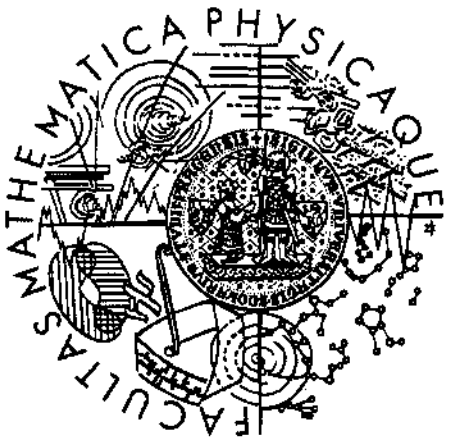
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Vladimír Zák

Automatické doplňování diakritiky pro slovenštinu

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Václav Novák
Studijní program: informatika, obecná informatika

2007

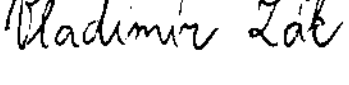
Poďakovanie

Na tomto mieste by som chcel poďakovať svojmu vedúcemu bakalárskej práce pánovi Mgr. Václavovi Novákovi za podnetné pripomienky, pozorné čítanie mojich výsledkov a vedenie.

Prehlasujem, že som svoju bakalársku prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičiavaním práce a jej zverejňovaním.

V Prahe 28. mája 2007

Vladimír Žák



Obsah

Abstrakt **6**

Úvod **7**

1 Základný popis projektu **8**
1.1 Diakritika 8
1.2 Doplňovanie diakritiky 9
1.3 Model 11
1.4 Zdroje štatistík a testovacích textov 12

2 Program **14**
2.1 Použité algoritmy a programovacie techniky 15
2.2 Package upravyTextu 15
2.3 Package upravyStatistik 16
2.4 Package prechodTextom 20
2.5 Package testy 21
2.6 Webová aplikácia 22

3 Užívateľské rozhranie **24**
3.1 Inštalačné DVD 24
3.2 Dávková aplikácia 25
3.3 Webová aplikácia 26

4 Testy a dosiahnuté výsledky **28**
4.1 Test parametra alfa 28
4.2 Porovnanie s programom Aviváž 29
4.3 Analýza chýb v doplňovaní diakritiky 30

Záver **32**

A Testovanie parametra alfa **33**

B Porovnanie s programom Aviváž **37**

C Ukážky zlého doplnenia diakritiky **39**

Názov práce: Automatické doplňovanie diakritiky pre slovenčinu
Autor: Vladimír Žák
Katedra (ústav): Ústav formálnej a aplikovanej lingvistiky
Vedúci bakalárskej práce: Mgr. Václav Novák
e-mail vedúceho práce: novak@ufal.mff.cuni.cz

Abstrakt

V predloženej práci popisujeme program na doplňovanie diakritiky do slovenských textov. Diakritika bola v slovenčine zavedená kvôli možnosti zapísať viac hlások, ako umožňujú pôvodne písmená latinskej abecedy. V informačných technológiách sa často z rôznych dôvodov vynecháva. Náš program sa zaoberá jej rýchlym doplnením do textov, v ktorých chýba. Používa na to štatistické metódy, pomocou ktorých sme vypracovali model na výpočet najpravdepodobnejšieho doplnenia diakritiky do každého slova. Pri výpočte pravdepodobnosti doplnenia daného slova pozeráme na výskyty samotného slova v textoch, ako aj na výskyty predchádzajúceho slova. V bakalárskej práci popisujeme aj programátorskú a užívateľskú dokumentáciu programu. Jeho testy ukazujú, že zvolená cesta je správna a v porovnaní s konkurenčnými programami dosahujeme lepšie výsledky.

Kľúčové slová: diakritika, doplňovanie diakritiky, unigram, bigram,

Title: Automatic Slovak Diacritics Reconstruction
Author: Vladimír Žák
Department: Institute of Formal and Applied Linguistics
Supervisor: Mgr. Václav Novák
Supervisor's e-mail address: novak@ufal.mff.cuni.cz

Abstract

In the following document we describe our program for adding graphic accents into slovak texts. Diacritic marks were established into slovak language to be able to capture more speech sounds in texts than historic latin alphabet would allow. In information technologies diacritic marks are often left out for various reasons. Our program is designed to quickly add them into texts where they are missing. The program uses statistical methods, with the help of which we created a model, according to which most probable diacritic marks are filled in each word. To calculate the probability of completion of the word, we consider the number of occurences of the word in texts, and also the occurences of the preceding word. Another things we decribe in the bachelor project are both user and programming documentation of our application. The tests show, that the selected method is appropriate and in comparison with competing products we achieve better results.

Keywords: diacritics, diacritics reconstruction, unigram, bigram

Úvod

V súčasnosti sa v digitálnej podobe nachádza veľké množstvo textov v slovenčine, ktorým chýba diakritika. Príčinami zámerného nepoužívania diakritiky sú: okolnosti vzniku informačných technológií, kompatibilita s existujúcimi systémami, čisto ľudské dôvody (návyk, jednoduchosť používania). Občas však každý potrebuje napísať čosi aj s diakritikou, preto sa niekedy hodí previesť rýchlo a jednoducho staršie texty bez diakritiky do správnej podoby. Presne na to slúži náš program.

Výsledkom práce na tomto projekte sú 2 aplikácie. Prvá, určená pre bežného užívateľa, je webová aplikácia, prístupná na internete cez ľubovoľný internetový prehliadač. Užívateľsky príjemne a intuitívne umožní klientovi zadať svoj text bez diakritiky, doplniť ho a vrátiť späť. Doplnenie môže prebiehať buď bez zásahu zvonka, alebo si užívateľ môže zapnúť možnosť interakcie pri istých typoch slov.

Druhá je dávková aplikácia, určená skôr pre testovanie programu, prípadne na dopĺňanie diakritiky do rozsiahlejších súborov. Ovláda sa pomocou príkazového riadka, cez ktorý sa programu zadávajú rôzne parametre.

Samotná bakalárska práca je rozdelená do 4 kapitol a niekoľko dodatkov. V prvej kapitole si bližšie charakterizujeme pojem diakritiky, čo to je, prečo bola do písma zavedená, v čom spočíva problém jej doplnenia. Ďalej sa ešte v tejto kapitole pozrieme na to, akým spôsobom sme sa my rozhodli riešiť samotné doplňovanie, priblížime si náš pravdepodobnostný model a ukážeme si, odkiaľ sme zohnali potrebné údaje o štatistikách slov v slovenskom jazyku.

Druhá kapitola je v podstate programátorskou dokumentáciou nášho projektu. Podrobne popisuje použité programátorské techniky, spôsob rozdelenia problému na časti a ich vzájomné prepojenie. Pozrieme sa aj na konkrétne Java balíky a triedy.

Tretia kapitola je zasa užívateľskou dokumentáciou. Popisuje, aké má užívateľ možnosti pri použití jednej, či druhej aplikácie. Zároveň vysvetľuje použitie priloženého DVD.

Posledná kapitola sa zaoberá testovaním programu a porovnaním s konkurenčným programami tohoto typu. Pozrieme sa tiež na to, akú celkovú úspešnosť môžeme očakávať od nášho programu a kde sa aplikácia najčastejšie mýli. Skúsime aj navrhnúť prípadné možné vylepšenia do budúcnosti.

Kapitola 1

Základný popis projektu

Úlohou nášho programu je doplniť diakritiku do slovenských textov. Pozrime sa na úvod našej práce, čo to vlastne diakritika je, v čom spočíva problém jej doplnenia a akými rôznymi spôsobmi to môžeme dosiahnuť.

1.1 Diakritika

Na zaznačenie ústnych jazykových prejavov používame dohodnutý súbor písmen (grafických znakov, grafém, litier), ktoré spolu označujeme pojmom písmo. Písmenami označujeme tie hlásky, ktoré dokážu rozlíšiť význam slov. Býva pravidlom, že označujeme jednu hlásku jedným písmenom, ale existujú aj výnimky z tohto pravidla (hlásku, ktorá znie ako 'ch' značíme 2 písmenami 'c' a 'h', podobne to je aj u hlások 'dz' a 'dž', naopak písmená 'i','y' označujú len jedinú hlásku). Niektoré hlásky sa označujú písmenami, ktoré obsahujú tzv. diakritické znamienka. Podľa [1] sú to tieto znamienka:

- **dĺžeň:** á, é, í, ó, ú, ý, ĺ, ŕ
- **mäkčeň:** č, š, ž, ť, ď, ň, ľ
- **dve bodky:** ä
- **vokáň:** ô

Diakritika je [2] ľudové označenie pre súbor diakritických znamienok. Slovo pochádza z gréčtiny (diakritikós) a pôvodne malo význam rozlišujúci. Iné pomenovania pre tieto znamienka sú: rozlišovacie znamienka, diakritické značky, rozlišovacie značky. Píšu sa v okolí samotného písmena (v slovenčine väčšinou nad písmeno). Ich úlohou je zmeniť výslovnosť písmen, prípadne celého slova, vyznačovať tón reči, rozlišovať homonymá (rovnako znejúce slová rôzneho významu), upozorňovať na skratky a číslovky. Bodka nad i a j nie je diakritickým znamienkom (nič neodlišuje, je neoddeliteľnou súčasťou písmena). Motív zavedenia diakritických znamienok je jednoduchý. Európske jazyky obsahujú [3] viac vysloviteľných hlások, ako je súbor základných 26

neakcentovaných (bez diakritiky) písmen. To je možné riešiť tak, že hlásky budeme vyjadrovať nielen pomocou jednotlivých písmen, ale aj ich kombináciou (napr. v nemčine skupina písmen tsch, ktorá sa vyslovuje 'č'). Druhý možný prístup je použitý práve v slovenčine (a mnohých ďalších slovanských a iných jazykoch), kde sa počet použiteľných písmen zvýši použitím diakritiky.

Pozrime sa ešte na ďalšie významy slova diakritický podľa Slovníka cudzích slov. [4]

1. slúžiaci na rozlíšenie, rozlišovací

2. gram. diakritické znamienko dĺžeň, mäkčeň a pod.

3. fyz. zodpovedajúci polovici kritickej hodnoty

V ďalšom texte budeme pod pojmom diakritika rozumieť už len sadu znamienok mäkčeň, dĺžeň, vokáň a dve bodky.

1.2 Doplňovanie diakritiky

Nepoužívanie diakritiky je pomerne nový fenomén, ktorý začína spolu s nástupom počítačov a hlavne internetu do bežného života. Kým na písomnú komunikáciu slúžili listy a rôzne ďalšie písomnosti, na vynechávanie diakritiky nebol žiadny rozumný dôvod. Príchod počítačov, internetu, e-mailovej komunikácie a pod. to zmenil z niekoľkých dôvodov:

1. Počítače vznikli v krajinách, kde sa akcentované znaky nepoužívajú (USA), nebol na ne spočiatku braný ohľad. Hlavný nástroj na zadávanie textu počítaču je klávesnica, tá je takisto na začiatku (a v podstate dodnes) prispôsobená najmä písaniu 26 pôvodných anglických znakov. Ostatné znaky národných abecied tam boli pridané až dodatočne. Stále je jednoduchšie písať len pomocou písmen bez diakritiky.

2. Text bez diakritiky je v slovenčine pomerne dobre zrozumiteľný. Existujú síce príklady, kedy nie je úplne jasné, o aké slovo sa skutočne jedná, ale to sa dá väčšinou vyčítať z kontextu.

