Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Výběr informací z lékařských zpráv

Plzeň 2015 Stanislav Škovran

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 5. května 2015

Stanislav Škovran

Poděkování

Děkuji Prof. Ing. Karlu Ježkovi, CSc., za odborné vedení mé bakalářské práce a za cenné rady, které mi pomohly tuto práci vytvořit.

V Plzni dne 5. května 2015

Stanislav Škovran

Abstract

Text mining for medical records.

The introduction of the theoretical part provides a basic outline of the possibility of general textmining followed by a description of the Java library Apache OpenNLP. The problem of anonymity of medical records is considered as well.

The practical part contains description of main ideas of the Java programs Anonymizer and ReportsInfoMiner. These programs are the essential part of the thesis. They perform the anonymization of the medical data and create relation database with structured data. The user documentation of the programs is placed at the attachment.

Abstrakt

V teoretické části je uveden obecný přehled vytěžování informací z nestrukturovaných textů. Je také zmíněna knihovna Apache OpenNLP. Dále je probírán problém anonymizace lékařských zpráv.

Realizační část se soustřeďuje na popis programu Anonymizer a ReportsInfoMiner. Tyto programy jsou náplní bakalářské práce a slouží k anonymizaci zpráv a k jejich strukturalizaci do relační databáze. V příloze je pak uživatelská dokumentace k oběma programům.

Obsah

1 Úvod

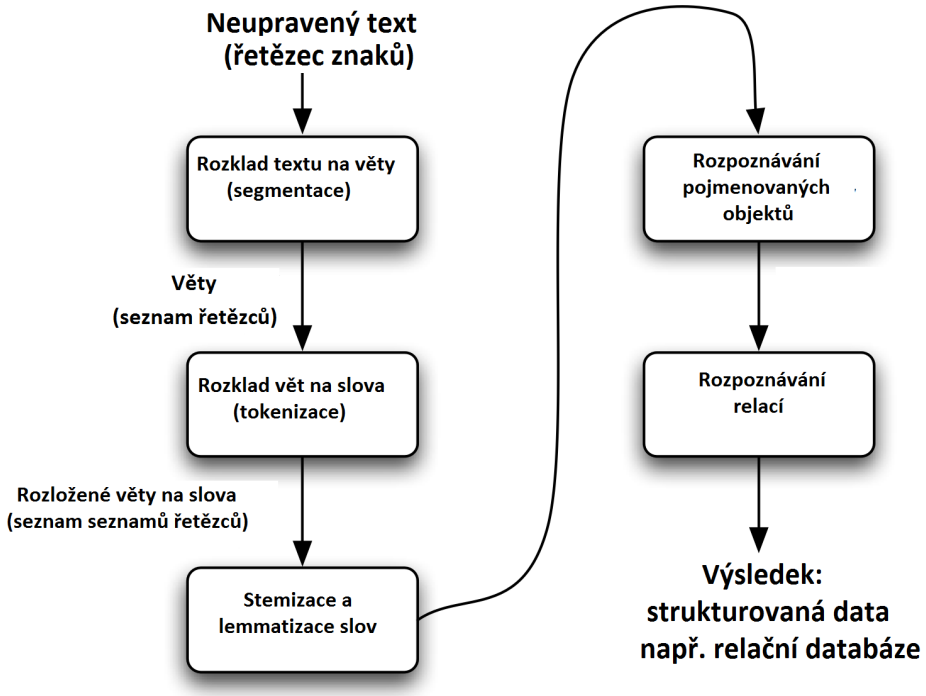
V lékařské dokumentaci se často vyskytují zprávy psané jako souvislý, nestrukturovaný text. Je obtížné z většího množství takovýchto textů vybírat důležité informace, případně vytvářet statistiky úkonů, podaných léků a dalších položek. Cílem této práce je vytvořit program, který dokáže vytěžit strukturované informace z takovýchto zpráv. Výstupem by měla být relační databáze. Její strukturu a definici vyhledávaných entit by měl mít uživatel možnost konfigurovat bez zásahu do programového kódu.

Se zpracováním lékařských zpráv je také spojen problém anonymizace dat. Ukazuje se jako důležité mít možnost nahradit ve zprávách osobní údaje (jméno, příjmení) anonymními tvary. S takto anonymizovanými zprávami pak provedeme strukturalizaci do databáze a jsou připraveny například pro statistické přehledy. Program pro anonymizaci je také součástí této bakalářské práce.

Jako programovací jazyk jsem použil Javu. Důvodem je využití javovské knihovny OpenNLP pro některé operace s textem.

2 Počítačové zpracování přirozeného jazyka

Proces zpracování přirozeného jazyka si můžeme zobrazit na obrázku 2.1.



Obrázek 2.1: Zpracování přirozeného jazyka

Projdeme si postupně všechny kroky tohoto procesu.

2.1 Rozklad textu na věty

Tento krok se zdá být nejjednodušší. Na rozdělení se dají použít interpunkční znaménka. Znaky „?!“ znamenají téměř vždy konec věty. Výjimky typu ”Yahoo! Inc.”jsou vzácné. Horší situace je se znakem tečka. Mějme následující text:

"Tak např. USA mají hlavní město Washington D.C. Situace ve světě na počátku 20. století. Na počátku 20. století se velmoci zaměřovaly na výboje mimo evropský kontinent – dobývání nových území v Asii a Africe. Koncem 19. st. však byly kolonie již rozděleny mezi velmoci, přičemž se měnil poměr sil mezi nimi."

Vidíme, že orientace podle interpunkčních znamének a velkých počátečních písmen zde nepovede k přesnému rozkladu na věty. Do algoritmu by se musel zabudovat seznam zkratek a provést i další úpravy. Jinou možností je využít na rozklad volně dostupný software. Rozhodl jsem se vyzkoušet knihovnu Apache OpenNLP [1].

OpenNLP je javovská knihovna pro zpracování textu v přirozeném jazyce, využívající strojové učení. Samotný algoritmus strojového učení je založen na principu maximalizace entropie. Knihovna je rozdělena do několika komponent:

SentenceModel - rozdělení textu do jednotlivých vět
TokenizerModel - rozdělení vět do slov
TokenNameFinderModel - nalezení konkrétních entit ve větách
DoccatModel - kategorizace celých dokumentů
ChunkerModel - rozdělení vět do syntakticky souvisejících skupin slov
POSModel - identifikace slov jako slovních druhů (Part-of-Speech Tagger) v závislosti na kontextu

Původně jsem předpokládal, že modul TokenNameFinderModel použiji na vyhledávání entit ve zdravotních zprávách, ale ukázalo se, že lepší výsledky dosáhnu vlastním algoritmem. Nakonec využívám knihovnu Apache-OpenNLP pouze pro rozklad textu na jednotlivé věty.

Všechny komponenty OpenNLP pracují na podobném principu. Vytvoří se instance modelu, která se inicializuje souborem s daty získanými při strojovém učení, potom se vytvoří instance vlastního nástroje, která zpracuje zkoumaný text:

Ukázka části kódu pro zpracování řetězce **String vstupniText** :

```
// priprava modelu
// "cz-sent.bin" je binarni soubor s daty ze strojoveho uceni
InputStream modelIn = new FileInputStream("cz-sent.bin");
SentenceModel model = new SentenceModel(modelIn);
// vytvoreni nastroje a zpracovani textu
SentenceDetector = new SentenceDetectorME(model);
String sentences[] = SentenceDetector.sentDetect(vstupniText);
```

V poli **sentences** jsou uloženy jednotlivé věty z **vstupniText**.

Ukažme si výsledek na našem konkrétním příkladu. Pole **sentences** obsahuje následující položky:

<Tak např. USA mají hlavní město Washington D.C.>
<Situace ve světě na počátku 20. století.>
<Na počátku 20. století se velmoci zaměřovaly na výboje mimo evropský kontinent – dobývání nových území v Asii a Africe.>
<Koncem 19. st. však byly kolonie již rozděleny mezi velmoci, přičemž se měnil poměr sil mezi nimi.>

Rozložení proběhlo naprosto správně.

2.2 Rozklad vět na slova

I zde by se dala použít knihovna OpenNLP. Používám ale vlastní algoritmus, který prochází větu a jako oddělovače slov bere všechny znaky, které nejsou obsaženy v následující sadě písmen:

"aáäbcčdd'eéěfghiíjklmnňoóöpqrřsštt'uúůüvwxyýzž"

Současně používám tento oddělovací algoritmus i jako filtr, který odstraňuje ze slov všechny znaky, které neodpovídají písmenu. Pokud jsem používal knihovnu OpenNLP, měl jsem problém například při zpracování vstupních textů ve formátu xml, kdy si knihovna nedokázala poradit s dělícími značkami <,>.

2.3 Stemmizace a lemmatizace slov

Vzhledem k velké ohebnosti češtiny je problém při vyhledávání slov v textu z důvodu různých tvarů jednoho slova. Řešením může být tzv. lemmatizace a stemmizace.

Při procesu lemmatizace je hledán základní neboli normalizovaný tvar daného slova tzv. „Lemma“. Za normalizovaný stav je považován u podstatných jmen první pád jednotného čísla, u přídavných jmen první pád jednotného čísla základního stupně mužského rodu, u sloves jde o tvar slova v infinitivu a podobně. Příklad: Být - je, byla, bude. Pán - pánům, pánovi, pane.

Stemming je podobný procesu lemmatizace až na skutečnost, že se v tomto případě nehledá normalizovaná forma, nýbrž kořen (stem) slova. Příklad: vodárna, vodovod, vodník, zavodnit, odvodnit, voda, ... společný kořen slova je „vod“.

Pro vyhledávání slov v textu budu používat metodu lemmatizace. Algoritmů pro lemmatizaci je celá řada. Dají se rozdělit do dvou základních skupin a to vyhledávací nad bází dat nebo algoritmy využívající pravidla o znalosti jazyka. Zde použiji vyhledávací algoritmus nad databází.
V programu Anonymizer budu používat metodu stemmizace. Využiji databázi jmen a příjmení se stemy.

2.4 Rozpoznávání objektů a vytvoření databáze

V okamžiku, kdy máme text rozdělený na věty a věty rozdělené na jednotlivá slova, můžeme přistoupit k poslednímu kroku zpracování - nalezení vybraných slov a případných relací mezi nimi a zařazení výsledku do relační databáze. Při vyhledávání slov využijeme lemmatizaci, tedy spíše opačný proces - k základnímu tvaru slova najdeme všechny odvozené tvary, ty vyhledáváme v textu a do databáze ukládáme základní tvar.

