Masarykova univerzita
Fakulta informatiky

Automatická detekce struktury webových komunikačních kanálů

Bakalářská práce

Tomáš Bílek

Brno, jaro 2013

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Tomáš Bílek

Vedoucí práce: Mgr. Jan Rygl

Poděkování

Děkuji Mgr. Janu Ryglovi za odborné vedení bakalářské práce a za jeho trpělivost a cenné rady. Dále bych chtěl poděkovat Mgr. Jaroslavu Kupčíkovi za pomoc.

Shrnutí

Práce se zabývá návrhem algoritmu a následným vývojem aplikace pro automatické detekování struktury dat z webových domén. Zaměřuje se na hledání data, nadpisu, autora a článku na českých zpravodajských serverech, blozích a podobně strukturovaných webových zdrojích. K hledání využívá vlastnosti textu a specifické značky ve zdrojovém kódu internetových stránek.

Klíčová slova

webová stránka, web, obsah, crawler, extrakce, detekce struktury, zpravodajský server, dolování textu, boilerpipe, Beautiful Soup

Obsah

1 Úvod

Obsah internetu se neustále mění. Přibývají nové weby, nové informace. Pro výzkumy založené na českém jazyce potřebujeme co nejaktuálnější data, ať už jde o statistiky tvrzení vyslovených politiky, analýzu stylistiky jednotlivých mluvčí a nebo o stavbu korpusu. Člověk sám by nikdy nedokázal zpracovat veškeré nové informace a aktualizace, proto vzniká potřeba pro použití programů a aplikací, které umí získávat a zpracovat informace automaticky. Zde ale narážíme na překážku. Zatímco člověk dokáže přesně rozlišit strukturu webové stránky s informacemi, software s tímto komplexním problémem má potíže. Webové stránky jsou vytvářeny různorodými způsoby. Struktura informace o autorovi na jedné stránce nemusí odpovídat té samé informaci na stránce jiné. Z tohoto důvodu nelze pro tento účel vyvinout obecnou aplikaci. Při užším zaměření na konkrétní typy webů však tento problém řešitelný je. Tato práce se věnuje problému vyhledávání konkrétních informací ze stránek českých zpravodajských domén a blogů a přináší vlastní řešení v podobě aplikace.

1.1 Cíle práce

Bakalářská práce se zabývá problémem automatického vyhledání a určení struktury dat z webových zpravodajských domén a blogů. Tato data jsou použitelná pro stavbu korpusu. Práce se zaměřuje na české zpravodajské weby a blogy, ale navržené postupy jsou obecně použitelné i pro cizojazyčné domény.

Předmětem práce je návrh algoritmů a následné vytvoření aplikace, která analyzuje vybrané dokumenty ze zadané domény a detekuje formát, ve kterém jsou na dané doméně uloženy dokumenty. Aplikace pracuje s dokumenty v českém jazyce a má webové rozhraní pro zadání několika adres domény. Ze zadaných adres se pokusí automaticky zjistit cestu ke článku, nadpisu, datu a času vložení a autora článku. Výstupem aplikace je seznam cest k datům s číslem určujícím přesnost hledaného výsledku. Uživatel má možnost interaktivně ověřit výstup a v případě potřeby změnit cesty dat, která se nepodařilo úspěšně vyhledat.

1.2 Základní pojmy

- URL – zkratka pro Uniform Resource Locator, řetězec znaků, používá se pro přesnou identifikaci dokumentů na internetu. Někdy se pro URL používá označení „webová adresa“.

- Web – označení pro World Wide Web, celosvětová síť hypertextových dokumentů propojených přes internet.

- Tag – element v kódu HTML jazyka. HTML dokumenty jsou tvořeny ze stromu

HTML elementů a každý element může obsahovat vlastnosti (atributy) s hodnotami.

- Stránka – myšleno webová stránka, dokument, který je možné zobrazit prostřednictvím webového prohlížeče.
- Parsování – označení pro proces konverze HTML dokumentu do DOM stromu

1.3 Struktura práce

V první kapitole byl představen úvod do problematiky dolování z textových dat, byly uvedeny cíle této práce a zavedeny související pojmy používané dále v textu práce. Ve druhé kapitole najdeme přehled a popis bádání a nástrojů souvisejících s problematikou dolování z textových dat a důvod vzniku aplikace vyvíjené v rámci této práce. Třetí kapitola čtenáře seznamuje s aktuální situací výzkumu zabývajícího se extrakcí informací, uvádí známé postupy a algoritmy a zdůvodňuje výběr algoritmu na čištění stránek použitého pro vývoj aplikace. Čtvrtá kapitola zahrnuje popis a vývoj vlastního algoritmu umožňujícího detekci struktury dokumentů. V páté kapitole najdeme rozbor implementace aplikace. Kapitola přináší přehled dostupných nástrojů a důvody výběru konkrétního nástroje. Podrobně nás seznamuje s postupem implementace specifických částí a stručně představuje webové rozhraní. Šestá kapitola se věnuje testování aplikace na vybraných zpravodajských serverech a popisuje systém hodnocení. Sedmá a poslední kapitola je závěrem práce a zahrnuje možný budoucí rozvoj aplikace.

2 Související výzkum

V oblasti dolování z textových dat je neutichající zájem o aplikace a algoritmy, které jsou schopné rozeznat části HTML dokumentu představující článek od ostatních stavebních bloků webové stránky jako jsou navigační lišty, hlavičky, reklamy, diskuze, odkazy, obrázky. Můžeme se setkat s různými názvy pro tuto problematiku. Web content extraction, text extraction, boilerplate removal, content block extractor, to všechno jsou názvy používané různými autory pro stejný problém. Tyto aplikace jsou užitečné jak pro sběr dat k okamžitému použití, tak pro ukládání dat pro pozdější potřeby, například pro tvorbu korpusu.

Zájem o podobné algoritmy vznikl už na úplném začátku 21. století [12]. Bohužel, v té době stále ještě mnoho webových stránek vznikalo s použitím Microsoft Frontpage. Většina metod uváděných ve vědeckých pracích z té doby je dnes nepoužitelných kvůli předpokladům a heuristikám, které se nedají přenést na dnešní postupy pro vývoj webu [20].

2.1 Korpus

Český jazyk se neustále vyvíjí. Vznikají nová slova, výrazy. Abychom ale mohli začlenit nové slovo do jazyka, potřebujeme pro něj také vytvořit pravopisná pravidla a výklad významu. K tomuto účelu slouží příručky českého jazyka, encyklopedie, slovníky. Existující české slovníky[1] se bohužel opírají o jazykovou základnu starou přes čtyřicet let, není tedy možné je vydávat za zrcadlo současného jazyka [2].

S rozvojem internetu začaly vznikat soubory počítačově uložených textů určených k jazykovému výzkumu – korpusy. Obsahují rozsáhlý počet textů nasbíraných z různých zdrojů. Nezanedbatelnou část zdrojů zde představuje internet. Korpusy v budoucnosti mohou mimo jiné pomoci k vytvoření moderního slovníku českého jazyka. Budováním korpusů českého jazyka se v České republice zabývá Ústav Českého národního korpusu. Největší korpus nese název Český národní korpus a na jeho vývoji se podílí i Fakulta informatiky Masarykovy univerzity. Centrum zpracování přirozeného jazyka na Fakultě informatiky Masarykovy univerzity se mimo jiné věnuje i vývoji korpusových nástrojů jako je Manatee, Bonito, NoSketch Engine [26].

Při tvorbě specializovaného textového korpusu je potřeba pracovat s textem ve formě článků, zápisů, beletrie. Dostat se k takovému textu je v prostředí dnešního internetu mnohdy nelehký úkol. Internet je plný více či méně relevantních informací. Často se k jedné konkrétní informaci musíme dostat přes množství odkazů a stránek, které nás ve výsledku vůbec nezajímají. V případě tvorby korpusu, kde vyžadujeme obrovské množství takových informací, není možné jednoduše stáhnout celý obsah internetové stránky. Chceme pracovat pouze s podstatným textem, ne s textem reklam, nebo textem

1. Poválečně užívané slovníky jsou jen dva – Slovník spisovného jazyka českého a Slovník spisovné češtiny

navigačních lišt. Odlišení podstatných částí internetové stránky a určení typu informace, kterou tato část obsahuje, představuje nelehký úkol. Aplikace vyvíjená v rámci této práce automatizuje proces získávání informací a určování typu získané informace.

2.2 Stahování dokumentů

2.2.1 Web crawler

Webový prohledávací modul, web crawler (dále jen crawler) je aplikace, která automaticky prohledává internet podle předem určené strategie [18]. Tento proces se označuje jako web crawling a je nejčastěji využívaný ve vyhledávačích. Používají ho všechny známé vyhledávače (Google, Yahoo, MSN a další [23]). Crawler stáhne obsah nových stránek nebo zkontroluje změnu obsahu na dříve indexovaných stránkách. Pracuje se seznamem URL adres, které prochází a sbírá z nich odkazy na další stránky podle určené metodiky. Informace o obsahu pak uloží do databáze, která je přístupná vyhledávači [17]. Mezi nejznámější komerční crawlery patří 80legs[2], Googlebot[3], Bingbot[4]. K dispozici jsou i open-source[5] crawlery jako Abot[6], PHP-Crawler[7], Scrapy[8].

Pro aplikaci vyvíjenou v rámci této práce není potřeba použití crawleru, ale aplikace zpracovává výstup crawleru a je s ním tedy provázaná.

2.3 Obsah stránek

Webové stránky jsou kromě textu složeny z obrázků, tabulek, odkazů, reklam a dalších objektů. Příklad typické webové zpravodajské stránky vidíme na obrázku 2.1. Při budování korpusu chceme pracovat s uceleným textem (celými větami, odstavci). Na většině stránek se tento text nachází na okraji nebo ve středu stránky, obklopený reklamními bannery, flash aplikacemi, navigačními lištami, zápatím a další grafikou. Tyto převládající prvky se souhrnně nazývají boilerplate. Čištěním obsahu stránky rozumíme odstraňování boilerplate a výtahu požadovaného uceleného textového obsahu [4].

2.3.1 Boilerplate

Výrazem boilerplate se označuje jakýkoliv text, který může být znovu použit v novém kontextu nebo aplikaci bez toho, aby se musel nějak výrazně lišit od originálu. Mnoho

2. http://www.alexa.com/siteinfo/80legs.com
3. http://support.google.com/webmasters/bin/answer.py?hl=cs&answer=182072
4. http://en.wikipedia.org/wiki/Bingbot
5. Open-source je software s legálně dostupným zdrojovým kódem a možností úpravy
6. http://code.google.com/p/abot/
7. http://astellar.com/php-crawler/
8. http://scrapy.org/
9. http://www.photoshop.com/, použitý screenshot ze stránky http://www.novinky.cz/domaci/297422-za-chyby-v-cerpani-dotaci-mohou-kraje-i-ministerstvo.html

počítačových programátorů užívá výraz boilerplate code [9]. Používá se, když programátor musí vytvořit velké množství kódu s minimálními požadavky na logiku a funkčnost. V HTML je například používán následující boilerplate jako základní prázdná šablona pro většinu webových stránek[10]:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title></title>
</head>
<body>

