VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

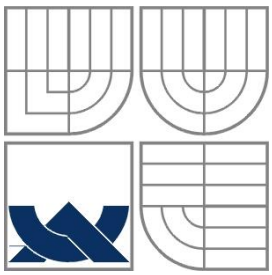
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

FORMÁT XML PRO ZNAČKOVÁNÍ SLOVNÍKŮ

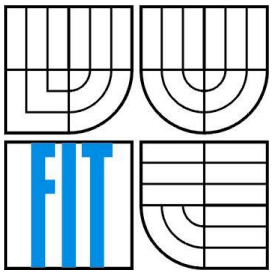
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE LUKÁŠ VAVREČAN
AUTHOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

FORMÁT XML PRO ZNAČKOVÁNÍ SLOVNÍKŮ

XML DICTIONARY TAGGING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

LUKÁŠ VAVREČAN

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. RNDr. Pavel Smrž, Ph.D.

BRNO 2015

Abstrakt

Bakalářská práce se zabývá zpracováním elektronických slovníků ve formátě LMF, jejich analýzou a převodem do jednotné podoby. Cílem je vytvořit systém, který díky hromadným změnám ulehčí strojové zpracování slovníků. Přiblížíme si potřebné technologie, zejména Python3, formát XML a LMF.

Abstract

This thesis deals with the processing of electronic dictionaries in the LMF format, their analysis and conversion into an unified form. The aim is to create a system, that can make the machine processing of dictionaries easier, thanks to the mass changes. We will introduce the necessary technology, especially Python3, format XML and LMF.

Klíčová slova

XML, slovník, LMF, Python3, regulárne výrazy, ElementTree

Keywords

XML, dictionary, LMF, Python3, regular expressions, ElementTree

Citace

Vavrečan Lukáš: Formár XML pro značkovaní slovníků, bakalářská práce, Brno, FIT VUT v Brně, 2015

Formát XML pro značkovaní slovníků

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doc. RNDr. Pavla Smrža, Ph.D
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lukáš Vavrečan
13.5.2015

Poděkování

Rád by som poďakoval vedúcemu bakalárskej práce doc. RNDr. Pavlovi Smržovi, Ph.D za ochotu a odborné rady pri vypracovaní práce. Ďalej by som rád poďakoval kolegovom, ktorí za mňa brali večernú pohotovosť, aby som mohol po večeroch vypracovať túto prácu.

Obsah

1 Úvod

V súčasnej dobe existuje množstvo prevodníkov, ktoré dokážu previesť prirodzený jazyk do strojovej podoby. Problém je však vo forme, do akej to prevedú. Rôznorodosť týchto výstupov má za následok skomplikovanie ďalšej práce, ktorú by sme nad týmto strojovým prekladom mohli vykonávať.

Cieľom mojej práce je vytvoriť prostriedky, ktoré dokážu aspoň z časti zjednotiť formu zápisu, a tak uľahčiť pochopeniu týchto dát. Práca je rozdelená do niekoľkých celkov. Úvodná kapitola je zameraná na rozbor XML. Okrem teoretickej časti, kde sa dozvieme vlastnosti XML, jeho výhody a syntax, porovnáme si aj technológie, ktoré boli najlepšími kandidátmi pre vytvorenie systému. Ďalej je popis slovníkov a LMF.

Nasledujúca kapitola približuje problémy pri prevode slovníkov do jednotnej podoby , ich analýzu a návrh riešenia.

Ďalšia kapitola popisuje implementáciu jednotlivých systémov a priblíži výhody programovacieho jazyka Python a dôvody, prečo som si ho vybral.

Na záver sa spomenie aj testovanie, keďže to nesmie pri vývoji chýbať.

2 XML

Formát XML (eXtensible Markup Language), tj. rozšíriteľný značkovací jazyk, bolo definované konzorciom W3C ako formát pre prenos obecných dokumentov a dát. Návrh vychádza zo štandardu SGML (Standard Generalized Markup Language) rovnako ako aj formát HTML (Hyper – Text Markup Language). Na rozdiel od HTML, kde je daná pevná sada značiek (tagov), XML môže byť definované rôznorodo. [1]

2.1 Formát XML

Pri návrhu XML sa autori z konzorcia W3C riadili nasledujúcimi princípmi: [1]

- musí byť použiteľný v rámci internetu
- mal by podporovať širokú škálu aplikácii
- musí byť kompatibilný s formátom SGML
- musí byť ľahké vytvárať programy, ktoré manipulujú s dokumentami XML
- XML dokumenty by mali byť čitateľné a pochopiteľné aj pre človeka

2.2 Vlastnosti XML

2.2.1 Štandardný formát pre výmenu informácií

Pri zdieľaní elektronických dokumentov môžeme naraziť na problém spôsobený potrebou vlastniť konkrétny software. Z tohto dôvodu, bolo potrebné vytvoriť jednoduchý otvorený formát, ktorý nie je úzko spätý s určitou platformou či technológiou, ale je založený na jednoduchom texte a je spracovateľný ľubovoľným editorom. Práve preto vznikol XML, ktorého špecifikácia je voľne dostupná všetkým. [2]

2.2.2 Medzinárodná podpora

XML už od počiatku počítal s viacerými jazykmi, a preto implicitne používa znakovú sadu ISO 10646 (Unicode). Súčasťou je prístupné aj iné kódovanie, ale musí byť v dokumente explicitne určené. [2]

2.2.3 Vysoký informačný obsah

Pomocou XML tagov vyznačujeme v dokumente jednotlivé časti textu, vďaka čomu sú tak XML dokumenty informačne bohatšie. Môžeme tak vyhľadávať dáta na základe ich významu.

Na druhú stranu sa tak znižuje prehľadnosť pre človeka a XML súbory sú veľké. XML taktiež neponúka možnosť definovať vzhľad, musíme použiť externý jazyk. [2]

2.2.4 Jednoduchá konverzia do iných formátov

Pri používaní XML dokumentov potrebujeme daný dokument zobraziť, a keďže neposkytuje žiadne prostriedky pre definíciu vzhľadu, musíme použiť niektorý štýlový jazyk. Súboru pravidiel, ktoré definujú spôsob prevedenia dokumentu, sa vraví štýl.

Jeden vytvorený štýl môžeme aplikovať na mnoho dokumentov, rovnako ako môžeme aplikovať jeden dokument na viacero štýlov. Výsledok je napr. postscriptový súbor, HTML kód.
Medzi najznámejšie štýlové jazyky patria kaskádové štýly (CSS), XSL (eXtensible Stylesheet Language). [2]

2.2.5 Automatická kontrola štruktúry dokumentu

Keďže XML neobsahuje preddefinované tagy, treba si definovať vlastné. Tieto značky môžeme definovať v súboru DTD (Document Type Definition) a následne môžeme automaticky kontrolovať, či XML odpovedá tejto definícii. DTD neumožňuje kontrolu typov dát (údaje o času, čísla). Program, ktorý tieto kontroly vykonáva, sa nazýva parser (viac sa o nich dozvieme v podkapitole 2.4). [2]

2.3 Syntax XML

Na rozdiel od HTML, XML je case-sensitive (rozlišuje veľké a malé písmená),takže tag <Meno> nie je to isté ako <meno>.
Prvým riadkom býva XML deklarácia:

Príklad:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

Pravidlá XML deklarácie sú nasledovné:

- Musí byť umiestnená na prvom mieste na prvom riadku v XML dokumente
- Musí obsahovať atribút „version“
- Ak sú deklarované aj ostatné atribúty, musia byť v poradí uvedenom vyššie
- XML deklarácia musí byť malými písmenami (okrem kódovania)

Všimnime si, že XML deklarácia nemá ukončovací tag

Dokument sa skladá z elementov, pričom element môže obsahovať ďalšie elementy. Element začína a končí tagom, medzi ktorými sa nachádzajú dáta. Element môže obsahovať atribúty, ktoré sa uvádzajú v začiatočnom tagu elementu.