3. Väčšina aplikácii je už síce dobre pripravená na podporu písmen s diakritikou, stále je však veľa takých, ktoré s tým majú problémy. Prípadne nie je štandardizovaný postup, ako s diakritikou pracovať a ako ju interpretovať.

V súčasnosti máme situáciu, kedy niektorí užívatelia internetu diakritiku používajú a niektorí nie. Prípad, že potrebujeme z nejakého textu diakritiku odstrániť, je triviálny. Definujeme si všetky akcentované písmená, ďalej na ktoré sa majú po odstránení diakritiky zmeniť, prejdeme text a všetky takto definované zmeny vykonáme (konkrétna implementácia bude uvedená aj v tejto práci 2.2). Zložitejší prípad je ten, keď z nejakého dôvodu potrebujeme do textu diakritiku doplniť.

Slovo môže obsahovať viac písmen, ktorým je možné diakritiku doplniť. Pri každom takomto písmene sú minimálne 2 možnosti, buď diakritiku doplníme, alebo nie. Niektoré písmená majú možných doplnení diakritiky dokonca viac. Toto všetko nám sťažuje rozhodnutie, ako správne diakritiku doplniť. Niektoré doplnenia, samozrejme, nemajú v danom jazyku (slovenčina) vôbec význam (napr. slovo 'klasika' má možné doplnenie na 'klásiká'). Pre naše potreby nemôžeme brať do úvahy len spisovné slová, pretože ľudia často používajú aj také, ktoré sú na hranici medzi spisovným jazykom a hovorovým štýlom, prípadne nárečím. Aj z tohto pohľadu majú niektoré slová len jedno možné doplnenie diakritiky (ako príklad uveďme slovo 'hladisko', kde jediné relevantné doplnenie je 'hľadisko'). Stále nám však zostáva veľa slov, kde je doplnenie nejednoznačné. Aj s prihliadnutím na kontext slov vo vete, prípadne viacerých viet, existujú príklady viet, kde sa nedá s úplnou istotou určiť, ktoré doplnenie je skutočne správne. Pekným príkladom môže byť nasledujúca veta: „Išiel s kozou na trh." Myslíme kozu, alebo kožu? Vidíme, že problém doplnenia diakritiky do textu nie je triviálny problém a v niektorých prípadoch je nájdenie správneho riešenia principiálne nemožné.

Na doplňovanie diakritiky existuje viacero rôznych metód. Medzi najjednoduchšie patrí taká, kde máme slovník, pomocou ktorého vieme každému slovu priradiť diakritiku. Nejednoznačnosť neriešime, len musí byť vždy určené, ktoré slovo sme si vybrali na doplnenie. Rozšírením tohto prístupu je vykonanie morfologickej, prípadne syntaktickej analýzy slov, resp. viet. To nám môže pomôcť práve pri tých nejasných slovách. Iné riešenie nám ponúkajú štatistické metódy. Pozrieme sa, s akou frekvenciou sa vyskytujú jednotlivé slová v slovenčine a diakritiku priraďujeme tak, aby sme dostali tie slová, ktoré sa vyskytujú najčastejšie. Táto metóda sa dá vylepšiť tým, že nebudeme prihliadať len na výskyt jednotlivých slov, ale aj na dvojice, či trojice slov (pokiaľ to má ešte štatistický význam).

Prístup v našom programe a bakalárskej práci je nasledovný. Pripravíme si model, kde budeme mať ku každému slovu bez diakritiky pripravené možné jeho doplnenia a zároveň ku každému doplneniu aj pravdepodobnosť tohto doplnenia. Podobne si takýto model pripravíme aj pre každú dvojicu slov [1] bez diakritiky. Tu bigramu a slovu, ktoré je v tejto dvojici druhé v poradí opäť priradíme možné doplnenie tohto slova a pravdepodobnosť takéhoto doplnenia. Postup, ako unigramu alebo bigramu prisúdiť pravdepodobnosť doplnenia bude podrobne popísaný v ďalšej kapitole 1.3. Ďalej sa musíme ešte rozhodnúť, ako z týchto dvoch pravdepodobností získame konečnú pravdepodobnosť, pomocou ktorej samotné doplnenie prevedieme. Keď už máme takéto modely pripravené, doplňovanie textu bude prebiehať takto: budeme brať postupne slová jedno po druhom a každému istým spôsobom prisúdime nejaké doplnenie diakritiky. Keď prejdeme celý text, máme vety s doplnenou diakritikou.

[1] V ďalšom texte budeme jednotlivé slová označovať zaužívaným pojmom **unigram**, dvojicu pojmom **bigram**.

1.3 Model

Celá teória okolo štatistického spracovania prirodzeného jazyka je podrobne rozpracovaná v [5]. Pre naše potreby sme použili časť 6. kapitoly. Pozrime sa najprv na unigramy. Majme z textu vytiahnuté ľubovoľné slovo bez diakritiky, ozn. ho napr. w. Pre toto slovo existujú rôzne možnosti doplnenia, tieto doplnené slová označme postupne w_1, w_2, ..., w_k. Pravdepodobnosť, akú majú jednotlivé doplnenia, zistíme z ich výskytov v jazyku. To znamená, že v obrovskom množstve textov v danom jazyku zistíme, ako často sa slovo w vyskytuje s nejakou konkrétnou diakritikou. Problémom získania takého množstva textov a tiež požadovaných počtov výskytov sa zaoberá ďalšia kapitola 1.4.

Predpokladajme teraz, že vieme rýchlo zistiť pre dané slovo w jeho celkový výskyt v našom modeli, ozn. funkciu, ktorá nám tento počet vráti $c(w)$. Ďalej si ešte zaveďme funkciu, ktorá nám pre dané slovo vráti toto slovo bez diakritiky, ozn. ju $o(w)$. Máme počty výskytov slov v nejakej množine textov. Z toho vieme podľa pravidiel pre výpočet pravdepodobnosti (viď napr. [6]), že pre pravdepodobnosť konkrétneho doplnenia w_i platí:

$$P(w_i) = \frac{c(w_i)}{\sum_{w_j : o(w_j)=o(w_i)} c(w_j)} \tag{1.1}$$

Náš model je vytvorený na základe veľkého množstva textov, predpokladáme teda, že dobre ukazuje výskyty jednotlivých slov. Zároveň očakávame, že v prípade nejakého nového textu sa toto rozdelenie bude blížiť očakávanému. Správnosť takéhoto prístupu zhodnotíme na konci práce, keď budeme vyhodnocovať úspešnosť nášho programu. Ak by sme mali určovať doplňované slovo len na základe unigramového modelu, doplňované slovo by bolo to s najväčšou pravdepodobnosťou podľa vzťahu 1.1. Maximum pravdepodobnosti pritom hľadáme cez všetky možné doplnenia slova w, ktorými sú w_1, w_2, ..., w_k. Označme nájdené slovo s najväčšou pravdepodobnosťou w_u, potom platí:

$$w_u = \underset{i \in \{1..k\}}{\operatorname{argmax}} P(w_i) \tag{1.2}$$

V prípade bigramov je situácia analogická. Majme teraz dvojicu slov bez diakritiky, označme ju $w_{a-1}w_a$ (znamená to, že slovo w_a nasleduje po w_{a-1} a medzi nimi je jeden zo štandardných oddeľovačov slov, v našom prípade najčastejšie medzera, prípadne ide o slová, kde jedno sa nachádza na konci jednej vety, kým ďalšie na začiatku druhej). Táto dvojica má rôzne možnosti doplnenia druhého slova (na prvé slovo v našom modeli neberieme ohľad, zostáva stále brané bez diakritiky), ktoré opäť označme w_1, ..., w_k. Pre pravdepodobnosť doplnenia dvojice $w_{a-1}w_a$ na $w_{i-1}w_i$ platí tentokrát vzťah:

$$P(w_i|w_{i-1}) = \frac{c(w_{i-1}w_i)}{\sum_{w_j : o(w_j)=o(w_i)} c(w_{i-1}w_j)} \tag{1.3}$$

Podobne ako pri unigramoch vyberieme z možných doplnení druhého slova bigramu to, ktoré má podľa 1.4 najväčšiu pravdepodobnosť, označme ho w_b. Maximum

hľadáme cez všetky možné doplnenia slova w_a, ktorými sú w_1, w_2, ..., w_k:

$$w_b = \underset{i \in \{1..k\}}{\operatorname{argmax}} P(w_i|w_{i-1}) \tag{1.4}$$

Máme teraz situáciu, kedy vieme ľubovoľnému slovu v texte bez diakritiky prisúdiť jeho najpravdepodobnejšie doplnenie, podobne každej dvojici slovo vieme prisúdiť doplnenie druhého slova. Potrebujeme skombinovať oba postupy. Vezmime si znova ľubovoľnú dvojicu slov $w_{a-1}w_a$. Slovu w_a vieme samotnému podľa vzťahu 1.1 prisúdiť pravdepodobnosti doplnenia rôznych diakritík, podobne pre celú dvojicu $w_{a-1}w_a$ vieme podľa 1.3 prisúdiť pravdepodobnosti doplnenia slova w_a. Musíme sa rozhodnúť, ktorému prístupu dáme prednosť. Označme základný parameter, ktorým dávame váhu na unigramy α, analogicky ten pre bigramy β. Oba parametre môžu nadobúdať hodnoty z intervalu $[0,1]$ a pre ich vzájomný pomer platí vzťah:

$$\alpha + \beta = 1 \tag{1.5}$$

Pre takto určené parametre α, β vieme vypočítať aj celkovú pravdepodobnosť doplnenia diakritiky slova w_a v dvojici $w_{a-1}w_a$, pre pravdepodobnosť doplnenia na $w_{i-1}w_i$ máme niekoľko možností:

1. Existujú aj doplnenia v unigramovom modeli aj bigramovom, potom spojením vzťahov 1.1, 1.3 a 1.5 dostávame predpis pre výpočet pravdepodobnosti doplnenia diakritiky konkrétneho slova w_a v našom modeli:

$$P(w_i|w_{i-1}) = \alpha P(w_i) + \beta P(w_i|w_{i-1}) \tag{1.6}$$

2. Pre doplnenie w_i neexistuje v bigramovom doplnení totožné doplnenie, pretože sme jednoducho dvojicu $w_{i-1}w_i$ v textoch vôbec nenašli. Tým pádom jej váha vo výsledku klesá na nulu a úpravou vyššie uvedeného 1.7 dostávame výsledok pre tento prípad:

$$P(w_i|w_{i-1}) = \alpha P(w_i) \tag{1.7}$$

Konečne teda pre danú dvojicu slov $w_{a-1}w_a$ vieme vypočítať pravdepodobnosť doplnenia konkrétnej diakritiky. My si z týchto doplnení vyberieme vždy tú s najväčšou pravdepodobnosťou podľa nášho modelu. Matematický zápis dostaneme pomocou 1.6, pričom výsledné slovo označme w_d. Opäť hľadáme maximum pravdepodobnosti doplnenia diakritiky cez všetky možné doplnenia slova w_a, ktorými sú w_1, w_2, ..., w_k:

$$w_d = \underset{i \in \{1..k\}}{\operatorname{argmax}} P(w_i|w_{i-1}) \tag{1.8}$$

1.4 Zdroje štatistík a testovacích textov

Na vybudovanie modelu tak, ako bol popísaný v predchádzajúcej kapitole 1.3, je potrebná rozsiahla databáza textov, aby sme pokryli čo najväčší rozsah daného

jazyka. Dôležitá je nielen veľkosť takejto databázy, ale aj to, aby pokryla široké rozpätie rôznych žánrov. Máme niekoľko možností ako postupovať.