</body>
</html>
```

2.4 Extrakce informací

Abychom mohli přesně (nebo aspoň částečně) identifikovat informace, o které máme zájem, nestačí jenom vyčistit stránku od boilerplate. Čistá stránka může obsahovat informace, které jsou sice přínosné a v nějaké formě podstatné, ale nás zajímá třeba jen jedna konkrétní. Pro příklad si můžeme představit internetový zpravodajský deník. Chceme v článku o počasí najít informaci, kolik stupňů Celsia budou dosahovat zítřejší teploty. Po vyčištění od nežádoucích prvků na stránce zbyde jen nadpis, článek a několik málo jiných informací (obvykle o autorovi článku a data vložení). Nás ale nezajímá celý článek, nechceme znát údaje o srážkách nebo bio předpověď, chceme jednu konkrétní informaci. Pro člověka většinou není problém tuto informaci najít. Rychle se zorientuje v textu a najde si konkrétní údaj. Strojové hledání této informace však představuje netriviální úkol, pro který neexistuje obecné řešení. Počítači musíme přesně definovat, který údaj má hledat. Čím přesněji informaci nadefinujeme, tím má počítač větší šanci informaci opravdu najít. V případě hledání zítřejší teploty bychom nejspíš počítači přikázali, ať hledá větu, ve které se vyskytuje slovo „zítra“, možná „zítřejší“, případně datum následujícího dne a poté nejspíš číselnou informaci se znakem °C. I v tomto případě ale počítač může nalézt chybný údaj, nebo nemusí údaj nalézt vůbec (například pokud se v textu místo slova „zítra“ a „zítřejší“ nachází „zítřkem počínaje…“). Druhý problém představuje jazyk. Marně bychom hledali slovo „zítra“ na stránce psané v angličtině. Proto musíme pro každou konkrétní informaci v konkrétním jazyce vytvořit vlastní způsob hledání, resp. algoritmus. Obsah této práce je zaměřen (mimo jiné) na vývoj takových algoritmů.

Obrázek 2.1: Typická webová zpravodajská stránka, vytvořeno v Adobe Photoshop [98]

3 Čištění textu

V této kapitole jsou popsány rozdílné přístupy k čištění webového obsahu a výtahu informací, které je možné aplikovat na webové stránky vytvářené současnými technologiemi.

3.1 Hybridní přístup

Tzv, „A hybrid approach for extracting informative content from web pages“ [28] spočívá v kombinaci dvou různých metod, které se vzájemně doplňují. V první fázi se používá metoda strojového učení: automatizovaný proces, během kterého se vytvoří pravidla pro vloženou množinu dat. Je zde využit Document Object Model (dále jen DOM), obrázek 3.1. DOM je systémově a jazykově nezávislé rozhraní, které umožňuje programům a skriptům dynamicky přistupovat k obsahu stránky a upravovat ho [31]. Pomocí DOM je pak algoritmicky odhalen obsah s informacemi. Ve druhé fázi jsou aplikována pravidla získaná z první fáze a funkcemi pro manipulaci řetězci[1] jsou nalezeny požadované informace. Pokud ovšem druhá fáze nepřinese uspokojivé výsledky, proces se vrací k první fázi a celý se opakuje.

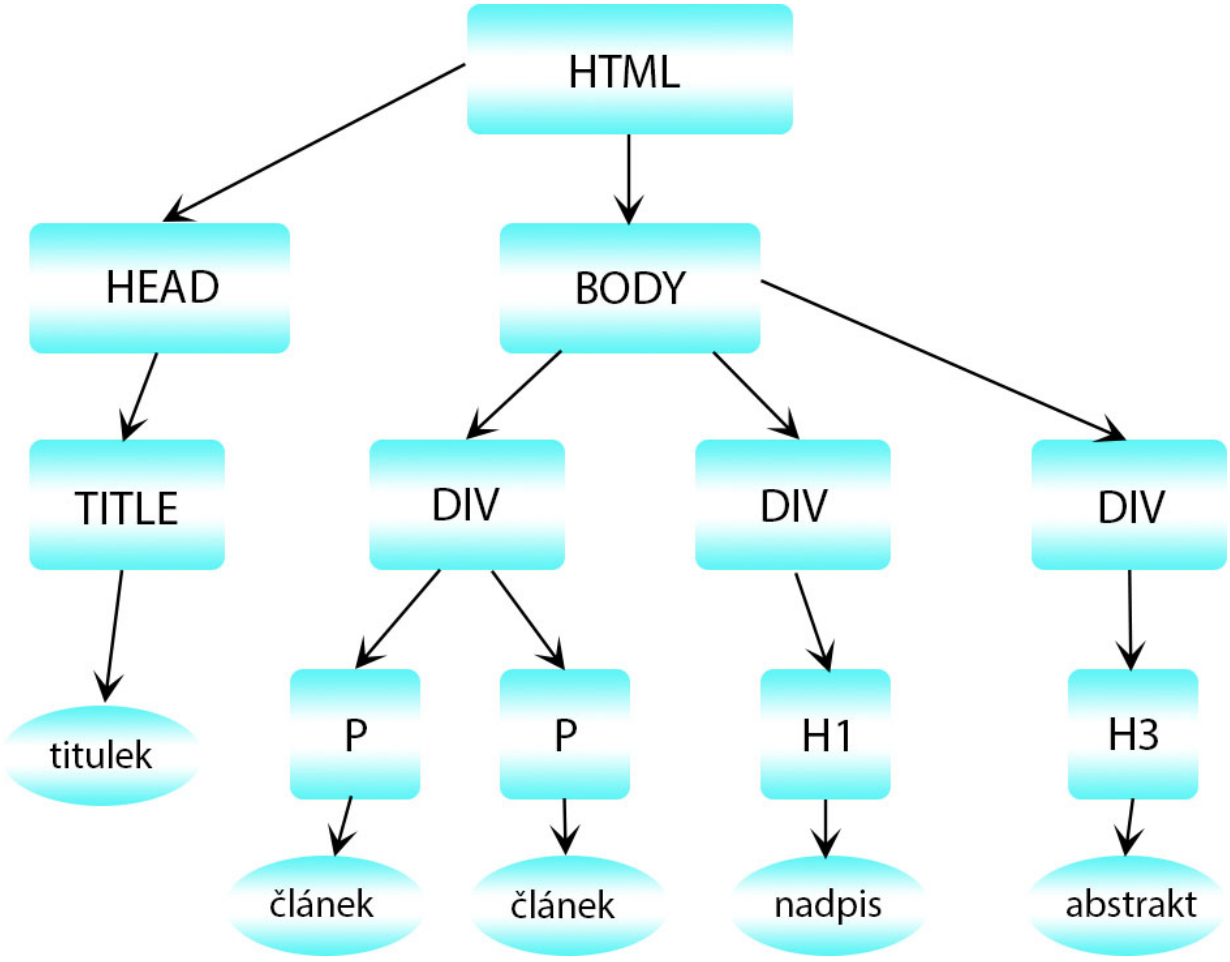
HTML stránky jsou vytvářené hierarchickou strukturou HTML tagů. Vytvoření DOM stromu v prvním kroku je klíčové pro následné procházení struktury stránky. DOM strom umožňuje uložit stránku do paměti a umožňuje přístup k operacím pro manipulaci s prvky stránky. Poté je použito čtyř různých algoritmů určených pro strojové učení (Bayesova naivního klasifikátoru, Bayesovských sítí, algoritmu nejbližších k-sousedů a algoritmu rozhodovacích stromů) na vstupní datové množiny.

Bayesův naivní klasifikátor vychází z předpokladu nezávislosti atributů mezi sebou. Znamená to, že efekt, který má hodnota každého atributu na danou třídu není ovlivněn hodnotami ostatních atributů. Díky tomuto zjednodušení se tento klasifikátor označuje jako „naivní“ [25]. Ačkoliv je tento předpoklad často nevhodný pro aplikaci na problém v reálném světě, kdy jsou na sobě atributy silně závislé, tento klasifikační přístup pomáhá snížit rozměrnost díky zjednodušení problému.

Pokud předpoklad Bayesova naivního klasifikátoru neplatí, jsou využity Bayesovské sítě (někdy také pravděpodobnostní sítě). Zachycují vzájemné pravěpodobnostní závislosti atributů pomocí hran v acyklicky orientovaném grafu. Uzel v grafu představuje náhodný atribut a hrana mezi uzly reprezentuje pravděpodobnostní závislost mezi atributy. Uzly a hrany určují strukturu sítě a tabulky podmíněných pravděpodobností udávají parametry struktury. Pro vytvoření struktury jsou použity algoritmy K2 [11] a TAN [8].

Algoritmus nejbližších k-sousedů [6] je metoda, ve které se klasifikují prvky reprezentované vícedimenzionálními vektory do dvou nebo více tříd. Ve fázi učení se předzpracuje trénovací množina (training dataset) tak, aby všechny příznaky měly

1. řetězec textových znaků – string, výraz používaný pro text v programovacích jazycích



Obrázek 3.1: DOM strom

hodnotu 0 a rozptyl 1 - toto umístí každý prvek trénovací množiny do některého místa v N-rozměrném prostoru. Ve fázi klasifikace je dotazovaný prvek umístěn do téhož prostoru a je nalezeno k nejbližších sousedů. Objekt je pak zařazen do té třídy, kam patří většina z těchto nejbližších sousedů [29].

Rozhodovací stromy jsou užitečným nástrojem rozhodovací analýzy, vhodným především pro víceetapové rozhodovací procesy s jedním kritériem rozhodování. Umožňují zobrazení logického vývoje časově na sebe navazujících alternativních rozhodnutí a náhodných situací. Jejich cíl je stanovit optimální strategii rozhodovatele, tedy posloupnost rozhodnutí, která vede k nejlepší očekávané hodnotě zvoleného kvantitativního kritéria. Rozhodovací stromy jsou zvláštní případy grafu, skládají se tedy z uzlů a hran. Uzly rozhodovacího stromu představují etapy rozhodovacího procesu, ve kterých se střídá rozhodování subjektu a náhodné rozhodování. Z rozhodovacích uzlů vycházejí hrany, které reprezentují deterministické činnosti závislé na vůli rozhodovatele (různé varianty rozhodnutí). Ze situačních uzlů vycházejí hrany představující náhodné volby

vyskytující se s určitými pravděpodobnostmi. Náhodné volby tvoří úplnou soustavu jevů, a proto se součet pravděpodobností jejich výskytu musí rovnat jedné [13]. Pro implementaci algoritmu rozhodovacích stromů je zde použitý framework Weka a C4.5 algoritmus [27].

Po uplatnění všech čtyř algoritmů se vybere ten, který pro vstupní datovou množinu vykazuje nejlepší výsledky. Následně je vytvořen korektně formovaný dokument v XML formátu. Obsahuje čtyři rozdílné třídy tagů (page, title tag, information tag a main tag) a jejich identifikační popis, atributy a obsah, jak vidíme na obrázku 3.2. Tento dokument reprezentuje pravidla pro uložení informačního obsahu a umožňuje procházet obsah i cestu k obsahu pomocí funkcí pro manipulaci s řetězci, v čemž spočívá druhá fáze celého procesu. Je zde použit algoritmus pracující s regulárními výrazy a funkcemi pro hledání řetězce nebo jeho částí spolu s počítáním počtu začátků a konců tagů. Pokud se počet začátků i konců tagů shoduje, dokument byl správně vytvořen a řetězcové funkce snadno najdou informační obsah. Pokud se v tomto procesu vyskytne chyba a obsah nebo jeho část nebylo možné najít, vrací se celý proces na začátek k bodu vytváření DOM stromu a je opakován.

