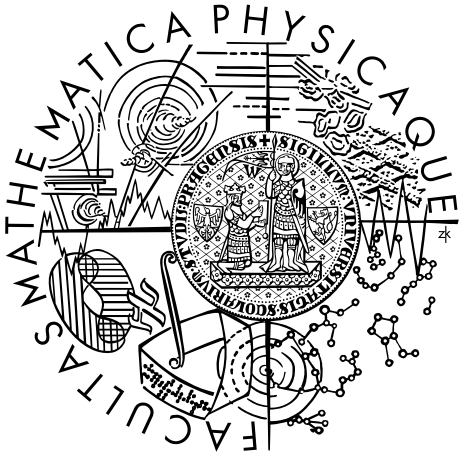
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Dominik Macháček

Automatická extrakce konkordancí z Internetu

Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Vincent Kríž
Studijní program: Informatika
Studijní obor: obecná informatika

Praha 2016

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Automatická extrakce konkordancí z Internetu

Autor: Dominik Macháček

Ústav: Ústav formální a aplikované lingvistiky

Vedoucí bakalářské práce: Mgr. Vincent Kríž, Ústav formální a aplikované lingvistiky

Abstrakt: Konkordance jsou věty, které obsahují zadané cílové slovo. Jsou důležitými předměty zkoumání ve všech oblastech lingvistiky. Velký počet konkordancí má také využití při řešení problému lexikální desambiguace. Jazykové korpusy ale neposkytují dostatečný počet konkordancí například některých anglických sloves.

V této práci se zabýváme návrhem a implementací konzolové aplikace pro automatickou extrakci zadaného počtu anglických konkordancí. Aplikace bere na vstupu cílové slovo, slovní druh a počet vět, a následně na Internetu vyhledá a extrahuje zadaný počet anglických vět obsahujících cílové slovo jako zadaný slovní druh. Vytvořili jsme také knihovnu v Pythonu, s jejíž pomocí se dá aplikace modifikovat pro libovolný jiný jazyk, a zveřejnili jsme ji na serveru PyPI. Součástí práce je také webová stránka umožnující vyzkoušet si aplikaci přes webové rozhraní.

Klíčová slova: automatická extrakce, konkordance, internet

Title: Automatic concordance extraction from the Internet

Author: Dominik Macháček

Institute: Institute of Formal and Applied Linguistics

Supervisor: Mgr. Vincent Kríž, Institute of Formal and Applied Linguistics

Abstract: Concordances are sentences containing given target word. They are profitable research objects in all linguistics fields. A big amount of concordances is exploited during lexical desambiguation problem solving. Language corpora are not able to supply sufficient number of concordances of some English verbs.

In this thesis we elaborate a design and implementation of a console application for automatic extraction of given number of English concordances. The application gets on its input a target word, a part-of-speech and a number of sentences. Consecutively it seeks out and extracts on the Internet desired number of English sentences containing a target word as given part-of-speech. We created also a Python library which allows a modification of the application to any arbitrary language. We published it on PyPI server. A part of a work is also a webpage allowing users to try out the application through web interface.

Keywords: automatic extraction, concordance, internet

Děkuji Vincentu Krížovi za příkladné vedení mého ročníkového projektu a bakalářské práce.

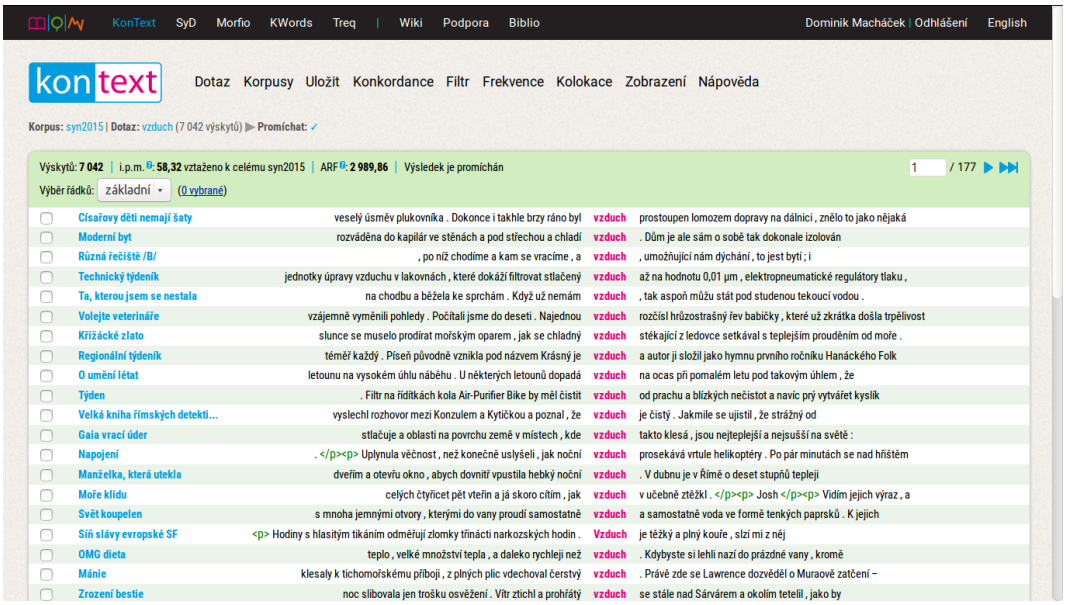
Obsah

1. Úvod

Zvolme libovolné cílové slovo v libovolném jazyce. Konkordance je věta, která obsahuje zadané cílové slovo. Příklad konkordancí je na obrázku 1.1.

Velký počet konkordancí s jedním klíčovým slovem má využití například v sémantice při řešení problému rozlišování významů víceznačných slov,[1] což je klasický otevřený problém v oblasti počítačové lingvistiky. Největší problém přitom tvoří rozlišování sloves.

Metoda CPA (*Corpus pattern analysis*, Hanks, 2013) řeší tento problém především právě pro anglická slovesa, a to tak, že jejich významy popisuje pomocí syntagmatických[2] a sémantických vzorů. Existují metody (Kríž, 2012) pro automatické určování těchto vzorů. Ukázalo se ale, že mohou být vylepšeny o sémantické příznaky, pro jejichž vytvoření existují statistické modely (Holub a kol., 2012), pro něž je potřeba co největší množství konkordancí cílových sloves. Čím větší počet konkordancí mají k dispozici, tím je metoda pro automatickou desambiguaci přesnější. Současné zdroje nenabízejí dostatečný počet konkordancí od některých slov a proto se tento problém snažíme v naší práci vyřešit.



Obrázek 1.1: Konkordance cílového slova *vzduch* zobrazené kontextovým manažerem KonText.

Konkordance mají také využití ve všech oblastech lingvistiky, ve kterých se používají jazykové korpusy, například pro statistickou analýzu jazykových jevů, ověřování hypotéz, objevování a ověřování gramatických pravidel apod. Korpusy jsou lingvistům přístupné obvykle přes tzv. *korpusové manažery*,[3] což jsou speciální programy pro efektivní vyhledávání v korpusových datech. Příkladem korpusového manažeru je například KonText na obrázku 1.1. Výsledkem vyhledávání korpusových manažerů jsou konkordance, které jsou předmětem dalšího zkoumání.

[1] `https://en.wikipedia.org/wiki/Word-sense_disambiguation`
[2] `http://oltk.upol.cz/encyklopedie/index.php5/Syntagma`
[3] `https://wiki.korpus.cz/doku.php/pojmy:korpusovy_manazer`

Zdroje konkordancí používané v současné době jsou jazykové korpusy. Popisujeme je detailně v kapitole 2. Z nich lingvisté získávají konkordance z textů, které jsou do korpusů zařazeny. To má ale mnohé nevýhody, protože v korpusech se obvykle najde jen málo konkordancí málo frekventovaných slov. Proto jsme v kapitole 3 analyzovali, evaluovali a vybrali vhodné internetové zdroje konkordancí. S jejich pomocí jsme vytvořili aplikaci, která dostane na vstupu cílové slovo a počet vět a poté z Internetu automaticky extrahuje věty, které obsahují požadované cílové slovo. Přitom je schopna stáhnout statisíce až miliony kvalitních konkordancí.

V naší práci jsme vytvořili konzolovou aplikaci ConcordanceCrawler, která je navržena pro extrakci konkordancí v angličtině, a knihovnu v Pythonu, s pomocí které se dá modifikovat pro jakýkoliv jiný jazyk, případně vyměnit její moduly za jiné. Její popis je v kapitole 4. Je zveřejněna pod open-source licencí na serveru PyPI,[4] může ji proto volně stáhnout a používat kdokoliv na celém světě. Vytvořili jsme také webové demo (viz kapitola 7), které umožňuje vyzkoušení naší aplikace online bez nutnosti instalace. Je zveřejněno na odkazu `http://quest.ms.mff.cuni.cz:10381/`.

V kapitole 5 popisujeme detailně strategii pro extrakci konkordancí z dokumentu a jeho části. Naše aplikace je schopna vracet konkordance v jednom zadaném kódování, používá filtr anglického jazyka, přesnou metodu větné segmentace a také automatickou lemmatizaci a automatické určení slovních druhů, takže je schopna stahovat pouze konkordance obsahující cílové slovo v zadaném slovním druhu a ne v jiných.

V kapitole 6 se zabýváme testováním výkonu a kvality aplikace a v závěrečné kapitole jsme shrnuli celou práci, subjektivně zhodnotili klady aplikace a popsali možnosti jejího dalšího rozšíření.

V příloze A je k nahlédnutí příklad použití aplikace, její vstupy a výstupy. V příloze B je obsah elektronické přílohy této práce.

[4] `https://pypi.python.org/pypi/ConcordanceCrawler/`

2. Zdroje konkordancí využívané v současné době

Konkordance se získávají z autentických textů. Texty v mluvené nebo psané podobě jsou často sdružovány do tzv. *jazykových korpusů*. Ty jsou budovány s cílem sloužit jako záznam a pokud možno co nejobjektivnější model jazykové empirie v určité oblasti.[1] Proto vznikají korpusy, které jsou podle tématu textů, které obsahují, obecné neboli obsahující texty ze všech oblastí, a také specializované, například korpusy z textů jediného autora, texty z parlamentních diskusí, akademických publikací apod. Přitom se korpusy snaží být co nejvíce reprezentativní, aby danou jazykovou populaci zrcadlily co nejlépe.

Největší existující korpusy anglických textů jsme shrnuli do tabulky 2.1.

korpus	**homepage**	**miliard slov**
NOW Corpus	`http://corpus.byu.edu/now/`	2.8
Global Web-Based English	`http://corpus.byu.edu/glowbe/`	1.9
Wikipedia Corpus	`http://corpus.byu.edu/wiki/`	1.9
Corpus of Contemporary American English	`http://corpus.byu.edu/coca/`	0.52
British National Corpus	`http://www.natcorp.ox.ac.uk/`	0.1
Corpus of American Soap Operas	`http://corpus.byu.edu/soap/`	0.1

Tabulka 2.1: Největší existující korpusy anglických textů a počet slov, která obsahují. Podle `www.corpus.byu.edu`.

Největší korpus angličtiny je NOW Corpus, který obsahuje přes 2.8 miliardy slov z webových zpravodajských serverů z 20 zemí. Tento korpus se začal tvořit v roce 2010 a každý den v něm přibývají 4 miliony slov z nových článků. Druhý největší korpus je Global Web-Based English s 1.9 miliardy slov z internetových textů z 20 anglicky mluvících zemí, další je stejně velký Wikipedia Corpus vytvořený z textů z Wikipedie. Tyto tři korpusy jsou speciální, obsahují texty pouze z jedné určité oblasti a proto je možné, že některé jazykové jevy se v nich nevyskytují.

Největší obecné korpusy angličtiny jsou Corpus of Contemporary American English (COCA) a British National Corpus (BNC), ty ale obsahují pouze 500, respektive 100 milionů slov.

Maximální počet anglických konkordancí z největších korpusů

Ze seznamu frekvencí lemmat[2] z British National Corpus (Kilgarriff, 1995) jsme zjistili, že některá lemmata se v BNC vyskytují málo často, konkrétně pouze asi 800-krát na 100 milionů slov. Takovými lemmaty jsou například slovesa *chase, relate, urine* nebo *produce*, méně častá lemmata ale v seznamu ani nejsou. To znamená, že z BNC bychom mohli získat 800 konkordancí takovýchto lemmat,

[1] `http://wiki.korpus.cz/doku.php/pojmy:korpus`

[2] `http://wiki.korpus.cz/doku.php/pojmy:lemma`

z COCA asi 4 160 (za předpokladu, že jsou i v americké angličtině a v COCA stejně častá) a z uvedených velkých jazykových korpusů s 1.9 miliardou slov asi 15 200 konkordancí. Z NOW Corpusu bychom získali zhruba 23 200 konkordancí, pokud jsou ta slova stejně častá i v těchto korpusech jako v BNC.

I ty největší korpusy jsou tedy jako zdroje konkordancí značně omezené. Jelikož v posledních desetiletích vzrostlo používání Internetu, je přirozeným krokem, že se k němu obrátila pozornost vědců k potenciálně neomezenému zdroji textů.

2.1 Práce využívající Internet jako zdroj textů

Prozkoumali jsme existující vědecké práce, které se zabývají získáváním textů z Internetu.

Tvorba webových korpusů

Existují práce, které se zabývají tvorbou webových korpusů. Mezi ně patří například práce Vysoce kvalitní webový korpus češtiny (Spoustová a Spousta, 2012).

WebCorp

Projekt WebCorp[3] je souprava nástrojů, která poskytuje přístup k World Wide Webu jako ke korpusu. Vznikl v roce 2002 na University of Liverpool. Od té doby se vyvíjí a je používán do současnosti.

V současné době poskytuje mimo jiné[4] webovou složbu WebCorp Live, jejíž rozhraní pro zadávání dotazů je podobné jako rohraní vyhledávačů. Uživatel má možnost zadat klíčové slovo, vybrat pevnou šířku kontextu, filtr odkazů a jazyk, na výběr je angličtina, čínština a němčina. WebCorp následně provede dotaz na vyhledávač FAROO nebo Bing, získá odkazy, stáhne dokumenty a z nich extrahuje konkordance obsahující zadané slovo. Kromě konkordancí vypisuje také URL dokumentu společně s kódováním a datem vzniku.

Konkordance vypisuje ve formátu *keyword-in-context* (KWIC),[5] tedy s pevně zvoleným počtem znaků či slov před a za klíčovým slovem. Na jeden dotaz WebCorp stáhne a zpracuje 100 dokumentů a vypíše všechny konkordance, které v něm jsou nalezeny. Screenshot stránky s výsledkem vyhledávání WebCorp Live je na obrázku 2.1.

WebCorp není vhodný pro naše použití, protože je schopen poskytnout pouze malé množství konkordancí s kontextem, jehož šířka je omezená, to znamená, že není schopen vypisovat celé věty. Navíc se výstup nedá stáhnout.

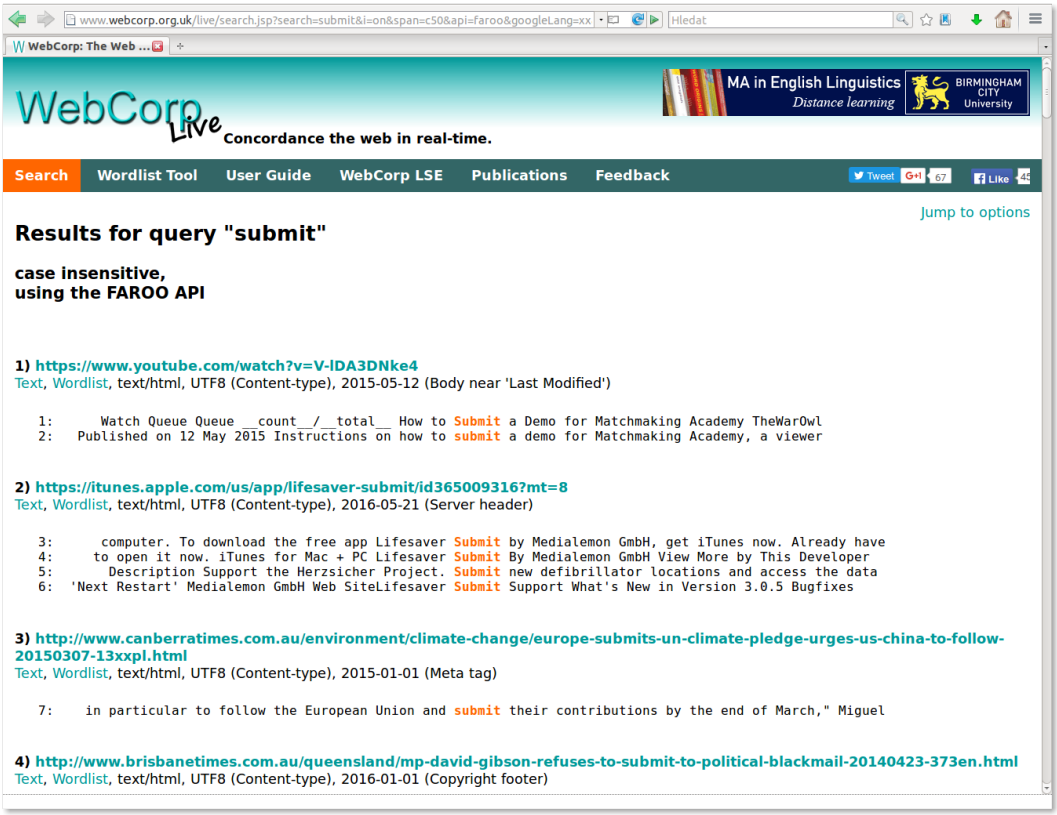
GoogleLing

Článek GoogleLing: The Web as a Linguistic Corpus (Smarr a Grow, 2002) popisuje software, který umožňuje jakýkoliv webový vyhledávač nebo online kor-

[3] `http://www.webcorp.org.uk/live/`

[4] Poskytuje také službu WebCorp Linguist's Search Engine spravovanou na Birmingham City University. Je to několik webových korpusů přístupných přes jednotný korpusový manažer. Viz `http://wse1.webcorp.org.uk/`. Je potřeba ji odlišovat od WebCorp Live.

