VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ČIŠTĚNÍ, EXTRAKCE TEXTU A PŘEVOD WEBOVÝCH STRÁNEK DO VERTIKÁLNÍHO FORMÁTU

CLEANING, EXTRACTION OF TEXT AND TRANSFORMATION OF WEB PAGES INTO VERTICAL FORMAT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MILOŠ ŠVAŇA
AUTHOR

VEDOUCÍ PRÁCE Ing. JAROSLAV DYTRYCH
SUPERVISOR

BRNO 2016

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Švaňa Miloš**

Obor: Informační technologie

Téma: **Čištění, extrakce textu a převod webových stránek do vertikálního formátu**

Cleaning, extraction of text and transformation of web pages into vertical format

Kategorie: Web

Pokyny:

1. Seznamte se s dostupnými nástroji pro analýzu textu v přirozeném jazyce.
2. Seznamte se s dostupnými nástroji pro tvorbu korpusů ve Výzkumné skupině znalostních technologií.
3. Na základě získaných poznatků navrhněte a realizujte nástroj, který umožní čištění a vertikalizaci velkého množství dat z webu. Zaměřte se při tom na zachování odkazů a jejich kontextů a na rychlost a efektivitu zpracování.
4. Vyhodnoťte realizované řešení a porovnejte zvolený přístup z hlediska efektivity a integrovatelnosti.
5. Vytvořte stručný plakát prezentující práci, její cíle a výsledky.

Literatura:

- dle doporučení vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Funkční prototyp

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Dytrych Jaroslav, Ing.**, UPGM FIT VUT

Datum zadání: 1. listopadu 2015

Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Táto práca za zaoberá problematikou extrakcie textu z webových stránok, rozlíšením dôležitého obsahu a jeho prevodom do vertikálneho formátu, ktorý je vhodný na ďalšie spracovanie z pohľadu analýzy prirodzeného jazyka. Analyzuje existujúce riešenie a jeho komponenty so zameraním predovšetkým na jeho nedostatky a popisuje návrh a implemetáciu riešenia nového využívajúce získané znalosti.

Abstract

This thesis deals with the topic of extraction of text from web page, recognition of important contents and its transformation to vertical format, which can be used as a suitable input for other natural language processing tasks. It analyzes the existing solution and its components with emphasis on its disadvantages and describes the design and implementation of new solution based on obtained knowledge.

Kľúčové slová

Vertikalizácia, web, CommonCrawl, extrakcia textu, Justext, Boilerpipe, klasifikácia textu, spracovanie prirodzeného jazyka.

Keywords

Vertcalization, web, CommonCrawl, text extraction, Justext, Boilerpipe, text classification, natural language processing.

Citácia

ŠVAŇA, Miloš. *Čištění, extrakce textu a převod webových stránek do vertikálního formátu.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Dytrych Jaroslav.

Čištění, extrakce textu a převod webových stránek do vertikálního formátu

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Jaroslava Dytrycha. Ďalšie informácie mi poskytli Doc. RNDr. Pavel Smrž, Ph.D. a Ing. Lubomír Otrusina. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

. . . . . .
Miloš Švaňa
16. mája 2016

Poďakovanie

Na tomto mieste by som rád poďakoval členom členom Výzkumnej skupiny znalostních technológií, za poskytnutie dôležitých informácií potrebných k vypracovaniu tejto bakalárskej práce, a predovšetkým p. Ing. Jaroslavovi Dytrychovi za jeho cenné rady, trpezlivosť a priateľský prístup.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Kapitola 1

Úvod

Táto práca sa zaoberá návrhom a implementáciou nástroja na extrakciu dôležitého obsahu webových stránok a jeho transformáciou do vertikálneho formátu, ktorý má nahradiť súčasné riešenie majúce mnoho nedostatkov. Vertikálny formát je vhodný na ukladanie textových dát a ich ďalšiu analýzu.

Kapitola 2 zoznámi čitateľa s problematikou prevodu do vertikálneho formátu a súčasnou implementáciou so zameraním predovšetkým na problémy s ňou spojené. Na základe tejto analýzy sa zostavia požiadavky na nové riešenie.

V kapitole 3 je rozoberaná problematika získavania dôležitého obsahu z webových stránok. Predstaví dostupné nástroje a teoretické znalosti, na ktorých sú postavené. Jednotlivé riešenia porovnáva z hľadiska výkonu, kvality výstupu a možností modifikácie, na základe čoho sa z dostupných možností zvolí jedna, ktorá sa použije ako súčasť nového vertikalizátora.

Návrh štruktúry aplikácie je popísaný v kapitole 4. Čitateľa zoznámi s jednotlivými komponentami vertikalizátora a ich vzájomným prepojením. Stručný popis jednotlivých technológií použitých pri implementácii sa nachádza v kapitole 5.

Kapitola 6 sa venuje implementácii nového vertikalizátora. Popisuje hotové riešenie z pohľadu užívateľa a podrobnejšie rozoberá fungovanie komponentov, ktorých implementácia bola netriviálna, alebo sa odklonila od návrhu.

V kapitole 7 je čitateľ oboznámený so spôsobom a výsledkami testovania nového riešenia. Je porovnané s riešením súčasným z hľadiska výkonu, použitia pamäte a kvality výstupu. Tiež sú z hľadiska doby spracovania analyzované rôzne konfigurácie vertikalizátora.

Kapitola 8 na záver zhrňuje dosiahnuté výsledky, poukazuje na možné zlepšenia či rozšírenie funkcionality a naznačuje smerovanie ďalšieho vývoja.

Kapitola 2

Analýza súčasného stavu

Jedným z projektov, na ktorých pracuje Výzkumná skupina znalostních technologií (KNOT)[1] je spracovanie databázy internetových stránok z projektu CommonCrawl[2] , čo predstavuje desiatky až stovky terabajtov dát. Tieto dáta sú uložené vo formáte Web Archive, ktorého popisu sa táto kapitola venuje neskôr.

Prvou fázou spracovania je vytvorenie tzv. vertikálneho súboru, skrátene vertikálu, pomocou programu zvaného vertikalizátor. Vertikál je súbor, ktorý na každom riadku obsahuje jednu pozíciu, t. j. slovo, číslo, oddeľovač alebo štruktúrnu značku [3]. Textové dáta v tomto formáte sú následne ďalej spracovávané z hľadiska analýzy prirodzeného jazyka napr. nástrojmi na odstránenie duplicít, analýzu vetnej syntaxe a slovných druhov, či vytvorenie indexu pre vyhľadávanie.

Aby bolo spracovanie čo najrýchlejšie, vertikalizátor je spustený v niekoľkých procesoch súčasne na serveroch, ktoré má k dispozícii výskumná skupina, a na uzloch superpočítača Salomon[3].

Táto kapitola sa ďalej venuje analýze formátu vertikálneho súboru, možných vstupov, samotného procesu vertikalizácie, jeho súčasnej implementácii a jej nedostatkom, na základe čoho sú potom zostavené požiadavky na nové riešenie.

2.1 Formát vertikálneho súboru

Vertikál je textový súbor s názvom obyčajne končiacim príponou *.vert*, na ktorého každom riadku sa nachádza práve jedna pozícia - slovo, číslo, oddeľovač alebo štruktúrna XML značka. Na riadku sa za pozíciou môžu nácházdať ďalšie parametre oddelené tabulátorom [3]. Prehľad XML značiek používaných vo vertikále sa nachádza v tabuľke 1.

Formát vertikálu používaný výskumnou skupinou KNOT je inšpirovaný formátom používaným Centrom zpracování přirozeného jazyka na Fakulte informatiky Masarykovej univerzity[4]. [3] ďalej uvádza, že vertikálne súbory by bežne mali mať veľkosť v desiatkách kilobajtov, v tomto prípade sa ale jedná o desiatky až stovky megabajtov. [3] popisuje niekoľko ďalších štruktúrnych značiek definujúcich napr. zoznamy, tabuľky, básne či časti zdrojového alebo iného počítačového kódu. Tieto značky momentálne vertikál výskumnej skupiny KNOT neobsahuje. Pridáva však značky `<link>` a `<length>` s významom popí-

[1] http://knot.fit.vutbr.cz/
[2] https://commoncrawl.org/
[3] https://www.it4i.cz/
[4] https://nlp.fi.muni.cz/web3/

`<doc>`	Párová značka, obaľuje vertikál získaný z jednej webovej stránky, prípadne iného samostatného dokumentu. Má parametre `title` - titulok stránky, väčšinou získaný z HTML prvku `<title>`, a `url` - absolútna URL adresa webovej stránky.
`<head>`	Párová značka, obaľuje vertikalizovaný titulok webovej stránky, obsahovo zhodný s parametrom `title` značky `<doc>`. Nachádza sa vo vnútri prvku `<doc>`. Pri vertikalizácii iných typov vstupov, ktorým sa táto práca nevenuje, môže táto značka obsahovať ďalšie informácie o vertikalizovanom dokumente, napr. dátum vytvorenia.
`<p>`	Párová značka, obaľuje jeden odstavec textu.
`<s>`	Párová značka, obaľuje jednu vetu textu. Vety sa vždy nachádzajú vnútri odstavca.
`<g/>`	Nepárová značka, vkladá sa medzi dve pozície, ktoré v pôvodnom texte neboli oddelené žiadným prázdnym znakom, napr. medzerou. Príkladom použitia je bodka na konci vety, ktorá sa nachádza bezprostredne za posledným slovom.
`<link="URL">`	Nepárová značka, syntax zápisu nie je z pohľadu XML validná. Definuje odkaz. Reťazec `URL` predstavuje absolútnu adresu odkazu. Nachádza sa vždy na jednom riadku za posledkou pozíciou odkazu.
`<length=N>`	Nepárová značka, syntax zápisu nie je z pohľadu XML validná. `N` je prirodzené číslo, ktoré určuje počet pozícií, pred značkou `<link>` patriace do ňou definovaného odkazu. Do tohto počtu za nezapočítávajú štruktúrne značky. Nachádza sa na jednom riadku s poslednou pozíciou odkazu a značkou `<link>`.

Tabuľka 2.1: Prehľad značiek používaných vo vertikálnom formáte.

saným v tabuľke 2.1. Dôležitou modifikáciou je tiež používanie reťazca `__IMG__`, ktorým sa vo vertikáli nahrádza výskyt obrázka a slúži tak napríklad ako náhrada HTML značky `<img>`. Za týmto slovom sa podobne ako pri iných odkazoch nachádzajú značky `<link>` s absolútnou adresou obrázku a `<length>`, pričom dĺžka odkazu je v tomto prípade 1.

2.2 Formát Web Archieve (WARC)

Formát WARC slúži na ukladanie viacerých digitálnych zdrojov do jedného súboru spolu s ďalšími súvisiacimi informáciami. Jedná sa o revíziu formátu ARC používanom v rámci projektu Internet Achieve [5] [12]. CommonCrawl tento formát používa na ukladanie dokumentov, ktoré získava prechádzaním celého World Wide Webu. Formát je špecifikovaný štandardom ISO 28500 [11]. V ďalšom popise sa práca zameriava na aspekty špecifické pre WARC archívy získané z repozitára CommonCrawl, ktorú sú spracovávané vertikalizátorm.