```
<?xml version="1.0" encoding="utf-8" ?>
<page>
<title tag="H2 class=BlackHead" parenttag="DIV
id=SecondLevelColOne">No agreement reached on Kosovo ™s
future status in Security Council</title>
<information tag="SPAN class=date" parenttag="DIV
id=SecondLevelColOne">16 January 2008 “</information>
<main tag="DIV id=SecondLevelColOne" parenttag="">No
agreement reached on Kosovo ™s future status in Security
Council 16 January 2008 “ … News Tracker: past stories on this
issue Uncertainty on Kosovo ™s future status could lead to
instability, warns Ban Ki-moon Video</main>
</page>
```

Obrázek 3.2: XML dokument, převzato z [28]

3.2 Čištění pomocí DOM

Metoda tohoto přístupu je popsána v práci „An Approach for Text Extraction From Web News Page“ [16]. Spočívá ve využití principu a vlastností DOM stromu jakožto hlavního nástroje k extrakci informací ze zpravodajských článků.

V prvním kroku se vhodným HTML parserem [2] stáhne zdrojový kód stránky a vytvoří se z něj DOM strom. Proces konverze HTML dokumentu do DOM stromu se nazývá parsování.

Druhý krok by se dal nazvat jako detekce a čištění nepotřebných informací. Vychází se zde z předpokladu, že struktura stránky mimo <body> tag (dále jen *body*) nenese žádné relevantní informace a tudíž může být odstraněna. Nejprve se pomocí DOM funkcí spočítá počet tagů vnořených do *body* struktury a počet tagů mimo ni. Většina textových informací se nachází uvnitř *body*, takže by *body* měl obsahovat více vnořených tagů, než zbytek struktury, tedy obsah <html> tagu bez *body*. Pokud je tento předpoklad pravdivý, celá non-*body* struktura je odstraněna.

Ve třetím kroku pokračuje proces čištění odstraněním odkazů. Na zpravodajské stránce se nachází dva typy informací. Prvním je text obsahu, což je klíčová informace a prvek, který chceme najít. Druhým je sekundární text, což jsou navigační odkazy, komentáře, popisy obrázků, reklamní odkazy a popisy odkazů atd. Nicméně, některé zpravodajské stránky obsahují odkazy a popis odkazu i v článku. Tento typ odkazu chceme uchovat, protože je součástí hledaného textu. Aby nebyly odstraněny podstatné odkazy, jsou zavedeny veličiny Ll (link length) a Cl (content length), představující délku cesty k odkazu a délku jeho textového popisu. Poté jsou z DOM stromu vybrány tagy, které obsahují blokovou strukturu (<table>,) a podle nich jsou vytvořeny stránkové bloky. Pro každý blok se spočítají Ll a Cl a prostřednictvím vzorce $\frac{Cl}{Ll}$ je vypočítána významnost bloku. Bloky s hodnotou výpočtu pod stanovenou okrajovou hodnotou jsou odstraněny.

Čtvrtý a poslední krok představuje zavedení nové veličiny označené jako WB pro „váha bloku“ (weight of block). Zde se vychází z předpokladu, že blok obsahující hledaný článek v sobě zahrnuje také nadpis článku (pravda v 98 % případů podle zmiňované studie). Vzorec pro spočítání WB je následující:

$$WB = \frac{TL}{WL}(ST + 1)$$

TL je označení pro délku textu počítaného bloku, WL je celková délka textu stránky a ST značí odpovídající hodnotu nadpisu nacházejícím se uvnitř bloku. Bloky jsou poté seřazeny podle své váhy a první blok představuje hledané textové informace.

2. syntaktický analyzátor, zde byl použit htmlparser.net

3.3 Čištění pomocí vizuálních prvků

Studie nazvaná "Cleaning web pages for effective web content mining" [22] pojednává o metodě čištění webových stránek pomocí vizuálních prvků stránky. Je zde aplikován VIPS (vision-based page segmentation) algoritmus [7] pro rozdělení stránky do bloků. Vstupem tohoto algoritmu je několik stránek ze stejné domény a výstupem jsou stránky rozdělené do bloků podle důležitosti.

VIPS algoritmus využívá DOM stromu a vizuálních prvků stránky jako je barva pozadí, velikost písma, umístění jednotlivých částí atd. pro rozdělení stránky do bloků s rozdílnými vlastnostmi. Potom jsou z každého bloku v listech stromu získány informace jako je identifikace, obsah, pozice a procento obsažených odkazů a následně je z DOM stromu a těchto informací vytvořen dokument ve formátu XML. Pomocí informací jako je délka a šířka bloku a jeho relativního umístění vzhledem k ostatním prvků na stránce se následně vypočítá jeho důležitost. Myšlenka tohoto postupu těží z faktu, že nepodstatné prvky stránky jsou většinou rozmístěny po krajích a samotný text se nachází uprostřed stránky. Čím blíž k okraji je blok umístěn, tím menší bude jeho důležitost. Důležitost pozice je v intervalu <0, 1>, kde 1 znamená nejnižší důležitost. U každého bloku se pak spočítá jeho poměr délky textu ku délce textu odkazů. Čím více se tento poměr blíží jedné, tím je větší procento odkazů v tomto bloku a tím spíš se bude jednat o nepodstatný blok. Tímto způsobem rozdělíme každou vstupní stránku do jednotlivých bloků. Dalším krokem je porovnání bloků stránek mezi sebou. Zde je uplatněna myšlenka, že bloky s reklamami, komentáři a navigační bloky mají mezi stránkami na stejné doméně stejnou strukturu a mnohdy i obsah. Bloky s textem a článkem se svojí velikostí a obsahem liší. Čím více se konkrétní bloky jednotlivých stránek mezi sebou shodují, tím pravděpodobněji se jedná o boilerplate a může být odstraněn. Úroveň similarity bloků je zde opět vyjádřena v intervalu <0, 1>, kde 1 znamená naprostou shodu.

Z každé stránky se vybere předem stanovený počet bloků s nejvyšší důležitostí spočítané poměrem similarity bloků, procenta odkazů a relativní pozice podle vzorce:

Důležitost bloku = 1 – ($\frac{1}{2}$ úroveň similarity + $\frac{1}{3}$ procento odkazů + $\frac{1}{6}$ pozice)

Vyšší důležitost mají bloky blížící se jedné a na nich mohou být použity algoritmy pro hledání konkrétního textu. Problém klasifikace webové stránky zde byl převeden na prostý problém klasifikace textu.

3.4 Čištění pomocí seskupování

Čištění stránky pomocí seskupování a následného označování je prezentováno v práci „Automatic Data Record Detection in Web Pages" [14]. Tento přístup je trochu odlišný od ostatních, protože nevyžaduje trénovací množinu dat ani více stránek ze stejného webového portálu na vstup. Pracuje s každou zadanou stránkou zvlášť. Je zde zave-

den pojem volně přeložitelný jako textový prvek (text token), který představuje textový obsah tagu a všechny jeho rodiče a potomky[3]. Každá webová stránka je zde reprezentována pomocí množiny textových prvků. Textovým prvkům je přiřazen poziční atribut reprezentující pozici prvku vzhledem k ostatním prvkům.

Předpokládá se, že podobné textové prvky patří do stejné kategorie na stránce (do stejné tabulky, ke stejnému článku, odstavci atd.). Využívá se zde algoritmus nejbližších k-sousedů-HAC [30] pro seskupení podobných prvků do jedné kategorie. Algoritmus pracuje tímto způsobem:

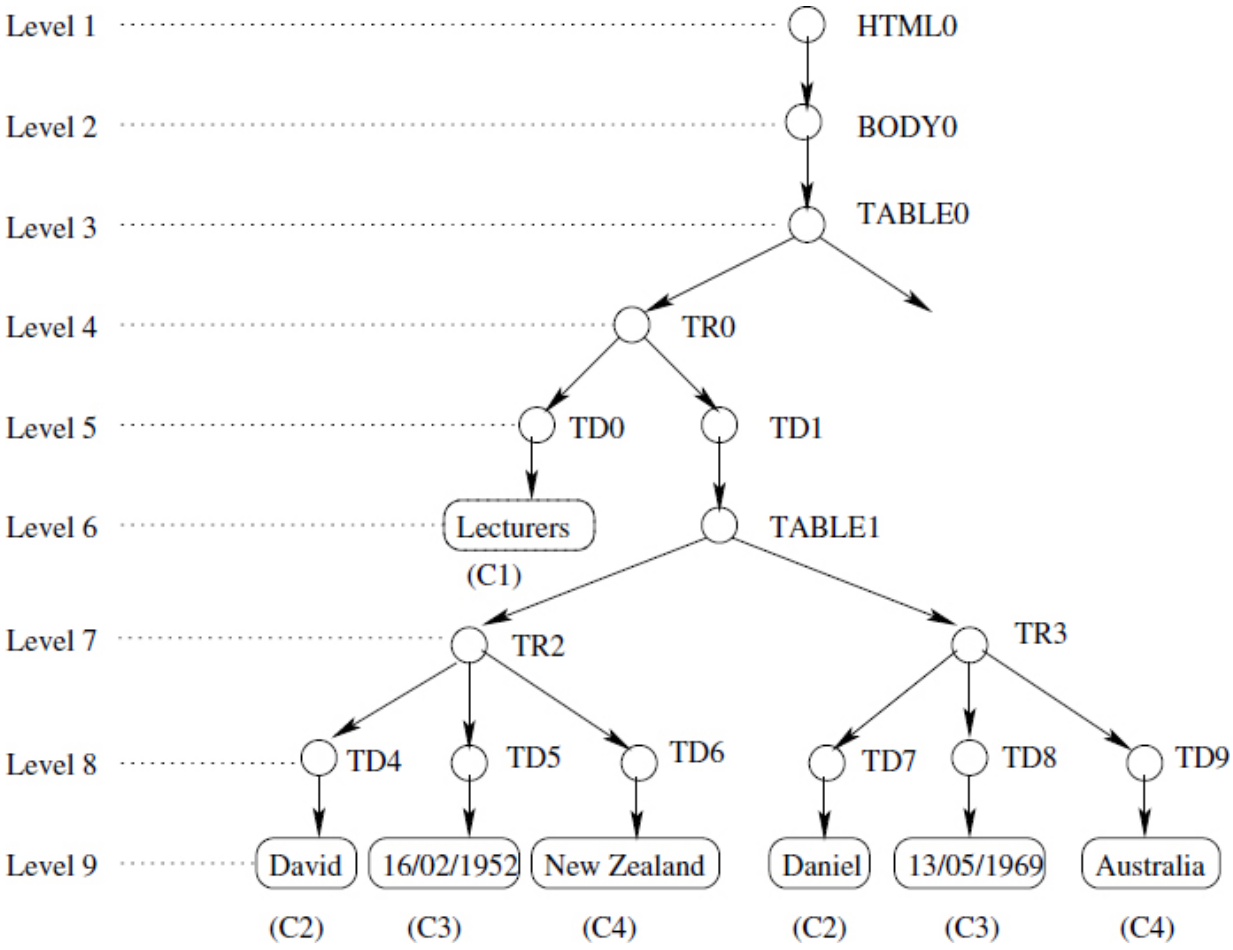
Krok 0: Inicializuj D, K, *delta*
Krok 1: Označ všechny textové prvky jako samostatnou kategorii.
Krok 2: Porovnej každou kategorii s jejími nejbližšími K sousedními kategoriemi (podle pozičního atributu). Podobné kategorie jsou seskupeny.
Krok 3: Urči představitele kategorie výběrem textového prvku se středovou hodnotou (mediánem) pozičního atributu.
Krok 4: Zvyš K o *delta*.
Krok 5: Opakuj kroky 2 – 4 dokud nejsou všechny kategorie porovnány a žádné kategorie už nemohou být seskupeny, nebo po D opakování.

Čísla K a *delta* jsou určeny na základě celkového počtu textových prvků „t“ na stránce. Ve studii se osvědčilo určení K = $\frac{t}{20}$ a *delta* = $\frac{t}{10}$.

Toto seskupení do kategorií by v ideálním případě seskupilo všechny stejné textové prvky do stejné kategorie a postačilo by k určení požadovaných informací. Reálně však toto dostačující není, stránky často obsahují nepravidelnosti a kategorie pak obsahují chybné textové prvky. Proto jsou zde zavedeny další dvě označení, a to číslo úrovně a seznam rodičů. Pomocí DOM je vytvořen strom pro zadanou stránku. Při tvorbě stromu se vynechávají tagy, které nepřidávají žádnou další úroveň struktury, jako jsou <b>, <i> a <br> tagy. Díky tomuto stromu je ted' možné určit úroveň vnoření každého textového prvku a seznam jeho rodičů. Jak můžeme vidět v obrázku 3.3, úroveň vnoření textového prvku „David“ je 9 a jeho seznam rodičů je [TD4, TR2, TABLE1, TD1, TR0, TABLE0, BODY0, HTML0]. Tyto dvě informace jsou přidruženy ke každému textovému prvku.

Následně je možné seskupovat kategorie na základě jejich úrovně vnoření a jejich rodičovského tagu. Tady se vychází z předpokladu, že textové prvky stejného charakteru mají stejnou úroveň vnoření a stejný seznam rodičů. Tímto seskupením dostáváme několik kategorií obsahujících stejné nebo podobné textové prvky. Nicméně, občas nastane situace, kdy podobné kategorie mají rozdílného rodiče a tudíž nejsou seskupeny. Z pozorování vyplývá, že každá kategorie obsahuje několik prvků patřících do jiné kategorie. Je stanoveno pravidlo, že dvě kategorie je možné seskupit, pokud obě mají stejnou úroveň vnoření a obsahují stejného prarodiče. Tímto seskupením dostáváme kategorie reprezentující rozdílné množiny dat na stránce, jako je například článek, navigační lišta, záhlaví atd. Výsledky jsou uloženy do dokumentu ve formátu XML, který případně

3. rodič a potomek tagu – stejné jako rodič a potomek uzlu ve stromových strukturách



Obrázek 3.3: Příklad DOM stromu, převzato z [14] a upraveno

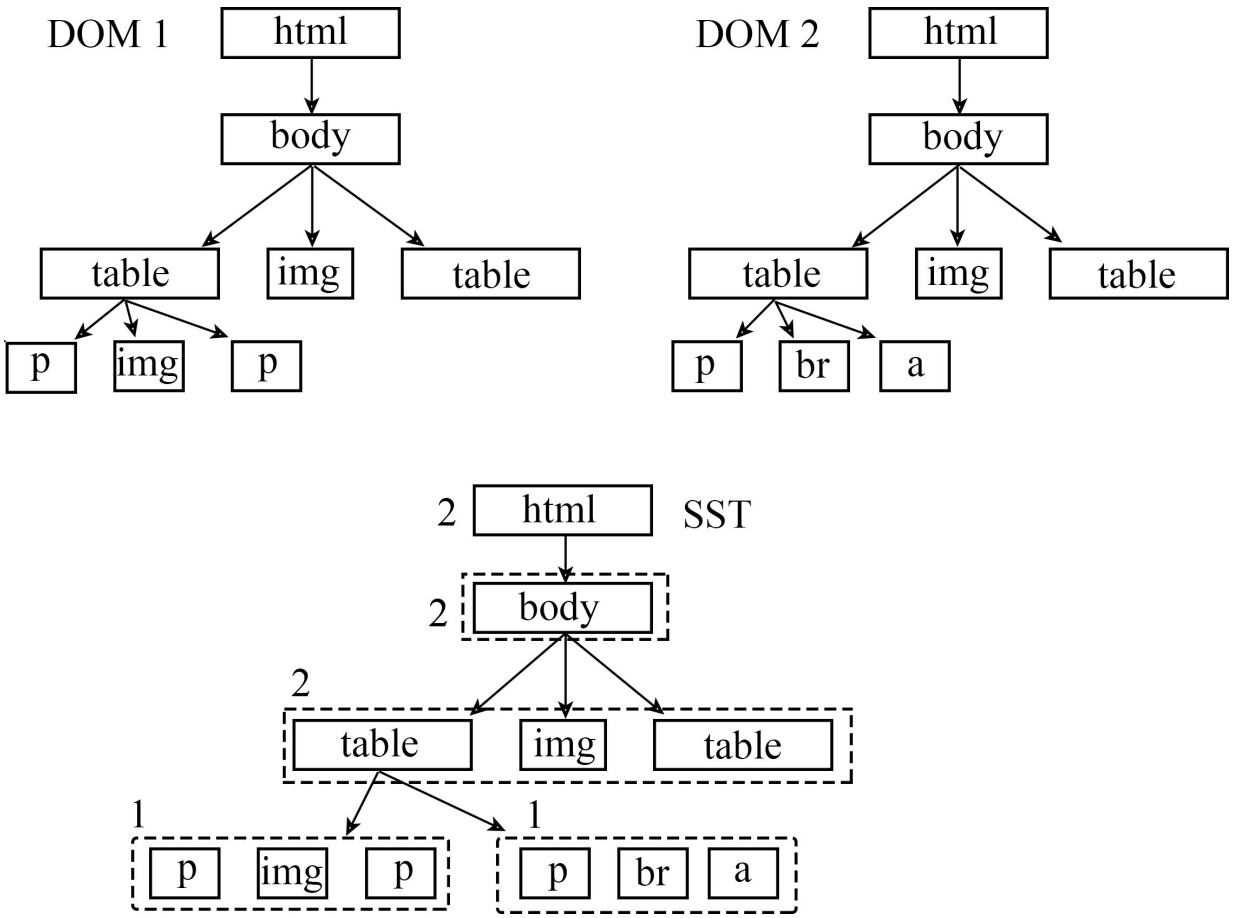
umožní algoritmům pro klasifikaci textu jednoduchý výtah požadovaných textových informací.

3.5 Čištění pomocí SST

Způsob čištění použitím Site Style Tree (vysvětleno později v textu) vznikl na základě práce „Study to Eliminating Noisy Information in Web Pages based on Data Mining“ [15]. Zde je opět využit předpoklad, že nepodstatné informace na stránce sdílejí obvykle stejnou strukturu a stejný obsah a podstatné informace se ve svém obsahu, velikosti a vzhledu liší.