[5] `https://en.wikipedia.org/wiki/Key_Word_in_Context`



Obrázek 2.1: Screenshot stránky s výsledkem vyhledávání WebCorp Live.

pusový manažer modifikovat, aby vypisoval výsledky na dotazy zadané v dotazovacím jazyku, který vytvořili autoři. Umožňuje vyhledávání konkordancí podle omezení na slovní druhy, omezení uvnitř vět a mezi slovními n-gramy. Software zobecňuje dotaz pro vyhledávač a na výsledku aplikuje nástroje automatického zpracování přirozených jazyků, aby byly nalezeny požadované konkordance. Předběžné testy podle autorů ukazují, že tento software velmi vylepšuje možnosti použití webu jako korpusu.

Software se nám nepodařilo najít, vyzkoušet ani o něm zjistit více informací. V článku je odkaz,[6] na kterém má být vystaven pro veřejnost, na něm se ale nic nenachází.

[6] `http://nlp.stanford.edu/googleling`

3. Internetové zdroje pro získávání konkordancí a jejich problémy

Internet je jako zdroj autentických anglických textů velmi rozsáhlý a snadno přístupný. Prozkoumali jsme proto několik přístupů pro získávání alespoň stovek tisíc anglických vět se zadaným slovem, které se vyskytují na Internetu.

První možností jsou tzv. *webové crawlery*,[1] což jsou programy, které prochází web a stahují všechny dokumenty ze všech stránek, které navštíví. Začínají na jedné nebo několika málo pevně daných URL a poté přechází na všechny odkazy, které se nacházejí v dokumentu na URL a které dosud nenavštívily. Tímto způsobem se teoreticky dá projít celý web. Ve skutečnosti je ale web ohromně rozsáhlý a pro některá slova, která se mohou vyskytnout na vstupu naší aplikace, může trvat velmi dlouho, než se webový crawler dostane k dokumentům, které obsahují konkordance s těmi slovy. Tento přístup se tedy pro naše použití nehodí.

Další možností jsou webové vyhledávače. Jeden z nich se ukázal být vhodným a využitelným zdrojem. V první sekci proto popisujeme webové vyhledávače, v sekci 3.2 jejich omezení a možné způsoby, jak je vyřešit. V sekci 3.3 popisujeme existující knihovny a programy na stahování odkazů v Pythonu. V sekci 3.4 jsme porovnali vyhledávače podle míry blokování automatického provozu a v závěrečné sekci 3.5 popisujeme finální výběr zdroje konkordancí.

3.1 Webové vyhledávače

Webový vyhledávač[2] je služba, která slouží k hledání dokumentů, které obsahují požadované informace. Každý vyhledávač vypíše na zadaný dotaz takzvanou *SERP* (Search Engine Result Page, stránka s výsledky vyhledávače), na které jsou odkazy na stránky s požadovanými dokumenty, úryvky z těch dokumentů (takzvané *snippety*) a případně i další informace. Na jedné SERP je obvykle mezi 10 a 100 odkazů, zbylé odkazy bývají k dispozici na dalších stránkách. Výsledky bývají seřazeny podle důležitosti.

V tabulce 3.1 uvádíme seznam některých webových vyhledávačů, které jsou volně přístupné na webu.

Podrobněji jsme zkoumali tři webové vyhledávače, a to Google, Bing a Seznam.cz.

3.2 Omezení webových vyhledávačů

Webové vyhledávače se nedají využít jednoduše a přímočaře. Jsou zde totiž různá omezení, která podrobně popisujeme v této části.

[1] `https://en.wikipedia.org/wiki/Web_crawler`

[2] `https://en.wikipedia.org/wiki/Web_search_engine`

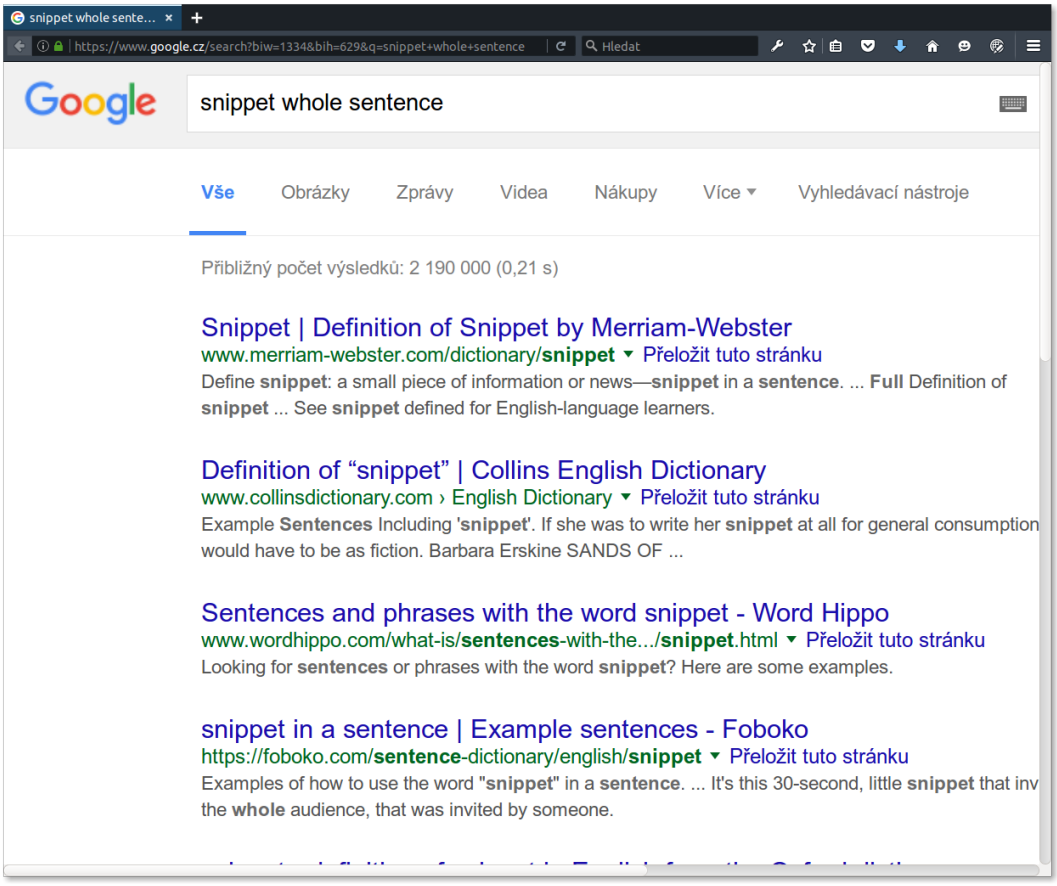
vyhledávač	adresa
Google	`http://www.google.com`
Bing	`http://www.bing.com`
Yandex	`http://www.yandex.ru`
Seznam.cz	`http://www.seznam.cz`
Yahoo!	`http://www.yahoo.com`
Ask	`http://www.ask.com`
DuckDuckGo	`http://www.duckduckgo.com`
Baidu	`http://www.baidu.com`
FAROO	`http://www.faroo.com`

Tabulka 3.1: Webové vyhledávače.

3.2.1 Ve snippetech nejsou vždy celé věty

Není možné brát konkordance rovnou ze snippetů, protože u všech vyzkoušených vyhledávačů se často stává, že ve snippetu není celá věta obsahující hledané klíčové slovo, ale pouze její úryvek ukončený třemi tečkami. Příklad takových snippetů je vidět na obrázku 3.1.

Proto jsme se rozhodli, že aplikace stáhne celý dokument na nalezeném odkazu a extrahuje z něj konkordance. To má tu výhodu, že v dokumentu může být více konkordancí s klíčovým slovem. Tímto způsobem získáme všechny.



Obrázek 3.1: Screenshot SERP vyhledávače Google.

3.2.2 Limit počtu získatelných odkazů na jeden dotaz

Kdybychom nechali některý vyhledávač hledat přímo naše klíčové slovo, brzy zjistíme, že nám nenabídne neomezený počet různých odkazů. Například Seznam.cz nenabídne více než 100 SERP po 10 odkazech. Při hledání slova *solve* je 100. SERP[3] poslední, už na se na ní nenachází odkaz na žádnou další. Když změníme URL[4] tak, abychom dostali např. 200. SERP, dostaneme jako odpověď tu stejnou stou SERP.

Podobné je to u Bingu, tam dostaneme nanejvýš kolem 500 či 300 odkazů. Toto číslo se u různých dotazů liší, například u slova *how* je to kolem 500,[5] u slova *Prague* je jich kolem 274.[6] V každém případě ale nedostaneme desetitisíce odkazů, jak bychom potřebovali.

Stejně tak Google na žádný dotaz nezobrazí více než 1 000 výsledků, při pokusu získat více výsledků zobrazí HTML dokument se zprávou *„Litujeme, Google na žádný dotaz nezobrazí více než 1000 výsledků. (Požádali jste o výsledky od 8000.)“*[7]

Bazwords

Potřebujeme tedy nějakým způsobem zvýšit počet různých odkazů, které nám mohou vyhledávače vydat. Představme si, že chceme stáhnout konkordance se slovem *find*. Pak nám vyhledávač najde jisté množiny odkazů, když necháme hledat postupně *find Alabama*, *find car*, *find cake* a *find kvol*. Dá se očekávat, že ty množiny budou různé a budou mít jen málo společného, protože *Alabama, car, cake* a neexistující slovo *kvol* mají málo společného. Vyhledávač se bude snažit hledat stránky, na kterých se nachází obě slova, jedno klíčové slovo, které nás zajímá, a druhé jako návnada, nějaké nesmyslné nebo náhodné slovo. Nazýváme ho *bazword*.

Bazword musí být jen jedno slovo, protože potom se vyhledávač snaží najít stránky, na kterých je co nejvíce slov z dotazu, i když třeba ne všechna, a přitom může pominout naše cílové slovo.

Strategie výběru bazwords

Existují různé strategie výběru bazwords, my jsme vybrali čtyři, které jsme implementovali do naší aplikace. Uživatel má možnost použít přepínač `-b` nebo `--bazword-generator`, kterým si může vybrat generátor generující bazwords podle čtyř různých strategií:

- `RANDOM` – náhodně vybraná čtyřpísmenná slova z písmen anglické abecedy (například *ghec, dnvg, ahoj, qvns...*)

[3] `http://search.seznam.cz/?q=solve&count=10&pId=vAAg9hEyHX-ijUQXFQBL&from=990`

[4] `http://search.seznam.cz/?q=solve&count=10&pId=vAAg98TK1peOjzDCytK_&from=2000`

[5] `http://www.bing.com/search?q=how&go=Odeslat+dotaz&qs=ds&first=599&FORM=PORE`

[6] `http://www.bing.com/search?q=prague&go=Odeslat+dotaz&qs=ds&first=290&FORM=PERE3`

[7] `https://www.google.cz/search?q=hello&ei=_0sfV8vZOIqsa7S-utAJ&start=8000&sa=N&biw=1334&bih=629`

- `WIKI_ARTICLES` – na anglické Wikipedii je odkaz na náhodný článek[8], generátor postupně stahuje náhodné články a jako bazwords postupně vrací slova z článků
- `WIKI_TITLES` – podobné jako předchozí generátor, ale vrací pouze slova z titulků náhodných článků
- `NUMBERS` – vrací postupně celá přirozená čísla 0, 1, 2, 3, ...

3.2.3 Omezení strojového provozu

Vyhledávače se většinou snaží detekovat a omezovat strojový provoz. Ten, kdo generuje příliš mnoho dotazů v krátkém čase nebo přistupuje k vyhledávači jiným způsobem než ručně přes prohlížeč, může být zablokován, a to obvykle tak, že se mu objeví zpráva, že má opsat nějaký text z obrázku a prokázat se, že je člověk. (Jedná se o tzv. CAPTCHA neboli Turingův test). Toto blokování trvá nějakou dobu, od několika hodin po několik dní, nebo může být i trvalé.

Je v našem zájmu vybrat takový zdroj a způsob získávání odkazů (a potažmo konkordancí), který bude pokud možno nejrychlejší. To se dá udělat různými způsoby, obejitím blokace, vyjednáním výjimky nebo výběrem vyhledávače, který blokuje automatický provoz pokud možno málo.

Obejití blokace

Napadlo nás, že blokaci obejdeme tak, že přesně zjistíme, za jakých podmínek vyhledávač začne blokovat a pak budeme stahovat odkazy či konkordance tak, aby stahoval co nejrychleji a současně aby tyto podmínky nikdy nebyly splněny.

Problémem je, že přesné fungování blokace u vyhledávačů, a hlavně u Googlu, není snadné zjistit, protože je poměrně náročné vyzkoušet a ověřit limit, u kterého ještě nedojde k blokaci. To se dá vyzkoušet pouze metodou pokusu a omylu, přičemž další pokus můžeme ze stejného počítače provést až potom, co vyprší časový limit blokace.

Dalším důvodem je, že informace o blokaci nejsou veřejně dostupné a pokud jsou, tak se může jednat o staré a neaktuální údaje[9]. Navíc vyhledávač může podmínky blokace kdykoliv změnit, takže kdybychom vytvořili program, který počítá s určitými parametry blokace, a ony se změnily, tak náš program náhle může přestat fungovat.

Vyřešení CAPTCHA

Blokace by se dala obejít tak, že bychom vytvořili automatický nástroj, který opíše text z obrázku a tím vyřeší CAPTCHA. Toto je ale poměrně náročné, CAPTCHA jsou vytvářeny tak, aby je počítače automaticky nebyly schopny vyřešit. Také se může stát, že naučíme počítač řešit určitý typ CAPTCHA, ale vyhledávač ho časem vymění za jiný a museli bychom začít od začátku. Každopádně cestou vyřešení CAPTCHA jsme se nevydali.

[8] `https://en.wikipedia.org/wiki/Special:Random`

[9] Například na stránce `http://google-scraper.squabbel.com/` jsou tyto údaje z roku 2012.

Pokus o spolupráci s vyhledávačem

Mohli bychom se pokusit vyjednat u provozovatele vyhledávače výjimku, aby neblokoval automatický provoz z námi vytvořené aplikace, alespoň pokud se její uživatel prokáže, že ji používá pouze pro vědecké účely.

Proto jsme emailem kontaktovali firmu Seznam.cz. Použili jsme k tomu emailovou adresu, kterou nám doporučila pracovnice této firmy. Řekla nám, že na zprávu na tuto adresu určitě odpoví. Nicméně i po více než 5 měsících zůstala naše zpráva bez odezvy.

Stejně tak jsme kontaktovali provozovatele vyhledávače Bing a rovněž jsme se nedočkali žádné odpovědi.

Pokusili jsme se napsat email také do společnosti Google, ovšem tady jsme ztroskotali už na vyhledání té správné emailové adresy. Nikde na stránkách této společnosti se nedala najít, nepodařilo se nám ji ani najít s pomocí webového vyhledávače.

Bing Search API

Vyhledávač Bing provozuje službu Bing Search API. Když se u ní programátor zaregistruje a zaplatí požadovanou částku, může strojově přes svoji aplikaci bez omezení provést určitý počet dotazů za určité období[10]. Například 5 000 dotazů za měsíc je zdarma, milion dotazů za měsíc ale stojí 2 000 USD. Tyto peníze by musel platit každý uživatel naší aplikace.

K registraci do této služby je potřeba vytvořit účet u Microsoft Services. Pokusili jsme se o to a nebylo to nijak jednoduché. Domníváme se, že pokud bychom pro používání naší aplikace požadovali už jenom vytvoření toho účtu, tak by to mnohé potenciální uživatele naší aplikace odradilo. Navíc další možné uživatele by odradila nutnost platby. Proto jsme se rozhodli tuto službu nepoužít.

3.3 Existující knihovny a programy na stahování odkazů

Zjistili jsme, že existuje několik knihoven a programů v Pythonu, které umožňují stahování odkazů z vyhledávačů Bing nebo Google a jsou volně dostupné. Několik jsme jich vyzkoušeli a zjišťovali o nich, zda jsou vhodné pro použití v naší aplikaci a zda by naše aplikace mohla získávat konkordance stejným nebo podobným způsobem.

Knihovny stahující z Bingu

Seznam knihoven stahujících z Bingu uvádíme v tabulce 3.2. Všechny používají placené Bing Search API, které jsme se rozhodli nepoužít.

[10] Ceník této služby je k dispozici zde: `https://datamarket.azure.com/dataset/5BA839F1-12CE-4CCE-BF57-A49D98D29A44`

název	homepage
pyBingSearchAPI	`https://github.com/xthepoet/pyBingSearchAPI`
bingsearch	`https://github.com/guitarparty/bingsearch`
py-bing-search	`https://github.com/tristantao/py-bing-search`

Tabulka 3.2: Knihovny pro stahování odkazů z vyhledávače Bing.

Knihovny stahující z Googlu

Rovněž jsme vyzkoušeli sedm knihoven pro stahování odkazů z Googlu. Většina z nich nefungovala, nedala by se použít pro některou verzi Pythonu nebo využívala postup, který už Google nepodporuje. Jejich seznam následuje v tabulce 3.3, podrobnější popis je v elektronické příloze.

název	homepage
google	`https://breakingcode.wordpress.com/2010/06/29/google-search-python/`
Google-Search-API	`https://github.com/abenassi/Google-Search-API`
xgoogle	`http://www.catonmat.net/blog/python-library-for-google-search/`
google-search-cli	`https://github.com/zweifisch/google-search-cli`
web_search	`http://www.connellybarnes.com/code/web_search/`
GoogleScraper	`https://github.com/NikolaiT/GoogleScraper`
googler	`https://github.com/commx/googler`

Tabulka 3.3: Knihovny pro stahování odkazů z vyhledávače Google.