2.5 Anonymizace osobních údajů v textu

Vzhledem k citlivé povaze zpracovávaných lékařských dat se ukazuje jako nezbytné provést před zveřejněním textu jeho anonymizaci. Spočívá v nahrazení jmen a příjmení anonymními tvary. Protože počet různých jmen v textu není obecně omezený, zvolil jsem pro anonymizaci očíslované tvary jmen Eva Nováková a Adam Novák. Tedy například text

Až při závěrečné čtvrtině nedělního Pražského maratonu spatříte utíkat subtilní blondýnku s číslem R 6644 na červeném podkladě, jakým se budou lišit čísla členů štafet, vězte, že ho nese nejlepší česká triatlonistka ***Vendula Frintová****. Před ní ve stejném týmu osobností poběží skifař* ***Ondřej Synek****, biatlonista* ***Michal Šlesingr*** *a herec a moderátor* ***Dalibor Gondík****.*

se převede na text

Až při závěrečné čtvrtině nedělního Pražského maratonu spatříte utíkat subtilní blondýnku s číslem R 6644 na červeném podkladě, jakým se budou lišit čísla členů štafet, vězte, že ho nese nejlepší česká triatlonistka ***Eva001 Nováková001****. Před ní ve stejném týmu osobností poběží skifař* ***Adam001 Novák001****, biatlonista* ***Adam002 Novák002*** *a herec a moderátor* ***Adam003 Novák003****.*

Součástí anonymizace musí být také vytvoření souboru se seznamem všech nahrazených jmen a příjmení spolu s jejich náhradami - pro případnou zpětnou interpretaci. Tedy zde by to bylo:

Vendula	*Eva001*
Frintová	*Nováková001*
Ondřej	*Adam001*
Synek	*Novák001*
Michal	*Adam002*
Šlesingr	*Novák002*
Dalibor	*Adam003*
Gondík	*Novák003*

3 Program Anonymizer

Uživatelská příručka je v Přílohách. Zde popíšu princip programu a dosažené výsledky.

3.1 Základní mechanismus

Anonymizer slouží k nahrazování jmen a příjmení v textu anonymními náhradami. Největší problém je samozřejmě nalézt v textu všechna jména a příjmení. Původně jsem chtěl použít knihovnu OpenNLP ale pak jsem našel na internetu soubor se seznamem jmen a příjmení a jejich 5. pádů. [2]. Na uvedené adrese je ke stažení zip soubor se šesti soubory ve formátu csv. Tyto soubory obsahují pro každé jméno a příjmení tři položky: četnost, 1. pád, 5. pád. Soubory jsem převedl do sqlite databáze NamesRoots.db, která obsahuje 4 tabulky: ženská jména, mužská jména, ženská příjmení, mužská příjmení. Každá tabulka má strukturu:

ID	- pořadové číslo (primary key)
Frequency	- četnost v populaci
Nominative	- 1. pád
Vocative	- 5. pád
Root	- kořen slova

Name	Type	Schema
Tables (4)		
FORENAME_FEMALE		CREATE TABLE FORENAME_FEMALE (ID INT PRIMARY KEY NOT NULL,Frequency INT,Nominative TEXT,Vocative TEXT,Root TEXT)
ID	INT	`ID` INT NOT NULL
Frequency	INT	`Frequency` INT
Nominative	TEXT	`Nominative` TEXT
Vocative	TEXT	`Vocative` TEXT
Root	TEXT	`Root` TEXT
FORENAME_MALE		CREATE TABLE FORENAME_MALE (ID INT PRIMARY KEY NOT NULL,Frequency INT,Nominative TEXT,Vocative TEXT,Root TEXT)
ID	INT	`ID` INT NOT NULL
Frequency	INT	`Frequency` INT
Nominative	TEXT	`Nominative` TEXT
Vocative	TEXT	`Vocative` TEXT
Root	TEXT	`Root` TEXT
SURNAME_FEMALE		CREATE TABLE SURNAME_FEMALE (ID INT PRIMARY KEY NOT NULL,Frequency INT,Nominative TEXT,Vocative TEXT,Root TEXT)
ID	INT	`ID` INT NOT NULL
Frequency	INT	`Frequency` INT
Nominative	TEXT	`Nominative` TEXT
Vocative	TEXT	`Vocative` TEXT
Root	TEXT	`Root` TEXT
SURNAME_MALE		CREATE TABLE SURNAME_MALE (ID INT PRIMARY KEY NOT NULL,Frequency INT,Nominative TEXT,Vocative TEXT,Root TEXT)
ID	INT	`ID` INT NOT NULL
Frequency	INT	`Frequency` INT
Nominative	TEXT	`Nominative` TEXT
Vocative	TEXT	`Vocative` TEXT
Root	TEXT	`Root` TEXT

Obrázek 3.1: Struktura databáze pro stemming

Table: FORENAME_FEMALE

	ID	Frequency	Nominative	Vocative	Root
	Filter	Filter	Filter	Filter	Filter
1	1	316559	Marie	Marie	Mari
2	2	274303	Jana	Jano	Jan
3	3	160317	Eva	Evo	Ev
4	4	150573	Hana	Hano	Han
5	5	148688	Anna	Anno	Ann
6	6	124871	Věra	Věro	Věr
7	7	119366	Lenka	Lenko	Lenk
8	8	110936	Kateřina	Kateřino	Kateřin
9	9	110046	Alena	Aleno	Alen
10	10	103702	Lucie	Lucie	Luci
11	11	102321	Petra	Petro	Petr
12	12	94475	Jaroslava	Jaroslavo	Jaroslav
13	13	85939	Ludmila	Ludmilo	Ludmil
14	14	82985	Helena	Heleno	Helen
15	15	81543	Martina	Martino	Martin

Obrázek 3.2: Část databáze pro stemming ženských jmen

Na obrázcích 3.1 a 3.2 je struktura databáze a část tabulky pro ženská jména. Položku Root jsem odvodil z rozdílu mezi 1. a 5. pádem + opravy pro některé typy jmen a příjmení. Root je klíčový v algoritmu pro určení, zda je slovo jménem.

Třída **NameStemmer** v balíku **stemmer** obsahuje hlavní algoritmus založený na spolupráci s databází NamesRoots.db. Hlavní metoda:

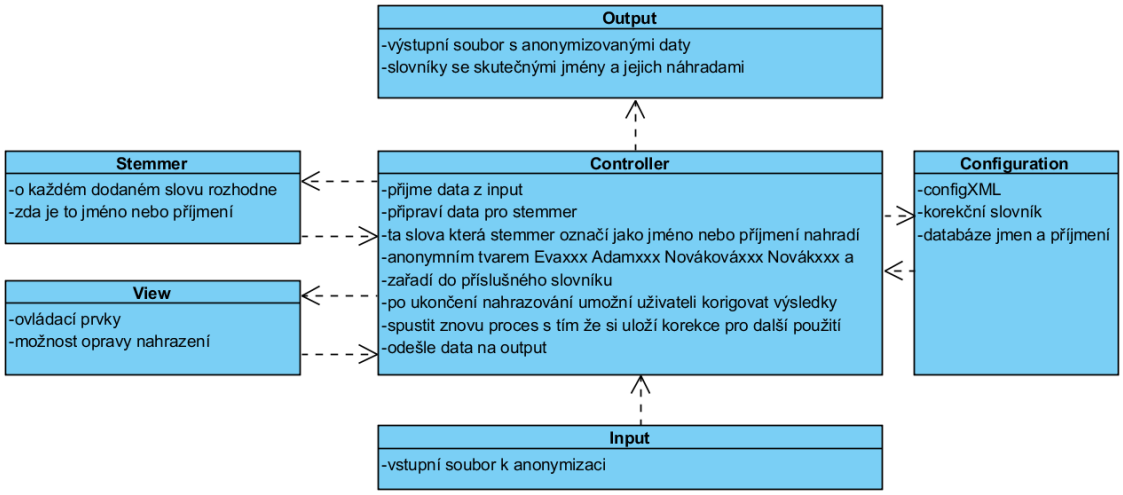
```
/**
 * Metoda digNames
 * vybere z vety slova, ktera by mohla byt jmena nebo prijmeni
 * @param words - pole prohledavanych slov
 * @param type - jaky typ hledam - FORENAME_MALE,
    FORENAME_FEMALEMALE,SURNAME_MALE, SURNAME_FEMALEMALE
 * @return
 * @throws java.lang.Exception
 */
public String[] digNames(String[] words, NameType type)throws
    Exception{
  List<String> ret = new ArrayList<>();
  for (String word : words){
    if (word == null || word.isEmpty()) continue;
    CharSequence cs = word.substring(0, 1);
    if (MyTools.UPPER_CASES.contains(cs)){
      // prvni pismeno je velke
      if (fitType(word,type))
        ret.add(word);

    }
  }
  return (String[]) ret.toArray(new String[ret.size()]);
}
```

Metodu **digNames** volám zvlášť pro mužské a ženské tvary jmen a příjmení (tedy celkem 4x). Ukládám totiž slovník náhrad do čtyř různých souborů. Metoda **fitType** je klíčová v celém algoritmu, zjišťuje, zda dané slovo odpovídá některému jménu z databáze:

```
/**
 * Metoda fitType
 * klicova metoda cele tridy, zjistuje, zda slovo je jmeno nebo
    prijmeni
 * @param word - zkoumane slovo
 * @param type - jaky typ jmena hledam
 * @return true = word je pozadovaneho typu
 * @throws Exception
```

```
 */
public boolean fitType(String word, NameType type) throws
    Exception{
  if (word == null || word.isEmpty())
    return false;
  if (type != NameType.SURNAME_FEMALE){
    /* odblokujeme zenska prijmeni - slova s koncovkou -ova (s
       dlouhym a) mohou byt jen zenska prijmeni, zkoumame je
       zvlast v metode isFemale
    */
    if (isFemale(word, NameType.SURNAME_FEMALE))
      return false;
  }
  // existuje word primo v DB ale ne jako Root? pak jsme hotovi
  NameComplete NC = getFromDB(word, type);
  if (NC != null) return true;

  //najdeme v DB nejdelsi Root, ktery se shoduje se zacatkem word
  NC = getRootFromDB(word, type);
  // nenasel se v DB ani shodny Root, koncime
  if (NC == null) return false;
  /* Root se nasel, posledni kontrola, zda opravdu Root odpovida
     word v jinem gramatickem tvaru*/
  return NC.checkCase(name) != Cases.ROOT_0;
}
```

Schema celého programu je na Obr. 3.3. Zdrojový kód hlavní metody je v příloze A.1.



Obrázek 3.3: Schema programu Anonymizer

3.2 Funkčnost programu

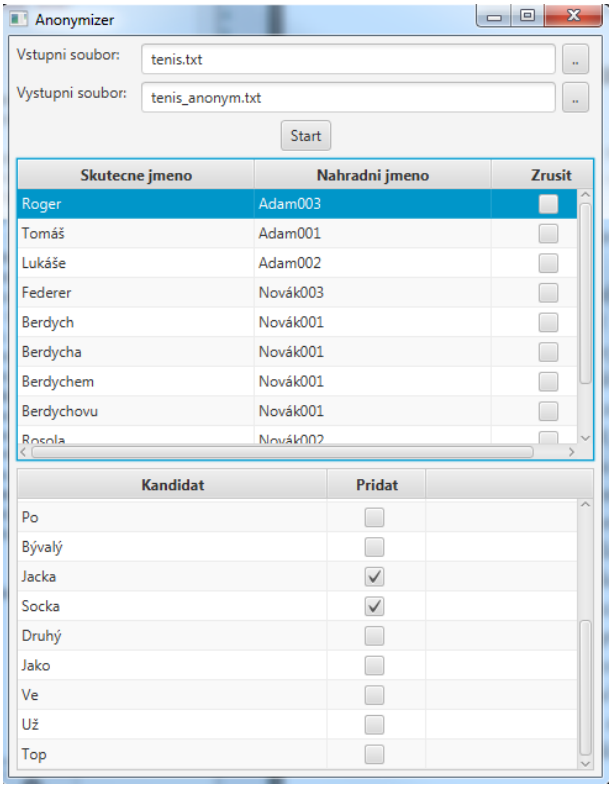
Ukážeme si funkčnost programu na testovacím příkladu a také vyzkoušíme výkonnost programu na větších datech.