[5] https://archive.org/index.php

Archív WARC je definovaný ako jednoduché spojenie viacerých WARC záznamov. Každý záznam obsahuje hlavičku podobnú nasledujúcej:

```
WARC/1.0
WARC-Type: request
WARC-Date: 2015-08-29T07:15:22Z
WARC-Record-ID: <urn:uuid:1b08c18f-7afc-4ba2-a3e6-474df5ccc174>
Content-Length: 270
Content-Type: application/http; msgtype=request
WARC-Warcinfo-ID: <urn:uuid:ccb85775-86f1-489f-9b23-fbc7dc4e891d>
WARC-IP-Address: 69.89.25.172
WARC-Target-URI: http://02186.mytownmatters.com/?p=23962
```

Hlavička začína riadkom `WARC/1.0` a následne obsahuje položky popisujúce daný záznam, napr. jeho typ, dátum vytvorenia, unikátne ID, dĺžka a typ obsahu, vždy jedna položka na riadok. Za hlavičkou sa nachádza jedným prázdnym riadkom oddelený samotný obsah záznamu, ten je nasledovaný dvoma prázdnymi riadkami. Záznamy môžu byť typu *request*, *response* a *warcinfo*. Prvým záznamom v rámci súboru je typu *warcinfo* a obsahuje informácie o WARC archíve ako celku, napr. meno programu, ktorým bol súbor vytvorený, autora súboru a podobne. Záznam typu request *obsahuje* telo HTTP požiadavku. Z hľadiska spracovania je najdôležitejší typ *response*, ktorý obsahuje HTTP odpoveď vrátane samoteného obsahu, teda napr. zdrojový kód v jazyku HTML alebo XML. Pre záznamy HTTP odpovedí ďalej platí, že sa v archíve nachádzajú vždy za zodpovedajúcim záznamom HTTP požiadavku.

2.3 Formát predspracovanej Wikipedie

Okrem súborov WARC je ďalším možným vstupom vertikalizátora súbor obsahujúci predspracované stránky z anglickej Wikipedie[6]. Takýto súbor má syntax podobnú XML a HTML, ale ako celok nie je validným dokumentom - nemá žiadny koreňový prvok. Jednotlivé stránky Wikipedie sa nachádzajú v prvkoch `<doc>`. Tento prvok má parametre `id` (unikátne id), `title` (titulok stránky) a `url` (absolútna URL adresa stránky). Vo vnútri tohto prvku sa nachádza samotný text stránky formátovaný pomocou základných HTML značiek pre nadpisy, odstavce, zoznamy, odkazy či tabuľky.

2.4 Súčasné riešenie

Činnosť vertikalizátora je pomerne zložitá. Nespočíva iba v prostom prevode z jedného formátu do druhého, dochádza aj k modifikácii samotného obsahu tak, aby výsledný súbor neobsahoval nechcené textové dáta. Celý proces popisuje táto podkapitola.

2.4.1 Načítanie vstupu

Pokiaľ je vstupom WARC súbor, je nutné získať telá jednotlivých HTTP odpovedí. Nie všetky odpovede sú však webové stránky v jazyku HTML, v archívoch sa často vyskytujú napr. iné XML súbory. Tieto sú vo výstupom vertikále neželané. WARC súbory môžu byť naviac komprimované napr. pomocou nástroja GNU zip[7]. Situácia je podobná aj v prípade

[6] https://en.wikipedia.org/wiki/Main_Page
[7] http://www.gzip.org/

predspracovanej Wikipedie - súbor je nutné rozdeliť na jednotlivé stránky.

2.4.2 Odstránenie nedôležitého obsahu

Veľmi dôležitým úkonom je odstránenie nedôležitých častí webovej stránky, ktorými sú rôzne menu, panely, reklamy, prihlasovacie formuláre či tlačidlá sociálnych sietí. Sekundárnym a tiež požadovaným efektom tohto procesu je odstránenie HTML značiek a získanie čistého textu. Tejto problematike sa práca podrobne venuje v kapitole 3.

Jednou z nevýhod súčasného riešenia je nutnosť predspracovania, keďže používaná knižnica má problém so spracovaním niektorých neštandardných znakov, či príliš dlhých riadkov v zdrojovom kóde webových stránok. Dôsledkom odstraňovania HTML značiek je tiež nutnosť pred vykonaním tohto procesu do zvláštneho zoznamu uložiť odkazy a obrázky, ktoré sú požadovanou súčasťou výsledného vertikálu.

2.4.3 Detekcia jazyka

Po odstránení nedôležitého obsahu, dochádza k detekcii jazyka. Výsledný vertikál by mal obsahovať iba texty vo zvolenom jazyku, všetok ostatný obsah je ďalej ignorovaný. Výskumná skupina sa momentálne zameriava predovšetkým na angličtinu.

2.4.4 Rozdelenie na pozície

Poslednou fázou spracovania je rozdelenie textu na jednotlivé pozície, pričom je nutné identifikovať odstavce a vety. Vo výslednom vertikálnom súbore budú reprezentované značkami `<p>` a `<s>`, ktorých význam je vysvetlený v podkapitole 2.2. Tiež sa obnovia uložené odkazy, do výsledného vertikálu sa vložia značky `<link>` a `<length>` a medzi pozície, ktoré neboli v pôvodnom texte oddelené žiadnym prázdnym znakom, sa vloží značka `<g/>`.

2.5 Problémy súčasného riešenia

Súčasný vertikalizátor má radu nedostatkov. V hľadiska nárokov na systémové prostriedky sa jedná hlavne o veľmi vysoké využitie operačnej pamäte, a to až v rádoch jednotiek gigabajtov. Táto skutočnosť výrazne obmedzuje počet paralelne bežiacich procesov vertikalizátora a to hlavne na uzloch superpočítača Salomon. Spracovanie jedného WARC archívu s veľkosťou 1GB zaberie približne 40 minút. Toto sa však nedá zatiaľ pokladať za nedostatok, keďže k vertikalizátoru so všetkými prídavnými funkciami neexistuje alternatíva na porovnanie.

Ďalším z problémov, ktorému sa práca venovala už v predchádzajúcej kapitole, je nutnosť predspracovania pred použitím nástroja na odstránenie nedôležitého textu. S týmto nástrojom súvisí tiež nemožnosť modifikácie tohto procesu, teda nemožnosť zasiahnuť do rozlišovania medzi dôležitým a nedôležitým textom. Z formálneho hľadiska problém predstavuje tiež štruktúra zdrojového kódu - jediný súbor, resp. trieda v programovacom jazyku Java s veľkosťou takmer 60 kilobajtov.

2.6 Požiadavky na nové riešenie

Riešením na nedostatky rozoberané v predchádzajúcej podkapitole je návrh a implementácia nového vertikalizátora. Požiadavky na nové riešenie boli spísané na základe rozhovorov

s pracovníkmi výskumnej skupiny KNOT a vlastnej analýzy.

2.6.1 Vstup a výstup

Základným požiadavkom je pochopiteľne schopnosť spracovávať rovnaké typy vstupov, teda archívy WARC, ktoré môžu byť komprimované pomocou nástoja GZip či algoritmu LZMA[8], predspracované stránky Wikipedie a zdrojový kód jednej samostatnej webovej stránky v jazyku HTML. Výstup by mal dodržiavať pravidlá pre vertikálny formát definované v kapitole 2.2. Nie je však nutné, aby súčasné aj nové riešenie vracali pre rovnaký vstup aj rovnaký výstup. Nový vertikalizátor by mal vytvárať vertikál kvalitnejší, čo je však veľmi ťažko merateľná a subjektívna metrika. Kvalitou sa v tomto poňatí rozumie výber vhodného obsahu z textových dát na vstupe. Otázne je, či za kvalitnejší výstup považovať taký, ktorý obsahuje iba dôležitý obsah, no jeho značná časť mohla byť vertikalizátorom odstránená, alebo výstup obsahujúci väčšinu dôležitého textu ale aj časti obsahu nedôležitého.

2.6.2 Systémové nároky

Ďalšie požiadavky predstavujú vyriešenie nedostatkov pôvodného riešenia, teda z hľadiska systémových nárokov zníženie množstva využívanej operačnej pamäte minimálne o polovicu a podľa možností zníženie času transformácie vstupu na vertikány súbor.

2.6.3 Úprava funkcionality

Čo sa týka možnosti modifikácie procesu rozlišovania medzi dôležitým a nedôležitým obsahom, treba navrhnúť a implementovať univerzálne riešenie. Jednou z možných modifikácií, ktoré bolo navrhnuté pracovníkmi výskumnej skupiny, je pokladanie častí webovej stránky obsahujúcich odkazy na Wikipediu za dôležitý obsah a to i v prípade, že nástroj na to určený by bez úprav tento obsah odstránil. Tiež je vhodné pokúsiť sa nájsť alternatívu nástroja, ktorá by nevyžadovala alebo zjednodušila predspracovanie pred jeho samotným použitím.

[8] https://en.wikipedia.org/wiki/LZMA

Kapitola 3

Odstránenie nedôležitých častí webových stránok

Jednou z najdôležitejších a najzložitejších úloh vertikalizátora je odstránenie nedôležitého obsahu. Jedná sa o časti webových stránok, ktoré sa v rámci jedného webového portálu takmer nikdy nemenia, napr. navigácia, reklamy, prihlasovací či vyhľadávací formulár, tlačidlá a zásuvné moduly sociálnych sietí a podobne. Na obrázku 3.1 sú na príklade webovej stránky www.novinky.cz zelenou farbou zvýraznené dôležité časti. Ostatný obsah je z hľadiska ďalšieho spracovania nepodstatný, či dokonca škodlivý. Ako uvádza [4], jeho zachovanie môže napríklad znižovať relevanciu výsledkov vyhľadávania.

Klasifikácia dôležitého resp. nedôležitého obsahu sa môže vykonávať na základe veľkého množstva príznakov. Tie môžu byť podľa [5] získavané na štyroch rôznych úrovniach:

- úrovni individuálnych textových blokov (prvkov webovej stránky),
- celého HTML dokumentu,
- vykreslenej webovej stránky v grafickej podobe,
- webového portálu ako zoskupenia webových stránok s rovnakou šablonou a opakujúcími sa prvkami.