Výjimku tvoří následující knihovna.

GoogleScraper

GoogleScraper (Tschacher, 2015) je nástroj, který umožňuje stahování odkazů (ale i obrázků, videí a zpráv) z vyhledávačů Google, Bing, Yahoo, Yandex, Baidu a DuckDuckGo. Měl by mít několik módů, stahování přes proxy, paralelní vícevláknové stahování a takzvaný *selenium mode*. To je stahování odkazů tím, že ovládá skutečný prohlížeč. V tomto módu vyhledávač nemůže snadno detekovat automatický provoz.

Po vyzkoušení jsme ale zjistili, že spousta funkcionalit této knihovny nefunguje a vlastně není vůbec dokončená. Například nefungovalo ověření proxy serverů a selenium mode.

Tato knihovna se v naší aplikaci dala využít na parsování SERP z Bingu. Na to používá pythonovskou knihovnu `lxml`, která je napsaná v jazyce C a proto je velmi rychlá. Celkem jednoduše se dala z této knihovny extrahovat funkce, která dostala HTML dokument SERP vyhledávače Bing a vrátila seznam odkazů, které se na stránce nacházely, společně s nějakými dalšími potenciálně užitečnými informacemi o nalezených odkazech.

3.4 Porovnání vyhledávačů podle míry blokování automatického provozu

Provedli jsme experiment, který měl porovnat Bing a Google podle toho, jak moc blokují automatický provoz.

Zkusili jsme 60 sekund generovat paralelně X dotazů za sekundu na jeden vyhledávač z jednoho počítače. Přitom jsme sledovali, kolik z našich dotazů uspělo a dostalo na odpověď SERP. Hodnotu X jsme postupně zvyšovali, dokud všechny dotazy pro oba vyhledávače byly úspěšné. S pokusem jsme skončili, když z jednoho vyhledávače místo SERP chodily stránky oznamující blokaci.

vyhledávač	% úspěšných dotazů
Bing	100 %
Google	89 %

Tabulka 3.4: Poměr úspěšných dotazů při 40 dotazech za sekundu po dobu 60 sekund u vyhledávačů Bing a Google.

Pokus dopadl tak, že Google nás zablokoval při 40 dotazech za sekundu asi po 2130 SERP z 2400, po zopakování po několika minutách zablokování stále trvalo. Naproti tomu vyhledávač Bing stále vracel SERP s odkazy. Po zopakování a zvýšení délky pokusu na 180 sekund vyhledávač stále vracel výsledky.

Předpokládáme, že když Bing neblokuje provoz při této zátěži, tak nebude blokovat ani při menší zátěži stovek dotazů za minutu několik dní až měsíců v kuse. Tuto zátěž totiž nejspíš bude generovat obvyklý provoz naší aplikace.

Můžeme tedy říct, že Bing neblokuje automatický provoz tolik jako Google a je pro použití v naší aplikaci vhodnější.

Seznam.cz jsme do pokusu nezahrnuli z toho důvodu, že i kdyby se ukázalo, že je z hlediska blokování alespoň stejně dobrý jako Bing, tak nemá tu výhodu, že pro něj existuje funkční a hotový parser z GoogleScraperu.

3.5 Finální výběr zdroje konkordancí

Nakonec jsme se rozhodli, že konkordance budeme získávat tímto způsobem:

Budeme stahovat odkazy z vyhledávače Bing. Bing jsme vybrali proto, že náš automatický provoz velmi pravděpodobně nebude blokovat, a protože můžeme použít hotové parsování jeho SERP.

Počet stáhnutelných odkazů se budeme snažit zvýšit s pomocí bazwords. Poté navštívíme všechny odkazy a extrahujeme z dokumentů, které se na nich nacházejí, hledané konkordance.

Pro stahování a navštěvování odkazů a extrakci konkordancí vytvoříme vlastní aplikaci. Pro dílčí úkoly jako segmentace dokumentů na věty, filtrace konkordancí atp. se vždy budeme snažit najít, porovnat a vybrat existující knihovny v jazyce Python.

4. Implementace – aplikace a knihovna ConcordanceCrawler

Naše aplikace a knihovna pro automatickou extrakci konkordancí z Internetu má název ConcordanceCrawler. Je naprogramována v jazyce Python a zveřejněna pod open-source licencí na serveru PyPI.[1] Uživatelům a čtenářům této práce je proto k dispozici na odkazu

```
https://pypi.python.org/pypi/ConcordanceCrawler/
```

V této kapitole uvádíme její uživatelskou a programátorskou dokumentaci a odkazujeme také na dokumentaci ConcordanceCrawleru jako knihovny.

4.1 Uživatelská dokumentace aplikace

ConcordanceCrawler je konzolová aplikace primárně určená pro platformu UNIXových operačních systémů. Při spuštění očekává zadání uživatelem definovaných parametrů, mezi nimiž je cílové slovo a požadovaný počet konkordancí. Některé parametry jsou povinné, některé volitelné, některé mají výchozí hodnoty, které platí, pokud uživatel neurčí jinak.

4.1.1 Instalace

Aplikace je naprogramována v jazyce Python a dá se spouštět ve verzích Pythonu 2.7, 3.2 a novějších. Pro spuštění musí mít uživatel nainstalovaný interpretr Pythonu[2] v jedné z těchto verzí. Rovněž je vhodné mít správce pythonovských balíčků `pip`.[3]

Všechny závislé knihovny se nainstalují automaticky až na jednu, a to knihovnu `lxml`. Tu doporučujeme nainstalovat globálně systémovým správcem balíčků. Například na Debianu nebo Ubuntu se to provede příkazem

```
sudo apt-get install python3-lxml
```

Tento příkaz platí pro instalaci pro Python3, pro Python2 doporučujeme následovat instrukcí na stránce této knihovny.[4]

Poté můžeme nainstalovat ConcordanceCrawler:

```
pip install ConcordanceCrawler
```

[1] `https://pypi.python.org/pypi`
[2] `https://www.python.org/`
[3] `https://pip.pypa.io/en/stable/`
[4] `http://lxml.de/installation.html`

Pokud máme ConcordanceCrawler správně nainstalovaný, můžeme zadat příkaz `ConcordanceCrawler -h`, kterým si necháme vypsat jeho nápovědu k parametrům.

Pokud chceme používat také automatickou lemmatizaci a určování slovních druhů, pak musíme ještě doinstalovat knihovnu `textblob`. Není to nezbytně nutné, ConcordanceCrawler můžeme používat i bez toho. Doinstalování `textblobu` je volitelné, protože stáhne knihovnu `nltk` a ta je poměrně velká. Provedeme to příkazy

```
pip install textblob
python -m textblob.download_corpora
```

4.1.2 Spuštění

Aplikace ConcordanceCrawler se dá spustit ve dvou módech. V prvním módu začne stahovat konkordance úplně od začátku, ve druhém módu navazuje na předchozí extrakční úkol, který byl ukončen ještě před dokončením.

Nový extrakční úkol

Nový extrakčním úkol vytvoříme například následujícím příkazem:

```
ConcordanceCrawler submit
```

Tímto se nám stáhne 10 konkordancí slova „*submit*“ jako libovolný slovní druh. Konkordance se budou vypisovat na terminál ve formátu JSON, na standartní chybový výstup (kterým je také terminál) se budou vypisovat údaje o průběhu stahování.

Při vytváření extrakčního úkolu ale můžeme použít také následující parametry:

- `word [word ...]` – Takto zadáme cílové slovo. Musíme zadat alespoň jeho slovníkový tvar, můžeme ale zadat i jeho další morfologické tvary. Pokud nepoužijeme automatickou lemmatizaci a určování slovních druhů, pak se budou hledat konkordance s alespoň jedním ze slov, která zde uvedeme.

- `-p REGEX, --part-of-speech REGEX` je regulární výraz morfologické značky cílového slova. Pokud použijeme jinou hodnotu než výchozí (tou je `.*`, což znamená jakýkoli slovní druh), pak budou hledány pouze ty konkordance, ve kterých se nacházejí slova, která mají lemma shodné s cílovým slovem a jejichž morfologická značka odpovídá regulárnímu výrazu `REGEX`. K tomu je nutno mít nainstalovanou knihovnu `textblob`.

  Morfologické značky jsou převzaty z Penn Treebank (Marcus a kol., 1993), jejich seznam je dostupný zde.[5] Například regulárnímu výrazu `V.*` odpovídají všechny tvary sloves, `N.*` můžeme použít pro podstatná jména a `J.*` pro přídavná jména.

[5] `http://www.clips.ua.ac.be/pages/mbsp-tags`

Příklad použití: Předpokládejme, že naše cílové slovo je *fly* a náš regulární výraz je `V.*`, to znamená, že chceme stahovat slovesa. Potom slovo *flies*, jehož morfologická značka je `VBZ` (sloveso, 3. osoba singuláru prézentu), odpovídá, stejně jako značka `VBD` (sloveso v minulém čase) slova *flew*. Naproti tomu podstatné jméno *fly* (moucha) s morfologickou značkou `NN` neodpovídá, takže věty s tímto slovem budou ignorovány.

Tento způsob zadání jsme zvolili proto, že umožňuje velice variabilní použití, uživatel má možnost zadat stahování třeba jakéhokoliv slovního druhu, pouze podstatných jmen, pouze sloves, sloves ve 3. osobě nebo podstatných jmen v singuláru atp. Podobný nebo stejný způsob zadávání slovního druhu se běžně používá v korpusové lingvistice, tudíž můžeme předpokládat, že pro uživatele nebude problém to správně používat.

- `-n N`, `--number-of-concordances N` – počet konkordancí, které mají být staženy. Výchozí počet je 10.
- `-m M`, `--max-per-page M` – maximální počet konkordancí, které mohou být extrahovány z jednoho dokumentu. Pokud uživatel nepoužije tento přepínač, počet nebude omezený.
- `--disable-english-filter` – tento přepínač vypne filtr angličtiny, takže konkordance budou extrahovány ze všech dokumentů bez ohledu na jeho jazyk. Použití tohoto přepínače má velký vliv na kvalitu korpusu.
- `-e ENCODING, --encoding ENCODING` – nastavení kódování. Pokud se tento přepínač nepoužije, pak jsou extrahovány konkordance ze všech dokumentů bez ohledu na kódování. Pokud uživatel zadá ASCII, tak jsou extrahovány konkordance z vět, které obsahují pouze znaky, které se nacházejí v ASCII. Pokud uživatel zadá jiné kódování, třeba UTF-8, pak jsou extrahovány konkordance pouze v těch dokumentech, které o sobě uvádějí, že jsou v tom kódování.

  Toto nastavení má vliv na kvalitu korpusu a na rychlost stahování.
- `-b {RANDOM,WIKI_ARTICLES,WIKI_TITLES,NUMBERS}` nebo `--bazword-generator {RANDOM,WIKI_ARTICLES,WIKI_TITLES,NUMBERS}` je přepínač pro výběr generátoru bazwords. Výchozí hodnota je `RANDOM`. Generátory bazwords jsme popisovali už v sekci **??**, zde je ale uvádíme ještě jednou:
  - `RANDOM` – náhodně vybraná čtyřpísmenná slova z písmen anglické abecedy (například *ghec, dnvg, ahoj, qvns...*)
  - `WIKI_ARTICLES` – na anglické Wikipedii je odkaz na náhodný článek,[6] generátor postupně stahuje náhodné články a jako bazwords postupně vrací slova z článků
  - `WIKI_TITLES` – podobné jako předchozí generátor, ale vrací pouze slova z titulků náhodných článků
  - `NUMBERS` – vrací postupně celá přirozená čísla 0, 1, 2, 3, ...

[6] `https://en.wikipedia.org/wiki/Special:Random`

- `-o OUTPUT`, `--output OUTPUT` je jméno výstupního souboru (nebo speciálního souboru), do kterého se budou ukládat konkordance. Může být například `concordances.xml`, `concordances.json` nebo i speciální soubor, jakým je `/dev/null`. Pokud už soubor existuje, pak se přepíše. Výchozí hodnotou je standartní výstup.
- `-f {json, xml}, --format` je formát výstupního souboru. Může být buďto XML nebo JSON, ten je výchozí.
- `-v {0,1,2,3}, --verbosity` – nastavení úrovně „upovídanosti“ logovacích výpisů programu.
- `--buffer-size BUFFER_SIZE` – nastavení maximálního počtu objektů v bufferech, které jsou použity pro zajištění toho, že na výstupu nebudou dvě stejné konkordance a že se nebude dvakrát stahovat stejný odkaz. Výchozí hodnota je 1 000 000. Nastavení příliš malého čísla může vést k tomu, že po `BUFFER_SIZE` stažených konkordancích se na výstupu objeví nějaká konkordance podruhé, nastavení příliš velkého čísla může vést k tomu, že se buffery naplní tak, že dojde paměť.
- `--backup-file BACKUP_FILE` – název zálohového souboru, do kterého se uloží nastavení extračního úkolu, aby extrakční úkol mohl být obnoven. Výchozí název je `ConcordanceCrawler.backup` a vytváří se v aktuálním pracovním adresáři.
- `--backup-off` – s tímto přepínačem se nebude vytvářet zálohový soubor

Navázání na extrakční úkol

Ve druhém módu je aplikace schopna rozšířit seznam konkordancí, který stahovala při předchozím spuštění, když bylo stahování přerušeno ještě před dokončením. V tom případě se spustí následujícím příkazem:

```
ConcordanceCrawler --continue-from-backup BACKUP_FILE \
     --extend-corpus EXTENDED_OUTPUT_FILE
```

`BACKUP_FILE` je název souboru zálohy, `EXTENDED_OUTPUT_FILE` je seznam konkordancí, který byl vytvořen v předchozím extrakčním úkolu a který se bude rozšiřovat.

Aplikace načte do paměti stažené konkordance a odkazy, na kterých se nacházely, a poté se nebude navštěvovat stejné odkazy a stahovat stejné konkordance opakovaně.

4.1.3 Příklad spuštění

Následující příkaz spustí extrakční úkol, kterým se stáhne jedna konkordance se slovem *hello*. Seznam konkordancí z tohoto extrakčního úkolu je na obrázku 4.1.

```
ConcordanceCrawler hello -n 1 -v 3
```

```
[
  {
    "start": 0,
    "keyword": "hello",
    "end": 5,
    "url": "http://grokbase.com/t/mysql/maxdb/0469p5zksn/error-in-compl
       ete-backup",
    "date": "2016-04-30 14:45:01.220790",
    "concordance": "Hello Marco,you can find the output and error
       output of TSM's adint2 in the files dbm.ebp and dbm.ebl.",
    "id": 1
  }
]
```

Obrázek 4.1: Ukázkový seznam konkordancí ve formátu JSON obsahující jednu konkordanci se slovem *hello*.

Následuje další příklad, kterým stáhneme jednu konkordanci se slovesem *think* v libovolném tvaru s použitím automatické lemmatizace a určování slovních druhů. Uvádíme také logovací výpisy.

```
ConcordanceCrawler think -p 'V.*' -n 1
```

```
2016-04-30 14:52:53,220 STATUS: ConcordanceCrawler version 1.0.0
   started, press
Ctrl+C for  interrupt
2016-04-30 14:52:53,811 DETAILS: crawled SERP, parsed 50 links
2016-04-30 14:52:53,812 STATUS: Crawling status
serp         1 (0 errors)
links crawled   50 (0 filtered because of format suffix, 0
   crawled repeatedly)
pages visited   0 (0 filtered by encoding filter, 0 filtered by
   language filter,
0 errors)
concordances    0 (0 crawled repeatedly)
2016-04-30 14:53:04,387 ERROR: 'HTTPConnectionPool(host='
   spknclothing.com',
port=80): Read timed out. (read timeout=10)' occured during
   getting
http://spknclothing.com/about/
2016-04-30 14:53:09,411 DETAILS: page http://spknclothing.com/
   visited,
1 concordances found
2016-04-30 14:53:09,411 STATUS: Crawling status
serp         1 (0 errors)
links crawled   50 (0 filtered because of format suffix, 0
   crawled repeatedly)
pages visited   1 (0 filtered by encoding filter, 0 filtered by
   language filter,
1 errors)
concordances    0 (0 crawled repeatedly)
2016-04-30 14:53:09,411 STATUS: Crawling status
```

```
serp        1 (0 errors)
links crawled   50 (0 filtered because of format suffix, 0
   crawled repeatedly)
pages visited   1 (0 filtered by encoding filter, 0 filtered by
   language filter,
1 errors)
concordances    1 (0 crawled repeatedly)
[
  {
    "date": "2016-04-30 14:53:09.411287",
    "concordance": "not everyone thinks about BMX when they think
        of Maine but these guys have a great scene...",
    "url": "http://spknclothing.com/",
    "id": 1,
    "keyword": "think",
    "end": 19,
    "start": 13
  }
]
```

Stejný výstup bez použití automatické lemmatizace a určování slovních druhů bychom mohli získat také příkazem

```
ConcordanceCrawler think thinks thinking thought -n 1
```

Další příklady vstupů a výstupů aplikace ConcordanceCrawler jsou v příloze A.

4.2 Programátorská dokumentace

V této sekci uvádíme programátorskou dokumentaci naší aplikace.

4.2.1 Použitý programovací jazyk a nástroje

Při implementaci jsme použili programovací jazyk Python. Existují dvě hlavní verze tohoto jazyka, Python 2 a Python 3. Přitom na světě existují programátoři, kteří upřednostňují verzi 2, a jiní, kteří upřednostňují verzi 3. Podle průzkumu z roku 2014, jehož výsledky byly zveřejněny na Python Wiki (2014), asi 78 % respondentů z řad programátorů v Pythonu používá více Python 2 než Python 3. Novější průzkum k dispozici nemáme. Přitom se už nepokračuje ve vývoji Pythonu 2, ale jen verze 3. Dá se čekat, že verze 3 se bude v budoucnu používat častěji. Proto jsme se rozhodli vytvořit naši aplikaci tak, aby ji bylo možno používat s oběma verzemi Pythonu. Konkrétně podporujeme verze 2.7, 3.2, 3.3, 3.4 a 3.5.

