Masarykova univerzita
fakulta informatiky

Inteligentní vyhledávání na WWW

diplomová práce

Josef Mariánek
2001

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

Josef Mariánek

Na tomto místě bych rád poděkoval vedoucímu své diplomové práce RNDr. Pavlu Smržovi, Ph. D. za odbornou pomoc, náměty a připomínky, kterými mě dovedl k cíli. Rovněž bych chtěl poděkovat lidem pracujícím zdarma na vývoji SW vázaného licencí GNU, bez nichž by tato práce nikdy nemohla vzniknout.

Abstrakt

V diplomové práci se zabývám inteligentním vyhledáváním na webu. V teoretické části nejdříve popisuji standardy, se kterými se zde můžeme setkat. Následuje část věnovaná základnímu rozdělení vyhledávacích služeb. Poté se zabývám typy dotazů, které lze vyhledávacím službám položit, a popisuji techniky (a technologie), které tyto služby při zpracování dotazů používají. V několika dalších kapitolách popisuji konkrétní implementace vyhledávacích systémů – Google, Cora, ResearchIndex, Ahoy!, WebWatcher a další. Teoretickou část uzavírá kapitola zaměřená na integraci zdrojů WWW.

V praktické části popisuji implementaci systému sloužícího k vyhledávání vědeckých publikací konkrétních osob, která je dostupná na webu. Projekt je implementován pod OS Linux a používá technologii XML, relační databázi a některé další, volně šiřitelné nástroje.

Klíčová slova

WWW, vyhledávání, knihovny, Dublin Core, BibTeX, katalog, adresář, W3QS, vyhledávací robot, databáze, SQL, strojové učení, datamining, XML, parser, XERCES-C, XQL, Google, PageRank, Ahoy!, DRS, Stanford InfoBus, ResearchIndex, Cora, Just Research, C++, STL

Zadání diplomové práce

Prostudujte základní literaturu o vyhledávání na WWW stránkách a o existujících řešeních pro inteligentní vyhledávání informací. Na základě získaných znalostí zpracujte a vhodným způsobem prezentujte systém pro inteligentní vyhledávání vědců, zabývajících se určitým tématem, jejich technických zpráv a článků.

- Seznamte se se základními technikami vyhledávání dokumentů se zaměřením na vyhledávání relevantních WWW stránek.
- Seznamte se s existujícími implementacemi pro inteligentní vyhledávání, jejich výhodami a nedostatky.
- Implementujte systém pro vyhledávání osob pracujících v určité oblasti výzkumu, shromažďování jejich publikací dostupných na WWW a vyhledávání v takto vzniklé databázi.

Obsah

TEORETICKÁ ČÁST – inteligentní vyhledávání na WWW

Úvod 6

Formáty dokumentů na WWW
1. Standardy klasických a elektronických knihoven 7
2. Aplikace knihovnických standardů na WWW 8

Základní dělení vyhledávacích služeb na WWW
1. Kritéria pro posouzení kvality vyhledávací služby 10
2. Katalogy a adresáře 10
3. Indexy WWW 11
4. Metavyhledávače 11

Dotazy na vyhledávače
1. Klíčová slova, masky, fráze, pole a limity 12
2. Booleovské dotazy 12
3. W3QS a W3QL 13

Používané techniky a technologie
1. Roboti 14
2. Databázové technologie 14
3. Strojové učení 14
4. Datamining 15
5. XML 16

Univerzální vyhledávací systémy
1. Google 18
2. WebWatcher 24

Specializované vyhledávací systémy
1. Cora 25
2. ResearchIndex 29
3. Ahoy! 31
4. Collection of Computer Science Bibliographies 34
5. DL New Zealand 34

Elektronické informační zdroje na MU 35

Integrace IS na WWW
1. Architektura Stanford InfoBus 38

PRAKTICKÁ ČÁST – systém pro vyhledávání vědeckých publikací

1. Použité nástroje 42
2. Instalace a spuštění systému 43
3. Základní schema systému 44
4. Implementace pomocných nástrojů 45
5. Implementace subsystému pro vyhledávání osob 46
6. Implementace subsystému pro vyhledávání dokumentů 48
7. Konverze informací o dokumentech do souborů formátu XML 50
8. Indexace XML souborů do relační databáze 52
9. Transformace XML 56
10. Zpracování jednoduchých dotazů 57
11. Zpracování dotazů na URL 58
12. Výsledky a zhodnocení, návrhy na další vývoj 59

Obsah přiloženého CD 62

Literatura 63

TEORETICKÁ ČÁST

Úvod

Vznik WWW znamenal zlom v dějinách Internetu. Místo textových klientů se prosadil intuitivně ovládaný prohlížeč, který podstatu fungování Internetu schovává za grafické „klikací“ hypertextové stránky. Objevují se nové protokoly, nové technologie a hlavně extrémně rychle roste počet uživatelů a množství informací, které jsou na síti dostupné.