Príklad:
<osoba meno="Lukáš" priezvisko="Vavrečan"> študuje na FIT VUT v Brne <osoba>

Aby bol dokument považovaný na správne štruktúrovaný, musí mať nasledujúce vlastnosti: [2]

- Musí mať práve jeden koreňový element (root)
- Neprázdne elementy musia byť ohraničené počiatočnou a ukončovacou značkou:
  <meno>Lukáš</meno>
  Alebo zapísané skrátenou formou:
  <meno/>
- Prázdny element nemusí mať ukončovací tag:

- Všetky hodnoty atribútov musia byť uzatvorené v úvodzovkách:
  <osoba meno="Lukáš">
  <osoba meno='Lukáš'>
- Elementy môžu byť vnorené, ale nemôžu sa prekrývať:
  Nesprávne:
  <osoba><adresa>Brno</osoba></adresa>
  Správne:
  <osoba><adresa>Brno</adresa></osoba>.
- Komentár:
  <!-- komentár -->
- Niektoré znaky nemôžeme jednoducho zapísať, pretože by kolidovali s XML syntaxou. Preto ich musíme nahradzovať tzv. escape sekvenciami, ktoré sú uvedené v tabuľke:

„	&quot;	uvodzovky
'	&apos;	apostrof
<	&lt;	nenší ako
>	&gt;	väčší ako
&	&amp;	ampersand

Tabuľka 2.1: tabuľka escape sekvencií

Jednoduchá ukážka XML dokumentu:

```
<?xml version="1.0" encoding="UTF-8"?>
<list>
      <odosielatel>Lukas</odosielatel >
      <prijimatel>Tomas</prijimatel >
      <hlavicka>Pripomienka</hlavicka>
      <telo>Nezabudni na vikend</telo>
</list>
```

Ukážka 2.1: XML

2.4 XML parsery

Základ tejto práce je manipulovanie s XML dokumentami. Najčastejšou operáciou je načítavanie, k tomuto účelu slúži tzv. parser.

XML parser je knižnica, ktorá analyzuje súbor, aby identifikoval jednotlivé komponenty. Zároveň kontroluje syntax, či je správne naformátovaný dokument a hlási prípadné chyby.

Aby mohli aplikácie s XML dokumentom jednoducho pracovať, využívajú aplikačné rozhrania (API). Najčastejšie používané prístupy sú SAX, ktoré sú založené na udalostiach, a DOM, ktorý je založený na stromoch a objektoch.

2.4.1 SAX

SAX (Simple API for XML) je rozhranie založené na udalostiach, kedy sa pri sekvenčnom čítaní spracovávaného dokumentu narazí napr. na:

- Počiatok dokumentu
- Počiatočná značka elementu
- Koncovú značku elementu
- Znakové dáta
- Inštrukcie pre spracovanie

SAX informuje o udalosti, keď vidí uzol pre značku, atribút, text alebo externú entitu. Ak narazí napr. na element, vracia tento element, zoznam jeho atribútov a obsah. SAX samo o sebe nie je XML procesorom. Je potrebné pripojiť vlastné funkcie (handlere pre zachádzanie s udalosťami). Každá udalosť vyvolá korešpondujúcu funkciu (metódu), ktorú píše programátor. Tento spôsob spracovania je tzv. prúdovo orientovaný. Spracovávanie dáva výsledky, aj napriek tomu, že nemá k dispozícii celý XML dokument. Základnou myšlienkou je spracovať element a potom ho „zabudnúť“.
Výhody a nevýhody sú popísané v tabuľke 2.2. [1]

VÝHODY	NEVÝHODY
jednoduché pre použitie	moc toho nedokáže
veľmi rýchle	nie je užitočný, ak treba dokument meniť v pamäti
nie je treba veľká pamäť	obmedzená práca s dokumentom, napr. nedá sa vracať „dozadu“
môže zachrániť dáta z nesprávneho dokumentu	
možnosť práce s prúdom XML dát	

Tabuľka 2.2: Analýza spracovania XML pomocou SAX

2.4.2 DOM

DOM (Document Object Model) je objektovo orientované rozhranie nezávislé na jazyku a platforme. Pôvodne vznikol kvôli potrebe prístupu k niektorým typom elementov na HTML stránke prostredníctvom skriptovacích jazykov. Dokument môže byť spracovaný a výsledok môže byť zahrnutý späť do stránky.

Pri spracovaní XML dát generuje procesor vo vnútornej pamäti strom, ktorý odpovedá spracovávanému dokumentu. Rozhranie DOM definuje metódy pre prístup a (na rozdiel od SAX) taktiež modifikáciu stromu.

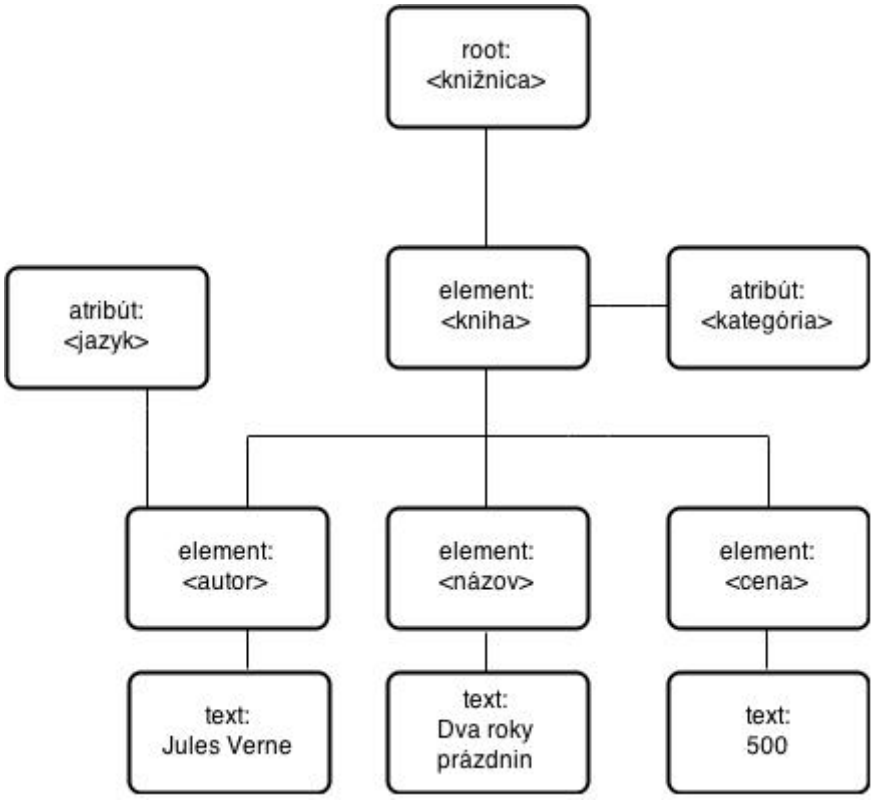
DOM definuje jazykovo nezávislé rozhranie, ale aj väzby na rôzne programovacie jazyky ako C++, JavaScript. Štandard zahrňuje DOM Level 1, 2, 3, ktoré pre spracovanie XML dát predstavujú de jure štandard (oficiálne podporené). [1]

VÝHODY	NEVÝHODY
veľmi užitočné pre dynamickú modifikáciu a prístup k stromu	môže byť pomalé a spotrebovať veľa pamäti
užitočné pre dotazovanie nad XML dátami	DOM je pre programovanie trochu ťažkopádne
rozhranie je rovnaké pre rôzne prog. jazyky	

Tabuľka 2.3: Analýza spracovania XML pomocou DOM

Príklad prevedenia XML na DOM strom:

```
<knižnica>
      <kniha kategória="román">
            <titul jazyk="cze">Dva roky prázdnin</title>
            <autor>Jules Verne</autor>
            <cena>500</cena >
      </kniha>
</knižnica>
```



Obrázok 2.1.: XML prevedené na DOM strom

2.4.3 ElementTree

XML je hierarchický dátový formát a prirodzenou cestou , ako ho reprezentovať, je pomocou stromu. Preto som od začiatku inklinoval k DOM. Avšak, problém bol v jeho rýchlosti a spotrebe pamäte. Našťastie, od verzie Python 2.5 pribudlo nové API, ktoré je oproti DOM rýchlejšie a spotrebuje omnoho menej pamäte.