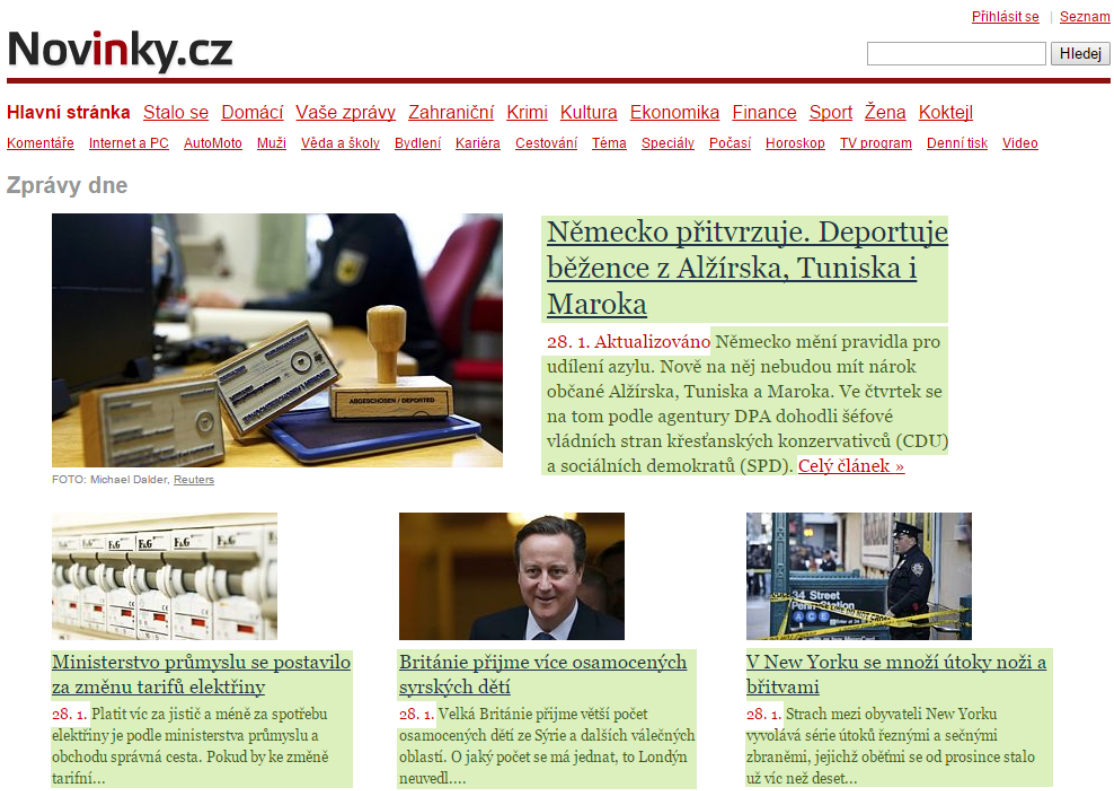
Posledné dve úrovne pokladá [5] za problematické. Vykreslenie webovej stránky je výpočtovo náročná operácia a je otázne, či posledný menovaný prístup nie je závislý na doméne. Spoločné znaky v rámci webového portálu sa musí pre každý prípad klasifikátor naučiť osobitne a môžu len ojedinele fungovať pre prípady iné.

3.1 Dostupné riešenia

Táto podkapitola postupne predstavuje niekoľko hotových riešení na extrakciu dôležitého obsahu, ktoré [4] považuje za relevantné, a analyzuje vhodnosť ich použitia vo vertikalizátore z hľadiska spôsobu klasifikácie.

3.1.1 Algoritmus Body Text Extraction (BTE)

Algoritmus vychádza z dvoch predpokladov: relevantná časť HTML dokumentu sa sa zväčša nachádza v jednom súvislom úseku a oproti nedôležitým častiam, charakterizovaným v úvode



Obr. 3.1: Dôležité časti webovej stránky.

tejto kapitoly, obsahuje menšie množstvo HTML značiek. Hľadajú sa teda najdlhšie úseky textu s minimom HTML značiek. [4] hovorí, že toto riešenie nie je veľmi presné v situáciách, kedy dôjde k prerušeniu textu reklamou alebo sa relevantné informácie nachádzajú v tabuľke, ktorej zdrojový kód má vysokú hustotu HTML značiek.

Z popisu algoritmu usudzujem, že toto riešenie by na druhú stranu mohlo dobre fungovať pre webové stránky niektorých spravodajských serverov s jednoduchou štruktúrou. Nie je však vhodné pre použitie vo vertikalizátore, pretože jeho vstup nie je obmedzený len na tento typ obsahu.

3.1.2 Boilerpipe

Boilerpipe pri klasifikácii obsahu používa už spomínané povrchné textové príznaky. Algoritmus bol trénovaný predovšetkým na textoch zo služby Google News[1], pričom sa testovali rôzne kombinácie príznakov. Podľa experimentov najlepšie výsledky boli dosiahnuté pri použití kombinácie príznakov hustoty textu a hustoty odkazov.

Toto riešenie je použité aj v súčasnom vertikalizátore. Odhliadnúc od kvality výstupu a presnosti samotného algoritmu jeho implementácia v podobe knižnice pre jazyk Java sa nedá považovať za velmi kvalitnú. Vstup je nutné pred použitím knižnice očistiť, problém spôsobujú napríklad niektoré príliš dlhé slová, riadky či špeciálne znaky.

[1] https://news.google.com/

3.1.3 Justext

Justext vytvorilo Centrum zpracování přirozeného jazyka na Fakulte informatiky Masarykovy univerzity. Algoritms delí obsah webových stránok na odstavce na základe výskytu HTML značiek, ktoré sa bežne používajú na rozdelenie obsahu do menších logických celkov, čím sa snaží napodobovať identifikáciu vizuálne oddelitelných blokov pri pohľade na vykreslenú webovú stránku. Každý takýto odstavec je potom zaradený do jednej z tried *bad*, alebo *good*, pričom trieda *bad* označuje nedôležitý obsah. Klasifikácia prebieha v dvoch fázach.

Prvá fáza

V prvej fáze sa klasifikujú jednotlivé odstavce nezávislo na okolí na základe nasledujúcich príznakov:

- dĺžka textu,
- hustota odkazov,
- hustota tzv. stop slov, teda najpoužívanejších slov daného jazyka.

Posledný spomenutý príznak nie je jazykovo nezávislý a je preto nutné pred samotnou klasifikáciou poznať (alebo si zvoliť) jazyk dokumentu a poskytnúť odpovedajúci zoznam stop slov. Rozhodovacie hranice pre jednotlivé príznaky sú konštantné a boli určené experimentálne, je však možné ich užívateľský nastaviť. V prvej fáze nie sú všetky odstavce nutne konečne označené ako *bad* alebo *good*, môžu sa zaradiť do triedy *short* alebo *near-good* a následne postupujú do druhej fázy klasifikácie.

Druhá fáza

V druhej fáze sa do úvahy berie kontext klasifikovaného odstavca, teda okolité odstavce. Dochádza ku konečnému rozhodnutiu pre odstavce označené triedy *short* a *near-good*, pričom sa vychádza zo skúsenosti, že dôležité bloky sú obklopené inými dôležitými blokmi a nedôležité bloky sú obklopené inými nedôležitými blokmi. Aplikujú sa pravidlá typu *ak sa pred aj za odstavcom triedy near-good nachádza odstavec triedy good, zaradí sa aj práve klasifikovaný odstavec do triedy good* [8].

3.2 Porovnanie nástrojov

Z riešení prezentovaných v predchádzajúcej podkapitole som sa rozhodol brať ďalej v úvahu Justext a Boilerpipe. Algoritmus BTE som pre spomínané nedostatky z ďalšieho porovnávania vylúčil.

3.2.1 Kritériá hodnotenia

Pri rozhodovaní, ktorý nástroj je vhodné použiť ako súčasť vertikalizátora, zohrávajú úlohu predovšetkým nasledujúce kritériá:

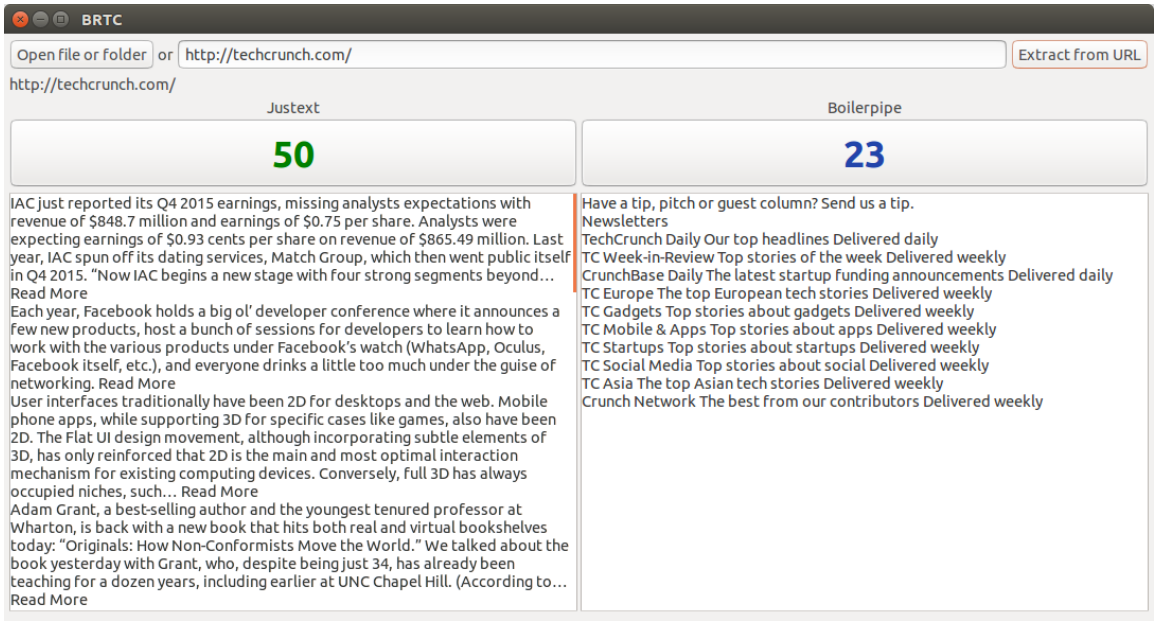
- rýchlosť spracovania,
- odolnosť voči problémovým vstupom,

- možnosť modifiácie,
- kvalita výstupu.

Z týchto kritérií je možné objektívne porovnať nástroje iba z hľadiska prvých dvoch spomenutých. Kvalita výstupu i možnosť modifikácie sú metriky veľmi subjektívne.

3.2.2 Spôsob porovnávania

Za účelom porovnania nástrojov som vytvoril jednoduchú aplikáciu, ktorá spracuje rovnaký vstup oboma nástrojmi, zobrazí výsledky užívateľovi, a ten má následne možnosť udeliť bod jednému alebo druhému nástroju na základe toho, ktorý výstup považuje za kvalitnejší. Pokiaľ oba nástroje vrátia výstup rovnaký, nemusí užívateľ udeliť bod ani jednému. Ak jeden z nástrojov zlyhá, bod je pridelený automaticky jeho konkurentovi. Aplikácia zároveň na pozadí počíta čas spracovania oboma nástrojmi, ktorý spolu s užívateľom udeleným skóre akumuluje a ukladá do súboru vo formáte JSON.



Obr. 3.2: Snímka obrazovky porovnávacej aplikácie.

3.2.3 Výsledky porovnania

Nástrojom popísaným v predchádzajúcom odstavci som prorovnal výstupy pre celkovo **300** webových stránok s výsledným skóre **102** pre Boilerpipe a **156** pre Justext. Skóre je subjektívne, pričom som uprednostňoval odstránenie nedôležitého textu pred zachovaním dôležitého. Súčet skóre je 258, pri zvyšných vzorkoch nebol bod udelený ani jednému výsledku, pretože boli totožné alebo veľmi podobné.

Čas spracovania som porovnával pomocou rovnakého nástroja, k vzorkom použitým na testovanie kvality výstupu som pridal ďalších **98**, pri ktorých som skóre určoval náhodne, dôležitá bola už iba doba spracovania. Knižnici Boilerpipe trvalo spracovanie **1 minútu a 44 sekúnd** - v priemere **0,261 sekundy** na súbor. Justext súbory spracoval za **22 sekúnd** - v priemere **0,056 sekundy** na súbor.

Testovacia sada obsahovala webové stránky náročné na rozpoznanie dôležitého obsahu, na ktorých chceli pracovníci výskumnej skupiny uskutočniť kvalitatívne porovnanie oboch nástrojov.

Porovnanie možností modifikácie bolo tiež subjektívne. Ako lepší nástroj som opäť zvolil Justext, v ktorého zdrojovom kóde sa veľmi rýchlo dali vyhľadať body, kde by sa dala modifikovať výsledná trieda odstavca podľa požiadavkov užívateľa. Rolu zohrali aj väčšie osobné skúsenosti s jazykom Python, v ktorom je Justext implementovaný.