S tím je však spojen velký problém současného Internetu, totiž problém organizace informací a vyhledávání. Lze ho výstižně charakterizovat větou: „Co nelze najít, neexistuje.“

Při hledání řešení vzniklé situace lze čerpat v oblasti Internetu zdánlivě vzdálené. Jedná se o problematiku klasických a v dnešní době zejména elektronických knihoven. V mnoha směrech se však situace na Internetu od problémů knihoven vzdaluje. Internet roste příliš rychle a jeho „katalogizaci“ je potřeba automatizovat. Vznikají autonomně pracující vyhledávací systémy (search engines), které pro svoji práci používají metody matematické statistiky, strojového učení a umělé inteligence. Objevuje se také potřeba integrace těchto systémů. V následujících kapitolách se zabývám právě těmito otázkami současného Internetu.

Popisuji univerzální vyhledávací systém Google, který se dnes řadí k nejvýkonějším vyhledávacím systémům na světě. Podrobně rozebírám tzv. PageRank – techniku, díky které dosahuje Google tak kvalitních výsledků.

Hlavní oblastí, na kterou je však tato diplomová práce zaměřena, je popis specializovaných vyhledávacích systémů a technik, které používají. Specializované vyhledávací systémy nepokrývají tak širokou oblast webu jako např. Google, ale v oblasti, na kterou jsou zaměřeny, dosahují mnohem přesnějších výsledků. Popisuji například systém Cora, který slouží k vyhledávání vědeckých dokumentů. Aplikací strojového učení dosahuje při extrakci a klasifikaci informací výsledků blízkých lidskému zpracování.

V závěru teoretické části se zabývám otázkami propojení informačních systémů. Popisuji architekturu Stanford Infobus – jednu z možných cest, jak integraci technicky zrealizovat.

Formáty dokumentů na WWW

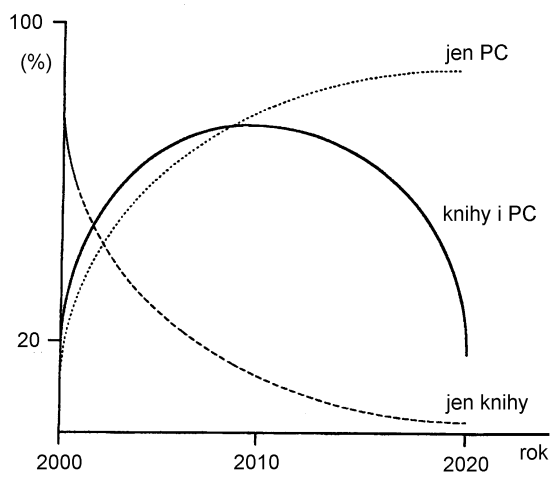
1. Standardy klasických a elektronických knihoven

Klasické knihovny, nabízející tištěnou literaturu a dokumenty, již za sebou mají několik století své existence (vynález knihtisku – Guttenberg, 15. stol.). Za tu dobu se jejich systém organizování tištěných dokumentů vypracoval téměř k dokonalosti.

V současné době například existuje mezinárodně platný systém anglicko amerických katalogizačních pravidel (AACR2R), který přesně popisuje, jak se má vytvořit bibliografický záznam daného dokumentu (nejen tištěného). Jedná se o několikasetstránkový spis, který řeší nejrůznější problematické případy, které při katalogizaci mohou nastat. AACR2R byla využita i při vytváření strojově čitelného mezinárodně platného formátu bibliografického záznamu UNIMARC. Počet položek AACR2R i UNIMARC se počítá na stovky. Do strukturovaného záznamu o dokumentu lze zanést snad jakoukoliv jeho vlastnost, včetně např. jeho hmotnosti a typu papíru, na kterém je vytištěn.

Knihovny disponují i dalšími mezinárodně platnými standardy pro evidenci dokumentů. Jedná se například o mezinárodní desetinné třídění (UDC), který se používá při vytváření systematických katalogů, nebo např. standard LCSH (Library of Congress Subject Headings) používaný při vytváření katalogů předmětových.

V současné době je v knihovnickém světě snaha aplikovat zkušenosti z klasických knihoven na knihovny elektronické. Elektronické knihovny katalogizují své informační zdroje v elektronické podobě. Kromě knih se katalogizuje mnoho dalších mediálních zdrojů. V podstatě je snaha katalogizovat vše, co má nějakou informační hodnotu, tedy i dokumenty na WWW.



Graf předpokládaného využití informačních zdrojů na různých médiích v závisloti na čase (převzato z [5])

Existuje mnoho automatizovaných knihovních systémů (AKS), např. Aleph, Voyager, ..., bohužel málo z nich dodržuje mezinárodně platné standardy. Výsledkem je v mezinárodním měřítku velmi obtížný proces jejich integrace.

Je zřejmé, že v případě prostoru WWW lze o dodržování standardů podobných těm, které známe z elektronických knihoven, jen těžko mluvit. Na WWW ukládají svá data denně miliony uživatelů, po kterých nelze žádat, aby každý svůj dokument katalogizovali ve formátu UNIMARC. Přesto jsou zde jisté snahy, jak alespoň částečně vnést do dokumentů na WWW pořádek už při jejich vytváření.