ElementTree je ľahké používať, lebo reprezentuje XML strom ako štruktúru zoznamov a atribúty sú reprezentované ako slovníky – asociatívne pole, ktoré je implementované pomocou transformačných tabuliek. Každý nod stromu je Element, čo je objekt kontajnerov pre zachádzanie hierarchických štruktúrovaných dát v pamäti. [3]

3 Slovník

Slovník je abecedne zoradený zoznam slovnej zásoby vysvetľujúci slova z rôznych hľadísk. Zostavovaním slovníkov sa zaoberá lexikografia.

Heslové slovo, pod ktorým je text uvedený v slovníku, sa volá lemma. Pri vytváraní lemma existujú určité konvencie, ako napr. slovesá sa uvádzajú v neurčitku, podstatné mená v nominatíve jednotného čísla, a pod.

Slovníky boli pôvodne v knižnej podobe a stále sa väčšina slovníkov v tejto podobe nachádza. S pribúdajúcim časom sa však slovníky čoraz častejšie objavujú v digitálnej podobe (najmä na internete). [4]

3.1 Typy slovníkov

Existuje viacero delení slovníkov:

1) Podľa rozsahu
   a) Malé, vreckové – do 10 000 hesiel
   b) Stredné – do 60 000 hesiel
   c) Veľké – nad 60 000 hesiel
2) Podľa typu
   a) Výkladové (jednojazyčné) – sú napísané v jednom jazyku
      i) Slovník súčasného jazyka (významové, pravopisné, frazeologické, atď.)
      ii) Slovníky jednotlivých historických období
      iii) Etymologické
      iv) Slovníky popisujúce slovnú zásobu profesných skupín (filozofický, lekársky, biblický, atď.)
   b) Prekladové (viacjazyčné) – slúži pre preklad z jedného jazyka do druhého. K slovám jedného jazyka existuje preklad v druhom jazyku, často aj s výslovnosťou, komentármi, príkladmi. Niektoré väčšie slovníky obsahujú aj druhu časť, v ktorom sú slová pre spätný preklad z druhého jazyka do prvého. Slovníky môžu byť aj špecializované, napr. sa obmedzujú len na odborné termíny z konkrétnej oblasti. [4]

4 LMF

LMF (Lexical Markup Framework) je ISO (International Organization for Standardization, konkrétne ISO/TC37) štandard pre spracovanie prirodzeného jazyka a strojovo čitateľné slovníky. Cieľom je štandardizácia zásad a metód týkajúcich sa jazykových prostriedkov v kontexte viacjazyčnej komunikácie a kultúrnej rozmanitosti. [5]

4.1 Ciele LMF

Cieľom LMF je poskytnúť spoločný model pre vytváranie a používanie lexikálnych zdrojov, riadiť výmenu dát medzi týmito prostiedkami a umožniť zlúčenie veľkého počtu jednotlivých elektronichých zdrojov za účelom vytvoriť rozsiahle globálne elektronické zdroje.

Typy jednodlivých konkretizácii LMF možu zahŕňať jednojazyčné alebo viacjazyčné lexikálne zdroje. Rovnaké špecifikácie musia byť použité pre malé i veľké lexikóny, jednoduché i komplexné, pre písané i hovorové lexikálne reprezentácie. Popisy siahajú od morfológie, cez syntax až po počítačom pomáhané preklady. Pokryté jazyky nie sú obmedzené iba na európske jazyky, ale pokrývajú všetky prirodzené jazyky. LMF je schopný reprezentovať väčšinu lexikónov, vrátane WordNet, EDR a PAROLE slovníkov. [5]

4.2 Štruktúra LMF

ISO/TC37 sú v súčasnej době rozpracované ako špecifikácie na vysokej úrovni a zaoberajú sa so slovnou segmentáciou,anotáciou, štruktúrou, multimediálnymi kontajnermi a slovníkmi. Tieto normy sú založené na špecifikácii zaoberajúcimi sa konštantami, konkrétne kategórii údajov, jazykovými kódmi, skriptovacími kódmi, kódmi krajín a Unicode.

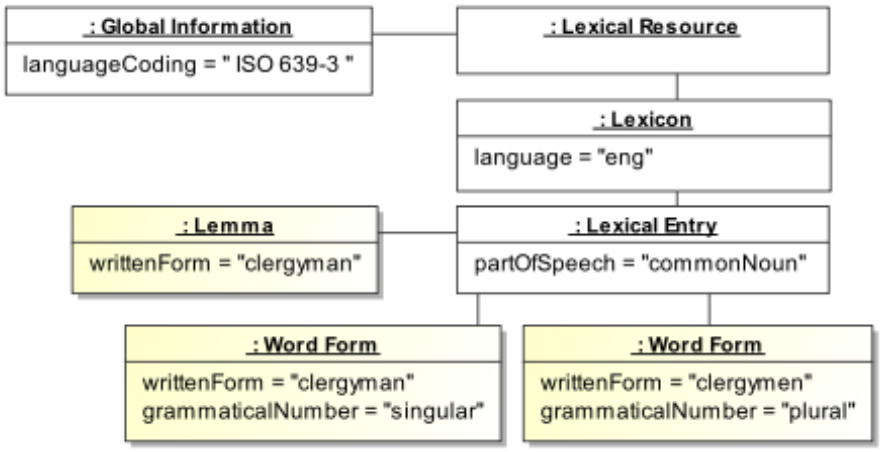
Špecifikácia na vysokej úrovni poskytuje konštrukčné prvky, ktoré sú skrášlené štandardizovanými konštantami. Nízko úrovňova špecifikácia poskytuje štandardizované konštanty ako metadata.

LMF sa skladá z nasledujúcich komponent:

- Základný balík, ktorý predstavuje štrukturálnu kostru popisujúcu základnú hierarchiu informácií v lexikálnom zázname
- Rozšírenia balíka, ktoré popisujú opätovné použitie základných súčastí v spojení s ďalšími komponentami potrebnými pre konkrétne lexikálne zdroje. Rozšírenia sú špecificky zamerané na morfológiu, MMR, NLP syntax a sémantiku, viacslovné expresné vzory. [5]

4.3 Jednoduchý LMF

V nasledujúcom príkladu je lexikálny záznam spojený s dvomi formami lemmy – konkrétne sa jedná o singulár a plurál tohoto slova. Kódovanie je nastavené pre celý dokument. Jazyk je nastavený pre celý lexikon, ako môžeme vidieť v nasledujúcom UML diagrame:



Ukážka 5.1.: Ukážka diagramu LMF

Elementy Lexical Resource, Global Information, Lexicon, Lexical Entry, Lemma a WordForm určujú štruktúru slovníku. Sú špecifikované v LMF dokumente. [5]

Diagram 5.1 môžeme previesť do LMF reprezentácie:

```
<LexicalResource dtdVersion="15">
    <GlobalInformation>
        <feat att="languageCoding" val="ISO 639-3"/>
    </GlobalInformation>
    <Lexicon>
        <feat att="language" val="eng"/>
        <LexicalEntry>
            <feat att="partOfSpeech" val="commonNoun"/>
            <Lemma>
                <feat att="writtenForm" val="clergyman"/>
            </Lemma>
            <WordForm>
                 <feat att="writtenForm" val="clergyman"/>
                 <feat att="grammaticalNumber" val="singular"/>
            </WordForm>
            <WordForm>
                <feat att="writtenForm" val="clergymen"/>
                <feat att="grammaticalNumber" val="plural"/>
```

```
            </WordForm>
        </LexicalEntry>
    </Lexicon>
</LexicalResource>
```

Ukážka 5.1: LMF

5 Strojovo čitateľné slovníky

5.1 Úvod do problematiky

Aj napriek tomu, že existujú postupy prevodu slovníkov do jednotnej podoby, nikdy nie sú dokonalé a stále sa stretávame s problémami pri samotnom prevode. Práve tieto problémy boli podnetom pre moju prácu.