Ve zdrojových kódech jsme používali konstrukce, které budou fungovat ve všech verzích. Také jsme při výběru každé externí knihovny přihlíželi i k tomu, zda je kompatibilní se všemi námi podporovanými verzemi Pythonu.

Nástroje použité při vývoji

Zdrojový kód jsme vyvíjeli ve veřejném gitovém repozitáři na GitHubu na této adrese: `https://github.com/Gldkslfmsd/concordance-crawler`

Ke všem důležitým částem programu jsme průběžně psali a spouštěli unittesty, abychom se ujistili, že nové změny v kódu nezapříčinily nefunkčnost programu.

Repozitář ConcordanceCrawleru jsme propojili se službou Travis.[7] To je nástroj, který po každém pushi do repozitáře automaticky nainstaluje aplikaci, spustí ji a provede unittesty. To všechno dělá na všech verzích Pythonu, které podporujeme. Tímto jsme se průběžně ujišťovali, že naše aplikace stále funguje ve všech verzích Pythonu.

Rovněž používáme automatický deployment. Pokaždé, když uděláme několik změn v kódu aplikace, vydáme novou verzi programu a automaticky ji vystavíme na PyPI, takže je ihned k dispozici uživatelům.

Issue tracker

Používáme také githubí issue tracker,[8] což je nástroj, přes který může kdokoliv z uživatelů aplikace reportovat domnělé chyby, dávat návrhy na nové funkce, pokládat otázky, navrhovat úpravy zdrojového kódu atd. Všechno v issue trackeru je veřejné, takže se může zabránit tomu, aby byla jedna věc probírána zbytečně vícekrát nezávisle na sobě.

4.2.2 Popis částí programu

Celý zdrojový kód aplikace a knihovny tvoří balíček `ConcordanceCrawler`[9]. Ten obsahuje další tři balíčky, `app`, ve kterém sídlí kód pro konzolovou aplikaci. V jeho modulu `app.py` je hlavní metoda, která se volá po zavolání aplikace přes příkazový řádek, zadání a parsování všech argumentů spuštění stahování. Dále tam jsou třídy zajišťující formátování výstupu ve formátech XML nebo JSON a funkce pro načtení dat při restartu aplikace.

Dalším balíčkem je `tests` obsahující unittesty všech hlavních částí programu.

Třetím balíčkem je `core`. Ten obsahuje kódy tvořící hlavní funkcionalitu aplikace a knihovny. Níže jej popíšeme důkladněji.

Balíček core

Balíček core obsahuje moduly pro stahování odkazů, třídu `Visitor` pro navštívení odkazu a extrakci konkordancí z něj a spoustu modulů pro jeho pomocné funkce, pro extrakci textů z dokumentu, segmentaci na věty, hledání konkordancí atd. Obsahuje také balíček `eng_detect` pro detekci anglických dokumentů. Ten popíšeme důkladně v další kapitole.

Třídy

Hlavní třídou, která zajišťuje stahování konkordancí, je `ConcordanceCrawler`. Tato třída stahuje odkazy z vyhledávače Bing, každý odkaz navštíví a extrahuje z něj konkordance.

[7] `https://travis-ci.org/`

[8] `https://github.com/Gldkslfmsd/concordance-crawler/issues`

[9] Balíček je v Pythonu složka obsahující další balíčky či moduly. Moduly jsou samostatné textové soubory obsahující zdrojový kód programu.

Ve výchozím nastavení stahuje konkordance v angličtině, používá na ně filtr angličtiny. Obsahuje ale také metodu `setup`, kterou se dají vyměnit funkce ConcordanceCrawleru podle přání uživatele knihovny.

Třída `ConcordanceCrawler` se nijak nestará o ošetřování chyb a výjimek. Na to je její potomek `LoggingCrawler`, který zajišťuje logování postupu stahování a ošetřování výjimek. Výjimky a chyby vzniklé při extrakci dokumentu se řeší tak, že se zaloguje, že došlo k výjimce, a pokračuje se s dalším odkazem.

Programátoři využívající ConcordanceCrawler mají na výběr, zda budou používat `ConcordanceCrawler` bez ošetřování chyb, či `LoggingCrawler` s ignorováním chyb a logováním postupu stahování.

Konzolová aplikace ConcordanceCrawleru používá potomka `LoggingCrawleru`, který má název `EnglishLoggingCrawler`.

Pro navštívení odkazů a extrakci konkordancí se používá třída `Visitor`. Extrakci konkordancí z dokumentu popisujeme důkladně v samostatné kapitole číslo 5.

Stahování dokumentů

Pro stahování dokumentů z Internetu používáme knihovnu `requests`. Stahování některých dokumentů trvalo velmi dlouho, a to až několik hodin, když například HTML dokument byl velmi dlouhý. Takové odkazy mohou celou extrakci výrazně a zbytečně zpomalit. Není jich ale mnoho, asi jen 0.3 %[10] ze všech stažených odkazů, proto nevadí, když je vynecháme.

Proto jsme se rozhodli použít knihovnu `stopit`. Ta vytvoří nové vlákno, ve kterém měří čas vykonávání bloku kódu, v němž se stahuje dokument přes knihovnu `requests`, a když to překročí stanovený limit, tak vykonávání přeruší a vyhodí výjimku. Délku stahování každého dokumentu jsme proto omezili na 60 sekund, což je pro naprostou většinu dokumentů dostatečné a zpracovávání extrakčního úkolu to příliš nezpomaluje.

Získávání odkazů

Odkazy získáváme z vyhledávače Bing, a to tak, že stáhneme HTML dokument SERP z URL, do které zadáme hledané klíčové slovo. Ze stažené SERP potom extrahujeme odkazy. Na to používáme kód, který jsme převzali z GoogleScraperu (Tschacher, 2015). Je v modulu `ConcordanceCrawler.core.parsing.py`. Z toho kódu jsme pouze vymazali pro nás nepotřebné části.

Pro získávání většího počtu možných odkazů používáme generátory bazwords. Implementovali jsme čtyři tyto generátory. Jsou to třídy, které při každém zavolání metody `get_bazword` vrátí jedno bazword. Jsou popsány v části 3.2.2.

Získané odkazy před navštívením filtrujeme. Uživatel knihovny má možnost vytvořit si vlastní filtr na odkazy. V konzolové aplikaci vynecháme ty odkazy, které končí na .pdf, .iso, .doc, .ppt a tak podobně, protože tyto koncovky mají obvykle dokumenty v jiných než plaintextových formátech. Na ty se naše aplikace nespecializuje, podrobněji je to popsáno v sekci 5.2.

[10]Toto číslo pochází z extrakčního úkolu *have* popsaného v tabulce 6.1.

Zajištění unikátnosti konkordancí

Naše aplikace musí být schopna zajistit, že nestáhne z různých dokumentů stejnou konkordanci vícekrát a stejně tak nenavštěvuje vícekrát stejný odkaz, protože předpokládáme, že obsah odkazů se v čase příliš nemění, takže tam najdeme stejný obsah se stejnými konkordancemi, takže by to bylo zbytečné. To děláme tak, že každý zpracovaný odkaz a každou nalezenou konkordanci si uložíme do paměti do datové struktury, které říkáme `LimitedBuffer`. Používáme dvě instance této třídy, jednu na konkordance a druhou na odkazy. Pokud objekt už v bufferu máme, tak odkaz nenavštívíme nebo konkordanci nevypisujeme. Přitom dotaz na existenci prvku v bufferu, stejně jako vložení prvku, je implementován tak, aby měl konstantní časovou složitost. Proto uvnitř obsahuje hashovací množinu a frontu.

Buffer je ale omezený na 1 milion položek, aby nám nedošla paměť. Pokud každá konkordance a každý odkaz má průměrně 200 znaků, pak v nejhorším případě můžou oba buffery zabírat nanejvýš asi 500 MB, což se vejde do paměti každého běžně používaného počítače.

Když je buffer plný, tak se při ukládání nového objektu smaže ten objekt, který je tam nejdéle. Jelikož si uchováváme pouze omezený počet objektů v paměti, může se stát, že potom, co je objekt z bufferu vytlačen, se do něj dostane znovu a tím se dostane stejná konkordance podruhé na výstup. Na druhou stranu zjištění, zda už byla konkordance stažena, se děje velmi rychle a nezpomaluje to extrakční úkol.

Velikost bufferu může určit uživatel na vstupu přepínačem `--buffer-size BUFFER_SIZE`, takže si může vybrat sám, zda upřednostňuje jistotu, že konkordance budou unikátní, nebo jistotu, že aplikaci nikdy nedojde paměť.

Předčasné ukončení stahování

Je důležité, aby naše aplikace skončila v případě, že nastane nějaký problém znemožňující stahování konkordancí, protože jinak by zbytečně zatěžovala počítač, na kterém je spuštěna, nebo by mohla generovat zbytečně velký počet dotazů na vyhledávač a tím ho donutit k blokaci.

Aplikace si proto počítá počty chyb při zpracovávání SERP, počty stažených SERP a počty stažených konkordancí. Pokud je počet chyb při zpracovávání SERP větší než 10, aplikace skončí, protože buďto nemá přístup k Internetu, nebo vyhledávač nefunguje.

Pokud už aplikace zpracovala více než 10 SERP a stále ještě nestáhla ani jednu konkordanci, rovněž skončí, protože zpracovávání zadaného extrakčního úkol se zdá být neperspektivní. Může to být z toho důvodu, že se třeba nedají najít žádné odkazy na dokumenty, které by to obsahovaly zadané slovo, uživatel zadal neexistující značku slovního druhu, neexistující kódování apod.

Na každé SERP se obvykle nachází 50 odkazů, aplikace zvládne zpracovat průměrně asi 2 112 dokumentů za hodinu[11]. To znamená, že o tom, zda je extrakční úkol perpektivní či nikoliv, se obvykle rozhodne asi za 14 minut.

[11] Toto zjištění popisujeme v kapitole 6

4.3 Dokumentace knihovny

Podrobná dokumentace knihovny je k dispozici na následujícím odkazu:

```
https://github.com/Gldkslfmsd/concordance-crawler/wiki/Customizing-Concordance
Crawler
```

Je napsána v angličtině. Je to popis, jak použít `LoggingCrawler` ve vlastním kódu a popis všech funkcí, které se dají nastavit metodou `setup` včetně popisu jejich vstupů a očekávaných výstupů.

Její součástí je i ukázkový zdrojový kód, kterým se dají extrahovat české konkordance.

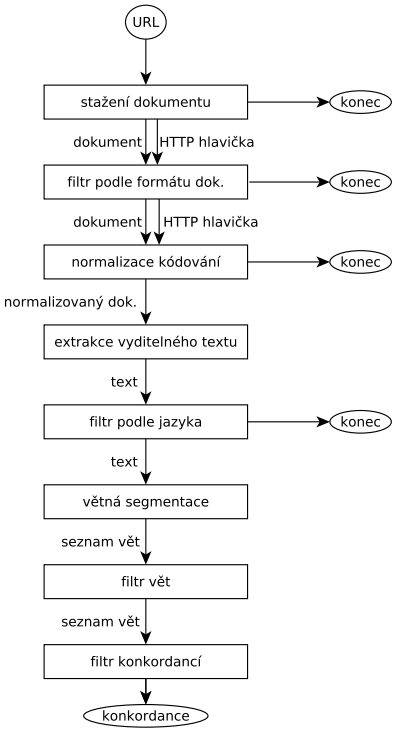
5. Extrakce konkordancí z dokumentu

V první sekci této kapitoly stručně popisujeme celý algoritmus extrakce konkordancí z dokumentů, v dalších částech se pak zabýváme jednotlivými částmi, a to formátem dokumentů v sekci 5.2, kódováním v sekci 5.3 a extrakcí samotného textu v části 5.4.

Sekce 5.5 je věnována určení jazyka dokumentu. Popisujeme několik existujících metod, naši vlastní implementaci a její evaluaci. V další sekci 5.6 popisujeme větnou segmentaci a v sekci 5.7 hledání konkordancí v seznamu vět s pomocí automatické lemmatizace a určování slovních druhů.

5.1 Algoritmus extrakce

Na obrázku 5.1 popisujeme algoritmus extrakce konkordancí z dokumentu. Metoda pro extrakci bere na vstupu URL dokumentu. Na výstupu vrací seznam konkordancí, které se v dokumentu nacházejí. Pokud se dokument nepodařilo stáhnout, nebyl v požadovaném formátu, byl v neznámém kódování nebo nebyl v angličtině, pak metoda nevrací seznam konkordancí, ale `None`.



Obrázek 5.1: Algoritmus extrakce.

Stahování dokumentů jsme popsali už v předcházející kapitole, všechny ostatní části s výjimkou filtru vět popisujeme v samostatných sekcích této kapitoly.

Aplikace má přepínač `-e ENCODING, --encoding ENCODING`. Pokud jako `ENCODING` uživatel zadá ASCII, pak aplikace nehledá konkordance ve větách obsa-

hujících některý znak, který se nenachází v ASCII. Filtrování vět podle ASCII se děje na místě vyznačeném v obrázku. V ostatních nastaveních se neděje nic, filtrem projdou všechny věty.

5.2 Formát dokumentů

ConcordanceCrawler se specializuje pouze na extrakci z plaintextových[1] dokumentů. Dokumentů v ostatních formátech není mnoho, mezi staženými odkazy jich je kolem 6 %.[2] Navíc extrahovat správně text z každé neplaintextového formátu je poměrně náročné, protože každý ten formát bychom museli zpracovávat jiným způsobem, mohlo by to být jak náročné na implementaci, tak i na výpočetní výkon. Úsilí na to vynaložené by se velmi pravděpodobně nevyplatilo, zřejmě by vedlo pouze k malému zvýšení rychlosti zpracovávání extrakčních úkolů.

Přesné určení formátu dokumentu je také poměrně obtížný úkol. Informaci o formátu dokumentu můžeme získat třeba z přípony URL odkazu, na které se dokument nachází. Některé takové odkazy končí na `.pdf`, `.ppt`, `.doc`, `.docx`, `.avi` atp., pak je pravděpodobné, ale ne jisté, že se na něm bude nacházet neplaintextový dokument. V naší konzolové aplikaci takové odkazy filtrujeme a ani nenavštěvujeme. Ve zdrojovém kódu je seznam sedmi takovýchto přípon.

I tak se ale může stát, že neplaintextový dokument nesídlí na takovémto odkazu. Další informací o formátu je hodnota `Content-Type` v hlavičce HTTP odpovědi. Nepokračujeme s extrakcí u dokumentů, které mají uvedenou jinou hodnotu než text.

Také toto rozlišování nemusí být stoprocentně dokonalé, servery ho nemusí dodržovat. Přesnější určení jsme se ale rozhodli neimplementovat. I kdyby se stalo, že netextový binární dokument projde dále, je pravděpodobné, že bude obsahovat binární data a v nich se nenajde žádný výskyt cílového slova, takže se v něm konkordance nenajde a proto nemůže projít na výstup nekvalitní konkordance.

Dále zahazujeme příliš velké dokumenty. Pokud jsou tak dlouhé, že se za 60 sekund nestihnou stáhnout, tak se zahodí už během stahování. Pokud jsou ale větší než 20 MB, tak je rovněž přeskočíme. To proto, že jsme zjistili, že knihovny, které používá ConcordanceCrawler pro extrakci vyditelného textu či určení slovních druhů, mají s dlouhými dokumenty problém, potřebovaly příliš mnoho času a paměti. Toto omezení vylepší postup stahování, ale přitom příliš neomezuje zdroj dokumentů. Je jich totiž velmi málo. Během stahování dlouhých úkolů popsaných v tabulce 6.1 jich bylo asi jen 0.05%.

5.3 Kódování dokumentů

Pro zpracovávání výsledného korpusu je nutné, aby pokud možno všechny stažené konkordance byly v jednom kódování, jinak by byly konkordance nečitelné. Právě proto je v knihovně ConcordanceCrawler místo pro funkci, která všechny dokumenty převede na stejné zadané kódování nebo je úplně zahodí. Pro převod kódování je ale nutné znát výchozí kódování, jinak se může proběhnout špatně.

[1] `https://en.wikipedia.org/wiki/Plain_text`

[2] Toto číslo pochází z výsledku dlouhého extrakčního úkolu *have* popsaného v tabulce 6.1.

Můžeme se zkusit spolehnout na informaci o kódování, které uvádí autor dokumentu, případně server, který ten dokument poskytuje. Tento údaj se dá získat z hlavičky HTTP odpovědi. Pokud je dokument v HTML, pak někdy má tento údaj v tagu `meta`.

Pokud bychom uznali, že údaj od autora je nepřesný, mohli bychom si stáhnout nějaké reálné dokumenty, najít vlastnosti, podle kterých se různá kódování odlišují, a rozlišovat je podle toho. Mohli bychom také použít nějakou metodu strojového učení. To je ale poměrně náročné, proto jsme nejprve provedli experiment, během kterého jsme se snažili zjisit, jak přesné bývají metaúdaje o kódování u dokumentů, které ConcordanceCrawler stahuje. Přitom jsme se zaměřili pouze na ty dokumenty, které prošly filtrem angličtiny, protože ostatní bychom stejně vynechali.

5.3.1 Experiment ověřování metainformací o kódování

Stáhli jsme 84 dokumentů, které byly staženy ConcordanceCrawlerem a prošly filtrem neanglických dokumentů, a podívali jsme se, jaké hodnoty o kódování uvádějí. Výsledky jsou uvedeny v tabulce 5.1.