Mějme následující text:

Tomáš Berdych porazil ve čtvrtém kole turnaje v Indian Wells v české tenisové bitvě Lukáše Rosola 6:2, 4:6 a 6:4. Coby nasazená devítka se probil mezi osm nejlepších. Nyní ho čeká druhý hráč světa Roger Federer ze Švýcarska. Zápas dvou momentálně nejlepších českých tenistů vypadal v prvním setu jako jasná záležitost pro devátého nasazeného Berdycha. Úvodní sadu získal za 27 minut, ve druhé se ale hra vyrovnala a Rosol dokázal soupeři dvakrát sebrat servis. Třetí set, v němž oba spolehlivě hájili svá podání, rozhodl Berdych brejkem v desáté hře. Po hodině a 52 minutách proměnil druhý mečbol a Rosola porazil i ve druhém vzájemném zápase. Federer oslavil na turnaji v Indian Wells 50. vítězství. Bývalý první hráč světa, jenž už na kalifornském podniku získal rekordní čtyři tituly, porazil domácího Jacka Socka 6:3, 6:2. Druhý nasazený hráč, jenž v Indian Wells startuje už po patnácté, se bez problémů vyrovnal s větrným počasím a dvaadvacetiletého soupeře porazil za hodinu a devět minut. Jako jasný favorit půjde i do souboje s Berdychem. Ve vzájemné bilanci vede 12:6. „Už si ani nepamatuju, kdy jsme spolu hráli poprvé, ale sledoval jsem, jak se jeho hra vyvíjela a jak se dokázal usadit v

Top 10,“ řekl Federer na Berdychovu adresu na tiskové konferenci po utkání.

Na obrázku 3.4 je vidět výsledek anonymizace textu programem Anonymizer



Obrázek 3.4: Výsledek anonymizace po 1. kroku

V první tabulce jsou zobrazeny provedené náhrady, ve druhé tabulce všechna slova začínající velkým písmenem, která nebyla vyhodnocena jako jména nebo příjmení. Ve sloupci "Přidat"máme možnost nastavit je také k náhradě. To jsme provedli pro jméno Jacka Socka. Spustíme znovu celý proces a napodruhé dostaneme ve výstupním souboru **tenis_anonym.txt** následující text:

Adam001 Novák001 porazil ve čtvrtém kole turnaje v Indian Wells v české tenisové bitvě Adam002 Novák002 6:2, 4:6 a 6:4. Coby nasazená devítka se probil mezi osm nejlepších. Nyní ho čeká druhý hráč světa Adam003 Novák003 ze Švýcarska. Zápas dvou momentálně nejlepších českých tenistů vypadal v prvním setu jako jasná záležitost pro devátého nasazeného Novák001a. Úvodní sadu získal za 27 minut, ve druhé se ale hra vyrovnala a Novák002 dokázal soupeři dvakrát sebrat servis. Třetí set, v němž oba spolehlivě hájili svá podání, rozhodl Novák001 brejkem v desáté hře. Po hodině a 52 minutách proměnil druhý mečbol a Novák002 porazil i ve druhém vzájemném zápase. Novák003 oslavil na turnaji v Indian Wells 50. vítězství. Bývalý první hráč světa, jenž už na kalifornském podniku získal rekordní čtyři tituly, porazil domácího Novák004 Novák005 6:3, 6:2. Druhý nasazený hráč, jenž v Indian Wells startuje už po patnácté, se bez problémů vyrovnal s větrným počasím a dvaadvacetiletého soupeře porazil za hodinu a devět minut. Jako jasný favorit půjde i do souboje s Novák001. Ve vzájemné bilanci vede 12:6. „Už si ani nepamatuju, kdy jsme spolu hráli poprvé, ale sledoval jsem, jak se jeho hra vyvíjela a jak se dokázal usadit v Top 10," řekl Novák003 na Novák001 adresu na tiskové konferenci po utkání.

Vidíme zde hlavní nedostatek programu - nahrazené anonymní tvary jsou jen v 1. pádu. Odstranění tohoto nedostatku by si vyžádalo další analýzu a rozšíření třídy NameStemmer. Pro naše účely získávání strukturovaných informací z lékařských zpráv je ale zatím toto nahrazování dostatečné. Ještě si uvedeme jak vypadají soubory, kde jsou uloženy slovníky náhrad:

Soubor **tenis_surname.txt**:

Federer	- Novák003
Berdycha	- Novák001
Berdych	- Novák001
Berdychem	- Novák001
Berdychovu	- Novák001
Rosol	- Novák002
Rosola	- Novák002

Soubor **tenis_forename.txt**:

Roger - Adam003
Tomáš - Adam001
Lukáše - Adam002
Lukáš - Adam002

Program správně rozeznal různé tvary stejných jmen a příjmení a přiřadil jim stejnou náhradu.

Výkonnost programu:

Program jsem použil pro anonymizaci lékařských zpráv z Domova seniorů. Obsahovaly 36 000 položek. Anonymizace trvala cca 120 minut na následující konfiguraci:

Windows 7 64-bit SP1
Intel Core i5-4460 CPU @ 3,20GHz, 8,0GB RAM.

4 Program ReportsInfoMiner - vytěžování informací

Uživatelská příručka je opět v Přílohách.

4.1 Základní mechanismus

Program ReportsInfoMiner slouží k vytěžení a strukturalizaci dat ze souvislého textu. Uživatel má možnost ovlivnit parametry programu. Vzhledem ke specifikaci lékařských zpráv jsem zvolil dva formáty vstupních dat:

1. textový soubor *.txt
2. sqlite databázový soubor *.db

Textový formát jsem vybral pro jeho obecnost a databázový formát proto, že lékařské zprávy mohou být částečně strukturovány a uživatel požaduje vytěžovat informace jen z určité části. Příklad:

Mějme zprávy v excelovském souboru - obrázek 4.1.

	Datum	Čas	Zápis	ID obyvatele
138	22.3.2014	19:08	Zápis: Strženiny na obou horních končetinách a defekt v sakru jsou nadále v péči sester. Sestru není nutné	58
139	22.3.2014	9:44	Zápis: Dnes proběhla kontrola opruzenin v oblasti konečníku a tísel - nadále přetrvávají. V ošetřování přímé péče - omýt, osušit a namazat tenkou vrstvou zinkové masti.; Dne: 22.3.2014	58
140	20.3.2014	11:02	Zápis: (Zapsal: xxx) Při ranní hygieně zjištena opruzenina v oblasti konečníku a tříšel. V ošetřování přímé péče. při ranní a večerní hygieně omýt, osušit a namazat tenkou vrstvou zinkové masti.; Dne: 20.3.2014	58
141	24.3.2014	7:15	Zápis: Opruzeniny v oblasti konečníku a tříšel přetrvávají. Nadále v ošetřování přímé péče. Při ranní a večerní hygieně omýt, osušit a namazat tenkou vrstvou Zinkové masti.; Dne: 24.3.2014	58
142	7.3.2014	9:58	Zápis: Po domluvě se zdravotní sestrou se u obyvatelky jedná o trvalou změnu pigmentace. Sakrum se bude při ranní a večerní hygieně promazávat Regenerační masti.; Dne: 7.3.2014	60
143	5.3.2014	20:23	Zápis: Při ranní hygieně byla zjištěna suchá zrohovatělá kůže v sakru. Při ranní a večerní hygieně promazávat Regenerační masti. Sestra byla informována.; Dne: 5.3.2014	60
144	1.3.2014	6:53	Zápis: Mírné začervenání na levém kotníku přetrvává,nehoří se.Při ranní a večerní hygieně promazávat Emspomou. Pokud bude obyvatel v lůžku dávat antidekubitní botičky, vypodkládat dolní končetiny tak aby se paty nedotýkali podložky. Při vysazování do invalidního vozíku vkládat do ponožky vatu.; Dne: 1.3.2014	30

Obrázek 4.1: Příklad zdravotních zpráv ve formátu Excel

Uživatel může chtít extrahovat jen informace ze sloupce se zápisem. Ostatní položky by rád zachoval. Excelovský soubor snadno převedeme na sqlite databázi (excel -> csv formát -> sqlite) - Obr. 4.2.

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: Zapis

	field1	field2	field3	field4
	Filter	Filter	Filter	Filter
1	22.3.2014	19:08	Zápis: Strženiny na obou horních končetinách a defekt v sakru jsou nadále v péči sester. Sestr...	58
2	22.3.2014	9:44	Zápis: Dnes proběhla kontrola opruzenin v oblasti konečníku a tísel - nadále přetrvávají. V oše...	58
3	20.3.2014	11:02	Zápis: (Zapsal: xxx) Při ranní hygieně zjištena opruzenina v oblasti konečníku a třísel. V ošetřov...	58
4	24.3.2014	7:15	Zápis: Opruzeniny v oblasti konečníku a třísel přetrvávají. Nadále v ošetřování přímé péče. Při ...	58
5	7.3.2014	9:58	Zápis: Po domluvě se zdravotní sestrou se u obyvatelky jedná o trvalou změnu pigmentace. S...	60
6	5.3.2014	20:23	Zápis: Při ranní hygieně byla zjištěna suchá zrohovatělá kůže v sakru. Při ranní a večerní hygie...	60
7	1.3.2014	6:53	Zápis: Mírné začervenání na levém kotníku přetrvává,nehorší se.Při ranní a večerní hygieně pr...	30

Obrázek 4.2: Zprávy převedené z excelu do sqlite.

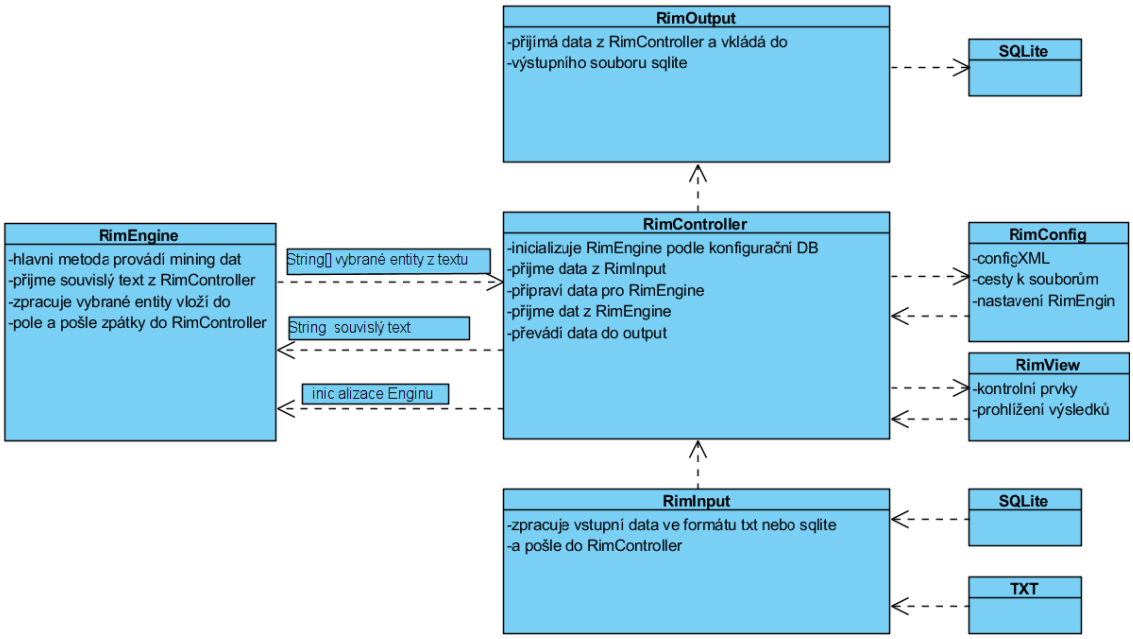
Výsledek zpracování programem je na Obr. 4.3.