Prvým problémom sú chyby, ktoré vznikajú pri skenovaní a OCR. Či už sa bavíme o nekvalitnom skenovaním, alebo chybnom OCR programu, ovplyvňuje to celkový výsledok slovníku. Pri prevode môže vzniknúť neúplný slovník, kde sa preskočia heslá, ba dokonca aj celé záznamy, čo môže viesť k nevalidnému formátu XML. Pri takýchto slovníkoch je automatická oprava často problémová a niekedy až nemožná. V ďalších kapitolách sa zameriam na riešenie týchto problémov.

5.2 Rozdelenie obsahu značiek

Pri prevode do strojovej podoby sa stretávame s nepríjemnosťou, kde je viacero príkladov, prekladov, alternatív uvedených v jednej značke, oddelené napr. bodkočiarkou. Pre človeka je jasné, kde končí jeden a kde začína druhý príklad, ale pre stroj sa jedná o jeden príklad. Ak chceme tieto výrazy rozdeliť, musíme mať jasne definovaný oddeľovač, ktorý môžeme použiť v našom systéme.

V slovníku vypadá toto združenie nasledujúco:

```
<WordForm>
    <feat att="writtenForm" val=" utrhnout se; odtrhnout se; oprostit se; urvat se"/>
</WordForm>
```

Z vyššie uvedeného príkladu môžeme vidieť, že sa jedná o preklad jedného slova z cudzieho jazyka, do českého. Ideálne by sme potrebovali túto značku obsahujúcu štyri preklady rozdeliť do štyroch separátnych značiek, aby sme sa mohli ľahšie orientovať v slovníku:

```
<WordForm>
    <feat att="writtenForm" val=" utrhnout se"/>
    <feat att="writtenForm" val=" odtrhnout se "/>
    <feat att="writtenForm" val=" oprostit se "/>
    <feat att="writtenForm" val=" urvat se"/>
</WordForm>
```

Týmto spôsobom dosiahneme prehľadný výpis výrazov, s ktorým môžeme ďalej pracovať.

Pri vytvorení systému bude treba rozmýšľať nad možnosťou špecifikácie konkrétnych značiek, poprípade predkov. Slepo vyhľadávať v celom strome a snažiť sa rozdeliť daný výraz, by mohlo mať nechcené výsledky. Jedným problémom by bolo spomalenie programu, druhým by mohlo byť rozdelenie nechceného textu.

Taktiež treba vybrať rozumný preddefinovaný oddeľovač, aby aj v prípade nezadania oddeľovača na vstupu, program fungoval a rozdelil významy. Samozrejme ideálne je vedieť presný oddeľovač, ktorý sa používa v danom slovníku. Vtedy si budeme istí, že máme správne vstupy, a tým pádom dosiahneme požadované výsledky. Problém však môže nastať, ak zadáme ako oddeľovač znak, ktorý má konkrétny zmysel vo vete, ako napr. čiarka. Systém neviem určiť, či sa jedná o oddeľovač, a tým pádom to má rozdeliť do separátnych značiek, alebo sa jedná o znak patriaci do súvetia. Toto bude mať za následok rozdelenie súvetia do dvoch značiek a bude to brať ako dva separátne výrazy.

5.3 Skratky

V slovníku sa často objavujú skrátené formy hesiel, tzv. skratky. Dôvod používania skratiek je jednoduchý: úspora (času, miesta, vyjadrovania apod.).

Keďže chceme touto prácou dosiahnúť aj zrozumiteľnejší slovník, je vhodné vykonať nahradenie skratiek za ich plný význam.

Skratky sa v slovníkoch objavujú v nasledujúcich tvaroch:

- „skr.“ – tu patria ustálené značky, ktoré sa vždy píšu rovnakým spôsobom. Zvyčajne končia spoluhláskou, napr. podst. – podstatné meno, p. – pán.
- „x.“ , kde „x“ je prvý znak heslového slova. Jedná sa o slovo v základnom tvare.
- „x-yz“, kde „x“ je prvý znak heslového slova a „yz“ je koncovka pozostávajúca z 1..n znakov.
- Prípona typu „-xy“ , kde „x“ je písmeno z heslového slova a „y“ je 1…n písmen, ktoré chceme nahradiť.

Úryvok zo slovníka obsahujúci skratky:

```
<Lemma>
    <feat att="writtenForm" val="pekný"/>
</Lemma>
<WordForm>
    <feat att="writtenForm" val="p. chlapec"/>
</WordForm>
```

```
<WordForm>
    <feat att="writtenForm" val="p-é dievča"/>
</WordForm>
<WordForm>
    <feat att="writtenForm" val="-á obloha"/>
</WordForm>
<Lemma>
    <feat att="writtenForm" val="Liptovský Mikuláš"/>
</Lemma>
<WordForm>
    <feat att="writtenForm" val="L-ého M-a"/>
</WordForm>
<Lemma>
    <feat att="writtenForm" val="Ireneus"/>
</Lemma>
<WordForm>
    <feat att="writtenForm" val="-ea"/>
</WordForm>
```

Tu sa treba zamyslieť nad spôsobom nahradzovania. Ak máme prvé písmeno slova (vo forme „x.“) a nahradzujeme skratku iba základnou formou slova, alebo sa jedná o všeobecne známu skratku (povolanie, skratky slovných pádov apod.), nahradenie bude jednoduché. Známe skratky je najlepšie vydefinovať niekde v separátnom súbore a prvotná kontrola skratky by mala byť zameraná práve na tieto známe skratky.

Pri skratkách predstavujúce príponu to bude zložitejšie. Treba si uvedomiť, že sa môže jednať jednak o jednoduchú príponu, ktorú iba dosadíme za základ slova, alebo sa jedná o príponu, ktorá odstráni časť základu. Zároveň existujú ešte prevzaté slová z cudzieho jazyka (končia napr. na „um“, „eus“), ktoré sa správajú trošku inakšie.

Ak vezmeme všetko vyššie opísané do úvahy, výstup by mal vypadať nasledovne:

```
<Lemma>
    <feat att="writtenForm" val="pekný"/>
</Lemma>
```

```
<WordForm>
    <feat att="writtenForm" val="pekný chlapec"/>
</WordForm>
<WordForm>
    <feat att="writtenForm" val="pekné dievča"/>
</WordForm>
<WordForm>
    <feat att="writtenForm" val="pekná obloha"/>
</WordForm>
<Lemma>
    <feat att="writtenForm" val="Liptovský Mikuláš"/>
</Lemma>
<WordForm>
    <feat att="writtenForm" val="Liptovského Mikuláša"/>
</WordForm>
<Lemma>
    <feat att="writtenForm" val="Ireneus"/>
</Lemma>
<WordForm>
    <feat att="writtenForm" val="Irenea "/>
</WordForm>
```

5.4 Alternatívne výrazy

V slovníkoch sa podobne ako so skratkami, stretávame aj s alternatívnymi výrazmi (ďalej už len alternatívy). Alternatívy bývajú rozdelené lomítkom (/), prípadne zátvorkami. Rovnako, ako skratky, alternatívy predstavujú skrátenú formu zápisu, kde namiesto toho, aby sme vypisovali dva rôzne výrazy, s minimálnym rozdielom, do dvoch separátnych značiek, zapíšu sa do jednej značky.

