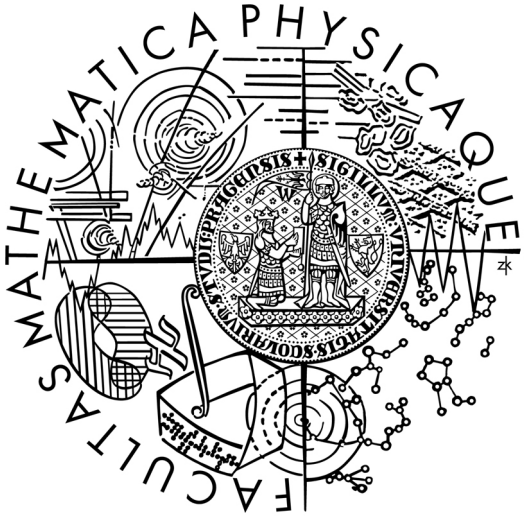
Univerzita Karlova v Praze
Matematicko – fyzikální fakulta

BAKALÁŘSKA PRÁCE



Jakub Záhumenský

Hľadanie odpovede v odpovediach

Ústav formální a aplikované lingvistiky

Vedúca bakalárskej práce : Mgr. Barbora Vidová Hladká, Ph.D.

Študijný program : Informatika, obecná informatika

2011

Týmto by som sa chcel poďakovať svojej vedúcej ročníkového projektu a bakalárskej práce Mgr. Barbore Vidovej Hladkej, Ph.D. za nekonečnú trpezlivosť, pomoc a podporu pri písaní tejto práce.

Prehlasujem, že som túto bakalársku prácu napísal samostatne, výhradne s použitím zdrojov uvedených v literatúre.

V Prahe dňa 17.5.2011 Jakub Záhumenský

Obsah

Prehľad obrázkov a tabuliek v práci

Název práce : Hľadanie odpovedí v odpovediach
Autor : Jakub Záhumenský
Kontakt : zahumensky.jakub@gmail.com
Katedra : Ústav formalní a aplikované lingvistiky
Vedúca práce : Mgr. Barbora Vidová Hladká, Ph.D.
Kontakt : hladka@ufal.mff.cuni.cz

Abstrakt : Témou tejto práce je navrhnúť a implementovať dialógový systém, ktorý bude simulovať rozhovor uživateľa s reálnou osobnosťou. Využívať budeme korpus reálnych rozhovorov zozbieraných z webových stránok. V implementácii budeme používať prístup vyhľadávania najpodobnejšej otázky v korpuse s otázkou uživateľa. Odpoveďou na uživateľovu otázku bude následne odpoveď na nájdenú najpodobnejšiu otázku z korpusu.
V práci budeme využívať morfologickú a syntaktickú rovinu jazyka, rovnako ako frekvenčnú analýzu pomocou tf-idf váh, na určenie najpodobnejšej otázky. Otázky budú zozbierané v korpuse, ktorého vytvorenie je súčasťou tejto práce. Konkrétne v tejto práci budeme zbierať rozhovory s významnou českou osobnosťou, Václavom Havlom.
Aplikácia bude pracovať s textami v českom jazyku.

Kľúčové slová : dialógový systém, tf-idf váhy, frekvenčná analýza, morfologická, syntaktická, tool_chain, Václav Havel, korpus

Title : Searching for the answer in answers

Author : Jakub Záhumenský
Contact : zahumensky.jakub@gmail.com
Department : Institute of Formal and Applied Linguistics
Supervisor : Mgr. Barbora Vidová Hladká, Ph.D.
Contact on supervisor : hladka@ufal.mff.cuni.cz

Abstract : We design a question-answering system Interviewer that enables users to fictionally (virtually) interview this person by asking questions as similar as possible to questions that journalists have already asked.
The interviews with a given person posted on the web are being collected as a corpus of (question, answer) pairs. The user asks his/her question and the Interviewer system searches questions in the corpus to provide the answer that belongs to the most similar question. Matching questions is based on the frequency analysis and on the applications coming from natural language processing, namely tagging and parsing.
We work with the interviews with Vaclav Havel posted on his personal page.

Keywords : dialogue system, tf-idf weights, frequentional analysis, morphological, syntactical, tool_chain, Václav Havel, corpus

Kapitola 1

Úvod

Dialógové systémy sa javia ako perspektívna časť informatiky. Ich využitie môže s vývojom informatiky rásť a v kombinácii s nástrojmi na rozpoznávanie reči môžu byť jednou z možností pre asistívne technológie, orientované na zlepšenie kvality života a pomoc pre hendikepovaných ľudí (najmä nevidomých).

V tejto práci sa jeden takýto systém, konkrétne dialógový systém simulujúci rozhovor s konkrétnou osobnosťou, pokúsime navrhnúť a implementovať. Na to je potrebné zozbierať korpus reálne uskutočnených rozhovorov, ktorý k tomu budeme využívať. Odpoveď na uživateľovu otázku budeme hľadať práve v tomto korpuse, pomocou nájdenia najpodobnejšej otázky k uživateľom zadanej otázke. Na otázku od uživateľa nebudeme klásť žiadne obmedzenia.

Teoretická lingvistika chápe jazyk ako systém rovín. My budeme v našej aplikácii využívať dve z nich – morfologickú a syntaktickú.

K implementácii tohto systému budeme využívať externú aplikáciu obsiahnutú v rámci Českého akademického korpusu 2.0 [10] s názvom ***tool_chain*** (kapitola 3.1.1). Táto aplikácia vstupný text zanalyzuje podľa morfologickej aj syntaktickej roviny jazyka a až s takto spracovaným textom potom bude naša aplikácia pracovať. Výstupom analýzy aplikácie tool_chain je anotovaný text vo formáte CSTS (bližší popis v kapitole 3.1.3).

Text tejto práce má nasledovné členenie.

V druhej kapitole zanalyzujeme zadanie, určíme si cieľ našej práce a popíšeme ako a prečo budeme postupovať.

Kapitola tretia obsahuje podrobný popis riešenia, ako aj popis nástroja tool_chain, ktorý je v našej práci veľmi podstatný. Obsahuje tiež popisy všetkých podstatných algoritmov a postupov, ktoré v našej práci budeme využívať. Konkrétne napríklad váhy tf x idf vo frekvenčnej analýze či algoritmus Zhang Shasha a výpočet tree edit distance v analýze syntaktickej.

Štvrtá kapitola obsahuje uživateľskú dokumentáciu aplikácie spolu s prehľadnými obrázkami z prostredia aplikácie.

V piatej kapitole uvádzame programátorskú dokumentáciu, v ktorej je uvedený prehľad súborov aplikácie, popis hlavných algoritmov a tiež najdôležitejších tried spolu s ich štruktúrami a metódami.

Šiesta kapitola patrí vyhodnoteniu výsledkov testovania aplikácia a ich zhodnotenie.

Posledná, siedma, kapitola je už záver, v ktorom sa zamyslíme nad výsledkami, ich odôvodnením a tiež návrhmi na zlepšenie.

Kapitola 2

Analýza zadania

2.1. Analýza zadania a vývoj dialógových systémov

Podstatnou vlastnosťou systémov sprostredkujúcich komunikáciu medzi človekom a strojom by mala byť ich uživateľská prívetivosť a jednoduchosť. Veľkou výhodou takéhoto systému je určite možnosť uživateľa vyjadriť sa prirodzeným jazykom, najlepšie svojim rodným. Nútiť uživateľa učiť sa nejakú striktnú predlohu zadávania otázok ho môže naopak odradiť od používania takéhoto systému.

Najzaujímavejšími reprezentantmi takýchto systémov sú podľa nás systémy, ktoré dokážu odpovedať na otázky uživateľa, takzvané *question-answering systems (QA)*. Prvopočiatky týchto systémov siahajú do 70-tych rokov, kedy boli vytvorené prvé systémy, ako napríklad cestovný asistent s cieľom rezervovať spiatočnú cestu, v experimente GUS [8].

Neskôr v 90-tych rokoch (presnejšie v roku 1992), vznikla iniciatíva Text REtrieval Conference (TREC)[1]. Je sponzorovaná National Institute of Standards and Technology (NIST) a Ministerstvom obrany USA a jej zámerom je podporovať výskum v oblasti information retrieval, poskytnutím infraštruktúry potrebnej na veľké evaluácie metodológií získavania informácií z textu.

Najnovším prírastkom do rodiny týchto systémov je v poslednom čase viackrát v médiach spomínaný počítač firmy IBM, Watson[2], ktorý dokázal poraziť ľudí v známej súťaži Jeopardy. Watson môžeme považovať za doteraz najlepší vyvinutý QA systém.

Aby sme nezabudli na podobné systémy využívajúce český jazyk, spomeňme dva z nich. Prvým a zároveň starším z nich je Text-and-Interference Based Approach to Question Answering system (TIBAQ , (Hajičová at al., 1995)), a druhým, mladším, je CL-SR Track vybudovaný na CLEF 2007 (Cross Language Evaluation Forum) (Češka,Pecina 200)

2.2. Teoretická lingvistika a roviny jazyka

Lingvistika ako vedná disciplína sa zaujíma o prirodzený jazyk. Delí sa do mnohých podoborov, z ktorých jedným je aj teoretická lingvistika. Tá skúma všeobecné princípy fungovania prirodzeného ľudského jazyka ako dorozumievacieho prostriedku. Stredom jej záujmu sú napr. morfológia, syntax, fonológia, fonetika atď.

Nás bude zaujímať ako teoretická lingvistika nahliada na jazyk. Ten je z jej pohľadu podľa Fukčního Generativního Popisu akýsi systém rovín. My z nich budeme využívať dve.

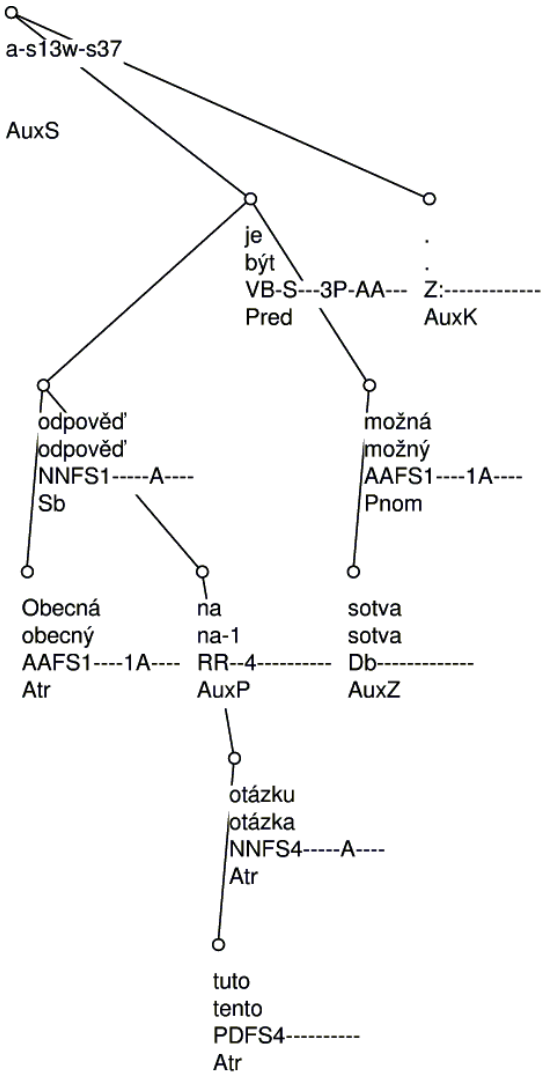
---

[1] *http://trec.nist.gov*

[2] *http://www-943.ibm.com/innovation/us/watson/*

Prvou z nich je morfologická rovina, ktorá charakterizuje vetu pomocou morfologických informácií o jej tokenoch. My túto rovinu budeme využívať v morfologickej analýze. Budeme však, kvôli významu vety a otázky ako takej,potrebovať aj slovné lemmata. Potrebujeme totiž nájsť významovo najpodobnejšiu otázku, čo by iba s využitím morfologických informácií nešlo. V tomto prípade by totiž slová *českým, americkým, generálním*, či *dobrým*, boli z tohto pohľadu rovnaké. Preto k morfologickým informáciám týchto slov budeme pridružovať aj lemma.

Druhou z rovín je syntaktická rovina, ktorá vznikne doplnením o vzťahy medzi tokenmi vo vete. To v zjednodušenej forme poznáme zo školy ako vetné členy. Túto rovinu budeme využívať v syntaktickej analýze, ale opäť, z rovnakých dôvodov ako pri morfologickej analýze, spolu s lemmami.



Obr.1. Závislostný strom vety „Obecná odpověď na tuto otázku je sotva možná.“ spolu s lemmami a morfologickými informáciami.

2.3. Cieľ práce

Na začiatok je určite vhodné si správne definovať ciele, ktoré v našej práci chceme splniť. V našom prípade sa budeme snažiť vymyslieť a implementovať algoritmus, ktorý by s využitím dostatočne obsiahlého korpusu rozhovorov dokázal nájsť vhodnú odpoveď na uživateľovu otázku.

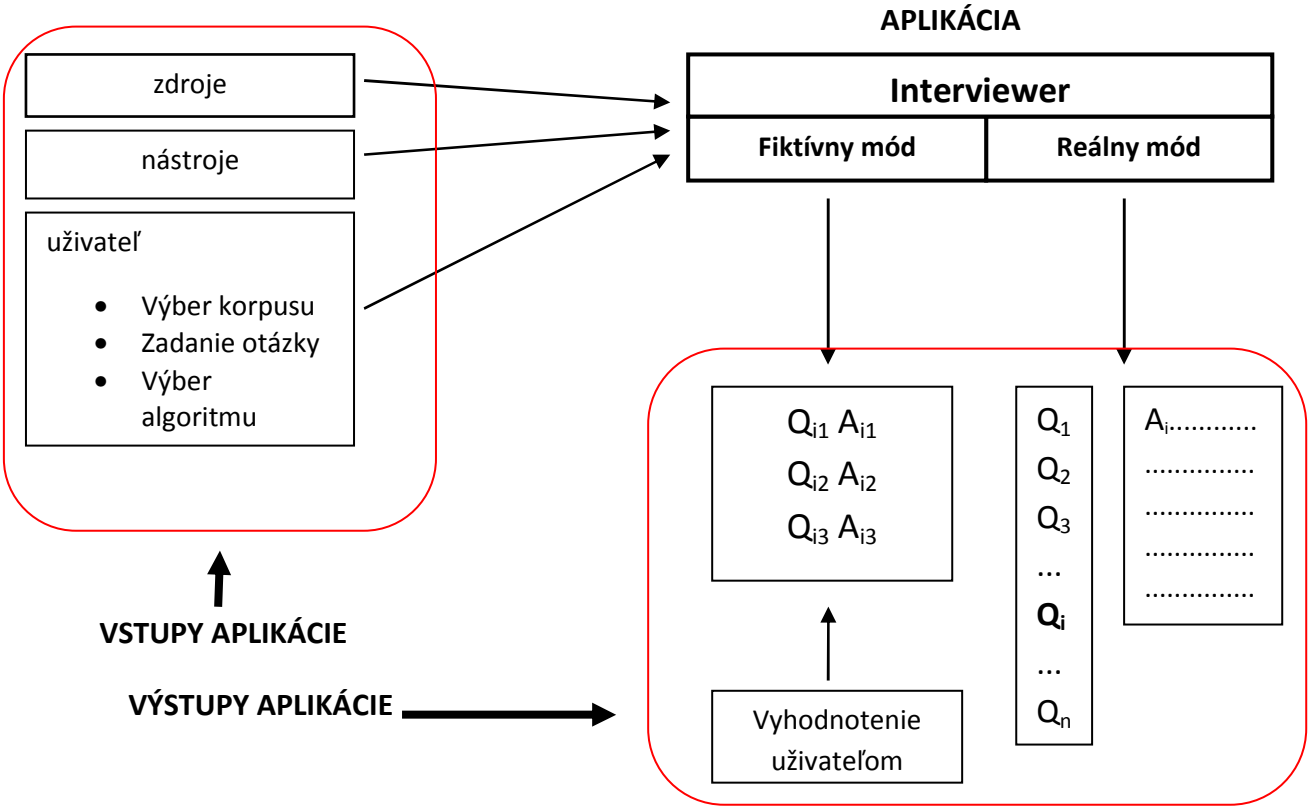
V našej práci budeme implementovať 3 rôzne procedúry, využívajúce rôzne roviny jazyka.