Table: t_RIM

	___ID	field1	field2	POTIZE	LOKALIZACE	LECBA	LEKY	_OriginalniText	field4
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	22.3.2014	19:08	strženiny				Zápis: Strženiny na obou horních k...	58
2	2	22.3.2014	9:44	opruzenina	konečník	omýt, osušit, namazat	zinkový	Zápis: Dnes proběhla kontrola opru...	58
3	3	20.3.2014	11:02	opruzenina	konečník, tříslo	omýt, osušit, namazat	zinkový	Zápis: (Zapsal: xxx) Při ranní hygien...	58
4	4	24.3.2014	7:15	opruzenina	konečník	omýt, osušit, namazat	zinkový	Zápis: Opruzeniny v oblasti koneční...	58
5	5	7.3.2014	9:58			promazávat	mast	Zápis: Po domluvě se zdravotní sest...	60
6	6	5.3.2014	20:23			promazávat	mast	Zápis: Při ranní hygieně byla zjištěn...	60
7	7	1.3.2014	6:53	začervenání	kotník	promazávat	emspomou	Zápis: Mírné začervenání na levém ...	30

Obrázek 4.3: Tabulka se záznamy po zpracování programem RimInfoMiner.

Sloupec field3 byl nahrazen čtyřmi novými sloupci POTIZE, LOKALIZACE, LECBA, LEKY, které obsahují vytěžená data a sloupcem _OriginalniText, který obsahuje původní text.
Vzhledem k povaze výstupních strukturovaných dat je výstup v jediném formátu a to sqlite databáze s jednou tabulkou.



Obrázek 4.4: Schema programu ReportsInfoMiner

Hlavní částí celého programu je třída RimEngine.

```
/**
 * trida pro extrakci dat ze souvisleho textu
 * @author A12B0064K
 */
public class RimEngine {

  // rozdelovac textu na vety (pomoci OpenNLP)
  private SentenceDetectorME m_SentenceDetector;

  /* datovy objekt, obsahujici konfiguraci miningu - co se ma
     hledat a kam to zaradit */
  RimTableContainer m_ColumnsEntities;

  /**
   * nastavi modulu SentenceDetectorME - pro rozdeleni textu do vet
   * @param cfg konfiguracni objekt s cestami k souborum a dalsimi
      udaji
   * @throws Exception
   */
  public void initOpenNLP(RimConfig cfg)throws Exception{
```

```
  InputStream is = new FileInputStream(cfg.m_cSentenceFile);
  SentenceModel model = new SentenceModel(is);
  is.close();
  m_SentenceDetector = new SentenceDetectorME(model);
}

/**
 * inicializace m_ColumnsEntities - objekt, kde je ulozena
    konfigurace miningu
 * @param miningConfiguration - objekt vytvoreny z konfiguracni
    databaze miningu
 */
void setMiningConfiguration(RimTableContainer
   miningConfiguration){
  m_ColumnsEntities = miningConfiguration;
}

boolean isInitialized(){
  return m_ColumnsEntities != null;
}

/**
 * rozlozi cText do jednotlivych vet
 * @param cText plny text
 * @return pole stringu obsahujici jednotlive vety
 */
String[] splitIntoSentences(String cText){
  String sentences[];
  sentences = m_SentenceDetector.sentDetect(cText);
  return sentences;
}

/**
 * hlavni metoda enginu, vrati RimTableContainer, ktery je ve
    stejnem tvaru jako vstupni m_ColumnsEntities, ale jeho prvky
    jsou nalezene entity v textu
 * @param cText - vstupni nestrukturovany text z ktereho tezime
    informace
 * @return strukturovany objekt s nalezenymi entitami a jejich
    umistenimi
 * @throws Exception
 */
public RimTableContainer textMining(String cText)throws Exception{
```

```
    // rozdeleni textu na vety
    String[] sentences = splitIntoSentences(cText);
    RimTableContainer ret = null;

    // prochazime text po vetach, do RimTableContainer pridavame
       nalezena slova
    for (String oneSentence : sentences){
      String[] words = MyTools.tokenizeSentence(oneSentence);
      for (String word : words){
        String[] found = m_ColumnsEntities.getBaseShapes(word);
        if (found != null){
          /* v textu nalezen tvar slova, ktere hledame, pridame do
             vystupu, vcetne jeho zakladniho tvaru a sloupce kam
             prijde zaradit */
          // found[0] - nazev sloupce kam vyraz prijde
          // found[1] - zakladni tvar nalezeneho slova
          // found[2] - puvodni tvar nalezeneho slova
          if (ret == null)
            ret = new RimTableContainer();
          ret.putShapeItemToColumn(found[0], found[1], found[2]);
        }
      }
    }
    return ret;
  }
}
```

Základní datová struktura, která slouží ke konfiguraci miningu a také k uložení výsledku je třída RimTableContainer. Slouží k uložení 3D-tabulky, kde jeden rozměr jsou skupiny entit (např Léky), druhý rozměr jsou jednotlivé entity ve skupině (např. antibiotika, paralen) a třetí rozměr jsou gramatické tvary jedné entity (např. paralenu, paralenem, paraleny). Objekt je realizován pomocí třídy HashMap. Zdrojový kód je v příloze A.2. Nastavení miningu je uloženo v sqlite databázi **health_care.db**, která je součástí konfigurace programu. Změnou této databáze můžeme měnit parametry vyhledávání, jak je popsáno v uživatelské příručce. Tato databáze je vlastně obrazem datového objektu **RimTableContainer**, obsahuje ale pouze základní tvary slov. Jejich další gramatické tvary se doplní do **RimTableContainer** až programově z databáze **vocabulary.db**, která obsahuje celkem přes dva miliony českých slov a jejich tvarů. Pokud k jednomu odvozenému tvaru existuje více

základních tvarů, jsou uvedeny všechny, oddělené "|"- Obr. 4.5.

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: vocabulary

	id	word	base_shapes
	Filter	Filter	Filter
1	1	abakus	abak|abakus
2	2	abaku	abak|abaka|aba...
3	3	abatyše	abatyše
4	4	abatyši	abatyše
5	5	abatyší	abatyše
6	6	abatyším	abatyše
7	7	abatyších	abatyše
8	8	abatyšemi	abatyše
9	9	abdikace	abdikace
10	10	abdikaci	abdikace
11	11	abdikací	abdikace
12	12	abdikacím	abdikace
13	13	abdikacích	abdikace
14	14	abdikacemi	abdikace
15	15	abeceda	abeceda
16	16	abecedy	abeceda
17	17	abecedě	abeceda
18	18	abecedu	abeceda

|< < 1 - 18 of 2176498 > >|

Obrázek 4.5: Část tabulky s českými slovy a jejich tvary.

Při prvním spuštění programu se načte obsah **health_care.db**, vloží do **RimTableContainer**, doplní tvary slov z **vocabulary.db**. Tato operace trvá jistou dobu, řádově 10-20 sekund. Při ukončení programu se objekt **RimTableContainer** uloží pomocí serializace do souboru **health_care.map**. Při příštím otevření programu se načítá **RimTableContainer** serializací z tohoto souboru což je již rychlá operace.

Jednotlivé moduly programu (**RimInput**, **RimOutput**, **RimEngine**) jsou relativně samostatné části, které mohou být dále rozšířeny. Hlavně vstup a výstup je možno na základě požadavků upravit na další formáty aniž by se muselo něco měnit na výkonné jednotce RimEngine.

4.2 Práce s SQLite databází

V celém projektu využívám databázi SQLite. Je pro to několik důvodů. Je to souborový databázový systém, jedna databáze = jeden soubor. Není třeba spouštět žádný server - stačí knihovní funkce Javy. Navíc existuje volně k dispozici výborný SQLite manager **DB Browser for SQLite** [3]. Pro naše účely naprosto dostačující software s intuitivním ovládáním. Všechny screenshoty v této práci, které se týkají databází, jsou z prostředí tohoto programu. Program je součástí CD u této práce.

Javovská knihovna pro práci s SQLite databází je k dispozici například tady [4]. Pro práci s SQLite jsem si vytvořil wrapper **MySQLiteTool**. Zdrojový kód je v příloze.

4.3 Funkčnost programu

Ukážeme si funkčnost programu na testovacím příkladu.

V předchozí kapitole jsme ukázali, jak funguje program, pokud je vstupní soubor ve formátu SQLite. Teď si ukážeme případ, kdy vstupem bude obecný textový formát. Všude budeme předpokládat formátování UTF-8.

Mějme následující text: (jde o autentický zdravotní záznam z Domu seniorů, který prošel anonymizací)

Paní Nováková020 odmítla oběd z důvodu nevolnosti. Podány piškoty a čaj.; Dne: 7.10.2013"
Postřeh přímé péče - z důvodu zhoršení mobility, bude u obyvatelky RHB vyzkoušen přesun zvedákem s područkama.; Dne: 7.10.2013"
Při odpolední výměně inkontinentních pomůcek měla obyvatelka 1x řídkou stolici, SZP informována.; Dne: 7.10.2013"
Mluvil jsem s obyvatelkou. V současné době se cítí slabá, ale sama mluvila, že by opět ráda chodila. Bavili jsme se o možnosti instalace závěsu - to razantně odmítla. Uvedla, že se s prvním rozhodnutím unáhlila, pak až jí došlo, že by jí především chybělo madlo, které má nyní na dveřích. Využívá ho při vstávání z WC, při chůzi v chodbičce. To je hlavní důvod odmítnutí. Souhlasila s návrhem, že pokud bychom nějak vyřešili madla a možnost přichycení, závěs by akceptovala. Obyvatelka mi děkovala mi za návštěvu a rozovor (i na jiná témata), a že hledáme možnosti a řešení jejího docházení na WC.
Přijedou v pátek odpoledne na návštěvu, zůstanou do soboty. Zdá se mu, že obyvatelka ""chřadne"". Myslí si, že je to především z důvodu omezení nabídky aktivit, že je maminka odevzdanější, pasivnější a chybí jí ""elán""a motivace. Během podvečerních hodin vykonala obyvatelka stolici opět do plenkových kalhotek bez toho, aby si zazvonila, na lůžku při večerní hygieně absolutně nespolupracovala, manipulace ve dvou osobách byla složitá a namáhavá, na hýždích a v sakru jsou patrná začervenání a otlaky, motivace k tomu, aby se obyvatelka nechala napolohovat jinak nebyla úspěšná, ačkoliv obyvatelce bylo vysvětleno, co všechno se může stát v případě toho, že polohu v lůžku nezmění, ani si ji nenechá změnit zaměstnanci.;Podána antibiotika. Dne: 7.10.2013"
Obyvatelka byla dnes vysazena do invalidního vozíku a dána na chodbu, kde si povídala s ostatními obyvateli a byla spokojená. Na vozík se obyvatelka přesunula sama jen s mírnou dopomocí. ; Dne: 8.10.2013"
Obyvatelka byla po celý den spavá a téměř nepila a nejedla. Prosím, obyvatelka pravidelně a aktivně nabízet tekutiny a kontrolavat, zda-li jí, popřípadně dopomoct u jídla.; Dne: 9.10.2013"
(Zapsal: Eva019 Nováková022) Obyvatelka z rána nesnědla nic. Při podávání oběda jídlo odmítla. Jako důvod uvedla, že jíst nebude, at jí nic nenutíme.; Dne: 9.10.2013"
Obyvatelka při noční a ranní kontrole komunikovala bez problémů, orientovaná. Při polohování a výměně inkontinence spolupracovala. Nabízené tekutiny neodmítala, pila s lahvičky sama, bez problémů. Signalizaci v noci nepoužila.; Dne: 12.10.2013"

Program vezme jako jednu jednotku zprávy jeden odstavec, který je ukončený znakem konce řádku. Pro ni vytvoří záznam v databázi. Převede jen ty záznamy, ve kterých se vyskytuje aspoň jedna entita, kterou máme zaznamenánu v konfiguračním souboru **health_care.db**. Výsledek je na obrázku 4.6

___ID	POTIZE	LOKALIZACE	LECBA	LEKY	_OriginalniText
Filter	Filter	Filter	Filter	Filter	Filter
1	řídký, stolice				Při odpolední výměně inkontinentních pomůcek měla obyva...
2	stolice, začervenání			antibiotika	Během podvečerních hodin vykonala obyvatelka stolici opět ...
3	inkontinence				Obyvatelka při noční a ranní kontrole komunikovala bez prob...