Kapitola 4

Návrh

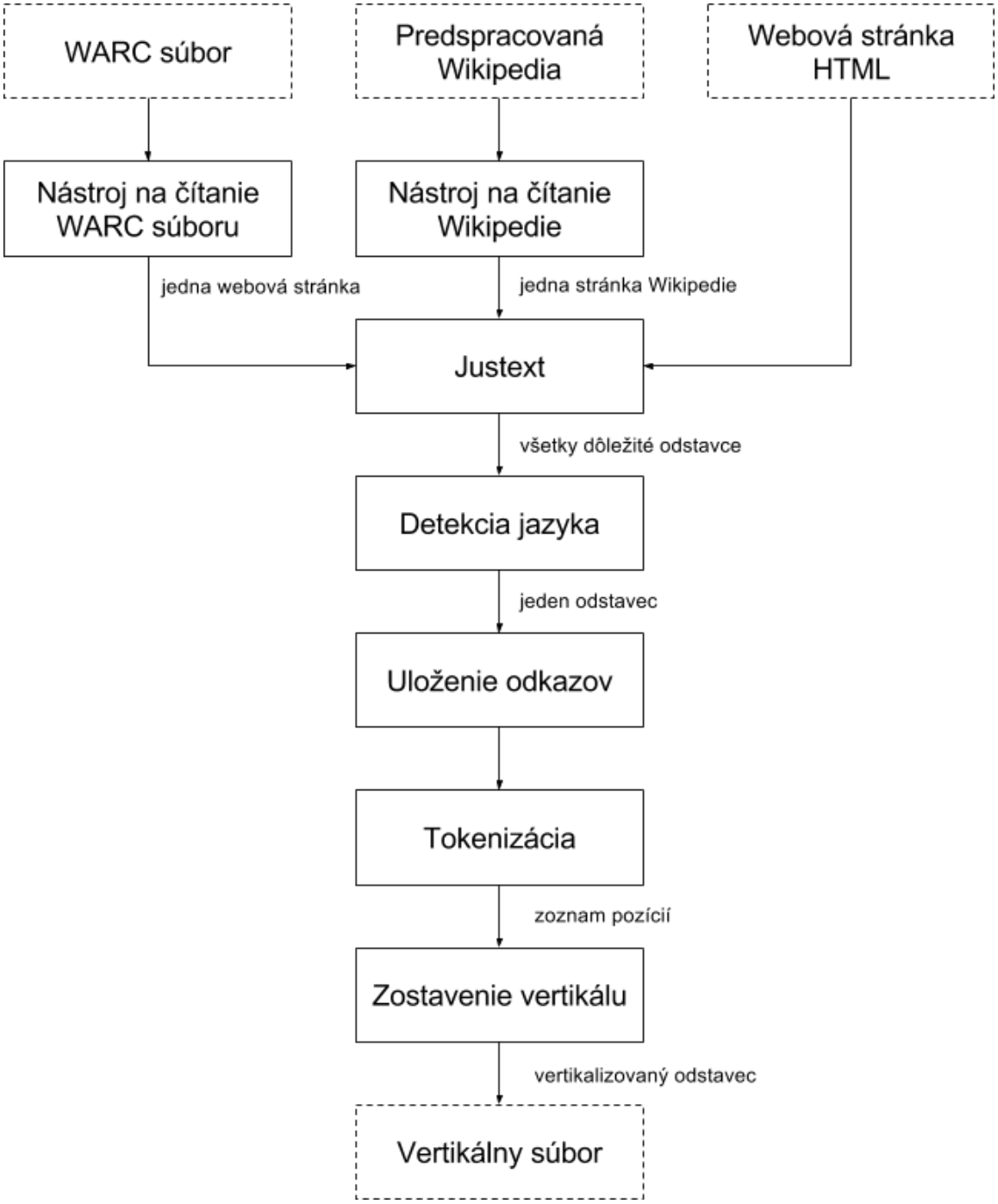
Celý vertikalizátor je navrhnuntý ako sada nástrojov, ktoré svoj výstup predávajú ďalšiemu nástroju podobne ako rúry[1] v operačných systémoch na báze UNIXu.

Architektúra z hľadiska toku dát a prepojenia jednotlivých komponentov je znázornená na obrázku 4.1. Vstup, v prípade, že sa jedná o archív WARC, sa predá nástroju na čítanie týchto archívov, ktorý postupne bude prechádzať jednotlivé webové stránky. Ich zdrojový kód sa následne predá nástroju Justext, ktorý z nich získa dôležitý obsah. Následne sa detekuje jazyk, a v prípade, že sa jedna o jazyk požadovaný, bude obsah ďalej spracovaný tokenizerom, ktorý ho rozdelí na jednotlivé pozície, z ktorých sa vytvorí finálny vertikál.

Ako hlavný implementačný jazyk bol zvolený Python. Hlavným dôvodom pre tento výber bola predovšetkým exitujúca implementácia algoritmu Justext práve v tomto jazyku, úlohu však zohrala aj väčšia osobná skúsenosť.

Kapitola postupne rozoberá návrh jednotlivých komponentov zobrazených na obrázku 4.1 s dôrazom kladeným na funkčné požiadavky a rozhranie.

[1] http://www.linfo.org/pipes.html



Obr. 4.1: Tok dát vo vertikalizátore a prepojenie komponentov.

4.1 Reprezentácia webovej stránky

Vstupom pre samotný proces vertikalizácie je zdrojový kód webovej stránky, je ho nutné získavať aj z WARC súborov a súborov predspracovanej Wikipédie. Okrem zdrojového kódu je nutné uchovávať aj ďalšie informácie: URL adresu a voliteľne titulok získaný napríklad z HTML značky `<title>`. Obe tieto informácie sú súčasťou výsledného vertikálneho súboru,

ako parametre prvku `<doc>`.

Každá stránka bude reprezentovaná ako objekt triedy `Webpage` obsahujúci nasledujúce položky:

- `payload`: samotný zdrojový kód webovej stránky,
- `url`: absolútna URL adresa,
- `title`: nepovinne titulok, webová stránka ho nemusí obsahovať.

4.2 Čítanie súboru WARC

4.2.1 Dostupné riešenia

V repozitári Python Package Index[2] sa nachádza niekoľko hotových riešení na prácu so súbormi WARC. Z ponúkaných možností sa vybrali 3: *warc*[3] , *warcat*[4] a *warctools*[5] . K poslednej menovanej voľbe nebola k dohľadaniu dokumentácia na použitie vo forme knižnice, preto bola z ďalšieho porovnávania vylúčená.

Na porovnanie zvyšných dvoch možností sa vytvoril jednoduchý program v jazyku Python, ktorého úlohou bolo pomocou daného nástroja získať zo testovacieho WARC archívu telá všetkých HTTP odpovedí. Knižnica Warc prečítala zo súboru niekoľko stoviek z približne 30 000 požadovaných záznamov, potom prechádzanie ukončila bez chybovej hlášky. Nástroj Warcat prečítal záznamy všetky, bol teda z hľadiska rýchlosti otestovaný na niekoľkých ďalších WARC archívoch. Zároveň sa pre porovnanie rovnaké archívy spracovali aj riešením použitým v súčasnom vertikalizátore. Výsledky meraní sa nachádzajú v tabuľke 4.1.

Testovací súbor	**Warcat (Python)**	**WarcReader (Java)**
58-warc.gz	5m 41s	2m38s
59-warc.gz	5m 49s	2m40s
65-warc.gz	5m 29s	2m32s
91-warc.gz	5m 32s	2m31s
102-warc.gz	5m 28s	2m32s

Tabuľka 4.1: Rýchlosť spracovania WARC súborov nástrojom Warcat.

Z tabuľky si je možné všimnúť, že čas spracovania je oproti súčasnému riešeniu viac ako dvojnásobný, čo sa nedá považovať za uspokojivé.

4.2.2 Vlastné riešenie

Keďže žiadne z vyššie spomínaných riešení nedosahovalo požadovanú rýchlosť a spoľahlivosť, vznikla nutnosť vytvoriť riešenie vlastné. Toto bude implementované ako trieda WarcReader, ktorej objektom bude pri inicializácii ako parameter poskytnutá cesta k WARC súboru.

[2] https://pypi.python.org/pypi
[3] https://pypi.python.org/pypi/warc/0.2.1
[4] https://pypi.python.org/pypi/Warcat/2.2.3
[5] https://pypi.python.org/pypi/warctools/4.9.0

Objekty triedy budú disponovať metódou, ktorá pri každom zavolaní poskytne z archívu jednu webovú stránku ako inštanciu triedy Webpage.

Treba tiež pripomenúť požiadavok na zníženie pamäťových nárokov. Z tohto dôvodu by sa v pamäti nemal nachádzať celý WARC súbor, resp. zoznam obsahujúci všetky webové stránky, ktorý sa postupne prechádza.

4.3 Čítanie súboru s predspracovanou Wikipediou

Na čítanie jednotlivých stránok zo súborov predspracovanej Wikipedie neexistuje s výnimkou starého vertikalizátora, ktorý sa práca snaží nahradiť, žiadne hotové riešenie.

Formát je však pomerne jednoduchý, jednotlivé stránky sa nachádzajú v prvku `<doc>`, ktorého začiatočná a ukončovacia značka sa nachádzajú na samostatnom riadku. Riešením bude teda čítanie súboru po riadkoch, ukladanie týchto riadkov do jedného reťazca a vytvorenie nového objektu triedy Webpage vždy po prečítaní riadku s ukončovaciou značkou `</doc>`. Po vzniku tohto objektu sa z ďalších riadkov začne vytvárať nový reťazec. Rovnako ako v prípade čítania zo súboru WARC je nutné minimalizovať pamäťové nároky tým, že v operačnej pamäti bude k dispozícii práve jedna stránka Wikipedie.

4.4 Čítanie súboru HTML

Na načítanie obsahu súboru HTML, teda jednej samostatnej webovej stránky, nie sú potrebné žiadne špeciálne postupy. Použijú sa nástroje dostupné v rámci štandardnej knižnice.

4.5 Justext a modifikácia klasifikácie obsahu

Existujúca implementácia algoritmu Justext má jednoduché rozhranie: ako vstup očakáva zdrojový kód webovej stránky a zoznam stop slov, ako výstup vráti zoznam všetkých získaných odstavcov a ich zaradenie do tried *good* alebo *bad*.

Jedným z požiadavkov na nové riešenie je možnosť modifikácie procesu klasifikácie. Ako vhodné miesto pre zásah do pôvodnej implementácie bolo zvolené ukončenie druhej fázy. Po jej skončení sa pre každý odstavec zavolá užívateľom určená funkcia, ktorá ako vstup príjma text, triedu pridelenú pri klasifikácii a zodpovedajúci úsek zdrojového kódu v jazyku HTML. Táto funkcia musí následne vrátiť novú triedu a (prípadne pozmenený) text odstavca.

Ďalším zásahom do algoritmu Justext bude pridanie možnosti pre zachovanie všetkých odstavcov, čo je využiteľné pri spracovaní stránok Wikipedie, kde sa predpokladá, že všetok obsah je dôležitý. Intuitívnym riešením by bolo ďalej spracovávať aj odstavce klasifikované ako bad, tento prístup však nie je efektívny, keďže klasifikácia sa stále vykoná a dôjde teda k plytvaniu výpočtovým výkonom. Po aktivovaní tejto voľby by sa mal preto algoritmus zastaviť už po extrahovaní textu jednotlivých odstavcov.