Rozoznávame dva druhy zápisu alternatív:

- rozdelenie lomítkom

Príklad 1:

```
<WordForm>
    <feat att="writtenForm" val="dnes/zajtra plánujem odovzdať bakalársku prácu"/>
</WordForm>
```

- vyjadrenie zátvorkami

Príklad 2:

```
<WordForm>
    <feat att="writtenForm" val="preplávať úžin(o)u"/>
</WordForm>
```

Pri jednoduchom príklade, aký je uvedený vyššie, stačí rozdeliť danú značku do dvoch separátnych. Líšiť sa budú iba v jednom slove:

Príklad 1:

```
<WordForm>
    <feat att="writtenForm" val="dnes plánujem odovzdať bakalársku prácu"/>
    <feat att="writtenForm" val="zajtra plánujem odovzdať bakalársku prácu"/>
</WordForm>
```

Príklad 2:

```
<WordForm>
    <feat att="writtenForm" val="preplávať úžinou"/>
    <feat att="writtenForm" val="preplávať úžinu"/>
</WordForm>
```

Komplikovanejšie to bude až pri viacerých lomítkach, prípadne kombinácia lomítka a zátvorky:

```
<WordForm>
    <feat att="writtenForm" val="dnes/zajtra/pozajtra plánujem odovzdať bakalársku prácu
    a preplávať úžin(o)u"/>
</WordForm>
```

V tomto prípade sa musí vymyslieť algorizmus, ktorý bude prechádzať celý výraz a spraví všetky možné kombinácie (v tomto prípade 6).

Obyčajné rozdeľovanie nemôžeme aplikovať na výrazy, ktoré obsahujú predložky, resp. zvratné slovesá. Vtedy treba brať predložky a zvratné slovesá ako súčasť slova pred/za ním. Tieto „špeciálne slová“ by bolo dobré vydefinovať do samostatného súboru.

Príklad:

```
WordForm>
    <feat att="writtenForm" val="zahnal sa/nezahnal sa na neho/na ňu"/>
</WordForm>
```

Požadovaný výsledok:

```
<WordForm>
    <feat att="writtenForm" val="zahnal sa na neho"/>
    <feat att="writtenForm" val="zahnal sa na ňu"/>
    <feat att="writtenForm" val="nezahnal sa na neho"/>
    <feat att="writtenForm" val="nezahnal sa na ňu"/>
</WordForm>
```

5.5 Výslovnosť

Niektoré slovníky obsahujú aj výslovnosť slova. Bohužiaľ, výslovnosť je občas písaná v skrátenej forme, čím sa stáva ťažsie zrozumiteľná. Aby sme tomu lepšie porozumeli, je vhodné previesť túto skrátenú formu na plný tvar.

Aby sme mohli nahrádzať skrátenú formu za úplnú, potrebujeme najskôr nájsť úsek slova, ktorý budeme meniť. Dôležitým faktorom je vstup, na základe ktorého vieme dopredu určiť, o ktorú časť slova sa jedná. Prakticky to možno rozdeliť na začiatok, stred, koniec. Vstup by mal okrem pomlčiek obsahovať aspoň dva ďalšie znaky, v opačnom prípade nevieme jasne určiť, čo máme nahradiť. Jednoznakové skratky vieme nahradiť jedine triviálne prípady (napr. i -> y, z -> s). Tieto prípady musia byť explicitne vypísané v mapovacej tabuľke.

Príklad vstupu:

```
<WordForm>
    <feat att="writtenForm" val="Louisiana "/>
    <feat att="phoneticForm" val="lujsi-"/>
    <feat att="grammaticalGender" val="feminine"/>
</WordForm>
```

Na príklade môžeme vidieť, že sa jedná o nahradzovanie zo začiatku (pomlčka je iba na konci slova). Nahradenie v tomto prípade bude pomerne jednoduché, lebo prvé aj posledné písmeno vieme nájsť v heslovom slove. Po namapovaní začiatku a konca prejdeme k samotnému nahradeniu a dostaneme výsledné slovo. Tu si treba uvedomiť, že výskytov môže byť viacero, preto by bolo dobré porovnávať aj indexy. V prípade, že rozdiel indexov bude v rozumnej miere, vykonáme nahradenie.

Ukážka výstupu:

```
<WordForm>
    <feat att="writtenForm" val="Louisiana "/>
    <feat att="phoneticForm" val="lujsiana"/>
    <feat att="grammaticalGender" val="feminine"/>
</WordForm>
```

Ďalším typov vstupu môže byť skratka, ktorá nebude vyhovovať predošlému porovnávaniu, takže začiatočná alebo koncová časť výslovnosti nebude odpovedať žiadnemu znaku z heslového slova.

Príklad:

```
<WordForm>
    <feat att="writtenForm" val="Wyoming"/>
    <feat att="phoneticForm" val="vajou-"/>
    <feat att="grammaticalGender" val="masculine"/>
</WordForm>
```

Pre tento prípad musí existovať mapujúca tabuľka, v ktorej je definovaná dvojica „výslovnosť – písaný výraz“. Po úspešnom nájdení výslovnosti v tabuľke, nahradíme túto výslovnosť písanou formou, a proces sa môže od začiatku pustiť – hľadá sa prvé písmeno pred pomlčkou.

```
<WordForm>
    <feat att="writtenForm" val="Wyoming"/>
    <feat att="phoneticForm" val="vajouming"/>
    <feat att="grammaticalGender" val="masculine"/>
</WordForm>
```

5.6 Chyba syntaxe

V prípade, že vstupné XML bude obsahovať syntaktickú chybu, na výstup sa vypíše informácie o nevalidnom súbore. Táto kontrola sa vykoná pri snahe rozparsovať XML súbor.

6 Implementácia

Ovládanie aplikácie je konzolové. Spracovanie vstupov predávaných pri spustení programu je vykonávané pomocou modulu *optparse.* Na vstupu program očakáva minimálne dva parametre:

- **-i -** predstavuje vstupný súbor, ktorý chceme spracovať
- **-o –** predstavuje výstupný súbor, do ktorého chceme zapísať výsledok

Program okrem vyššie uvedených ponúka nasledovné prepínače:

- **-t** (task)**–** predstavuje úlohu/funkciu, ktorú chceme spustiť
  - **eq/equivalents** – rozdelenie príkladov, prekladov
  - **abbr/abbreviations** – nahradenie/doplnenie skratiek
  - **alt/alternatives** – rozčlenenie alternatív z jedného tagu do separátnych
  - **ph/phonetics** – doplnenie fonetických foriem
- **-p** (parent element)- môžeme definovať rodičovský element, na ktorý sa má daná funkcia aplikovať
- **-a** (attribute name) – možnosť definovať názov atribútu elementu, na ktorý sa má daná funkcia aplikovať
- **-s** (splitter) – možnosť definovať delimiter, na základe ktorého sa budú preklady, príklady rozdeľovať do separátnych tagov. Štandardne je nastavená bodkočiarka (**;**)

Povinné parametre sú –i , -o , zvyšné sú voliteľné. V prípade, že sa na vstupu objavia iba povinné parametre, vykonajú sa všetky funkcie nad daným slovníkom. Všetky funkcie vytvárajú separátny .log súbor, kde sa zaznamenávajú všetky zmeny vykonané danou funkciou. Vďaka tomuto detailu som prišiel počas vývoja na pár nedostatkov, ktoré som opravil.

6.1 Python

V počiatočnej fáze si treba vybrať vhodný programovací jazyk. V mojom prípade vyhral Python, či už z dôvodu, že je multiplatformný, alebo kvôli jeho jednoduchosti a prehľadnosti.