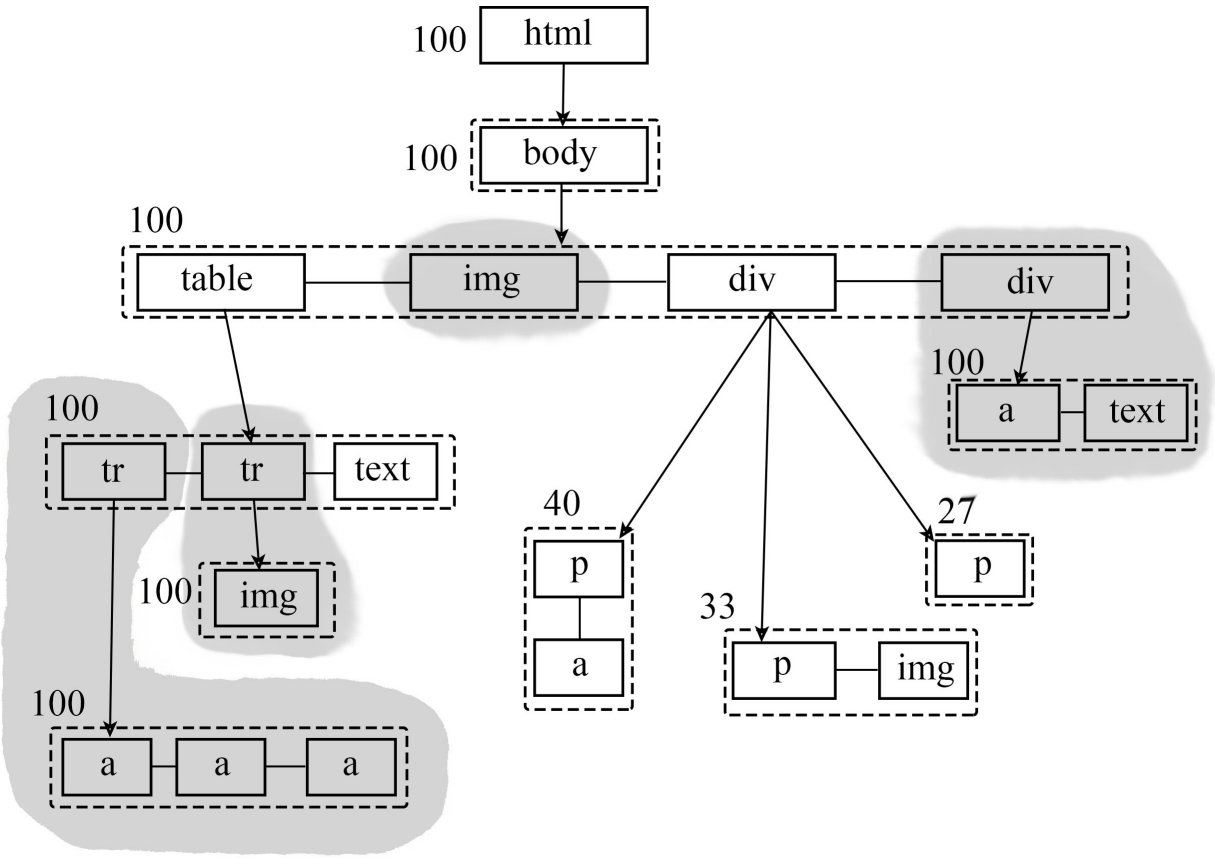
Zavádí se zde nová konstrukce nazvaná Site Style Tree (dále jen SST) pro vyjádření společných prvků stránky a jejich obsahu. Stránky jsou nejdříve parsovány do DOM stromů. Jak vidíme na obrázku 3.4, SST představuje kombinaci DOM stromů stránek.

Můžeme snadno vypozorovat, že obě stránky mají společnou strukturu, až na tagy (p, img, p) ve spodní úrovni stromu 1 a tagy (p, br, a) ve spodní úrovni stromu 2. Oba stromy jsou sjednoceny do SST následujícím způsobem. Shora – dolů projdeme každý uzel (tag) DOM stromu a snažíme se najít odpovídající uzel v DOM stromu druhé stránky. Pokud ho najdeme, porovnáme posloupnosti potomků obou uzlů. Když se posloupnosti shodují, zvýšíme číslo počtu konkrétní posloupnosti v SST. Pokud se posloupnosti potomků neshodují, zkopírujeme obě posloupnosti pod jeden uzel v SST. Takto pokračujeme v porovnávání až na spodní úroveň obou stromů.



Obrázek 3.4: Tvorba SST, převzato z [15] a upraveno

Pro každou vstupní stránku je crawlerem vyhledán předem určený počet dalších stránek ze stejné domény a pro každou je vytvořen DOM strom. Poté jsou DOM stromy a SST porovnávány a spojovány tak dlouho, dokud neporovnáme poslední DOM strom se SST. Výsledný SST vytvořený ze 100 stránek můžeme vidět na obrázku 3.5. Čísla u jednotlivých posloupností tagů udávají počet zastoupení určité posloupnosti mezi celkovým počtem spojených stránek.



Obrázek 3.5: Celkový SST, převzato z [15] a upraveno

Z výsledného SST můžeme jednoduše určit, které uzly jsou významné. Listy SST, které mají vysoké číslo zastoupení jsou prvky, které se na stránkách pravidelně opakují a tím pádem nejsou významné. Listy s nižším počtem zastoupení představují prvky, které jsou pro každou stránku rozdílné a nejspíše obsahují relevantní obsah. Pomocí algoritmů, které zde nebudeme podrobněji popisovat jsou potom odstraněny nevýznamné podstromy SST. Základní myšlenka je, že uzly obsahující potomky s nízkým číslem zastoupení nesou podstatnou informaci a nejsou odstraněny. Ostatní uzly obsahující opakující se podstromy odstraněny jsou. Na obrázku 3.5 jsou odstraněny šedé části SST.

Po odstranění nepodstatných struktur nám výsledný SST udává, kde a jakým způsobem jsou na konkrétní stránce uloženy relevantní informace.

3.6 Čištění pomocí vlastností textu

Práce „Boilerplate Detection using Shallow Text Features“ [19] popisuje postup čištění stránek na základě vlastností „mělkého“ textu. Tím myslíme všechen text, který se na dané stránce vyskytuje. Tento přístup k problému je velmi odlišný od všech ostatních, protože nepracuje s DOM stromem, s bloky stránky, neporovnává mezi sebou množinu stránek ze stejného portálu, nepotřebuje trénovací množinu stránek a funguje pro každou stránku samostatně.

Základní myšlenka studie vyplývá z pozorování chování autora při vytváření informativní stránky. Pokud chce autor sdělit nějakou novou informaci, obvykle používá více či méně podrobný popis a snaží se co nejvíce osvětlit svoje sdělení pro co nejlepší pochopení. Využívá k tomu gramatické konstrukce jako jsou celé věty, odstavce, nadpisy. Taková informace se typicky vyskytuje v článku. Krátký text, zkratky, nekompletní nebo prosté věty skládající se z malého počtu slov autor využívá, pokud chce ekonomicky a rychle sdělit informaci, o které se očekává, že jí příjemce pochopí bez většího úsilí (např. „Kontaktujte nás“, „Čtěte více“). Takovýto text se často využívá v hlavičkách stránek a v navigaci, boilerplate obecně.

Metoda čištění pomocí vlastností textu je založena na počítaní slov a hustotě odkazů mezi prvky stránky. Delší text s malou hustotou odkazů znamená nejspíš podstatnou informaci, krátký text s velkým počtem odkazů je pravděpodobně boilerplate. Využívají se zde pouze určité prvky struktury stránky jako jsou titulní tagy (h1 – h6), tag odstavce (p), tag pro členění (div) a tag pro odkaz (a). V podrobném výzkumu byly použity vlastnosti jako počet slov v prvku, průměrná délka slova, průměrná délka věty, hustota textu v každém prvku a jejich kombinace. Navíc se využilo předpokladu, že informativní text je většinou obklopen ostatními prvky stránky, boilerplate. Relativní pozice prvku vůči ostatním prvkům na stránce nám dává cennou informaci při rozhodování, jestli se jedná o podstatný prvek.

V důkladných testech těchto vlastností na čtyř různých korpusech a na kolekcích stránek nasbíraných pomocí GoogleNews[4] a CleanEval [5] bylo zjištěno, že kombinací tří vlastností (relativní pozice, průměrná délka slova a počet slov v prvku) můžeme odlišit podstatnou informaci od boilerplate. Nejlepších výsledků bylo dosaženo, když všechny prvky s více než deseti slovy a poměrem hustoty textu ku hustotě odkazů menším než 33 % byly považovány za podstatnou informaci. Aplikováním dalších vlastností bylo rozlišování mezi hlavním textem a boilerplate ještě více zpřesněno. Pro kolekci GoogleNews bylo dosaženo přesnosti až 95,93 %. Navíc tato metoda dosahuje výsledků mnohem rychleji, než všechny ostatní testované metody, protože se jedná o prosté počítání složek textu. Podrobná analýza viz [19].

4. kolekce zpravodajských stránek nasbíraná z portálu `http://news.google.com/`

3.7 Shrnutí

Drtivá většina dnešních technik pro čištění webových stránek a detekování podstatných informací využívá možností DOM stromu. Druhým oblíbeným způsobem je porovnávání více stránek ze stejného webového portálu mezi sebou a hledání shodných struktur. Majorita technik využívá kombinaci těchto přístupů nebo některé z technik uvedených v předchozích kapitolách k dosažení co nejvyšší přesnosti. Efektivitu algoritmů je možné porovnat díky projektu CleanEval [5], který pro algoritmy stanovuje porovnávací kritéria. Díky jednotným kritériím je dnes možné porovnat řadu dostupných algoritmů z hlediska přesnosti, rychlosti atd.

Nejpřesnějším a nejefektivnějším řešením je prozatím algoritmus uvedený v práci „Boilerplate Detection using Shallow Text Features“ [19], proto byl vybrán pro použití v aplikaci vyvíjené v rámci této práce.

4 Algoritmus

4.1 Navržený přístup

Uživatel poskytne aplikaci několik rozdílných URL ze stejné domény. Nástroj nejprve stáhne zdrojové kódy zadaných webových stránek a odstraní nežádoucí obsah jako jsou komentáře, odkazy, obrázky, boilerplate. Poté analyzuje stromovou strukturu cesty k textu a pro každou zadanou stránku si tuto cestu uloží. Následně provede průnik uložených cest mezi stránkami. Vychází se zde z faktu, že zpravodajské weby jsou vyvíjeny s použitím šablon a dá se předpokládat, že každá relevantní informace má stejnou cestu [1] pro každou stránku domény. Pokud tedy na jedné stránce cesta k informaci o autorovi vypadá takto,
html/body/div/p [32]
očekáváme, že na druhé stránce stejné domény je cesta k informaci o autorovi shodná. V dalším kroku se aplikace pokusí heuristicky vybrat shody, které odpovídají hledaným informacím. Diagram celého postupu je znázorněn na obrázku 4.1. Nalezené prvky potom vypíše na obrazovku a nechá uživatele, aby zkontroloval a vybral úspěšně nalezená data.

4.2 Vlastní algoritmus

V této podkapitole je popsán algoritmus vyvinutý pro nalezení konkrétních prvků na vyčištěné stránce českého zpravodajského serveru. Poté, co nám existující nástroje pomohou získat ze stránky čistý text, přichází na řadu prohledávání textu a získávání informací, o které máme zájem. Jak je uvedeno výše, na tento problém neexistuje a ani nemůže existovat obecný algoritmus, musíme si proto vytvořit vlastní.

Úkolem je vytvořit algoritmus, který prohledá čistou stránku webového serveru a nalezne informaci o autorovi článku, datu, nadpisu a článek samotný. Celkový postup je rozdělen na čtyři části, každá pro konkrétní hledanou informaci. První myšlenka algoritmu vychází z faktu, že ve vyčištěné zpravodajské stránce se již nenachází mnoho dalších údajů, kromě našich, hledaných. Proto při každé nalezené informaci algoritmus po uložení tuto informaci odstraní z prohledávaného dokumentu, aby se dalších ledání provádělo na menším a menším obsahu. Druhá myšlenka vyplývá z následující logické úvahy: pokud hledaná informace není nalezena, musíme rozšířit hledání. Algoritmus tedy při neúspěchu rozšiřuje postupně hledání až na nevyčištěnou stránku. V případě rozšířeného hledání se ale už v dokumentu nachází mnohem víc obsahového materiálu, včetně případného boilerplate. Nalezení konkrétní informace je pak obtížnější a může se stát, že sice nalezneme informaci například o datu, ale nebude to datum zveřejnění článku, ale datum příspěvku do diskuze pod článkem. Proto zavádíme údaj nazvaný skóre, který určuje, s jakou přesností nalezená informace odpovídá konkrétní hledané

1. Z anglického path, vyjádření pozice prvku uloženého ve stromové struktuře webové stránky

informaci. Každá část algoritmu hledá rozdílná data a má tedy pokaždé trochu jiný průběh. Autor, datum a nadpis článku jsou sice hledány pomocí podobného principu, pro každou část se však snažíme najít jiné zlomky informací a každá část má rozdílnou pravděpodobnost na úspěch. Způsob hledání článku se liší od předchozích tří a provádí se až nakonec, kdy už by v dokumentu nemělo být tolik ostatních dat.

Předpokládáme, že v případě zpravodajských stránek má každý zveřejněný článek svůj nadpis, datum a autora článku. Tento předpoklad můžeme využít, když nám uživatel poskytne více stránek ze stejné domény. Můžeme provést průnik stránek a pořád by nám měly hledané informace zůstat. Samotné hledání tedy nejdřív aplikujeme na průnik vyčištěných stránek. Pokud se zde požadovaná informace nenachází, rozšiřujeme hledání na jednotlivou vyčištěnou stránku. Pokud ani tady neuspějeme, nezbývá nám než hledat na celé, nevyčištěné stránce. Tady už ale očekáváme, že výsledky mohou být nepřesné.