Obrázek 4.6: Výsledek strukturalizace textu.

Množství nalezených informací by šlo zvětšit změnou konfigurace **health_care.db**.

Výkonnost programu

Program jsem zkoušel na anonymizovaných datech z Domu seniorů. Původně měl záznam 25000 položek zápisů v excelovském souboru. Převedl jsem ho do sqlite databázového souboru a spustil na něj program **ReportsInfoMiner**. Po zhruba tříhodinovém zpracování vyšel jako výsledek soubor s 8600 záznamy v databázi - Obr. 4.7.

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: SPS_anonym_RIM

	___ID	field1	field2	field3	POTIZE	LOKALIZACE	LECBA	LEKY
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
88	88	Novák010 Adam004	12.8.2014	18:28				strava
89	89	Novák010 Adam004	10.8.2014	12:11	zvracet			strava
90	90	Novák010 Adam004	14.8.2014	13:12				strava
91	91	Novák010 Adam004	14.8.2014	14:57				strava
92	92	Novák010 Adam004	19.8.2014	07:26	začervenání		omýt, osušit, namazat	zinkový
93	93	Novák010 Adam004	17.8.2014	10:49	začervenání		omýt, osušit, namazat	zinkový
94	94	Novák010 Adam004	17.8.2014	12:40				strava
95	95	Novák010 Adam004	23.8.2014	06:33	začervenání		omýt, osušit, namazat	zinkový
96	96	Novák010 Adam004	21.8.2014	07:05	začervenání		omýt, osušit, namazat	zinkový
97	97	Novák010 Adam004	21.8.2014	16:00	řídký			
98	98	Novák010 Adam004	25.8.2014	06:34	začervenání			
99	99	Novák010 Adam004	29.8.2014	12:46		ruka		
100	100	Novák010 Adam004	2.9.2014	12:28	bolest	ruka	promazat, promazávat	mast, emspomou
101	101	Novák010 Adam004	9.9.2014	10:59	inkontinence			
102	102	Novák010 Adam004	9.9.2014	13:19	inkontinence, bolest			
103	103	Novák010 Adam004	8.9.2014	11:00	bolest	kyčel		emspomou
104	104	Novák010 Adam004	16.9.2014	10:00			podávat	
105	105	Novák010 Adam004	23.9.2014	17:27				strava
106	106	Novák010 Adam004	23.9.2014	10:06	teplota			
107	107	Novák010 Adam004	6.10.2014	15:24		záda	promazat	emspomou
108	108	Nováková020 Eva017	3.9.2013	12:40		noha		
109	109	Nováková020 Eva017	3.9.2013	13:23	bolest	koleno, noha		
110	110	Nováková020 Eva017	27.9.2013	15:41	bolest			
111	111	Nováková020 Eva017	30.9.2013	17:50	řídký	břicho		strava
112	112	Nováková020 Eva017	30.9.2013	18:37	průjem, modřina, zvracet			
113	113	Nováková020 Eva017	30.9.2013	09:34	inkontinence			

|< < 88 - 113 of 8604 > >|

Obrázek 4.7: Část strukturovaných dat.

Zhodnocení výsledků

Program spolehlivě najde všechny entity, které jsme zadali v konfigurační databázi. Jak vidíme na Obr. 4.6 a na Obr. 4.7, někdy najde v jednom záznamu jen jednu entitu, ke které neexistuje další bližší určení. Pak by asi mělo nastoupit dolaďování konfigurace, přidávání dalších entit. To už záleží na uživateli, jaké informace a jak přesně bude chtít z textu dostat.

Problém je, pokud entita v textu vystupuje v opačném smyslu, například "Od rána bez teplot." Program najde pouze entitu "teplota" a vloží ji do databáze. Zde bych viděl asi hlavní směr dalšího vylepšení programu.

5 Závěr

Výsledkem bakalářské práce jsou dva programy pro anonymizaci a strukturalizaci textu. Původní podnět k bakalářské práci přišel z praxe, konkrétně z jednoho Domova pro seniory. Byl jsem v kontaktu s panem ředitelem a on mi popsal jak by si představoval program, který by mu pomohl automatizovat sledování péče o seniory. Měl konkrétní požadavky, co by se mělo zahrnout do vyhledávání. Ale hlavně chtěl, aby měl možnost program konfigurovat a volit, co sledovat a jak výsledky strukturovat. Bohužel pan ředitel byl brzy po zahájení prací na bakalářské práci odvolán. Nové vedení jsem už nekontaktoval. Ale myslím, že programy, které jsou součástí této práce by vyhověly původním požadavkům.

Programy se dají použít i mimo oblast zdravotnictví. Změnou konfigurační databáze lze vyhledávat entity i z jiných oblastí. Program **Anonymizer** je úplně obecný a nepředpokládá předem žádný charakter textu. Pracuje i s formátovanými textovými soubory, např. xml.

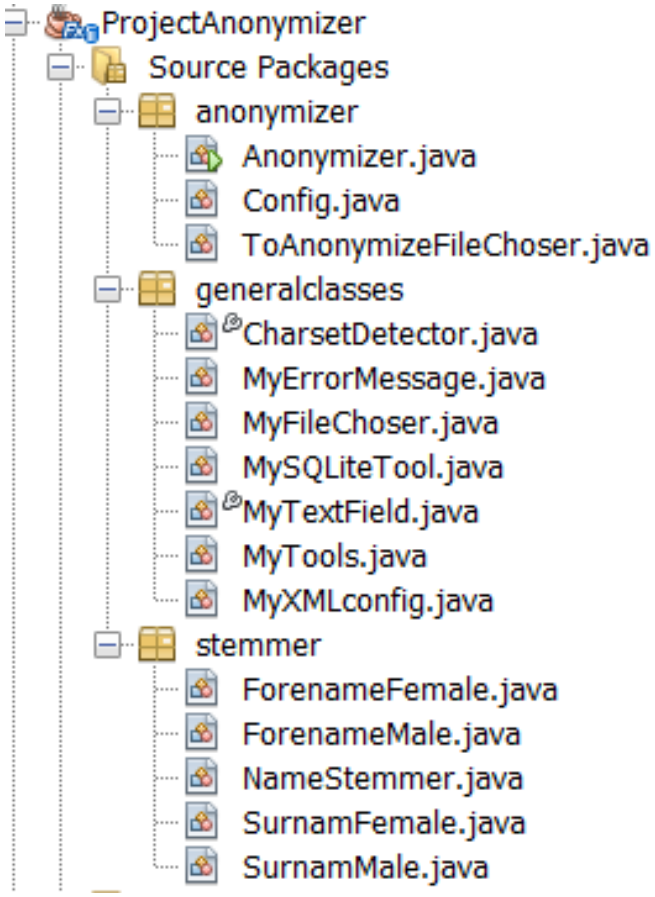
Na obou programech je možný další vývoj, u **Anonymizeru** by další zlepšení určitě spočívalo ve skloňování anonymních náhrad podle původního textu. U programu **ReportsInfoMiner** je možné další zlepšení jádra programu - třídy **RimEngine** - využít sofistikovanější metody vyhledávání entit a například rozlišit, kdy je entita ve větě v negativním smyslu - např. "Bolesti ustoupily". Výhoda koncepce programu je, že lze pracovat na zlepšení **RimEngine** aniž by se zasahovalo do zbytku programu.

Oblastí, která souvisí s vytěžováním informací z lékařských zpráv je také datamining, tedy práce nad již vytvořenou databází pomocí sql dotazů a zjišťování různých statistických údajů. Tímto se tato práce vůbec nezaobírá, ale je to určitě další zajímavé téma k rozšíření programu.

A Programová dokumentace

Oba programy jsou psány v jazyku Javafx, proto je třeba mít nainstalovánu Javu verze 8 a novější.

A.1 Program Anonymizer



Obrázek A.1: Struktura programu Anonymizer

Struktura programu je na Obr. A.1. Skladá se ze tří balíků:

anonymizer	- hlavní třídy Anonymizeru
generalclasses	- pomocné utility
stemmer	- práce se stemmizační databází

Zdrojový kód hlavní metody třídy Anonymizer pro anonymizaci textu:

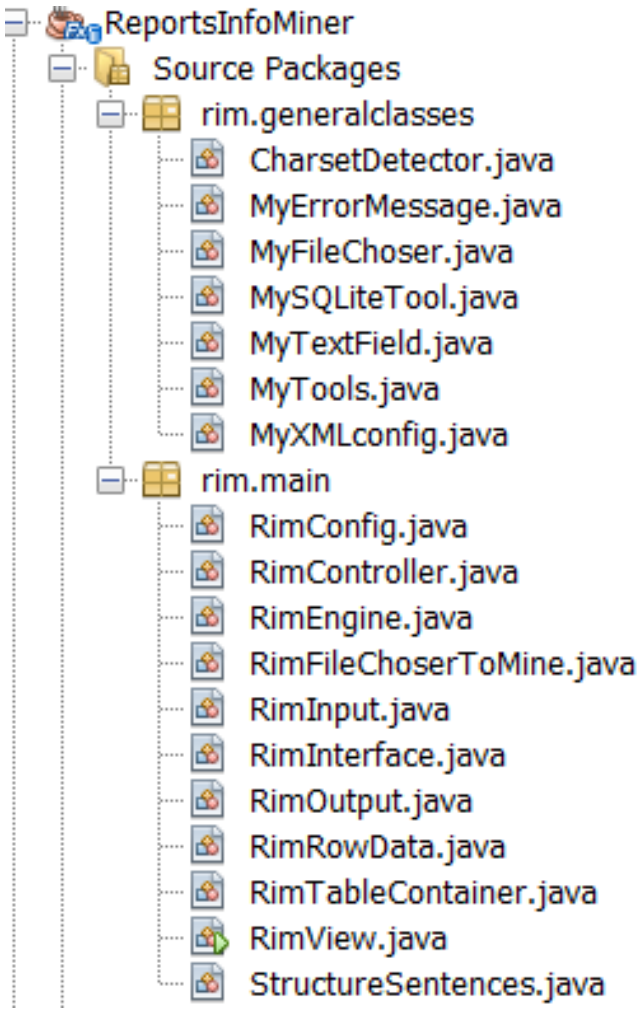
```
//#############################################################################
/**
 * hlavni metoda pro anonymizaci
 * 1. otevre soubor pro anonymizaci
 * 2. rozlozi obsah na jednotlive vety
 * 3. rozlozi vety na jednotliva slova
 * 4. ve slovech najde jmena a prijmeni
 * 5. zjisti, zda jde skutecne o jmena a prijmnei (pomoci
    slovniku korekce)
 * 6. nahradi jmena a prijmeni anonymnimi nahradami
 * 7. vyhleda dalsi kandidaty na nahrazeni (podle upper case)
 * 8. vlozi do slovniku skutecne jmeno (ve vsech vyskytujicich se
    tvarech) a jeho nahradu
 * @throws Exception
 */
void anonymize() throws Exception{

  // nacteni cest k souborum
  updateDataFromView();

  //aktualizuje seznam kandidatu pro nahrazeni - prida ty
     odsouhlasene v minulem behu
  updateCandidates();

  //nacte seznam jmen, ktera se oznacila v minulem behu jako
     nejmena
  getTableWrong();

  //zalozime nove slovniky, kde bude seznam nahrad - vytvareji se
     pri kazdem behu znovu
  m_DictionaryFornameMale = new HashMap<>();
  m_DictionaryFornameFemale = new HashMap<>();
  m_DictionarySurnameMale = new HashMap<>();
  m_DictionarySurnameFemale = new HashMap<>();

  try (
```

```
// soubor pro ulozeni vysledku
BufferedWriter bw = m_Config.getStoreWriter()) {
if (bw == null){
  // vystupni soubor neni v poradku
  return;
}

// rozklad vstupniho souboru na vety
String[] lines =
    MyTools.getLines(m_Config.m_cFileToAnonymize,
    m_Config.m_Charset);

// objekty pro nahrazeni jmen a prijmeni ve vete
SentenceObject soSurnamesMale = new
    SentenceObject(NameType.SURNAME_MALE,
    m_DictionarySurnameMale);
SentenceObject soSurnamesFemale = new
    SentenceObject(NameType.SURNAME_FEMALE,
    m_DictionarySurnameFemale);
SentenceObject soFornamesMale = new
    SentenceObject(NameType.FORENAME_MALE,
    m_DictionaryFornameMale);
SentenceObject soFornamesFemale = new
    SentenceObject(NameType.FORENAME_FEMALE,
    m_DictionaryFornameFemale);

/* cyklus pres vsechny vety */
for (String line : lines){
  // rozklad na slova
  String tokens[] = MyTools.tokenizeSentence(line);
  // vyndame pocatecni a koncove znaky, ktere nejsou
     pravdepodobne soucasti jmen
  tokens = washTokens(tokens);
  // nalezeni kandidatu na jmena a prijmeni
  String[] forenamesFemale = m_Stemmer.digNames(tokens,
     NameType.FORENAME_FEMALE);
  tokens = removeDuplicates(tokens, forenamesFemale);
  String[] forenamesMale = m_Stemmer.digNames(tokens,
     NameType.FORENAME_MALE);
  tokens = removeDuplicates(tokens, forenamesMale);
  String[] surnamesMale = m_Stemmer.digNames(tokens,
     NameType.SURNAME_MALE);
  tokens = removeDuplicates(tokens, surnamesMale);
```

```
String[] surnamesFemale = m_Stemmer.digNames(tokens,
    NameType.SURNAME_FEMALE);
tokens = removeDuplicates(tokens, surnamesFemale);
// v tokens uz zbyly jen "nonamy" - ulozime je pro pripadne
    odsouhlaseni uzivatelem
addFirstCapitalNonames(tokens);
//vyhodime jmena, ktera uz jsou zahrnuta v jinem seznamu
    jmen
forenamesMale = removeDuplicates(forenamesMale,
    forenamesFemale);
surnamesMale = removeDuplicates(surnamesMale,
    forenamesMale);
surnamesMale = removeDuplicates(surnamesMale,
    forenamesFemale);
surnamesFemale = removeDuplicates(surnamesFemale,
    forenamesMale);
surnamesFemale = removeDuplicates(surnamesFemale,
    forenamesFemale);
surnamesFemale = removeDuplicates(surnamesFemale,
    surnamesMale);

// pridame slova, ktera byla drive oznacena jako jmena
List<String[]> ret = addFromCandidates(tokens,
    surnamesMale, surnamesFemale);
surnamesMale = ret.get(0);
surnamesFemale = ret.get(1);

// kontrola spravnosti nalezenych entit podle slovniku
    korekce - vyhodime "nejmena"
forenamesFemale =
    correctListToReplaceFromCorrection(forenamesFemale,
    m_CorrectionForenames);
forenamesMale =
    correctListToReplaceFromCorrection(forenamesMale,
    m_CorrectionForenames);
surnamesFemale =
    correctListToReplaceFromCorrection(surnamesFemale,
    m_CorrectionSurnames);
surnamesMale =
    correctListToReplaceFromCorrection(surnamesMale,
    m_CorrectionSurnames);

// nahrazeni pravych jmen anonymy a pridani do slovniku
```

```
        line = soFornamesMale.replace(line, forenamesMale);
        line = soFornamesFemale.replace(line, forenamesFemale);
        line = soSurnamesMale.replace(line, surnamesMale);
        line = soSurnamesFemale.replace(line, surnamesFemale);
        //zapis zmenene vety do vystupniho souboru
        bw.write(line);
        bw.newLine();
      }
    }
    setTables();

    //m_Scene.getRoot().setCursor(Cursor.DEFAULT);
  }
}
```

A.2 Program ReportsInfoMiner



Obrázek A.2: Struktura programu ReportsInfoMiner

Struktura programu je na Obr. A.2. Skladá se ze tří balíků:

rim.main	- hlavní třídy programu
rim.generalclasses	- pomocné utility

Zdrojový kód hlavní datové třídy **RimTableContainer**:

```
/**
 * trida pro predavani konfigurace miningu do enginu a soucasne i
    pro ulozeni vysledku miningu, slouzi k ulozeni 3D tabulky
    stringu do pameti
 * datovy objekt m_DataObject ma tuto strukturu:
 * HashMap
 *    key : String-nazev tabulky s entitami (napr. LEKY)
 *    value : EntitiesColumn - HashMap
 *             key : entita z tabulky
 *             value: ArrayList-vektor se vsemi gramatickymi tvary
    entity
 * Je to vlastne 3-d matice, kde jeden rozmer je konkretni tabulka s
 * entitami (napr. LEKY), druhy rozmer jsou seznamy entit (tedy
    seznam leku v tabulce)
 * a treti rozmer je seznam gramatickych tvaru pro jeden lek
 * @author A12B0064K
 */
public class RimTableContainer implements java.io.Serializable{
  LinkedHashMap<String, EntitiesColumn> m_DataObject = new
      LinkedHashMap<String, EntitiesColumn>();
  boolean m_bIsEmpty;
  RimTableContainer(){
    m_bIsEmpty = true;
  }

  boolean isEmpty(){
    return m_bIsEmpty;
  }

  /**
   * ukladani objektu na disk pomoci serializace
   * @param cFile
   * @throws Exception
   */
  void writeToFile(String cFile) throws Exception {
    try{
      FileOutputStream fos;
      fos = new FileOutputStream(cFile);
      ObjectOutputStream oos = new ObjectOutputStream(fos);
      oos.writeObject(m_DataObject);
```

```
    oos.close();
  }catch(Exception e){
    throw new RuntimeException(e);
  }
}

/**
 * nacteni objektu z disku pomoci serializace
 * @param cFile
 * @return
 * @throws Exception
 */
boolean readFromFile(String cFile) throws Exception {
  File fl = new File(cFile);
  if (!fl.exists() || !fl.canRead())
    return false;
  try{
    FileInputStream fis = new FileInputStream(cFile);
    ObjectInputStream ois = new ObjectInputStream(fis);
    m_DataObject = (LinkedHashMap<String,
        EntitiesColumn>)ois.readObject();
    ois.close();
    return true;
  }catch(Exception e){
    return false;
  }
}


/**
 * vytvori strukturu podle pt, vlozi vsechny klice
 * @param pt
 */
void cloneContainer(RimTableContainer pt) {
  if (pt == null){
    return;
  }
  m_DataObject.clear();
  for (Iterator<String> it = pt.getColumnIterator();
      it.hasNext();){
    String cColumn = it.next();
    putColumn(cColumn);
  }
```

```
}

Iterator<String> getColumnIterator(){
  return m_DataObject.keySet().iterator();
}

int getSize(){
  return m_DataObject.size();
}

/**
 * vlozi novy sloupec
 * @param cColumn
 */
void putColumn(String cColumn){
  m_DataObject.put(cColumn, new EntitiesColumn());
}

/**
 * vlozi do sloupce novou polozku
 * @param cColumn sloupec
 * @param cItem polozka
 */
void putItemToColumn(String cColumn, String cItem){
  if (getColumn(cColumn) == null)
    putColumn(cColumn);
  getColumn(cColumn).putBasicWord(cItem);
}

/**
 * vlozi k polozce novy gramaticky tvar
 * @param cColumn sloupec kde je polozka
 * @param cBasic polozka
 * @param cShapeItem novy gramaticky tvar
 */
void putShapeItemToColumn(String cColumn, String cBasic, String
    cShapeItem){
  if (getColumn(cColumn) == null)
    putColumn(cColumn);
  getColumn(cColumn).putWordInGrammarShape(cBasic, cShapeItem);
}

void removeColumn(String cColumn){
```

```
  m_DataObject.remove(cColumn);
}

EntitiesColumn getColumn(String cColumn){
  return (EntitiesColumn)m_DataObject.get(cColumn);
}

/**
 * vraci "souradnice" nalezeneho slova v datovem objektu String[3]
 *  String[0] - nazev sloupce, kde bylo slovo nalezeno
 *  String[1] - zakladni tvar nalezeneho slova
 *  String[2] - puvodni tvar slova ( = word)
 *
 * @param word hledane slovo
 * @return "souradnice" nalezeneho slova v datovem objektu
    String[3],
 *         null pokud neni nalezeno
 */
String[] getBaseShapes(String word){
  for (Iterator<String> it = this.getColumnIterator();
      it.hasNext();){
    String cColumn = it.next();
    RimTableContainer.EntitiesColumn oneColumn =
       this.getColumn(cColumn);
    String originWordFound = oneColumn.getBaseShape(word);
    if (originWordFound != null){
      String ret[] = {cColumn, originWordFound, word};
      return ret;
    }
  }
  return null;
}

  /**
   * trida, ktera reprezentuje jeden sloupec, jako klic ma
      zakladni tvar slova
   *  a jako hodnotu vektor
   *  gramatickych tvaru tohoto slova (lek - leku, lekem atd.)
   *
   */
  public class EntitiesColumn extends LinkedHashMap<String,
     ArrayList<String>>{
  EntitiesColumn(){};
```

```
/**
 * vlozi novy prvek do mapy, jako klic je nove slovo word,
 * jako hodnota je prazdne pole gramatickych tvaru
 * @param word vkladane slovo
 */
void putBasicWord(String word){
  put(word, new ArrayList<String>());
  m_bIsEmpty = false;
}

/**
 * prida novy gramaticky tvar shape zakladnimu tvaru word,
    pokud word
 * jeste v mape neexistuje, vytvori se
 * @param word zakladni tvar vkladaneho slova
 * @param shape vkladane slovo
 */
void putWordInGrammarShape(String word, String shape){
  if (get(word) == null)
    putBasicWord(word);
  m_bIsEmpty = false;
  get(word).add(shape);
}

/**
 * @return iterator pres klice
 */
Iterator<String> getKeyIterator(){
  return this.keySet().iterator();
}


/**
 * pokud je word nalezen v seznamu gramatickych tvaru, vrati
    zakladni tvar
 *  slova, jinak vraci null
 * @param word obecny tvar slova
 * @return nalezeny zakladni tvar slova, null pokud neni nalezen
 */
String getBaseShape(String word){
  for (Iterator<String> jt = this.getKeyIterator();
      jt.hasNext();){
```

```
        String originWord = jt.next();
        ArrayList<String> al = this.get(originWord);
        for (String cp : al){
          if (cp.equalsIgnoreCase(word))
            return originWord;
        }
      }
      return null;
    }

    /**
     * vytvori seznam vsech zakladnich tvaru slov oddelene carkou,
         napr.:
     * "mast, antibiotika, paralen"
     * @return seznam zakladnich tvaru slov oddelenych carkou
     */
    String getColumnDataString(){
      String ret = "";
      for (Iterator<String> jt = this.getKeyIterator();
          jt.hasNext();){
        String originWord = jt.next();
        if (originWord != null)
          ret += (ret.isEmpty()?"":", ") + originWord;
      }
      return ret;
    }

  }

}
```

A.3 Wrapper pro práci s SQLite databází

Zdrojový kód třídy pro práci s SQLite DB.