Samotný Justext je dostupný v dvoch implementáciách – jedna v jazyky Python, druhá v jazyky C++. Pred akoukoľvek modifikáciu bude nutné porovnať obe možnosti z hľadiska doby spracovania, je však nutné mať už implementované čítanie WARC súborov.

4.6 Detekcia jazyka

Na detekciu jazyka sa použije už hotové riešenie *langid*. Toto poskytuje výstup v podobe jednoduchého textového identifikátora najpravdepodobnejšieho jazyka prípadne kompletný zoznam dvojíc *jazyk - pravdepodobnosť*. Na účely detekcie anglického textu postačí prvá popísaná možnosť. Ďalšie skúmané nástroje riešiace tento problém neboli schopné spracovať niektoré špeciálne znaky v texte, alebo sa jednalo o rozhranie pripájajúce sa na vzdialený server komerčného poskytovateľa. Predpokladám, že sieťová komunikácia by vytváranie vertikálneho súboru značne spomalila. Otázne by boli aj podmienky prevádzkovateľa služby.

Spôsob klasifikácie algoritmu Justext smeruje k úvahám, či je vôbec nutné jazyk detekovať samostatným nástrojom. Na základe poskytnutého zoznamu stop slov môže byť veľká väčšina textov v inom ako požadovanom jazyku označená ako bad a teda vylúčená z vertikálneho súboru. Túto myšlienku je však nutné experimentálne overiť.

4.7 Uloženie odkazov a obrázkov

Ako už bolo spomenuté v kapitole 2, výstup vertikalizátora by mal obsahovať URL adresy odkazov a obrázkov z pôvodnej webovej stránky. Súčasné riešenie pomocou regulárnych výrazov tieto extrahuje a ukladá do zoznamu. K tomuto procesu dochádza ešte pred samotnou klasifikáciou dôležitého a nedôležitého obsahu, čo má za následok nadbytočné využitie pamäte, keďže veľká časť uložených odkazov a obrázkov sa do výsledného vertikálu neuloží z dôvodu oklasifikovania zodpovedajúceho odstavca ako *bad*.

Pre nové riešenie tento fakt predstavuje priestor na zníženie nárokov na pamäť. K uloženiu odkazov dôjde až po vyradení nedôležitého obsahu a detekcii jazyka. Je však nutná ďalšia úprava implementácie algoritmu Justext tak, aby umožňoval zachovanie HTML značiek `<a>` a `<img>`.

Celý úkon je nutné vykonať predovšetkým kvôli nadväzujúcemu rozdeleniu na pozície, ktoré by sa aplikovalo aj na HTML značky a rozložilo ich na menšie časti.

4.8 Rozdelenie na pozície

Aj v tomto prípade sa použije už existujúce riešenie, konkrétne modul tokenizer z knižnice NLTK[6]. Keďže výsledný vertikálny súbor má obsahovať informáciu o rozdelení textu do viet, musí tento proces prebiehať v dvoch fázach: vo fáze prvej dôjde k rozdeleniu textu odstavca na vety, vo fáze druhej sa text viet rozdelí na jednotlivé pozície. NLTK poskytuje funkciu pre oba kroky. Toto riešenie bolo zvolené kvôli v porovnaní s ďalšími možnosťami kvalitnejšej dokumentácii a poskytnutiu viacerých alternatív pri delení viet na jednotlivé pozície.

Z jednotlivých pozícií je ďalej nutné odstrániť alebo nahradiť niektoré nežiadúce znaky, napríklad riadiace znaky či symbol `|` anglicky označovaný ako *vertical bar*, ktorého výskyt spôsobuje problémy pri vytváraní indexu pre plnotextové vyhľadávanie. Z tohto dôvodu sa nahrádza za znak `¦`, označovaný ako *broken bar*. Táto funkcionalita sa dá implementovať pomocou regulárnych výrazov.

[6] http://www.nltk.org/

4.9 Zostavenie vertikálu

Zostavenie textového reťazca, ktorý sa uloží do vertikálneho súboru, bude implementovať trieda *Verticalizator*. Táto bude riadiť celý priebeh vertikalizácie jednej webovej stránky predanej ako objekt triedy *Webpage*.

Vertikál bude vytváraný po menších častiach postupne pripájaných k pribežnému výsledku, nie ako jeden celok po ukončení všetkých predchádzajúcich procesov. Ako prvá bude vytvorená začiatočná značka `<doc>`, s parametrami `title` a `url` nastavenými podľa hodnôt uložených v objekte reprezentujúcom webovú stránku. Následne sa na jednotlivé pozície rozdelí titulok a uzavretý do prvku `<head>` sa pridá k začatému reťazcu. Pokračuje sa spracovaním obsahu webovej stránky nástrojom Justext. Pre každý vrátený odstavec za uložia odkazy a obrázky, dôjde k rozdeleniu na pozície a vytvoreniu jeho vertikalizovanej podoby, pričom celý odstavec bude uzavretý v prvku `<p>` a jednotlivé vety v prvku `<s>`. Medzi pozície, ktoré neboli v pôvodnom texte oddelené žiadnym prázdnym znakom, sa vloží značka `<g/>` a za posledné pozície odkazov sa doplnia potrebné parametre. Celý proces by mal prebiehať tak, že k priebežne zostavovanému reťazcu sa budú pridávať nové riadky, zásah do už zostavenej časti vertikálu je nežiadúci.

Predchádzajúci odstavec popisoval zostavenie vertikálu pre jednu webovú stránku. Vertikalizátor tento postup vykoná pre každý záznam načítaný zo vstupného archívu WARC alebo predspracovanej Wikipedie. Takto vytvorené časti vertikálu sa kvôli úspore pamäte budú hneď zapisovať do výstupného súboru.

Kapitola 5

Použité technológie

Kapitola popisuje nástroje použité pri implementácii vertikalizátora. Jedná sa predovšetkým o použité programovacie jazyky a knižnice, zmienené sú aj technológie, ktoré nie sú použité priamo, ale ich znalosť je predpokladom požadovaným pre jeho správnu funkčnosť.

5.1 Python

Python je vysokoúrovňový, interpretovaný, univerzálny programovací jazyk vytvorený Guidom van Rossumom. Prvá verzia bola vydaná v roku 1991. Medzi jeho hlavné výhody patrí dobrá čitateľnosť resp. kvalita zdrojového kódu, vysoká rýchlosť vývoja, prenositeľnosť softvéru v ňom vytvoreného na mnohé platformy, dobrá štandardná knižnica a množstvo knižníc tretích strán, vrátane implementácie algoritmu Justext, či jednoduchá integrácia s ďalšími programovacími jazykmi [7]. Z týchto dôvodov bol Python zvolený ako hlavný implementačný jazyk nového vertikalizátora. Hlavnou nevýhodou jazyka vyplývajúcou z faktu, že je to jazyk interpretovaný, je nižší výkon v porovnaní s jazykmi kompilovanými [7]. Preto bola pre odstraňovanie nedôležitých častí pomocou algorimu Justext nakoniec zvolená implementácia tejto funkcionality v jazyku C++.

Najnovšou stabilnou verziou v čase tvorby tejto práce bola Python 3.5. Použitá bola staršia ale stále udržiavaná verzia Python 2.7, ktorá je prednastavená na serveroch KNOT aj superpočítači Salomon.

5.2 Langid

Langid je knižnica pre programovací jazyk Python určená na detekciu jazyka textu. [6] uvádza, že sa jedná o knižnicu rýchlu, pripravenú pre veľké množstvo (v čase písania práce 97) jazykov a schopnú spracovať aj HTML či XML dokumenty bez ovplyvnenia výsledku klasifikácie výskytom značiek. Použitou a v čase tvorby vertikalizátora aktuálnou verziou je 1.1.6.

5.3 NLTK

NLTK je knižnica pre jazyk Python určená na spracovanie prirodzeného jazyka. Prvá verzia vznikla v roku 2001 na University of Pensylvania [2]. Obsahuje množstvo modulov, vo vertikalizátore je však použitý iba balík *nltk.tokenize*, ktorý rozlišuje jednotlivé vety a následne pozície. Použitá bola aktuálna verzia 3.1.

5.4 HTML

HTML (HyperText Markup Language) je značkovací jazyk odvodený zo všeobecnejšieho jazyka SGML a je používaný predovšetkým na tvorbu webových stránok. Jeho prvú verziu vytvoril v roku 1992 Tim Bernes-Lee [10]. Súčasným štandardom je HTML5.

Vstupom vertikalizátora sú webové stránky, resp. ich zdrojový kód v jazyku HTML, nie však nutne v najnovšej verzii. Jeho znalosť je tak dôležitým predpokladom na správnu implementáiu vertikalizátora.

5.5 C++

C++ je univerzálny, objektovo orientovaný, kompilovaný programovací jazyk. Vytvorený bol v 80. rokoch 20. storočia v Bell Labs Bjarnom Stroustrupom. Vychádza z jazyka C a pridáva k nemu podporu objektovo orientovaného programovania [9]. Poslednou špecifikáciu je C++14. Tento jazyk je použitý na implementáciu algoritmu Justext, a to z dôvodu výpočtovej náročnosti procesu odstránenia nedôležitých častí webových stránok.

5.6 Htmlcxx

Htmlcxx[1] je CSS a HTML parser pre jazyk C++. Súčasnou verziou je 0.85. Je použitý pri implementácii algoritmu Justext v C++.

5.7 PCRECPP

PCRECPP[2] (Perl Compatible Regular Expression library) je knižnica, na prácu s regulárnymi výrazmi v podobe pôvodne používanej v jazyku Perl. Podobne ako Htmlcxx je súčasťou C++ implementácie algoritmu Justext. Použitá bola verzia 8.31 dostupná v repozitároch operačného systému.

5.8 HTTP

HTTP (HyperText Transfer protokol) je aplikačný protokol pôvodne určený na výmenu Hypertextových dokumentov (webových stránok), dnes je jeho využitie širšie a dá sa použiť na prenos veľkého množstva druhov obsahu. Je definovaný normou RFC 7540 [1] a funguje na pricípe dotaz-odpoveď. Najnovšou verziou protokolu je HTTP/2 štandardizovaná v roku 2015.

Znalosť tohto protokolu je predpokladom pre implementáciu nástroja na čítanie súborov WARC, ktorý sa v zjednodušenom poňatí dá pokadať za zoznam HTTP požiadavkov a odpovedí.

[1] http://htmlcxx.sourceforge.net/
[2] https://github.com/vmg/pcre

Kapitola 6

Implementácia

Kapitola popisuje výslednu podobu implementovaného vertikalizátora z pohľadu užívateľa. Následne sa venuje implementačným detailom niektorých jeho častí, ktorých riešenie je netriviálne, a zmenám oproti riešeniu prezentovanému v predchádzajúcej kapitole.

6.1 Spustenie a parametre

Vertikalizátor je implementovaný ako skript programovacieho jazyka Python. Neposkytuje žiadne grafické rozhranie ani iný spôsob interakcie behom procesu vertikalizácie, konfigurovať sa dá pomocou parametrov pri spustení, ktorých význam je popísaný v tabuľke 6.1.

Pokiaľ nie je uvedená žiadna hodnota parametra `--input`, číta vertikalizátor štandardný vstup. Na štandardný výstup sú počas vertikalizácie vypisované správy oznamujúce aktuálnu akciu, prípadne jej ukončenie a čiastočné výsledky, napr. detekovaný jazyk.