4.2.1 Datum a autor

Hledání konkrétních dat provádíme pomocí regulárních výrazů. Vývojáři stránek do svých zdrojových kódů zpravidla začleňují identifikace, které pomáhají při hledání konkrétního textu. V případě informace o datu se obvykle někde v kódu, který tuto informaci zobrazuje, vyskytuje popisek *date*, nebo *datum*. Pro autora článku se používá popisek *author*, případně *authors*. Tyto popisky představují důležitý zdroj. Algoritmus prochází stránku a pomocí regulárních výrazů hledá konkrétní popisek. Pokud ho nenajde, rozšíří hledání. Zavádíme i další regulární výrazy, které pomáhají zpřesnit nalezený údaj, ale klíčovou informaci pořád představuje právě popisek v kódu.

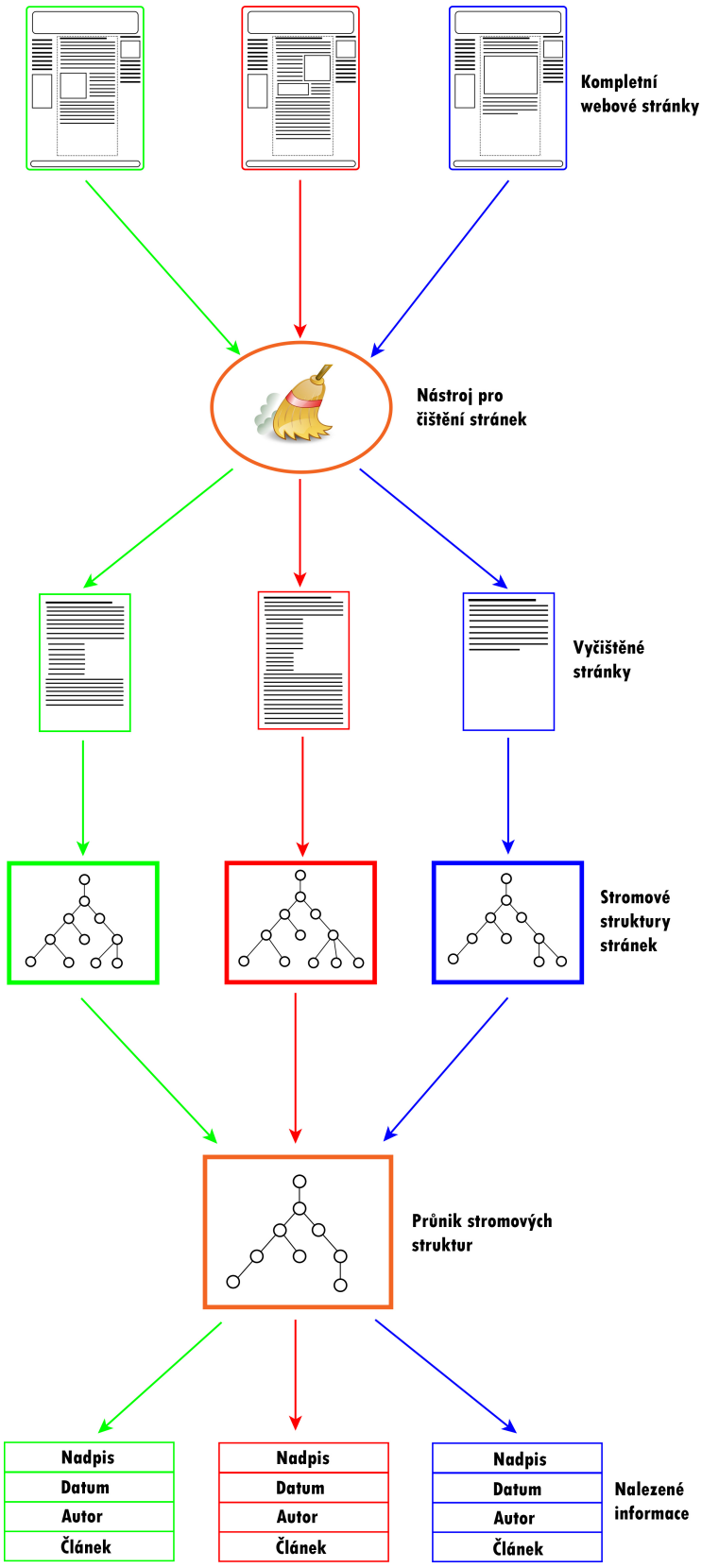
4.2.2 Nadpis

Hledání nadpisu zjednodušuje fakt, že na většině zpravodajských stránek se nadpis nachází na stejném místě. Tag *h1* se v HTML používá přesně pro účel nadpisu. Vývojáři stránek ho proto k tomuto účelu hojně využívají a zjednodušují nám tím hledání. Pro kontrolu zavádíme i regulární výrazy hledající velké písmeno na začátku věty a tečku za větou. Po prohledání průniku vyčištěných stránek sice rozšiřujeme hledání dále, ale pokud nadpis není v *h1* tagu, jinde nejspíš také nebude.

4.2.3 Článek

Jako poslední hledá algoritmus článek. Tady těžíme ze skutečnosti, že článek je zpravidla nejdelší textová část na stránce. Hledáme proto oddíly stránky s největším počtem vět. Článek se nachází na každé zpravodajské stránce, proto se musí nutně nacházet v průniku stránek. Jestli se stane, že největší textová část se nenachází v průniku, pak se nejspíš jedná o prvek na stránce odlišný od článku. Takovým prvkem může být dlouhá tabulka nebo abstrakt s pár větami článku. Pokud tato situace nastane, algoritmus tuto část odstraní a pokračuje v hledání největší textové části.

Všechny vyhledané informace jsou poskytnuty uživateli k ověření pomocí webového rozhraní. Uživatel má pak možnost zkontrolovat, jestli algoritmus našel požadované informace a případné špatně nalezené výsledky odstranit. Konečný výsledek může být uložen do souboru.



Obrázek 4.1: Diagram navrženého přístupu

5 Implementace

V kapitole implementace jsou uvedeny postupy a metody, které byly zvoleny pro vytvoření praktické části práce. Nejprve je popsán výběr nástrojů a základní problémy, se kterými se aplikace musela vypořádat, následuje a rozbor kódu aplikace.

5.1 Výběr nástrojů

V podkapitole 4.1 Navržený přístup byly definovány základní funkce aplikace. Můžeme je rozdělit do několika kroků:

- Stáhnutí zdrojového kódu stránek ze zadaných URL adres
- Vyčištění stránek
- Konvertování stránek do DOM stromů
- Nalezení průniku DOM stromů
- Nalezení cesty k požadovaným informacím

Nejprve bylo potřeba vybrat z existujících nástrojů ty, které by v aplikaci mohly být použity. S tímto bodem se pojí i otázka výběru verze jazyka Python. Verze Python 3 je sice nejnovější verzí jazyka, ale právě kvůli tomu pro ni ještě nebylo vyvinuto tolik podpůrných nástrojů. Verze Python 2.7.4 je poslední funkční verzí Pythonu 2, zachovává funkčnost veškerých starších nástrojů a je pořád plně podporována vývojáři. Jazyk Python ve verzi 2.7.4 nabízí mnoho knihoven a vývojových rámců usnadňujících vývoj aplikací.

V naší aplikaci můžeme použít nástroj pro stažení stránek z URL adres, nástroj pro vyčištění stránek a nástroj pro parsování. Svoji roli zde také hraje modul regex obsažený ve standardních knihovnách Pythonu. Regex umožňuje tvorbu, zpracování a aplikování regulárních výrazů. Pro průnik DOM stromů obecný nástroj neexistuje, ale samotný jazyk nabízí funkce, které tuto práci usnadňují. Nalezení požadovaných informací, tedy datum článku, autora, nadpis a samotný článek je natolik specifický úkol, že pro něj obecný nástroj ani existovat nemůže. Tuto funkcionalitu musíme vytvořit.

5.1.1 Stažení zdrojových kódů

Při výběru nástroje pro stažení zdrojového html kódu byly zváženy tři nejoblíbenější knihovny pro tuto funkci.

- URLlib
- URLlib2

- Requests

Knihovna třetí strany pojmenovaná Requests přináší rozšířenou funkcionalitu a revitalizaci knihoven urllib a urllib2. První dvě uvedené knihovny jsou součástí standardních knihoven jazyka Python 2.7.4 s tím rozdílem, že urllib2 rozšiřuje základní urllib a nabízí komplexnější funkcionalitu. Pro potřeby naší aplikace, kdy chceme pouze stáhnout zdrojový html kód a předat ho jiné knihovně, nám bohatě postačí knihovna urllib.

5.1.2 Vyčištění stránek

Standardní knihovny Pythonu 2.7.4 neobsahují žádný nástroj pro tento úkol, proto bylo vybíráno pouze z knihoven třetích stran. K dispozici jsou knihovny implementující postupy zmiňované ve druhé kapitole nazvané Související výzkum. Podle údajů vyplývajících z CleanEval [5] a podle srovnání uvedených v práci „Evaluating Web Content Extraction Algorithms“ [21] je zatím nejlepším řešením knihovna Boilerpipe [1]. Na základě těchto údajů byla knihovna Boilerpipe zvolena pro řešení problému čištění stránek v naší aplikaci.

Boilerpipe je knihovna původně napsaná pro Javu Christianem Kohlschütterem. Obsahuje algoritmy které detekují a odstraňují na webových stránkách prvky a věci, které nenesou relevantní informace, jako jsou boilerplate, popisky, hlavičky a zápatí, komentáře. Je určena pro extrakci článků ze zpravodajských portálů. Algoritmy, které tato knihovna používá jsou založeny na práci "Boilerplate Detection using Shallow Text Features[1]"[19] od Christiana Kohlschüttera prezentované na WSDM 2010 – Třetí Mezinárodní ACM [2] konferenci o webovém vyhledávání a data miningu v New Yorku [1]. V roce 2012 byla tato knihovna přepsaná i do jazyka Python.

5.1.3 Parsování

Pro parsování stránky do DOM stromu jsme opět nuceni použít nástroj třetí strany. Naštěstí je většina těchto nástrojů už dlouhodobě používána a pořád plně podporována. Pro Python existují následující možnosti[3]:

- lxml
- HTQL
- Scrapy
- html5lib

1. Detekování boilerplate pomocí vlastností textu
2. Association for Computing Machinery – profesní sdružení počítačových odborníků
3. `http://lxml.de/`, `http://htql.net/`, `http://scrapy.org/`, `https://github.com/html5lib`, `http://wwwsearch.sourceforge.net/mechanize/`, `http://www.crummy.com/software/BeautifulSoup/`

- Mechanize
- Beautiful Soup

V rámci této práce byly podrobněji vyzkoušeny knihovna Beautiful Soup a vývojový rámec Scrapy. Funkce rámce Scrapy nejsou omezeny jen na parsování, nabízí i funkce pro tvorbu vlastního crawleru, jeho aktivní sledování, funkce pro logování, pro stahování obrázků, ukládání v různých formátech a další. Beautiful Soup obsahuje zlomek funkcionality Scrapy, ale pro naší aplikaci je tato funkcionalita dostatečná. Beautiful Soup je knihovna napsaná v jazyce Python, navržená pro analyzování a parsování HTML a XML dokumentů. Poskytuje funkce pro procházení, úpravu a vyhledávání v parsovaném stromu. Takových knihoven je dnes k dispozici více, avšak hlavní výhoda Beautiful Soup spočívá v její jednoduchosti, univerzálnosti a také faktu, že se dokáže vypořádat s nekorektně vytvořeným HTML kódem (neuzavřené nebo špatně použité tagy). Knihovna automaticky překládá vstupní dokumenty do Unicode a výstupní do UTF-8 a zároveň dokáže detekovat formát vstupního dokumentu [3].

5.2 Technické problémy

Hlavním problémem, na který narážíme hned zpočátku, je kódování. Kódování zpravodajské stránky nemůžeme s jistotou určit (i když je téměř pravidlo pro české zpravodajské portály používat kódování UTF-8). V hlavičce stránky je sice uvedeno, jaké kódování stránka používá, ale samotný text může být v jiném kódování. Můžeme implementovat přepínač, který bude postupně zkoušet dekódovat dokument různými typy kódování, ale ani tak nemáme jistotu, že autor dokumentu nepoužívá nějaký znak v jiném kódování (mnohdy aniž by o tom věděl). S tímto problémem se naštěstí dokáže vypořádat knihovna Beautiful Soup, která dokáže dekódovat i špatně zakódovaný dokument, takže implementace přepínače není zapotřebí.

Dalším problémem je fakt, že každá zpravodajské stránka je vyvíjena jiným způsobem a tomu také odpovídá zdrojový kód. Postupy, které najdou informace na jedné stránce, nemusí tyto informace najít na stránce jiné. Musíme proto volit cestu, která co nejpřesněji umožní najít data na co největším počtu stránek. Je žádoucí, aby algoritmus pro hledání fungoval tak, že v případě nenalezení informace specifickým postupem aplikuje jiný typ postupu, dokud informaci nenalezne.

Diskuzní fóra a diskuze samotné mají úplně jinou strukturu než zpravodajské stránky a blogy. Hledání informací v diskuzi by se muselo rozdělit na hledání v jednotlivých příspěvcích. To by vyžadovalo vývoj a implementaci nového algoritmu a poté by se v aplikaci volil způsob extrakce informací podle typu vložených stránek. Museli bychom také použít jinou (méně přesnou) knihovnu pro odstraňení boilerplate, protože použitá Boilerpipe odstraňuje diskuzní příspěvky jako nežádoucí obsah. Z těchto důvodů nebyla extrakce informací z diskuzních fór implementována, ale může být předmětem dalšího výzkumu.

5.3 Aplikace

V jazyce Python lze vyvíjet aplikace a programy jak procedurálním, tak objektovým programováním. Pro vývoj aplikace v této práci bylo použito převážně objektového způsobu programování s občasným využitím výhod programování procedurálního.

Samotná aplikace se skládá z jediné třídy nazvané *Recognizer* a ta obsahuje několik samostatných metod. Vstupním argumentem třídy *Recognizer* je seznam několika URL adres ze stejného webového zpravodajského portálu. Aplikace může pracovat i s jedinou zadanou adresou, ale výsledky jsou přesnější při zadání aspoň dvou adres. O spuštění a správný chod aplikace se stará metoda *start*.

5.3.1 Příprava