MySQLiteTool:

```
/**
 * wrapper pro knihovnu databaze sqlite
 *
 * @author A12B0064K
 */
public class MySQLiteTool {

  // aktualni spojeni s databzi
  Connection m_Connection;

  // aktualni tabulka databaze, se kterou se pracuje
  String m_cActiveTable;

  // resultset napriklad pro vysledek sql dotazu
  ResultSet m_ResultSet;

  // seznam jmen sloupcu aktualni tabulky
  List<String> m_ColumnsName;

  /**
   * vytvori aktualni spojeni na databazi v souboru cDBfile
   * zaroven ho dosadi do promenne m_Connection
   * @param cDBfile - soubor s SQlite databazi
   * @return aktualni spojeni
   * @throws Exception
   */
  public Connection createConnectionToSQLite(String cDBfile) throws
      Exception{
    Class.forName("org.sqlite.JDBC");
    m_Connection =
        DriverManager.getConnection("jdbc:sqlite:"+cDBfile);
    return m_Connection;
  }

  /**
```

```
 * vraci aktualni spojeni
 * @return
 */
public Connection getConnection(){
  return m_Connection;
}

/**
 * nastavuje aktualni tabulku v databazi
 * @param cTable
 */
public void setActiveTable(String cTable){
  m_cActiveTable = cTable;
}

/**
 * ze souboru s formatem csv vytvoru SQlite databazi
 * @param cCSVinput - vstupni csv soubor
 * @param cSQLiteOutput - vystupni sqlite soubor s databazi
 */
public static void createSQLiteFromCSV(String cCSVinput, String
    cSQLiteOutput) throws Exception{
  final String SEPARATOR = "#@#";

  String Charset = MyTools.getCharset(cCSVinput);
  if (Charset == null){
    throw new RuntimeException("File "+cCSVinput+" is
        unreadable.");
  }

  File out = new File(cSQLiteOutput);
  if (out.exists()){
    out.delete();
  }

  MySQLiteTool sq = new MySQLiteTool();
  sq.createConnectionToSQLite(cSQLiteOutput);
  sq.getConnection().setAutoCommit(false);

  String[] lines = MyTools.getLines(cCSVinput, Charset);
  if (lines == null || lines.length==0){
    throw new RuntimeException("File "+cCSVinput+" is empty.");
  }
```

```
// nejdriv vytvorime strukturu tabulky
String[] cHeader = lines[0].split(SEPARATOR);
int iPocetSloupcu = cHeader.length;
if (iPocetSloupcu == 0){
  throw new RuntimeException("File "+cCSVinput+" has no
      columns.");
}

String[] cDataType = new String[iPocetSloupcu];
for(int i=0; i<iPocetSloupcu; i++){
  cDataType[i] = "TEXT";
}
cDataType[0] = "INT";
String cTableName = MyTools.getFileName(cSQLiteOutput);
sq.createSQliteTable(cTableName, cHeader,cDataType);

// naplnime tabulku v db daty
String header = lines[0].replace(SEPARATOR, ",");
int iLine = 0;
for (int i=1; i<lines.length; i++){
  iLine++;
  String line = lines[i];
  Statement stmt = sq.getConnection().createStatement();
  //String[] splitLine = MyTools.separateString(line, ",",
      iPocetSloupcu);
  line = "'"+line.replace(SEPARATOR,"','")+"'";
  String sql = "INSERT INTO "+cTableName+" ("+header+") " +
               "VALUES ("+String.valueOf(iLine)+','+line+");";
  try{
    stmt.executeUpdate(sql);
  }
  catch(Exception e){
    System.out.print(sql+"\n");
  }
  stmt.close();
}
sq.getConnection().commit();


sq.getConnection().close();
sq.close();
```

```
}


/**
 * vytvori sqlite tabulku
 *
 * @param cName - nazev tabulky
 * @param cStructure - nazvy jednotlivych sloupcu tabulky
 * @param cDataType - typy promennych ve sloupcich
 * @throws Exception
 */
public void createSQliteTable(String cName, String[] cStructure,
    String[] cDataType) throws Exception {
  if (m_Connection == null || cStructure.length == 0) return;
  Statement stmt = m_Connection.createStatement();
  if (cName == null)
    cName = m_cActiveTable;

  String sql = "CREATE TABLE " + cName +
              " (____ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY,";
  for (int i=0; i<cStructure.length-1; i++){
    sql += cStructure[i] + " "+cDataType[i]+",";
  }
  sql += cStructure[cStructure.length-1] + "
      "+cDataType[cStructure.length-1]+")";
  try{
    stmt.executeUpdate(sql);
  }catch (Exception e){
    throw new RuntimeException(e);
  }
  stmt.close();
}

/**
 * vraci prvni record odpovidajici SQL prikazu
 * @param sql SQL prikaz
 * @return resultset, pokud zadny nenalezne, vraci null
 * @throws Exception
 */
public ResultSet getFirstRecord(String sql) throws Exception {
  Statement stmt = getConnection().createStatement();
  ResultSet rs = stmt.executeQuery(sql);
  if (rs.next()){
```

```
    return rs;
  }
  return null;
}

/**
 * vraci nasledujici record odpovidajici SQL prikazu
 * @param rs predchozi resultset
 * @return resultset, pokud zadny nenalezne, vraci null
 * @throws Exception
 */
public ResultSet getNextRecord(ResultSet rs) throws Exception {
  if (rs.next()){
    return rs;
  }
  return null;
}

/**
 * uzavre spojeni s databazi
 * @throws Exception
 */
void close() throws Exception{
  if (m_Connection != null)
    m_Connection.close();
}

/**
 * vraci seznam nazvu sloupcu tabulky
 * @param cTable nazev tabulky, pro kterou chceme seznam sloupcu,
 * pokud je null, vezme se aktualni tabulka
 * @return seznam nazvu sloupcu tabulky
 * @throws Exception
 */
List getColumnsName(String cTable)throws Exception {
  if (cTable == null)
    cTable = m_cActiveTable;
  DatabaseMetaData meta = getConnection().getMetaData();
  ResultSet res = meta.getColumns(null, null, cTable, null);
  List ret = null;
  while (res.next()) {
    if (ret == null)
      ret = new ArrayList<String>();
```

```
    ret.add(res.getString("COLUMN_NAME"));
  }
  return ret;
}

/**
 * vraci prvni radek tabulky
 * je ulozen v HashMap, kde key je nazev sloupce a value je
    hodnota ve sloupci
 * @return prvni radek tabulky
 * @throws Exception
 */
public LinkedHashMap<String, String> getFirstRow()throws
    Exception{
  String sql = "SELECT * FROM '"+ m_cActiveTable+ "'";
  m_ResultSet = getFirstRecord(sql);
  m_ColumnsName = getColumnsName(null);
  return getRow();
}

/**
 * vraci dalsi radek tabulky
 * je ulozen v HashMap, kde key je nazev sloupce a value je
    hodnota ve sloupci
 * @return dasli radek tabulky
 * @throws Exception
 */
public LinkedHashMap<String, String> getNextRow()throws Exception{
  if (m_ResultSet == null)
    return null;
  m_ResultSet = getNextRecord(m_ResultSet);
  return getRow();
}

private LinkedHashMap<String, String> getRow()throws Exception{
  LinkedHashMap<String, String> RowData = new
      LinkedHashMap<String, String>();
  if (m_ResultSet != null){
    for (String cColumn : m_ColumnsName){
      RowData.put(cColumn, m_ResultSet.getString(cColumn));
    }
    return RowData;
  }
```

```
    return null;
  }

  /**
   * vraci radek row, ktery ma polozky oddelene carkami
   * @param row
   * @return
   */
  private String[] getValues(Map<String, String> row){
    String[] ret = {"","",};
    for (Iterator<String> it = row.keySet().iterator();
        it.hasNext();){
      String cKey = it.next();
      ret[0] += (ret[0].isEmpty()?"":",") + cKey;
      ret[1] += (ret[1].isEmpty()?"'":"','") + row.get(cKey);
    }
    if (!ret[1].isEmpty())
      ret[1] += "'";
    return ret;
  }

  /**
   * vlozi do aktualni tabulky radek s daty
   * @param Row data radku
   * @param iRow cislo radku - pouziva se jako primary key pro
       sloupec ____ID
   * @throws Exception
   */
  public void insertRow(Map<String, String> Row, int iRow) throws
      Exception {
    String[] vl = getValues(Row);
    String sql = "INSERT INTO "+m_cActiveTable+" (____ID,"+vl[0]+")
        " +
                "VALUES ("+String.valueOf(iRow)+","+vl[1]+");";
    Statement stmt = m_Connection.createStatement();
    try{
      stmt.executeUpdate(sql);
    }catch(Exception e){
      throw new RuntimeException(e);
    }}

}
```

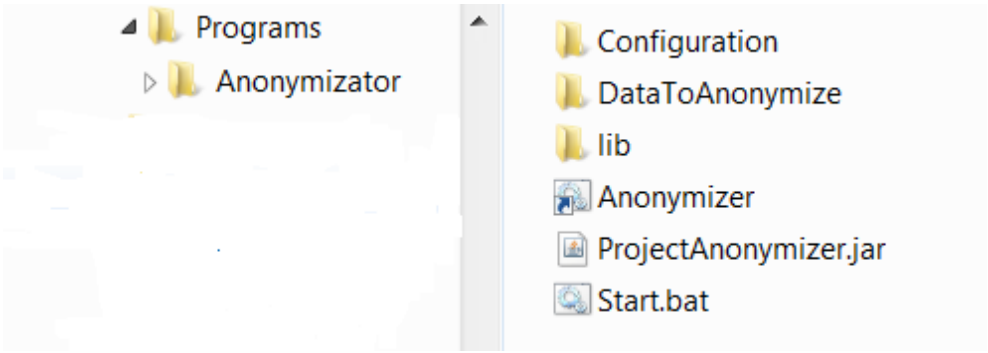
B Uživatelská dokumentace

B.1 Softwarové požadavky

Oba programy vyžadují nainstalovanou Javu verze 8 a novější. Program je odzkoušen na Windows 7 ale měl by fungovat i na jiných platformách.