Python je dynamický, objektovo orientovaný skriptovací programovací jazyk, ktorý je vyvíjaný ako Open Source. Umožňuje používať nie len objektovo orientované paradigma, ale aj procedurálne a funkcionálne. Vďaka tomu má vynikajúcu vyjadrovaciu schopnosť. Kód je v porovnaní s inými jazykmi krátky a dobre čitateľný. Je považovaný za jeden z najjednoduchších jazykov, vhodný pre začiatočníkov.

Poprednou vlastnosťou jazyka je produktívnosť z hľadiska písania programov. Je možné ho používať na písanie malých skriptov, rovnako ako pre vývoj veľkých softwarových projektov. Ponúka podporu k integrácii s ostatnými jazykmi. Ľahko sa vkladá do iných aplikácií, kde slúži ako skriptovací jazyk.

Python ponúka mocné vstavané typy (zoznam, slovníky, reťazce apod.), funkcie, knižníc, ktoré obsahujú stovky modulov poskytujúce rutiny pre širokú škálu služieb vrátane regulárnych výrazov a TCP/IP relácií.

V roku 2008 sa Python vývojári rozhodli, že prídu na trh s novou verziou Python. Spočiatku to nazvali „Python 3000“, čo sa časom zmenilo na verziu 3.x. Radikálne na tejto novej verzii bolo práve to, že nebola spätne kompatibilná s Python 2.x. [6,7]

6.2 Regulárne výrazy

Aby sme boli schopní meniť dáta a pracovať s nimi, musíme najskôr byť schopní ich jednoducho získať. Na to slúžia regulárne výrazy. Regulárny výraz je spôsob, akým sa na základe textového vzoru rozpoznávajú a často získavajú dáta. Pomocou nich môžeme v Pythonu analyzovať text a získavať potrebné dáta. Výraz je definovaný reťazcom, ktorý môže obsahovať špeciálne znaky, ktoré umožňujú, aby jednému regulárnemu výrazu odpovedalo množstvo rôznych konkrétnych reťazcov. Takého znaky nazývame metaznaky. [8]

6.3 Kontrola slov

Nositeľom informácie je najmä značka *feat*, v ktorej nájdeme atribut *att* s názvom typu informácie,a *val*, ktorý nesie obsah informácie. Kontrolu slov budeme prevádzať práve nad týmito informáciami (konkrétne nad typom WrittenForm) a to hlavne pri vytváraní plných slov zo skratiek. Pri výslovnosti to nemá význam, keďže vzniknuté slová sa v slovníku nevyskytujú.

Pre kontrolu použijeme software *Enchant,* o ktorom sa dozvieme v ďalšej kapitole.

6.4 Enchant

Enchant je software, ktorý vznikol za účelom zjednotiť prístup k viacerým existujúcim programom, ktoré kontrolujú pravopis.

Enchant zabaľuje spoločnú sadu funkcií prítomných v rôznych existujúcich programoch, a obsahuje stabilné API. Je schopný mať viacero načítaných programov naraz (aspell, ispell, myspell atď.). [9]

6.5 Systém na rozdelenie obsahu značiek

Systém má za úlohu rozdeliť príklady, preklady, apod. uvedené v jednej značke, do viacerých. Tým dosiahneme prehľadný výpis výrazov, s ktorým môžeme ďalej pracovať.

Najskôr si musíme určiť oddeľovač, ktorý zvyčajne býva bodkočiarka, čiarka, lomítko apod. Pre preštudovaní slovníkov, som priamo v kóde preddefinoval bodkočiarku (**;**). Samozrejme v prípade, že máme slovník, kde sa používa ako oddeľovač iný znak, napr. čiarka, tak môžeme použiť v konzole parameter „*-s*“, ktorým preddefinovanú hodnotu prepíšeme nami zadanou hodnotou.

Na vstup dostane metóda rozparsované XML, oddeľovač a rodičovskú značku, pod ktorou máme jednotlivé značky na spracovanie.

Príklad zápisu v slovníku:

```
<WordForm>
    <feat att="writtenForm" val=" utrhnout se; odtrhnout se; oprostit se; urvat se"/>
</WordForm>
```

V prípade, že daná značka obsahuje oddeľovač, rozdelí sa celý výraz na *n* (n = počet oddeľovačov + 1) separátnych značiek. Keďže má vstupná značka tri oddeľovače, vzniknú štyri separátne značky s rovnakým typom (writtenForm). Tento typ je skopírovaný z pôvodnej značky.

```
<WordForm>
    <feat att="writtenForm" val=" utrhnout se"/>
    <feat att="writtenForm" val=" odtrhnout se "/>
    <feat att="writtenForm" val=" oprostit se "/>
    <feat att="writtenForm" val=" urvat se"/>
</WordForm>
```

Kvôli rôznorodosti slovníkov, kde sa nedalo zovšeobecniť značku, nad ktorou sa majú dané úpravy konávať, bol pridaný parameter „*-p*“, ktorý určuje rodičovskú značku. Týmto si môžeme určiť konkrétnu značku v prípade, že vieme o rozdieloch v danom slovníku. Preddefinovaná hodnota v kóde je WordForm, táto hodnota bola určená na základe analýzy slovníkov.

Problém môže nastať, ak zadáme ako oddeľovač čiarku. Systém neviem určiť, či sa jedná o oddeľovač, a tým pádom to má rozdeliť do separátnych značiek, alebo sa jedná o znak patriaci do súvetia. Toto bude mať za následok rozdelenie súvetia do dvoch značiek a bude to brať ako dva separátne výrazy.

6.6 Systém na nahradenie skratiek

Systém slúžiaci na nahradenie skrátených foriem slov za ich plný ekvivalent.

Základný princíp nahradzovania je nasledovný:

- v značke *feat* hľadáme bodku (prehľadávame iba zadané rodičovské značky – napr. WordForm)
- v prípade, že sa tam bodka nachádza, najskôr skontrolujeme, či sa jedná o všeobecne známu skratku (názov povolania, oslovenie apod.) - skúšame ju nájsť v tabuľke v súbore enumerations.py. Následne skratku nahradíme za jej plnú formu
- v prípade, že danú skratku nenájdeme v tabuľke známych skratiek, spravím analýzu skratky a na základe tejto skratky vykonáme ďalšie kroky (opísané nižšie)
- skratku nahradíme za jej plný tvar

Skratky sa v slovníkoch objavujú v nasledujúcich tvaroch:

- „skr.“ – pokiaľ nájde bodku, tak prehľadáva zoznam všetkých skratiek, ktorý je definovaný v separátnom súbore (enumerations), ak tam danú skratku nájde, nahradí ju jej plným ekvivalentom, ktorý je definovaný v enumerations.

  Ukážka záznamu v enumerations.py:

  {"abbr":"podst\.", "full":"podstatné jméno"}

- „x.“ , kde „x“ je prvý znak heslového slova. Skratka sa nahradí za plný tvar, ktorý je uvedený v značke Lemma.

  Vstup:

```
<Lemma>
    <feat att="writtenForm" val="pekný"/>
</Lemma>
<WordForm>
    <feat att="writtenForm" val="p. chlapec"/>
</WordForm>
```

  Výstup:

```
<WordForm>
    <feat att="writtenForm" val="pekný chlapec"/>
</WordForm>
```

- „x-y“, kde „x“ je prvý znak heslového slova a „y“ je koncovka obsahujúca 1..n znakov, ktorú chceme zmeniť v heslovom slove. Po nájdení prvého znaku nájdeme odpovedajúce heslové slovo. Následne sa snažíme aplikovať koncovku na základe pravidiel:
  - buď je za pomlčkou samohláska a posledné písmeno je samohláska, tak sa to nahradí
  - v ostatných prípadoch sa to pridá na koniec, okrem výnimiek:napr. Karluv, -ova

  Systém počíta aj s prípadom, kde heslo pozostáva z viacerých slov. V tomto príklade aplikuje prvú skratku na prvé heslové slovo a druhú skratku za druhé.

