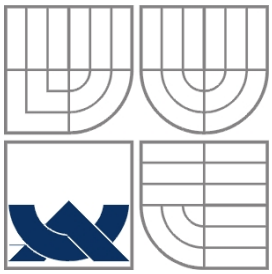
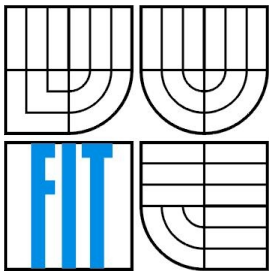


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



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

GENERÁTOR VĚDECKÝCH WEBOVÝCH PORTÁLŮ

SCIENTIFIC WEB PORTAL GENERATOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Petr Pavelka

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. RNDr. Pavel Smrž, Ph.D.

BRNO 2009

Abstrakt

Generátor vědeckých webových portálů je rozsáhlý projekt, jehož cílem je vytvoření systému, který umožní budování webových portálů za účelem poskytování přehledu vědeckých prací z konkrétních oblastí. Jednou z částí tohoto projektu je i modul pro vyhledávání domovských stránek autorů publikací. Cílem této bakalářské práce je implementace právě tohoto modulu formou skriptu nebo programu. Náplní mé práce je nalezení vhodné metody extrakce sémantické informace ze značkovaného textu, návrh a vývoj systému, který tuto metodu realizuje a sestavení množiny, na které bude systém otestován. Výstupem bakalářské práce je prototyp programu nebo skriptu, který na základě vstupního souboru ve formátu XML se jmény vědeckých pracovníků vygeneruje výstupní XML soubor s URL adresami na jejich domovské stránky a stránky se seznamem publikací. Převážná část mé práce je věnována důkladné analýze daného problému z různých hledisek, rozčlenění na podproblémy a výběru nejvhodnějších variant řešení.

Klíčová slova

Internet, URL, Yahoo, Google, Yahoo BOSS, Vyhledávací API, Python, UrlLib, xml.dom.minidom, regulární výrazy, SgmlLib, XML, HTML

Abstract

Scientific web portal generator is a large project. It's purpose is to create a system for building web portals for providing the view of science publications of authors. One of the parts of this project is module for searching of homepages. The goal of this bachelor's thesis is the implementation of of this module using the program or script. Content of this Thesis is to find the best way of method of extraction the semantic information from the marked text, design and development o the system and creating the test set for the purpose of testing. The output of this thesis should be prototype of program or script, which will generate XML file with the URL adresses of homepages of authors. System should have XML input. Almost entire thesis is about analyzing of this problem and choosing the best solution.

Keywords

Internet, URL, Yahoo, Google, Yahoo BOSS, Search API, Python, UrlLib, xml.dom.minidom, regular expressions, SgmlLib, XML, HTML

Generátor vědeckých webových portálů

Prohlášení

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

.....

Jméno Příjmení

V Brně dne 19. května 2009

Poděkování

Tímto bych chtěl poděkovat panu doc. RNDr. Pavlu Smržovi, Ph.D. za jeho odbornou pomoc, kterou mi při řešení práce poskytl.

Obsah

Úvod

Většina lidí se téměř každý den potýká se stále stejným problémem – vyhledáváním informací. S příchodem internetu a jeho rozšířením mezi běžnou populaci se možnosti vyhledávání informací mnohonásobně rozšířily. Toto masové šíření informací všech možných druhů s sebou však přineslo i další komplikace. Největším problémem je jak informaci nalézt a jak zjistit zda je relevantní. Samotné vyhledání informace v dnešní době zajišťují specializované webové stránky, které se nazývají vyhledávače. V začátcích internetu byly spíše rozšířené vyhledávače katalogového typu, které prezentovaly odkazy na jiné stránky strukturovaně podle tematických okruhů. Se zvětšováním počtu stránek tento typ vyhledávačů ustoupil do pozadí. V dnešní se spíše používají vyhledávače, které indexují obsah jiných stránek a odpovídají na vyhledávací dotazy. Nejznámější a nejvíce používaný je asi vyhledávač Google, který si svou popularitu vybudoval díky své jednoduchosti a kvalitním odpovědím na vyhledávací dotazy.

K účelům této bakalářské práce se katalogové vyhledávače nehodí, protože adresy domovských stránek se v katalogu obvykle nevyskytují. Jedinou vhodnou volbou jsou vyhledávače, které jsou schopny vrátit odpověď na vyhledávací dotaz a disponují dostatečně rozsáhlou databází dat. V následujícím textu budu mít tedy pod pojmem „vyhledávač“ na mysli pouze tento typ vyhledávačů.

Cílem této bakalářské práce je vytvoření systému, který bude schopen vyhledat na internetu velmi specifický typ informací. Indexování stránek je samozřejmě nemyslitelné a proto je nutné využít služeb některého z existujících vyhledávačů. Vyhledávače jsou primárně určeny pro lidi a proto je zpracování výsledků vyhledávání pomocí programu komplikované. Některé vyhledávače poskytují i jiná rozhraní, která jsou vhodná pro programové zpracování a právě na tento typ rozhraní se ve své práci zaměřím.

Vyhledání domovské stránky autora a stránky s publikacemi je komplexní problém a proto je vhodné tento problém rozdělit na několik dílčích podproblémů, jejichž samostatné řešení je jednodušší.

Celou práci jsem se tedy rozhodl rozdělit do následujících fází:

- volba vhodného vyhledávacího rozhraní a implementačního jazyka
- sestavení testovací množiny
- implementace systému
- testování a zdokonalování systému.

První dvě fáze práce byly realizovány v rámci semestrálního projektu. Implementace systému byla z velké části realizována až v rámci této bakalářské práce, protože v průběhu vývoje bylo nutné mnohokrát zasáhnout do vytvořeného skriptu a doplnit ho o další funkce.

1 Analýza technologií a prostředků k implementaci systému

1.1 URL

Tato zdánlivě nedůležitá zkratka je klíčová pro pochopení další problematiky a proto jí budu věnovat zvýšenou pozornost. URL (Uniform Resource Locator) je řetězec znaků s pevně definovanou strukturou, který přesně identifikuje konkrétní zdroj informací v internetu. Někdy se můžeme v praxi setkat i se zkratkou URI (Uniform Resource Indetifier). Pokud bychom chtěli dát URL a URI do vzájemného kontextu, pak nejjednodušší definicí by asi bylo tvrzení, že URL je podmnožinou URI, protože URI může identifikovat i jiné než internetové zdroje. V následujícím textu se budu zabývat výhradně URL a existenci URI opomenu. V případě hlubšího zájmu o tuto problematiku je možné nahlédnout do [1].

1.1.1 Syntaxe URL

Řetězec URL se skládá z několika částí, přičemž ne vždy musí být všechny části v URL přítomné. Syntaxe řetězce URL je následující:

```
schéma://doménové_jméno:port/cesta/soubor?dotaz#kotva
```

Jednotlivé části URL mají následující význam:

- **Schéma**, vyjadřuje způsob (protokol), jakým je daný zdroj informací zpřístupněn.
- **Doménové jméno**, označuje vzdálený server na kterém je daný zdroj informací zpřístupněn
- **Port**, je vzdálený port, na kterém naslouchá služba, která ma za úkol zpřístupňovat zdroje konkrétního typu informací na vzdáleném serveru.
- **Cesta**, označuje umístění zdroje informací na vzdáleném serveru
- **soubor**, je jméno konkrétního souboru na vzdáleném serveru, který má být zpřístupněn.
- **Dotaz**, je řetězec znaků, který obsahuje parametry pro skript, který běží na straně serveru
- **Kotva**, je řetězec znaků, který identifikuje místo v daném dokumentu

V denní praxi se obvykle setkáváme se schématy http, https, ftp apod. Při používání klientských programů (webové prohlížeče, ftp klienti) se často stává, že uživatel schéma neuvede, nicméně většina programů je schopna schéma automaticky doplnit podle svého určení (webový prohlížeč doplní http://, ftp klient ftp://). S většinou schémat v rámci URL adres je také svázaný i konkrétní port, který se používá pro danou službu. Klientské programy obvykle doplňují číslo portu automaticky podle použitého schématu a proto se ve většině případů číslo portu v URL neuvádí. Samozřejmě pokud vzdálená služba běží na nestandardním portu, pak je nutno uvést i číslo portu.

1.1.2 Absolutní a relativní URL

V rámci HTML a XHTML dokumentů na webu se často můžeme setkat s tzv. relativní URL. Pro pochopení pojmu relativní URL je nutné nejdříve definovat pojem absolutní URL. Jako absolutní URL je označována URL adresa, která obsahuje schéma a doménové jméno. Relativní URL adresa je pak taková adresa, která tyto údaje neobsahuje. Součástí cesty v relativní URL adrese mohou být i znaky vyjadřující relativní umístění ve struktuře adresářů na vzdáleném serveru. Samozřejmě samotná relativní URL adresa v rámci internetu by byla k ničemu a proto je nutné ji vždy doplnit s pomocí absolutní URL adresy dokumentu, ve kterém se relativní URL vyskytla. Proces doplňování relativní URL adresy na absolutní je většinou realizován webovým prohlížečem a proto si tohoto faktu často ani nevšimneme, ovšem pokud chceme zpracovávat html nebo xhtml kód skriptem, pak musíme s výskytem relativních URL adres počítat.

1.2 Regulární výrazy

Při sestavování této kapitoly jsem čerpal z [2], kde lze také nalézt kompletní seriál na toto téma. Regulární výrazy jsou řetězce speciálního významu. Jejich prostřednictvím můžeme provádět následující operace:

- extrahovat z textových dat údaje, které nás zajímají
- měnit textová data do požadované podoby
- vyhledávat a nahrazovat v textových editorech a dalších programech

Regulární výrazy jsou univerzálním nástrojem na zpracování textu. Jsou typické zejména pro prostředí Unixových operačních systémů, kde je využívá celá řada programů (např. grep, nebo sed).