Prvou procedúrou bude frekvenčná analýza, ktorá nebude využívať žiadnu z rovín jazyka, ale iba štatistické informácie korpusu. Druhá procedúra, morfologická analýza, bude využívať rovnomennú rovinu jazyka, a takisto tretia z procedúr, syntaktická analýza, bude taktiež využívať rovnomennú rovinu jazyka. Bude nás zaujímať, ktorá z týchto procedúr bude subjektívne najpresnejšia a najvýhodnejšia. Subjektívne preto, pretože ťažko objektívne hodnotiť presnosť odpovede na otázku (hodnotiť správnosť odpovede, keď je známe, že napríklad politik sa dokáže vyjadriť tak, aby sa témy otázky ani nedotkol, by bolo zrejme scestné), najmä ak nebudeme mať k dispozícii dostatočne veľký súbor rozhovorov.

Okrem toho budeme zozbieravať rozhovory z rôznych dostupných zdrojov a budovať z nich čo najrozsiahlejší korpus. Konkrétne sme sa rozhodli pre rozhovory s Václavom Havlom, jednak preto, že je to významná osobnosť nielen českej, ale aj československej histórie, ale aj preto, že bolo uskutočnených a zverejnených pomerne mnoho rozhovorov s ním. Kvantita rozhovorov je samozrejme pre našu prácu rozhodujúcim faktorom. Čerpať budeme najmä z osobnej internetovej stránky pána Havla [11].

2.4. Náčrt aplikácie

Načrtneme predstavu aplikácie, ako by sme chceli aby fungovala, spolu s tým aké vstupy a výstupy budeme očakávať.



Obr.2 Náčrt fungovania aplikácie

Ako je vidieť na obrázku (*Obr.10)*, aplikácia očakáva 3 vstupy. Prvým je zdroj textov, teda korpus rozhovorov, s ktorým bude pracovať. Druhým vstupom je externý nástroj, v našom prípade už spomínaný tool_chain (sekcia 3.1.1.), ktorý nám pomôže so spracovaním textov a otázky od uživateľa. Tá je jedným zo vstupov, ktoré aplikácia potrebuje od uživateľa. Okrem nej musí uživateľ vybrať algoritmus, ktorým bude aplikácia vyhľadávať odpoveď a taktiež korpus, v ktorom bude vyhľadávať.

Aplikácia bude pracovať v 2 rôznych módoch – fiktívnom a reálnom. Fiktívny mód bude slúžiť na vyhľadávanie odpovede k otázke uživateľa. V reálnom móde si uživateľ bude môcť prehliadať všetky otázky a odpovede k nim v aktuálnom korpuse rozhovorov.

Z rozličnosti módov vyplýva aj rozličnosť výstupov aplikácie v oboch z nich. Vo fiktívnom móde bude výstupom programu trojica najpodobnejších otázok v korpuse spolu s odpoveďami na ne. S nimi môže uživateľ vyjadriť svoju (ne)spokojnosť, ktorá sa zapíše spolu s výsledkom vyhľadávania do súboru s log-om aplikácie. V reálnom móde sa uživateľovi zobrazí zoznam všetkých otázok v korpuse, po označení ktorejkoľvek z nich sa vo vedľajšom okne zobrazí odpoveď k nej.

Kapitola 3

Popis riešenia

Podstatou celej aplikácie je vyhľadať v korpuse rozhovorov vhodnú odpoveď k uživateľom zadanej otázke. To sa pokúsime dosiahnuť pomocou porovnávania otázok v korpuse s otázkou získanou od uživateľa a vyhľadania tej najpodobnejšej k nej. K tomu budeme využívať až 3 rôzne prístupy.

Podstatným predpokladom správneho fungovania je nástroj na spracovanie jazyka. Ten získame z Českého akademického korpusu 2.0 [7] [8]. Nástroj má názov tool_chain a pomôže nám súbor s rozhovormi zanalyzovať vo všetkých potrebných rovinách jazyka .

3.1. Definície a predpoklady

3.1.1. tool_chain

Tool_chain je skript obsiahnutý v rámci Českého akademického korpusu 2.0 [7] [8]. Je určený pre UNIXové systémy a jeho úlohou je automatické spracovanie českých textov. V našej aplikácii bude hrať dôležitú rolu.

Vstupom pre tool_chain je súbor s českým textom v kódovaní ISO-8859-2. Skript je možné spustiť s rôznymi prepínačmi, pre rôzne úrovne spracovania textu.
Hlavnými prepínačmi tool_chain pre spracovanie súboru sú :

-t . . . tokenizácia
-A . . . morfologická analýza
-T . . . tagovanie
-P . . . parsovanie

V našej aplikácii budeme využívať všetky tieto možnosti, tzn. budeme tool_chain spúšťať s parametrom –tATP. V praxi to znamená, že skript najprv prevedie **tokenizáciu** textu, teda rozdelí text na slovné jednotky a interpunkčné znamienka (ďalej pod spoločným názvom *tokeny*). Výstupom tokenizácie je takzvaná *vertikála,* čo je súbor, v ktorom je každé slovo alebo interpunkčné znamienko samostatne na zvláštnom riadku. Zároveň sa prevedie aj tzv. *segmentácia, č*o je označenie začiatkov odstavcov a viet. Tento súbor je navyše prevedený

do formátu CSTS (Czech Sentence Tree Structure) (kapitola 3.1.3), čo spočíva v pridaní hlavičky pred pôvodný obsah súboru a pridanie jednoduchých značiek k slovám, pomocou ktorých je potom možné odlíšiť slová začínajúce veľkými písmenami, interpunkčné znamienka, slová zapísané číslicami atď.

Výsledok tokenizácie, teda vertikála vo formáte CSTS, je následne vstupom pre ďalšiu časť spracovania textu - **morfologickú analýzu**. Pri nej dochádza k spracovaniu jednotlivých slovných foriem a určeniu takzvaných lemmat (napr. prvý pád singuláru pre podstatné mená, infinitív pre slovesá) a možné morfologické interpretácie.

Vzhľadom k veľkej homonymii českého jazyka (veľa slov, ktoré su homonymami – rovnako znejúcimi slovami s rôznym významom) môže byť jednému slovu priradených viacero morfologických značení – viacero reťazcov s morfologickou informáciou, ktoré teoreticky prichádzajú pre dané slovo do úvahy. Výsledkom morfologickej analýzy je množina všetkých takýchto do úvahy pripadajúcich morfologických informácií pre dané slovo.

O relatívne (vzhľadom k nie 100% presnosti) presné určenie správnej morfologickej značky sa stará **tagovanie** (alebo disambiguácia / desambiguácia) . To je úlohou taggera. Ten z v predchádzajúcej fáze určených dvojíc lemmat a morfologických značiek vyberá tie, ktoré by pre daný kontext mali byť správne. Tagger, ktorý používa tool_chain, pracuje na základe skrytých markovovských modelov s využitím priemerovaného perceptronu, čo je štatistická metóda.

Poslednou, no nemenej dôležitou, súčasťou tool_chain je takzvaný parser. Ten predstavuje ďalšiu úroveň spracovania textov, naväzujúcu na tagovanie, nazvanú **parsovanie**. Pri parsovaní sa každému slovu vety určí jeho syntaktická závislosť na inom slove vety a jeho analytická funkcia. Parser používaný v tool_chain je založený na rovnakej metodológii ako tagger. Jeho vstupom je text , ktorý je výstupom taggeru (obsahujúci práve jednu dvojicu lemma – morfologická značka). Výstupom parseru je stromová štruktúra ohodnotená analytickými funkciami.

3.1.2 Úspešnosť tool_chain

Súbor nástrojov zoskupených v programe tool_chain neurčuje potrebné informácie so 100% presnosťou. Úspešnosť našej aplikácie je tak závislá na úspešnosti tool_chain.

Od tool_chain budeme v programe potrebovať určiť *lemma*, teda základný tvar slova, z ktorého spracovávané slovo vzniklo (časovaním, skloňovaním ...). Rovnako budeme potrebovať aj morfologickú informáciu o danom slove. Takisto budeme potrebovať závislostný strom vety, ktorý nám dodá parser. Vo všetkých týchto úlohach nemusí tool_chain uspieť v plnej miere a môže dodať nekorektné výsledky.

Niektoré slová nedokáže tool_chain analyzovať vôbec. Takéto slová označí morfologickou značkou ***X@,*** s čím sa budeme musieť pri morfologickej analýze, kedy túto informáciu využívame, vysporiadať.

3.1.3 Formát CSTS

Formát CSTS (skratka pre Czech Sentence Tree Structure) je formát dát, ktoré budeme v našej aplikácii používať. S týmto formátom pracuje aj tool_chain. V CSTS sú všetky roviny anotácie uchovávane v jednom súbore (na rozdiel od novo používaného PML, kde je každá rovina anotácie uchovaná v samostatnom súbore). Používa sa tiež v Pražskom závislostnom korpuse a Českom národnom korpuse.

Súbor vo formáte CSTS pozostáva z hlavičky a samotných dát. Každý token v súbore je na samostatnom riadku. Začiatok vety je označený značkou **<s>.** Pre slová je vyčlenená značka ***<f>*** , pre interpunkciu značka ***<d>*.** Ďalej v každom riadku nasleduje anotácia slovného tokenu (*<f>*) na všetkých rovinách. Značka **<l>** označuje lemma tokenu, značka **<t>** obsahuje morfologickú značku vo formáte CPMT (The Czech Positional Morphological Tags). Tento formát predstavuje 15 znakový reťazec, ktorý na každej pozícii obsahuje informáciu o pevne určenej kategórii (viac v kapitole 3.3.1.). Nasleduje element **<r>**, ktorý je jednoznačnou identifikáciu tokenu v rámci vety, element **<g>**, určujúci riadiaci uzol tokenu a element **<A>**, v ktorom je uložená analytická funkcia, ktorú v zjednodušenej podobe môže čitateľ poznať zo základnej školy – napr. hlavné vetné členy prísudok a podmet, rozširujúce vetné členy predmet, prívlastok atď. Tieto tri informácie (označené značkami <r>, <g> a <A>) určujú závislostný strom vety, ktorý predstavuje syntaktickú informáciu. Každý takýto strom má navyše pridaný technický koreň, ktorý je nadradený slovám vo vete, ktoré nemajú žiadne nadradené slová. Takto spája všetky nezávislé vetvy do jedného stromu.

```
<s id=otazky3.tmp-001-p1s1>
<f id=otazky3.tmp-001-p1s1W1-Tm>Jaké<l>jaký<t>P4NS1----------<r>1
<g>3<A>Pnom
<f id=otazky3.tmp-001-p1s1W2-Tm>to<l>ten<t>PDNS1----------<r>2<g>
3<A>Sb
<f id=otazky3.tmp-001-p1s1W3-Tm>bylo<l>být<t>VpNS---XR-AA---<r>3
<g>0<A>Pred
<f id=otazky3.tmp-001-p1s1W4-Tm>být<l>být<t>Vf--------A----<r>4
<g>3<A>Pnom
<f id=otazky3.tmp-001-p1s1W5-Tm>českým<l>český<t>AAMS7----1A----
<r>5<g>6<A>Atr
<f id=otazky3.tmp-001-p1s1W6-Tm>prezidentem<l>prezident<t>
NNMS7-----A----<r>6<g>4<A>Pnom
<D>
<d id=otazky3.tmp-001-p1s1W7-Tm>?<l>?<t>Z:-------------<r>7<g>0
<A>AuxK
```

Obr.3 Veta „Jaké to bylo být českým prezidentem?“ vo formáte CSTS

3.1.4 Tf x idf váha

Tf x idf je skratkou slov *term frequency x inverse document frequency*. Často sa používa v information retrieval a text miningu, či vo vyhľadávačoch. Vyjadruje relevanciu slova vzhľadom k dokumentu v korpuse. Dôležitosť slova pre daný dokument podľa tf x idf rastie s počtom výskytov slova v dokumente a klesá s počtom dokumentov, ktoré toto slovo

obsahujú.

Pomocou tejto metódy priradíme každému termu v dokumente váhu, ktorá sa odvíja od frekvencie výskytu tohto termu v danom dokumente a frekvencie jeho výskytu v celej kolekcii dokumentov. Potrebujeme zistiť váhu medzi termom ***t*** v otázke uživateľa a dokumentom ***d*** (v našom prípade otázkou v korpuse. Najjednoduchšou možnosťou by bolo priradiť váhu totožnú s počtom výskytov termu ***t*** v dokumente ***d***. Tento prístup je označovaný ako $\boldsymbol{tf_{t,d}}$ (*term frequency*, kde indexy označujú term a dokument, ktorého sa váha týka).