B.2 Program Anonymizer

Adresářová struktura programu je na Obr. B.1:

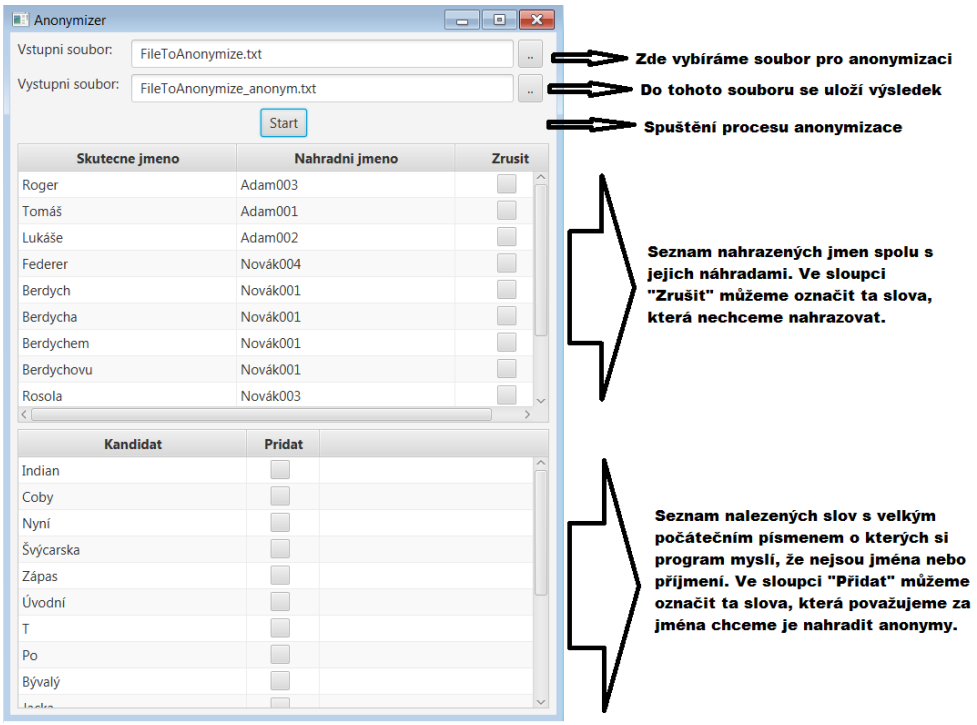


Obrázek B.1: Adresář programu Anonymizer

Popis adresářů:

Configuration	- konfigurační soubory programu
DataToAnonymize	- testovací data v souboru FileToAnonymize.txt
lib	- knihovny projektu

Program se spouští dávkovým souborem start.bat
Ovládací dialog je na Obr. B.2. Ovládání programu je jednoduché a intuitivní. Důležité je, že pokud nějaké slovo označíme za ”nejméno”, nebude se už příště nahrazovat. Program se tak vlastně ”učí”.



Obrázek B.2: Program Anonymizer

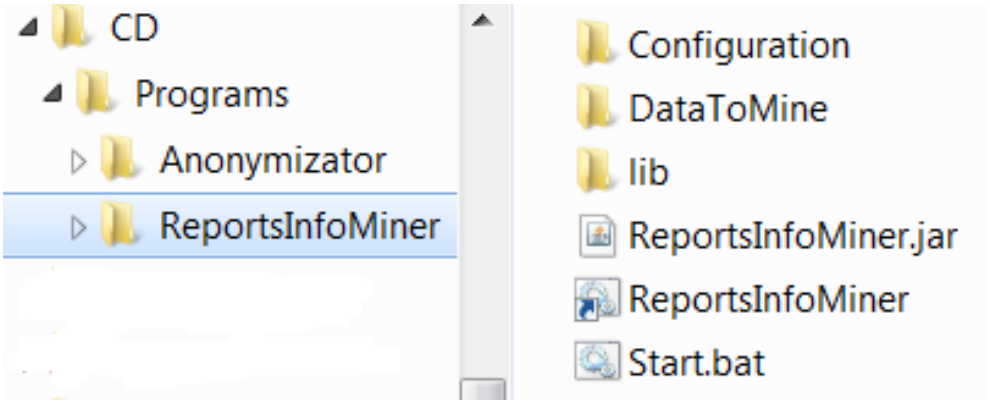
B.3 Program ReportsInfoMiner

Adresářová struktura programu je na Obr. B.3:

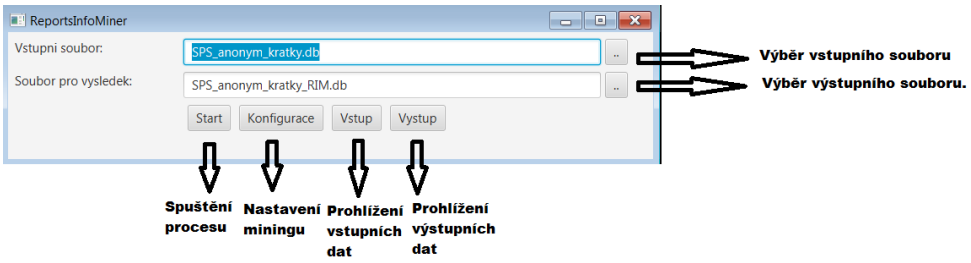
Popis adresářů:

Configuration	- konfigurační soubory programu
DataToMine	- testovací data
lib	- knihovny projektu

Program se spouští dávkovým souborem start.bat
Ovládací dialog je na Obr. B.4.



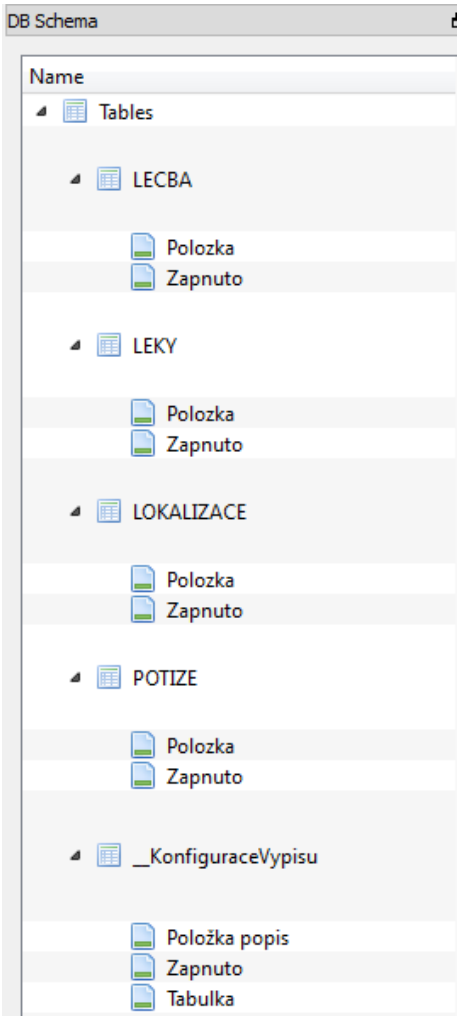
Obrázek B.3: Adresář programu ReportsInfoMiner



Obrázek B.4: Program ReportsInfoMiner

Důležitá poznámka: Pokud zpracováváme text z sqlite databáze, musí se tabulka, ze které v databázi převádíme, jmenovat stejně jako soubor databáze (bez extenze). Tedy např. v souboru inputdata.db musí být tabulka inputdata, tu bude pak program zpracovávat.

Podrobně popíšeme akci **Konfigurace**. Nastavují se v ní parametry výběru klíčových slov v dokumentu a jejich zařazení do databáze.
Nejdříve si popíšeme strukturu konfigurační databáze health_care.db - Obr. B.5.



Obrázek B.5: Struktura konfigurační databáze

Databáze má 4 tabulky (**LECBA, LEKY, LOKALIZACE, POTIZE**) pro hledané entity a jednu tabulku (**_KonfiguraceVypisu**) pro seznam tabulek entit. Výpis tabulky **LECBA** je na Obr. B.6

Table: LECBA

	Polozka	Zapnuto
	Filter	Filter
1	omýt	1
2	osušit	1
3	namazat	1
4	promazat	1
5	promazávat	1
6	nahlásit	1
7	polohovat	1
8	vložit	1
9	zavést	1
10	odebírat	1
11	odebírání	1
12	mazat	1
13	promazávání	1

Obrázek B.6: Výpis tabulky LECBA

V prvním sloupci je položka, kterou v textu hledáme. Musí být zadána v základním tvaru, tzn. sloveso v infinitivu, podstatné jméno v 1. pádu jednotného čísla atd. Ve druhém sloupci je příznak, zda se daná položka ve vyhledávání bude využívat. Lze tedy například dočasně některé výrazy z vyhledávání vyjmout, aniž bychom je museli mazat z tabulky. Položka Filter nad sloupci je součást SQLite Browseru, slouží k filtraci položek sloupce při prohlížení tabulky.

Výpis tabulky **__KonfiguraceVypisu** je na Obr. B.7.

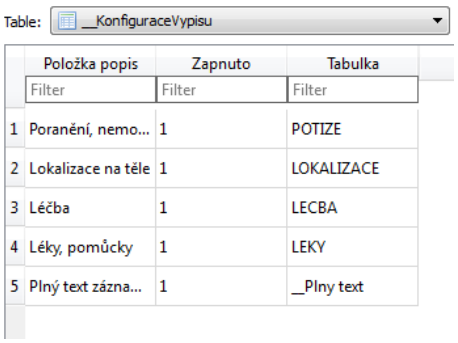


Table: __KonfiguraceVypisu

	Položka popis	Zapnuto	Tabulka
	Filter	Filter	Filter
1	Poranění, nemo...	1	POTIZE
2	Lokalizace na těle	1	LOKALIZACE
3	Léčba	1	LECBA
4	Léky, pomůcky	1	LEKY
5	Plný text zázna...	1	__Plny text

Obrázek B.7: Výpis tabulky __KonfiguraceVypisu

V třetím sloupci je název tabulky s entitami, pokud přidáme novou tabulku do databáze, musíme sem také přidat její záznam. Jinak se prohledávání jejich položek v textu neprovede. Ve druhém sloupci je vypínač, obdobně jako u jednotlivých položek, můžeme zde dočasně vypnout celou tabulku hledaných entit. První sloupec je pouze popis, bez dalšího významu.
Speciální význam má poslední položka **__Plny text**. Jejím vypnutím a zapnutím určíme, jestli ve výsledku bude také sloupec s plným originálním textem, ze kterého se extrahovaly entity.

Přidáním nebo ubráním tabulek v konfigurační databázi a přidáním nebo ubráním jejich položek můžeme ovlivňovat, co bude program v textu vyhledávat a jak bude vyhledané entity umísťovat ve výsledné databázi.

C Obsah přiloženého CD

Na přiloženém CD jsou následující data:

Programs	- oba programy spolu s knihovnami
Results	- anonymizovaný soubor Zpravy_anonym.db a jeho strukturalizace Zpravy_anonym_RIM.db
Sources	- zdrojové soubory
Text	- Text práce v pdf a latex

Literatura a odkazy

[1] Knihovna OpenNLP, *https://opennlp.apache.org/*

[2] Seznam všech jmen v ČR s doplněným oslovením (5. pádem - vokativem) *http://www.validace.cz*

[3] DB Browser for SQLite *http://http://sqlitebrowser.org/*

[4] SQLite JDBC *http://mvnrepository.com/artifact/org.xerial/sqlite-jdbc/*

[5] Charu C. Aggarwal, ChengXiang Zhai *Mining Text Data*