Vo svojej práci som na vybudovanie modelu použil zdroje Slovenského národného korpusu (SNK). [7] Korpus všeobecne je rozsiahla databáza textov v danom jazyku. SNK nám priamo na svojich stránkach poskytuje rôzne prídavné informácie a štatistiky. Pre naše potreby sú najdôležitejšie práve tie, ktoré vypovedajú o frekvencii unigramov a bigramov v týchto textoch. Sú upravené tak, že na každom riadku je vždy jedno slovo, resp. dvojica slov a počet jeho (jej) výskytov v korpuse. Zotriedené sú podľa frekvencií. Texty, z ktorých sú tieto štatistiky, spĺňajú všetky potrebné požiadavky. Sú dostatočne rozsiahle (rádovo 340 mil. slov) a pokrývajú široké spektrum jazykových štýlov. Samozrejme, tieto informácie treba ešte ďalej upraviť do tej podoby, ako bolo popísané vyššie 1.3. Presný postup je popísaný v nasledujúcej kapitole, kedy si detailne rozoberieme štruktúru programu.

V prípade, žeby sme chceli takéto štatistiky pripraviť sami, postupovali by sme takto. Vzali by sme rozsiahle texty s diakritikou, z nich diakritiku odstránime a dostávame množstvo paralelných textov, ktorých porovnávaním zistíme početnosť jednotlivých doplnení diakritiky. Konkrétna implementácia by sa ešte musela vysporiadať s tým, že pracujeme s veľkými súbormi, takže by sme si museli dať pozor na pamäťovú a časovú náročnosť programu.

Kapitola 2

Program

Táto kapitola detailne popisuje program na doplňovanie diakritiky, použité algoritmy a programátorské techniky, konkrétnu implementáciu jednotlivých tried a ich použitie.

Výsledkom projektu sú 2 rôzne aplikácie. Prvá, budeme ju označovať dávková aplikácia, je spúšťaná pomocou príkazového riadku s rôznymi parametrami. Najdôležitejšími sú vstupný a výstupný súbor (ostatné parametre budú vysvetlené detailnejšie neskôr). Vstupný súbor je vo formáte čistý text, aplikácia tento text postupne prechádza a pokúša sa dopĺňať diakritiku k jednotlivým slovám. Slová sú dopĺňané pomocou nášho vytvoreného modelu 1.3. Doplnený text je potom uložený do zvoleného výsledného súboru.

Druhá aplikácia má charakter webovej aplikácie. Beží na webovom serveri a reaguje na požiadavky klientov, ktorí sa k nej pripájajú. Principiálne funguje podobne ako prvá, rozdiel je vo forme vstupných dát (zadávajú sa cez rozhranie webového formulára), vložení pomocných textových polí do výstupu (rôzne html tagy kvôli zobrazeniu v prehliadači) a forme výstupu (opäť ako webová stránka). Podobne aj celá komunikácia medzi užívateľom a aplikáciou prebieha cez webové rozhranie (zadávanie rôznych parametrov a pod.).

Program bol vyvíjaný v jazyku Java (verzia JDK 1.5.0_11, nájdeme ju napr. na [8]). Základy tohto jazyka sa môžeme naučiť napr. z [9]. Problémy pri práci s češtinou a slovenčinou v Jave je riešený v [10]. Dávkový súbor aj webová aplikácia boli vyvíjané pomocou IDE NetBeans (nájdeme ho na stránkach [11]). Webové stránky sú napísané v jazyku JSP (úvod do tejto technológie je spracovaný v [12]). Skúšky lokálneho spustenia webovej aplikácie som prevádzal pomocou kontajnera Tomcat 5.5 [13]. V nasledujúcich kapitolách sa najprv všeobecne pozrieme na použité programátorské postupy a následne si postupne prejdeme všetky balíky Java tried, ktoré projekt obsahuje. Zameriame sa pritom hlavne na logiku ich fungovania a menej na konkrétnu implementáciu.

2.1 Použité algoritmy a programovacie techniky

Jadro programu je jednoduché, prechádzame vstupný text (nech už sme ho získali akýmkoľvek spôsobom), rozdelíme ho na riadky, prípadne slová, jednotlivému slovu doplníme diakritiku a zapíšeme ho do výstupu. Jedná sa teda pri hrubom pohľade o jeden cyklus cez celý text. V prípade dávkovej aplikácie je interaktivita zahrnutá priamo do tohto jedného cyklu, v prípade webovej aplikácie si len pamätáme rozličné možnosti doplnenia nejasných slov (to je slov, pri ktorých nemáme dostatočnú istotu doplnenia), poskytneme užívateľovi nový formulár s týmito možnosťami, on si vyberie a následne získame v druhom cykle výsledný text.

Algoritmicky komplikovanejšou časťou projektu je práve doplnenie diakritiky jedného konkrétneho slova. Principiálne je riešené tak, že sa pozrieme na všetky možné doplnenia (možné z hľadiska daného jazyka) a z nich vyberieme to, ktoré má najväčšiu pravdepodobnosť (podľa vopred určených kritérií). Musíme mať teda pripravenú databázu slov (slovník) a pravdepodobnosť doplnenia týchto slov. Takúto databázu máme pre jednotlivé slová, ale aj pre dvojice slov (doplňame druhé z nich). Veľkosť týchto súborov je relatívne veľká (desiatky resp. stovky MB), preto si pre rýchle vyhľadávanie tieto súbory zotriedime algoritmom pre vonkajšie triedenie a vybudujeme nad nimi indexovaný súbor. Pomocou tohto indexu (v ktorom sa vlastne vždy nachádzajú položka a umiestnenie v súbore) sme potom schopní rýchlo vyhľadávať v primárnom súbore.

2.2 Package upravyTextu

Tento balík obsahuje jedinú triedu UpravyRiadka. Táto obsahuje množstvo statických metód, ktoré nám upravujú textový reťazec do požadovaného tvaru, prípadne vracajú nejakú jeho časť a pod. Najčastejšie v našom programe pracujeme s riadkami, preto je väčšina metód orientovaná práve na úpravu riadka.

Metódy s názvami typu *String prveSlovo(String riadok)* vezmú riadok a vracajú jedno konkrétne požadované slovo (podľa poradia). Oddeľovačom slov je skoro vždy medzera, až na pôvodný súbor s unigramami, kde je oddeľovačom tabulátor. Ten som však v ďalšom priebehu pretransformoval takisto na medzeru kvôli konzistentnosti aplikácie. Podobnú funkciu má väčšina metód v tejto triede, niektoré vracajú dvojice slov aj s medzerou za slovami, iné bez. Sú to všetko triviálne metódy nemá zmysel sa nimi úplne detailne zaoberať, využívajú zväčša metódu *String substring(String s)*. Za zmienku stoja ešte podobné metódy *String odstranDiakritiku(String riadok)* a *String odstranDiakritiku2(String riadok)*. Prvá zo zadaného reťazca odstráni diakritiku (dôležité je, že to, čo vlastne diakritika je, definujeme sami tým, že určíme, ktoré písmená sa prepisujú na ktoré, tým pádom môžeme vlastne určiť diakritiku daného jazyka). Druhá odstraňuje diakritiku len z prvého slova. To sa nám zíde v ďalších triedach, ako neskôr uvidíme.

2.3 Package upravyStatistik

Balík obsahuje triedy, ktoré nám postupne upravia štatistiky z SNK do požadovanej podoby a vytvoria potrebné vyhľadávacie štruktúry. Väčšina z týchto tried sa dajú spustiť aj ako samostatné aplikácie (najčastejšie majú parametre vstupný a výstupný súbor). Pozrime sa na jednotlivé triedy podrobnejšie.

Class ZmenaKodovania

Štatistiky získané zo SNK sú textové súbory uložené v kódovaní UTF-8. Aby sa nám s nimi v ďalšom priebehu jednoduchšie pracovalo, zmeníme toto kódovanie na také, ktoré už potom budeme môcť bez problémov ostatnými triedami čítať. Pracuje podobne ako mnoho ďalších tried, otvorí vstupný a výstupný súbor, zmení na znakovo orientovaný prúd, ktorý ešte zabalí do bufferovo orientovaného prúdu. Takto môžeme súbor čítať po riadkoch, čo nám značne urýchli prácu s veľkými súbormi.

Class ZotriedenieSuboru

Mimoriadne dôležitá trieda, ktorej hlavná metóda má za úlohu prevziať textový súbor (vstupný parameter) a zotriediť jeho riadky podľa abecedy. Triedené súbory sú veľmi veľké (rádovo stovky MB), preto je nutné použiť vonkajšie triedenie. Vzhľadom na to, že samotné triedenie nie je priamo súčasť bežiacej aplikácie, ale slúži len na jednorázovú úpravu databázy, nemusí byť úplne efektívny. Na jeho realizáciu som použil algoritmus Vonkajšieho triedenia uvedeného v publikácii [14], prepísaného, samozrejme, do jazyka Java z Pascalu. Táto trieda má viacero využití, ktoré zhrnieme v ďalších odsekoch.

Class OdstranenieDuplikatov, Class OdstranenieDuplikatovU

Predstavme si teraz situáciu, na začiatku vždy máme databázu textov vo formáte UTF-8, toto kódovanie zmeníme triedou ZmenaKodovania a výsledný súbor zotriedime. Nech už pracujeme s unigramami alebo bigramami, pri zmene kódovania nám zákonite musí dôjsť k tomu, že niektoré riadky sú identické až na počty výskytov (napr. v prípade unigramov sa tam nachádza riadok '? 23' ale aj '? 45'). To je podľa mňa spôsobené výskytom zvláštnych znakov v databáze, z ktorých niektoré sa pri zmene kódovania prepíšu na rovnaké symboly. Určite sa to však netýka normálnych bežných slov, ktoré nás zaujímajú najviac. Problém riešim jednoducho, v prípade identických unigramov, resp. bigramov vyhľadá trieda OdstranenieDuplikatovU, resp. OdstranenieDuplikatov ten riadok, kde má toto slovo najväčší výskyt. Do výsledku sa už zapíše len tento výskyt unigramu či bigramu.

Class MalePismena

Pomocná trieda, ktorej úlohou je zmeniť v súbore (daný ako parameter) všetky veľké písmená na malé. Výsledok ukladá do výstupného súboru (tiež daný ako parameter). Toto prevedenie urobíme potom, ako máme už databázu zotriedenú a vyčistenú od identických riadkov. Zmenu veľkých písmen na malé prevádzame kvôli tomu, že ušetríme časť priestoru, ako aj kvôli zjednodušeniu práce. Otázka, nakoľko tým ovplyvníme konečný výsledok doplňovania diakritiky, zostáva otvorená a je rozumné sa vrátiť k nej pri diskusii o dosiahnutých výsledkoch.

Class SpojenieDuplikatov, Class SpojenieDuplikatovU

Po použití triedy MalePismena sa nám opäť v databáze vyskytujú riadky, ktoré až na počet výskytov sú rovnaké. Ide o tie prípady, keď predtým sa tieto riadky líšili len veľkosťou písmen (samozrejme z hľadiska počítača sú a A úplne odlišné písmená). Tieto riadky treba tiež zlúčiť, ale na rozdiel od triedy OdstranenieDuplikatov to tieto triedy robia tak, že počty výskytov v riadkoch s rovnakým unigramami resp. bigramami spočítajú a do výsledného súboru uložia len konečný súčet.