Používanie iba term frequency ako váhy pre term v dokumente by však malo jeden základný nedostatok, a to, že všetkým termom v uživateľovej otázke by prináležala rovnaká dôležitosť. V skutočnosti však niektoré slová v otázke maju pre výsledok väčšiu dôležitosť ako ostatné. Napríklad, v kolekcii rozhovorov o politike je veľmi pravdepodobné, že slovo parlament sa bude v otázkach vyskytovať veľmi často. Na určenie dôležitosti termu sa tak používa takzvaná *document frequency* ($\boldsymbol{df_t}$), ktorá vyjadruje počet dokumentov (otázok), v ktorých sa vyskytuje term ***t***. Rovnako tak by sa dala použiť aj *collection frequency*, ktorá označuje počet výskytov termu v celej kolekcii dokumentov. Pre naše účely sa však viac hodí *document frequency* ako podrobnejší level štatistiky.

slovo	**cf**	**df**
try	10 422	8 760
insurance	10 440	3 997

Obr.4 Ukážka chovania sa collection frequency a document frequency v kolekcii dokumentov agentúry Reuters [5]

V tabuľke na *Obr.4* ilustrujeme ako sa správa collection frequency a document frequency v kolekcii dokumentov. Ako príklad použijeme kolekciu dokumentov agentúry Reuters. Ako je vidno v tabuľke, slovo *try* aj *insurance* majú takmer rovnakú collection frequency, ale ich document frequency sa líši vcelku výrazne. My však chceme málo dokumentom, ktoré obsahujú požadované slovo, priradiť väčšiu váhu, preto sa nám viac hodí document frequency.

K tomu budeme využívať *inverse document frequency*, ktorá sa pre term ***t*** v kolekcii s ***N*** dokumentmi vyráta podľa nasledujúcej formulky :

$$idf_t = \log \frac{N}{df_t}$$

Z nej vyplýva, že idf termu, ktorý sa v kolekcii nevyskytuje často, je vysoké, zatiaľ čo idf častého termu bude nízke. V nasledujúcej tabuľke opäť uvádzame príklad z kolekcie agentúry Reuters obsahujúcej 806 791 dokumentov :

slovo	df_t	idf_t
car	18 165	1.65
auto	6 723	2.08
insurance	19 241	1.62
best	25 235	1.5

Obr.5 Príklad správania sa document frequency a inverse document frequency v kolekcii dokumentov Reuters [5]

Na výpočet *tf x idf* skombinujeme oba tieto pojmy, čím vytvoríme vyváženú ohodnocovaciu funkciu pre všetky termy vo všetkých dokumentoch. Tf x idf priradí termu ***t*** v dokumente ***d*** váhu :

$$tf \times idf_{t,d} = tf_{t,d} * idf_t$$

Inými slovami*, tf-idf* priradí termu ***t*** v dokumente ***d*** váhu, ktorá :

1. je ***najvyššia***, ak sa term t vyskytuje často v rámci úzkeho okruhu dokumentov (týmto dokumentom teda prisudzuje silnú schopnosť vyčlenenia sa)
2. je ***nižšia***, ak sa term vyskytuje menej často, prípadne vo veľkom počte dokumentov
3. je ***najnižšia***, ak sa term vyskytuje v relatívne všetkých dokumentoch

Podrobnejšie informácie o *tf x idf* sa v prípade záujmu môže čitateľ dočítať na webových stránkach Stanfordskej univerzity [5] alebo v práci Gerarda Saltona a Christophera Buckleyho [3].

3.1.5. Výpočet podobnosti otázok

Ako sme už spomínali, našim cieľom bude nájsť v korpuse najpodobnejšiu otázku k otázke, ktorú získame od uživateľa.

Aby sme mohli podobnosť otázky vypočítať čo najrýchlejšie, budeme potrebovať vhodnú reprezentáciu dát. V našej aplikácii budeme používať rôzny prístup pri použití frekvenčnej a morfologickej analýzy, ako pri použití analýzy syntaktickej.

Ak sa uživateľ rozhodne využiť frekvenčnú, alebo morfologickú analýzu, budeme si uchovávať otázky z korpusu, rovnako ako uživateľovu otázku, v podobe vektorov *tf x idf* váh termov. Ak sa daný term v otázke nevyskytuje, jeho hodnota vo vektore bude 0. Podobnosť

dvoch otázok bude potom záležitosťou jednoduchého výpočtu, ktorým bude skalárny súčin 2 takýchto vektorov, prezentovaného v článku [3] .

Pri použití syntaktickej analýzy si budeme uchovávať iba otázku uživateľa a aktuálne spracovávanú otázku. Obe otázky budú reprezentované ako závislostné stromy. Pri výpočte budeme postupne prechádzať celý korpus, načítavať vetu po vete, a porovnávať ich s otázkou uživateľovou.

3.2. Frekvenčná analýza

Základným algoritmom aplikácie je algoritmus využívajúci frekvenčnú analýzu. Tento algoritmus pri vyhľadávaní najpodobnejšej otázky nevyužíva morfologickú ani syntaktickú rovinu jazyka. Pracuje výhradne s lemmatmi získanými pomocou tool_chain a ich frekvenciou v otázkach a v texte.

Analýza, rovnako ako každá z nasledujúcich, začína spracovaním uživateľovej otázky pomocou nástroja tool_chain. Takto spracovanú otázku následne rozdelí na jednotlivé tokeny, z ktorých, ak sa jedná o slovo, vyberie lemmata, s ktorými potom pracujeme .

Po zanalyzovaní vstupu je každé slovo z uživateľom zadanej otázky vyhľadávané v otázkach obsiahnutých v korpuse.

3.2.1. Synonymický slovník

Pri zamýšľaní sa, ako vylepšiť výsledky vyhľadávania sme došli k záveru, že úspešnosť aplikácie môže znižovať nejednoznačnosť vo vyjadrení otázky uživateľom a otázky v korpuse. Slová podstatné pre vyhľadávanie môžu byť vyjadrené rôznymi synonymami, ktoré by následne boli pre aplikáciu nerozlíšiteľné.

Do aplikácie sme preto pridali možnosť využiť synonymický slovník s jednoduchým formátom, ľahko editovateľný a doplniteľný. Aplikácia následne obsahuje voľbu, či pri hľadaní chcete použiť synonymický slovník, a tak je len na uživateľovi, či nejaký využije alebo nie. Základný slovník, ktorý vznikol úpravou slovníkov z českého thesaura [9], je priložený k aplikácii.

Štruktúra slovníka je nasledovná - každý riadok v slovníku obsahuje synonymá pre prvé slovo na riadku. Slová su navzájom oddelene zvislou čiarou, teda znakom '|'. Riadok v slovníku vyzerá takto :

$heslo|synonymum_1|synonymum_2|\ldots|synonymum_n$

kde *heslo* je slovo, pre ktoré riadok obsahuje synonymá a $synonymum_1$ až $synonymum_n$ sú synonymá k heslu. Zo štruktúry je viditeľné, že slovník je veľmi ľahko možné editovať (v prípade, že niektoré zo synoným nevyhovuje) alebo doplniť, či už synonymum k už existujúcemu heslu alebo úplne nový synonymický rad.

Aplikácia pri používaní slovníka potrebuje, aby každé slovo, ku ktorému chceme uviesť synonymá, bolo na samostatnom riadku uvedené na začiatku. To znamená, že ak je v slovníku riadok Angličan|Brit, aplikácia vníma slovo Brit ako synonymum slova Angličan, ale slovo Angličan nevníma ako synonymum slova Brit. Aby platil aj tento vzťah, je do slovníka potrebné doplniť riadok Brit|Angličan (Obr. 6 a slová *pak* a *potom*).

```
pak|potom
potom|pak
tak|takto|čili|tedy
abdikace|vzdání|rezignace
abdikovat|odstoupit|vzdát
abnormální|mimořádný|neobvyklý
```

Obr.6 Príklad synonymického slovníka

Uživateľ si pri zadávaní otázky bude môcť vybrať, či chce aby aplikácia slovník používala. V prípade, že slovník bude použitý, pre každý term v otázke od uživateľa bude aplikácia hľadať synonymický rad. Ak nejaký nájde, každé synonymum z tohto radu bude brané tak, že sa v otázke vyskytuje práve toľkokrát, koľko originálne slovo v otázke. Napríklad, majme otázku uživateľa *„Uvažoval jste tehdy o abdikaci?“* . V prípade, že by synonymický slovník obsahoval iba synonymické rady zobrazené na *Obr.6*, aplikácia by považovala aj slová *vzdání* a *rezignace,* akoby sa v otázke objavili práve jedenkrát (čo je počet výskytov pôvodného slova *abdikace* v otázke).

3.2.2. Stopword list

Ďalším vylepšením presnosti programu môže byť tzv. Stopword list, teda zoznam slov, ktoré aplikácia pri výpočte nemá brať do úvahy, aj keď sa vyskytnú v uživateľovej otázke, či v niektorej z otázok v korpuse. Priamo v aplikácií implementujeme hneď 2 verzie – **externý** list, ktorý je možno zadať ako súbor, v ktorom každé slovo je na samostatnom riadku, alebo **adaptabilný** list. Ten si vytvorí samotná aplikácia pri spracovaní korpusu pri štarte programu, na základe štatistiky korpusu. Hranicu početnosti slov, ktoré ma aplikácia považovať za irelevantné (teda slová v stopword liste), si bude môcť určiť uživateľ sám.

3.2.3. Postup výpočtu

Výpočet podobnosti uživateľovej otázky (ďalej v texte označovanej ako q_u) a otázky z korpusu (ďalej ako q_c) bude prebiehať nasledovne.

Mieru podobnosti označíme ako **sim(q_u, q_c)**. Otázku uživateľa, rovnako ako otázky z korpusu budeme reprezentovať ako vektory tf x idf váh. Každý z týchto vektorov má veľkosť n, kde n je počet *rozličných* termov v korpuse. V prípade, že term indexovaný číslom i, sa v otázke nenachádza, hodnota vektora q na pozícii i je q[i] = 0. V opačnom prípade je to tf x idf váha tohoto termu v danej otázke. Podobnosť dvoch otázok **sim(q_u, q_c)** potom zrátame ako skalárny súčin vektorov q_u a q_c.

3.2.4. Zložitosť algoritmu

Definujme n ako počet otázok v korpuse. Ďalej definujme t, ako počet rôznych termov, obsiahnutých v otázkach korpusu.

Potom priestorová zložitosť algoritmu je *O(nt)* pre vektory otázok, pretože každá otázka je uložená ako pole s t prvkami a máme n otázok, a *O(t)* pre ostatné štruktúry potrebné pre výpočet (napríklad mapy hashujúce slovo na index a naopak). Celkovo teda *O(t+nt).*

Pre výpočet potrebujeme n-krát urobiť skalárny súčin vektorov dĺžky t. Následne potrebujeme nájsť index otázky s najvyššou hodnotou podobnosti. Na súčiny potrebujeme čas O(nt), na vyhľadanie indexu potom O(t). Celkovo je teda časová zložitosť algoritmu *O(t+nt)*.

3.3 Morfologická analýza

Ďalším algoritmom, ktorý budeme v našej aplikácii využívať je algoritmus, ktorý berie do úvahy aj morfologické informácie vo vete. Myšlienka je podobná ako v prípade frekvenčnej analýzy, s tým rozdielom, že k slovnému lemma budeme priradzovať aj 15 znakový reťazec reprezentujúci vo formáte CSTS morfologickú informáciu.

3.3.1 Morfologická informácia

Ako sme už spomínali vyššie, morfologická informácia o slove je reprezentovaná 15 znakovým reťazcom. Každá pozícia v tomto reťazci obsahuje informáciu o danej morfologickej kategórii.

Konrétne rozlišujeme tieto kategórie (číslo znamená pozíciu v reťazci) :

1. slovný druh
2. slovný poddruh
3. rod
4. číslo
5. pád
6. privlastňovací rod
7. privlastňovacie číslo
8. osoba
9. čas
10. stupeň
11. negácia
12. aktivum/pasivum
13. nepoužité
14. nepoužité
15. varianta,štýlový príznak a pod.

Niektoré z týchto kategórií, ako napríklad slovný druh, sú určené jednoznačne. V prípade, že ich tool_chain nedokáže rozoznať, uvedie značku X@----------, a teda nemáme žiadnu

morfologickú informáciu o slove. Takéto slovo potom pri výpočte nebudeme brať do úvahy.

Iné však majú priradené značky, ktoré sa navzájom môžu prekrývať. Môže nám teda nastať prípad, kedy je rovnaké slovo s rovnakou informáciou pri tagovaní označené nie rovnakým reťazcom. Napríklad 3.pozícia v reťazci, reprezentujúca rod, má značku pre každý známy rod (ženský, stredný, mužský životný, mužský neživotný), ale aj značky, ktoré vyjadrujú viacero z týchto značiek (mužský životný alebo mužský neživotný – značka Y, nie ženský, teda niektorý z mužských alebo stredný – značka Z, ľubovoľný rod – značka X, atď.) Pri výpočte teda budeme generovať všetky možné reťazce, ktoré by teoreticky vyjadrovali rovnakú informáciu a budeme ich považovať za prítomné v uživateľovej otázke.

```
<s id=otazky3.tmp-001-p113s4>
<f id=otazky3.tmp-001-p113s4W1-Tm>Neblokuje<l>blokovat<t>VB-S---
3P-NA---<r>1<g>0<A>Pred
<f id=otazky3.tmp-001-p113s4W2-Tm>to<l>ten<t>PDNS1----------<r>2
<g>1<A>Sb
<f id=otazky3.tmp-001-p113s4W3-Tm>do<l>do-1<t>RR--2----------<r>
3<g>1<A>AuxP
<f id=otazky3.tmp-001-p113s4W4-Tm>jisté<l>jistý<t>AAFS2----
1A----<r>4<g>5<A>Atr
<f id=otazky3.tmp-001-p113s4W5-Tm>míry<l>míra<t>NNFS2-----A----
<r>5<g>3<A>Adv
<f id=otazky3.tmp-001-p113s4W6-Tm>naši<l>můj<t>PSFS4-P1-------
<r>6<g>8<A>Atr
<f id=otazky3.tmp-001-p113s4W7-Tm>politickou<l>politický<t>
AAFS4----1A----<r>7<g>8<A>Atr
<f id=otazky3.tmp-001-p113s4W8-Tm>scénu<l>scéna<t>NNFS4-----
A----<r>8<g>1<A>Obj
<D>
<d id=otazky3.tmp-001-p113s4W9-Tm>?<l>?<t>Z:-------------<r>9<g>
0<A>AuxK
```

Obr.7 Ukážka vety „Neblokuje to do jisté míry naši politickou scénu?“ vo formáte CSTS. Reťazec s morfologickou informáciou je za značkou <t> pri každom zo slov.

3.3.2. Postup výpočtu

Výpočet podobnosti bude prebiehať podobne ako pri frekvenčnej analýze (viď kapitola 3.2.3).

3.3.3. Zložitosť algoritmu

Zložitosť algoritmu bude, kvôli podobným štrutúram a postupu, takisto rovnaká, ako pri frekvenčnej analýze (viď kapitola 3.2.4.).