  Vstup:

```
<Lemma>
    <feat att="writtenForm" val="Liptovský Mikuláš"/>
</Lemma>
<WordForm>
    <feat att="writtenForm" val="L-ého M-a"/>
</WordForm>
```

  Výstup:

```
<WordForm>
    <feat att="writtenForm" val="Liptovského Mikuláša"/>
</WordForm>
```

- Prípona typu „-xy“ , kde „x“ je hľadané písmeno a „y“ je 1..n písmen z heslového slova, ktoré chceme nahradiť. Program sa najskôr snaží nájsť prvé písmeno za pomlčkou v posledných 3 písmenách, po úspešnom nájdení sa nahradí zvyšok slova za „y“. Ak nenájde, tak platí pravidlo z predošlého príkladu - nahradí sa za samohlásku

  Vstup:

```
<Lemma>
    <feat att="writtenForm" val="lízátko"/>
</Lemma>
<WordForm>
    <feat att="writtenForm" val="-kách"/>
</WordForm>
```

Výstup:

```
<Lemma>
    <feat att="writtenForm" val="lízátko"/>
</Lemma>
<WordForm>
    <feat att="writtenForm" val="lízátkách"/>
</WordForm>
```

Ak máme za pomlčkou iba jednu samohlásku, a heslové slovo končí taktiež na samohlásku, tak to jednoducho nahradíme:

Vstup:

```
<Lemma> <feat att="writtenForm" val="pekný"/></Lemma>
<WordForm> <feat att="writtenForm" val="-á"/></WordForm>
```

Výstup:

```
<WordForm> <feat att="writtenForm" val="pekná "/></WordForm>
```

V prípade, že heslové slovo končí na spoluhlásku, a zároveň nekončí na „eus“, „um“, tak sa samohláska pridá na koniec. V tejto časti je ešte ošetrená možnosť, ktorá môže nastať pri prevzatých slovách z cudzieho jazyka, ktoré končia na „eus“, „um“.

Vstup:

```
<Lemma> <feat att="writtenForm" val="oblačnost"/></Lemma>
<WordForm> <feat att="writtenForm" val="-i"/></WordForm>
```

Výstup:

```
<WordForm> <feat att="writtenForm" val="oblačnosti "/></WordForm>
```

V prípade prevzatých slov („-eus“, „-um“) to vypadá následovne:

Vstup:

```
<Lemma> <feat att="writtenForm" val="Ireneus"/></Lemma>
<WordForm> <feat att="writtenForm" val="-ea"/></WordForm>
```

Výstup:

```
<WordForm> <feat att="writtenForm" val="Irenea "/></WordForm>
```

6.7 Systém na rozdelenie alternatív

Systém vykonávajúci rozdelenie alternatívnych výrazov do separátnych značiek. Vstupom systému je okrem spracovaného stromu aj rodičovská značka, ktorá uvádza miesto hľadania alternatív. Následne nájde značku *feat* typu *WrittenForm* a snaží sa nájsť lomítko, ktoré považuje ako oddeľovač dvoch slov. Po nájdení si uloží zvyšok výrazu do premennej. K rozdeleným slovám, ktoré boli vložené do separátnych značiek, vloží zvyšok výrazu z premennej. Ukážeme si to na príklade:

Vstup:

```
<WordForm>
    <feat att="writtenForm" val="dnes/zajtra plánujem odovzdať bakalársku prácu"/>
</WordForm>
```

Výstup:

```
<WordForm>
    <feat att="writtenForm" val="dnes plánujem odovzdať bakalársku prácu"/>
    <feat att="writtenForm" val="zajtra plánujem odovzdať bakalársku prácu"/>
</WordForm>
```

V texte sa môže objavoviť aj viac alternatív za sebou, takže musíme dať tento algoritmus do cyklu. Cyklus pobeží *n-krát* , kde *n* predstavuje „počet lomítok+1“. Novo vzniknuté slová/výrazy sú taktiež pravopisne kontrolované pomocou Enchant.

Algoritmus prakticky funguje na báze vytvárania všetkých možných spôsobov, následne zmaže pôvodný výraz, odstráni duplicity a vypíše výrazy bez lomítka. Jednoduchšie si to bude ukázať na príklade:

Máme výraz: *a b /c d /e* .

Najskôr odstránime jedno lomítko a vznikne nám:

a b d /e

a c d /e

Na tieto dva výrazy aplikujeme funkciu ešte raz a dostaneme:

a b d

a b e

a c d

a c e

Teraz už iba dosadíme zvyšok originálneho výrazu do značky a máme štyri separátne výrazy.

Popísaný algoritmus platí aj na zátvorky s rozdielom, že najskôr zátvorku ignorujem, a tak dostanem prvý výraz, a potom ju použijem (odstránim zátvorky) a dostaneme druhý výraz.

Vstup:

<WordForm> <feat att="writtenForm" val="preplávať úžin(o)u"/></WordForm>

Výstup:

<WordForm>
<feat att="writtenForm" val="preplávať úžinou"/>
<feat att="writtenForm" val="preplávať úžinu"/>
</WordForm>

Lomítka aj zátvorky sa môžu vyskytovať v značke zároveň, takže treba počítať aj s touto variantou a umožniť kombinácie lomítka so zátvorkami:

<WordForm>
<feat att="writtenForm" val="dnes/zajtra/pozajtra plánujem odovzdať bakalársku prácu a preplávať úžin(o)u"/>
</WordForm>

Výstupom bude šesť separátnych značiek.

V texte sa môžu objaviť aj predložky, poprípade zvratné slovesá (sa,si), takže nemôžeme brať slovo ako slovo. Rozdeľujeme ich na základe definovanej tabuľky v enumerations.py, kde je zoznam týchto výnimkových slov. Predložky, resp. zvratné slovesá, berem ako súčasť slova.

Vstup:

WordForm>
<feat att="writtenForm" val="zahnal sa/nezahnal sa na neho/na ňu"/>
</WordForm>

Výstup:

WordForm>
<feat att="writtenForm" val="zahnal sa na neho"/>
<feat att="writtenForm" val="zahnal sa na ňu"/>
<feat att="writtenForm" val="nezahnal sa na neho"/>
<feat att="writtenForm" val="nezahnal sa na ňu"/>
</WordForm>

6.8 Systém pre zápis výslovnosti

Systém slúžiaci pre zápis plnej formy výslovnosti, namiesto skrátenej.
Samotný algoritmus môžeme rozdeliť na dve časti:

- nahradenie skrátenej formy za plnú na základe nájdeného prvého a posledného znaku skratky s reťazcom v heslovom slove. V tomtu prípade postačí jednoduché nahradenie úsekov.
- nahradenie na základe mapovacej tabuľky, kedy sa pôvodná skrátená forma nenachádza v heslovom slove, ale namapovaním príslušného reťazca na iný, získame nový reťazec, ktorý už v heslovom slove nájdeme.

Kombinácia oboch metód je možná, dokonca si dovolím tvrdiť, ze najčastejšie nahradenie je riešené kombináciou.