Zhrnutie 1

Po postupnom použití hlavných metód tried ZmenaKodovania, ZotriedenieSuboru, OdstranenieDuplikatov, MalePismena, SpojenieDuplikatov na súbor s výskytom bigramov zoradeným práve podľa ich výskytu (systémom, že výstupný parameter jednej aplikácie je vstupným parametrom nasledujúcej), dostávame súbor (pracovne ho označme *bigramy_ciste*), ktorý nám bude nápomocný v ďalšom priebehu. Podobne vytvoríme aj súbor *unigramy_ciste*, len použijeme príslušné unigramové triedy OdstranenieDuplikatovU a SpojenieDuplikatovU. Na prevedenie týchto aplikácii môžeme použiť príslušný dávkový súbor, už v závislosti na operačnom systéme.

Class OdstranenieDiakritiky, Class OdstranenieDiakritiky2

Ďalšie z pomocných tried, prvá v súbore, ktorý je daný ako vstupný parameter príkazového riadku, odstráni akúkoľvek diakritiku, druhá odstraňuje diakritiku len v prvom slove. To má význam pri bigramoch, ako opíšeme v ďalších odstavcoch.

Zhrnutie 2

Vezmime si už vytvorený súbor *unigramy_ciste*, použime ho ako vstupný parameter OdstranenieDiakritiky. Dostaneme súbor s odstránenou diakritikou, ktorý zotriedime ZotriedenieSuboru, aby sme dostali opäť k sebe identické riadky až na výskyty. Použijeme SpojenieDuplikatovU a dostávame nový pomocný súbor *unigramy_bez*. Ten obsahuje výskyty jednotlivých unigramov bez diakritiky zotriedený podľa abecedy. Tiež sa nám bude neskôr ešte hodiť pri príprave konečnej štruktúry.

Podobným spôsobom (v podstate identickým až na prvotný vstupný súbor, ktorým je tentokrat *bigramy_ciste* a triedou, pomocou ktorej spájame duplikáty SpojenieDuplikatov), vytvoríme súbor *bigramy_bez*. Tu ešte potrebujeme jednu pomocnú štruktúru naviac, a to výskyty takých bigramov, zotriedených podľa abecedy, kde druhé slovo je s diakritikou a prvé nie. Použijeme teda postup OdstranenieDiakritiky2, ZotriedenieSuboru a SpojenieDuplikatov na vstup *bigramy_ciste*, dostávame pomocný súbor *bigramy_bez2*. Teraz už máme všetko potrebné na vytvorenie samotných vyhľadávacích štruktúr.

Class VytvorenieStrukturyU, Class VytvorenieStrukturyU2

Naším cieľom je vytvoriť štruktúru, z ktorej sa bude dať jasne vidieť, akú má daná diakritika pre konkrétne slovo pravdepodobnosť. Uveďme malý príklad (reálny): slovo sokom má 2 zmysluplné doplnenia a to 'sokom' a 'šokom'. V súbore *unigramy_ciste* sa dozvieme, že výskyt prvej možnosti je 110, kým druhej 617. V subore *unigramy_bez* sa zas dozvieme, že výskyt pôvodného slova bez diakritiky je 727. Vytvoríme nový súbor (*unigramy_struktura*), kde tieto informácie dáme k sebe. Vypočítať pravdepodobnosť doplnenia je potom už jednoduché, ako sme to ukázali v kapitole o vytváraní nášho modelu 1.3.

Náš cieľ dosiahneme v 3 krokoch. Trieda VytvorenieStrukturyU vezme ako vstupný parameter súbor *unigramy_ciste*. Prechádza ho po riadkoch, každému riadku odoberie diakritiku a zapíše [slovo bez diakritiky] [slovo s diakritikou] [výskyt danej diakritiky] [0]. Túto pomocnú štruktúru musíme zotriediť, aby sa riadky, ktoré majú na začiatku zhodné unigramy dostali k sebe. Následne použijeme triedu VytvorenieStruktury2, ktorá prechádza našu práve vytvorenú pomocnú štruktúru ako aj subor *unigramy_bez* a na miesta, kde sme predtým dali 0 doplňa skutočné počty s akou sa unigramy bez diakritiky v textoch nachádzajú. Výsledkom je súbor *unigramy_struktura* v požadovanom tvare, z ktorého sa dá rýchlo spočítať hľadaná pravdepodobnosť.

Class VytvorenieStruktury, Class VytvorenieStruktury2

Princíp vytvárania pomocnej štruktúry pre bigramy (*bigramy_struktura*) je podobný ako pri unigramoch. Naším cieľom je tentokrát dosiahnuť, aby v nej boli riadky v tvare [slovo1 bez diakritiky] [slovo2 bez diakritiky] [slovo2 s diakritikou] [výskyt daného doplnenia] [celkový výskyt bigramu].

Opäť potrebujeme 3 kroky. Použijeme triedu VytvorenieStruktury s vstupným parametrom *bigramy_bez2*, prejdeme súbor, odstránime diakritiku z každého riadka a zapíšeme v požadovanom tvare. Zotriedime tento výsledok a použijeme VytvorenieStruktury2 s prídavným parametrom *bigramy_bez* na získanie konečného výsledku.

Class SpojenieStruktur

Pomocné štruktúry (*unigramy_struktura* a *bigramy_struktura*), ktoré sme dosiaľ vytvorili, by nám už (ako ďalej uvidíme) stačili na vyhľadávanie pravdepodobnosti doplnenia daného slova. Problém je, že sú to 2 rôzne súbory, ktoré by sa tým pádom striedavo pre každé slovo využívali a v prípade, ak by boli nevhodne uložené na disku, veľmi by spomaľovali beh aplikácie. Riešením je ich spojenie do jedného veľkého súboru, pracovne nazvanom *struktura*, v ktorom budeme mať informácie tak o unigramoch ako aj o bigramoch. Inak je táto trieda jednoduchá, jej hlavná metóda najprv číta súbor *unigramy_struktura*, oddelí prázdnym riadkom a číta *bigramy_struktura*. Všetko priebežne zapisuje do už spomínaného pomocného súboru.

Zhrnutie 3

Výsledkom doterajšej našej popísanej práce je jeden výsledný súbor *struktura* (prípadne 2 súbory *unigramy_struktura* a *bigramy_struktura*), v ktorom máme v každom riadku uloženú informáciu o pravdepodobnosti, s akou sa daný unigram alebo bigram dopĺňa. Ako som už však viackrát spomenul, pracujeme s veľkými subormi (konkrétne tento má okolo 1 GB), prehľadávať pri každej požiadavke celý súbor sekvenčne od začiatku je nereálne, podobne aj jeho načítanie do pamäte do počítača. Riešením je vybudovať nad týmto primárnym súborom index, ktorý bude obsahovať odkazy do databázy. Pristupovať k súboru už musíme náhodne, nie sekvenčne. Index spravíme dostatočne malý, aby sa vošiel do pamäte, zároveň však nie príliš, aby sme nemuseli načítavať z primárneho súboru príliš veľké bloky dát.

Class VytvorenieIndexu, VytvorenieIndexuU, VytvorenieIndexuK2

Princíp vytvárania indexu si popíšeme na triede VytvorenieIndexuU (je najjednoduchšia). V metóde main otvoríme primárny súbor ako *RandomAccesFile* (s náhodným prístupom) a v cykle cez celý súbor prevádzame nasledujúce činnosti. Načítame si pomocou metódy *long getFilePointer*() odkaz na príslušné miesto v pamäti, načítame celý momentálny riadok metódou *String nacitajRiadok*(*RandomAccesFile f*), čítame po bytoch, koniec riadka nám signalizuje hodnota '\n'. Do výsledného indexu zapíšeme unigram a odkaz do primárneho súboru. Skočíme pomocou metódy *int skoc*(...) o daný počet bytov (hodnota tejto konštanty závisí od našej voľby, ovplyvníme tým hustotu odkazov a veľkosť indexu) a dokončíme riadok (metódou *int dokoncRiadok*(...)) na ktorom práve sme, aby sme pri ďalšom behu začínali na začiatku riadka. Takto dostaneme kompletný indexový súbor.

Dôležitou konštantou v tejto triede je *public static final int max_blok*. Rozhoduje o tom, koľko budeme v súbore skákať, teda aké budú vzdialenosti medzi 2 odkazmi. Hodnotu tejto konštanty je potrebné určiť experimentálne, ďalej závisí aj od toho či použijeme združenú vyhľadávaciu štruktúru *struktura*, alebo osobitné

unigramy_struktura a *bigramy_struktura*. Pre naše potreby som nakoniec zvolil hodnotu konštanty 500 (pre unigramy).

Podobne ako som to popísal vyššie vytvoríme index aj nad *bigramy_struktura* pomocou VytvorenieIndexu, so zvolenou konštantou velkost_bloku 10 000. Trochu odlišná situácia je pri vytváraní indexu nad struktura. Začiatok je podobný ako pri VytvorenieIndexuU, do výsledného súboru ukladáme unigramy a odkazy do primárneho súboru. V okamihu, keď zrazu zistíme, že sme už prešli do bigramovej časti štruktúry (jednoducho tak, že začiatky riadkov sú PRED tými ktoré už boli), nájdeme oddeľovač týchto 2 častí (prázdny riadok) a začneme pracovať so súborom ako trieda VytvorenieIndexu.

2.4 Package prechodTextom

Balík prechodTextom obsahuje 2 typy tried. Jeden typ sú triedy, ktoré nám umožňujú v už vytvorených štruktúrach nájsť pre dané slovo možné doplnenia a ich pravdepodobnosti. Druhým typom je trieda Hlavna, ktorá zastrešuje celú aplikáciu a tvorí užívateľské rozhranie pre aplikáciu spúšťanú z príkazového riadka.

Class NajdiVyskytK2

Postupne ako som vyvíjal aplikáciu, vzniklo viac tried s názvom NajdiVyskyt (specialne pre unigramy ci bigramy), funkčnosť aj účel všetkých je však podobná, preto popíšem len tú poslednú z nich, ktorá pracuje so súbormi *struktura* a *index*. Základné metódy, ktoré táto trieda poskytuje sú:

- String[] NajdiSlovaU(String slovo)
- double[] NajdiPravdepodobnostiU(String slovo)
- String[] NajdiSlovaB(String stare, String slovo)
- double[] NajdiPravdepodobnostiB(String stare, String slovo)

Ako už názov napovedá, ich úlohou je pre dané slovo, prípadne dvojicu slov vyhodiť pole String-ov (slov, ktoré znamenajú možné doplnenia diakritiky), prípadne pole double (pravdepodobnosti tohto doplnenia). Ich princíp je podobný, popíšme si napr. *String*[] *najdiSlova*(*String slovo*).

Pri inštanciovaní objektu triedy NajdiVyskyt má konštruktor 2 parametre. Ide práve o názvy súborov *struktura* a *index*. Musíme zistiť, koľko riadkov má index, vytvoríme polia Stringov polozka1 (unigramy), polozka2 (bigramy) a long odkaz1, odkaz2 práve takejto veľkosti (spolu). Do týchto polí si celý index načítame. V rámci konštruktoru ešte otvoríme primárny súbor. Keď už máme tieto prípravné práce za sebou pustíme sa do vyhodnotenia konkrétneho slova.