3.4 Syntaktická analýza

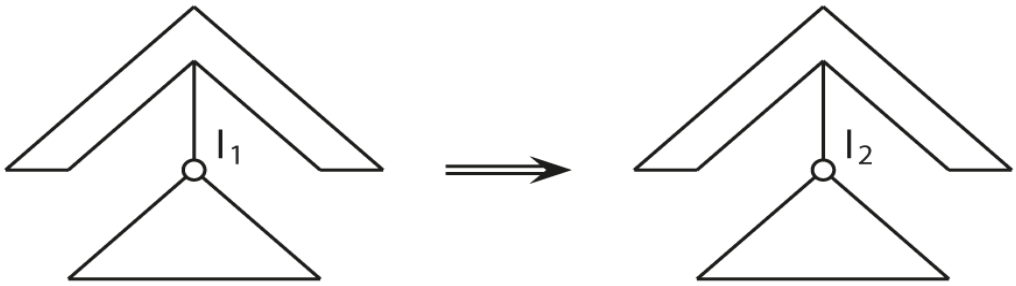
Tretím algoritmom, ktorý aplikácia využíva, je algoritmus syntaktickej analýzy. Spočíva v porovnávaní syntaktického stromu uživateľovej otázky a stromov otázok v korpuse. V tom nám pomôže algoritmus na výpočet tzv. *„tree edit distance“*

3.4.1. Tree edit distance

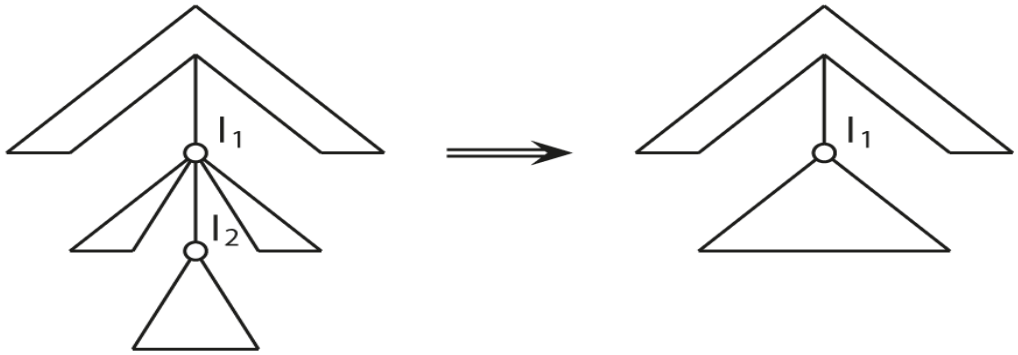
Nakoľko je syntaktická rovina jazyka vo Funkčním Generatívnim Popise vyjadrená pomocou zakoreneného stromu, pre porovnávanie podobnosti otázok budeme potrebovať algoritmus, ktorý nám vyjadrí mieru podobnosti dvoch stromov. Ako ideálny sa v tomto prípade javí práve ***Tree edit distance,*** o ktorom pojednáva článok [1] .

Tento algoritmus porovnáva stromy pomocou hodnoty pretvorenia prvého porovnávaného stromu na druhý. Tá je vyjadrená ako najnižší súčet cien všetkých operácií, ktoré k tomu sú potrebné. Rozlišujeme 3 rôzne operácie na stromoch :

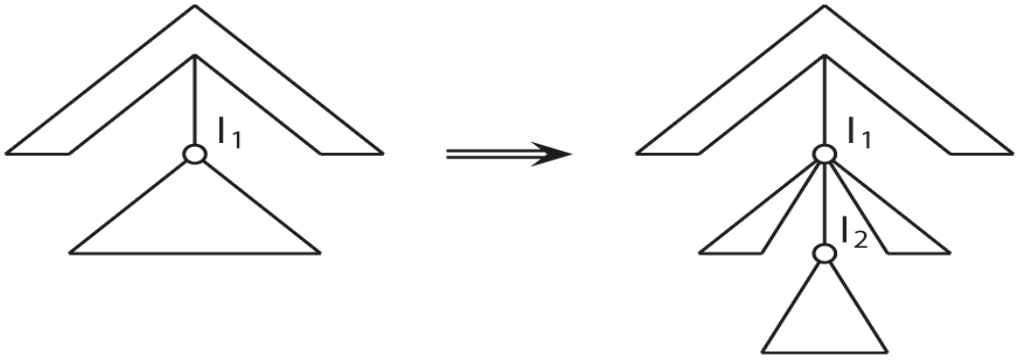
- **premenovanie vrcholu v strome** (Obr.8)
- **zmazanie vrcholu zo stromu** (Obr.9)
- **vloženie vrcholu do stromu** (Obr.10)



Obr.8 Premenovanie vrcholu (l_1 na l_2) v strome



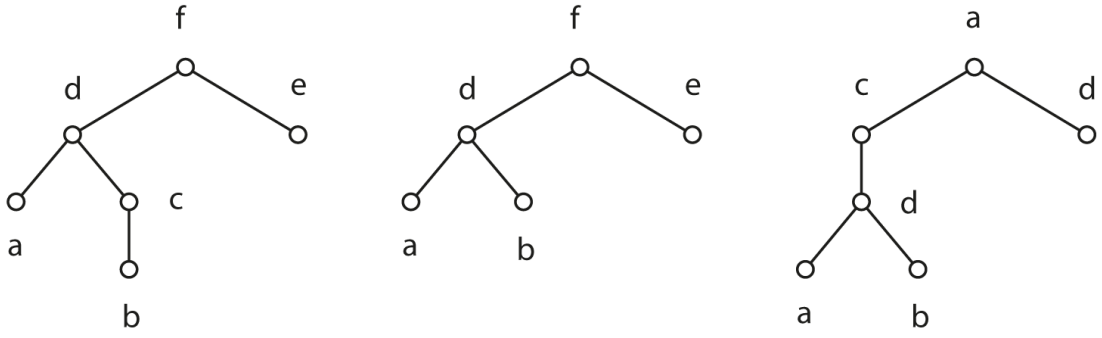
Obr.9 Zmazanie vrcholu (l_2) zo stromu



Obr.10 Vloženie vrcholu (l_2) do stromu

Pre každú z týchto operácií je možné si určiť rôznu cenu. V našej aplikácii používame pre operácie *vloženie vrcholu* (ins) a *zmazanie vrcholu* (del) cenu 1, pre operáciu premenovanie (ren) budeme rozlišovať, či sú vrcholy rôzne – cena premenovania 1, alebo rovnaké – cena premenovania 0.

Úlohou aplikácie, je nájsť pre každú dvojicu *uživateľova otázka (UQ)– otázka z korpusu (CQ)* sekvenciu týchto operácií, ktorá bude viesť k pretvoreniu syntaktického stromu UQ na strom CQ a zároveň bude mať zo všetkých takýchto sekvencií najnižšiu cenu.



*Obr.11 Príklad pretvorenia stromu. Na prvom obrázku zľava strom A, na druhom obrázku strom, ktorý z neho vznikol zmazaním vrcholu **c** (del(c)). Na poslednom obrázku strom, ktorý vznikol z prostredného vložením vrcholu **c** (ins(c)) a premenovaním vrcholu **f** na **a** (ren(f⟶a)) a vrcholu **e** na **d** (ren(e⟶d)).*

3.4.2 Algoritmus Zhang-Shasha

Pre naše potreby budeme využívať na výpočet tree edit distance algoritmus, ktorý uviedli Kaizhong Zhang a Dennis Shasha v roku 1989 [4]. K nemu budeme potrebovať objasniť zopár pojmov.

Označíme tree edit distance medzi zakorenenými stromami T_1 a T_2 ako $\delta(T_1,T_2)$. Nech každý vrchol stromov má pridelený názov z nejakej konečnej abecedy Σ. V našom prípade sú abecedou všetky slová českého jazyka spolu s interpunkčnými znamienkami. Symbol λ

bude reprezentovať špeciálny prázdny symbol. Definujme $\sum_\lambda = \sum \cup \lambda$. Potom každú z operácií môžeme zapísať ako $(l_1 \longrightarrow l_2)$, kde $(l_1, l_2) \in (\sum_\lambda \times \sum_\lambda) \setminus (\lambda, \lambda)$. Operácia je premenovanie, ak $l_1 \neq \lambda$ a zároveň $l_2 \neq \lambda$, zmazanie, ak $l_2 = \lambda$, a vloženie, ak $l_1 = \lambda$.

Každej takejto operácii priradíme hodnotu pomocou metrickej operácie $\gamma(l_1 \longrightarrow l_2) = \gamma(l_1, l_2)$. Cena sekvencie operácií $S = s_1, s_2, \ldots, s_k$, ktorá vedie k pretvoreniu stromu T_1 na T_2, je potom daná ako $\gamma(S) = \sum_{i=1}^{k} \gamma(s_i)$. Edit distance medzi stromami T_1 a T_2 , označený $\delta(T_1, T_2)$ je následne formálne vyjadrený ako :

$\delta(T_1, T_2) = min \{ \gamma(S) \mid S$ je sekvencia operácií takých, že transformujú strom T_1 na $T_2\}$

Ďalej si označíme les (les = množina navzájom neprepojených stromov) ako *F*, a uzol v ňom ako *v*. Potom pod výrazom *F - v* rozumieme les, ktorý vznikne odobraním uzlu *v* z lesa *F*. Výrazom *F – T(v)* budeme rozumieť les, ktorý vznikne zmazaním vrcholu *v* a všetkých jeho potomkov.

Vo výpočte edit distance nám pomôže nasledujúce lemma :

***Lemma 1**: Nech F_1 a F_2 sú usporiadané lesy a γ metrická funkcia definovaná na názvoch uzlov. Nech v a w sú najpravejšie (v zmysle najviac vpravo) korene (ak nejaké existujú) stromov F_1, respektíve F_2. Potom platí :*

$\delta(\vartheta, \vartheta) = 0$

$\delta(F_1, \vartheta) = \delta(F_1 - v, \vartheta) + \gamma(v \longrightarrow \lambda)$

$\delta(\vartheta, F_2) = \delta(\vartheta, F_2 - w) + \gamma(\lambda \longrightarrow w)$

$$\delta(F_1, F_2) = min \begin{cases} \delta(F_1 - v, F_2) + \gamma(v \longrightarrow \lambda) \\ \delta(F_1, F_2 - w) + \gamma(\lambda \longrightarrow w) \\ \delta(F_1(v), F_2(w)) + \delta(F_1 - T_1(v), F_2 - T_2(w)) + \gamma(v \longrightarrow w) \end{cases}$$

Z tohto lemmatu je vidieť, že je možné edit distance vypočítať pomocou dynamického programovania a rátania relevantných podproblémov. Zhang a Shasha tento postup skrátili výpočtom podproblémov iba v kľúčových koreňoch stromov. Tie definujeme ako množinu (T je zakorenený, usporiadaný strom):

$kk(T) = \{koreň(T)\} \cup \{v \in V(T) |\ v\ má\ ľavého\ súrodenca\}$

Špeciálne podlesy stromu T sú všetky lesy *F(v),* kde $v \in V(T)$. Relevantné podproblémy stromu T vzhľadom ku kľúčovému koreňu sú prefixy všetkých špeciálnych podlesov *F(v).*

Lemma 2 : *Pre každý uzol $v \in V(T)$, F(v) je relevantný podproblém.*

Je ľahko nahliadnuteľné, že podproblémy, ktoré sa môžu vyskytnúť vo vyššie uvedenej rekurzii, sú buďto podlesy tvaru F(v), kde $v \in V(T)$, alebo prefixy špeciálneho podlesu stromu T. Z toho vyplýva, že na výpočet edit distance medzi dvoma lesmi F_1 a F_2, teda $\delta(F_1, F_2)$, nám postačí vypočítať $\delta(S_1, S_2)$ pre všetky relevantné podproblémy S_1 a S_2 stromov T_1, respektíve T_2.

Viac o tomto algoritme je možné sa dočítať v už spomínanej práci [4] a na webových stránkach [2] a [6].

3.4.3. Postup výpočtu

V tomto prípade budeme pri výpočte postupovať inak ako v predchádzajúcich dvoch algoritmoch. Súbor nebudeme predspracúvávať, ale prechádzať postupne po zadaní otázky od uživateľa. Tú najprv spracujeme pomocou tool_chain a postavíme jej závislostný strom. Potom budeme prechádzať korpus. Vždy načítame jednu vetu z korpusu, postavíme jej závislostný strom a porovnáme ho so stromom otázky uživateľa. Takto nájdeme 3 najpodobnejšie stromy, a otázky v ktorých sa nachádzajú vrátime ako najvhodnejšie.

Zavedieme teda hodnotu **sim(q_u,q_c),** ktorá vyjadruje podobnosť medzi otázkou uživateľa a otázkou z korpusu. V tomto prípade bude sim(q_u,q_c) = tree-edit-distance (T_u,T_c), teda hodnote tree-edit distance medzi závislostným stromom otázky uživateľa a stromom otázy z korpusu. Po prejdení všetkých otázok v korpuse nájdeme tú, ktorej hodnota sim(q_u,q_c) je **najnižšia**. Táto otázka je podľa tohto postupu najpodobnejšia otázke od uživateľa.

3.4.4. Zložitosť algoritmu

Procedúra bude prechádzať celý súbor s otázkami postupne a porovnávať jednotlivé otázky s uživateľovou. Porovnanie dvoch stromov prebieha v čase $O(|T_1||T_2|min\{D_1, L_1\}min\{D_2, L_2\})$, kde $|T_1|$ a $|T_2|$ označujú počet listov stromov T_1 a T_2 (teda počet slov a interpunkčných znamienok v uživateľovej otázke a otázke z korpusu), D_1 a D_2 označujú hĺbky stromov a L_1 a L_2 počet listov stromu T_1, resp. T_2. Takýchto porovnaní bude potrebné urobiť n, kde n označuje počet viet v súbore s otázkami. Preto časová zložitosť celej procedúry je $\boldsymbol{O(n|T_1||T_2|min\{D_1, L_1\}min\{D_2, L_2\})}$.

Priestorová zložitosť v tejto procedúre je skromnejšia, potrebujeme si uchovávať oba stromy a už vyrátanú podobnosť prejdených stromov. Vyrátané podobnosti budeme uchovávať v poli veľkosti n, kde n je počet otázok v korpuse. V pamäti si budeme držať strom otázky od uživateľa a načitať budeme vždy strom nasledujúcej vety v korpuse. Po ohodnotení podobnosti tento zahodíme a načítame ďalšiu otázku. Priestorová náročnosť algoritmu je teda $O(n+|T_1||T_2|)$.

Kapitola 4

Uživateľská dokumentácia

Aplikácia pracuje v 2 módoch – móde fiktívnych rozhovorov a móde reálnych rozhovorov. V móde fiktívnych rozhovorov má uživateľ možnosť zadať svoju otázku, na ktorú sa program pomocou niektorého z algoritmov pokúsi nájsť vhodnú odpoveď. Mód reálnych rozhovorov predstavuje zoznam všetkých otázok v korpuse, v ktorých je možné i vyhľadávať.