1.2.1 Základy regulárních výrazů

Nejjednodušším regulárním výrazem je obyčejné písmeno - třeba r. Když se v textu hledá řetězec, který by tomuto regulárnímu výrazu vyhověl, hledá se jednoduše písmeno „r“. Implictně většina nástrojů rozlišuje velikost písmen, ale většinou je toto chování možné vypnout. Např. v jazyce python v modulu re má většina funkcí volitelný parametr „flags“, který slouží ke specifikaci právě tohoto chování.

Ve většině případů však nehledáme jediné písmeno ale celá slova. Proto lze regulární výrazy řetězit (tzn. skládat za sebe). Regulární výraz abcd vlastně představuje zřetězení čtyř regulárních výrazů o jednom písmenu. Při zpracování regulárních výrazů obvykle platí, že se zpracovávají po řádcích. Modul re v jazyce python však ve výchozím nastavení nebere na konce řádků v řetězci ohled a považuje celý řetězec za jediný řádek. Toto chování lze opět změnit parametrem flags.

Nyní vysvětlím použití regulárních výrazů na krátkém příkladu, který by se dal použít v rámci této práce.

Řekněme, že chci v HTML dokumentu hledat počáteční značky hypertextových odkazů tzn. <a> a dále předpokládejme, že počáteční tag může obsahovat 0 – n atributů. Pro vyhledání tagu bez atributů by stačil regulární výraz „<a>“. Pokud však budeme mít odkaz ve tvaru text, pak regulární výraz „<a>“ nenajde nic. Do výrazu je potřeba nějak zakomponovat, že mezi „<a“ a „>“ se může nacházet 0-n znaků. V regulárních výrazech existují

znaky se speciálním významem. Jedním z nich je znak „.“, který označuje libovolný znak. Pomocí znaku „*“ pak provedeme zřetězení 0-n výrazu, který se nachází bezprostředně před znakem „*“. Mohlo by se zdát, že výraz „<a.*>“ je správný, při jeho použití bychom ale zjistili, že by se vybral celý odkaz včetně koncové značky. Je to proto, že znak „*“ se snaží „pohltit“ co nejvíce znaků. Abychom tomu zabránili, tak je nutné omezit množinu znaků, které znak „*“ řetězí. Toho docílíme např. nahrazením tečky řetězcem „[^>]”. Hranaté závorky označují množinu znaků a stříška v rámci hranatých závorek označuje negaci (tzn. použijí se všechny znaky, kromě následujících). Výsledný regulární výraz by tedy měl podobu „<a[^>]*>“. Pokud bychom chtěli ještě navíc extrahovat odkaz na stránku pak to můžeme provést např. pomocí tohoto výrazu:

```
<a[^>]*href=”([^”]*)”[^>]*>
```

Novým prvkem jsou v tomto případě kulaté závorky, které označují místo v řetězci, které si má regulární výraz zapamatovat. Nástroj, který zpracovává regulární výraz je pak schopen obsah kulatých závorek vrátit.

V modulu re jazyka jsou tyto hodnoty ukládány v rámci pole „group“ které je součástí objektu, který vrací vyhledávací funkce.

Pro účely použití regulárních výrazů v této práci je vhodné zmínit ještě jednu věc a to escape sekvence. Některé znaky mají v rámci regulárních výrazů speciální význam (např. znak tečka). Pokud chceme zrušit speciální význam těchto znaků, tak je nutné použít tzv. escape sekvence. Escape sekvenci konkrétního znaku vytvoříme jednoduše tak, že danému znaku předřadíme zpětné lomítko. Takže například regulární výraz „\.“ skutečně bude hledat tečku a ne libovolný znak.

1.2.2 Možné komplikace při použití regulárních výrazů

Použití regulárních výrazů je sice relativně jednoduché, ale je několik věcí, na které je vhodné dávat pozor:

- Intervaly znaků vycházejí z kódování ASCII. To znamená, že například výrazu [a-z] vyhoví libovolné malé písmeno anglické abecedy. Doplnit velká písmena není žádný velký problém ([a-zA-Z]), ale s českými znaky, resp. jakýmkoliv znakem cizích abeced je potíž. Je nutné vždy prostudovat dokumentaci k danému nástroji, který regulární výraz zpracovává, a zjistit zda podporuje zpracování těchto znaků, příp. jak toto chování správně nastavit.

- Je potřeba vždy regulární výraz pečlivě zkontrolovat a otestovat. Často může vyvolávat problémy nadměrné používání konstrukcí typu „.*“ nebo nesprávné nastavení nástroje pro zpracování regulárního výrazu.

1.3 Platforma Yahoo BOSS

Yahoo BOSS[1] je nová webová vyhledávací platforma, která umožňuje vývojářům využívat služeb vyhledávacího stroje Yahoo. Samozřejmě existuje více podobných služeb jako např. Yahoo API nebo Google SOAP[2] API. Bohužel vyhledávač Google ukončil koncem roku 2006 podporu Google SOAP API a není již možné získat klíč vyžadovaný pro používání služby. Vyhledávač Google v současné době poskytuje pouze vyhledávací API pro AJAX, což je pro mé účely nevhodné. Yahoo API je v podstatě předchůdcem Yahoo BOSS a má několik nevýhod, z nichž nejpodstatnější je omezení

1 Build your Own Search Service
2 Simple object access protocol - http://www.w3.org/TR/SOAP/

maximálního počtu dotazů na 5000 za den. Vzhledem k těmto faktům jsem si Yahoo BOSS vybral za primární rozhraní pro získaní výsledků vyhledávacího dotazu.

1.3.1 Charakteristika

Komunikace s rozhraním Yahoo BOSS je poměrně jednoduchá. Parametry vyhledávání jsou součástí URL a výstupem je dokument ve formátu XML nebo JSON[3]. Rozhraní je velice flexibilní a umožňuje modifikovat vyhledávání různými parametry. Z hlediska vyhledávání domovských stránek autorů publikací jsou užitečné následující volby:

- omezení oblasti vyhledávání pomocí parametru „sites“ na konkrétní domény
- specifikace konkrétního jazyka a regionu
- možnost změnit počet výsledků, vrácených na jeden dotaz a posouvat se v seznamu výsledků
- omezení vyhledávání pouze na konkrétní typ dokumentu (např. HTML nebo PDF)
- možnost vyžádat si k výsledku dotazu i další informace, jako například abstrakt stránky, nebo seznam klíčových slov

Podrobnou dokumentaci k jednotlivým funkcím tohoto rozhraní lze nalézt na [3].

1.4 HTML, XHTML a XML

Při analýze těchto značkovacích jazyků jsem čerpal z [5, 6, 7, 8, 9]. Tato problematika je klíčová pro pochopení fungování analýzy a proto jsem jí věnoval poměrně velký prostor. Některé části textu byly doslovně převzaty z uvedených zdrojů

1.4.1 HTML

HTML je zkratka označující HyperText Markup Language, tedy v doslovném překladu značkovací jazyk pro hypertext. Je to jeden z jazyků, který se používá v prostředí webových prohlížečů a dá se říci, že je patří mezi nejvíce rozšířené. Účelem tohoto jazyka je publikace informací prostřednictvím internetu.

Podobně jako většina značkovacích jazyků i tento jazyk vychází z jazyka SGML (Standard Generalized Markup Language). Vývoj HTML byl ovlivněn vývojem webových prohlížečů, které zpětně ovlivňovaly definici jazyka.

Vývoj jazyka

Historie jazyka HTML sahá až do roku 1990 kdy Tim Berners-Lee vyvinul první verzi jazyka HTML. Spolu s jazykem HTML byl vyvinut i protokol HTTP (HyperText Transfer Protocol), který má za úlohu přenos HTML dokumentů po síti. Cílem první verze jazyka HTML bylo zjednodušení prezentace informace informací, protože v té době se používaly jazyky TeX nebo jazyk SGML. Zároveň s první verzí jazyka se objevil i první webový prohlížeč, který byl pojmenován World Wide Web.

3 JavaScript Object Notation - http://www.json.org/

Od těchto prvních kroků následoval velmi rychlý rozvoj webu a všech technologií, které s ním souvisejí. Jazyk HTML má velké množství verzí, z nichž se příliš neuchytily a některé se staly na dlouhou dobu široce využívaným standardem.

Verze 5

7. března 2007 byla konsorciem W3 založena nová pracovní skupina HTML, jejímž cílem je vývoj nové verze HTML. Jako název nové specifikace bylo odhlasováno HTML 5. Specifikace této verze, ještě není hotová a její vypuštění se předpokládá někdy v roce 2012. Jak bylo uvedeno výše, tak od verze 2.0 je jazyk HTML aplikací jazyka SGML. O připravované verzi HTML 5 však již toto tvrzení platit nebude.

Popis a syntaxe jazyka HTML

Množiny značek a jejich atributy se mezi verzemi mírně liší nicméně většina základních principů platí bez ohledu na verzi. Značky (označované také jako tagy) jsou uzavírány mezi znaky „<“ a „>“.

Příkladem zápisu značky může být „<a>“. Většina tagů v jazyce HTML je párových (v jazyce XHTML jsou párové všechny). To znamená, že je nutné použít jak otevírací značku tak uzavírací značku. Otevírací a uzavírací značky mají podobný význam jako závorky v matematických výrazech – tzn. ohraničují nějakou část výrazu. HTML tagy samozřejmě neohraničují části výrazů ale části textu a tím jim dávají význam, který udává použitá značka. Část dokumentu tvořená otevírací značkou, nějakým obsahem a odpovídající uzavírací značkou tvoří tzv. element (prvek) dokumentu. Součástí obsahu elementu mohou být další vnořené elementy. Součástí otevírací značky elementu mohou být atributy, které obsahují doplňující informace o vlastnostech elementu.