Metóda najdi má ako základ metódu *void napln*(*String slovo*). Tá volá 2 metódy, prvá je *void najdiOdkaz*(*Stringslovo*), ktorá pre dané slovo vyhľadá v indexe (čiže

v poliach kde je index načítaný), kde sa začína blok, v ktorom sa nachádza naše hľadané slovo. Druhou metódou je *void naplnPolia(String slovo)*. V tej najprv na začiatku skočíme v primárnom súbore tam, kde nás poslal už získaný odkaz, načítame do pamäti blok veľkosti velkost_bloku (je to práve tá veľkosť bloku, ktorú sme zadávali pri vytváraní indexu). Načítame aj malý úsek pred (kvôli tomu že sme mohli nejaké slovo tesne preskočiť) a za (pri vytváraní indexu vždy dočítavame do konca riadka). V takto načítanom úseku, ktorý je celý čas zotriedený podľa abecedy, hľadáme práve naše slovo. Tie riadky, kde sa na začiatku nachádza rozdelíme a získame z nich potrebné informácie. Tie uložíme do pomocných polí, ktoré potom spracúvajú už vyššie spomínané metódy.

Class Hlavna

Ako som už uviedol vyššie, trieda Hlavna zastrešuje všetky triedy a v aplikácii spúšťanej cez príkazový riadok tvorí aj užívateľské rozhranie. Tomu, ako aj rôznym parametrom, ktoré je možné zadať, sa budem venovať v užívateľskej časti, tu je dôležité, že jedným z parametrov je aj vstupný súbor, do ktorého textu budeme doplňovať diakritiku a výstupný súbor, kam tento text zapíšeme.

Konštruktor triedy nám otvorí vstupný a výstupný súbor a vytvorí inštanciu triedy NajdiVyskytK2, pomocou ktorej vyhľadávame info o jednotlivých slovách. Dôležitá je metóda *void prejdiSubor*(), ktorá berie riadok za riadkom a na každom riadku vykoná metódu *void najdiSlova*() a výsledok zapíše do výstupného súboru.

Metóda *void najdiSlova*() rozseká reťazec (v princípe riadok) na slová. Na tie, ktoré nie sú bielymi znakmi, prípadne interpunkčnými znamienkami, zavolá metódy *void vyhodnotSlovo*() a *void upravVelkost*(). Druhá z nich slúži len na to, aby sme slovo, ktorému už prisúdime nejakú diakritiku, upravili do pôvodného stavu, čo sa týka veľkosti písmen (pri hľadaní musíme každé slovo zmeniť tak, aby v ňom boli len malé písmená, keďže tak sme vytvárali všetky štruktúry).

Prichádzame k metóde *void vyhodnotSlovo*(), tá si na dané slovo, prípadne dvojicu slov(pamätáme si staré doplňané slovo) zavolá metódy z triedy NajdiVyskytK2. Ich výsledky si uložíme do pomocných polí. Teraz pomocou *void prejdiMoznosti*() určíme, ktoré doplnenie má najväčšiu pravdepodobnosť a nastavíme toto slovo do výstupu. Konkrétne počítanie pravdepodobnosti pomocou nášho modelu má na starosti metóda *double spocitajPravdepodobnost(int index)*, problematike jej počítania sme sa už venovali v kapitole 1.3. Interakciu užívateľa zabezpečuje metóda *String spytajSa*(), ktorá v prípade nejasného slova ponúkne užívateľovi na výber všetky možnosti doplnenia.

2.5 Package testy

V tomto balíku nájdeme len jednu triedu PorovnajSubory. Jej úlohou je vziať 2 textové súbory a skontrolovať slovo po slove, či sa tieto súbory zhodujú. Ak sa nezhodujú, vyjadriť nejakým spôsobom mieru podobnosti súborov. Ďalej ešte vie

vypísať tie slová, v ktorých sa súbory líšia. Všetky tieto funkcie sa nám budú neskôr hodiť pri testovaní správnosti nášho programu.

V metóde main sa volá *double urciZhodu*(), ktorá na konci určí konečnú mieru podobnosti súborov, zároveň seká oba súbory na riadky a volá na nich metódu *void porovnajRiadky*(). Tá zase rozdeľuje riadky na slová a tie už priamo porovnáva. V prípade, že sa nezhodujú, nezvýši počet kladných prípadov a prípadne vypíše, v ktorom slove nastala chyba. Po prejdení oboch súborov spočíta *double spocitajPravd*() mieru podobnosti ako podiel zhodných slov a všetkých slov.

2.6 Webová aplikácia

Webová aplikácia síce principiálne robí to isté, čo aplikácia spúšťaná cez príkazový riadok, sú tam však niektoré významné rozdiely. Najdôležitejším je úplne iné užívateľské rozhranie. Je tu aj pozmenená funkcia triedy Hlavna, ktorá v tomto prípade slúži len na zastrešenie možností, ponúkaných dávkovou aplikáciou. Program je implementovaný pomocou technológie JSP, hlavne kvôli ľahkému začleneniu Java kódu. Webová aplikácia teda zúročuje všetky možnosti, ktoré nám poskytuje už vyvinutý program. Zároveň poskytuje užívateľovi pohodlie rozhrania internetového prehliadača. Pozrime sa bližšie na štruktúru webových stránok a niektoré dôležité triedy.

JSP súbory

Úvodnou stránkou je *index.jsp*, ktorá poskytne užívateľovi základnú informáciu o programe a možnosť zvoliť si niektoré parametre aplikácie. Najdôležitejším prvkom stránky je formulár, kam užívateľ zadá text, ktorému chce doplniť diakritiku. Tento formulár sa odošle na spracovanie súboru *odpoved.jsp*, ktorý sa už potom stará o komunikáciu s užívateľom.

Stránka *odpoved.jsp* má 2 hlavné úlohy. Obsahuje javaBean *prechod*, ktorý nám umožňuje využívať metódy triedy Rozhranie (bude o nej reč neskôr) a vetví aplikáciu podľa užívateľových volieb. V prípade, že sme práve dostali text a nie je potrebná interakcia s užívateľom, doplníme diakritiku do textu a vrátime ju späť. Ak interakcia potrebná je, vrátime späť formulár, kde na mieste nejasných slov sú tagy select, ktoré umožnia užívateľovi zvoliť si tú správnu možnosť. V ďalšom kroku tieto voľby vyhodnotíme a vrátime späť ako celistvý text.

class Rozhranie

Táto trieda, ktorá sa vyskytuje len vo webovej aplikácii, je skutočne rozhraním medzi jsp súborom odpoved a možnosťami pôvodnej aplikácie. Obsahuje všetky potrebné metódy, *void vypis*() zavolá metódu triedy Hlavna *String mAplet*(*String text*), ktorá nám vráti text s doplnenou diakritikou (prípadne aj s html tagmi, popíšeme ju v ďalšom odseku). Metóda *void spojText*() vykonáva úlohu v tej vetve programu,

keď treba dať dokopy čiastočne doplnený text spolu s tým, čo si interaktívne zvolil užívateľ.

class Hlavna

Pracuje podobne ako pôvodná trieda, jej hlavnou metódou však nie je main (trieda nie je volaná samostatne), ale *String mAplet(String text)*, ktorá takisto doplní do textu diakritiku, ale text už nečerpá zo súboru, ale je to priamo parameter tejto metódy. Rozdiel je ešte v tom, že v prípade, že je povolená a potrebná interakcia, nepýta sa program hneď užívateľa na voľbu, ale do výsledného textu zahrnie html tagy select, ktorý sa potom zobrazí na jsp stránke.

Kapitola 3

Užívateľské rozhranie

V tejto kapitole si podrobne rozoberieme, aké možnosti ponúka užívateľovi náš program. V prvej časti sa pozrieme na dávkovú aplikáciu, ako ju nainštalovať na počítač a ako s ňou potom pracovať. V druhej sa pozrieme na bližšie na webovú aplikáciu.

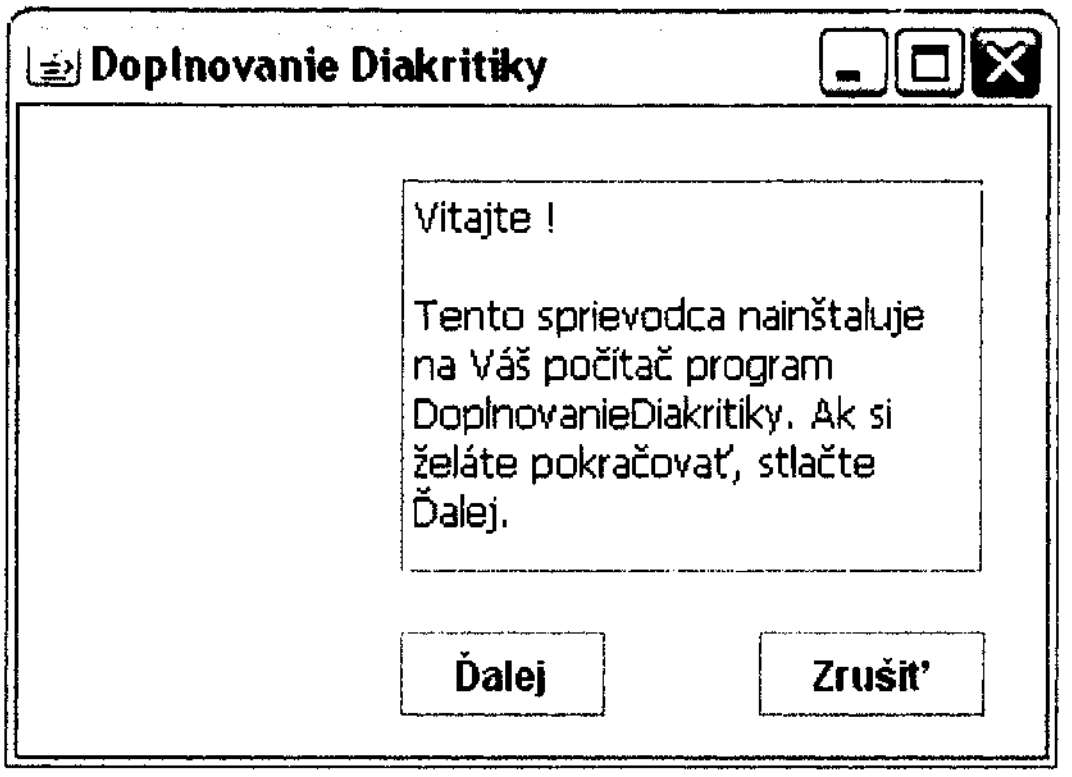
3.1 Inštalačné DVD

Na DVD, ktoré je pripojené k tejto práci, sa nachádzajú kompletné 3 projekty, ktoré sú v súvislosti s touto aplikáciou vyvíjané, vrátane zdrojových kódov, preložených class súborov, jsp súborov a javadoc dokumentácie:

1. **dopln**: Dávková aplikácia, ktorá je kompletne popísaná v predchádzajúcej kapitole. 2
2. **web**: Webová aplikácia, tiež je podrobne popísaná v tej istej kapitole. 2
3. **instal**: Pomocná aplikácia napísaná v Jave, ktorá pomáha užívateľovi preniesť potrebné súbory z DVD na jeho počítač.

Okrem toho sa na disku nachádza adresár texty, v ktorom sú vypracované databázy slov, ich možných doplnení a výskyty týchto doplnení. Tieto sú spracované podľa postupu opísanom v 2 a výsledkom sú 2 súbory struktura.txt a index.txt. Používajú ich dávková aj webová aplikácia.