Kvôli používaniu externého nástroja tool_chain, je použitie aplikácie možné iba na UNIXových systémoch s nainštalovaným tool_chain.

4.1. Spustenie aplikácie

Aplikáciu je možné jednoducho spustiť dvojklikom na spustiteľný súbor **Interviewer.ja .** V prípade, že si chce uživateľ vytvoriť zástupcu, môže tak urobiť vytvorením symlinku, príkazom

ln –s cesta_k_Interviewer.jar názov_linku

a zmenením práv vytvoreného súboru na spustiteľný.

Rovnako tak je možné aplikáciu spustiť príkazom *java –jar Interviewer.jar* , samozrejme s plnou cestou k súboru Interviewer.jar.

4.2. Data aplikácie - korpus

Korpus predstavuje vcelku jednoduchý plaintextový súbor, obsahujúci jednotlivé rozhovory spolu so základnými informáciami o nich. Štruktúra korpusu je kvôli následnému spracovaniu pevne daná a je potrebné ju striktne dodržiavať.

Štruktúra korpusu je nasledovná (text za // sú naše komentáre, v korpuse sa nenachádzajú !):

<id n=1> *// poradové číslo rozhovoru v korpuse*

<id n=1>
<Zdroj>http://www.zdrojovastranka.com</Zdroj> *// zdroj odkiaľ sme rozhovor čerpali*
<Kdy>10. Března 2007</Kdy> *// dátum uverejnenia rozhovoru*
<Kde>Miestne noviny</Kde> *// médium, v ktorom bol rozhovor*
<Kdo>Ján Novák</Kdo> *// autor rozhovoru*
<Nazev>Rozhovor o rozhovore</Nazev> *// titulok rozhovoru*

Otázka 1

Odpoveď 1

Otázka 2

Odpoveď 2

.
.
.
Otázka n

Odpoveď n

<id n=2> …

Korpus je potrebné pred použitím v aplikácii spracovať priloženým skriptom **rozdelovac.sh** (jeho popis nájde čitateľ v sekcii 4.5.2.), ktorý ho rozdelí na 2 súbory – odpovede a otázky. Po rozdelení je súbor s otázkami automaticky spracovaný aplikáciou ***tool_chain*** , ktorá prevedie tokenizáciu, morfologickú a syntaktickú analýzu. Takto spracovaný text uloží vo formáte CSTS do súboru s rovnakým názvom ako má súbor s korpusom, ale s príponou *.csts*. Aplikácia potom pracuje iba s týmito novovytvorenými súbormi. Všetky takto vytvorené súbory **musia byť uložené** v podadresári adresáru, kde sa nachdádza spustiteľný súbor Interviewer.jar, s názvom ***data.***
Skript si ale vytvorí aj ďalšie 2 pomocné súbory – jeden súbor s otázkami z korpusu a druhý s odpoveďami. Oba majú rovnaký názov ako pôvodný súbor s rozhovormi, doplnený o reťazec „_odpovede.roc“ resp. „_otázky.roc“. V našom prípade, keď máme korpus rozhovorov s Václavom Havlom, v súbore pomenovanom havel.txt, nám skript vytvoril 3 súbory :
súbor *havel.csts* - obsahuje otázky z korpusu spracované nástrojom tool_chain
súbor *havel_otázky.roc* – obsahuje otázky z korpusu
súbor *havel_odpovede.roc* – obsahuje odpovede z korpusu

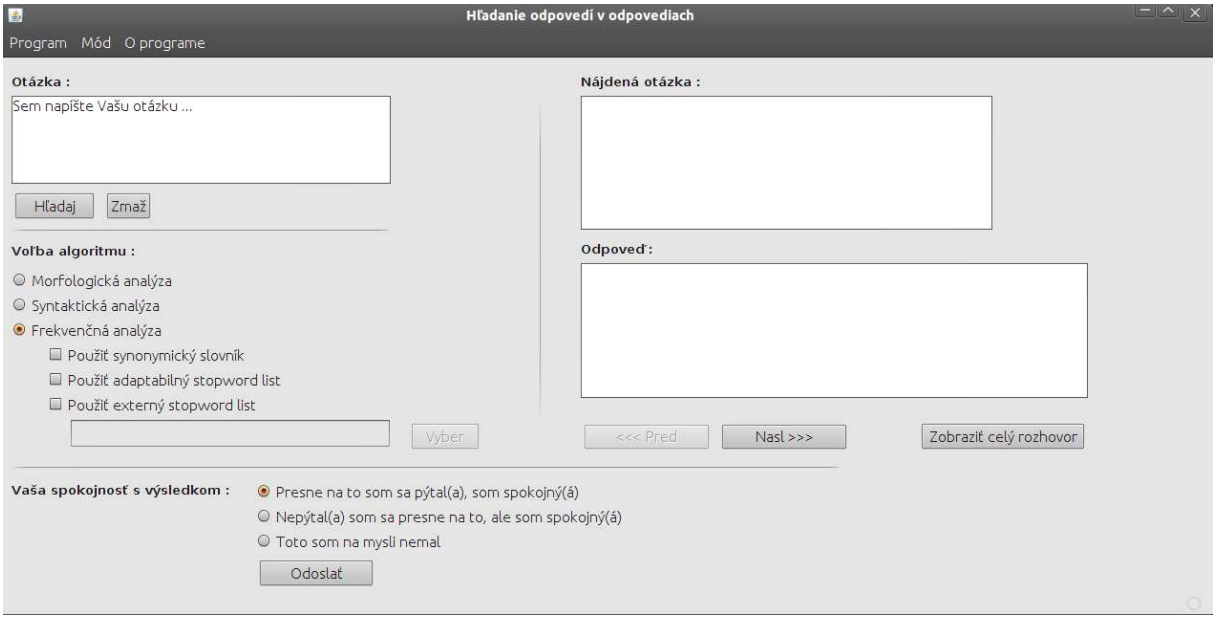
4.2.1 Výber korpusu

Takisto má uživateľ možnosť vybrať si korpus, s ktorým bude pracovať. Aplikácia pri štarte prehľadá svoj podadresár */data*, v ktorom musia byť umiestnené všetky dáta potrebné pre fungovanie aplikácie, na súbory s príponou .csts. V prípade, že nejaký nájde, nastaví si ho ako defaultný súbor, s ktorým bude pracovať. V prípade, že sa v tejto zložke nenachádza žiaden súbor s koncovkou .csts, aplikácia skončí s chybovou hláškou.

Korpus môže uživateľ sám zmeniť cez menu **Program – Zmeň zdroj**, alebo klávesovou skratkou **CTRL+C.** Tento výber si aplikácia zapamätá a pri ďalšom spustení s ním bude pracovať ako defaultným.

4.3. Mód fiktívnych rozhovorov

Prvým z dvoch módov je mód fiktývnych rozhovorov. Uživateľ má možnosť zadať svoju otázku, na ktorú sa aplikácia pokúsi nájsť vhodnú odpoveď.



Obr.12 Úvodné okno aplikácie

4.3.1. Vstupy aplikácie

Aplikácia pre správny výpočet potrebuje niekoľko vstupov od uživateľa. Prvými sú textové súbory, ktoré sa nachádzajú v adresári /data v adresári aplikácie. Korpus je reprezentovaný dohromady 3 súbormi – jedným s príponou **.csts**, ktorý obsahuje otázky z rozhovorov spracované nástrojom tool_chain, a dvoma plaintextovými s príponou **.roc**, z ktorých jeden obsahuje otázky z korpusu a druhý naopak odpovede. O rozdelenie korpusu do týchto súborov sa postará UNIXový skript, s názvom **rozdelovac**, ktorý je priložený k aplikácii.

Po spustení aplikácie sa zobrazí grafické prostredie, ktoré uživateľovi uľahčuje ovládanie aplikácie. Na začatie výpočtu potrebuje od uživateľa, aby do okna určeného pre vstup od uživateľa (Obr.13) zapísal svoju otázku, na ktorú chce v korpuse nájsť odpoveď.

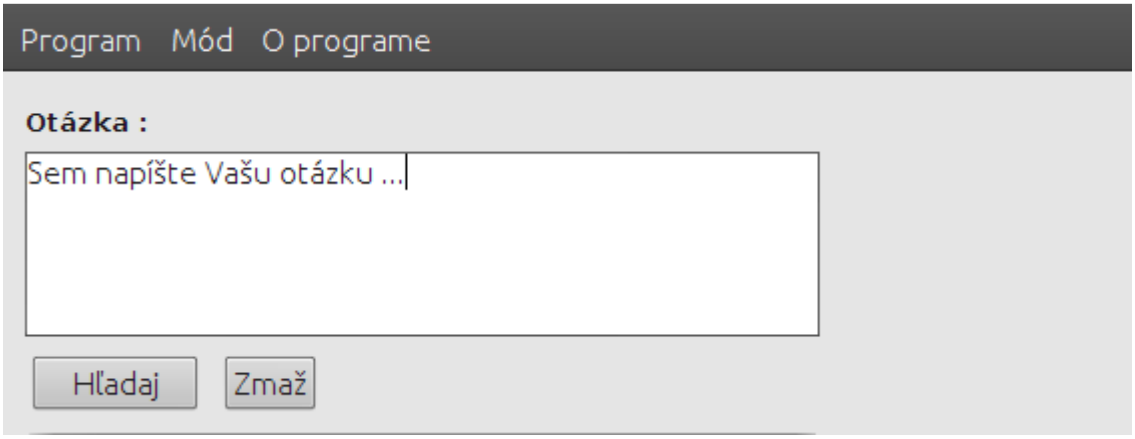
4.3.2. Podmienky na uživateľovu otázku

Cieľom aplikácie je nájsť odpoveď na otázku, ktorú jej zadá uživateľ. Na štýl a formu otázky nekladieme žiadne nároky, no vzhľadom na spôsoby výpočtu v jednotlivých analýzach uvádzame pár odporúčaní.

1. Pri výbere algoritmu frekvenčnej analýzy na spôsobe zadania otázky nezáleží, pretože algoritmus využíva iba slovné základy (lemmata), získané z tejto otázky. V praxi to

znamená, že otázka „Co si myslíte o panu Kalouskovi“ je pre tento spôsob analýzy totožná s otázkou „co se myslet o pan Kalousek“. Inými slovami, je možné otázku zadať aj heslovite, bez ohľadu na morfologickú , či syntaktickú zmysluplnosť.

2. Ak si uživateľ vyberie niektorý zo zvyšných dvoch algoritmov, je pre správnu funkčnosť vhodné, aby zadal otázku vo forme, ako by ju vyslovil, ak by sa na to isté skutočne pýtal danej osobnosti. Je totiž potrebné, aby tool_chain čo najpresnejšie dokázal určiť ako morfologické, tak syntaktické informácie o vete.



Obr.13 Okno, do ktorého uživateľ vpisuje svoju otázku

4.3.3. Výber parsera

Po spustení programu je potrebné určiť cestu k súboru, ktorý bude vykonávať analýzu plaintextu – v našom prípade k tool_chain. K výberu parsera sa uživateľ dostane cez menu **Program – Nastav Parser** , alebo klávesovou skratkou **CTRL+P** .

Po zadaní cesty si túto aplikácia zapamätá a pri ďalšom spustení použije ako defaultnú.

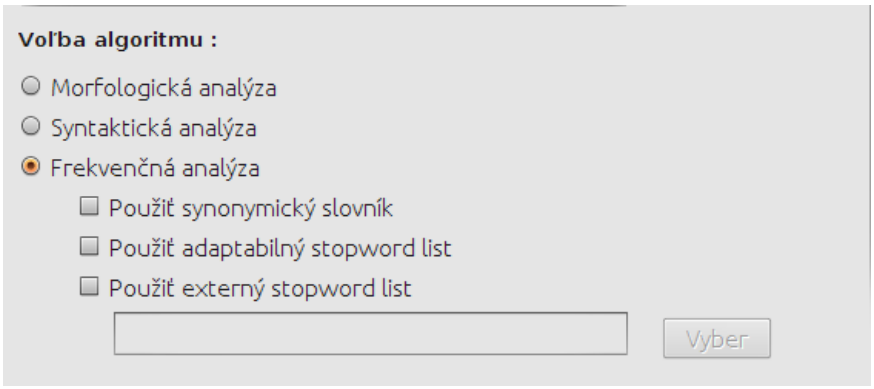
4.3.4. Zobrazenie štatistiky korpusu

Aplikácia si po štarte uchováva informácie o aktuálne spracovávanom korpuse. Konkrétne presný počet rozhovorov, ktoré sa v korpuse nachádzajú, počet otázok, počet slov, ako rôznych, tak aj celkovo, a počet viet.

Všetky tieto štatistiky si uživateľ môž zobraziť voľbou ponuky z menu **Program – Štatistiky**, alebo klávesovou skratkou **CTRL+S.** Informácie sa následne zobrazia v dialógovom okne.

4.3.5. Výber algoritmu

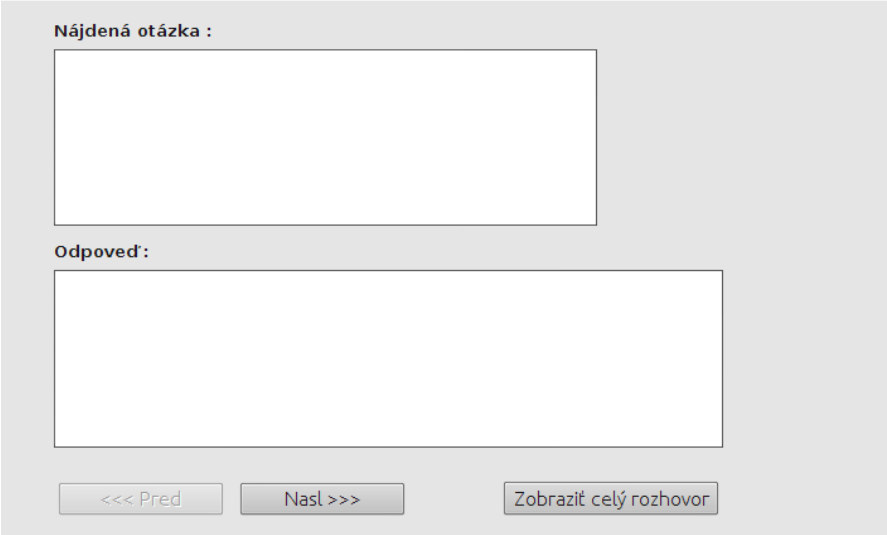
Uživateľ po zadaní otázky potrebuje aj vybrať algoritmus, ktorým chce svoju otázku spracovať. Na výber má z vyššie spomínaných troch algoritmov – frekvenčnej, morfologickej a syntaktickej analýzy. V prípade, že si vyberie frekvenčnú analýzu, sprístupní sa mu možnosť vybrať si z troch doplňujúcich možností ako vylepšiť algoritmus – adaptívneho stopword listu, externého stopword listu a synonymického slovníka. Funkcie všetkých týchto pomocných možností sme uviedli v predchádzajúcej kapitole.