Některé značky v jazyce HTML jsou nepárové. To znamená, že nemají žádný obsah a nepoužívají koncovou značku. Příkladem může být značka pro vytvoření horizontální čáry „<hr>“ nebo značka pro vložení oddělovače řádku „
“

Podívejme se nyní podrobněji na tag <a>, který slouží k definování hypertextových odkazů v HTML dokumentech. Tag <a> je párový a může obsahovat různé atributy. Zápis tohoto tagu v HTML dokumentu může vypadat například takto:

```
<a href="http://www.priklad.cz">priklad</a>
```

Z hlediska této práce je pro nás podstatný pouze atribut href, který obsahuje URL na jinou webovou stránku. Atribut se vyskytuje ve všech hypertextových odkazech. Existuje však jeden speciální případ tagu <a>, ve kterém se atribut href nevyskytuje. Jedná se o tzv. kotvu, která identifikuje místo v dokumentu. Tag <a> má pak následující podobu:

```
<a name="misto">Misto v dokumentu</a>
```

Na takto definované místo v dokumentu se lze odkazovat pomocí klasického odkazu s atributem href. Při aktivaci takovéhoto odkazu pak prohlížeč nejen načte stránku, na kterou odkaz směřoval, ale zároveň se posune na místo, kde umístěna daná kotva. Pro každou verzi jazyka HTML existuje definice pravidel DTD (Document Type Definition). Od verze 4.01 musí být odkaz na deklaraci DTD v dokumentu uveden pomocí klíčového slova DOCTYPE. DTD definuje pro určitou verzi elementy a atributy, které lze používat.

Dokument může mimo značkování obsahovat i další prvky:

- Direktivy – začínají znaky <!, jsou určeny pro zpracovatele dokumentu (prohlížeč).

- Komentáře – pomocné texty pro programátora, nejsou součástí obsahu dokumentu a nezobrazují se (prohlížeč je ignoruje). Příklad komentáře je uveden níže.
- Kód skriptovacích jazyků.
- Definice událostí a kód pro jejich obsluhu.

Struktura HTML dokumentů

Dokument v jazyce HTML by měl mít následující strukturu:

1. deklaraci DTD
2. kořenový element html (značky <html> a </html>), který reprezentuje celý dokument. Kořenový element by měl být přítomen v každém HTML dokumentu. Pokud element html chybí tak je většina prohlížečů schopna si jeho umístění „domyslet“. Ovšem jeho vynechávání samozřejmě není v pořádku a může to vést k nedefinovanému chování parseru.
3. hlavičku elementu, která obsahuje metadata, vztahující se k celému dokumentu. Může zde být definován název dokumentu, jazyk, kódování, klíčová slova, popis, použitý styl zobrazení. Hlavička je uzavřena mezi značky <head> a </head>. Pokud element head chybí, tak je opět doplněn prohlížečem.
   Hlavičky HTML dokumentů jsou poměrně významné, protože obsahují informace, které vyhledávače mohou využít při indexování stránky.
4. Tělo dokumentu – obsahuje vlastní text dokumentu. Vymezuje se značkami <body> a </body>. Element body je povinný, ale jeho otevírací a koncová značka povinná není, prohlížeč je opět schopen doplnit ji sám podle kontextu.

Příklad HTML dokumentu ve verzi 4.01:

```
  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

  <html>

     <!-- komentář -->

     <head>

        <title>Titulek stránky</title>

     </head>

     <body>

        <h1>Nadpis první úrovně</h1>

        <p>Odstavec <a href="http://www.w3.org/">W3C</a></p>

     </body>

  </html>
```

Parsování HTML dokumentů

Pro parsování HTML dokumentů existuje celá řada prostředků. Těmi nejznámějšími, které asi používáme každý den jsou webové prohlížeče. Účelem webových prohlížečů je však především prezentovat HTML dokument v grafické podobě. Samotné parsování (syntaktická analýza) nedá dokumentu grafickou podobu. To zajišťují až styly, kterými je popsán grafický vzhled každého elementu v HTML dokumentu.

Pro účely této práce je však spíše podstatnější samotný zdrojový kód a syntaktická analýza než vzhled a styly, které ho ovlivňují. Účelem této kapitoly je především uvědomit si jeden fakt – HTML dokument přes to, že je strukturován pomocí tagů, je stále obyčejný textový dokument a lze ho tedy zpracovat nejen parserem ale i regulárními výrazy. Při extrakci dat z HTML dokumentu, je potřeba učinit rozhodnutí, zda je výhodnější použít parser nebo regulární výraz. Obecně lze říci, že pokud hledám výskyt nějakého textu v dokumentu tak je vhodnější použít regulární výraz a pokud nám jde spíše o extrakci informací z nějakého logicky uspořádaného celku, pak je vhodnější použít parser.

1.4.2 XHTML

XHTML je zkratka pro extensible hypertext markup language („rozšiřitelný hypertextový značkovací jazyk“). Jedná se o značkovací jazyk pro tvorbu hypertextových dokumentů podobného charakteru jako je jazyk HTML. Tento jazyk byl vyvinut konsorciem W3 aby nahradil HTML 4.01. V roce 2007 však došlo k založení pracovní skupiny, která má za cíl vytvořit novou verzi HTML, která ponese označení HTML 5 a její XML variantu XHTML 5. Vedle toho paralelně pokračuje i vývoj XHTML 2.0.

Rozdíly XHTML oproti HTML

V XHTML na rozdíl od HTML musí být všechny tagy ukončené a to včetně nepárových jako jsou
, <hr> nebo <img>. Zápis může mít více podob. Buď použijeme klasické (a validní) <img></img> nebo zkrácené <img/> nebo mírně upravené <img />. Vzhledem ke kompatibilitě a dalším problémům, které převážně souvisejí s prohlížeči je nejvhodnější použití poslední varianty.

Dalším podstatným rozdílem, oproti jazyku HTML je, že v XHTML musí být všechny tagy a jejich atributy zapsány malými písmeny. Je to z toho důvodu, že jsou takto deklarované v odkazované DTD. Všechny hodnoty atributů musí být uzavřeny do uvozovek. Dokument musí začínat XML deklarací. Její použití není povinné, pokud je dokument kódován v UTF-8 nebo pokud určujeme kódování vyšší protokolem (http například).

1.4.3 XML

Zkratka XML je odvozena od anglického názvu eXtensible Markup Language, což v doslovném překladu znamená „rozšiřitelný značkovací jazyk“.

Při objasňování pojmu XML se často používá jako příklad jazyk HTML. S tímto jazykem má však XML společný spíše vzhled zdrojového kódu než sémantiku. Jazyk HTML je množina zvolených „tagů“ - tedy značek, které umožňují formátování textového dokumentu. Sémantika (tj. význam) tagů v jazyce HTML je pevně určena a množinu tagů nelze rozšiřovat. Naproti tomu XML standard pouze popisuje syntaxi dokumentu, sémantiku data uložená v XML získají až poté, co jsou zpracována nějakým systémem (například programem). XML dokument může obsahovat libovolné množství různých tagů za předpokladu, že je dodržena správná syntaxe. Účelem jazyka XML není definovat vzhled textových dokumentů, tak jak je tomu v jazyce HTML, ale členit data podle jejich logického uspořádání. Díky tomu se data uložená v XML mnohem jednodušeji zpracovávají a je to také jeden

z hlavních důvodů proč se formát XML stal uznávaným a často používaným formátem pro ukládaní dat.

Jazyk XML však rozhodně není první svého druhu. Ve skutečnosti XML navazuje na mnohem starší jazyk SGML, což je standard pro vytváření "značkovaných" dokumentů . Použití této normy v praxi se však ukázalo jako příliš složité. Proto byl na základě SGML navržen nový standard – XML. XML si zachovalo většinu výhod jazyka SGML (především jeho univerzálnost). Hlavním rozdílem mezi XML a SGML je značně jednodušší vytváření a následné zpracování dokumentů. Podstatným faktem však je, že každý validní XML dokument je zároveň validním SGML dokumentem.

Velmi významnou vlastností XML dokumentů je fakt, že jsou „human readable“. Jednoduše řečeno lze je číst a editovat v libovolném textovém editoru.

Syntaxe XML dokumentů

Pravidla pro psaní XML dokumentů jsou poměrně jednoduchá:

- obsah tagu (elementu) musí začínat startovním tagem ve tvaru <jmeno_tagu>
- konec elementu je vyznačen koncovým tagem </jmeno_tagu>. Zde je potřeba zdůraznit, že koncový tag je povinný
- tagy se nesmí křížit. To znamená, že počáteční i koncový tag musí být na stejné úrovni (uvnitř stejného elementu)
- prázdný tag je vyznačen jako <jmeno_tagu/>
- uvnitř počátečního tagu mohou být atributy v následující podobě <jmeno_tagu atribut1="abc" atribut2="xyz">
- znaky se speciálním významem (např. <) musí být nahrazeny speciálními escape sekvencemi
- nejvyšší element v hierarchii elementů je označován jako "root element" a může být pouze jeden.

Každý XML dokument by měl také obsahovat hlavičku, ve které je uvedena verze standardu a kódování dat v dokumentu. Hlavička tedy může vypadat například takto:

```
<?xml version="1.0" encoding="utf-8" ?>
```

Pro ujasnění syntaxe XML dokumentu nyní uvedu krátký příklad, jak by mohl XML dokument vypadat:

```
<?xml version="1.0" encoding="utf-8" ?>

<authors>

    <author>

        <name>FirstName LastName</name>

        <university>FIT VUT Brno</university>
```

```
            <homepage>http://www.fit.vutbr.cz/~LastName/index.html</homepage>
            <publications>http://www.fit.vutbr.cz/~LastName/pubs.html</publications>
        </author>
    </authors>
```

Výše uvedený příklad reprezentuje jednoduchou strukturu XML dokumentu. Na prvním řádku je uvedena hlavička, která definuje, že parser má použít verzi standardu 1.0 a že data jsou kódována v UTF-8. V tomto případě by vynechání znakové sady nemělo na následné zpracování dokumentu vliv, protože výchozí znakovou sadou pro XML dokumenty je UTF-8, nicméně je jistější vždy znakovou sadu uvádět.

Kořenovým elementem (uzlem) ve výše uvedeném příkladu je element authors. Tento element obsahuje elementy author a každý element author pak obsahuje elementy name, university, homepage a publications. Tyto elementy pak obsahují samotná data.

Důležité je uvědomit si, že takováto struktura XML dokumentu může uchovávat informace o libovolném množství entit typu author.

Atributy elementů jsem v tomto příkladu záměrně vynechal, protože nemám v úmyslu je používat. Zmínil jsem se o nich, protože je možné na ně narazit v XML výstupu rozhraní Yahoo BOSS.