Ďalej sa na DVD nachádza súbor citajMa.txt (v čistom textovom formáte pre jednoduchosť prehliadnutia na akejkoľvek platforme), ktorý v krátkosti užívateľovi predstavuje funkciu programu a popisuje aj priebeh samotnej inštalácie. Upozorňuje v prvom rade na nutnosť mať nainštalovanú nejakú verziu Javy (1.5, aspoň JRE, prípadne aj JDK). Samotnú inštaláciu vykonáva jar archív instal.jar, ktorý je buď spustiteľný priamo z DVD, alebo pomocou príkazového riadka. (Napr. v prípade, že sa nachádzame v priečinku DVD, je tento príkaz `java -jar instal.jar`. Úvodné okno ukazuje obrázok 3.1. Po odkliknutí sa nám zjaví podobné okno, kde budeme upozornení na to, že veľkosť prenášaných súborov je pomerne veľká a dostaneme



Obrázok 3.1: Pohľad na úvodné okno inštalačnej aplikácie, podáva úvodnú informáciu o programe a poskytuje možnosť pokračovať v inštalácii alebo jej zrušenie.

možnosť vybrať si adresár, do ktorého sa aplikácia nainštaluje. Po úspešnom skopírovaní súborov sa nám otvorí posledné tretie okno, ktorým sa inštalácia končí.

3.2 Dávková aplikácia

Keď už máme potrebné súbory skopírované na počítači, môžeme využívať funkcie programu na doplňovanie diakritiky. Tento sa spúšťa štandardným inštitútom príkazového riadka pomocou jar archívu dopln.jar. Predpokladajme teraz, že naším aktuálnym adresárom je práve ten, do ktorého sme nainštalovali našu aplikáciu. Jej spustenie potom prebieha nasledovne: `java -jar dopln.jar parametre`, kde *parametre* môžu mať nasledujúcu syntax:
`[-i] [-s texty] [-p prah] [-a alfa] vstupny_subor vystupny_subor`. Prejdime si ich teraz postupne podrobnejšie:

- **prepínač -i:** Týmto prepínačom dávame aplikácii najavo, že chceme, aby sa všetky slová pokúsila doplniť sama. To znamená, že každému slovu sa nájde najpravdepodobnejšie doplnenie (podľa nášho modelu) a to sa dá do výsledku.

- **prepínač -s:** Za týmto prepínačom nasleduje medzera a po nej cesta k suboru struktura (vytvorenej podľa popísaných pravidiel), nasleduje opäť medzera a cesta k suboru index (index nad struktura). V prípade, že tieto súbory nie sú vytvorené samotnou aplikáciou, je pravdepodobné, že ani sama aplikácia nebude fungovať korektne. Musia byť pripravené presne tak, ako je to popísané

v predchádzajúcej kapitole. Týmto prepínačom sa dá teoreticky (v prípade ďalšieho vývoja aplikácie) pomerne jednoducho rozšíriť program aj na ďalšie jazyky.

- **prepínač -p:** Nasleduje medzera a pravdepodobnostný prah v %. Ten vypovedá o tom, ako sa má aplikácia správať v prípade, ak je zapnutá voľba interakcie aplikácie s užívateľom. Vtedy sa pri každom slove pozrieme na všetky jeho možné doplnenia v našom modeli a pravdepodobnosti týchto doplnení (takisto v našom modeli). V prípade, že pravdepodobnosť doplnenia podľa nášho modelu je nižšia ako zadaný pravdepodobnostný prah, aplikácia sa užívateľa pýta na správne doplnenie a nedopĺňa slovo sama.

- **prepínač -a:** Nasleduje medzera a hodnota parametra alfa v %. Tento parameter nám určuje, akú váhu pri konečnom výpočte pravdepodobnosti daného slova má početnosť v rámci unigramov a akú v rámci bigramov (tá je vlastne 1 - alfa). Podrobnejšie sme sa problematike tohto parametra venovali v kapitole 1.3.

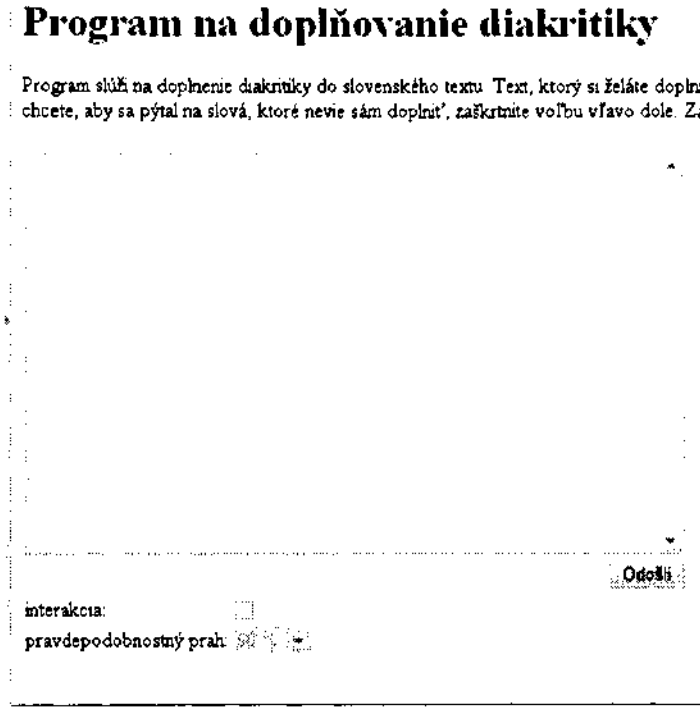
- **vstupny_subor:** Už podľa názvu ide o cestu k vstupnému textovému súboru. Tento by mal byť samozrejme bez diakritiky (diakritiku chceme dopĺňať), ale občasný výskyt diakritiky beh aplikácie neohrozí. Podmienkou správneho behu aplikácie je však skutočnosť, aby súbor, ktorý užívateľ zadal, naozaj existoval.

- **vystupny_subor:** Cesta k výstupnému súboru, v ktorom bude doplnená diakritika textu vo vstupnom súbore. Tento súbor samozrejme existovať nemusí, v prípade, že už existuje, bude prepísaný.

3.3 Webová aplikácia

Aplikácia beží na zvolenom webovom serveri, po zadaní správnej adresy sa nám zobrazí úvodná jsp stránka index.jsp (3.2). Tu má užívateľ na výber niekoľko možností. Do zobrazenej oblasti (TextArea) zadáva svoj text, ktorému chce doplniť diakritiku. V prípade, že si želá sám dopĺňať nejasný text, zaškrtne dole checkbox označený popisom interakcia. Zároveň si v choice zvolí pravdepodobnostný prah, ktorý má rovnaký význam, ako v dávkovej aplikácii. Tieto voľby si aplikácia stále pamätá, na každej ďalšej stránke má užívateľ právo tieto voľby zmeniť.

Po odoslaní formulára na server pomocou tlačidla s nápisom odošli sa zavolá stránka odpoved.jsp. Táto pri každom prechode skontroluje, v ktorej vetve sa nachádza a podľa toho zobrazí stránku v prehliadači. Prvá možnosť je, že nie je potrebná alebo nie je požadovaná diakritika. V tomto prípade sa zobrazí text s doplnenou diakritikou a možnosť zadania ďalšej požiadavky. Ak z nejakého dôvodu interakcia potrebná je, zobrazí sa neúplne doplnený text spolu s možnosťami (select) nápravy. Po odoslaní takého formulára sa stránka odpoved.jsp dostáva do poslednej vetvy, kedy spojí už doplnený text s voľbami užívateľa a zobrazí výsledok. V každom kroku má užívateľ možnosť meniť parametre aplikácie podľa svojich potrieb.



Obrázok 3.2: Pohľad na úvodné okno webovej aplikácie, podáva úvodnú informáciu o programe, dáva možnosť užívateľovi vložiť prvotný text a zmeniť parametre doplňovania diakritiky.

Kapitola 4

Testy a dosiahnuté výsledky

V tejto kapitole popíšeme, ako sme testovali funkčnosť samotného programu. Pozrieme sa na hľadanie optimálneho parametra $alfa$, t.j. miery, akú prikladáme v našom modeli unigramom a akú bigramom. Ďalej sa pozrieme na testy úspešnosti celého programu, porovnáme program s jeho konkurentom a pozrieme sa na najčastejšie chyby, ktorých sa dopúšťa a pokúsime sa prípadne navrhnúť nejaké vylepšenia.

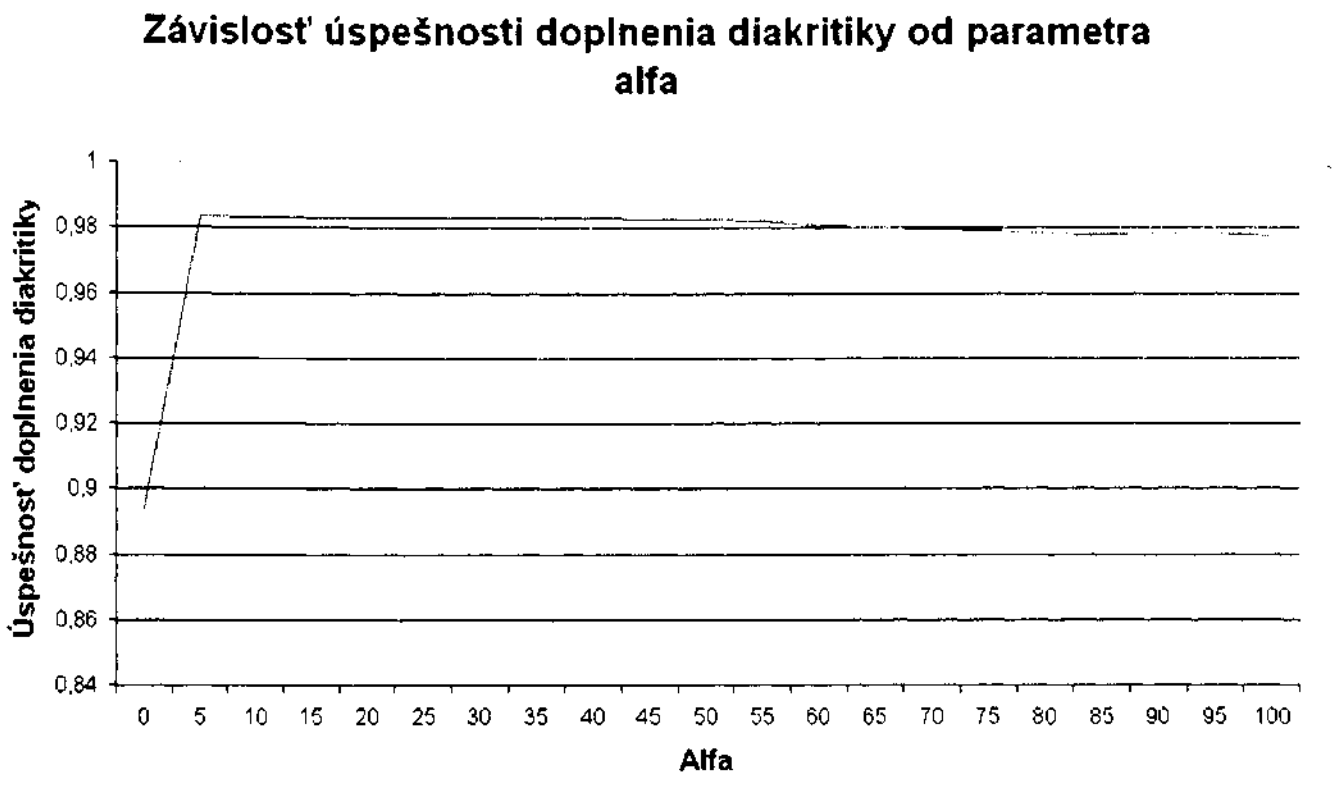
4.1 Test parametra alfa

Význam parametra $alfa$ sme si vysvetlili v kapitole, kde sme popisovali náš model 1.3. V krátkosti ide o to, akú váhu prikladáme pri výpočte pravdepodobnosti údajom, ktoré sme získali štúdiom samotných unigramov a akú pri štúdiu bigramov. Parameter môže nadobúdať hodnoty z intervalu $\langle 0,1 \rangle$.