Obr.14 Výber algoritmu procedúry

4.3.6. Výstup aplikácie

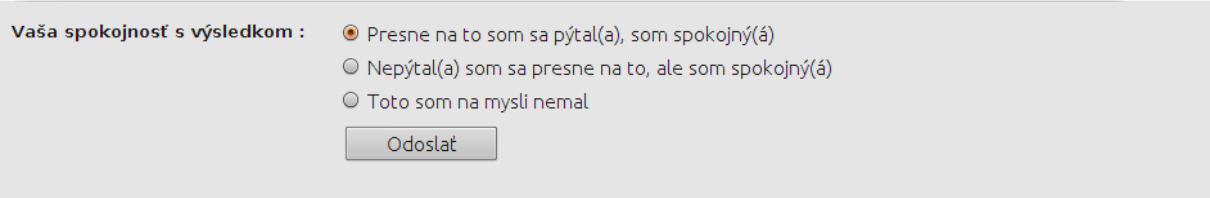
Výsledkom výpočtu každého z algoritmov sú 3 najpodobnejšie otázky z korpusu spolu s k nim priliehajúcimi odpoveďami, ktoré zobrazí v oknách Nájdená otázka, resp. Nájdená odpoveď. Medzi týmito 3 otázkami môže uživateľ prepínať pomocou tlačidiel označených **<<<Pred** a **Nasl>>>**.



Obr.15 Okná, v ktorých program zobrazí nájdenú otázku a odpoveď

4.3.7. Vyjadrenie spokojnosti

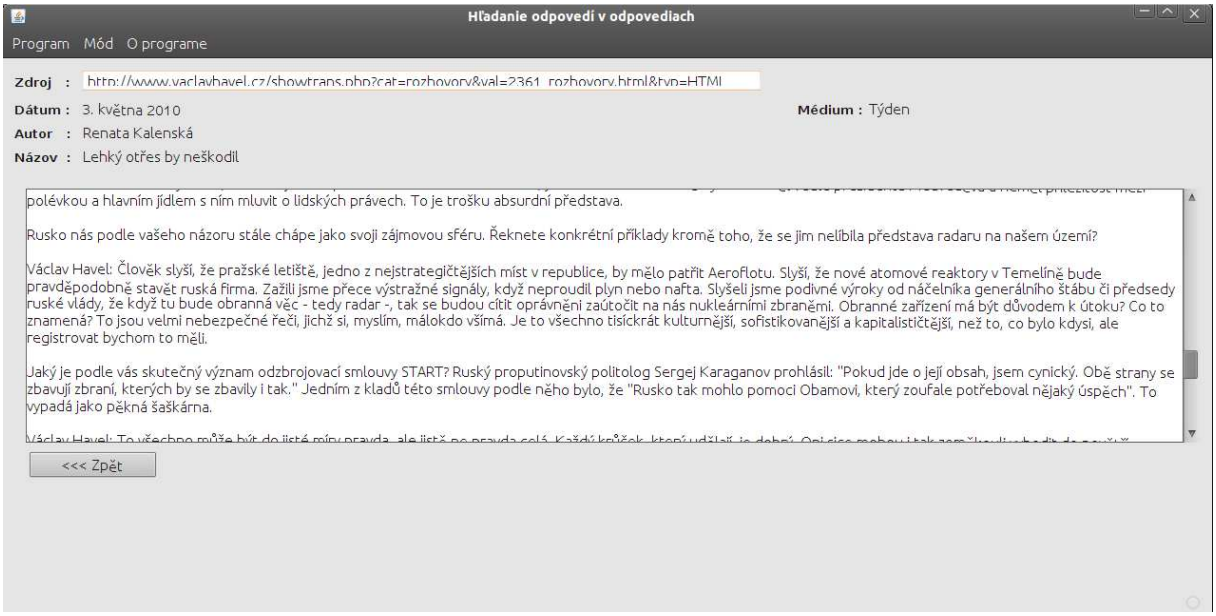
Aby sme mohli zisťovať ako je uživateľ spokojný s nájdenými odpoveďami, má možnosť vyjadriť svoju spokojnosť. Má na výber z troch úrovní spokojnosti. Po výbere jednej z nich potvrdí svoj výber a program následne zapíše do súboru s názvom Log v podadresári /data dátum spolu s uživateľovou otázkou, nájdenou otázkou a odpoveďou a tiež spokojnosťou uživateľa.



Obr.16 Panel na vyjadrenie spokojnosti uživateľa s výsledkom výpočtu

4.3.8. Zobrazenie celého rozhovoru

Po vyhľadaní najvhodnejších odpovedí má uživateľ možnosť zobraziť si celý rozhovor, z ktorého vyhľadané otázky a odpovede pochádzajú. Po stlačení tlačidla označeného **Zobraziť celý rozhovor** sa v novom okne zobrazí rozhovor, v ktorom je obsiahnutá práve zobrazovaná nájdená otázka.



Obr.17 Zobrazenie celého rozhovoru, z ktorého pochádza nájdená otázka a odpoveď

4.3.9. Zmena hranice pre stopword

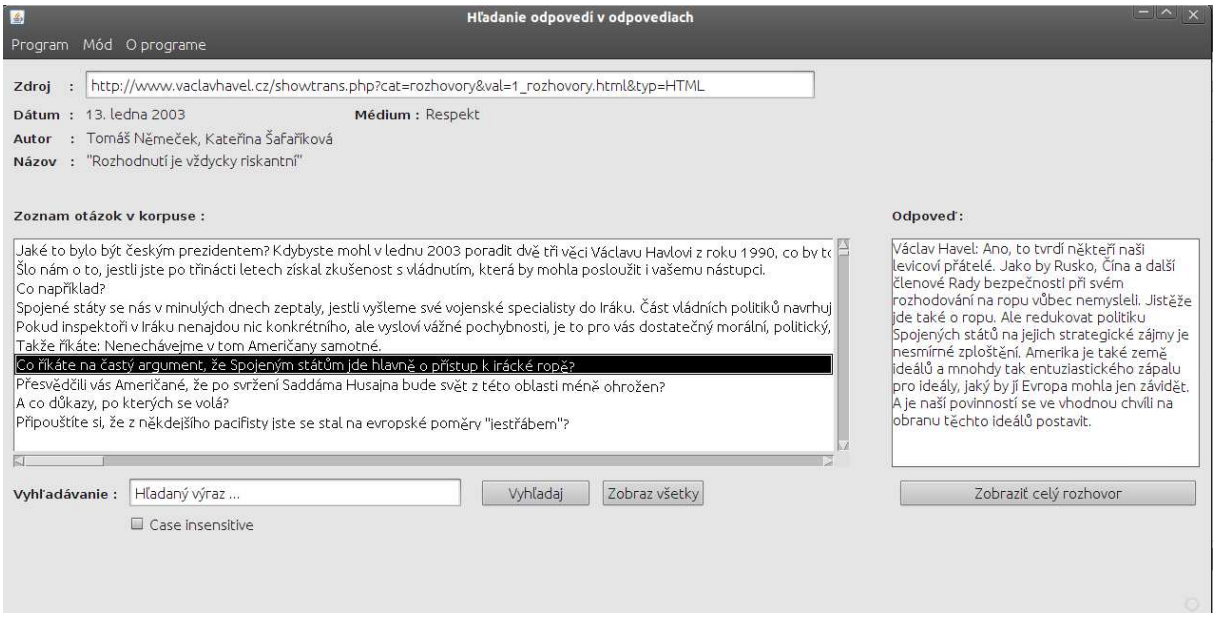
Jednou z možností ako zmeniť parametre výpočtu je taktiež zmena hranice pre stopword. Uživateľ si tak môže stanoviť počet výskytov slova, ktorý bude považovaný ako hraničný pre stopword. Defaultne je v aplikácii nastavený na hodnotu 200. Pre ilustráciu – ak pri výbere algoritmu frekvenčnej analýzy zaškrtne uživateľ možnosť použiť adaptabilný stopword list, slová, ktoré sa v korpuse vyskytujú 200 a viackrát, bude pri výpočte ignorovať.

4.4. Mód reálnych rozhovorov

Druhým módom je mód, v ktorom má uživateľ možnosť prechádzať všetky otázky z aktuálneho korpusu, a zobraziť odpovede na ne. Táto varianta má jednoduché uživateľské prostredie, tvorené informáciami o rozhovore, z ktorého vybraná otázka pochádza, zoznamom otázok a oknom s odpoveďou na vybranú otázku. V otázkach má uživateľ navyše možnosť vyhľadávať pomocou okna pod zoznamom otázok. Rovnako ako v móde fiktívnych rozhovorov, aj tu je možné zobraziť si celý rozhovor, z ktorého vybraná otázka pochádza.

4.4.1. Vyhľadávanie v otázkach z korpusu

Ako sme už spomínali, uživateľ má možnosť v otázkach vyhľadávať. Ovládanie je jednoduché, stačí vpísať hľadaný výraz do okna pod zoznamom otázok a kliknúť na tlačidlo označené **Vyhľadaj.** Aplikácia následne otázky, ktoré obsahujú hľadaný výraz zobrazí v zozname. V prípade, že ani jedna otázka podmienkam vyhľadávania nevyhovuje, program vypíše hlášku, že nebola nájdená vyhovujúca otázka.

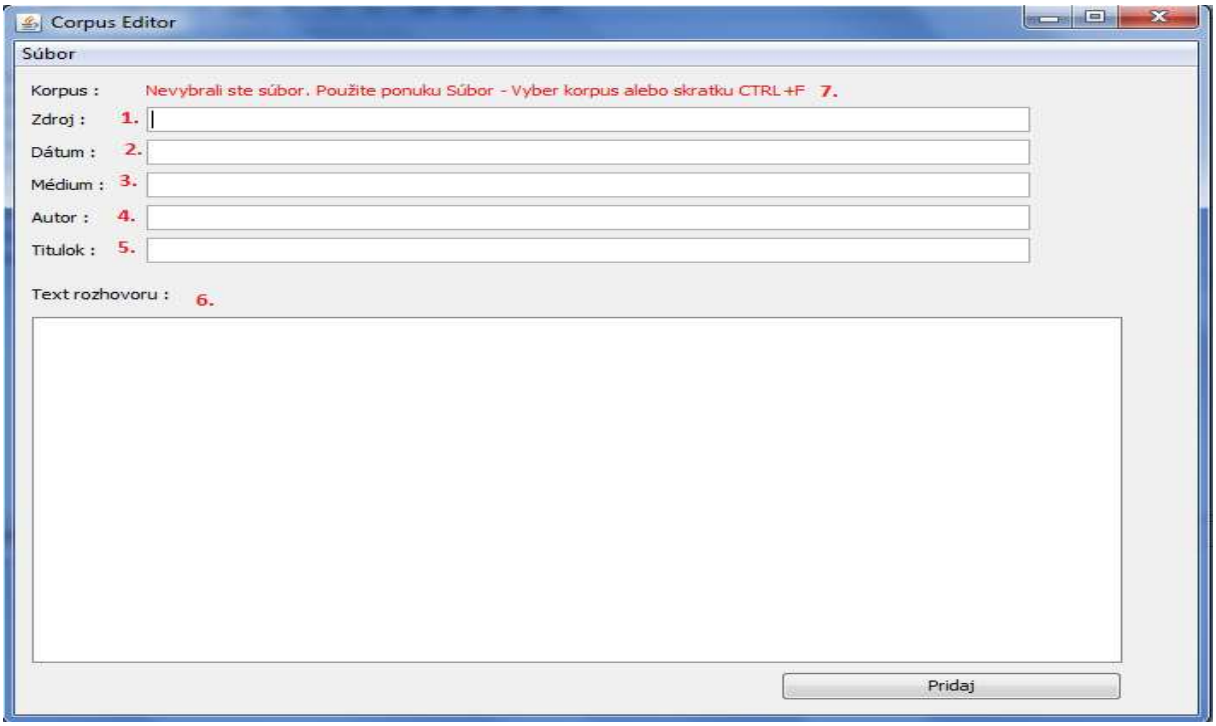


Obr.18 Mód reálnych rozhovorov

4.5. Pomocné skripty a programy

4.5.1 CorpusEditor

Pomocná aplikácia, ktorá jednoducho umožňuje uživateľovi pridať nový rozhovor do už existujúceho korpusu, alebo založiť nový korpus.



Obr.19 Náhľad aplikácie CorpusEditor

Na obrázku je vidieť náhľad aplikácie. Čísla označujú :
1. Pole, do ktorého uživateľ vpíše zdroj, z ktorého pochádza rozhovor
2. Dátum, kedy bol rozhovor uskutočnený
3. Médium, v ktorom bol rozhovor uverejnený
4. Autor rozhovoru
5. Titulok rozhovoru
6. Samotný text rozhovoru. Text musí začínať otázkou, pokračovať odpoveďou na ňu, ďalšou otázkou atď. Otázky a odpovede musia byť oddelené práve jedný prázdnym riadkom.
7. Názov korpusu, s ktorým aktuálne program pracuje

Po vyplnení všetkých políčok uživateľ potvrdí pridanie rozhovoru do korpusu tlačítkom **Pridaj.** Program následne na koniec súboru s korpusom pridá požadovaný rozhovor. V prípade, že chce uživateľ vytvoriť nový korpus, stačí vytvoriť nový prázdny súbor a pomocou tejto aplikácie do neho začať pridávať rozhovory.

Po pridaní nového rozhovoru do korpusu, je **nutné** upravený súbor pred jeho použitím v aplikácii Interviewer spracovať skriptom **rozdelovac.sh!** V opačnom prípade sa zmena neprejaví.

4.5.2 Skript na rozdeľovanie rozhovorov

K aplikácii je pribalený aj skript s názvom **rozdelovac,** ktorý je možno spustiť cez UNIXovský terminál. Jeho úlohou je korpus, ktorý musí byť vo formáte uvedenom v kapitole 4.2. , rozdeliť na súbory s ktorými bude následne pracovať Interviewer. V praxi to znamená, že súbor s rozhovormi a základnými informáciami o nich, rozdelí na súbor s otázkami a súbor s odpoveďami. Následne otázky spracuje pomocou programu tool_chain .

4.5.2.1. Použitie skriptu

Skript sa dá jednoducho spustiť z terminálu následujúcim príkazom :
sh rozdelovac [cesta_k_tool_chain] [subor_na_spracovanie]
Ten potom z pôvodného súboru vytvorí 3 súbory, s ktorými pracuje aplikácia. Pomenuje ich rovnako ako pôvodný, ale s pridaním suffixov. Predpokladajme súbor s názvom *base*. Skript potom vytvorí súbory *base.csts, base_otazky.roc, base_odpovede.roc* , ktoré uloží do adresára, v ktorom je skript spustený.