V rámci syntaxe jazyka XML je vhodné zmínit ještě poslední věc a to tzv. escape sekvence. Escape sekvence je pojem, kterým je označována posloupnost znaků, která ruší speciální význam některého znaku. V případě XML mají speciální význam např. znaky „<“ nebo „>“, které slouží k identifikaci tagů jazyka XML. Kdyby se takovýto znak volně vyskytl uvnitř dat, pak by to parser při analýze dokumentu pravděpodobně vyhodnotil jako chybu. Proto jsou tyto znaky nahrazovány tzv. escape sekvencemi. Příkladem escape sekvence pro znak „<“ je &lt. V rámci této práce je nutné dávat pozor zejména na znak „&”, který se může vyskytovat v rámci některých URL adres (tento znak obvykle odděluje jednotlivé parametry skriptu). Většina nástrojů, které slouží k exportu a importu z formátu XML, by měla být schopna automaticky převádět znaky se speciálním významem na escape sekvence a nazpět, nicméně nemusí to vždy. Pokud editujeme XML soubor ručně, pak je na tento fakt nutné dávat obzvláště pozor.

Využití XML

Standard XML byl vytvořen konsorciem W3, aby poskytl standardní formát pro zpracování dat a splňoval následující kritéria (mimo jiné):

- je "human readable" - je ho tedy možno číst a upravovat v libovolném textovém editoru
- je rozšiřitelný - umožňuje vytvářet vlastní formáty
- má jednoduchou syntaxi
- implementace parserů a dalších nástrojů na jeho zpracování je snadná (alespoň ve srovnání se SGML)

XML umožňuje vývojářům vytvářet poměrně snadno vlastní formáty pro uložení dat. Na standard XML je nabaleno velké množství dalších standardů, které umožňují jednoduše a účinně provádět kontrolu dokumentů, jejich vzájemné převody apod. XML formátem lze popsat takřka jakákoli data a je ho tedy možno využít jako obecný formát pro mnoho aplikací různého druhu.

1.5 Python

Python je moderní skriptovací jazyk, který disponuje celou řadou užitečných vlastností. Z hlediska této bakalářské práce jsou důležité zejména tyto vlastnosti:

- jedná se o interpretovaný jazyk s kompilací do bytekódu, což umožňuje snadnou přenositelnost mezi platformami
- obsahuje mnoho vysokoúrovňových datových typů jako například řetězce, seznamy nebo slovníky
- je možné vytvořit modul pro python v jazycích C/C++
- jazyk podporuje objektově orientované programování
- v základním instalačním balíku je velké množství různých modulů, které je možné ihned používat
- jazyk obsahuje garbage collector, který zajišťuje automatické uvolnění nepoužívané paměti

Veškeré informace o jazyku python byly čerpány z [4]

1.5.1 Moduly pythonu

xml.dom.minidom

Tento modul slouží k práci s dokumenty ve formátu XML. Umožňuje jejich vytváření, parsování a extrakci informací. Mezi nejdůležitější funkce tohoto modulu patří:

- funkce parseString, která ze vstupního řetězce vytvoří objekt reprezentující XML dokument. Tento objekt lze pomocí ostatních funkcí modulu xml.dom.minidom procházet a modifikovat.
- funkce toxml je opakem funkce parseString a slouží k převodu existujícího objektu na řetězec ve formátu XML.
- funkce GetDOMImplementation a createDocument, jejichž prostřednictvím lze vytvořit nový prázdný objekt, reprezentující XML dokument.
- funkce createElement a createTextNode, které slouží k vytvoření nového elementu a textového uzlu. Vytvořený textový uzel se následně naplní daty a propojí s některým objektem typu element, který je zase obvykle propojen s ostatními elementy.
- funkce appendChild, která slouží k výše zmiňovanému propojování uzlů a elementů

- funkce getElementsByTagName, kterou je možné volat v rámci celého dokumentu nebo z konkrétního uzlu. Tato funkce vrací seznam elementů daného jména. Objekty typu TextNode vytvořené funkcí createTextNode je možné získat voláním funkce childNodes nad konkrétním elementem, pro který chceme tyto objekty získat

Modul xml.dom.minidom samozřejmě obsahuje celou řadu dalších funkcí, jejichž dokumentaci lze v literatuře nebo online zdrojích. Výše uvedené funkce jsou minimem pro vytváření základní struktury XML dokumentu.

sgmllib

Tento modul poskytuje jednoduchý SGML parser, kterým se dají zpracovat textová data ve formátu SGML[4]. Podpora SGML formátu není úplná, nicméně pro účely parsování HTML, které je podmnožinou SGML, tento modul plně postačuje.

Modul obsahuje třídu SGMLParser, která zapouzdřuje parsování dokumentů. Tato třída implementuje rozhraní, které obsahuje funkce pro zpracování počátečních značek tagů včetně atributů, dat uvnitř tagů a koncových značek

Samotné parsování se spouští pomocí funkcí feed, která má jako jediný argument parsovaný řetězec. Parser pak volá odpovídající funkce pro každou značku v dokumentu a umožňuje tak strukturované zpracování celého dokumentu.

urlparse

Tento relativně malý modul umožňuje parsování a skládání URL adres. Proces parsování zajišťuje jediná funkce – urlparse. Výstupem této funkce je tuple, která obsahuje všechny základní elementy URL adresy. Z hlediska této práce je ještě zajímavá funkce urljoin, která na základě kompletní absolutní URL adresy a relativní URL adresy dokáže sestavit novou absolutní URL adresu. Modul urlparse obsahuje celou řadu další funkcí, které však není potřeba vzhledem k charakteru této práce zmiňovat.

urllib

Modul urllib je z hlediska této práce klíčový, protože zapouzdřuje funkce, které umožňují stahování webových stránek. Samotné stahování stránky provádí funkce urlopen, jejímž parametrem je url stahované stránky. Tato funkce je podobná funkci open pro čtení klasických souborů, protože stejně jako funkce open vrací datový proud (stream), který je možné číst. Na rozdíl od funkce open však nelze nad tímto typem streamu provádět operaci seek nebo write. Jedinou možnou operací je operace read.

Při stahování stránky samozřejmě může dojít k různým potížím. V takovém případě modul vyvolá odpovídající výjimku, kterou je vhodné zpracovat konstrukcí try – except.

V tomto bodě je ještě vhodné zmínit, že pomocí modulu socket lze měnit výchozí podmínky a fungování modulu urllib. Zajímavou funkcí z modulu socket je např. funkce setdefaulttimeout, která umožňuje nastavení doby, po kterou socket čeká v případě, že se nemůže dorozumět s druhou stranou. Změnou tohoto nastavení lze za určitých okolností (např. nestabilní připojení k internetu) zlepšit fungování skriptů, které jsou závislé na komunikaci přes internet.

4 Standard Generalized Markup Language

re

Tento modul zapouzdřuje práci s regulárními výrazy. Každý regulární výraz je nutné před jeho použitím zkompilovat pomocí funkce compile. Tato funkce kromě řetězce, který reprezentuje regulární výraz obsahuje i volitelný parametr, který může změnit vyhodnocení regulárního výrazu. V rámci této práce je tento parametr podstatný, protože ovlivňuje následující aspekty vyhodnocování:

- rozlišování velkých a malých písmen
- chování znaků se speciálním významem jako např. ^, $, \s

Zejména důležité je uvědomit si, že ve výchozím stavu je rozlišována velikost písmen, což je pro účely této práce nežádoucí.

Samotné vyhledávání pak zajišťují jednotlivé funkce modulu re, které jako jeden z parametrů očekávávají objekt vrácený funkcí compile a jako druhý parametr se obvykle předává řetězec nad kterým probíhá vyhledávání.

Z hlediska této práce jsou vhodné následující funkce tohoto modulu:

- funkce search prohledává předaný řetězec a snaží se nalézt místo, ve kterém je s daným regulárním výrazem shoda. Pokud takové místo nalezne, tak vrátí objekt, který obsahuje veškeré informace o této shodě.
- funkce finditer nepoužívá na rozdíl od většiny ostatních funkcí předkompilovaný regulární výraz, ale očekává přímo vyhledávaný řetězec a řetězec ve kterém se bude hledat. Výstupem této funkce je seznam objektů stejného typu jako vrací funkce search.

Funkce search je vhodná zejména k ověření zda se nějaký řetězec nachází v jiném řetězci. Funkce finditer se spíše hodí k vyhledání více pozic jednoho řetězce v jiném.

pickle

Tento malý modul slouží ukládání instancí objektů z paměti do souboru a jejich zpětnému načítání. Je vhodný zejména ve chvíli, kdy je potřeba rychle uložit nějaký objekt. Obvykle by se podobný problém řešil vlastnoručně napsanou funkcí, která by prováděla export do nějakého univerzálního formátu (např. XML). Pokud však

1.5.2 Slovníky, seznamy a další vysokoúrovňové datové struktury

Vysokoúrovňové datové struktury jsou klíčovým prvkem jazyka python a proto je vhodné jim věnovat patřičnou pozornost.

Seznamy

Seznamy v pythonu jsou v podstatě klasická pole, která disponují několika zajímavými vlastnostmi navíc. Prvky seznamu je možné indexovat o dost komplexněji než v klasických polích, které známe například z jazyků typu C, kde je proměnná typu pole pouze ukazatelem na začátek nějakého bloku paměti a index tento ukazatel pouze posouvá.

Seznamy v pythonu se indexují od nuly, ale umožňují indexovat prvky i pomocí záporného indexu. To umožňuje poměrně jednoduchý přístup k prvkům seznamu v opačném pořadí (tzn. od konce seznamu). Další důležitou vlastností seznamů je možnost indexovat více prvků najednou a vytvářet tak nové seznamy z již existujícího seznamu.