Snažili jsme se zjistit, v jakém kódování jsou ty dokumenty ve skutečnosti, ovšem není možné to zjistit s naprostou jistotou, protože nevíme, jaké úmysly měli jejich autoři. Nicméně unixový příkaz `file` o všech z nich tvrdil, že vypadají, jako kdyby byly v kódování UTF-8 nebo ASCII. Tento příkaz podle své manuálové stránky (Darwin, 1986 – 1999) se snaží uhodnout kódování podle různých rozsahů a sekvencí bytů tvořících tisknutelné znaky v různých kódováních. Všechny dokumenty jsme také zběžně prohlédli v textovém editoru a neviděli jsme, že by se v nich vyskytovaly netisknutelné znaky. Nemáme tedy důvod nevěřit, že by nebyly v těchto kódováních.

Průzkum Web Technology Surveys

Podle průzkumu zveřejněného na W^{3}Techs.com (2016), který se každý den opakuje, aby uváděl nejnovější výsledky, je na Internetu ze všech webových stránek 87 % v kódování UTF-8, jen 6 % v ISO-8859-1 a zbývající kódování jsou zastoupena pouze nepatrně.

Podíl dokumentů v UTF-8 zjištěný naším průzkumem se tedy shoduje s tímto zdrojem. V podílu dokumentů v ISO-8859-1 se náš údaj mírně liší, to ale může být způsobeno tím, že jsme zkoumali jen ty dokumenty, které prošly filtrem angličtiny, a také tím, že jsme zkoumali pouze 84 dokumentů a ne víc a protože jsme nevybrali reprezentativní vzorek ze všech webových stránek na Internetu, ale jen ty, které nám nabídl vyhledávač Bing. Najít reprezentativní vzorek ale ani nebylo naším cílem.

5.3.2 Vybraný způsob implementace

Rozhodli jsme se, že implementujeme zajištění uniformity kódování způsobem, při kterém nebudeme žádné dokumenty převádět na jiná kódování, ale pouze vynechávat nevyhovující dokumenty. Domníváme se ale, že to naše uživatele příliš neomezí a naopak to bude mít příznivý výsledek na kvalitu korpusu, protože takto nemůžeme udělat chybu při převodu kódování.

hlavička	metatag	`file`	souhlasí	počet
ISO-8859-1	ISO-8859-1	UTF-8	ne	1
ISO-8859-1	UTF-8	ASCII	ne	1
ISO-8859-1	—	UTF-8	ne	1
—	—	ASCII	ne	4
—	—	UTF-8	ne	1
—	GB2312	UTF-8	ne	1
—	ISO-8859-1	UTF-8	ne	1
—	UTF-8	ASCII	ano	6
—	UTF-8	UTF-8	ano	6
UTF-8	—	ASCII	ano[2]	3
UTF-8	—	UTF-8	ano	6
UTF-8	ISO-8859-1	ASCII	ne	1
UTF-8	UTF-8	data	ano[1]	1
UTF-8	UTF-8	ASCII	ano[2]	14
UTF-8	UTF-8	UTF-8	ano	37

[1] podle ručního ověření to byl obyčejný textový HTML dokument
[2] znaková sada ASCII je podmnožinou UTF-8

uváděné údaje		
GB2312	1	1 %
—	5	6 %
ISO-8859-1	3	4 %
UTF-8	73	87 %
nesouhlasící údaje	2	2 %
celkem	84	

dokumenty s platnými uvedenými údaji		
ověřeno UTF-8	73	95 %
neplatné údaje	4	5 %
celkem	77	

Tabulka 5.1: Údaje o kódování v hlavičce HTTP odpovědi a v metatagu stažených dokumentů

Uživatele necháváme vybrat si kódování přepínačem `-e ENC`, `--encoding ENC`, přičemž jsme implementovali tři možnosti, jak s uživatelovým výběrem naložit.

ASCII

Pokud uživatel zadá ASCII, pak jsou extrahovány všechny dokumenty a jsou vyfiltrovány ty věty, které obsahují nějaký znak, který se nenachází v ASCII. To je výhodné, protože jak UTF-8, tak ISO-8859-1 používá stejnou základní sadu znaků jako ASCII, mezi nimi je 128 znaků z anglické abecedy, základních spe-

ciálních znaků jako závorky, interpunkce, mezery atd. Tuto sadu obě kódování jenom rozšiřují o další znaky, a to každé trochu jiným způsobem. Nemusí nás tedy zajímat, v jakém kódování je celý dokumentu, ale pokud jedna konkordance v dokumentu je složena pouze ze základních znaků ASCII, tak ji nemusíme nijak konvertovat a můžeme ji použít.

Určené kódování

Pokud uživatel zadá nějaké určené kódování, například UTF-8, pak budou filtrovány všechny dokumenty, které nejsou v tomto kódování, respektive ty, které to o sobě neuvádějí.

Neurčené kódování

Pokud uživatel neurčí žádné kódování, tak potom budou vyhledávány konkordance ze všech dokumentů bez ohledu na kódování. Potom bude extrakce rychlejší, protože se nebudou kvůli kódování vynechávat žádné dokumenty, ale může to mít vliv na zhoršenou kvalitu korpusu.

5.4 Extrakce viditelného textu

Většina stažených dokumentů je ve formátu HTML, což je značkovací jazyk, ve kterém se prolíná text dokumentu s formátovacími značkami (tzv. tagy) pro webový prohlížeč. Tyto značky je potřeba oddělit od textu dokumentu, protože ty se čtenářům textu nezobrazují, ty nejsou součástí textů.

Proto ConcordanceCrawler odstraňuje všechny HTML tagy a komentáře z HTML. Je nutno zmínit, že tímto postupem nedostaneme vždy stejný text, který se lidskému uživateli zobrazí, když si přes svůj webový prohlížeč zobrazí HTML dokument. V prohlížeči totiž stáhne s HTML zároveň i javascript a ten může zobrazovaný obsah změnit. To ConcordanceCrawler pro jednoduchost nedělá.

Pro odstraňování tagů z HTML používá ConcordanceCrawler knihovnu `BeautifulSoup`.

5.5 Určení jazyka dokumentu

Pro naši konzolovou aplikaci jsme hledali knihovnu v Pythonu, která by byla schopná rozhodnout, zda je text v angličtině nebo v jiném jazyce. Žádnou knihovnu, která by byla určena přesně pro tento úkol, jsme nenašli, našli jsme pouze několik knihoven na rozpoznání více jazyků. Ty jsme podrobněji prozkoumali.

5.5.1 Knihovny v Pythonu na určení jazyka

Prozkoumali jsme čtyři v Pythonu na určení jazyka.

`ldig`

Použití knihovny `ldig`[3] jsme rovnou zamítli, protože jsme nenašli její dokumentaci a protože není určena pro libovolné texty, ale pouze pro krátké zprávy, takzvané *tweety*.

Knihovna `nltk`

Jednou možností na určení jazyka by bylo použít knihovnu `nltk` (Bird a kol., 2001-2008) podle návodu na webu `http://www.algorithm.co.il` (Goldberg, 2012). Určení jazyka by fungovalo tak, že se spočítají takzvaná *stopslova*, což jsou slova, která se v jednom jazyce obvykle často vyskytují a přitom nesou žádný nebo jen malý syntaktický význam (viz Wikipedia, 2016). Pokud by v textu bylo více anglických stopslov než neanglických, tak by byl text považován za anglický.

Knihovna `nltk` poskytuje korpus 2400 těchto slov v 11 jazycích (viz Bird a kol., kapitola 2, tabulka 1.2). To znamená, že tato metoda by mohla dobře fungovat pouze pro těch 11 jazyků, my bychom ale chtěli univerzální metodu, která rozezná angličtinu vedle jakéhokoliv jiného jazyka.

`langid`

Dále jsme vyzkoušeli knihovnu `langid` (Lui, 2011). To je poměrně velká knihovna používající `numpy`, což je další poměrně velká knihovna, jenom během stažení má 15 MB. Poskytuje mnoho funkcí pro vědecké výpočty, které nevyužijeme, proto bychom se jí chtěli vyhnout, pokud to bude možné.

`langid` kromě funkcí pro určení jazyka poskytuje i webový server, což je pro naše použití zbytečné. Knihovna se také dlouho načítá, a to asi 3 sekundy, protože zdrojové kódy obsahují velké objekty binárních dat. Navíc se nám nepodařilo zjistit, jakým způsobem funguje.

`langdetect`

Další prozkoumanou knihovnou je `langdetect`, Danilák (2014). Jedná se o přímé převedení knihovny `language-detection` (Shuyo, 2010), která byla napsána v Javě, do Pythonu.

Tato knihovna je určena pro rozpoznávání 50 jazyků. Funguje tak, že má v sobě profily těchto jazyků, což jsou frekvence znakových n-gramů, které vznikly extrakcí z různých jazykových verzí Wikipedie v těchto jazycích. Na tyto příznaky potom aplikuje naivní Bayesův klasifikátor.

Nevýhodou tohoto přístupu je, že zvládne rozlišit pouze těch 50 jazyků, na které je knihovna natrénována. Mohlo by se tedy stát, že nějaký neznámý jazyk bude knihovna považovat za angličtinu. Navíc obsahuje mnoho funkcí, které bychom nepoužili. Proto jsme se rozhodli tuto knihovnu nepoužít.

[3] `https://github.com/shuyo/ldig`

5.5.2 Vlastní implementace rozpoznání angličtiny

Nakonec jsme se rozhodli implementovat vlastní funkci pro rozpoznání angličtiny. Pojmenovali jsme ji `eng_detect`.

Je známo, že frekvence užitích jednotlivých písmen v textech v jednom jazyce (a tedy i v angličtině) není stejná, ale některá písmena se používají častěji a některá méně často. Tyto relativní frekvence jsou ale ve všech velmi dlouhých textech v jednom jazyce velmi podobné. To samé neplatí jen pro písmena, ale i pro dvojice a trojice následujících písmen, neboli bigramy a trigramy.

Různé jazyky se těmito frekvencemi odlišují, protože třeba v angličtině se často používá *the*, v němčině zase *sch*. Na tomto jsme založili náš algoritmus na rozpoznávání angličtiny.

Algoritmus rozpoznávání angličtiny

Nejprve vezmeme dlouhý text, u kterého víme jistě, že je v angličtině, a spočítáme z něj *vektor frekvencí unigramů, bigramů a trigramů* $\mathbf{X}$. Je-li S abeceda a $T \in S^*$ text délky l, pak vektor frekvencí unigramů, bigramů a trigramů definujeme jako

$$\forall n \in \{1,2,3\}\ \forall a_1, \ldots, a_n \in S$$

$$X_{a_1 \ldots a_n} := \frac{|\{1 \leq i \leq l-n-1 : T_i \ldots T_{i+n-1} = a_1 \ldots a_n\}|}{l-n-1}$$

Položky vektoru jsou tedy všechny možné 1,2,3-gramy, hodnoty vektoru jsou od 0 do 1 a znamenají podíl daného n-gramu ze všech n-gramů.

U textu, který máme rozpoznat, spočítáme také stejný vektor.

Poté spočítáme podobnost obou vektorů. K tomu nám poslouží kosínová vzdálenost.[4] Máme-li nenulové vektory $\mathbf{A}$, $\mathbf{B}$, pak kosinus jejich úhlu je roven

$$\cos\theta = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \cdot \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

přičemž kosinus toho úhlu je 1, pokud jsou vektory rovnoběžné, a 0, pokud jsou kolmé. Pokud jsou texty shodné nebo jsou ve stejném jazyce, pak mají stejný nebo velmi podobný vektor frekvencí 1,2,3-gramů a jejich kosínová vzdálenost se blíží 1. Vektory jsou téměř nebo úplně rovnoběžné. Pokud jsou texty naprosto odlišné, například když jsou z jazyků, které nepoužívají stejné písmo, jeden je v azbuce a druhý v latince, pak žádný 1,2,3-gram se nenachází v obou textech a jejich kosínová vzálenost je 0. Vektory frekvencí 1,2,3-gramů jsou kolmé.

Ve skutečnosti málokdy bude anglický text z dokumentu mít shodný profil frekvencí 1,2,3-gramů s naším referenčním vektorem. Musíme tedy určit hranici, od které budeme text považovat za anglický. Stanovíme ji podle validačních textů během implementace a zapíšeme do zdrojového kódu.

[4] `https://en.wikipedia.org/wiki/Cosine_similarity`

Zpracování dat

Jako referenční text jsme použili korpus textů pocházejících z anglické Wikipedie. Délka těchto textů činila 3.4 GB. Pro extrakci n-gramů jsme napsali skript v Pythonu, který korpus zpracovával paralelně na všech dostupných procesorech. Spouštěli jsme ho na 40-jádrovém serveru, díky čemuž extrakce trvala pouhých 8 minut.

Při konstrukci vektoru 1,2,3-gramů referenčího nebo detekovaného textu postupujeme následujícím způsobem:

- text převádíme na malá písmena, aby naše metoda fungovala i na textech pouze v malých či pouze ve velkých písmenech a abychom zahustili data.
- slova obsahující znak, který není v anglické abecedě, tedy například písmena s diakritikou, číslice, tečky, zavináč apod. jsme z referenčního textu úplně vyřadili, protože slova, která je obsahují, pravděpodobně nejsou anglická. Takto se nám nemohly do vektoru dostat n-gramy netypické pro angličtinu.

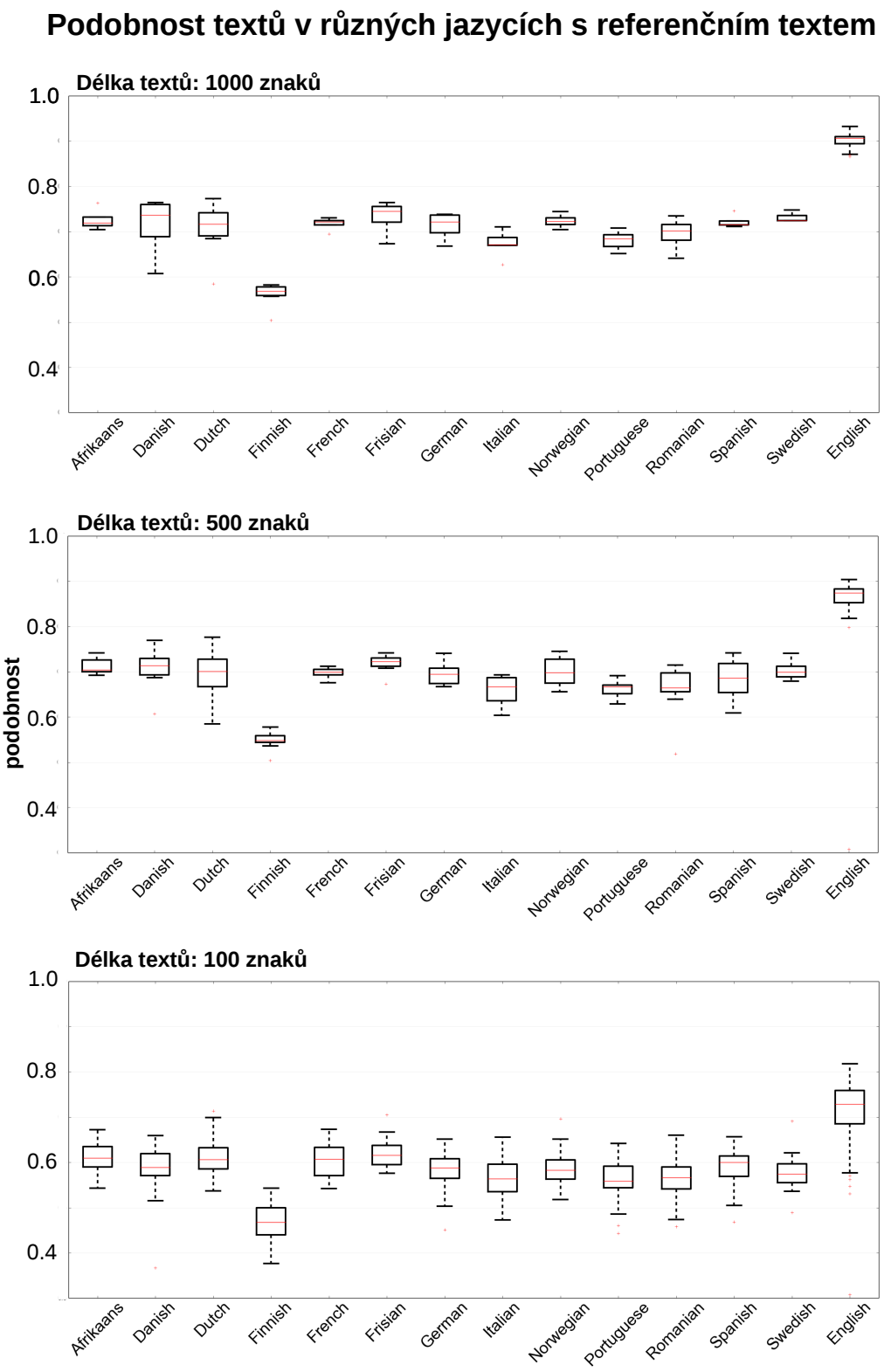
  V detekovaných textech ale neanglické znaky necháváme. Mohlo by se totiž stát, že detekujeme ruský text, který obsahuje odstavec v angličtině. Kdybychom ruské znaky vyřadili, ztratí se informace o tom, že tam byly, a text by byl považován za anglický.
- do n-gramů jsme zahrnuli také mezery, takže například bigramy slova *the* jsou `' t'`, `'th'`, `'he'`, `'e '`. To proto, aby se nám neztratila informace o délce slov, ta je u rozlišování jazyků také užitečná.
- interpunkční znaménka kromě mezer jsme úplně vyřadili, protože jejich používání není ve všech anglických textech jednotné. Například se používají různé unicodové znaky pro uvozovky a apostrofy.
- v Pythonu implementujeme vektor jako slovník, jehož klíče jsou řetězce označující n-gramy. Ty do slovníku vkládáme pouze v případě, že jejich frekvence je nenulová. Tímto způsobem šetříme paměť.