Kapitola 5

Programátorska dokumentácia

Celá aplikácia je napísaná v jazyku Java a využíva grafické možnosti tohto programovacieho jazyka na uľahčenie jej ovládania uživateľom. Popíšeme najzaujímavejšie a najdôležitejšie časti aplikácie a ich využitie v programe.

Aplikácia je vyvíjaná pre platformu UNIX, kvôli potrebe externých nástrojov určených výhradne pre túto platformu.

5.1. Prehľad súborov

- **CSTSFilter.java** obsahuje FileFilter, ktorý filtruje súbory s príponou .csts

- **Frequency.java** obsahuje metódy a datové štruktúry pre výpočet podobnosti otázky prostredníctvom algoritmu frekvenčnej analýzy. Konštruktor ***Frequency()*** vytvorí a inicializuje triedu. Samotný výpočet spustí metóda ***ZratajPodobnost()***, ktorá vráti indexy 3 najpodobnejších otázok v korpuse.

- **Functions.java** obsahuje pomocné funkcie potrebné pri výpočtoch, napríklad metódu Vyber(String riadok), ktorá z riadka vo formáte CSTS vyberie lemma.

- **Handler.java** riadi správanie sa celej aplikácie. Obsahuje metódy na spracovanie uživateľovej otázky, zápis spokojnosti uživateľa do logu, načítanie súboru obsahujúceho externý stopwordlist a pod.

- **Info.java** reprezentuje objekt s informáciami o riadku v korpuse, potrebnými pre vystavanie závislostného stromu

- **Interview.java** obsahuje informácie o rozhovore z korpusu. Konkrétne začiatok a koniec rozhovoru (poradie otázky v korpuse), autora, dátum uverejnenia, médium, v ktorom bol uverejnený, a titulok rozhovoru.

- **Interviewer.java** vstupný bod aplikácie, obsahuje triedu ***main().***

- **InterviewerAboutBox.java** trieda, zobrazujúca okno s informáciami o aplikácii

- **InterviewerView.java** vykresľuje grafické prostredie aplikácie a stará sa o obsluhu interaktívnych prvkov v aplikácii.

- **Morphology.java** obsahuje metódy na výpočet podobnosti pomocou morfologickej analýzy. Výpočet spustí metóda ***ZratajPodobnostM()***, ktorá takisto ako v triede Frequency vráti indexy 3 najpodobnejších otázok **.** Zároveň obsahuje dátové štruktúry, ktoré sú k tomuto výpočtu potrebné. Taktiež obsahuje HashMaps, ktoré substituujú morfologické informácie za ich nadmnožiny, či podmnožiny.

- **Node.java** trieda reprezentujúca uzol v závislostnom strome. Obsahuje informácie ako lemma slova, ktoré reprezentuje, zoznam synov atď.

- **Stats.java** objekt, obsahujúci štatistiky aktuálne spracovávaného korpusu. Udržiava počet rozhovorov, viet, otázok, rôznych slov a slov celkovo v korpuse.

- **SyntacticTree.java** zaisťuje výpočet podobnosti pomocou algoritmu syntaktickej analýzy. Porovnáva závislostný strom otázky uživateľa so stromami otázok v korpuse. Výpočet spustí metóda ***compareAllQuestions()***, hlavný algoritmus obsahujú metódy ***compareTrees()*** a ***forestDistance().***

- **Tree.java** reprezentuje závislostný strom a niektoré funkcie potrebné pre prácu s ním, napr. ***treeSize()*** , ktorá vráti veľkosť stromu.

5.2. Spracovanie uživateľovej otázky

Otázku, ktorú aplikácia získa od uživateľa, je pre ďalšiu prácu s ňou potrebné analyzovať pomocou tool_chain. Po jej zadaní sa zapíše do dočasného súboru v adresári */temp*. Ten sa následne skonvertuje pomocou programu **iconv** do formátu ISO-8859-2, s ktorým pracuje tool_chain. Následne na takto skonvertovaný textový súbor spustíme **tool_chain** s prepínačom **–tATP,** teda požadujeme tokenizáciu, morfologickú analýzu, tagovanie a syntaktickú analýzu. Výstup tool_chain budeme čítať do poľa Stringov (**String[] otazka**), pomocou triedy BufferedReader. Z toho vyselektujeme podľa používaného algoritmu buď iba slovné jednotky (tokeny), teda riadky začínajúce tagom **<f**, prípadne aj interpunkciu (v prípade syntaktickej analýzy), teda riadky začínajúce tagom **<d**.

5.3. Algoritmy na výpočet podobnosti

5.3.1. Frekvenčná analýza

Pri frekvenčnej analýze sa predspracuje korpus, s ktorým sa pracuje už pri štarte aplikácie, resp. pri znovuzvolení algoritmu. Aplikácia prejde celý súbor .csts a vyráta pre každé slovo jeho tf-idf váhu v korpuse pre každú z otázok. Vytvorí sa dvojrozmerné pole floatových hodnôt s názom **term_weights**, a s rozmermi [počet otázok v korpuse]x[počet rôznych slov (termov)]. Každá otázka je teda reprezentovaná ako vektor hodnôt tf-idf.

Algoritmus je implementovaný v súbore *Frequency.java*. Nakoľko nie je možné indexovať pole pomocou Stringov, každý term má pomocou HashMap s názvom **words** pridelený index, pomocou ktorého sa potom indexuje do poľa.

Ako prvé, prejde aplikácia celý súbor a zráta počet výskytov toho-ktorého slova v každom dokumente (otázke). Počet slov v dokumentoch si ukladá do poľa **term_freq**. Taktiež potrebujeme aj počet dokumentov, v ktorých sa term vyskytuje, tie sú uložené v poli **doc_freq**. Položka doc_freq[i] obsahuje počet dokumentov, v ktorých sa nachádza slovo, indexované číslom i.

Po predspracovaní súboru program čaká na vstup od uživateľa. Po zadaní otázky a jej spracovaní, zráta rovnako ako v prípade celého korpusu váhy termov v otázke s malým rozdielom vo vzorci výpočtu tf-idf. V závislosti na tom, či uživateľ označí možnosť použitia adaptabilného stopword listu, si aplikácia nájde slová, ktoré sa v korpuse vyskytujú viackrát ako je stanovená hranica pre stopword (hodnota **stopwordBoundary** v handleri), a tie pridá do zoznamu ArrayList **stopwords.** Termy obsiahnuté v tomto zozname následne pri výpočte preskakuje.

Výpočet podobnosti uživateľovej otázky s otázkami v korpuse prebieha prechádzaním vektorov všetkých otázok v korpuse a výpočtom podľa vzorca. To obstaráva funkcia **Count().** K tomu využíva skalárny súčin dvoch vektorov, ktorý obstaráva funkcia **InnerProduct()** v súbore *Functions.java.* Nakoniec vyberie pomocou funkcie **FindMaxIndex()**, tiež zo súboru *Functions.java,* indexy 3 otázok s najvyššou hodnotou v poli **similarity.** Otázky a odpovede odpovedajúce týmto indexom získa aplikácia zo súborov s príponou .roc, ktoré vzniknú pri spracovaní korpusu skriptom *rozdelovac.sh*, a to pomocou metód **GetQuestion()** a **GetAnswer()** implementovaných v súbore *Handler.java*.

5.3.2. Morfologická analýza

V tomto algoritme sa postupuje veľmi podobne frekvenčnej analýze. Jediným rozdielom je, že pri načítaní súboru z riadkov nevyberá iba lemma, ale rovnako tak morfologickú informáciu, teda 15 znakový reťazec, tzv. **tag** (viď kapitola 3.3.1.). Ten potom spojí s lemmatom a s medzerou do jedného reťazca, ktorý považuje za token. Celá procedúra

je implementovaná v súbore *Morphology.java*.

Vzhľadom k tomu, že niektoré morfologické značky nie sú jednoznačné, potrebujeme vygenerovať všetky možné reťazce, ktoré by vyhovovali tomu získanému z uživateľovej otázky. To obstaráva funkcia **generujInfo()**, ktorá k danej morfologickej informácii vráti všetky jej nadmnožiny a podmnožiny, ktoré môžu byť obsiahnuté v korpuse. To prebieha postupným substituovaním „ekvivalentných" morfologických značiek.

Nakoniec, podobne ako pri frekvenčnej analýze, funkcia **CountM()** vyráta podobnosť výpočtom, kedy porovnáva vektory otázky uživateľa a otázok v korpuse, následným vybratím 3 najpodobnejších otázok a vrátením ich indexov.

5.3.3. Syntaktická analýza

Výpočet podobnosti pomocou syntaktickej analýzy prebieha inak ako v predchádzajúcich 2 prípadoch. O celú procedúru sa stará trieda **SyntacticTree** implementovaná v súbore *SyntacticTree.java.*

Najprv potrebujeme načítať otázku od uživateľa a z nej postaviť závislostný strom. Ten zrekonštruujeme pomocou informácií obsiahnutých v súbore .csts. Každý riadok obsahuje číslo uzlu a taktiež číslo jeho predchodcu. Uzol v strome reprezentuje trieda **Node**, celý strom potom trieda **Tree**. O postavenie stromu sa stará metóda **buildTree()**.

Podobnosť 2 stromov ráta metóda **compareTrees()**, pomocou algoritmu na výpočet tree edit ditance. Ten je implementovaný čiastočne v tejto metóde a metóde **forestDistance().**

K tomu je potrebné nájsť tzv. „keyroots" pre každý strom a taktiež najľavejší list podstromu každého uzlu. O nájdenie keyroots sa stará metóda **findKeyRoots()**, o left-most leaf descendants (LMLD) metóda **findLMLD().**

Celý výpočet spustí metóda **compareAllQuestions()**, ktorá vráti pole s indexmi 3 najpodobnejších otázok.

5.4. Najdôležitejšie triedy, metódy a štruktúry

- **class Frequency**

  reprezentuje objekt, ktorý sa stará o výpočet podobnosti otázky od uživateľa s otázkami v korpuse a nájdenie 3 najpodobnejších otázok.

  Štruktúry a premenné :

  - **private int[][] term_freq** – dvojrozmerné pole, uchováva term frequency (tf) pre všetky termy vo všetkých otázkach. Prvý rozmer predstavuje všetky termy v korpuse, druhý rozmer otázky. Napr. term_freq[t][q] obsahuje tf termu t v otázke q.

- **private int[] doc_freq** – obsahuje document frequency (df) pre všetky termy. Napr. doc_freq[i] obsahuje počet dokumentov (otázok) korpusu, v ktorých sa vyskytuje term indexovaný číslom i.
- **private float[][] term_weights** – dvojrozmerné pole, obsahuje tf x idf pre všetky termy vo všetkých dokumentoch. Rovnako ako v term_freq, prvok term_weights[t][q] obsahuje váhu termu t v otázke q.
- **private float[] question_weight** – vektor uživateľovej otázky. Jeho veľkosť je rovná počtu rôznych termov v korpuse a obsahuje tf x idf všetkých týchto termov v otázke od uživateľa.
- **private float[] idfs** – obsahuje inverse document frequency (idf) pre všetky termy v korpuse. Napr. idfs[t] obsahuje idf termu indexovaného číslom t.
- **private HashMap words** – slúži na indexovanie termov pomocou čísel. Každý term z korpusu je namapovaný na nejaké celé číslo. V opačnom poradí, teda z čísla na term mapuje HashMap **words_reverse**
- **private ArrayList stopwords** – zoznam slov, ktoré sú označené ako stopwords, v prípade, že uživateľ zvolil algoritmus s využitím adaptabilného stopword listu
- **private float[] similarity** – pole vypočítaných podobností otázky uživateľa s otázkami v korpuse. Veľkosť poľa je rovná počtu otázok v korpuse. Každá položka označuje podobnosť otázky uživateľa s danou otázkou v korpuse. Napr. similarity[i] je podobnosť uživateľovej otázky s i-tou otázkou v poradí v korpuse.

Metódy :

- **private ArrayList vyhladajTermy(File f)** – vyhľadá a vráti zoznam všetkých rôznych termov obsiahnutých v súbore f vo formáte CSTS
- **protected void findStopWords(int hranica)** – vyhľadá všetky termy s početnosťou väčšou ako určená hranica a pridá ich do zoznamu *stopwords*
- **public int[] zratajPodobnost(String uq, boolean adapt, boolean loaded, boolean synonyms)** – spustí výpočet podobnosti otázky *uq* s otázkami v korpuse. Boolean hodnoty indikujú použitie adaptabilného a externého stopword listu, prípadne synonymického slovníka.
- **private void count()** – samotný výpočet podobností, naplní pole *similarity*
- **private void init()** – inicializuje dátové štruktúry a premenné
- **private void zratajTF(File f, int pocetDokumentov, int[][] term_frequencies)** – zráta tf pre všetky termy v súbore *f* , ktorý je vo formáte CSTS. Premenná *pocetDokumentov* slúži na inicializáciu poľa a predstavuje počet otázok v korpuse. Výsledky bude priradzovať do poľa *term_frequencies*.

- **private void zratajTFQ(String[] otazka,boolean adapt,boolean loaded,boolean synonyms)** - zráta tf pre všetky termy v uživateľovej otázke *otazka*. Boolean hodnoty preberá od metódy *zratajPodobnost(...).*

- **class Morphology**

- objekt zodpovedný za výpočet podobností pomocou morfologickej analýzy, teda s využitím morfologickej informácie. tagu
- väčšina štruktúr a metód je podobná tým v triede Frequency, metódy majú na konci pridané ešte písmeno M (napr. countM() alebo initM()), obsahuje však aj niektoré štruktury a metódy navyše

Štruktúry a premenné :

- **private HashMap<Character,ArrayList<Character>> rod, cislo, pad, privlRod, privlCislo, osoba, cas** – zoznamy možných substitúcií v jednotlivých morfologických kategóriách. Napr. pre rod písmeno 'M' mapuje zoznam <X,Y,Z>.

Metódy :

- **private void initMaps()** – naplní vyššie spomínané HashMaps
- **private ArrayList<String> nahradSpoj(String riadok)** – ak je na riadku slovný token, vyberie z neho lemma a morfologickú informáciu, následne vygeneruje všetky možné „ekvivalentné" morfologické informácie a vráti ich v zozname
- **private ArrayList<String> generujInfo(String povInfo)** – z pôvodnej morfologickej informácie *povInfo* vygeneruje všetky do úvahy pripadajúce informácie

- **class SyntacticTree**

trieda, ktorá zaobstaráva porovnávanie otázok z hľadiska syntaktickej podobnosti. Otázky reprezentuje ako závislostné stromy a porovnáva ich na základe tree edit distance.

Štruktúry a premenné :

- **final int del,ins, ren** – hodnota operácií delete, insert a rename pri výpočte tree edit distance. V našom prípade defaultne všetky nastavené na hodnotu 1.