Pri testovaní optimálnej hodnoty $alfa$ som využil 10 rôznych textov, ich zoznam je uvedený v dodatku A. Tieto pokrývajú rôzne žánre, ktoré sa používajú v písomnom styku, v približne takom rozsahu ako sú zastúpené v SNK. Z 10 textov je teda 6 publicistických (zväčša novinové články), 2 sú odborné a 2 sú umelecké.

Na každom z textov z textov som previedol nasledujúci postup: odstránil som diakritiku pomocou upravyStatistik.OdstranDiakritiku, vyskúšal spustiť dávkovú aplikáciu bez interakcie a postupným menením $alfa$ od 0 až do 1 po 0,05 (teda po 5 %). Nakoniec som získaný výsledok porovnal s pôvodným súborom pomocou testy.PorovnajSubory, čím som zistil úspešnosť doplnenia v danom prípade. Pre zautomatizovanie celého procesu som si napísal dávkové súbory test.bat a test2.bat pre prostredie príkazového riadka DOS. Tieto súbory sa takisto nachádzajú na priloženom DVD v časti testy. Výsledky si môžeme pozrieť v tabuľkách v dodatkoch (A).

Pre lepšiu prehľadnosť som zo získaných údajov vyrobil aj graf (4.1). Z uvedených výsledkov vyplýva, že úspešnosť doplňovania diakritiky s klesajúcim α stúpa, až na hodnotu parametra rovnú 0, kedy sa prudko zníži. To môžeme interpretovať tak, že použitie bigramového modelu v programe má zmysel a zvyšuje úspešnosť doplňovania diakritiky. Zároveň však nemôžeme úplne zavrhnúť unigramy, lebo pri ich ignorovaní sú výsledky značne horšie (napr. v našom prípade klesla úspešnosť o približne 10 %).



Obrázok 4.1: Graf nám ukazuje úspešnosť doplnenia diakritiky v závislosti od parametra alfa.

Pre potreby webovej aplikácie (kde si užívateľ nemá možnosť sám tento parameter ovplyvňovať), som sa preto rozhodol použiť hodnotu $\alpha = 0.1$.

4.2 Porovnanie s programom Aviváž

Tento program je jedným z riešení problému doplňovania diakritiky do slovenských textov. Je k dispozícii na internete a voľne šíriteľný, pôvodne vznikol podobne ako tento náš program ako študentská práca. Pre zaujímavosť som porovnal výsledky dosahované oboma programami.

Porovnanie som urobil podobne nasledovne. Zobral som 10 rôznych textov v slovenskom jazyku s diakritikou, ich zoznam nájdeme v B. Ich výber bol urobený podobne ako v predchádzajúcej kapitole. S každým súborom som potom previedol tieto činnosti. Odstránil som z neho diakritiku pomocou upravyStatistik.OdstranenieDiakritiky, doplnil diakritiku naším program aj programom Aviváž. Oba výsledné doplnené súbory som potom porovnal s pôvodným súborom s diakritikou. Dosiahnuté výsledky som kvôli prehľadnosti usporiadal do tabuľky 4.1.

Z výsledkov jednoznačne vyplýva, že náš program dopĺňa diakritiku s výrazne lepšou úspešnosťou. V každom z desiatich doplňovaných súborov bol lepší, najvýraznejšie pri doplňovaní odborného článku o fotosyntéze, najmenej pri rozhovore v novinách (jedine v tomto prípade rozdiel v úspešnosti klesol pod 10 %). V priemere

Text	Počet slov	Doplň	Aviváž	Rozdiel
1	446	0,991031	0,854260	0,136771
2	473	0,985201	0,871036	0,114165
3	974	0,967146	0,833676	0,133470
4	968	0,986570	0,897727	0,088843
5	523	0,990439	0,873805	0,116635
6	633	0,993681	0,867299	0,126382
7	623	0,975923	0,834671	0,141252
8	1198	0,971619	0,795492	0,176127
9	1468	0,989101	0,890326	0,098774
10	879	0,988623	0,835039	0,153584
priemer	**819**	**0,983934**	**0,855333**	**0,128600**

Tabuľka 4.1: Porovnanie nášho programu na doplňovanie diakritiky s programom Aviváž

sme dosiahli skoro o 13 % lepší výsledok ako tento konkurenčný program.

4.3 Analýza chýb v doplňovaní diakritiky

Na konci práce sa pozrime detailnejšie na slová, v ktorých robí program chyby. Tento pohľad nebude úplne presný, skôr nám ide o to, všimnúť si, kde asi program zlyháva a čo by sa s tým dalo prípadne spraviť. Použijeme texty z predchádzajúcej kapitoly (ich detailnejší popis sa nachádza v dodatku B) a triedu testy.PorovnajSubory, ktorá dáva možnosť nechať vypisovať slová, v ktorých sa súbory líšia. Pre ukážku som niektoré z týchto slov nechal vypísať do tabuľky v dodatku C.

Aj z tejto malej vzorky môžeme odhadnú určité typy chýb, ktoré programy nášho typu budú robiť vždy:

- **vlastné mená:** Slovník nikdy nemôže pokryť všetky vlastné mená (mená, priezviská, geografické názvy, ďalšie jedinečné pomenovania), ešte k tomu v rôznych svetových jazykoch. Ak sa v takýchto slovách vyskytuje diakritika, program ju nedoplní a tým pádom spraví chybu.

- **nesprávna a cudzie slová:** Opäť ide o slová, ktoré sa bežne v slovníku nevyskytujú. Môžu to byť chybné slová (vzniknuté nesprávnou gramatikou alebo preklepom), slová v cudzích jazykoch a pod. Výsledkom sú rovnaké chyby ako v predchádzajúcom prípade.

Okrem takýchto druhov chýb si však môžeme všimnúť ďalšie, ktoré už súvisia so samotným jazykom:

- **príd. mená vs. príslovky:** Dvojice slov typu pekné/pekne, dobré/dobre, voľné/voľne sa v slovenčine vyskytujú veľmi často. Frekvencia jedného, či druhého

z dvojice je už závislá na konkrétnom bigrame. Väčšinou sa nedá povedať, že jeden zo slovných druhov jednoznačne prevláda. Program teda v tomto prípade len tipuje správny výsledok a tieto prípady sú častým zdrojom chýb.

- **pods. meno vs. príd. meno:** Ide o podobný prípad ako vyššie, len sa zamieňajú dvojice typu Holandska/Holandská. Tieto prípady sú už menej časté a program už častejšie má indície na ktorú stranu sa pridať.

- **zámená:** Rôzne tvary zámen majú bez diakritiky rovnaké tvary, prípadne sa mýlia s inými slovnými druhmi, napr.: ťa/ta, má/ma. Aj tieto prípady sú pomerne častým zdrojom chýb.

Možností ako odstrániť chyby prvej kategórie (chýbajúce slová v slovníku) je niekoľko. Prvým je pokúsiť sa zostrojiť väčší slovník, ale v tomto smere sú možnosti obmedzené. Druhou je pozmeniť program tak, aby si v prípade neznámeho slova vypýtal jeho správnu diakritiku a túto si zapamätal. Pri vlastných menách sa pritom predpokladá len jedno korektné doplnenie, tým by sme sa vyhli tvoreniu rovnakých chýb v budúcnosti.

Chyby druhej kategórie (vyplývajú zo samotného jazyka), by sme mohli skúsiť riešiť pridaním trigramového modelu do nášho programu. Iným spôsobom by bolo pridať prvky morfologickej a syntaktickej analýzy, ale tieto úvahy už prekračujú rámec tejto práce.

Záver

Doplniť diakritiku do textu nebude nikdy možné na sto percent. Aj v reálnom živote môžu nastať prípady, kedy ani človek sám nie je schopný rozhodnúť, ako sa má dané slovo doplniť. Strojové doplňovanie robí principiálne ešte viac chýb. V našej práci sme sa pokúsili k ideálnemu prípadu aspoň priblížiť.

Na doplňovanie diakritiky sme použili štatistické metódy. Pozreli sme sa, ako často sa doteraz konkrétne doplnenie v textoch vyskytovalo a podľa toho sme sa rozhodli, ktoré použijeme v tomto prípade. Úspešnosť programu sme v porovnaní s podobnými programami zlepšili tým, že sme do úvahy brali aj štatistiky dvojíc slov.

Táto bakalárska práca završuje dlhodobú prípravu ročníkového projektu a samotného programu na doplňovanie diakritiky. Opisuje proces získania potrebných štatistík slov z SNK, ich úpravu do požadovaného tvaru, prípravu programov schopných rýchlo a presne hľadať v týchto databázach potrebné informácie o konkrétnom slove, či dvojici slov. Výsledkom sú 2 plnohodnotné aplikácie, schopné reálnej prevádzky s praktickým využitím.

Možnosti zlepšenia, samozrejme, stále existujú. Dá sa viac popracovať na grafickom vzhľade webového rozhrania aj dávkovej aplikácie. Podobne sa určite dá zlepšiť implementácia konkrétnych tried alebo metód, ktoré spracúvajú texty. Uvažovať sa dá aj o zmene logiky programu. Napríklad by sme rozlišovali malé a veľké písmená, interpunkčné znamienka a pod. Všetky tieto aspekty by mohli vylepšiť samotný program a tiež zlepšiť jeho výsledok, čiže zvýšiť úspešnosť doplňovania diakritiky.

Dodatok A

Testovanie parametra alfa

Všetky texty sa v plnom znení nachádzajú aj na priloženom DVD.