Stanovení prahu podobnosti

Pořídili jsme si validační texty v angličtině a různých dalších jazycích, které jsme našli na Internetu, většinou na zpravodajských portálech a různých jazykových verzích Wikipedie. Vybrali jsme hlavně texty v germánských a románských jazycích, protože ty by měly být angličtině nejpodobnější ze všech. Tímto náš model natrénujeme pro ten nejhorší případ. Přibrali jsme ještě finštinu, ostatní jazyky jsme na tomto místě vynechali, je ale zřejmé, že metoda bude dobře fungovat i na nich, protože jsou angličtině méně podobné.

Spočítali jsme kosínovou vzdálenost vektorů těchto textů od referenčního vektoru, a to pro různě dlouhé texty, protože délka textu ovlivňuje podobnost s referenčním textem. V delších textech je totiž větší pravděpodobnost, že se tam objeví i méně časté n-gramy.

Výsledky uvádíme v grafu na obrázku 5.2. Graf je standartně používaný boxplot.[5] Anglické texty jsou úplně vpravo. Opravdu je vidět, že čím delší jsou



Obrázek 5.2: Graf podobnosti textů v různých jazycích s referenčním textem. Texty mají délku 1000, 500 a 100 znaků.

texty, tím větší je rozestup v podobnosti s referenčním vektorem mezi anglickými a neanglickými texty. A mimochodem je také vidět, že germánské jazyky jsou angličtině o něco bližší než románské. Nejpodobnější ze všech jí je fríština, což souhlasí s tvrzeními jazykovědců.[6]

Dlouhé anglické a neanglické texty tedy půjdou velmi dobře oddělit podle jednoho kritéria, podobnosti s referenčním anglickým textem. Tuto hranici stanovujeme zvlášť pro různé délky textů. Děláme to tak, že vezmeme několik dalších anglických textů, rozdělíme je na úryvky zadaných délek a od každého úryvku spočítáme podobnost. Z těchto čísel pak spočítáme interquartile range (IQR[7]), což je vzdálenost 1. a 3. kvartilu,[8] a jako hranici stanovíme hodnotu 1. kvartilu mínus 1.5-krát IQR.

5.5.3 Test přesnosti metody `eng_detect` na reálných dokumentech

Pro stanovení přesnosti naší metody na určení jazyka jsme naší konzolovou aplikací stáhli a uložili několik desítek dokumentů a extrahovali z nich viditelný text. Všechny dokumenty jsme ručně rozdělili na anglické a neanglické. Poté jsme je nechali detekovat naší metodou `eng_detect`.

Výsledek uvádíme v tabulce 5.2. Metoda má precision 95 %, to znamená, že 95 % dokumentů, které označí jako anglické, jsou skutečně anglické. Vzhledem k tomu, že upřednostňujeme kvalitu před kvantitou je metoda `eng_detect` vhodná pro použití v aplikaci ConcordanceCrawler.

Je dokument v angličtině?			
		podle `eng_detect`	
		ano	**ne**
skutečnost	**ano**	39 43 %	8 9 %
	ne	2 2 %	41 46 %
		accuracy:	89%
		precision:	95%

Tabulka 5.2: Výsledky testu metody `eng_detect` na reálných dokumentech. Charakteristiky *accuracy* a *precision* jsou vysvětleny na `https://en.wikipedia.org/wiki/Evaluation_of_binary_classifiers`.

[5] `https://en.wikipedia.org/wiki/Box_plot`
[6] `https://en.wikipedia.org/wiki/English_language#Classification`
[7] `https://en.wikipedia.org/wiki/Interquartile_range`
[8] `https://en.wikipedia.org/wiki/Quartile`

5.6 Větná segmentace

Další důležitou částí ConcordanceCrawleru je segmentace textu na věty. Věty se obvykle dají oddělovat podle teček, vykřičníků a otazníků. Kdybychom vždy rozdělili větu, kdykoliv narazíme na některý z těchto znaků, tak v mnohých případech větu rozdělíme nesprávně, tečky se totiž v angličtině píšou i u zkratek (třeba *e.g.*), v URL, je potřeba také rozlišovat tři tečky, tečky v desetinných číslech, v IP adresách a datumech (jako například *6.5.2016*) a v mnohých dalších situacích, kde neznamenají konec věty.

Kdybychom rozdělovali věty tam, kde se v textu nachází tečka (či vykřičník a otazník) a za ní mezera, tak se nám zase může stát, že větu rozdělíme nesprávně tam, kde je nějaká zkratka. Navíc nerozdělíme ty věty, v kterých autor neudělal mezeru za tečkou na konci věty. Takovéto texty se na Internetu také nacházejí.

Opět jsme hledali knihovny či jiné způsoby, jak v Pythonu tento úkol zvládnout. Vyzkoušeli jsme jeden regulární výraz a knihovny `nltk` a `segtok`.

Segmentace pomocí regulárního výrazu

Dá se sestrojit prakticky libovolně složitý regulární výraz, který bude dělit text na věty. Vyzkoušeli jsme tento výraz, který jsme převzali z `http://stackoverflow.com/q/25735644`. Na jemu odpovídajících znacích má dojít k rozdělení.

```
(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?)\s
```

Tento regulární výraz je schopen správně rozdělit větu na tečce či otazníku, po kterém následuje mezera, ale přitom hlídá i další pravidla.

Pokud je za větou více teček či otazníků, rozděluje až za posledním z nich, dále nerozděluje dvoupísmenné zkratky jako *M.D.*

Na druhou stranu toto například nezná zkratku *etc.* a podobné a neporadí si s vynechanými mezerami na konci věty ani s vykřičníky. Dá se ale téměř libovolně rozšířit i pro další pravidla, to je ale poměrně náročné na implementaci.

Segmentace pomocí `nltk`

Knihovna `nltk` poskytuje funkci `sent_tokenize` na větnou segmentaci. Funguje tak, že dělí věty podle jakýchkoliv teček, a to i když jsou součástí zkratky, což je často velmi nepřesné (viz Davis, 2012). Toto tvrzení jsme testovali a potvrdili při srovnání segmentačních metod, které popisujeme v sekci 5.6.

Segmentace pomocí `segtok`

`segtok` je knihovna pro pro segmentaci na věty a na slova pro románské a germánské jazyky, mimo jiné i pro angličtinu. Její autor ji velmi dobře popisuje v článku na svém blogu, viz Leitner (2015).

Tato knihovna odděluje věty na všech druzích teček, vykřičníků a otazníků, které se nacházejí v Unicodu. Odděluje věty pouze tam, kde je tečka následována mezerou a další věta začíná velkým písmenem. Dále je schopna správně pracovat s číslovanými odrážkami používajícími tečku a citáty v uvozovkách.

Srovnání segmentačních metod a finální výběr

Provedli jsme srovnání zmíněných tří metod, a to tak, že jsme napsali anglický text, na kterém by se měly ukázat rozdíly ve fungování segmenterů. Obsahoval různé zmíněné jevy, s kterými se není snadné při segmentaci vypořádat. Potom jsme spustili segmentaci na všech třech metodách a do tabulky jsme zapsali, zda metody umí s těmi jevy pracovat správně. Záznamy o testu jsou k nahlédnutí v elektronické příloze této práce.

Na základě tohoto srovnání jsme se rozhodli vybrat `segtok`. Dále jsme ještě měřili rychlost zpracovávání vzorového textu, přitom se `segtok` ukázal jako nejrychlejší. Trvalo mu to 0.174 milisekundy, regulárnímu výrazu 0.328 milisekundy a `nltk` 3.28 milisekundy. Dalším důvodem pro výběr `segtoku` je, že umí pracovat se všemi druhy interpunkčních znamének v Unicodu.

Během testování implementace jsme zjistili, že na výstupu naší konzolové aplikace se vyskytuje poměrně velký počet vět, které jsou špatně rozděleny. Stává se to v případě, že jedna věta končí tečkou a za ní bez mezery následuje další věta. Proto jsme se rozhodli, že na text rozdělený `segtok`em aplikujeme ještě jedno další pravidlo. Podíváme se na každé slovo, které ve větě obsahuje tečku (nebo otazník či vykřičník). Pokud nalevo i napravo od toho interpunkčního znaménka je nějaké existující anglické slovo, rozdělíme to na tom místě na věty. V opačném případě to necháme být. Pro to určení, zda se jedná o anglické slovo, jsme do vzorového kódu vložili slovník anglických slov, který jsme převzali z `nltk`.

5.7 Hledání konkordancí

Chceme, aby náš ConcordanceCrawler hledal v dokumentech všechny morfologické tvary uživatelem zadaného slova podle zadaného slovního druhu. Na to je potřeba automatické určování slovních druhů (tomu se říká také *part-of-speech tagging*) a také lemmatizace neboli určení lemmatu, slovníkového tvaru každého slova. Například lemma slovesa *flies* je *fly* (letět), naše aplikace by ale neměla najít konkordance s podstatným slovem *fly* (moucha).

Hledali jsme proto knihovny pro automatickou lemmatizaci a určování slovních druhů.

Našli jsme dvě. Knihovnu `gensim` (Řehůřek a Sojka, 2010), ta ale nefunguje v Pythonu 3, proto ji nemůžeme použít.

Druhá, kterou jsme našli, je `textblob` (Loria, 2015), což je knihovna, která poskytuje užitečné funkce pro zpracování přirozeného jazyka z knihoven `nltk` (Bird a kol., 2001-2008) a `pattern`. Přínosem `textblobu` je, že poskytuje API, jehož použití je jednodušší než přímé použití těch knihoven. Jinak ale poskytuje naprosto stejné výsledky jako `nltk`.

Nevýhodou `textblobu` je, že vyžaduje instalaci `nltk`, což je poměrně velká knihovna a není příliš jednoduché ji nainstalovat. Jelikož ale jinou alternativu nemáme, rozhodli jsme se, že `textblob` se nebude instalovat současně s ConcordanceCrawlerem, ale vytvoříme přepínač pro zadání tagu slovního druhu. Když uživatel použije tento přepínač a nebude mít nainstalovaný `textblob`, tak naše aplikace skončí s chybou a vypíše zprávu o tom, že pro požadovanou funkci je potřeba mít nainstalovánu knihovny `textblob`. Vypíše také příkaz, kterým se dá

nainstalovat. Teprve potom bude uživatel moci používat automatickou lemmatizaci a part-of-speech tagging.

6. Výsledky testování aplikace

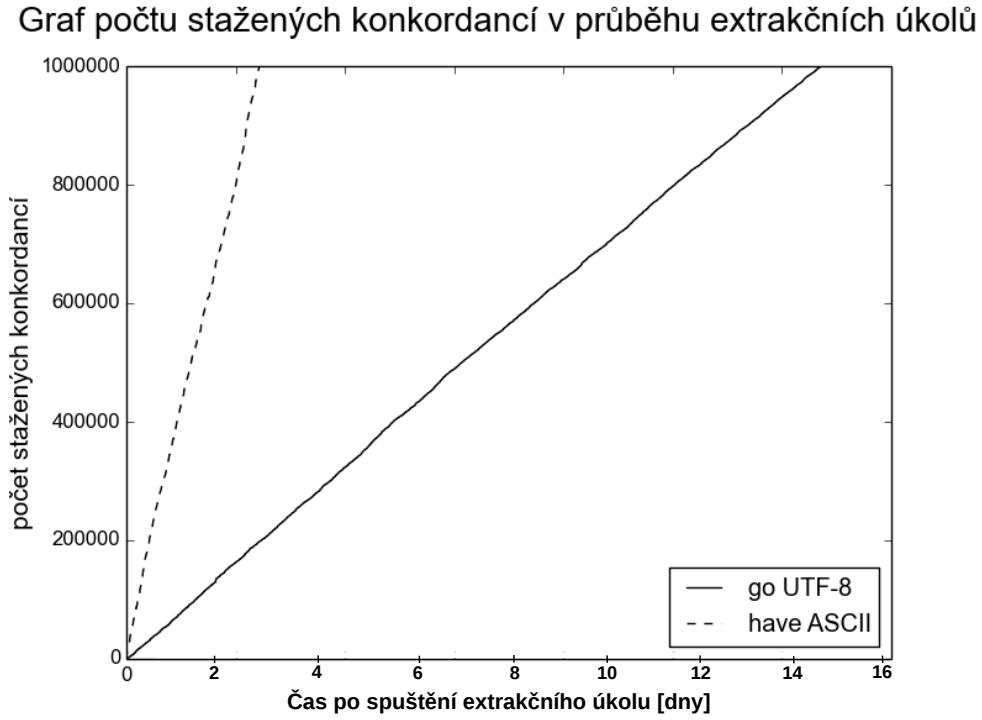
V této kapitole uvedeme výsledky testování naší aplikace. V sekci 6.1 jsme zjišťovali to, zda naše aplikace nevyčerpá zdroj konkordancí a zda je schopna běžet bez chyby donekonečna a stále stahovat konkordance. V sekci 6.2 jsme měřili rychlost extrakce na různých slovech a v poslední sekci 6.3 jsme testovali kvalitu získaných konkordancí.

6.1 Vyčerpatelnost zdroje konkordancí

Spustili jsme naši aplikaci několikrát tak, aby stáhla 1 milion konkordancí s různými slovy.

Stahovali jsme konkordance anglických sloves *have* a *go*. V tabulce 6.1 uvádíme výsledky těchto extrakčních úkolů, a to celkový počet stažených SERP, odkazů na dokumenty, počet stažených dokumentů, výsledný počet konkordancí a další údaje. Extrakční úkoly běžely bez přerušení několik dní. V případě slovesa *have* se nám podařilo stáhnout 1 milion konkordancí za necelé 4 dny a u slovesa *go* stejný počet asi za 15 dní.

Na obrázku 6.1 je graf stažených konkordancí v závislosti na čase po spuštění extrakčního úkolu. Je vidět, že tento počet u všech úkolů roste rovnoměrně a téměř lineárně s časem a že v době ukončení extrakce nebyl zdroj konkordancí vyčerpán ani nezačal slábnout. Domníváme se tedy, že Internet je jako zdroj konkordancí alespoň pro častá slova téměř nevyčerpatelný a dá se z něj získat alespoň několik milionů konkordancí.



Obrázek 6.1: Graf počtu stažených konkordancí v průběhu vybraných dlouhotrvajících extrakčních úkolů.

Nastavení		
cílové slovo	**have**	**go**
slovní druh	sloveso	sloveso
kódování	ASCII	ASCII
generátor bazwords	RANDOM	RANDOM
verze Pythonu	3.4	2.7
Výsledný počet konkordancí		
konkordancí	1 000 000	1 000 000
počet opakovaně stažených konk.	119 421 (12 %)	103 709 (10 %)
Odkazy		
staženo SERP	7 527	64 718
staženo odkazů	119 071	1 050 095
odkazů přeskočených kvůli příponě	7 151 (6 %)	65 929 (6 %)
opakovaně stažených odkazů	18 589 (16 %)	298 727 (28 %)
Stažené dokumenty		
navštíveno odkazů	90 366	661 081
dok. filtrovaných kvůli kódování	20 (0.02 %)	273 (0.04 %)
dok. filtrovaných kvůli jazyku	30 335 (33.6 %)*	453 920 (69 %)
počet chyb při stahování dokumentů	2 954 (3 %)	24 355 (4 %)
Rychlost stahování		
celková délka stahování	67.5 hod (4 dny)	353 hodin (15 dní)
průměrný počet konk. za hodinu	14 815	2 833
navštíveno odkazů za hodinu	1 339	1 873
průměrně SERP za hodinu	112	183
prům. konk. z jednoho odkazu	11	1.5

Pozn: Zde ještě nebyla použita finální verze filtru angličtiny, ale méně přesná verze s nižším prahem. Tato hodnota se tedy nedá srovnávat s ostatními.

Tabulka 6.1: Tabulka výsledků dlouhotrvajících extrakčních úkolů.

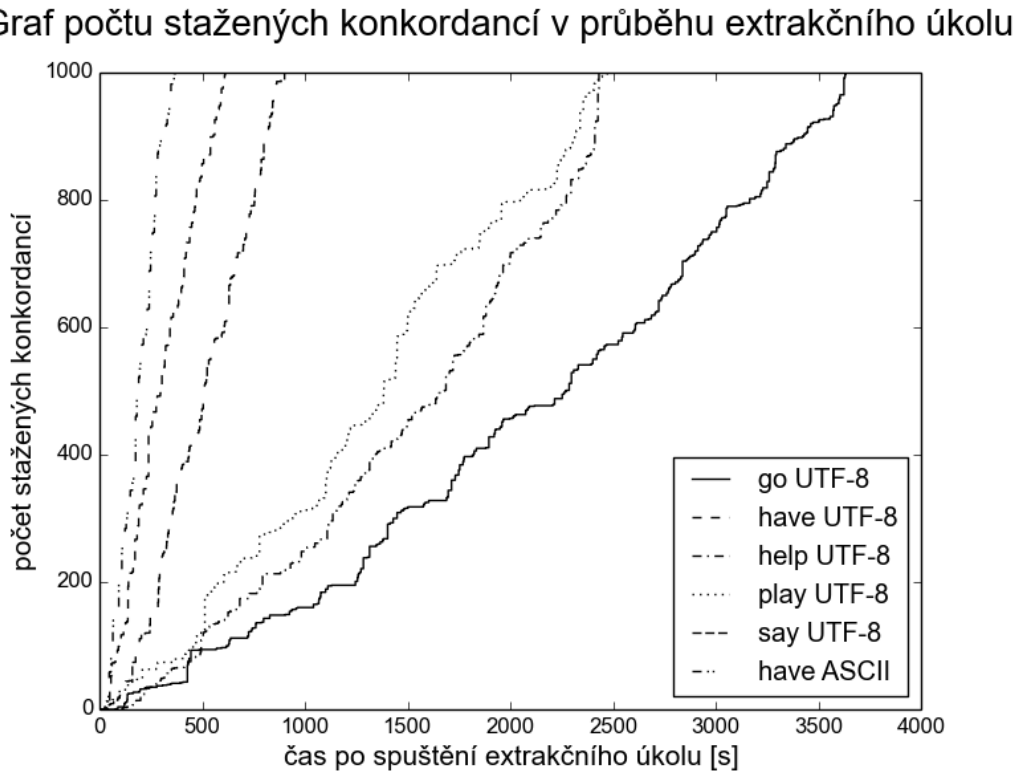
6.2 Rychlost stahování konkordancí

V tabulce 6.2 jsou výsledky vybraných extrakčních úkolů pro slovesa *go, have, help, play* a *say*, které jsme stahovali v kódování UTF-8 a ještě jednou *have* v kódování ASCII. Cílem bylo stáhnout 1 000 konkordancí těchto sloves. Na obrázku 6.2 je graf ukazující průběžný počet stáhnutých konkordancí při zpracovávání uvedených extrakčních úkolů. Je vidět, že rychlost stahování konkordancí závisí hlavně na zvoleném slově.