Najskôr si ukážeme príklad, kde sa jedná o nahradenie od začiatku a úspešne nájdeme posledné písmeno skratky:

Vstup:

```
<WordForm>
    <feat att="writtenForm" val="whisky "/>
    <feat att="phoneticForm" val="vi-"/>
</WordForm>
```

Nahradenie v tomto prípade bude pomerne jednoduché, lebo posledné písmeno vieme nájsť v heslovom slove. Rozdiel indexov je jedna, takže môžeme nahradiť,takže možeme nahradiť:

Výstup:

```
<WordForm>
    <feat att="writtenForm" val="whisky "/>
    <feat att="phoneticForm" val="visky"/>
</WordForm>
```

Rovnakým spôsobom by fungovala prípona „-ki“ s rozdielom, že by sa vyhľadávalo od konca

Vstup:

```
<WordForm>
    <feat att="writtenForm" val="whisky "/>
    <feat att="phoneticForm" val="-ki"/>
</WordForm>
```

Výstup:

```
<WordForm>
    <feat att="writtenForm" val="whisky "/>
    <feat att="phoneticForm" val="whiski"/>
</WordForm>
```

Obe metódy si môžeme ukázať na nasledujúcom príklade:

- heslové slovo: syntetizovat
- skratka, ktorú chceme aplikovať: -ty-
- postup:
    - snažíme sa nájsť „t“ v slove „syntetizovat“. Po úspešnom nájdení prejdeme na posledné písmeno v zo skratky, takže „y“.
    - „y“ sa nepodarilo v heslovom slove nájsť, takže sa pozerá do mapovacej tabuľky, kde nájdeme tam dvojicu: „y“ – „i“, takže namiesto „y“ hľadáme „i“
    - po úspešnom nájdení aj posledného písmena vykonáme kontrolu indexov, nahradenie a vznikne nám „syntetyzovat“

Keďze môže v heslovom slove existovať viac výskytov skratky, musím spraviť určité obmedzenie, aby sme napr. z deväť písmenového slova nedostali dvojpísmenové, napr. pri vyššie uvedenom príklade, kedy sme mali heslo „syntetizovat“ a chceli by sme aplikovať „-ty-“, tak by bolo riziko, že by vzniklo „syntyzovat“. Tento prípad som ošetril kontrolou indexov v slovách, kde rozdiel môže byť maximálne jeden znak.
Vstupná skratka by mal okrem pomlčiek obsahovať aspoň dva ďalšie znaky, v opačnom prípade nevieme jasne určiť, čo máme nahradiť. Jednoznakové skratky vieme nahradiť jedine triviálne prípady (napr. i -> y, z -> s). Tieto prípady musia byť explicitne vypísané v mapovacej tabuľke.

Ďalším typom vstupu môže byť skratka, ktorá nebude vyhovovať predošlému porovnávaniu, takže koncová časť výslovnosti nebude odpovedať žiadnemu znaku z heslového slova.

Príklad:

```
<WordForm>
    <feat att="writtenForm" val="Wyoming"/>
    <feat att="phoneticForm" val="vajou-"/>
    <feat att="grammaticalGender" val="masculine"/>
    <feat att="grammaticalFeature" val=" -u"/>
</WordForm>
```

Pre tento prípad musíme nájst v mapovacej tabuľke koniec danej skratky. Po úspešnom nájdení výslovnosti v tabuľke, nahradíme túto výslovnosť písanou formou a proces sa môže pustiť od začiatku.

```
<WordForm>
    <feat att="writtenForm" val="Wyoming"/>
    <feat att="phoneticForm" val="vajouming"/>
    <feat att="grammaticalGender" val="masculine"/>
    <feat att="grammaticalFeature" val="Wyomingu"/>
</WordForm>
```

Po kontrole štatistík v logovacích súboroch som zistil, že je úspešnosť približne 80%. Z tých 20% neúspešných je väčšina v nesprávnom tvare:

- snaha o nahradenie iba jedného písmena, kde ani ja sám neviem posúdiť, čo nahradiť
- chybný vstup skratky, napr. :
  LEMMA: zorientovať, skratka s-o- ;
  LEMMA : věro~ , skratka –o-u;
  LEMMA: trauma , skratka –ty-

7 Testovanie

Neoddeliteľnou súčasťou každého vývoja musí byť aj testovanie softwaru za účelom získať všetky možné informácie o systéme. Súčasťou testovania je hľadania chýb, vďaka ktorým vieme systém vylepšiť.

Testovanie som spočiatku vykonával na svojich vstupoch a slovníkoch, akonáhle som mal základnú funkcionalitu funkčnú, prešiel som na reálne slovníky na školskom serveri minerva. Zameral som sa na adresáre „../dictformXX/lmf_dicts/“ , kde sa nachádzajú slovníky v LMF podobe, ktoré boli prevedené v rámci bakalárskych prác iných študentov. Až pri testovaní na reálnych slovníkov som narazil na rôzne problémy, či už sa jednalo o pamäťovú a časovú náročnosť balíčku *xml.dom.minidom,* ktorý som neskôr zamenil za *xml.etree.ElementTree* , alebo šlo o neuvedomenie si rôznych prípadom,ktoré môžu nastať.

7.1 Logovanie

Logovanie ide ruka v ruke s testovaním. Každý systém vytvára logy v zložke ../logs/.*.log. Názov súborov sa líši na základe systému, ktorý daný súbor vytvoril, v názve je taktiež časová stopa.

8 Záver

Cieľom práce bolo vytvoriť systém pre prevod slovníkov vo formáte LMF do podoby lepšie využiteľné pre strojové spracovanie, takže v tvaru, kedy je maximum informácii uvedených explicitne. K dosiahnutiu tohto cieľa som vytvoril sadu systémov, ktoré sú zamerané na konkrétny problém.

Na začiatku bolo potrebné zanalyzovať štruktúru LMF, presne pochopiť obsah značiek, aby sme ich vedeli správne použiť ako zdroj informácií. Následne sa dalo pristúpiť k návrhu a samotnej implementácii.

V práci sa možno dočítať o každom systéme, dôvodu, prečo je daný systém potrebný a samozrejme, opis daného systému riešiaci konkrétny problém.

Z výsledkov vyplýva, že daná sada systémov je schopná s vysokou mierou úspešnosti previesť slovníky do viac zrozumiteľnej podoby. Hlavný dôvod neúspechu boli nesprávne vstupné dáta, ktorý obsahovali buď nezmysel, alebo znak, ktorý tam nepatril. V ďalšej fáze vývoja by bolo potrebné vydefinovať väčší počet príkladov kvôli správnej výslovnosti. V systéme, ktorý rozdeľuje značky na základe rozdeľovača by bolo dobré dotiahnuť prípad, keď je čiarka oddeľovačom. Tu by bolo potrebné rozpoznávať na základe kontextu, či sa jedná o súvetie, alebo dva separátne výraz.

V neposlednej rade by som sa zameral aj na užívateľské rozhranie pre jednoduchšie ovládanie .

9 Bibliografia

[1] MLÝNKOVÁ, Irena a Jaroslav POKORNÝ. *XML technologie: principy a aplikace v praxi*. 1. vyd. Praha: Grada, 2008, 267 s. Průvodce (Grada). ISBN 978-80-247-2725-7.

[2] KOSEK, Jiří. *XML pro každého: podrobný průvodce*. 1. vyd. Praha: Grada, 2000, 163 s. ISBN 80-7169-860-1.

[3] https://docs.python.org/2/library/xml.etree.elementtree.html

[4] http://cs.wikipedia.org/wiki/Slovn%C3%ADk

[5] http://en.wikipedia.org/wiki/Lexical_Markup_Framework

[6] HARMS, Daryl D a Kenneth MCDONALD. *Začínáme programovat v jazyce Python*. Vyd. 1. Brno: Computer Press, 2003, xviii, 456 s. ISBN 80-7226-799-x.

[7] PILGRIM, Mark. *Ponořme se do Python(u) 3: Dive into Python 3*. Praha: CZ.NIC, c2010, 430 s. CZ.NIC. ISBN 978-80-904248-2-1.

[8] GOYVAERTS, Jan a Steven LEVITHAN. *Regulární výrazy: kuchařka programátora*. Vyd. 1. Brno: Computer Press, 2010, 381 s. ISBN 978-80-251-1935-8.

[9] http://www.abisource.com/projects/enchant/

[10] http://aspell.net/

[11] https://docs.python.org/2/library/xml.dom.minidom.html

Seznam príloh

Príloha 1. CD so zdrojovými kódmi a písomnou správou vo formáte PDF