpenDialog(fc);
        if (fc.getSelectedFile() != null) {
                File file = fc.getSelectedFile();
                PATH = file.getAbsolutePath();
                fileFlag = true;
        } else {
                fileFlag = false;
        }
    }
}
```

# Příloha 2 – Obsah CD

Struktura souborového systému:

| /root | - | Aplikace | - | app | - | ContextSensitiveSC | - | aplikace |
|---|---|---|---|---|---|---|---|---|
| | I | | I | | I | | I | testdata |
| | I | | I | | I | | - | zdrojove_kody |
| | I | | I | | I | | | |
| | I | | I | | - | SpellCheckDemo | - | aplikace |
| | I | | I | | | | - | zdrojove_kody |
| | I | | - | lib | | | | |
| | I | | | | | | | |
| | - | Elektronicky_text | | | | | | |

**1.** *Aplikace* – složka data k aplikacím
- 1.1. *App* – složka obsahující aplikace
  - 1.1.1. ContextSensitiveSC – složka obsahující aplikaci popsanou v kapitole 5.
    - *1.1.1.1.* *Aplikace* – obsahuje spustitelný soubor aplikace.
    - *1.1.1.2.* *Testdata* – obsahuje testovací data a výstupy aplikace.
    - *1.1.1.3.* *Zdrojove_kody* – obsahuje zdrojové kódy aplikace.
  - 1.1.2. *SpellCheckDemo* – Obsahuje aplikaci popsanou v kapitole 4.
    - *1.1.2.1.* *Aplikace* – obsahuje spustitelný soubor aplikace.
    - *1.1.2.2.* *Zdrojove_kody* – obsahuje zdrojové kódy aplikace.
- 1.2. *Lib* – složka obsahující knihovny potřebné spuštění zdrojových kódů aplikací

***2.*** *Elektronicky_text* – složka obsahující elektronickou verzi diplomové práce