Jako většina ostatních datových struktur i objekt typu seznam disponuje několika užitečnými funkcemi, které je vhodné zmínit:

- funkce append(prvek) připojí prvek, předaný jako argument funkce, na konec seznamu, ze kterého byla funkce append zavolána
- funkce pop(index) vrátí prvek seznamu o daném indexu a zároveň tento prvek ze seznamu odstraní. V kombinaci s funkcí append tato funkce umožňuje jednoduchou realizaci struktur typu fronta
- funkce index(prvek) umožňuje zjistit index daného prvku v seznamu
- funkce sort umožňuje setřídění seznamu. Pokud se nejedná o seznam nějakého jednoduchého typu, pak je možné pomocí argumentu key specifikovat funkci, která dokáže zpracovat datový typ uvnitř seznamu a vrátí celé číslo, které pak poslouží k porovnání s ostatními položkami seznamu. Dále je možné pomocí volitelného parametru cmp specifikovat funkci, která se použije pro porovnání jednotlivých dvojic položek seznamu při řazení. Funkce sort ve výchozím nastavení řadí položky vzestupně, toto lze změnit volitelným parametrem reverse a nebo je možné po seřazení použít funkci seznamu reverse.

Nejvýznamnější vlastností seznamů je možnost použít je v cyklech typu for. V pythonu není klasický cyklus typu for s řídících celočíselnou proměnnou. Python implementuje pouze cyklus typu „for each“. To znamená že operace v těle cyklu se provedou nad všemi prvky kolekce (seznamu). Tento typ cyklu umožňuje snadnější zápis algoritmů, protože řízení cyklu je plně v kompetenci jazyka a je možné se soustředit pouze na to podstatné – tělo cyklu.

Slovníky

Slovníky, v jiných jazycích nazývané také jako asociativní pole, jsou pole prvků, kde indexem (který se v případě slovníků nazývá klíč) je vyšší datový typ (řetězec) a hodnotou může být kterýkoli jiný datový typ. Díky tomu je možné vytvářet slovníky slovníků (tj. slovník, jehož hodnoty jsou další slovníky) a jiné podobné složitější datové typy. Každý slovník v pythonu je objekt, který disponuje několika základními funkcemi:

- funkce keys() vrací seznam klíčů ve slovníku
- funkce values() vrací seznam hodnot ve slovníku
- funkce items() vrací seznam párů (klíč, hodnota)
- funkce has_key(klíč) dokáže zjistit zda už je ve slovníku přítomný daný klíč

2 Implementace Systému

Při vývoji skriptu jsem musel brát v úvahu velké množství faktorů, které by mohly ovlivňovat výslednou funkčnost. Za nejdůležitější faktory považuji následující:

- výběr jazyka, ve kterém bude skript implementován
- sestavení testovací množiny
- rozhraní skriptu
- způsob analýzy nalezených webových stránek

2.1 Výběr implementačního jazyka

Při výběru implementačního jazyka jsem musel přihlížet k mým aktuálním znalostem a charakteru úlohy. Úloha vyžaduje poměrně velké množství různorodých operací a rychlost není klíčovým požadavkem. Proto jsem se rozhodl vyhnout se jazykům typu C/C++, ve kterých by implementace trvala podstatně déle a byla by složitější. Zbyly mi tedy na výběr jazyky s vysokou úrovní abstrakce, které sice neprodukují nejrychlejší možný kód, ale umožňují snadnou a přehlednou implementaci algoritmů. Z dostupných jazyků jsem si vybral Python z několika důvodů:

- jedná se o velmi rozšířený jazyk
- je dostupný na většině běžně používaných platforem
- svou syntaxí přispívá k přehlednosti kódu
- ve své základní podobě obsahuje velké množství knihoven, které řeší většinu standardních úloh

2.2 Sestavení testovací množiny

Sestavení testovací množiny se paradoxně ukázalo jako jeden z nejtěžších úkolů. V rámci semestrálního projektu jsem manuálně sestavil množinu cca 500 jmen, které jsem náhodně sesbíral na oficiálních stránkách univerzit. Tento postup se však posléze ukázal jako nevyhovující, protože se obtížně vyhodnocovaly výsledky vyhledávání. U některých jmen jsem nebyl schopen nalézt domovskou stránku ani manuálně a některé domovské stránky neobsahovaly odkazy na publikace. Proto jsem se rozhodl, že sestavím další dvě množiny jmen, které jsem sesbíral na oficiálních webových stránkách dvou univerzit. U takovýchto množin jsem byl schopen jednoduše analyzovat výsledky skriptu a nalézt nedostatky. Bohužel ani tato metoda se neukázala jako ideální, protože skript bude v praxi pravděpodobně využíván na množiny jmen z mnoha různých univerzit. V konečné fázi vývoje skriptu byl však skript otestován na právě takto rozmanité množině a pravděpodobnost nálezu správného odkazu na domovskou stránku autora byla prakticky stejná jako při použití množin jmen z jediné univerzity.

2.3 Rozhraní skriptu

XML rozhraní skriptu je implementováno v rámci modulu xmlinterface.py. Tento modul obsahuje funkce pro čtení a zápis souborů a především funkce pro import a export interních datových typů skriptu do formátu XML a nazpět. Díky tomuto modulu je možné snadno načítat seznamy a slovníky z XML souborů nebo je ukládat. Skript si tak může zapamatovat své znalosti a použít je při příštím spuštění. Další výhodou tohoto přístupu je snadná analýza chování skriptu na základě shromážděných znalostí a možnost manuálně tyto znalosti upravit před dalším spuštěním.

Nejzajímavější funkcí modulu xmlinterface je možnost načítání a ukládání komplexních datových struktur. Tato funkce je implementována s pomocí společného rozhraní, které je součástí všech struktur, které se importují nebo exportují do formátu xml.

Modul xmlinterface v sobě obsahuje celkem šest funkcí. Dvě funkce pro čtení a zápis do souborů, dvě funkce pro import a export datových struktur typu seznam a dvě funkce pro import a export datových struktur typu slovník.

2.4 Komunikace s rozhraním Yahoo BOSS

Stručná charakteristika tohoto rozhraní byla uvedena v kapitole 1.3.1. Nyní se podívejme jak to v praxi funguje. K zasílání dotazů na rozhraní Yahoo BOSS používám knihovnu urllib jazyka Python. Tato knihovna obsahuje funkce urlopen, která vrací objekt typu stream, který se dá číst jako klasický soubor. Pomocí této funkce tedy odešlu dotaz na následující URL:

```
"http://boss.yahooapis.com/ysearch/web/v1/dotaz?
appid=pELZhdHV34FgmbRCnWHvOzV1kJtKUwDGa016syd8VIW365v4jkGTpjFFHIoiCw
FpOP2Y&format=xml&type=html&count=xx"
```

Modrým písmem jsou vyznačeny klíčové části URL, které nyní vysvětlím:

- dotaz je vyhledávaný řetězec
- xml je definice výstupního formátu
- html je definice typu dokumentů, které chci vyhledat
- xx je celé kladné číslo, které představuje počet výsledků, které chci vrátit na jeden dotaz

Po zpracování dotazu by rozhraní Yahoo BOSS mělo vrátit xml dokument s následující strukturou:

```
<?xml version="1.0" encoding="UTF-8" ?>

<ysearchresponse xmlns="http://www.inktomi.com/" responsecode="200">

        <resultset_web count="10" start="0" totalhits="8110" deephits="73500">
                <result>
                        <url>http://www.domena.com/~homepage/</url>
                </result>
                .
                .
                .
        </resultset_web>
</ysearchresponse>
```

Ve skutečnosti je výsledek dotazu poněkud složitější, protože obsahuje další elementy, ovšem pro účely této práce a následné zpracování parserem je postačující znalost této struktury.

Element resultset obsahuje obvykle tolik elementů result kolik je požadováno v parametru dotazu. Pro účely zkrácení příkladu jsem uvedl pouze jeden element result.

2.5 Způsob analýzy webových stránek

Po důkladné analýze výsledků testování první množiny jsem se rozhodl zaměřit na následující prvky analýzy:

- blacklistování některých doménových jmen
- vytvoření slovníku často se vyskytujících doménových a využití těchto dat ke změně pořadí nalezených výsledků
- analýza stránky a hledání klíčových slov
- analýza hypertextových odkazů na stránce, hledání odkazů na stránku s publikacemi a zjištění zda se opravdu jedná o stránku s publikacemi daného autora

Cílem těchto prvků analýzy je vyloučit nevyhovující stránky ze seznamu výsledků, upřednostnit stránky, u kterých je větší pravděpodobnost správného nálezu a nalézt správnou domovskou stránku .

2.5.1 Blacklistování doménových jmen

Blacklistování je realizováno formou seznamu doménových jmen. Tento seznam je načítán ze souboru při spuštění skriptu a je tedy možné ho manuálně upravit. Pro každý vrácený výsledek se kontroluje doménové jméno stránky a v případě, že se nachází na blacklistovaném seznamu, je takovýto výsledek automaticky vyřazen ze seznamu výsledků.

2.5.2 Vytvoření slovníku často se vyskytujících doménových jmen

Na základě zjištění, že některá doménová jména se často v URL adresách domovských stránek opakují, jsem se rozhodl zavést si slovník s následující strukturou:

- klíčem budou jednotlivá doménová jména
- hodnotou budou počty výskytů daného doménového jména při předchozím vyhledávání

Při každém spuštění a ukončení skriptu se tento slovník exportuje, což umožňuje nejen analýzu výsledků ale i manuální úpravu některých hodnot a tím ovlivnění následujícího chování skriptu.

Podle typu testovací množiny lze systém nastavit dvěma různými způsoby:

1. pokud se dá předpokládat, že se bude nacházet více správných výsledků pod jednou doménou, pak bude skript upřednostňovat výsledky, které mají společné doménové jméno s jinými výsledky
2. pokud se dá předpokládat, že minimum nebo žádné dva výsledky nebudou na stejné doméně, pak může skript snižovat prioritu výsledků, které jsou na stejné doméně.

Nastavení tohoto algoritmu lze modifikovat změnou řídících konstant, které jsou definovány na začátku modulu homepage_search.

2.5.3 Analýza stránky a hledání klíčových slov

K analýze stránky a vyhledání klíčových slov jsem se rozhodl využít dvou základních prostředků:

- regulární výrazy
- parser z modulu sgmllib

Každá stránka je po stažení kontrolována s pomocí funkce search z modulu re. Pokud stránka obsahuje kterékoli klíčové slovo, tak je zahájena další analýza, jinak je stránka vyřazena z dalšího zpracování jako nevyhovující.