1. zdroj: http://zlatyfond.sme.sk/dielo/23/Kukucin_Rysava-jalovica/?&p=text&kap=1
   druh textu: umelecký, staršia (mierne zastaralá) slovenčina
   krátky popis: Prvá kapitola poviedky Martina Kukučína Rysavá Jalovica
   veľkosť: 6,5 kB

2. zdroj: http://www.sme.sk/c/3259822/Americania-rokuju-o-protiraketovom-stite-v-Europe.html
   druh textu: publicistický, novinový článok
   krátky popis: novinový článok zo správ o zahraničí (výskyt vlastných podstatných mien)
   veľkosť: 3,1 kB

3. zdroj: http://spravy.pravda.sk/kasicky-o-lietadlach-rozhodneme-tento-rok-fgx-/sk_domace.asp?c=A070423_093842_sk_domace_p12
   druh textu: publicistický, rozhovor
   krátky popis: novinový článok, rozhovor s ministrom obrany (priama reč, zvláštne zvraty politikov)
   veľkosť: 6,5 kB

4. zdroj: http://www.ta3.sk/news/
   druh textu: odborný, fyzika
   krátky popis: najnovšie správy z oblasti astrofyziky, SAV
   veľkosť: 20,2 kB

5. zdroj: http://ekonomika.hnonline.sk/c4-10142150-20962570-k01000_detail-do-roku-2009-sa-dan-na-jednu-cigaretu-zvysi-asi-o-60-halierov
   druh textu: publicistický, ekonomika

krátky popis: novinový článok o zvyšovaní spotrebných dní na cigarety
veľkosť: 3,5 kB

6. zdroj: http://www.plus7dni.sk/plus7dni/aktualne/za-oponou.html
   druh textu: publicistický, z domova
   krátky popis: článok o otvorení novej budovy SND
   veľkosť: 4,6 kB

7. zdroj: http://www.sharkan.net/800-nedelna-chvilka-poezie-miroslav-valek
   druh textu: umelecký, básne
   krátky popis: výber básní z tvorby M.Válka
   veľkosť: 3,9 kB

8. zdroj: http://www.sme.sk/c/3259470/Konopka-Po-teste-mam-vzdy-chvilu-obavy.html
   druh textu: publicistický, šport
   krátky popis: novinový článok o slovenskom guliarovi M.Konopkovi
   veľkosť: 6,9 kB

9. zdroj: http://www.bleskovky.sk/cl/10/154434/Prokurator-navrhol-Jozefovi-M-11-5-rocny-trest
   druh textu: publicistický, domáce správy, bulvár
   krátky popis: novinový článok o odsúdení veľkopodnikateľa za podvod
   veľkosť: 2,1 kB

10. zdroj: vlastná študentská práca
    druh textu: odborný, fyzika
    krátky popis: kapitola z článku o vlnovej mechanike
    veľkosť: 3,6 kB

Získané výsledky som kvôli prehľadnosti usporiadal do tabuľky:

alfa	text1	text2	text3	text4	text5
0 %	0,938552189	0,901826484	0,894945491	0,824694024	0,901380671
5 %	0,983164983	0,993150685	0,987115956	0,963642909	0,980276134
10 %	0,983164983	0,993150685	0,987115956	0,963642909	0,980276134
15 %	0,983164983	0,993150685	0,987115956	0,963642909	0,980276134
20 %	0,983164983	0,993150685	0,987115956	0,963642909	0,980276134
25 %	0,984006734	0,993150685	0,987115956	0,963282937	0,978303748
30 %	0,984006734	0,993150685	0,987115956	0,962922966	0,978303748
35 %	0,984006734	0,993150685	0,987115956	0,962922966	0,978303748
40 %	0,984848485	0,993150685	0,987115956	0,962922966	0,978303748
45 %	0,984848485	0,99086758	0,987115956	0,963282937	0,978303748
50 %	0,984848485	0,99086758	0,987115956	0,963282937	0,978303748
55 %	0,982323232	0,99086758	0,988107037	0,963282937	0,980276134
60 %	0,982323232	0,97716895	0,988107037	0,963282937	0,980276134
65 %	0,980639731	0,974885845	0,988107037	0,961843053	0,980276134
70 %	0,980639731	0,974885845	0,988107037	0,961843053	0,980276134
75 %	0,97979798	0,974885845	0,988107037	0,961483081	0,980276134
80 %	0,97979798	0,97260274	0,988107037	0,960403168	0,978303748
85 %	0,976430976	0,97260274	0,988107037	0,960403168	0,978303748
90 %	0,976430976	0,97260274	0,986124876	0,959683225	0,980276134
95 %	0,976430976	0,97260274	0,983151635	0,959683225	0,980276134
100 %	0,975589226	0,97260274	0,981169475	0,958603312	0,980276134

Tabuľka A.1: Prvá časť výsledkov testovania parametra alfa

alfa	text6	text7	text8	text9	text10
0 %	0,902406417	0,861870504	0,871428571	0,883495146	0,955911821
5 %	0,989304813	0,964028777	0,98877551	0,990291262	0,993987976
10 %	0,989304813	0,962589928	0,98877551	0,990291262	0,993987976
15 %	0,989304813	0,962589928	0,98877551	0,990291262	0,993987976
20 %	0,989304813	0,962589928	0,98877551	0,990291262	0,993987976
25 %	0,989304813	0,962589928	0,987755102	0,990291262	0,993987976
30 %	0,989304813	0,962589928	0,987755102	0,990291262	0,993987976
35 %	0,989304813	0,962589928	0,987755102	0,990291262	0,993987976
40 %	0,989304813	0,962589928	0,987755102	0,990291262	0,993987976
45 %	0,989304813	0,962589928	0,986734694	0,990291262	0,993987976
50 %	0,989304813	0,962589928	0,986734694	0,990291262	0,991983968
55 %	0,989304813	0,95971223	0,986734694	0,990291262	0,991983968
60 %	0,989304813	0,95971223	0,986734694	0,987055016	0,991983968
65 %	0,989304813	0,95971223	0,986734694	0,987055016	0,991983968
70 %	0,989304813	0,956834532	0,986734694	0,987055016	0,991983968
75 %	0,989304813	0,956834532	0,984693878	0,987055016	0,991983968
80 %	0,989304813	0,953956835	0,983673469	0,987055016	0,991983968
85 %	0,989304813	0,953956835	0,983673469	0,987055016	0,991983968
90 %	0,989304813	0,956834532	0,983673469	0,987055016	0,991983968
95 %	0,989304813	0,958273381	0,983673469	0,987055016	0,991983968
100 %	0,989304813	0,958273381	0,981632653	0,987055016	0,991983968

Tabuľka A.2: Druhá časť výsledkov testovania parametra alfa

Dodatok B

Porovnanie s programom Aviváž

Všetky texty sa v plnom znení nachádzajú aj na priloženom DVD.

1. zdroj: http://www.sme.sk/c/3306990/Slovensko-dostalo-kralovsku-navstevu-VIDEO.html druh textu: publicistický, domáce spravodajstvo krátky popis: novinový článok o návšteve holandskej kráľovnej na Slovensku veľkosť: 3,1 kB

2. zdroj: http://svet.hnonline.sk/c6-10028770-21178530-k03000_detail-eu-a-rusko-na-summite-vzajomne-rozpory-nevyriesili
   druh textu: publicistický, zahraničné spravodajstvo
   krátky popis: novinový článok o rokovaní EU z Ruskom
   veľkosť: 3,2 kB

3. zdroj: http://zlatyfond.sme.sk/dielo/67/Kalinciak_Restavracia/?&p=text&kap=2
   druh textu: umelecký, staršia slovenčina
   krátky popis: prvá kapitola z diela Jána Kalinčiaka Reštavrácia
   veľkosť: 5,3 kB

4. zdroj: http://www.pcrevue.sk/buxus_dev/generate_page.php?page_id=48528&buxus_itnews=11d47d1935e56dba260238136c129a7b
   druh textu: odborný, informatika
   krátky popis: článok o testovaní lacných notebookov
   veľkosť: 6,3 kB

5. zdroj: http://www.dennikspor t.sk/clanok?id=26306&IDvydanie=491
   druh textu: publicistický, šport
   krátky popis: novinový článok o otvorení nového štadióna vo Wembley
   veľkosť: 3,3 kB

6. zdroj: http: www.sme.sk c 3305830 Do-druheho-piliera-pribudnu-novi-sporitelia.html
   druh textu: publicistický, ekonomika
   krátky popis: novinový článok o vstupe absolventov do druhého piliera dôchodkového poistenia
   veľkosť: 4,5 kB

7. zdroj: http://www.hains.sk/jan%20smrek.html
   druh textu: umelecký, básne
   krátky popis: výber básní z tvorby slovenského básnika J. Smreka
   veľkosť: 3,6 kB

8. zdroj: http://sk.wikipedia.org/wiki/Fotosyntéza
   druh textu: odborný, biológia
   krátky popis: článok z wikipédie o fotosyntéze
   veľkosť: 8,7 kB

9. zdroj: http://www.sme.sk/c/3293878/Boris-Zala-Dennik-Sme-je-vyrazne-neobjektivny.html
   druh textu: publicistický, rozhovor
   krátky popis: novinový rozhovor s Borisom Zalom o aktuálnej situácii, krátený
   veľkosť: 9,1 kB

10. zdroj: http://www.sme.sk/c/3287510/Reporteri-TA3-obcas-musia-nasadit-aj-zivot.html
    druh textu: publicistický, zaujímavosti
    krátky popis: článok z časopisu o práci mladých autorov
    veľkosť: 5,8 kB

Dodatok C

Ukážky zlého doplnenia diakritiky

Por.	správne doplnenie	naše doplnenie
1.	zvolila holandská	zvolila holandska
2.	strane Dómu	strane Domu
3.	Košice Jana	Košice Jána
4.	summite vzájomné	summite vzájomne
5.	médiá ktoré	médiá ktore
6.	rokovaní nemecká	rokovaní nemecka
7.	súcasnosti predsedá	súcasnosti predseda
8.	povedala Ferreová-Waldnerová	povedala Ferreova-Waldnerova
9.	neuvarených neosolených	neuvarených neosolenych
10.	neosolených neomastených	neosolenych neomastenych
11.	klobúkom tuto	klobúkom túto
12.	mrdne obstrihaným	mrdne obstrihanym
13.	až ziskrí	až ziskri
14.	o húsencoch	o husencoch
15.	len má	len ma
16.	no pomysli	no pomyslí
17.	vtedy dlhy	vtedy dlhý
18.	to ta	to tá
19.	a pomysli	a pomyslí
20.	dnes páni	dnes pani
21.	i váš	i vás
22.	s daňou	s danou
23.	testu ultralacných	testu ultralacnych
24.	volia volne	volia volné
25.	aplikovat výkonnostné	aplikovat výkonnostne
26.	prinášame technický	prinášame technicky

Tabuľka C.1: Niektoré príklady chybného doplnenia diakritiky naším programom

Literatúra

[1] Kolektív autorov: Pravidlá slovenského pravopisu. Veda Vydavateľstvo Slovenskej akadémie vied, Bratislava, 1991, str. 19.

[2] Wikipédia: Diakritické znamienko - - Wikipédia, Slobodná encyklopédia, 2007. http://sk.wikipedia.org/wiki/Diakritika

[3] Blažek Filip: Diacritics Project, Typo.cz, 2005, http://diacritics.typo.cz/index.php?id=12

[4] Ivanová–Šalindová M., Maníková Z.: Slovník cudzích slov, Slovenské pedagogické nakladatelstvo, Bratislava, 1990, str. 208.

[5] Manning, C. D., H. Schütze: Foundations of Statistical Natural Language Processing, The MIT Press, 1999, str. 192-207.

[6] Zvára K., Štěpán J.: Pravděpodobnost a matematická statistika, MATFYZPRESS, Praha, 1997.

[7] Jazykovedný ústav Ľ. Štúra: Slovenský národný korpus, 2007, http://korpus.juls.savba.sk/

[8] Sun Microsystems: Java Technology, 2007, http://java.sun.com/

[9] Herout P.: Učebnice jazyka Java, nakladatelství Kopp, České Budějovice, 2006.

[10] Herout P.: Java – grafické užívatelské prostředí a čeština, nakladatelství Kopp, České Budějovice, 2006.

[11] Netbeans: Welcome to Netbeans, 2007, http://www.netbeans.org/

[12] Burd B.: JSP–Java Server Pages–Podrobný průvodce, Computer Press, 2003

[13] The Apache Software Foundation: Apache Tomcat, 2007, http://tomcat.apache.org/

[14] Töpfer P.:Algoritmy a programovací techniky, Prometheus, Praha, 1995, str. 202–213.