Dále rychlost stahování závisí na zvoleném kódování. Stahování v ASCII u těchto slov bylo o něco rychlejší či zhruba stejně rychlé. To proto, že v ASCII není třeba vynechávat tolik dokumentů kvůli kódování.

Mezi uvedenými slovy se nejrychleji stahovalo *have* v ASCII a potom *have* v UTF-8. Po nich následovalo *say*, dále shodně *play* a *help* a nakonec *go*.

Rozdíl v rychlostech stáhnutí 1 000 konkordancí těchto sloves byl téměř devítinásobný, nejrychlejší stahování trvalo 7 minut, nejpomalejší 61 minut.



Obrázek 6.2: Graf počtu stažených konkordancí v průběhu vybraných krátkých extrakčních úkolů.

Nastavení						
cílové slovo	**go**	**have**	**help**	**play**	**say**	**have**
slovní druh	sloveso	sloveso	sloveso	sloveso	sloveso	sloveso
kódování	UTF-8	UTF-8	UTF-8	UTF-8	UTF-8	ASCII
generátor bazwords	čísla	čísla	čísla	čísla	čísla	čísla
verze Pythonu	3.4	3.4	3.4	3.4	3.4	3.4
Výsledný počet konkordancí						
konkordancí	1 000	1 000	1 000	1 000	1 000	1 000
opakovaně	49	22	25	52	17	12
Odkazy						
staženo SERP	52	10	37	43	15	7
staženo odkazů	2 643	524	1 898	2 200	732	368
z toho opakovaně	733	87	435	697	217	61
Stažené dokumenty						
navštíveno	1 854	418	1 363	1 411	481	275
filtr kódování	302	119	220	174	67	0
v procentech	16 %	28 %	16 %	12 %	14 %	0 %
filtr jazyka	1 136	189	785	933	238	172
v procentech	61 %	45 %	58 %	66 %	49 %	63 %
počet chyb	33	3	31	34	5	3
v procentech	2 %	1 %	2 %	2 %	1 %	1 %
Rychlost stahování						
délka stahování	61 min	10 min	41 min	41 min	15 min	7 min
prům. SERP za h.	51	60	54	63	60	60
prům. konk. za hod	984	6 000	1 463	1 463	4 000	8 571
prům. dok. za hod	1 824	2 508	1 995	2 065	1 924	2 357

Tabulka 6.2: Tabulka výsledků krátkých extrakčních úkolů.

Průměrná rychlost stahování

V následující tabulce 6.3 uvádíme průměrnou rychlost stahování konkordancí za hodinu, průměrnou rychlost zpracovávání dokumentů a průměrnou dobu zpracování jedné SERP. Hodnoty vznikly jako průměr ze všech extrakčních úkolů v tabulkách 6.1 a 6.2, pouze do průměrného počtu konkordancí za hodinu jsme nezahrnuli extrakční úkol *have* z tabulky 6.1, protože ten ještě používal starší verzi filtru angličtiny a proto je tato hodnota nesrovnávatelná s ostatními.

průměrně konkordancí za hodinu	3 407
průměrně dokumentů za hodinu	1 986
průměrně SERP za hodinu	80

Tabulka 6.3: Průměrná rychlost stahování konkordancí z krátkých i dlouhých extrakčních úkolů.

6.3 Kvalita získaných konkordancí

Určit kvalitu konkordancí je poměrně náročný úkol, protože se musí provést ručně, neexistuje na to žádný automatický nástroj.

Specifikovali jsme čtyři kritéria, která musí splňovat kvalitní konkordance.

1. Konkordance musí být *jedna věta*. Musí být právě jedna a musí být celá. Slova ve větě musí být oddělena mezerami.

   Za *jednu větu* považujeme posloupnost slov, která sdělují jednu ucelenou myšlenku. Například v souvislém textu, jakým je třeba předchozí odstavec, jsou tři věty, každá začíná velkým písmenem a končí tečkou a mezerou. Za větu ale považujeme také jednu položku v menu či v obsahu, každý nadpis, titulek apod. Větou je tedy také „Kvalita získaných konkordancí“ v nadpise této sekce.

2. Konkordance musí být v angličtině.

3. V konkordanci nesmí být *neplatné kódování*.

   Předpokládejme, že používáme běžný textový editor, například Vim, Gedit nebo PSPad, ten máme nastaven na kódování, ve kterém má být soubor konkordancí. Potom za konkordance s rozbitým kódováním považujeme ty věty, kdy se ve větě zobrazuje znak, který textovým editorem nelze zobrazit nebo z kontextu věty vidíme, že tam má být nějaký unicodový znak, ale místo něj je jiný, který tam evidentně nepatří (třeba písmeno *a* se zvláštním diakritickým znaménkem).

   Teoreticky není možné určit, kdy je kódování skutečně neplatné a kdy autor zamýšlel použít takový zvláštní znak, my takovéto situace ale vždy považujeme za chybu.

4. Konkordance musí obsahovat cílové slovo použité jako zadaný slovní druh.

   Pokud hledáme například sloveso *play*, pak nehledáme věty, které obsahují slovo *play*, které je použito jako název nějakého produktu. Stejně tak když

hledáme sloveso *walk* (*chodit, procházet se*), pak nechceme výskyty podstatného jména *walk*, které znamená *procházka*.

6.3.1 Test kvality konkordancí

Cílem našeho testu bylo určit poměr kvalitních konkordancí mezi všemi konkordancemi, které se mohou objevit na výstupu ConcordanceCrawleru.

Ke stanovení průběhu testu nás vedly tyto skutečnosti:

- Kvalita konkordancí závisí na zvoleném cílovém slově. Naše aplikace totiž umí poměrně dobře dělit na věty souvislý text, často ale selhává na rozdělení různých položek menu, seznamů a nadpisů, protože ty obvykle nekončí tečkou.

  Některá slova se častěji vyskytují v nadpisech a v menu, jiná zase více v souvislém textu. Proto stáhneme a vyhodnotíme konkordance z různých slov, ne jen z jednoho, a výslednou kvalitu zprůměrujeme.

- Kvalita závisí také na nastaveních naší aplikace. Například zvolíme-li kódování ASCII, tak pak prakticky nemůže nastat chyba v kódování. Proto je nutné, aby všechna stahování měla toto stejné nastavení.

- Zvolili jsme kódování ASCII. Pro každé kódování bychom mohli určit kvalitu zvlášť.

- Kvalitu stažených konkordancí musíme určit ručně. Proto musíme omezit počet konkordancí, které budeme kontrolovat, protože nejsme schopni jich projít příliš mnoho. Zkontrolovaných konkordancí ale nesmí být příliš málo, aby náš test byl vypovídající.

  Rozhodli jsme se, že zkontrolujeme po 20 konkordancích od 8 slov, celkem jich tedy bude 160.

- Mohlo by se stát, že při stahování naše aplikace narazí na dokument, ze kterého se stáhne nezanedbatelný počet nekvalitních konkordancí a zrovna ty vyhodnotíme, přestože z ostatních dokumentů by stáhla kvalitní konkordance. Takto by náš test byl nepřesný.

  Proto stáhneme větší počet konkordancí od každého slova a pak z nich náhodně vybereme vzorek, který budeme ručně kontrolovat. Takto je větší pravděpodobnost, že v našem zkontrolovaném vzorku bude stejný poměr kvalitních konkordancí jako mezi všemi konkordancemi.

  Pro každé slovo jsme stáhli 100 konkordancí.

Počty různých druhů chyb u vybraných zkontrolovaných konkordancí uvádíme v tabulce 6.4. Průměrný poměr nekvalitních konkordancí byl 25 %, rozptyl je 10 %.

Popis častých chyb

Nejvíce chyb nastávalo u větné segmentace, a to u jevů, na které naše metoda pro větnou segmentaci nebyla připravena. Chyby často nastávaly například proto, že se často nedařilo oddělovat nadpisy od vět, protože nadpis nekončil tečkou, případně se neoddělily různé položky z menu. Jedna taková konkordance je na obrázku 6.3.

Další zdroj chyb byl podobný jako například u slova *president*, protože poměrně hodně jeho konkordancí pocházelo z anglické Wikipedie. Po tečce na konci věty tam často ihned bez mezery následovala hranatá závorka, a to zapříčinilo, že věta nebyla správně oddělena. Příklad takové věty je na obrázku 6.4.

slovo	**slovní druh**	**počty druhů chyb****			
		1	**2**	**3**	**4**
a	jakýkoliv*	5	0	0	0
go	sloveso	7	0	0	1
have	sloveso	3	0	0	0
help	sloveso	6	0	0	0
play	sloveso	8	0	0	0
president	pod. jm.	6	0	0	0
run	sloveso	5	0	0	1
the	jakýkoliv*	1	0	0	0
celkem chyb		38	0	0	2
% chyb		24 %	0 %	0 %	1 %

* *Pozn.:* Stahovaly všechny výskyty slov bez ohledu na slovní druh.

** *Pozn.:* 1 – chyby ve větné segmentaci, 2 – chyby v jazyce, 3 – rozbité kódování, 4 – špatný slovní druh. Od každého slova jsme zkontrolovali 20 konkordancí.

Tabulka 6.4: Tabulka výsledků testu kvality konkordancí.

```
{
 "start": 21,
 "concordance": "Track and field Road running Racewalking Racerunning Cross \
 country running Multi-day race Ultramarathon Wheelchair racing Backward \
 running",
 "url": "https://en.wikipedia.org/wiki/Mile_run",
 "date": "2016-05-12 17:21:01.227776",
 "end": 28,
 "id": 49,
 "keyword": "run"
}
```

Obrázek 6.3: Příklad nekvalitní konkordance se slovem *run*. Zde nesprávně proběhla větná segmentace, jedná se o několik položek z menu, které nejsou rozděleny do samostatných vět.

```
{
 "start": 19,
 "concordance": "While historically presidents initiated the process for\
 going to war,[25][26] critics have charged that there have been several\
 conflicts in which presidents did not get official declarations, including\
 Theodore Roosevelt's military move into Panama in 1903,[25] the Korean\
 War,[25] the Vietnam War,[25] and the invasions of Grenada in 1983[27] and\
 Panama in 1990.[28] Along with the armed forces, the president also directs\
 U.S. foreign policy.",
 "url": "https://en.wikipedia.org/wiki/President_of_the_United_States",
 "date": "2016-05-12 17:20:15.357587",
 "end": 29,
 "id": 30,
 "keyword": "president"
},
```

Obrázek 6.4: Příklad nekvalitní konkordance se slovem *president*, u které neproběhla správně větná segmentace. Jedná se o dvě věty, ta první končí na předposledním řádku.

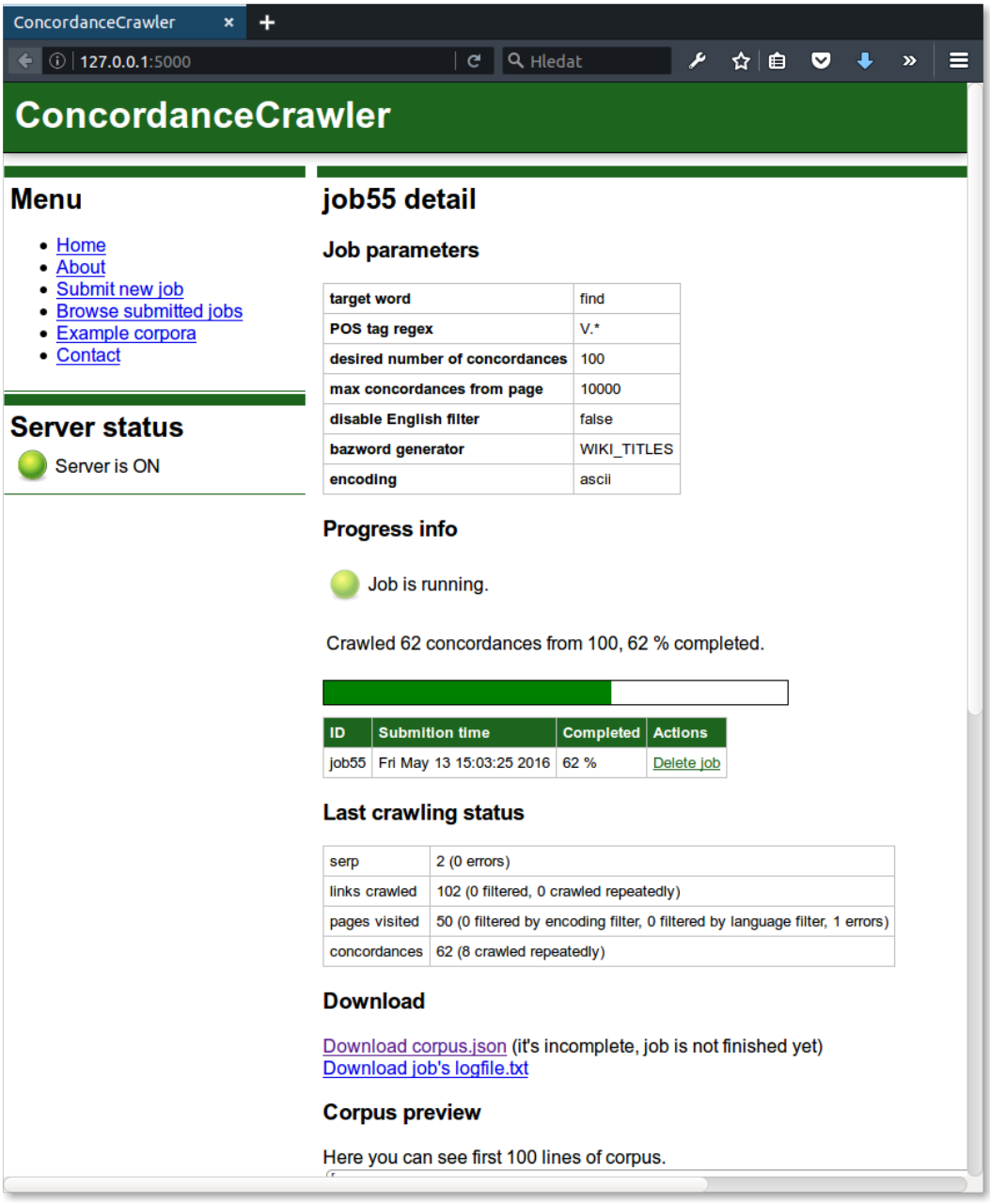
7. Webové rozhraní

Součástí práce bylo vytvořit webové rozhraní, které umožní uživateli zadat extrakční úkol, sledovat stav extrakce a stáhnout výsledný seznam konkordancí.

Webová stránka s tímto rozhraním je umístěna na serveru Ústavu formální a aplikované lingvistiky MFF UK a je dostupná na odkazu `http://quest.ms.mff.cuni.cz:10381/`. Na obrázku 7.1 je screenshot její stránky s detailem extrakčního úkolu.

Stránka je určena pro všechny potenciální uživatele naší aplikace, proto je její obsah v angličtině. Návštěvníci stránky si tu mohou jednoduchou formou vyzkoušet, jak aplikace funguje a jaké dává výsledky, aniž by ji museli instalovat a spouštět na vlastním počítači. Kromě možnosti zadání vlastního extrakčního úkolu jsou zde také odkazy na stažení aplikace, informace o aplikaci v angličtině a je tu také možnost stáhnout několik velkých ukázkových korpusů konkordancí.

Frontend této webové stránky jsme převzali z projektu RExtactor (Kríž a Hladká, 2015).



Obrázek 7.1: Screenshot stránky s detailem extrakčního úkolu z našeho webového rozhraní.

Závěr

V této práci jsme vytvořili ConcordanceCrawler, aplikaci pro automatickou extrakci konkordancí anglických slov z Internetu. Vytvořili jsme také knihovnu, s pomocí které půjde naše aplikace adaptovat pro libovolný jiný jazyk a rozšířit podle specifických nároků jednotlivých uživatelů. Knihovnu i aplikaci jsme zveřejnili jako balíček pod open-source licencí na serveru `pypi.python.org`, takže je přístupná uživatelům po celém světě. K aplikaci i ke knihovně jsme navíc vytvořili přehlednou uživatelskou dokumentaci v angličtině.

Rovněž jsme vytvořili webové rozhraní, přes které mohou uživatelé zadávat extrakční úkoly a sledovat stav extrakce, aniž by naši aplikaci museli sami instalovat a spouštět na svém počítači.

Pro naši aplikaci jsme analyzovali a vybrali vhodný internetový zdroj pro získávání konkordancí anglických slov. Zjistili jsme, že z námi vybraného zdroje se za dostatečně dlouhou dobu dá stáhnout libovolný požadovaný počet konkordancí běžných anglických slov přinejmenším do řádu jednotek milionů, aniž by se zdroj vyčerpal nebo začal slábnout. Také jsme zjistili, že náš zdroj konkordancí nás v čerpání neomezuje.

Také jsme vybrali metodu pro detekci kódování dokumentů. Umožňuje vybírat konkordance pouze v určitém kódování.