- **Tree userQuestion** – strom, do ktorého si trieda uloží otázku od uživateľa v podobe závislostného stromu
- **Tree examinedQuestion** – premenná, v ktorej si procedúra bude postupne uchovávať aktuálne spracovávanú otázku z korpusu ako závislostný strom.
- **Integer[] questions** – pole, v ktorom sú uložené tree edit distances (ted) pre každú z otázok v korpuse
- **ArrayList<Node> kr1** – vzostupne zotriedený zoznam uzlov stromu, ktoré figurujú ako tzv. „keyroots“**.** Zotriedený podľa poradového čísla uzlu po prechode stromom v postorder poradí
- **Node[] l1** – zoznam left-most leaf descendants pre každý uzol stromu. Napr. l1[i] obsahuje najľavejší list v podstrome, ktorého koreňom je uzol s poradovým číslom i.

Metódy :

- **private Tree loadUQ (String[] uq)** – postará sa o načítanie závislostného stromu uživateľovej otázky *uq*, ktorá už je spracovaná tool_chain
- **private Node buildTree (ArrayList<Info> words)** – postaví strom zo zoznamu slov vety, ktoré sú reprezentované ako zoznam objektov Info
- **private ArrayList<Node> findKeyRoots(Node tree)** – vráti zoznam keyroots stromu s koreňom *tree*
- **private Node[] findLMLD (Node tree)** - vráti pole s LMLD pre každý z uzlov v strome
- **public int compareTrees(Tree tree1, Tree tree2)** – vypočíta tree edit distance medzi stromami *tree1* a *tree2*.
- **protected int[] findMinIndex (Integer[] pole)** – nájde indexy 3 otázok s najmenším tree edit distance
- **public int[] compareAllQuestions(String question)** – spustí celý výpočet podobnosti otázky *question* s otázkami v korpuse, vráti index 3 najpodobnejších otázok

o class Handler

udržuje súbory, s ktorými aplikácia pracuje, je medzičlánkom medzi grafickým prostredím a triedami pre výpočty

Štruktúry a premenné :

- **protected File otazky** – textový súbor, v ktorom je uložený korpus vo formáte CSTS, s ktorým aplikácia aktuálne pracuje
- **protected File otazky_plain, odpovede_plain** – súbory s príponou .roc, obsahujú otázky a odpovede z korpusu ako plaintext
- **private File temp** – dočasný súbor, ktorý aplikácia používa na zápis otázky od uživateľa a jej následné spracovanie tool_chain
- **public File tool_chain** – udržuje umiestnenie tool_chain
- **protected File dictionary** – textový súbor, ktorý obsahuje synonymický slovník
- **public int stopwordBoundary** – hranica, minimálny počet výskytov termu, kedy sa začne považovať za stopword
- **private final File log** – textový súbor, do ktorého aplikácia zapisuje výsledky vyhľadávaní a spokojnosť uživateľa
- **public ArrayList<String> synonyma** – zoznam synoným, ktoré sú obsahnuté v synonymickom slovníku
- **public HashMap interviews** – mapuje otázku na rozhovor, z ktorého pochádza
- **protected Interview[] rozhovory** – pole rozhovorov v korpuse
- **Frequency frequency** – inštancia triedy Frequency, s ktorou aplikácia pracuje
- **Morphology morphology** – to isté pre triedu Morphology
- **SyntacticTree synTree** – to isté pre triedu SyntacticTree
- **Stats statistics** – inštancia triedy Stats, obsahujúca štatistiky o korpuse, s ktorým aplikácia aktuálne pracuje

Metódy :

- **private void init ()** – inicializuje premenné, nájde súbor s príponou .csts a priradí ho ako defaultný korpus
- **protected String getAnswer(int n)** – vráti n-tú odpoveď z korpusu
- **protected String getQuestion(int n)** – vráti n-tú otázku z korpusu
- **protected Interview getInterview(int i)** – vráti rozhovor, z ktorého pochádza i-tá otázka v korpuse
- **private String vyber(String riadok)** – z riadka vo formáte CSTS vyberie lemma
- **public String[] spracujOtazku(String otazka)** – uživateľovu otázku *otazka* zapíše do dočasného súboru, ktorý skonvertuje do potrebného formátu, spracuje tool_chain a výstup vráti ako pole Stringov, kde každá položka predstavuje jeden riadok výstupu
- **protected void loadStopWords(File file)** – načíta externý stopword list
- **private void hashInterviews ()** – namapuje rozhovory do poľa *rozhovory*

- **protected String showWhole(int i)** – vráti celý rozhovor, v ktorom sa nachádza otázka s poradovým číslom i v korpuse
- **public void zapisLog(String user_question,String found_question,String answer)** – do logfile zapíše výsledok výpočtu
- **public void zapisLogSpokojnost (String satisfaction)** - do logfile zapíše spokojnosť uživateľa s výsledkom

- **class Interview**

objekt, ktorý predstavuje jeden rozhovor z korpusu. Uchováva údaje o rozhovore, potrebné pre jeho zobrazenie.

Štruktúry a premenné :

- **String zdroj** – zdroj, odkiaľ rozhovor pochádza, napr. webová adresa
- **String kdy** – dátum, kedy bol rozhovor uverejnený
- **String kde** – médium, kde bol rozhovor uverejnený
- **String kdo** – autor rozhovoru
- **String nazev** – titulok rozhovoru
- **int zacatek** – poradové číslo počiatočnej otázky daného rozhovoru v korpuse
- **int konec** - poradové číslo poslednej otázky daného rozhovoru v korpuse

Metódy :

- **public static String[] getInfo(Interview i)** – vráti informácie o i-tom rozhovore v korpuse

Kapitola 6

Výsledky evaluácie

Aplikáciu sme po jej dokončení poslali 5 ľuďom aby ju otestovali a vyjadrili svoju spokojnosť. Zadanie znelo aby položili 10 otázok tak, akoby sa rozprávali priamo s pánom Havlom, teda pýtali sa priamo neho a následne zhodnotili svoju spokojnosť s výsledkom. Rovnakých 10 otázok mali vyhľadať pomocou všetkých 3 algoritmov.

Dokopy tak položili 50 otázok pre každý z algoritmov s takýmito výsledkami :

algoritmus / spokojnosť	Frekvenčná analýza	Morfologická analýza	Syntaktická analýza
spokojný	14 / 28%	10 / 20%	4 / 8%
nie celkom spokojný	24 / 48%	17 / 34%	9 / 18%
nespokojný	12 / 24%	23 / 46%	37 / 74%

Obr.20 Tabuľka s výsledkami evaluácie

Z výsledkov možno vyčítať, že najefektívnejším riešením sa javí frekvenčná analýza pomocou tf x idf váh. Ako najmenej úspešná vyzerá byť syntaktická analýza, ktorá využíva porovnávanie závislostných stromov viet.

Štatistiky korpusu

V našej práci sme pracovali s korpusom, ktorý sme zozbierali z osobnej stránky pána Václava Havla a rozhovorov s ním. V čase písania tejto práce mal následovné parametre :

Počet rozhovorov v korpuse : 97
Počet otázok : 1857
Počet viet : 3567
Počet rôznych slov : 6174
Počet slov celkovo : 43 625

Štatistiky zohľadňujú iba súbor s otázkami, spracovaný tool_chain. Aplikácia pri výpočte pracuje v podstate iba s týmto súborom, ostatné používa iba pri nájdení konkrétnej otázky alebo odpovede, preto sú ich vlastnosti nepodstatné.

Kapitola 7

Záver

Na záver sa pokúsime zhrnúť výsledky tejto práce, vyjadriť s čím sme spokojní, a čo sme naopak očakávali lepšie. Zhrnieme v čom vidíme nedostatky a možné zlepšenia, a taktiež ako si interpretujeme výsledky evaluácie.

7.1. Interpretácia výsledkov

Na prvý pohľad možno úspešnosť aplikácie nie je nejak ohromujúca, no musíme brať do úvahy, že podstatným činiteľom je v tomto prípade aj množstvo zozbieraných rozhovorov a ich tématický rozsah. Otázky uživateľov pri evaluácii sme tématicky nijak neobmedzovali, preto mohli smerovať aj do oblastí, ktoré pri reálne uskutočnených rozhovoroch neboli zasiahnuté. Môžme teda predpokladať, že pri dostatočnom počte rozhovorov a väčšej šírke záberu by výsledky boli podstatne lepšie.

Z výsledkov je zjavné, že ako najefektívnejší sa javí postup frekvenčnej analýzy, s použitím *tf x idf* váh. Jeho výhodou je, že berie do úvahy najmä samotné slová, a teda dáva najväčší dôraz na význam. Morfologická analýza pracuje podobne, je však trochu podrobnejšia a presnejšia, čo dáva väčšie nároky na rovnaké formulovanie otázky uživateľom tak, ako bola formulovaná autorom rozhovoru.

Syntaktická analýza zasa porovnávala závislostné stromy viet. Využívala na to algoritmus na výpočet tree edit distance, ktorý rátal počet operácií potrebných na pretvorenie jedného stromu na druhý. Zohľadňovala teda aj premenovávanie uzlov, ale celková podobnosť mohla byť klamlivá. Pre otázku, ktorá obsahuje málo slov, vyhodnotila ako najpodobnejšiu otázku, ktorá taktiež obsahovala málo slov a mala podobný závislostný strom, aj keď napríklad otázka uživateľa bola podmnožinou dlhšej otázky v korpuse. Logicky na úpravu stromu s 5 uzlami na strom s 20 uzlami bolo potrebné viac operácií, ako na strom so 7 uzlami, aj keď možno inak pomenovanými.

7.2. Návrhy na vylepšenie

Pôvodne sme v aplikácii chceli zahrnúť aj automatické sťahovanie rozhovorov z webstránky. Narazili sme ale na problém ako identifikovať nový článok ako rozhovor.

Každopádne by takáto možnosť bola vhodným doplnením a určite aj vylepšením aplikácie.

Najradikálnejšie vylepšenie by mohlo predstavovať doplnenie rozhovorov a snažiť sa pokryť čo najširší rozsah tém. To by zamedzilo nájdeniu nesprávnych, či nerelevantných odpovedí na uživateľskú otázku, ktorá sa týka témy, ktorá v korpuse obsiahnutá nie je .

Syntaktická analýza by mohla byť vylepšená implementovaním algoritmu, ktorý by bral do úvahy aj možnosť, že otázka uživateľa je podstromom otázky v korpuse.

Ďalšou možnosťou ako možno vylepšiť úspešnosť aplikácie môže byť skombinovanie použitých algoritmov, napríklad ich zreťazením a skombinovaním ich výsledkov. Napríklad nájdenie obsahovo najpodobnejších otázok pomocou frekvenčnej analýzy a následne z týchto vybrať najvhodnejšie pomocou morfologickej a syntaktickej analýzy by výkon aplikácie mohlo zvýšiť.

7.3. Sumár

V rámci tejto práce sme navrhli a implementovali 3 rôzne procedúry, ktorými sa pokúšame nájsť odpoveď na otázku uživateľa. Rozhodli sme sa pre prístup hľadania najpodobnejšej otázky v korpuse rozhovorov a ako odpoveď na otázku uživateľa priradiť odpoveď na najpodobnejšiu otázku v korpuse.

Zozbierali sme korpus rozhovorov s pánom Václavom Havlom z jeho osobnej stránky [11]. Tie sme spracovali a upravili do nami navrhnutej štruktúry a následne spracovali pomocou nástroja tool_chain.

V rámci procedúr sme implementovali výpočet tf x idf váh v dokumentoch, algoritmus na výpočet tree-edit distance, potrebný pre porovnávanie stromov, metódy na rekonštrukciu závislostných stromov zo súboru vo formáte CSTS, rôzne pomocné metódy potrebné k výpočtom.

Aplikáciu sme po jej dokončení poskytli na evaluáciu niekoľkými uživateľmi, výsledky ich skúšania sme spracovali do tabuľky, ktorá ja v práci uvedená v kapitole 6. Pokúsili sme sa zhrnúť vlastnosti aplikácie a navrhnúť jej možné vylepšenia do budúcnosti.

Literatúra

[1] Philip Bille : A Survey on Tree Edit Distance and Related Problems 2005, Journal Theoretical Computer Science, Volume 337 Issue 1-3.

[2] Resources for computing Tree Edit Distance http://web.science.mq.edu.au/~swan/howtos/treedistance/

[3] Gerard Salton and Christopher Buckley : Term-weighting approaches in automatic text retrieval. Information Processing and Management, vol. 24, issue 5. Pergamon Press, Inc. Tarrytown, NY, USA. 1988

[4] Kaizhong Zhang and Dennis Shasha. 1989 Simple fast algorithms for the editing distance between trees and related problems SIAM J. Comput., Vol. 18, No. 6. (December 1989), pp. 1245–1262

[5] Term frequency and weighting http://nlp.stanford.edu/IR-book/html/htmledition/term-frequency-and-weighting-1.html

[6] Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann : An Optimal Decomposition Algorithm for Tree Edit Distance http://www.cs.brown.edu/~shay/ted.pdf

[7] Český akademický korpus 2.0 http://ufal.mff.cuni.cz/rest/CAC/doc-cac20/cac-guide/cz/html/ch2.html

[8] D.C. Bobrow, R.M. Kaplan, M. Kay, D.A. Norman, H. Thompson, and T. Winograd. GUS, a frame-driven dialogue system 1977. Artificial Intelligence, Volume 8, pp. 155–173.

[9] OpenOffice.org, Czech Thesaurus (for OpenOffice.org 2.x) 2007-09-26 http://wiki.services.openoffice.org/wiki/Dictionaries#Czech_.28Czech_Republic.29

[10] Linguistic Data Consortium, Czech Academic Corpus 2.0 http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2008T22

[11] Osobná stránka pána Václava Havla http://www.vaclavhavel.cz

Obsah CD-ROM

Elektronická verzia práce, rovnako ako implementácia aplikácie Interviewer, je obsiahnutá na priloženom CD-ROM. Taktiež obsahuje zdrojové kódy a korpus rozhovorov s pánom Václavom Havlom.

doc/	obsahuje elektronickú verziu tejto práce vo formáte PDF a niektoré dokumenty z použitej literatúry
Interviewer/	
/src/	zdrojové kódy aplikácie Interviewer
/data/	dátové súbory potrebné pre fungovanie aplikácie,napr. korpusy
/logfile/	obsahuje súbor, do ktorého sa zapisujú výsledky vyhľadávaní spolu so spokojnosťou uživateľa
/Interviewer.jar	spustiteľný súbor aplikácie Interviewer
CorpusEditor/	
/CorpusEditor.jar	spustiteľný súbor aplikácie CorpusEditor
/src/	zdrojové kódy aplikácie CorpusEditor
rozdelovac.sh	shell skript určený na rozdelenie a spracovanie korpusu