Základní analýza webových stránek

Základní analýza je založena na předpokladu, že většina autorů bude mít na své stránce odkaz na stránku s publikacemi. Tento předpoklad nemusí být vždy 100% správný, nicméně ve většině případů je to nejjednodušší a nejrychlejší způsob jak ověřit, zda je testovaná stránka domovská.

Algoritmus této analýzy je velmi jednoduchý a skládá se z následujících kroků:

1. za předpokladu, že bylo na stránce nalezeno některé klíčové slovo, se HTML kód stránky předá parseru z modulu sgmllib.
2. Parser extrahuje všechny hypertextové odkazy, které na stránce nalezne.
3. U každého hypertextového odkazu se pomocí funkce search z modulu re zjistí zda URL nebo text elementu neobsahuje některé klíčové slovo. Pokud ano pak je takovýto odkaz považován za odkaz na stránku s publikacemi a vyhledávání končí.

Výhodou tohoto typu analýzy je rychlost. Nevýhodou je neschopnost najít domovské stránky, které v sobě přímo obsahují výčet publikací.

Rozšířená analýza webových stránek

Tento typ analýzy je založen předpokladu, že někdy mohou být publikace uvedeny přímo na domovské stránce a nebo může domovská stránka odkazovat na více stránek s publikacemi, z nichž jen jedna je ta správná.

Algoritmus této analýzy je následující:

1. pokud bylo na stránce nalezeno klíčové slovo, pak je stránka analyzována pomocí regulárních výrazů.
2. pomocí funkce finditer z modulu re jsou zjištěny pozice a četnosti všech klíčových slov
3. stejným způsobem se zjistí pozice a četnosti jednotlivých částí jména, které bylo použito při ve vyhledávacím dotazu (vycházím z předpokladu, že pokud autor vyjmenoval své publikace, pak najdu některou část jeho jména)
4. pokud se některá skupina jmen nachází za některým klíčovým slovem a tato skupina jmen má určitou minimální velikost (např. 5) pak je to považováno za nalezení výčtu publikací

5. pokud není stránka v kroku 4. označena za stránku s výčtem publikací, tak se provede podobná posloupnost kroků jako v případě základní analýzy - tzn. rozparsování dokumentu a nalezení všech odkazů na potenciální stránky s publikacemi. Je zde ovšem jeden podstatný rozdíl. Skript si postupně stahuje stránky, na které byly nalezeny odkazy na hlavní stránce a snaží se najít stejným postupem (krok 2. - 4.) výčet publikací.

6. Pokud i tento postup selže, pak je daný výsledek vyhledávání vyřazen z dalšího zpracování.

Výhodou je schopnost nalézt i stránky, které v sobě přímo obsahují publikace. Nevýhodou tohoto typu analýzy je mnohem nižší rychlost (často je nutné stahovat velké množství dalších stránek).

2.6 Algoritmus skriptu

V této kapitole stručně popíši jak jsou jednotlivé části skriptu logicky propojeny.

První akcí, kterou skript od svého spuštění provede je načtení inicializačních dat. Inicializační data jsou tvořena následujícími prvky:

- seznam blacklistovaných domén
- slovník obsahující doménové jména a četnost jejich výskytů
- seznam autorů

Načítání se provádí funkcemi modulu xmlinterface, nicméně cílem je získat datové struktury ve formátu slovníku nebo seznamu a k tomu může posloužit libovolné rozhraní.

Po načtení inicializačních dat se vezme seznam autorů a po jednom se začne následující zpracování:

- Pomocí funkce SendQuery se zajistí odeslání dotazu rozhraní Yahoo BOSS a zpracování odpovědi. Funkce vrací seznam interních datových struktur typu QueryResult
- Tento seznam je dále zpracován funkcí AnalyzeResults, která provede zpracování výsledků podle doménových jmen
- správná domovská stránka je vybrána s pomocí funkce AnalyzePages, tato funkce vrací již pouze jedinou položku ze seznamu (v případě alternativním domovských stránek jsou další struktury uloženy uvnitř seznamu, který je součástí hlavní struktury
- po nalezení správné domovské stránky je ještě modifikován slovník doménových jmen na základě doménového jména nalezené stránky
- poté je výsledek uložen do seznamu výsledků
- po zpracování všech autorů se provede export do XML formátu pomocí funkcí modulu xmlinterface

3 Vývoj systému a výsledky testů

Vývoj systému probíhal v periodicky se opakujících cyklech, které by se daly popsat v následujících bodech:

1. shromáždění a příprava testovacích dat
2. zpracování testovací množiny skriptem a analýza výsledků
3. upravení skriptu na základě dosažených výsledků

V průběhu testování jsem využíval dvě základní množiny. První množina sloužila k rychlému testování a obsahuje přibližně 100 jmen. Druhá množina sloužila k potvrzení výsledků a obsahuje přibližně 1000 jmen.

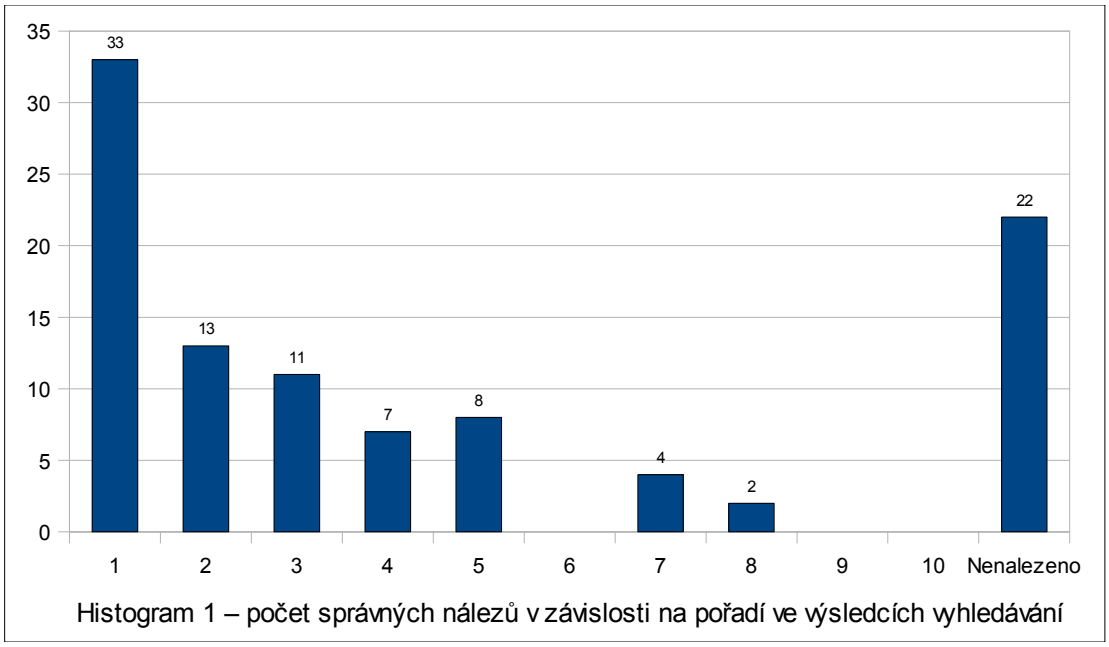
3.1 Zjištění spolehlivosti vyhledávacího rozhraní

Základním předpokladem pro nalezení domovské stránky je, aby se správná domovská stránka vůbec vyskytla mezi výsledky, které vrátí vyhledávací rozhraní. O tom zda se domovská stránka objeví v nalezených výsledcích rozhodují především dva faktory:

- přesnost vyhledávacího dotazu
- rank (míra relevance) stránky v rámci databáze daného vyhledávacího stroje

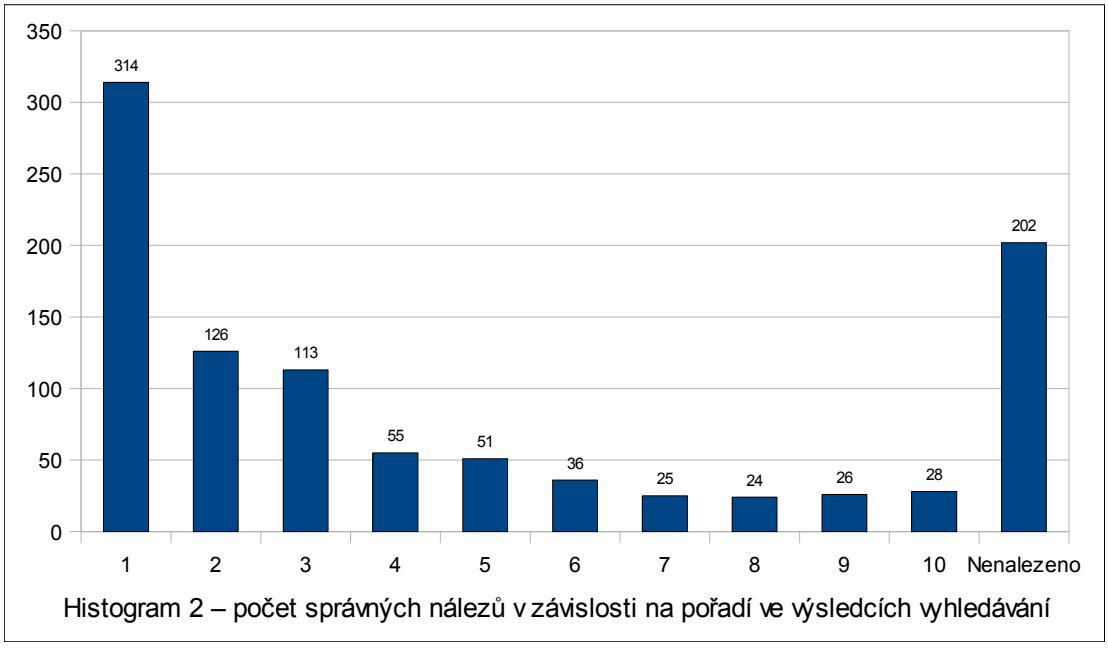
Bohužel ani jeden z těchto faktorů nelze ze strany skriptu ovlivnit. Na základě diskuze s kolegy, se kterými spolupracuji na projektu Generátoru vědeckých webových portálů, jsem dospěl k názoru, že nemá význam řešit další parametry dotazu (například domovská univerzita autora, titul apod.), protože tyto údaje pravděpodobně nebudou vůbec k dispozici. Proto jsem následující testování a vývoj skriptu postavil na předpokladu, že bude k dispozici pouze jméno autora.