`--input [cesta]`	Udáva cestu k súboru, ktorý má vertikalizátor spracovať. Môže ním byť (nástrojom gzip komprimovaný) WARC súbor, jedna webová stránka v jazyku HTML alebo súbor s predspracovanou Wikipediou. Parameter je voliteľný.
`--output [cesta]`	Udáva cestu k súboru, do ktorého bude uložený výsledný vertikál. Ak tento súbor neexistuje, vertikalizátor ho vytvorí, ak súbor existuje, bude jeho obsah prepísaný. Parameter je povinný.
`--wiki`	Prepínač, ktorého prítomnosť vertikalizátoru oznamuje, že na vstupe sa nachádza súbor s predspracovanou Wikipediou.
`--no-lang-detect`	Prepínač, ktorého prítomnosť vypína detekciu jazyka pomocou nástroja *langid*.

Tabuľka 6.1: Parametre vertikalizátora pri spustení.

6.2 Vstup

6.2.1 Čítanie súboru WARC

Postupné čítanie súboru WARC je implementované ako jednoduchý automat, čítajúci archív po jednotlivých riadkoch, ktorý sa môže nachádzať v jednom zo stavov:

- *počiatočný stav*: postupne sa prechádzajú riadky súboru až do momentu, kým sa nenarazí na riadok začnajúci reťazcom `WARC-Type: response`, ktorý je súčasťou WARC hlavičky záznamu odpovede. Jeho prečítanie zmení stav na `in_warc_response`,
- `in_warc_response`: Pokračuje sa ďalej v čítaní riadkov, v prípade že sa narazí na záznam hlavičky `WARC-Target`, získa sa z tohto záznamu pôvodná absolútna URL adresa dokumentu. Ďalej sa pomocou regulárneho výrazu hľadá riadok začínajúci samotnú HTTP odpoveď, napr. `HTTP/1.1 200`. Po jeho prečítaní sa automat dostane do stavu `in_http_response`,
- `in_http_response`: V hlavičke HTTP odpovede sa hľadá záznam `Content-Type: text/html`. Ak dôjde k jeho nájdeniu, nastaví sa príznak oznamujúci, že daná adresa obsahuje webovú stránku v jazyku HTML na `True`. Archív WARC môže obsahovať odpovede obsahujúce XML dokument či čistý text, tieto sú z ďalšieho spracovania vylúčené. Po prečítaní prázdneho riadku sa stav zmení na `in_payload`,
- `in_payload`: automat jednotlivé riadky ukladá do jedného reťazca ako samotný HTML zdrojový kód webovej stránky. Po prečítaní riadku `WARC/1.0`, sa vytvorí objekt triedy `Webpage`, ten sa pomocou príkazu `yield` vráti užívateľovi a stav automatu sa zmení na *počiatočný stav*.

6.2.2 Pridanie podpory štandardného vstupu

Pôvodný návrh nepočítal s podporou štandardného vstupu, táto možnosť však musela byť prídaná kvôli problematickému spracovaniu archívu WARC komprimovaného pomocou algoritmu LZMA vo formáte xz. Knižnice dostupné pre programovací jazyk Python, nefungovali spoľahlivo alebo bol čas spracovania v porovnaní s nástrojmi operačného systému niekoľkonásobne dlhší. Kým nástroj *unxz* súbor dekomprimoval v priemere 55 sekúnd, doba vertikalizácie rovnakého súboru sa predĺžila o v priemere 2 minúty a 34 sekúnd v porovnaní s obsahovo zhodnou verziou komprimovanou nástrojom *gzip*.

6.2.3 Justext

Ako bolo spomenuté v kapitole 4, na výber sú dve implementácie algoritmu Justext – v jazyku Python a C++. V rámci implementácie muselo dôjsť k porovnaniu oboch riešení z hľadiska výkonu, predpokladaným výsledkom pochopiteľne bolo rýchlejšie spracovanie C++ verziou. Experimenty tento predpoklad potvrdili pričom rozdiel v rýchlosti bol približne dvojnásobný. Python implementácia vstupný archív WARC spracovala v priemere za **16 minút a 37 sekúnd**, C++ verzia v priemere za **7 minút a 22 sekúnd**.

So súhlasom autora C++ implementácie, p. Istvána Endredyho, bola táto následne modifikovaná podľa požiadavkov. Prvým krokom bolo vytvorenie rozhrania knižnice pre programovací jazyk Python, aby bolo možné toto riešenie používať v spojení s ostatnými komponentami. Výsledkom tohto procesu je modul importovateľný bežným spôsobom pomocou kľúčoveného slova `import`. Modul poskytuje nasledujúce rozranie:

- `list<string> justext(html: string)` – funkcia ako parameter príjime zdrojový kód webovej stránky, extrahuje text odstavcov a prípadne vykoná ich klasifikáciu. Návratovou hodnotou je zoznam obsahujúci texty jednotlivých odstavcov,

- `void keep_everything()` – funkcia zavolaním nastaví príznak, ktorý spôsobí, že pri ďalších volaniach funkcie `justext()` nedôjde ku klasifikácii odstavcov a budú vrátené všetky,
- `void set_justext_modifier(modifier: callable)` – funkcia ako parameter prijíma funkciu, ktorá môže modifikovať výsledok klasifikácie algoritmom Justext. Funkcii bude pri volaní podľa návrhu predaný extrahovaný text odstavca, tomu zodpovedajúci kód HTML, a reťazec s triedou priradenou pri klasifikácii. Funkcia musí vrátiť ako dvojicu novú triedu (reťazec *good* alebo *bad*) a prípadne modifikovaný text odstavca.

Pri načítaní modulu sa vytvorí jediná inštancia triedy `Justext`, ktorá následne vykonáva klasifikáciu každej požadovanej webovej stránky.

Ako názorná ukážka modifikátora výsledkov klasifikácie je implementovaná funkcia `justext_wikilinks_modifier`. Táto všetky odstavce obsahujúce URL adresu ľubovoľnej stránky anglickej Wikipedie označí ako *good*, čo spôsobí ich pridanie do výsledného vertikálu.

6.3 Detekcia jazyka

V predchádzajúcej kapitole je prezentovaná idea, že detekciu jazyka môže do určitej miery na požadované účely nahradiť aj samotný algoritmus Justext. Táto myšlienka bola overená experimentálne až počas implementácie, keďže na jej otestovanie bola potrebná časť funkčnosti výsledného riešenia.

Výsledky pokusov ukázali, že detekcia jazyka pomocou nástroja *langid* vylúčila iba niekoľko jednotiek až desiatok webových stránok, čo je pri desiatkách tisícoch záznamov relatívne malý rozdiel. Na základe tohto faktu došlo k pridaniu parametra `-no-lang-detect`, o ktorom sa už zmienila tabuľka 6.1. Podrobné porovnanie tejto alternatívy s prednastavenou konfiguráciou sa nachádza v kapitole 7.3.

Tiež sa pri implementácii porovnávala možnosť detekcie jazyka na úrovni celej webovej stránky s úrovňou jednotlivých odstavcov. Druhá alternatíva však predstavovala značné predĺženie procesu vertikalizácie a z tohto dôvodu sa vo výslednom riešení používa alternatíva prvá.

6.4 Ukladanie a obnova odkazov a obrázkov

Ako bolo zmienené v kapitole 4, ukladanie a obnova odkazov a obrázkov je riešená podobným spôsobom ako v pôvodnej verzii vertikalizátora. K ukladaniu do zoznamu dochádza však iba pri častiach webovej stránky, ktoré sú považované za dôležité, čo prispeva k úspore pamäte, no vyžaduje ďalší zásah do implementácie algoritmu Justext.

V neupravenej podobe totiž vráti iba zoznam reťazcov s textami jednotlivých parametrov bez HTML značiek. Vyžaduje sa však, aby značky odkazov a obrázkov, teda `<a>` a `<img>` boli zachované. Aby tento fakt neovplyvnil výsledok klasifikácie, je toto implementované tak, že pri extrakcii textu sa vytvoria dva reťazce. Prvý bude obsahovať iba text a použije sa na klasifikáciu, druhý zachová aj spomínané značky.

Ako z hľadiska náročnosti implementácie jednoduchšie riešenie sa ukázalo miesto zachovania pôvodných značiek vytvorenie nových, čo ďalej umožňuje zjednodušiť regulárne

výrazy na ich vyhľadanie. Používajú sa iba malé písmena, jeden druh úvodzoviek a jediný parameter `href` v prípade značky `<a>` alebo `src` v prípade značky `<img>`.

Ukladanie odkazov do zoznamu sa vykonáva pomocou už spomínaného regulárneho výrazu, ktorý vo vrátenom odstavci vyhľadá požadované značky a hodnoty potrebných parametrov, tieto pridá ako n-ticu na koniec zoznamu a ich výskyt v reťazci odstavca nahradí za reťazce `__LINK__` a `__IMG__`. V prípade odkazu sa nenahradia iba značky samotné, ale aj text nimi obkolesený.

Obnova, resp. zápis do výsledného vertikálu, prebieha pri postupnom spracovávaní jednotlivých pozícií. Ak sa v rámci tohto procesu vyskytne jeden zo spomínaných reťazcov, dôjde k získaniu prvého prvku zoznamu odkazov a jeho odstráneniu (zoznam teda funguje ako fronta). V prípade obrázku sa do vertikálu zapíše reťazec `__IMG__` nasledovaný parametrami `<link>` a `<length>`, URL adresa sa získa zo spomínaného záznamu a prevedie sa na absolútnu. Spracovanie odkazu k tomuto procesu pridáva rozdelenie jeho textu na pozície. Tie sa následne zapíšu do vertikálu a za poslednú z nich sa umiestnia parametre zhodné s prípadom obrázka. V oboch situáciách je nutné výskyt znaku medzery či vertikálnej čiary v adrese zakódovať do URL formátu.

Kapitola 7

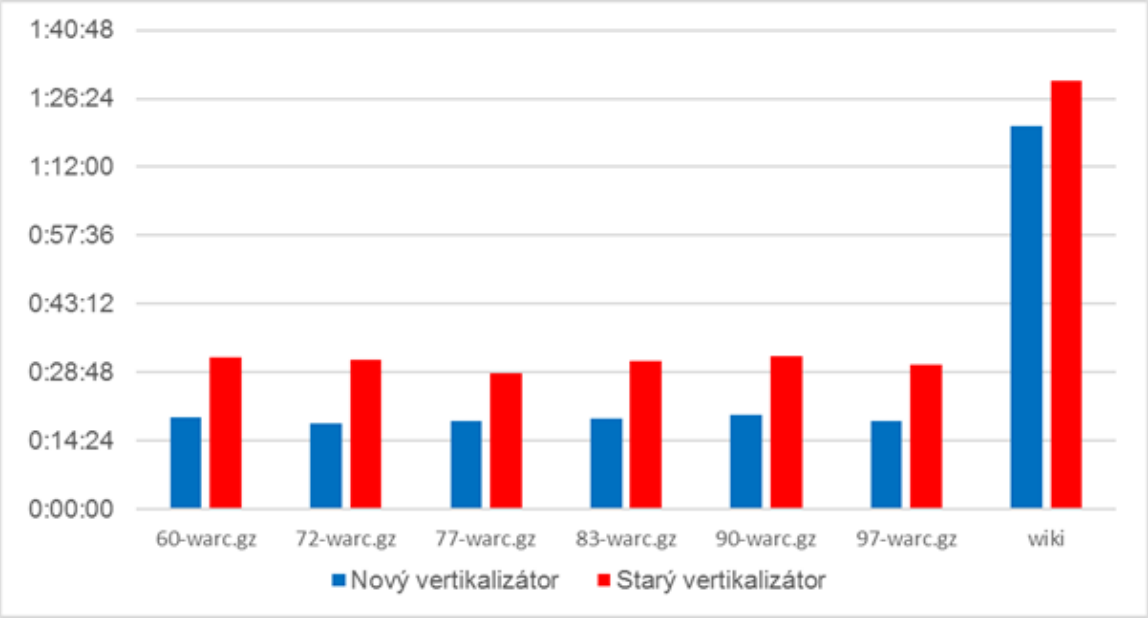
Testovanie

Táto kapitola sa venuje testovaniu vytvoreného riešenia. Zameriava sa predovšetkým na porovnanie s riešením starým z hľadiska doby spracovania, využitia operačnej pamäte a kvality výstupu. Ďalej porovnáva alternatívnu konfiguráciu s vypnutím detekcie jazyka a predvoleným nastavením, kedy k tomuto procesu dochádza. Testovanie sa uskutočnilo na niekoľkých archívoch WARC komprimovaných pomocou nástroja *gzip* a jednom súbore obsahujúcom predspracované stránky Wikipedie. Tieto testovacie vzorky boli vybrané náhodne zo súborov dostupných na serveroch výskumnej skupiny KNOT s výnimkou testu spracovania problémových súborov. Všetky testy prebiehali na serveri *athena2* (Intel Xeon CPU E5-2630 v2 @ 2.60GHz, 128 GB RAM, 4-kanálový radič, 12 x 4TB disk v HW RAID 6 s SSD cache) v časoch, kedy získané výsledky nemohli byť ovplyvnené inými bežiacimi úlohami.