Aplikace nejprve postupným vyvoláním několika metod připraví nezpracovaný zdrojový kód pro budoucí vyhledávání.

První metoda *(proces_urls)* stáhne zdrojové kódy zadaných stránek pomocí knihovny urllib, parsováním knihovnou Beautiful Soup z nich vytvoří stromové struktury a odstraní komentáře v kódu. Výsledné stromy uloží do proměnné *(procesed_urls)*). Následně je vyvolána metoda *(clean_urls)*, která s využitím knihovny Boilerpipe odstraní z uložených stromů boilerplate a další nežádoucí prvky a uloží výsledky do další pomocné proměnné *(cleaned_urls)*. Metody *divide_to_tagpaths* a *short_full_paths_to_string* jsou pouze pomocné, pro snadnější a rychlejší manipulaci s uloženými stromy, obsah nijak nemění.

Aplikace pokračuje vyvoláním další metody *(count_paths)*. Ta má za úkol spočítat, kolikrát se na každé jednotlivé stránce nachází shodná cesta k textu. Formálně řečeno, kolik je v každém stromu shodných cest k listovým prvkům. Tento výpočet nám později pomůže při určování článku. Každá zpravodajská stránka má svůj článek rozdělený do několika oddílů – odstavců, případně tabulek. Základní myšlenka počítání cest vychází z pozorování, že na vyčištěné stránce se již mnoho textových prvků se stejnou cestou nevyskytuje, takže článek představuje téměř jediný text rozdělený do stejně pojmenovaných oddílů. Finální počty jsou začleněny do výsledků pomocných metod.

V dalším kroku je vyvolaná metoda *intersect_pages*. Úkolem této metody je provést průnik všech cest k textu mezi zadanými stránkami. Vycházíme z předpokladu, že stránky ze stejné zpravodajské domény používají stejnou šablonu pro vkládání nových informací. Cesta k informaci o datu vložení článku, o autorovi, cesta k nadpisu i cesty k článku by proto měly být shodné. Výsledné cesty k textu obsažené v každé zadané stránce jsou uloženy v proměnné *(intersected_paths)*. Obsah této proměnné je při hledání konkrétních informací klíčovým prvkem úspěšného nalezení.

5.3.2 Hledání data

Problém hledání data a případně času vložení a aktualizace článku ze zpravodajské stránky v naší aplikaci řeší metoda *find_date*. Nejprve jsou definovány dva regulární výrazy. První umožňuje zjistit, jestli se v atributech některé z cest k textu nevyskytuje

výraz „date“ nebo „datum“. Vývojáři webových stránek často používají označení některých tagů pro lepší orientaci v kódu a také pro vyjádření konkrétního obsahu nebo funkcionality tagu. Ve zpravodajských portálech se obvykle používá označení „class=date“ a nebo „id=date“ pro tag, který obsahuje informaci o datu. Druhý regulární výraz zjišťuje, jestli text neodpovídá formátu data. Datum má velmi specifický formát, obsahuje aspoň jednu tečku, číslo dnu v měsíci, měsíc vyjádřený číslem nebo slovem a občas rok. Proto tento výraz dokáže s vysokou pravděpodobností určit, jestli se v textu nachází datum. Následuje algoritmus, který by se dal vyjádřit takto:

Krok 1: Aplikuj regulární výraz „date“ pro každou cestu k textu mezi cestami v průniku.
Krok 2: Pokud najdeš shodu, aplikuj druhý regulární výraz (formát data) na text nalezené cesty.
Krok 3: Pokud najdeš shodu, datum je nalezeno, vypiš a odstraň prvek data ze stromu, ukonči hledání. Pokud nenajdeš, pokračuj.
Krok 4: Aplikuj regulární výraz „date“ pro každou cestu k textu mezi cestami vyčištěné stránky.
Krok 5: Pokud najdeš shodu, aplikuj druhý regulární výraz na text nalezené cesty.
Krok 6: Pokud najdeš shodu, datum je nalezeno, vypiš a odstraň prvek data ze stromu, ukonči hledání. Pokud nenajdeš, pokračuj.
Krok 7: Aplikuj druhý regulární výraz (formát data) pro každý text mezi texty vyčištěné stránky.
Krok 8: Pokud najdeš shodu, datum je nalezeno, vypiš a odstraň prvek data ze stromu, ukonči hledání. Pokud nenajdeš, pokračuj.
Krok 9: Aplikuj regulární výraz „date“ pro každou cestu k textu mezi cestami v nevyčištěné stránce.
Krok 10: Pokud najdeš shodu, aplikuj druhý regulární výraz (formát data) na text nalezené cesty.
Krok 11: Pokud najdeš shodu, datum je nalezeno, vypiš a odstraň prvek data ze stromu, ukonči hledání. Pokud nenajdeš, ukonči hledání.

Algoritmus nejdříve zkouší najít informaci o datu mezi průnikem stránek. Pokud je zdrojový kód zpravodajské stránky napsán korektně, datum by se mělo vyskytovat zde. Bohužel, zdaleka ne všechny stránky jsou psány tímto způsobem, proto v případě neúspěchu musíme pokračovat v hledání. Pokračujeme rozšířeným hledáním na celé, vyčištěné stránce. Pokud i tady není nalezeno datum, algoritmus rozšíří hledání na celou, nevyčištěnou stránku. V tomto bodě už je ale nalezení korektní informace o datu vložení článku problémové, protože můžeme narazit na data vložení komentářů čtenářů v diskuzi, reklam, abstraktů s odkazy na jiné články a podobně. Kvůli tomu se ve výsledku může objevit velký počet nalezených shod a už je na uživateli aplikace, aby vybral korektní datum.

5.3.3 Hledání autora

Implementace algoritmu pro hledání autora článku se nachází v metodě *find_author*. Problém hledání autora je o trochu komplikovanější, než problém hledání data, protože se autorovo jméno nedá přesně vyjádřit regulárním výrazem. Jméno autora je na zpravodajských stránkách obvykle vyjádřeno standardním jménem a příjmením , přezdívkou a nebo zdrojem, ze kterého článek čerpal. Mohli bychom se pokusit hledat jméno na základě faktu, že samotné slovo křestního jména vždy začíná velkým písmenem, má obvykle počet znaků menší než deset, následuje mezera a slovo příjmení. Tato metoda by ale nenašla přezdívky typu „tyt“ a zdroje typu „ČTK“. Proto zde zavádíme trochu odlišný přístup, než při hledání data. Zavádíme proměnnou *score* počítající, s jakou pravděpodobností je nalezená informace o autorovi správná. Maximální dosažitelná hodnota skóre se pro jednotlivé hledané prvky (datum, autor, nadpis) liší. Hodnota velikosti udává, s jak velkou přesností je možné určit hledaný prvek (nadpis je možné určit velmi přesně, autora méně). Pokud je například skóre nalezené informace třicet ze třiceti, víme, že jsme s nejvyšší pravděpodobností našli správného autora. Naopak skóre jedna ze třiceti by znamenalo, že jsme sice nějakou informaci o autorovi našli, ale nejspíš nebude odpovídat pravému autorovi článku. Dále je definováno pět regulárních výrazů.

První z nich hledá, jestli se v atributech cesty vyskytuje výraz „author“, nebo „authors“. Stejně jako u data, vývojáři zpravodajských webových stránek takto obvykle označují tag, který vede k informaci o autorovi. Tato informace je klíčová pro správné nalezení autora a ostatní hledání autora se od této informace odvíjí. V prvních verzích aplikace byl zkoušen i český výraz „autor“, ale ten občas vykazuje špatné výsledky. Na některých zpravodajských portálech13 je například nadpis vyjádřen odkazem. To znamená zadání celé URL adresy do tagu <a>. Výraz „autor“ pak může najít nadpisy s tagem typu “a[@href='http://mujzpravodajskyportal.cz/zahranici/c1-56790-autor--tri-manzelky-byl-nalezen-mrtvy']“, což nechceme. Naštěstí, velká většina vývojářů českých zpravodajských stránek používá anglická označení pro tag vedoucí k informaci o autorovi. Český výraz „autor“ je nakonec zahrnutý také, protože je pořád lepší, když se nám ve výsledku objeví nesprávné informace, než kdyby se ty správné neměly objevit vůbec.

Druhý regulární výraz zjišťuje, jestli se v cestě nalezené prvním výrazem nachází slovo „photo“. Na zpravodajských stránkách se většinou nachází fotka související s článkem. Autor fotky je obvykle uveden v jejím popisu, ale toho najít nechceme. Když tento výraz nalezne shodu, víme, že se jedná o informaci o nesprávném autorovi a informaci ignorujeme. Podobně, třetí regulární výraz hledá shodu v textu s výrazem „Autor“, nebo „Autoři“. Na některých zpravodajských stránkách je tento text uveden před konkrétní informací o autorovi. Je ale obalen v jiném tagu, tedy má jinou cestu k textu, než samotný autor (i když popis tagu je shodný jako popis tagu autora). Ve výsledném hledání by se tento text objevil samostatně, jako další autor. Takové chování je nesprávné, proto tuto informaci opět ignorujeme.

Další dva regulární výrazy při shodě jen zvyšují proměnnou *score*, tedy zvyšují prav-

děpodobnost korektnosti nalezeného autora. V textu autora hledají velká počáteční písmena nebo výraz „ová“. V případě, že autorem je žena s českým jménem, její příjmení bude mít vždy koncovku „ová“, což je vcelku dobrý předpoklad o korektně nalezené informaci o autorce. Mužská příjmení se bohužel takto lehce kategorizovat nedají.

Samotný algoritmus je pak velmi podobný algoritmu hledání data. První vyhledávání je prováděno na průniku stránek, v případě úspěchu však cyklus nekončí. Vypíše a smaže nalezenou informaci a pokračuje dál, na hledání ve vyčištěné stránce. Důvod tohoto chování je ten, že v případě více autorů podílejících se na psaní článku se v průniku stránek může nacházet jen jeden. Přišli bychom tedy o ostatní autory. Ve výsledku se první nalezený už znovu neobjeví, protože prvek obsahující tuto informaci byl smazán. V případě neúspěchu je opět hledání rozšířeno na celou, nevyčištěnou stránku. Do výsledku se zároveň vypíše pravděpodobnost, nakolik je nalezená informace o autorovi správná.

5.3.4 Hledání nadpisu