Abych zjistil spolehlivost vyhledávání, upravil jsem skript tak aby zaznamenával, na které pozici se ve výsledcích nacházela správná domovská stránka. Výsledky tohoto testu je možné na Histogramu 1.



Histogram 1 – počet správných nálezů v závislosti na pořadí ve výsledcích vyhledávání

Z histogramu 1 je patrné, že při vyhledávání pouze na základě jména není vysoká pravděpodobnost, že se správný výsledek objeví na prvním místě. Ze 100 domovských stránek jich celkem 22 nebylo nalezeno vůbec a pouze 33% stránek je na prvním místě a přibližně 50% správných výsledků je na prvních 3 místech. To mne utvrdilo v názoru, že bude nutné použít další metody k extrakci správné URL adresy z výsledků vyhledávání.



Histogram 2 – počet správných nálezů v závislosti na pořadí ve výsledcích vyhledávání

Abych si ověřil správnost mých předpokladů, provedl jsem ještě další stejný test na množině o velikosti 1000 jmen. Histogram 2 zobrazuje výsledky tohoto druhého testu.

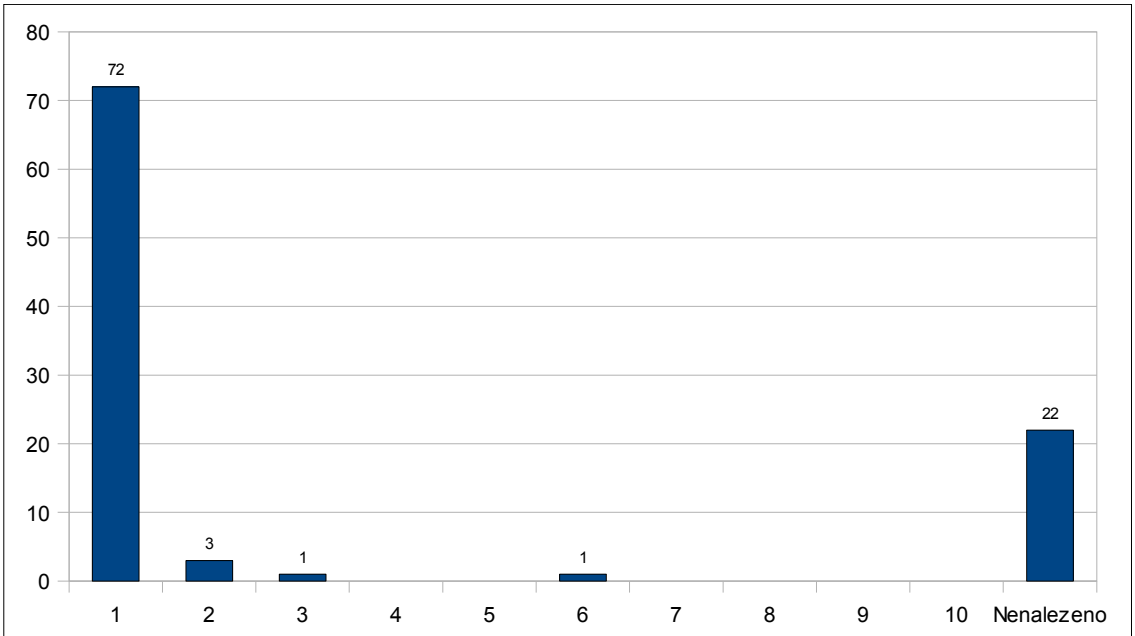
Na základě těchto výsledků jsem dospěl k závěru, že nebude nutné zvětšovat počet výsledků, které rozhraní Yahoo BOSS vrátí na jeden dotaz, protože 75% - 80% správných URL adres se nacházelo ve výsledcích.

3.2 Zpracování výsledků na základě doménového jména

Jednou z mála společných vlastností, které mají výsledky vyhledávání občas stejnou je doménové jméno. To nám dává určitý prostor pro analýzu a zpracování výsledků ještě před tím, než skript začne stahovat a analyzovat obsah jednotlivých stránek.

3.2.1 Změna pořadí výsledků na základě doménového jména

V ideálním případě by vyhledávací rozhraní vracelo správné domovské stránky na prvním místě. Jak je však patrné z předchozích testů, tak realita má do této ideální situace daleko a proto jsem se rozhodl implementovat algoritmus, který bude analyzovat doménová jména a měnit pořadí výsledků tak aby stránky s větší pravděpodobností správného nálezu byly na prvních místech seznamu výsledků. V první řadě jsem začal udělovat jednotlivým výsledkům bodové hodnocení podle původního pořadí a pak na základě analýzy doménového jména jsem toto hodnocení začal měnit. Histogram 3 zobrazuje výsledky testu, ve kterém jsem měnil pořadí výsledků na základě doménových jmen předchozích výsledků.



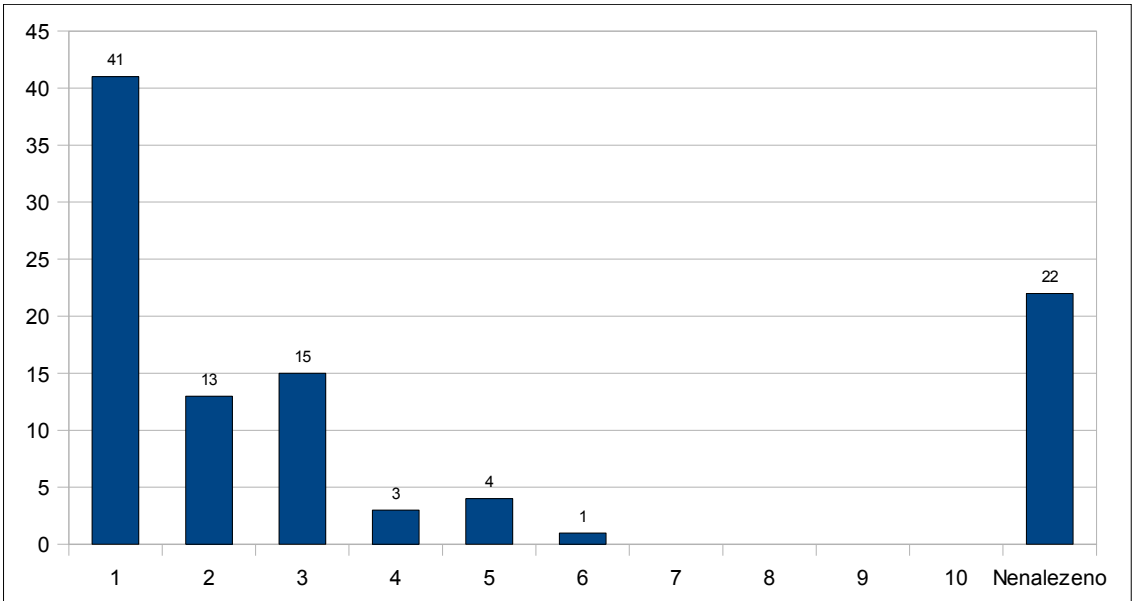
Histogram 3 – počet správných nálezů v závislosti na pořadí vyhledávání po aplikaci filtru s doménovým jménem.

Z výsledků tohoto testu je poměrně jasně patrné, že pokud se ve výsledcích vyskytuje větší počet stejných doménových jmen, tak je tento typ opatření mimořádně efektivní. Na 2 – 10 místě zůstalo pouze 5% z celkového počtu výsledků a 72% se nyní nachází na prvním místě. Nevýhodou tohoto opatření je, že je založeno na předpokladu, že se jednotlivé domovské se budou koncentrovat na jediné, nebo několika málo doménách. Pokud tomu tak není, což se klidně může stát u množin jmen z mnoha různých univerzit, pak naopak tento algoritmus zhoršuje výsledky vyhledávání, protože upřednostňuje stránky, které nejsou domovské. Typicky se v průběhu testů takto koncentrovaly stránky některých odborných elektronických zdrojů informací např. stránky z domény portal.acm.org. Jednu z možností jak tento problém řešit ukazuje následující kapitola pojednávající o blacklistování doménových jmen.

3.2.2 Blacklistování doménových jmen

Pro účely testování tohoto opatření jsem se rozhodl vyřadit předchozí opatření, které řadí položky podle doménového jména. Do blacklistu jsem pro začátek zařadil doménová jména některých sociálních sítí (např. myspace.com) a dále položky, které se během vyhledávání vyskytly.

Výsledky testu zobrazuje histogram 4.

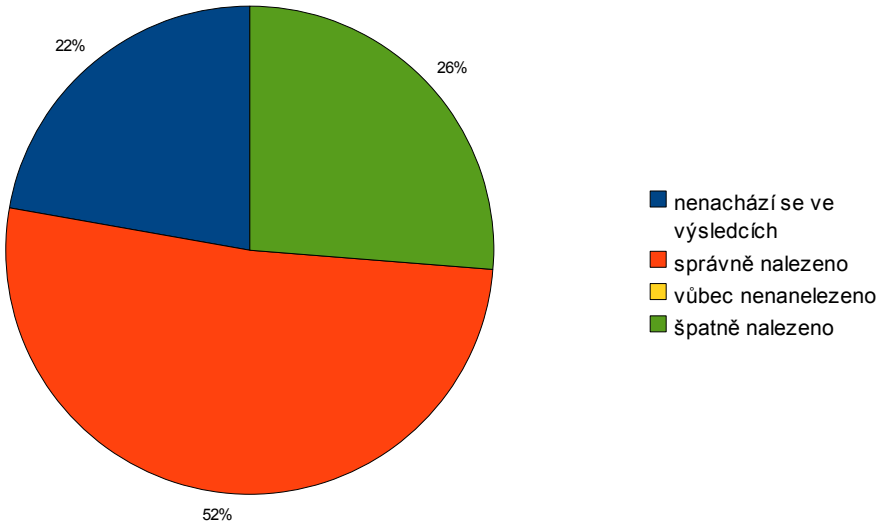


Histogram 4 – počet správných nálezů v závislosti na pořadí po aplikaci blacklistu

Jak je patrné tak oproti výsledkům z histogramu 1 je více výsledků na prvních 3 místech, nicméně ve srovnání s řazením stránek podle domén je toto opatření méně efektivní.