7.1 Rýchlosť spracovania

Testovanie rýchlosti spracovania sa uskutočnilo na šiestich archívoch WARC a jednom súbore s predspacovanou Wikipediou. Všetky testovacie vzorky boli spracované oboma riešeniami s ich prednastavenou konfiguráciou – s výnimkou vstupu a výstupu sa nenastavovali žiadne ďalšie parametre. Čas bol meraný pomocou nástroja *time*, pričom vo výslekoch testovania je uvedený čas reálny. Pri pozorovaní priebehu nástrojom *top* však naviac došlo k zisteniu, že staré riešenie prvých niekoľko sekúnd využíva viac procesorových jadier. Ak by sa teda v úvahu brala táto skutočnosť, ďalej uvedené hodnoty by boli v prípade pôvodného vertikalizátora vyššie.

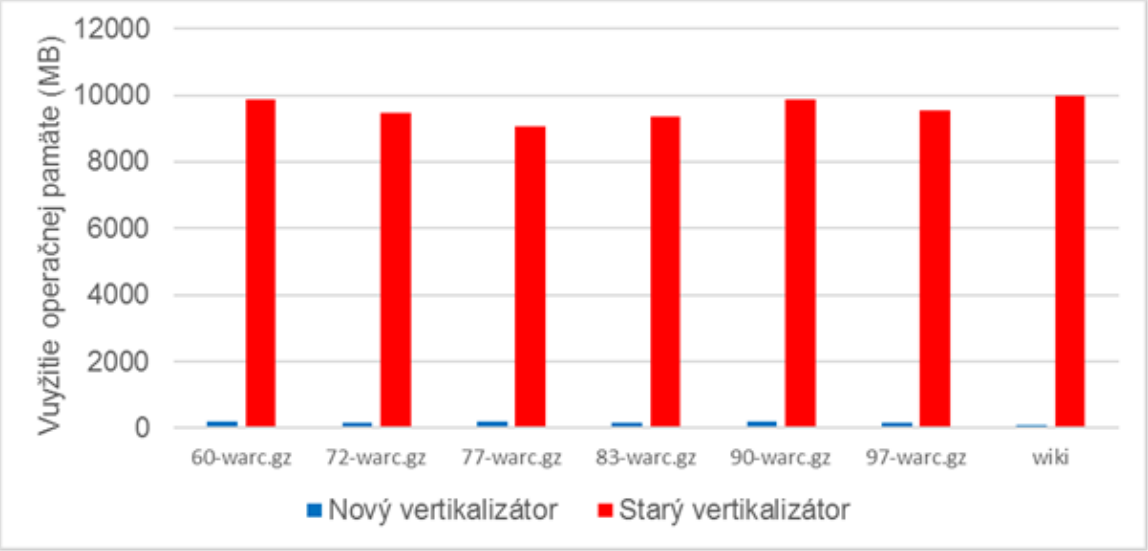
Obrázok 7.1 ukazuje výsledky testu. Na vodorovnej osi sa nachádzajú jednotlivé vzorky, na osi zvislej je čas spracovania. Modrou farbou sú označené výsledky nového riešenia, červenou farbou výsledky riešenia pôvodného. Ako si je možné všimnúť, v prípade archívov WARC je čas spracovania novým vertikalizátorm dvojtretinový až polovičný. Pri súbore s predspracovanou Wikipediou je nové riešenie rýchlejšie asi o 10 minút, čo predstavuje asi 11% z pôvodného času. Relatívny rozdiel je v tomto prípade menší predovšetkým kvôli zachovaniu celého obsahu súboru, zatiaľ čo pri odstraňovaní nedôležitého obsahu z WARC súborov je nové riešenie razantnejšie a tento obsah sa už ďalej nedostava do ďalšieho spracovania.



Obr. 7.1: Doba spracovania testovacích súborov novým a starým vertikalizátorom.

7.2 Pamäťové nároky

Testovanie využitia pamäte prebiehalo na rovnakých súboroch ako testovanie rýchlosti. Využitie pamäte bolo odčítané z nástroja *top* 10 minút ± 3 sekundy po spustení vertikalizácie, pričom ďalej prezentované údaje predstavujú použitú **fyzickú pamäť** *(resident memory size)*.



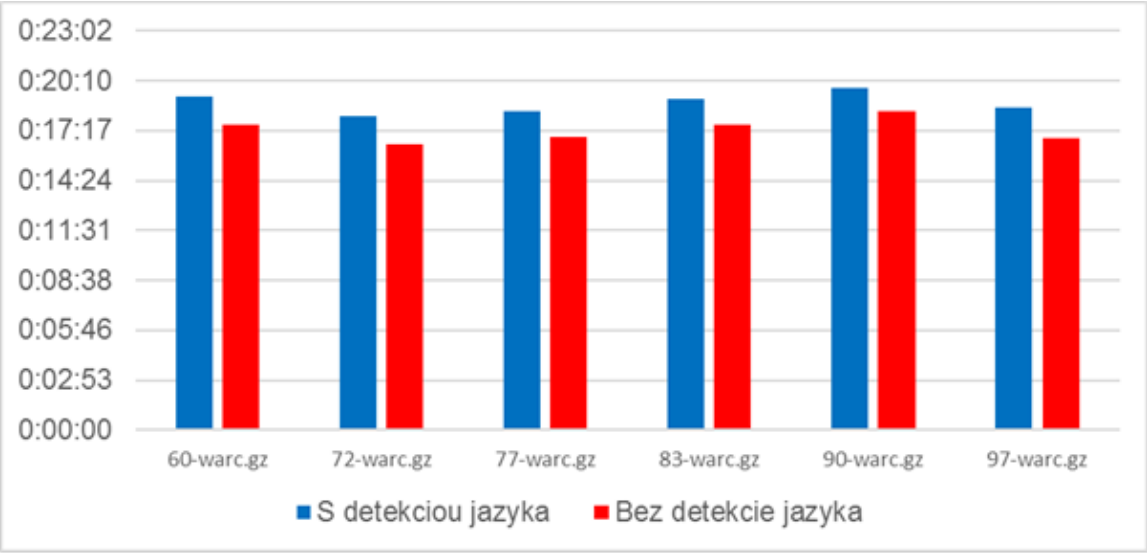
Obr. 7.2: Využitie pamäte starým a novým vertikalizátorom.

Na obrázku 7.2 si je možné všimnúť, že využitie pamäte je v novom riešení rádovo nižšie. Zatiaľ čo pôvodný vertikalizátor používa takmer 10 gigabajtov pamäte, vertikalizátor

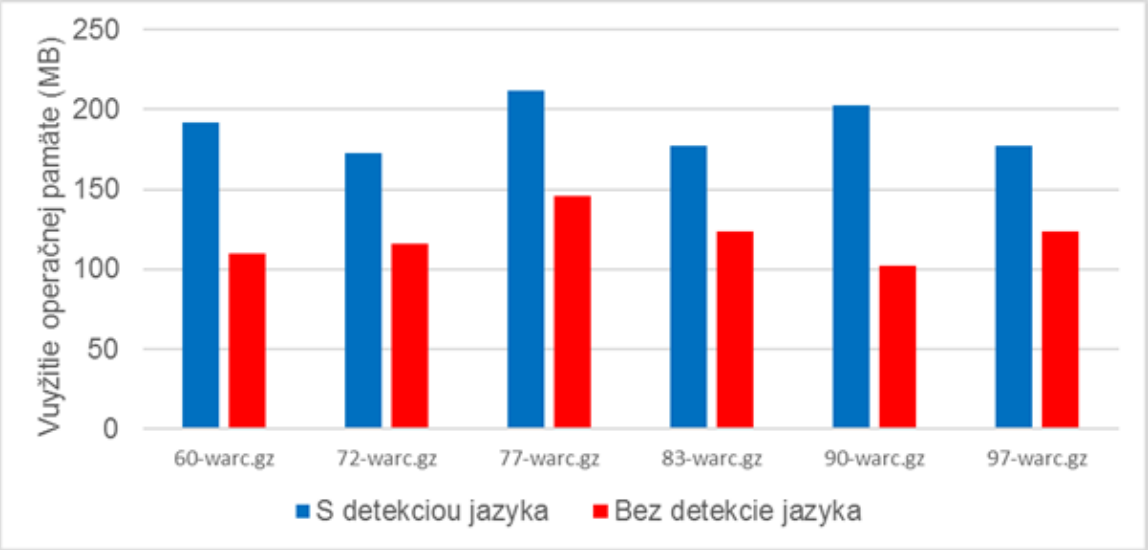
novo vytvorený využije približne 200 megabajtov. Pri spracovaní súboru s predspracovanou Wikipediou je to dokonca hodnota polovičná vďaka vynechaniu klasifikácie odstavcov a detekcie jazyka.

7.3 Vypnutie detekcie jazyka

Táto podkapitola porovnáva prednastavenú konfiguráciu vertikalizátora s alternatívnou pri ktorej nedochádza k detekcii jazyka nástrojom *langid* z hľadiska rýchlosti spracovania, využitia pamäte a rozdielov vo výstupných súboroch. Z porovnania je vylúčený proces vertikalizácie predspracovanej Wikipedie, kde k detekcii jazyka nedochádza ani v prednastavenej konfigurácii.



Obr. 7.3: Rýchlosť spracovania so zapnutou detekciou jazyka a bez detekcie jazyka.



Obr. 7.4: Využitie pamäte so zapnutou detekciou jazyka a bez detekcie jazyka.

Z obrázkov 7.3 a 7.4 vyplýva, že vypnutie detekcie jazyka v jednotlivých prípadoch zrýchlilo vertikalizáciu v každom prípade približne o 90 sekúnd. Využitie pamäte za znížilo približne o 1/3, v prípade súboru *90-warc.gz* až o polovicu. Tabuľka 7.1 udáva rozdiel v počte dokumentov pre konfiguráciu so zapnutou a vypnutou detekciou jazyka, pričom relatívny rozdiel predstavuje úbytok v počte dokumentov pre stav s detekciou zapnutou oproti detekcii vypnutej.