Hledáním nadpisu článku se zabývá metoda s názvem *find_title*. Způsob hledání je velmi podobný metodě hledání autora. Opět je zavedena proměnná *score* a hledání se postupně provádí na průniku cest a v případě neúspěchu na vyčištěné a eventuálně nevyčištěné stránce. Z pozorování vyplývá, že nadpis článku se ve zpravodajských stránkách nachází na dvou místech. Prvním je samotný tag <title>, který je umístěn v hlavičce stránky. V tomto tagu je ale většinou uvedeno více informací, než jen samotný nadpis článku. Můžeme zde často nalézt i kořenovou adresu zpravodajského portálu nebo sekci, ve které se článek nachází (např. „domácí“). Druhým místem je tag <h1>, nazývaný “nadpis první úrovně”. Obvykle slouží pro název celé stránky a měl by se na stránce vyskytovat pouze jednou. Udává nám přesně tu informaci, kterou hledáme. Proto první definovaný regulární výraz vyhledává tento tag. Druhý výraz vyhledává, jestli se v atributech nalezené cesty vyskytuje výraz „logo“. V případě rozšířeného hledání na nevyčištěné stránce se může tag <h1> nacházet i v některých reklamách a poutačích, ale ty mají často v popisu některého ze svých tagů výraz „logo“, proto tyto shody ignorujeme. Nadpis obvykle začíná velkým písmenem, nekončí tečkou a obsahuje počet znaků větší než 20. Další regulární výraz a ověřování těchto tvrzení jen zvyšují pravděpodobnost správného výsledku. Pokud se na stránce vůbec nevyskytuje tag <h1> (stává se u blogů), pak metoda vrátí jako výsledek obsah tagu <title>.

5.3.5 Hledání článku

O hledání samotného článku se stará metoda *find_article*. Předpokládáme, že cesta ke článku je shodná pro všechny stránky ze zpravodajského portálu, proto se nutně musí objevit v průniku stránek. Dále předpokládáme, že počet shodných cest k textu článku je vyšší než počet shodných cest k jinému textu. Algoritmus metody nejdřív kontroluje, jestli se cesta s nejvyšším zastoupením opravdu vyskytuje v průniku stránek. Může nastat situace, kdy se na stránce vyskytuje rozsáhlá tabulka a samotný článek je krátký. Po-

tom by nejvyšší počet zastoupení měla cesta ukazující na jednotlivé prvky tabulky, ale už by se neobjevila v průniku stránek. Pokud tento případ nastane, metoda tuto cestu odstraní a pokračuje ve vyhledávání cesty s druhým nejvyšším počtem zastoupení. Po nalezení této cesty a potvrzení, že se vyskytuje i v průniku stránek metoda vezme v úvahu případné tagy modifikace textu a tag odkazu <a>. Na některých českých zpravodajských portálech je pravidlo, že se v textu článku používají pro některá slova odkazy. K těmto slovům se cesta liší pouze přítomností prvního rodičovského tagu, proto ho zohledníme. Slova zvýrazněná tagem *bold, strong* nebo *h2, h3, h4* se v textu článku vyskytují málokdy, ale pro úplnost je také zohledníme. Zohledníme také tag , objevující se v blozích.

6 Webové rozhraní

Webové uživatelské rozhraní bylo primárně vyvinuto pro účely testování a zkoušení aplikace. Má pouze základní funkcionalitu a základní grafiku [1]. Úvodní stránka obsahuje 5 textových polí, do kterých uživatel zadá 1-5 URL adres stránek ze stejné rubriky jednoho zpravodajského serveru nebo blogových článků od stejného autora. K dispozici je také několik předvyplněných vzorů. Poté uživatel stiskne tlačítko „Zpracovat“ a počká, než aplikace vyhledá žádané informace (datum, autora, nadpis a článek) a cesty k těmto informacím ve stromové struktuře HTML stránek. Na výsledné stránce jsou zobrazeny nalezené informace a jejich cesty. Uživatel má možnost pomocí zaškrtávacího pole pro každý záznam určit, která informace se uloží do výsledného XML souboru. Stiskem tlačítka „Uložit“ se požadované záznamy uloží do XML souboru a ten je nabídnutý uživateli ke stažení. Webové rozhraní je k dispozici na stránce `http://nlp.fi.muni.cz/projekty/web_structure_detection/`.

1. grafika webového uživatelského rozhraní byla implementována ve spolupráci s Mgr. Jaroslavem Kupčíkem

7 Experimentální vyhodnocení

Testování aplikace bylo provedeno na 15 webových portálech a 4 blozích. Portály byly vybírány na základě velikosti počtu návštěv za únor 2013 podle údajů z projektu NetMonitor [24]. Z každého portálu bylo náhodně vybráno 5 stránek ze stejné rubriky a předáno aplikaci na zpracování. Pro testování aplikace na blozích bylo vybráno 3 – 5 článků od stejného autora. Systém hodnocení byl následující:

Nalezeno více výsledků, než je skutečný počet (případ A): - Pokud se na stránce s jedním autorem našel jeden skutečný autor, úspěšnost nalezení autora byla ohodnocena 100 %. Pokud byli nalezeni dva autoři, dostala ohodnocení 50 %, pokud čtyři, ohodnocení bylo 25 %, atd. Ohodnocení vyplývá z poměru skutečný počet výsledků / nalezený počet výsledků. Toto hodnocení bylo použito, pokud se mezi nalezenými výsledky vyskytují všechny skutečné výsledky.

Nalezeno méně výsledků, než je skutečný počet (případ B): - Pokud na stránce jsou tři autoři a aplikace nenajde žádného, úspěšnost bude 0 %, pokud najde jednoho, dostane 33 %, pokud najde dva, dostane 66 %. Zde se počítá poměr nalezený počet výsledků / skutečný počet výsledků. Toto hodnocení předpokládá, že každý nalezený výsledek odpovídá jednomu skutečnému výsledku.

Pokud bylo nalezeno více výsledků, než je skutečný počet, a mezi nimi se nenalézají všechny správné, bylo použito toto hodnocení: případ A x případ B.
- Pokud je pro stránku se čtyřmi autory nalezeno osm autorů, ale pouze dva jsou správně, hodnocení bude $\frac{2}{4}$ x $\frac{4}{8}$ = $\frac{1}{4}$, tedy 25 %. Pokud je pro stejnou stránku nalezeno třicet autorů, ale pouze jeden je mezi nimi správný, hodnocení je $\frac{1}{4}$ x $\frac{4}{30}$ = $\frac{1}{30}$, tedy 3,33 %.
- Výsledné ohodnocení nalezení jedné informace pro portál je průměrem ohodnocení každé z 5 vybraných stránek. Pokud tedy pro novinky.cz je nalezení data na první stránce ohodnoceno 50 %, na druhé 80 % a na zbylých třech 100 %, výsledné ohodnocení nálezu data na serveru novinky.cz bude $\frac{(50+80+300)}{5}$ = 86 %.

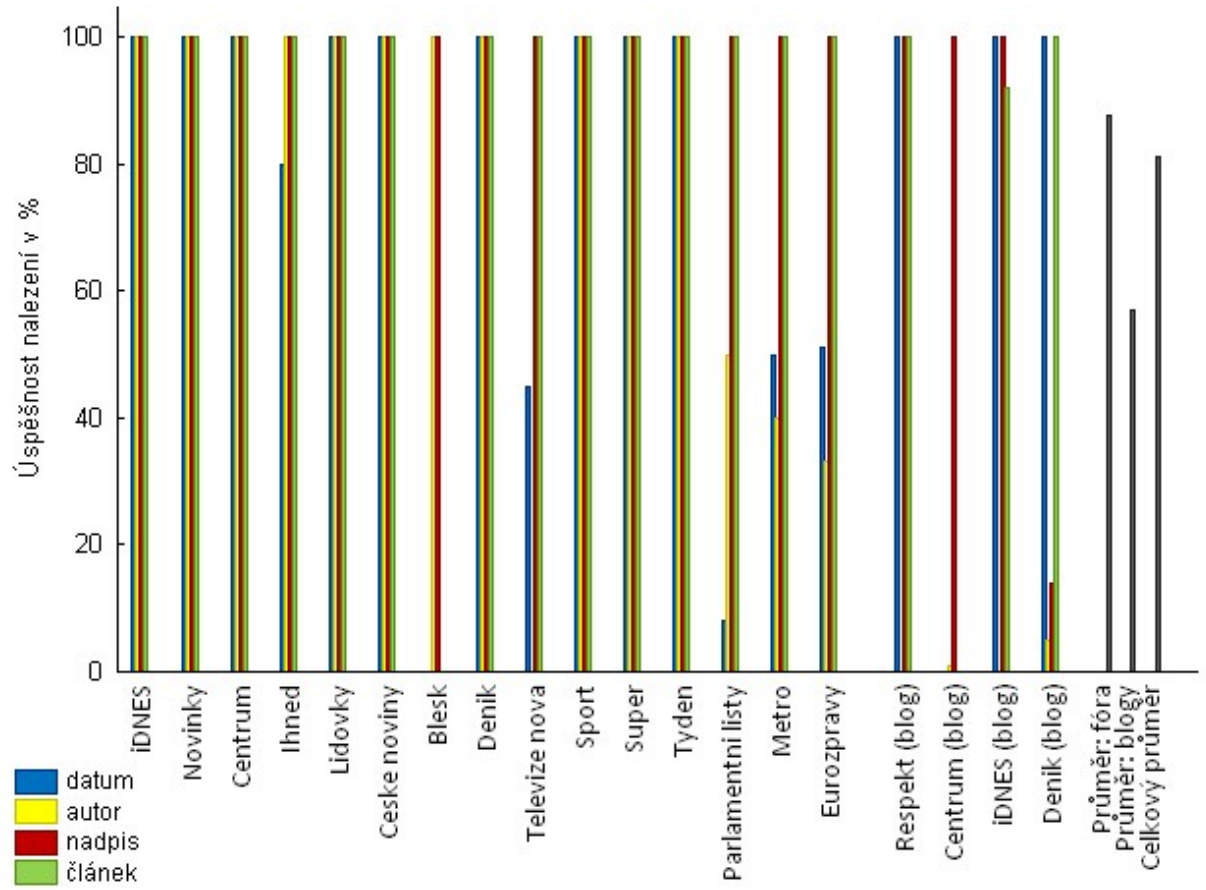
Toto hodnocení bylo aplikováno na hledání autora, data a nadpisu. Pro hledání článku bylo hodnocení velmi podobné. Počítal se počet nalezených oddílů článku oproti počtu celkových oddílů. Pokud jsou na stránce 4 odstavce textu a aplikace najde 3 odstavce, úspěšnost nalezení je 75 %. Pokud najde 5 odstavců, úspěšnost nalezení je 80 %. Pokud na stejné stránce najde 6 odstavců a z toho jsou 3 správné, úspěšnost bude 50 %.

Prvních šest portálů z tabulky 7.1 bylo použito pro zkoušení při vývoji aplikace, takže na nich byly přesné výsledky očekávány. I když nebyl zadán žádný požadavek na rychlost aplikace, v posledním sloupci tabulky 7.1 je uveden celkový čas výpočtu pro každý webový portál. Tento údaj může posloužit pro srovnání rychlosti s podobnými aplikacemi a pro budoucí vývoj a zkvalitňování aplikace.

Jak vidíme v grafu 7.1, výsledky úspěšnosti aplikace při testování na blozích zpravodajských portálů jsou řádově nižší, než na samotných stránkách portálů. Je to z toho

Tabulka 7.1: Výsledky testování

Portál	**datum (%)**	**autor (%)**	**nadpis (%)**	**článek (%)**	**čas (s)**
idnes.cz	100	100	100	100	14,73
novinky.cz	100	100	100	100	15,48
centrum.cz	100	100	100	100	21,88
ihned.cz	80	100	100	100	22,96
lidovky.cz	100	100	100	100	18,76
ceskenoviny.cz	100	100	100	100	5,4
blesk.cz	0	100	100	0	40,71
denik.cz	100	100	100	100	10,76
tn.cz	45	0	100	100	39,04
sport.cz	100	100	100	100	53,51
super.cz	100	100	100	100	8,89
tyden.cz	100	100	100	100	32,21
parlamentnilisty.cz	8	50	100	100	96,54
metro.cz	50	40	100	100	5,47
eurozpravy.cz	51	33	100	100	22,01
PRŮMĚR	87,61667 %				
blog.respekt.ihned.cz	100	0	100	100	4,6
aktualne.centrum.cz/blogy-a-nazory/	0	1	100	0	35
blog.idnes.cz	100	0	100	92	8,7
http://blog.denik.cz/	100	5	14	100	17,25
PRŮMĚR	57 %				
CELKOVÝ PRŮMĚR	81,171 %				



Obrázek 7.1: Diagram navrženého přístupu

důvodu, že autoři blogů nepíší každý svůj článek stejným způsobem a nepoužívají obvyklé způsoby označení prvků ve zdrojovém kódu, užívané vývojáři oficiálních webů.

V blozích na stránce `http://blog.respekt.ihned.cz/` je autor článku uveden ve stejném tagu jako datum článku, aplikace ho tedy najde spolu s datem a hledání autora nepřinese žádný výsledek.

Blogy stránky `http://blog.denik.cz/` jsou jediné dokumenty, na kterých aplikace nenalezne nadpis s naprostou přesností. Ten se zde nachází pouze v tagu <h2>. Aplikace tag <h2> hledá jako poslední, protože některé zpravodajské servery ho používají pro nadpis jednotlivých odstavců. Ve výsledku se ale už objevují i nadpisy různých jiných rubrik a rozcestníků.

Z testování vyplývá, že je potřeba v aplikaci zlepšit algoritmus hledání pro blogy. Kombinovaným hledáním na dvou typech stránek (zpravodajských portálů a blogů) dostáváme sice přijatelný výsledek, ale je zde prostor pro zlepšení. Přesnost tohoto výsledku lze zvýšit zavedením a implementací nového algoritmu zaměřeného specificky

na blogové hledání.

8 Závěr

Na internetu je k dispozici několik knihoven (případně aplikací, programů) pro čištění stránek, vyvinutých pro různé programovací jazyky. Pokud ovšem nechceme pouze vyčistit stránku, ale chceme také detekovat strukturu zbylých informací, pak máme jen několik málo možností. Při užším zaměření na české stránky a specifickými požadavky bychom takovou aplikaci hledali zbytečně [1]. Je nutné si tento typ aplikace vytvořit. Vývoj takové aplikace však není snadný úkol, protože neexistují obecně správné postupy pro hledání konkrétních informací.

Cílem této práce byl vývoj aplikace pro automatické rozlišení struktury textů ze zpravodajských serverů. Aplikace se zaměřuje na extrakci informací z českých zpravodajských portálů. Přehled aktuálních souvisejících výzkumů, použité postupy a proces vývoje aplikace jsou popsány v jednotlivých kapitolách práce. Vstupem aplikace je množina URL adres stránek ze stejné rubriky vybraného zpravodajského serveru, případně blogových článků od jednoho autora. Výstupem je nalezení cesty k datu, autorovi, nadpisu a samotnému článku pro každou vloženou stránku. Součástí aplikace je také jednoduché webové uživatelské rozhraní, umožňující výsledná data zkontrolovat, odstranit nežádoucí výsledky a konečný výstup uložit do souboru. Aplikace automatizuje proces vyhledávání konkrétních dat na zpravodajských stránkách a tím výrazně zjednodušuje a urychluje práci při budování korpusu. Tento typ aplikací je velmi nápomocný i pro vědecké a komerční účely a v následujících letech můžeme očekávat nárůst vývoje podobných knihoven, aplikací a programů.