Dále jsme navrhli a implementovali vlastní metodu pro detekci anglických dokumentů, která má přesnost (precision) 95 %. Je schopná odlišit anglický text od libovolného jiného jazyka, ne jen z malé omezené množiny jazyků.

Potom jsme vybrali a implementovali metodu pro automatickou segmentaci věty. Pro námi zvolenou definici věty uspěje asi u 75 % konkordancí.

Nakonec jsme vybrali knihovnu pro automatické rozpoznávání slovních druhů, takže naše aplikace dokáže najít konkordance slov, pouze pokud se ve větě vyskytují jako uživatelem zadaný slovní druh.

Dobré vlastnosti ConcordanceCrawleru

Zde shrneme několik dobrých vlastností naší aplikace a knihovny.

- ConcordanceCrawler je robustní. Dokáže běžet i měsíce v kuse, aniž by se zastavil, zasekl nebo skončil s nějakou chybou. Pokud nastane chyba při zpracovávání nějakého dokumentu nebo zpracovávání dokumentu trvá příliš dlouho, zahodí ho a pokračuje s dalším.

- Zároveň je ale ConcordanceCrawler šetrný. Pokud nastane nějaký problém, který znemožňuje stahování dokumentů, ConcordanceCrawler se brzy potom sám ukončí s chybou.

- ConcordanceCrawler umí navázat na předčasně ukončený extrakční úkol.

- Kladem ConcordanceCrawleru jsou také jeho logovací výpisy, díky kterým se dá v přímém přenosu sledovat podrobný průběh extračního úkolu. Tyto výpisy se vypisují na standartní chybový výstup a dá se nastavit jejich „upovídanost“ (verbosity).

- Konzolová aplikace se snadno používá, má pouze jeden povinný parametr, tím je cílové slovo. Ostatní parametry mají defaultní hodnoty, uživatel si ale může nastavit jiné.
- Konzolová aplikace poskytuje podrobnou anglickou nápovědu ke všem přepínačům, existuje k ní také podrobný návod na instalaci. Stejně tak existuje návod na používání ConcordanceCrawleru jako knihovny.
- ConcordanceCrawler je vytvořen v programovacím jazyce Python, je kompatibilní s jeho verzemi 2 i 3.

Možná vylepšení do budoucnosti

Na závěr uvádíme několik nápadů na budoucí vylepšení ConcordanceCrawleru.

- Hlavním nedostatkem ConcordanceCrawleru se ukázala být větná segmentace, která je schopna poměrně dobře dělit na věty souvislý text, ale neporadí si s větami neukončenými tečkou a s texty, které se vymykají pravidlům pravopisu.
- Existuje prostor pro vylepšení metody pro detekci a normalizaci kódování dokumentů a také pro detekci angličtiny.
- ConcordanceCrawler má velké rezervy ve výkonu. Prozatím používá pouze jeden procesor, mohl by ale pracovat paralelně.
- V hlavičce *User-Agent* v HTTP požadavcích by ConcordanceCrawler mohl uvádět, že je robot a kvůli jakému účelu navštěvuje servery. Je ale potřeba zjistit, zda to nebude mít negativní vliv na zdroj dokumentů.
- ConcordanceCrawler by mohl mít funkci detekování „hezkých“ vět. To by byly věty, které začínají velkým písmenem, končí tečkou, jsou smysluplné, obsahují sloveso v určitém tvaru, nejsou příliš krátké ani příliš dlouhé, neobsahují neexistující slova, neukončené závorky apod. Je možné, že by to uživatelé ocenili. Na druhou stranu takový korpus konkordancí nemusí být příliš autentický, protože potom se v něm nemusí objevit některé jazykové jevy.

Seznam použité literatury

Bird, S., Loper, E. a Klein, E. Natural Language Processing with Python — Analyzing Text with the Natural Language Toolkit. `http://www.nltk.org/book/`.

Bird, S., Loper, E. a Klein, E. (2001-2008). Natural Language Toolkit (NLTK). Dostupné online na `http://www.nltk.org/`.

Danilák, M. (2014). langdetect — Port of Google's language-detection library to Python. `https://github.com/Mimino666/langdetect`.

Darwin, I. F. (1986 – 1999). *FILE(1) BSD General Commands Manual*. Berkeley Software Distribution, Toronto, Kanada.

Davis, R. C. (2012). Testing out the nltk sentence tokenizer. `http://www.robincamille.com/2012-02-18-nltk-sentence-tokenizer/`.

Goldberg, I. (2012). Cheap language detection using NLTK. Dostupné online na `http://www.algorithm.co.il/blogs/programming/python/cheap-language-detection-nltk/`.

Hanks, P. (2013). *Lexical Analysis: Norms and Exploitations*. MIT Press.

Holub, M., Kríž, V., Cinková, S. a Bick, E. (2012). Tailored Feature Extraction for Lexical Disambiguation of English Verbs Based on Corpus Pattern Analysis. In Kay, M. a Boitet, C., editors, *Proceedings of the 24th International Conference on Computational Linguistics (Coling 2012)*, pages 1195–1209, Mumbai, India, 2012. IIT Bombay, Coling 2012 Organizing Committee.

Kilgarriff, A. (1995). BNC database and word frequency lists. `https://www.kilgarriff.co.uk/bnc-readme.html#lemmatised`.

Kríž, V. a Hladká, B. (2015). RExtractor: a Robust Information Extractor. In Gerber, M., Havasi, C. a Lacatusu, F., editors, *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 21–25, Stroudsburg, PA, USA, 2015. Association for Computational Linguistics. ISBN 978-1-941643-49-5.

Kríž, V. (2012). Klasifikátor pro sémantické vzory užívání anglických sloves. Diplomová práce. Univerzita Karlova, Matematicko-fyzikální fakulta, Praha. Dostupné online z `https://is.cuni.cz/webapps/zzp/detail/117042/`.

Leitner, F. (2015). segtok — a segmentation and tokenization library. `http://fnl.es/segtok-a-segmentation-and-tokenization-library.html`.

Loria, S. (2015). textblob – Simple, Pythonic text processing. Sentiment analysis, part-of-speech tagging, noun phrase parsing, and more. `http://textblob.readthedocs.io/en/dev/`.

Lui, M. (2011). langid.py – Language Identifier. `https://github.com/saffsd/langid.py`.

Marcus, M. P., Marcinkiewicz, M. A. a Santorini, B. (1993). *Building a Large Annotated Corpus of English: The Penn Treebank*, volume 19. MIT Press, Cambridge, MA, USA. http://dl.acm.org/citation.cfm?id=972470.972475.

Python Wiki (2014). 2.x-vs-3.x-survey. https://wiki.python.org/moin/2.x-vs-3.x-survey.

Řehůřek, R. a Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

Shuyo, N. (2010). Language Detection Library for Java. http://code.google.com/p/language-detection/.

Smarr, J. a Grow, T. (2002). GoogleLing: The Web as a Linguistic Corpus. http://www.web.stanford.edu/class/cs276a/projects/reports/jsmarr-grow.pdf.

Spoustová, J. a Spousta, M. (2012). A High-Quality Web Corpus of Czech. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012)*, pages 311–315.

Tschacher, N. (2015). GoogleScraper — A Python module to scrape several search engines (like Google, Yandex, Bing, Duckduckgo, Baidu and others) by using proxies (socks4/5, http proxy) and with many different IP's, including asynchronous networking support (very fast). . https://github.com/NikolaiT/GoogleScraper.

W^{3}Techs.com (2016). Usage of character encodings for websites. http://w3techs.com/technologies/overview/character_encoding/all.

Wikipedia (2016). Stop words — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Stop%20words&oldid=709047952. [Online; verze z 4. 5. 2016].

Seznam obrázků

Seznam tabulek

A. Ukázkové vstupy a výstupy

V této příloze uvedeme příklad použití ConcordanceCrawleru.

A.1 Spuštění ConcordanceCrawleru

Příkaz „`ConcordanceCrawler a`“ je ukázka nejjednoduššího spuštění ConcordanceCrawleru. Takto nám stáhne 10 konkordancí slova *a* jako libovolný slovní druh a bude je vypisovat na obrazovku ve formátu JSON.

Následujícím příkazem v terminálu spustíme ConcordanceCrawler tak, aby nám stáhl 1 000 konkordancí anglického slovesa *play*. „Upovídanost“ logovacích výspisů nastavíme na nejnižší debugovací úroveň. Jako generátor bazword použijeme přirozená čísla, konkordance budeme ukládat do souboru `play.json`. Formát konkordancí bude json, to je defaultní hodnota, kterou nemusíme určovat přepínačem. Budeme brát jen ty konkordance, které jsou v kódování ASCII. Budeme používat automatické určení slovních druhů, chceme jen konkordance, které obsahují *play* jako sloveso v libovolném tvaru.

```
ConcordanceCrawler -n 1000 play -v 0 -b NUMBERS -o play.json -e
   ASCII -p 'V.*'
```

V aktuálním adresáři se nám objevil soubor `ConcordanceCrawler.backup`. Pokud bychom chtěli přerušit stahování před dokončením a zase ho navázat tak, abychom rozšířili seznam konkordancí v souboru `play.json`, použijeme na to následující příkaz:

```
ConcordanceCrawler --continue-from-backup ConcordanceCrawler.
   backup --extend-corpus play.json
```

A.2 Ukázkový logovací výpis

Po spuštění se nám na terminálu bude vypisovat následující text, který ukazuje aktuální průběh extrakce. Kvůli stručnosti jsme ho zkrátili, celý má 26 227 řádků a je k nahlédnutí v elektronické příloze této práce.

Extrakce trvala 42 minut a skončila úspěšně.

```
2016-05-11 11:08:38,014 STATUS: ConcordanceCrawler version 1.0.1
   started, press Ctrl+C for      interrupt
2016-05-11 11:08:38,026 DEBUG: trying to download SERP http://www
   .bing.com/search?q=0+play&first=1&count=59&FORM=PERE1
2016-05-11 11:08:38,083 STATUS: Starting new HTTP connection (1):
    www.bing.com
2016-05-11 11:08:38,145 DEBUG: Setting read timeout to 10
2016-05-11 11:08:38,279 DEBUG: "GET /search?q=0+play&first=1&
   count=59&FORM=PERE1 HTTP/1.1" 200 None
2016-05-11 11:08:39,163 DETAILS: crawled SERP, parsed 50 links
2016-05-11 11:08:39,163 STATUS: Crawling status
serp            1 (0 errors)
links crawled   50 (0 filtered, 0 crawled repeatedly)
```

```
pages visited   0 (0 filtered by encoding filter, 0 filtered by
   language filter, 0 errors)
concordances    0 (0 crawled repeatedly)
2016-05-11 11:08:39,163 DEBUG: trying to download http://www.play
   .pl/
2016-05-11 11:08:39,191 STATUS: Starting new HTTP connection (1):
    www.play.pl
2016-05-11 11:08:39,280 DEBUG: Setting read timeout to 10
2016-05-11 11:08:39,314 DEBUG: "GET / HTTP/1.1" 200 69721
2016-05-11 11:08:39,701 DEBUG: page rejected by language filter
2016-05-11 11:08:39,701 DETAILS: page http://www.play.pl/ visited
   , 0 concordances found
2016-05-11 11:08:39,701 STATUS: Crawling status
serp            1 (0 errors)
links crawled   50 (0 filtered, 0 crawled repeatedly)
pages visited   1 (0 filtered by encoding filter, 1 filtered by
   language filter, 0 errors)
concordances    0 (0 crawled repeatedly)
2016-05-11 11:08:39,701 DEBUG: trying to download https://
   playframework.com/
2016-05-11 11:08:39,711 STATUS: Starting new HTTPS connection (1)
   : playframework.com

...

2016-05-11 11:50:23,692 DEBUG: trying to download https://www.
   pagat.com/tile/wdom/texas42.html
2016-05-11 11:50:23,708 STATUS: Starting new HTTPS connection (1)
   : www.pagat.com
2016-05-11 11:50:24,264 DEBUG: Setting read timeout to 10
2016-05-11 11:50:24,392 DEBUG: "GET /tile/wdom/texas42.html HTTP
   /1.1" 200 24949
2016-05-11 11:50:26,094 DETAILS: page https://www.pagat.com/tile/
   wdom/texas42.html visited, 27 concordances found
2016-05-11 11:50:26,106 STATUS: Crawling status
serp            43 (0 errors)
links crawled   2197 (10 filtered, 698 crawled repeatedly)
pages visited   1433 (1 filtered by encoding filter, 1063
   filtered by language filter, 31 errors)
concordances    1000 (49 crawled repeatedly)
```

A.3 Ukázkový soubor konkordancí ve formátu JSON

V našem pracovním adresáři nalezneme soubor `play.json`, ve kterém stažené konkordance. Zde uvádíme část toho souboru, celý má 9 002 řádků.

```
[
    {
        "end": 26,
        "id": 1,
        "date": "2016-05-11 11:09:16.690174",
        "start": 22,
        "keyword": "play",
        "url": "http://www.amazon.com/Disney-Infinity-3-0-Pixars-
           Machine-Specific/dp/B00YZ3VA3E",
```

```
        "concordance": "Something parents can play with their
           children."
    },
    {
        "end": 43,
        "id": 2,
        "date": "2016-05-11 11:09:16.690174",
        "start": 36,
        "keyword": "play",
        "url": "http://www.amazon.com/Disney-Infinity-3-0-Pixars-
           Machine-Specific/dp/B00YZ3VA3E",
        "concordance": "In my (admittedly short) experience
           playing Infinity 3.0, this playset is probably the
           best example of that notion."
    },
    {
        "end": 18,
        "id": 3,
        "date": "2016-05-11 11:09:16.690174",
        "start": 12,
        "keyword": "play",
        "url": "http://www.amazon.com/Disney-Infinity-3-0-Pixars-
           Machine-Specific/dp/B00YZ3VA3E",
        "concordance": "If you have played Mario or Crash
           Bandicoot, you know what I'm talking about."
    },
    {
        "end": 103,
        "id": 4,
        "date": "2016-05-11 11:09:16.690174",
        "start": 96,
        "keyword": "play",
        "url": "http://www.amazon.com/Disney-Infinity-3-0-Pixars-
           Machine-Specific/dp/B00YZ3VA3E",
        "concordance": "It was a little different watching my
           daughter play with the anger character along with our
           son playing with Yoda on the same screen."
    },
    {
        "end": 35,
        "id": 5,
        "date": "2016-05-11 11:09:16.690174",
        "start": 28,
        "keyword": "play",
        "url": "http://www.amazon.com/Disney-Infinity-3-0-Pixars-
           Machine-Specific/dp/B00YZ3VA3E",
        "concordance": "I wanted to feel like I was playing the
           characters in Riley's mind and that feeling isn't
           really their."
    },


...

]
```

A.4 Ukázkový soubor konkordancí ve formátu XML

Máme možnost si nechat konkordance uložit také ve formátu XML. Zde je ukázkový soubor konkordancí v tomto formátu:

```
<?xml version="1.0"?>
<concordances>
        <item>
                <concordance>We had struggled with finding a
                   trustworthy cleaning service for our ASC that
                   understood the important role they play in our
                    facility.</concordance>
                <start>114</start>
                <date>2016-05-14 15:13:10.704457</date>
                <end>118</end>
                <url>http://medicleanonline.com/</url>
                <id>1</id>
                <keyword>play</keyword>
        </item>
        <item>
                <concordance>I play the tenor saxophone and the
                   clarinet.</concordance>
                <start>2</start>
                <date>2016-05-14 15:13:13.757215</date>
                <end>6</end>
                <url>https://www.mormon.org/me/CGRN</url>
                <id>2</id>
                <keyword>play</keyword>
        </item>
        <item>
                <concordance>We bake cookies, have lessons about
                   the Savior, work on personal goals, go to the
                   park, go geocaching, play board games or just
                   go out for ice cream.</concordance>
                <start>103</start>
                <date>2016-05-14 15:13:13.757215</date>
                <end>107</end>
                <url>https://www.mormon.org/me/CGRN</url>
                <id>3</id>
                <keyword>play</keyword>
        </item>
        <item>
                <concordance>I&amp;apos;m not sure the kids will
                   remember all the games we played, the places
                   we went or what we did, but I hope they always
                    remember that we loved each other enough to
                   dedicate one night a week to be with each
                   other.</concordance>
                <start>53</start>
                <date>2016-05-14 15:13:13.757215</date>
                <end>59</end>
                <url>https://www.mormon.org/me/CGRN</url>
                <id>4</id>
                <keyword>play</keyword>
        </item>
</concordances>
```

B. Obsah elektronické přílohy

Součástí této práce je elektronická příloha. Zde popisujeme její adresářovou strukturu.

- `concordance-crawler/`
  - `ConcordanceCrawler/` – zdrojový kód aplikace a knihovny
  - `doc/` – dokumentace k ročníkovému projektu a ke knihovně
  - `examples/` – příklad použití knihovny ConcordanceCrawler
  - `research/` – rešerše použité při výběru knihoven pro různé části aplikace
    - `blocking-experiment/` – skripty použité při porovnání vyhledávačů podle míry blokování
    - `encoding/`
    - `lang_detection/`
    - `pos_tagging/`
    - `segmentation/` – zde jsou data o porování segmentačních metod
    - `stopit_demo/`
  - `webdemo-flask/` – soubory webového dema
  - `README.md` – uživatelská dokumentace ConcordanceCrawleru
  - `setup.py` – setup skript projektu ConcordanceCrawler, obsahuje metadata potřebná ke zveřejnění na PyPI
  - `.travis.yml` – konfigurační soubor pro službu Travis
- `img/` – obrázky z této práce
- `text/` – zdrojové soubory textu této práce
- `vystupy/`
  - `go.zip, have.zip` – zazipované soubory s 1 milionem konkordancí ve formátu JSON, pocházejí z extrakčních úkolů popsaných v kapitole 6
  - `play.json, play.out` – celé ukázkové soubory výstupů z přílohy A
- `prace.pdf` – text této práce