Vstup	**Absolútny rozdiel**	**Relatívny rozdiel**
60–warc.gz	77	0,5%
72–warc.gz	63	0,41%
77–warc.gz	73	0,47%
83–warc.gz	68	0,44%
90–warc.gz	102	0,61%
97–warc.gz	71	0,46%

Tabuľka 7.1: Rozdiel v počte dokumentov po vertikalizácii s detekciou a bez detekcie jazyka.

Z výsledkov meraní usudzujem, že poskytnutie možnosti vypnutia detekcie jazyka je validným prostriedkom na ďalšie zrýchlenie procesu vertikalizácie. Rozdiel v počte dokumentov je relatívne malý, došlo však k významnéj úspore času i pamäte. Predpokladám tiež, že samotný klasifikátor jazyka neposkytuje absolútnu presnosť, čo môže mať za následok odstránenie obsahu v požadovanom jazyku.

7.4 Kvalita výstupu

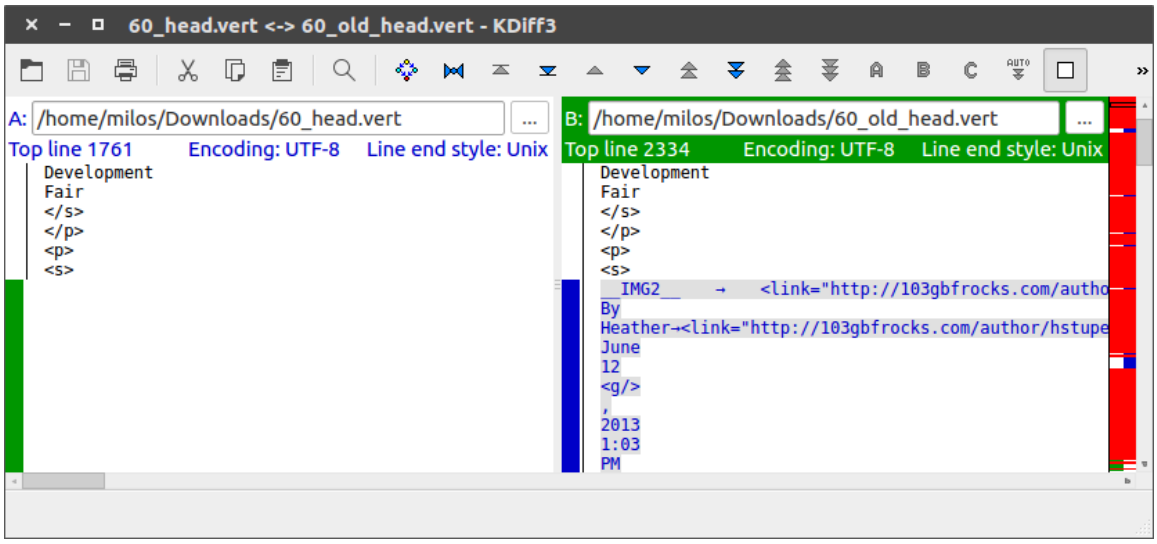
Pri testovaní tejto subjektívnej metriky bol použitý na zobrazenie rozdielov vo výstupných verikálnych súboroch nástroj *kdiff3*, viz. obrázok 7.5. Tabuľka 7.2 ďalej zobrazuje veľkosť týchto súborov pre jeden vstup spracovaný oboma riešeniami.

Vstup	**Pôvodný vertikalizátor (MB)**	**Nový vertikalizátor (MB)**
60–warc.gz	243	105
72–warc.gz	234	93
77–warc.gz	234	98
83–warc.gz	234	98
90–warc.gz	246	110
97–warc.gz	238	92

Tabuľka 7.2: Veľkosť výstupu pôvodného a nového vertikalizátora.

Z výsledkov možno usúdiť, že algoritmus Justext je z pohľadu odstraňovania nedôležitého obsahu agresívnejší ako algoritmus Boilerpipe. Výzvy na prihlásenie, povolenie Javascriptu, či rôzne chybové stavy boli novým riešením považované za nežiadúce, zatiaľ čo starý vertikalizátor ich zachoval. Došlo však aj k odstráneniu niektorých častí webových stránok, ktoré by som subjektívne považoval za dôležité. Malý vplyv na veľkosť výstupných súborov má aj fakt, že pôvodný vertikalizátor vkladá za poslednú pozíciu odkazov ďalšiu značku `<uri>`, ktorá je však obsahovo zhodná s URI celého dokumentu teda parametrom `url` značky `<doc>`.

Táto problematika vyžaduje ďalšiu analýzu, na základe ktorej by došlo k zmene prednastavených hodnôt pre rozhodovacie hranice algoritmu Justext, či vytvoreniu ďalších funkcií modifikujúcich výstup klasifikátora.



Obr. 7.5: Porovnanie kvality výstupu nástrojom kdiff3.

7.5 Problémové súbory

Poslednou fázou testovania bolo spracovanie 87 problémových súborov, ktoré časom spôsobovali pád za behu programu a následnú nutnosť úprav pôvodného riešenia. Pri spracovaní novým riešením nastal problém pri dvoch súboroch. Po analýze chyby sa našla príčina, ktorou bolo vyhľadávnie titulku pomocou regulárneho výrazu. Po obmedzení hľadania na prvých 20000 znakov zdrojového kódu už k žiadnym ďalším problémom nedochádzalo a všetky problémové súbory boli úspšene spracované.

Táto oprava by nemala mať veľký vplyv na obsah výstupných vertikálnych súborov. V praxi je podľa vlastného pozorovania i názorov členov výskumnej skupiny bežné, že titulok stránky sa nachádza na začiatku jej zdrojového kódu.

Kapitola 8

Záver

Nová implementácia vertikalizátora skátila čas spracovania archívov WARC takmer o polovicu. Dôležitejším výsledkom bolo však zníženie nárokov na pamäť z jednotiek gigabajtov na desiatky až nízke stovky megabajtov. Nedostatok operačnej pamäte predstavoval prekážku v spustení viac inštancií vertikalizátora súčasne, a to hlavne na superpočítači Salomon. Na využitie pamäte i dobu spracovania má nezanedbateľný vplyv detekcia jazyka. Vďaka spôsobu klasifikácie odstavcov pomocou algoritmu Justext má však relatívne malý vplyv na obsah výstupného vertikálneho súboru. Preto bola pridaná možnosť tento úkon vynechať. Pri testovaní bola požadovaným cieľovým jazykom angličtina, vhodná by však bola budúca analýza na jazykoch ďalších.

Prospešným je tiež pridanie možnosti modifikácie rozhodovania, ktoré časti webovej stránky sa pokladajú za dôležité, akú pôvodný vertikalizátor neposkytoval. Táto funkcionalita bola demonštrovaná vytvorením modifikátora, ktorého aplikácia spôsobí zachovanie odstavcov textu obsahujúce odkazy na stránky anglickej Wikipedie.

Otáznou ďalej ostáva kvalita výstupu. Algorimus Justext podľa porovnávaní obsahu výsledných vertikálych súborov uprednostňuje odstránenie nežiaducého obsahu aj za cenu vymazania dôležitého textu. Táto problematika bude vyžadovať ďalšiu analýzu a prípadnú úpravu rozhodovacích hraníc, či zásah do implementácie algoritmu.

V najbližšej dobe bude vertikalizátor otestovaný na celom aktuálnom archíve CommonCrawl a bude nutné vyriešiť prípadné problémy. Z výskumnej skupiny prichádzajú ďalšie návrhy na vylepšenia, napríklad podpora pre metadáta vo vertikále, rozpoznávanie viacslovných geografických názvov ako jednu pozíciu, či navrhnutie a implementácia medziformátu pre vertikalizátor. Do toho by mohli byť transformované iné typy vstupov, ktorých spracovaniu sa výskumná skupina venuje v rámci ďalších projektov.

Časti vertikalizátora budú v najbližšej dobe zverejnené ako samostatné knižnice na portáli Github.com a repozitári Python Package Index. Po dohode s p. Endrédym by som rád vytvoril univerálnejšiu verziu jeho C++ implementácie Justext a rovnako ju poskytol aj ako modul pre programovací jazyk Python.

Literatúra

[1] Belshe, M.; BitGo; Peon, R.; Google, Inc a ďalší: *RFC 7540*. [Online; navštívené 2.5.2016].
URL https://tools.ietf.org/html/rfc7540

[2] Bird, S.; Klein, E.; Loper, E.: *Natural Language Processing with Python*. O'Reilly media, Inc., 2009, ISBN 978-0-596-51649-9.

[3] Centrum zpracování přirozeného jazyka: *Popis vertikálu*. FI MUNI, [Online; navštívené 20.4.2016].
URL https://nlp.fi.muni.cz/cs/PopisVertikalu

[4] Endredy, I.; Novák, A.: *More Effective Boilerplate Boilerplate Removal - the GoldMiner Algorithm*. [Online; navštívené 22.4.2016].
URL http://www.gelbukh.com/polibits/2013_48/More%20Effective%20Boilerplate%20Removal%20-%20the%20GoldMiner%20Algorithm.pdf

[5] Kohlschütter, C.; Frankhauser, P.; Nejdl, W.: *Boilerplate Detection using Shallow Text Features*. [Online; navštívené 22.4.2016].
URL
http://www.l3s.de/~kohlschuetter/publications/wsdm187-kohlschuetter.pdf

[6] Lui, M.: *Standalone Language Identification (LangID) tool*. [Online; navštívené 1.5.2016].
URL https://github.com/saffsd/langid.py

[7] Lutz, M.: *Learning Python, Fifth edition*. O'Reilly media, Inc., 2013, ISBN 978-1-449-35573-9.

[8] Pomikálek, J.: *jusText algorithm*. FI MUNI, [Online; navštívené 23.4.2016].
URL http://corpus.tools/wiki/Justext/Algorithm

[9] Prata, S.: *Mistrovství C++:3. aktualizované výdaní*. Computer Press, a.s., 2007, ISBN 978-80-251-1749-1.

[10] Schafer, S. M.: *HTML, XHTML a CSS: bible pro tvorbu WWW stránek: 4. vydání*. Grada Publishing, a.s., 2009, ISBN 978-80-147-2850-6.

[11] Standards New Zealand: *Information and documentation — The WARC File Format*. 2006, [Online; navštívené 20.4.2016].
URL http://bibnum.bnf.fr/WARC/WARC_ISO_28500_version1_latestdraft.pdf

[12] Sustainability of Digital Formats: *WARC, Web ARChive file format*. [Online; navštívené 20.4.2016].
URL http://www.digitalpreservation.gov/formats/fdd/fdd000236.shtml

Prílohy

Zoznam príloh

Príloha A

Obsah CD

Priložené CD obsahuje nasledujúce adresáre:

- */doc* - obsahuje túto technickú správu vo formáte PDF vrátane verzie pre tlač,
- */tex* - obsahuje túto technickú správu vo formáte TeX,
- */src* - obsahuje zdrojové kódy nového vertikalizátora, obsah je podrobnejšie popísaný v priložnenom súbore *README.md*,
- */manual* - obsahuje návod na inštaláciu a spustenie nového vertikalizátora,
- */poster* - plagát z prílohy B vo formáte PNG a SVG.

Príloha B

Plagát

Plagát je k dispozícii vo formáte SVG a PNG na priloženom CD.