Do budoucna by se aplikace dala rozšířit o vyhledávání na více typech informačních stránek a dokumentů. Do úvahy přicházejí diskuze, fóra, elektronická pošta, off-line dokumenty. Toto rozšíření bude vyžadovat využití přepínače umožňujícího nastavení způsobu hledání pro konkrétní typ vstupního dokumentu. Přínosem bude také intenzivní a podrobné testování aplikace. Dále je žádoucí zkvalitnění vzhledu webového uživatelského rozhraní, případně vývoj grafického uživatelského rozhraní. Možné je i budoucí zaměření na zpřesnění procesu vyhledávání konkrétních dat, rozšíření typů hledaných dat, doladění hledání dat pro portály s nízkou úspěšností nalezení korektní informace nebo snížení doby běhu aplikace.

---

1. jediná další nalezená práce je na `http://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=40352`

A Dokumentace

- recognizer.py

  – hlavní a jediný skript řídící celý proces extrakce, obsahuje třídu Recognizer.

Třída Recognizer obsahuje tyto metody:

- __init__(self, u_list = None)

  Tato metoda inicializuje instanci třídy Recognizer. Vstupním argumentem je seznam URL adres stránek k prohledání, defaultně nastaven na None. Inicializuje také proměnné použité k ukládání mezivýsledků metod - *processed_urls, cleaned_urls, divided_tagpaths, intersected_paths, short_full_string.*

- process_urls(self)

  Metoda pro parsování vstupních stránek. K parsování používá knihovnu Beautiful Soup. Odstraní komentáře v kódu, DOM stromy uloží do proměnné *processed_urls* a vrátí seznam s DOM stromy.

- clean_urls(self)

  Metoda odstraní boilerplate ze stránek pomocí knihovny Boilerpipe. Čisté stránky uloží do proměnné *cleaned_urls* a vrátí je jako seznam.

- divide_to_tagpaths(self, bs_pages)

  Vstupním argumentem metody je seznam DOM stromů. Metoda rozdělí text stránky podle jeho obklopujících tagů a pro každý textový oddíl najde jeho cestu ve stromové struktuře. Seznam cest a textů stránek uloží do proměnné *divided_tagpaths* a vrátí ho.

- count_paths(self)

  Metoda spočítá, kolikrát se na stránce vyskytuje shodná cesta k textu. Výsledný počet uloží k seznamu každé cesty a jejího textu jako další prvek a tento seznam vrátí.

- intersect_pages(self)

  Metoda provede průnik cest k textu čistých stránek. Výsledkem je slovník, jehož klíči jsou cesty k textu vyskytující se na všech vstupních stránkách. Metoda slovník uloží do proměnné *intersected_paths* a vrátí ho.

- short_paths_to_string(self, page)

  Vstupní argument metody je seznam cest k textu na stránce. Metoda převede cestu do podoby řetězce. Cesta je vyjádřena jen jménem tagu a mezi jmény je mezera. Vrátí seznam takto převedených cest a jejich textů.

- short_full_paths_to_string(self, divided_tagpaths)

  Vstupní argument metody je seznam stránek s cestami k textu a jejich textem. Metoda převede cesty na dva řetězce. První řetězec je cesta vyjádřená jen jménem tagu a mezi jmény je mezera. Druhý řetězec je cesta vyjádřená jménem tagu a jeho atributy a odpovídá stylu vyjádření cesty pomocí XPath [30]. Seznamy takto převedených cest pro každou stránku uloží jako seznam do proměnné *short_full_string* a vrátí ho. Metoda nezahrnuje do výsledku cesty s atributy označenými *onload, onchange, onclick.*

- find_date(self)

  Metoda pro hledání data na stránce. Nejdřív hledá data mezi cestami zahrnutými v průniku stránek. V případě neúspěchu rozšíří hledání postupně na celou, čistou stránku a na celou, nevyčištěnou stránku. Vrátí seznam s nalezenými daty a číslem pravděpodobnosti úspěchu.

- find_author(self)

  Metoda pro hledání autora. Nejdřív hledá autora mezi cestami zahrnutými v průniku stránek. V případě neúspěchu rozšíří hledání postupně na celou, čistou stránku a na celou, nevyčištěnou stránku. Vrátí seznam s nalezenými autory a číslem pravděpodobnosti úspěchu.

- find_title(self)

  Metoda pro hledání nadpisu. Nejdřív hledá nadpis v <h1> tagu mezi cestami zahrnutými v průniku stránek. V případě neúspěchu rozšíří hledání postupně na celou, čistou stránku a na celou, nevyčištěnou stránku. Pokud ani zde neuspěje, hledá nadpis v <title> tagu. Vrátí seznam s nalezenými nadpisy a číslem pravděpodobnosti úspěchu.

- find_article(self)

  Metoda pro hledání článku. Zkontroluje, jestli se cesta k textu, která se na stránce objevuje nejčastěji, nachází mezi cestami v průniku stránek. Pokud ne, cestu odstraní a kontroluje druhou nejčastější cestu. Po nalezení nejčastější cesty k textu zahrnuté v průniku stránek vezme do úvahy případné tagy modifikace textu <b>, , <h2>, <h3>, <h4>, tak odkazu <a> a tag . Nalezené textové oddíly vrátí v seznamu pro každou stránku.

- start(self)

  Metoda, která spouští hledání. Jako jediná je určená k volání z vnějšku třídy. Volá vnitřní metody podle daného pořadí. Vrátí seznam s nalezenými daty, autory, nadpisy a články.

Literatura

[1] Code google boilerpipe. URL http://code.google.com/p/boilerpipe/.

[2] Český národní korpus. URL ucnk.ff.cuni.cz/doc/slovnik-korpus.rtf.

[3] Beautiful soup, 2013. URL http://www.crummy.com/software/BeautifulSoup/.

[4] Vít BAISA. Web content cleaning [online]. Diplomová práce, Masarykova univerzita, Fakulta informatiky, 2009. URL DostupnézWWW<http://is.muni.cz/th/139654/fi_m/>.

[5] Marco Baroni, Francis Chantree, Adam Kilgarriff, and Serge Sharoff. Cleaneval: a competition for cleaning web pages, 2008.

[6] David Bremner, Erik Demaine, Jeff Erickson, John Iacono, Stefan Langerman, Pat Morin, and Godfried Toussaint. Output-sensitive algorithms for computing nearest-neighbour decision boundaries. 33:593–604–, 2005. ISSN 0179-5376.

[7] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. Extracting content structure for web pages based on visual representation, 2003.

[8] Jie Cheng and Russell Greiner. Learning bayesian belief network classifiers: Algorithms and system. In Eleni Stroulia and Stan Matwin, editors, *Lecture Notes in Computer Science*, volume 2056, pages 141–151–. Springer Berlin Heidelberg, 2001. URL http://dx.doi.org/10.1007/3-540-45153-6_14.

[9] Wikipedia contributors. Boilerplate code, . URL http://en.wikipedia.org/w/index.php?title=Boilerplate_code&oldid=546332220.

[10] Wikipedia contributors. Boilerplate (text), . URL http://en.wikipedia.org/w/index.php?title=Boilerplate_(text)&oldid=543609743.

[11] GregoryF Cooper and Edward Herskovits. A bayesian method for the induction of probabilistic networks from data. 9:309–347–, 1992. ISSN 0885-6125. URL http://dx.doi.org/10.1007/BF00994110.

[12] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites, 2001.

[13] Jana Friebelová. Rozhodovací stromy. URL http://www2.ef.jcu.cz/~jfrieb/rmp/data/teorie_oa/STROMY.pdf.

[14] Xiaoying Gao, Le Phong Bao Vuong, and Mengjie Zhang. Automatic data record detection in web pages, 2007.

[15] Hu Guohua and Zhao Qingshan. Study to eliminating noisy information in web pages based on data mining. In *Natural Computation (ICNC), 2010 Sixth International Conference on*, volume 2, pages 660–663–, 2010.

[16] Mingsheng Hu, Zhijuan Jia, and Xiangyu Zhang. An approach for text extraction from web news page. In *Robotics and Applications (ISRA), 2012 IEEE Symposium on*, pages 562–565–, 2012.

[17] Dávid Šimanský. Nástroj pro sledování změn obsahu webů [online]. Bakalářská práce, Masarykova univerzita, Fakulta informatiky, 2012. URL DostupnézWWW<http://is.muni.cz/th/359267/fi_b/>.

[18] Mei Kobayashi and Koichi Takeda. Information retrieval on the web. 32:144–173–, 2000. ISSN 0360-0300.

[19] Christian Kohlsch, 252, tter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features, 2010.

[20] Tomaž Kovačič. Overview: Extracting article text from html documents, 2011. URL http://tomazkovacic.com/blog/14/extracting-article-text-from-html-documents/.

[21] Tomaž Kovačič. *Evaluating Web Content Extraction Algorithms*. PhD thesis, University of Ljubljana, 2012. URL http://eprints.fri.uni-lj.si/1718/.

[22] Jing Li and C. I. Ezeife. Cleaning web pages for effective web content mining. In Stéphane Bressan, Josef Küng, and Roland Wagner, editors, *Lecture Notes in Computer Science*, volume 4080, pages 560–571–. Springer Berlin Heidelberg, 2006. URL http://dx.doi.org/10.1007/11827405_55.

[23] Bing Liu and Filippo Menczer. Web crawling. In *Data-Centric Systems and Applications*, pages 311–362–. Springer Berlin Heidelberg, 2011. URL http://dx.doi.org/10.1007/978-3-642-19460-3_8.

[24] NetMonitor. Netmonitor. URL http://www.netmonitor.cz/netmonitor-o-projektu.

[25] Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI-01 workshop on "Empirical Methods in AI"*, 2001. URL http://www.intellektik.informatik.tu-darmstadt.de/~{}tom/IJCAI01/Rish.pdf.

[26] Pavel Rychlý. *Korpusové manažery a jejich efektivní implementace*. Brno, 2000. URL http://www.fi.muni.cz/~pary/disert.ps. Disertační práce.

[27] StevenL Salzberg. C4.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. 16:235–240–, 1994. ISSN 0885-6125. URL http://dx.doi.org/10.1007/BF00993309.

[28] Erdinç Uzun, Hayri Volkan Agun, and Tarik Yerlikaya. A hybrid approach for extracting informative content from web pages. 49:928–944–, 2013. ISSN 0306-4573. URL http://www.sciencedirect.com/science/article/pii/S0306457313000332.

[29] Marta Vomlelová. Učení založené na instancích. URL http://kti.mff.cuni.cz/~marta/sliInstance.pdf.

[30] Le Phong Bao Vuong, Xiaoying Gao, and Mengjie Zhang. Data extraction from semi-structured web pages by clustering, 2006.

[31] W3C. Document object model (dom). URL http://www.w3.org/DOM/.

[32] W3Schools. Xpath tutorial. URL http://www.w3schools.com/xpath/.