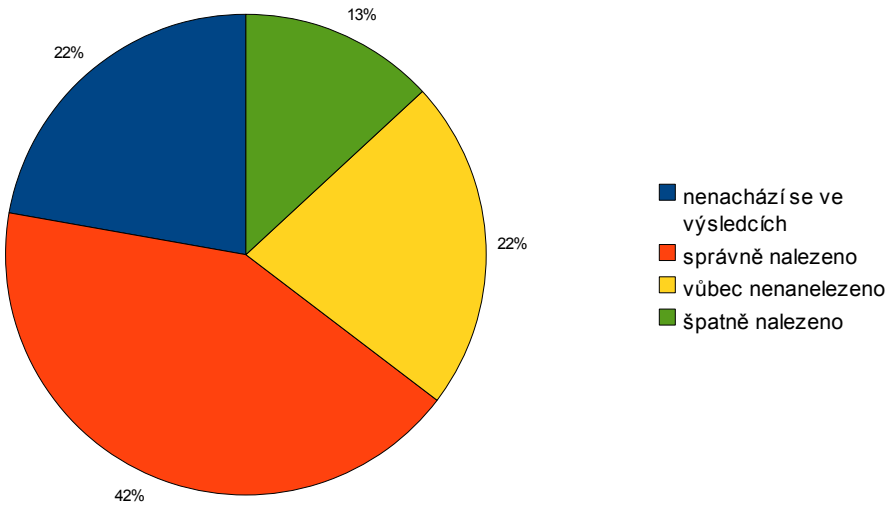
3.2.3 Testování úspěšnosti hledání při použití analýzy webových stránek

Pro účely tohoto testu jsem vyřadil analýzu doménového jména a blacklistování. K analýze stránek jsem použil jednodušší metodu analýzy popsanou v kapitole 2.5.3. Následující koláčový graf zobrazuje úspěšnost nálezu stránky:



Z grafu je patrné, že úspěšnost vyhledávání byla 52%, ostatní výsledky buď nebyly nalezeny vůbec nebo je skript přeskočil jako nevyhovující, nebo narazil na jinou stránku, která prošla analýzou dříve než stihl dospět ke správné stránce.

Při použití rozšířené analýzy byly výsledky testu následující:

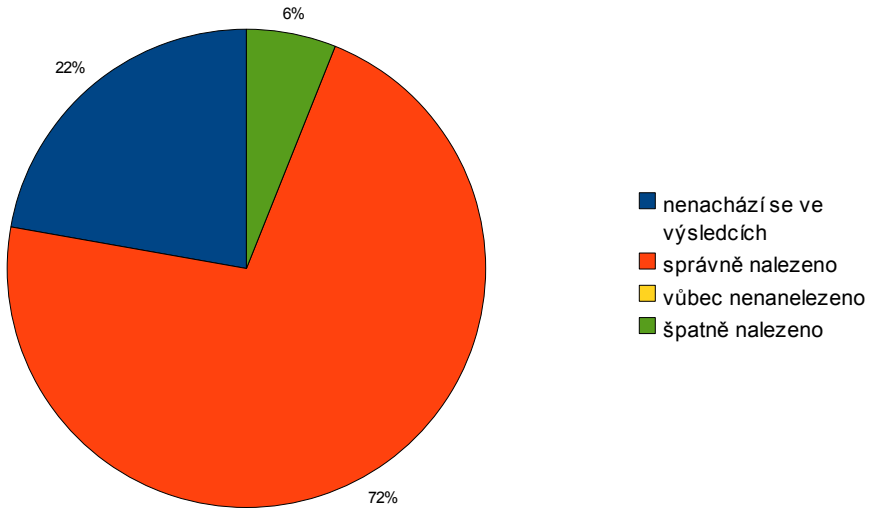


Vzhledem k tomu, že analýza byla nastavena poměrně přísně (alespoň 10 x výskyt autorova jména po sobě), se výrazně zmenšila množina špatných nálezů (tj. nálezů, které označily jako domovskou stránku jinou než správnou). Nicméně klesla úspěšnost správného vyhledávání a některé domovské stránky nebyly nalezeny vůbec. Navíc se tato analýza ukázala jako extrémně náročná na čas (přibližně 5x – 10x více), tyto fakta mne přivedly k názoru, že využití tohoto typu analýzy asi nebude velké.

3.2.4 Testování při zapojení všech prvků algoritmu

V konečné fázi testování jsem se rozhodl, že otestuji úspěšnost hledání při kombinaci všech prvků algoritmu – tj. Blacklistování, analýza doménových jmen a jednoduché analýzy.

Výsledky vyhledávání byly následující:



Po tomto testu jsem dospěl k závěru, že úspěšnost nálezu domovské stránky je již dostatečná. Ze všech domovských stránek jich bylo správně nalezeno 72%, v 6% případů bohužel i přes všechna opatření skript mylně identifikoval jinou stránku jako domovskou a ve 22% případů se stránka nevyskytla vůbec v seznamu výsledků a proto ji skript nemohl nalézt.

Závěr

V rámci této práce byl vytvořen poměrně flexibilní skript, který disponuje celou řadou funkcí. Na základě testování skriptu lze říci, že z hlediska možností, které jsou dány vyhledávacím rozhraním a použitými technologiemi, dosahuje skript vysoké úspěšnosti vyhledávání. Při hodnocení výsledků skriptu je potřeba brát v úvahu, že jisté procento autorů se ve výsledcích nikdy neobjeví. Při nasazení skriptu v praxi bude toto procento pravděpodobně o dost vyšší než v případě použitých testovacích množin. Řešením tohoto problému by mohlo být nasazení jiného vyhledávacího stroje a výsledky vzájemně porovnávat. To by však bylo poměrně komplikované, protože vyhledávač Google již neposkytuje vyhledávací rozhraní.

Během nasazení skriptu v praxi bude pravděpodobně nutné jeho doladění na základě výsledků a vstupní množiny autorů. Toto doladění bohužel nebylo možné v rámci této bakalářské práce, protože někteří z mých kolegů ještě stále pracují na svých částech projektu.

Co se týká dalšího zdokonalování skriptu, tak se domnívám, že je ještě celkem velký prostor pro zdokonalení analýzy webových stránek. Zejména detekce správné stránky s publikacemi je v rámci skriptu implementována pouze na základní úrovni. V tomto směru by pravděpodobně bylo vhodné propojit moji práci s kolegy, kteří se zabývají extrakcí výčtu publikací z webových stránek.

Poměrně velkým nedostatkem, kterým skript trpí je velká časová náročnost. Zejména při použití rozsáhlejších analýz skript strávil velké množství času stahováním stránek. Při obecných analýzách skriptu trvalo zpracování 100 výsledků kolem 3 minut. Ovšem při použití rozsáhlejší analýzy se tato doba protáhla až na 15 – 20 minut. To mi dost komplikovalo testování skriptu a je to také jeden z důvodů, proč jsem byl nucen spouštět testy na množinách, které obsahovaly maximálně 200 autorů.

Skript je díky rozhraní ve formě XML snadno propojitelný s kteroukoliv jinou aplikací. Velkou výhodou skriptu je zejména malá závislost na konkrétním vstupním formátu. Formát vstupního souboru je v podstatě definován jedním řádkem kódu. V případě složitějších vstupních formátů by nebyl problém definovat vlastní třídu, která zajistí načtení opakujících se struktur.

Tato práce byla i přínosem pro mne, protože jsem se pronikl do aktuálních technologií používaných na webu, z nichž za nejužitečnější považuji technologii XML. Dále jsem se naučil jazyk python, který jsem ještě nedávno považoval za bezvýznamný skriptovací jazyk, určený pro unixové systémové programování. Největším přínosem pro mne asi bylo pochopení regulárních výrazů, které pro mne byly velkou neznámou.

Veškeré informace nutné k propojení skriptu s kterýmkoliv systémem jsou k dispozici na přiloženém CD. Skript je také poměrně přehledně napsán a dokumentován. Jeho případné úpravy by proto neměly být problematické.

Literatura

[1] *Uniform Resource Locators.* [dostupné 25. 12. 2008]. [HTML dokument]. [http://www.ietf.org/rfc/rfc1738.txt].

[2] Satrapa P.: *Seriál regulární výrazy.* [dostupné 25. 12. 2008]. [HTML dokument]. [http://www.root.cz/serialy/regularni-vyrazy/].

[3] *Yahoo BOSS.* [dostupné 03. 12. 2008]. [HTML dokument]. [http://developer.yahoo.com/search/boss/boss_guide/].

[4] *Python v2.6.2 documentation.* [dostupné 05. 02. 2008]. [HTML dokument]. [http://docs.python.org/].

[5] *HyperText Markup Language.* [dostupné 18. 12. 2008]. [HTML dokument]. [http://cs.wikipedia.org/wiki/HyperText_Markup_Language].

[6] *Extensible HyperText Markup Language.* [dostupné 18. 12. 2008]. [HTML dokument]. [http://cs.wikipedia.org/wiki/Extensible_HyperText_Markup_Language].

[7] *Jak psat web.* [dostupné 21. 12. 2008]. [HTML dokument]. [http://www.jakpsatweb.cz/].

[8] *Extensible Markup Language.* [dostupné 21. 12. 2008]. [HTML dokument]. [http://cs.wikipedia.org/wiki/XML].

[9] Košata B.: *Seriál XML - eXtensible Markup Language.* [dostupné 14. 2. 2009]. [HTML dokument]. [http://www.root.cz/serialy/xml-extensible-markup-language/].