2. Aplikace knihovnických standardů na WWW

Pro různé typy dokumentů vyskytujících se na WWW existuje nepřeberné množství formátů, ve kterých mohou být fyzicky uloženy. Například pro grafická data existují formáty GIF, JPG, PS a mnoho dalších, z nichž každý má svoje specifické vlastnosti určující jeho nejvhodnější použití. Na tato data vedou odkazy z nativních dokumentů webu, tzv. hypertextových stránek. Hypertextové stránky bývají často ve formátu HTML (`www.w3.org`), ale obecně může jít i o jiný derivát standardu SGML (ISO 8879-1986, `www.sil.org/sgml/sgml.html`) pro popis elektronického dokumentu případně jiné proprietární formáty. V následujícím textu se budu zabývat standardy, které lze použít pro vytvoření bibliografické informace dokumentů na webu.

3.1. Dublin Core

Z knihovnického standardu UNIMARC byla pro potřeby univerzální jednoduché katalogizace dokumentů na WWW extrahována minimální podmnožina metaznaků, která o dokumentu poskytuje alespoň základní informace. Při volbě položek byl kladen důraz na to, aby bylo jejich vyplnění snadné, a tedy byl veřejností na webu přijat. Příkladem takové extrakce je systém Dublin Core, jehož seznam položek je uveden níže.

1. Název
2. Autor (nebo tvůrce)
3. Předmět (a klíčová slova)
4. Popis
5. Vydavatel
6. Původci (další)
7. Datum
8. Typ (kategorie zdroje, jako je domovská stránka, román, báseň, ...)
9. Formát (datová reprezentace zdroje)
10. Identifikátor
11. Zdroj
12. Jazyk
13. Vztah
14. Periodicita
15. Práva (management)

Integrace do HTML je provedena pomocí `META` tagů, kde parametr `name` specifikuje jeden z prvků Dublin Core a parametr `content` jeho obsah. Následuje příklad úplného popisu metadat používající specifikaci Dublin Core.

```
<TITLE>Dublin Core Metadata Element Set: Resource Page</TITLE>
<META name="DC.subject" content="dublin core metadata element set">
<META name="DC.subject" content="networked object description">
<META name="DC.publisher" content="OCLC Online Computer Library Center,
Inc.">
<META name="DC.author" type="name" scheme="AACR2" content="Weibel, Stuart
L.">
<META name="DC.author" type="email" content="weibel@oclc.org">
<META name="DC.author" type="name" scheme="AACR2" content="Miller, Eric
J.">
<META name="DC.author" type="email" content="emiller@oclc.org">
<META name="DC.title" content="Dublin Core Element Set Reference Page">
<META name="DC.date" type="creation" scheme="ISO" content="1996-05-28">
<META name="DC.form" scheme="IMT" content="text/html">
<META name="DC.language" scheme="ISO 639" content="en">
<META name="DC.identifier" scheme="URL"
content="http://purl.oclc.org/metadata/dublin_core">
```

Dublin Core v META značkách HTML dokumentu, převzato z [5]

Systém Dublin Core (nebo jemu podobný) by samozřejmě vnesl do dokumentů v prostoru WWW mnoho velmi cenných metainformací, nicméně v dnešní době zatím nelze na jeho podporu příliš spoléhat (1 % stránek obsahuje metainformace). Proto byly vyvinuty další technologie pro mapování WWW, které v dokumentech na webu nepředpokládají existenci jakýchkoli metainformací.

3.2. BibTex

Pro publikování dokumentů vytvářených v akademickém prostředí se často používá sázecí systém TeX [18]. Jádro systému tvoří překladač makrojazyka TeXu. Práce s TeXem totiž spočívá v přípravě zdrojového textu dokumentu, který je následně přeložen. Výsledkem překladu je vysázený dokument, který lze prohlížet a tisknout.

TeX je otevřený systém. Jeho makrojazyk umožňuje kromě využívání již připravených maker vytvářet makra nová. Takto vznikají různé nadstavby TeXu jako např. velmi rozšířený LaTeX. Program BibTex, kterému se chci věnovat v tomto oddílu, dále rozšiřuje možnosti LaTeXu při práci s citacemi.

Pokud chceme ve zdrovém textu LaTeXu citovat, použijeme k tomu makro `\cite`, například `\cite{Gettys90}`. Parametr makra je odkaz na bibliografický záznam. Ten může být pomocí tzv. prostředí `thebibliography` a jeho položek `\bibitem` zapsán přímo ve zdrojovém dokumentu, nebo může být využita databáze bibliografických záznamů ve formátu BibTeX [17].[1] Použití BibTeXu sice znamená nutnost udržovat bibliografickou databázi, ta ale může být následně opakovaně využita.

3.2.1. Formát BibTeX

Zdrojový kód bibliografické databáze BibTeXu je textový. Pomocí `@STRING` je možné definovat makra pro často používané řetězce, pomocí `@COMMENT` lze vkládat poznámky a specifikací `@PREAMBLE` lze určit program, který ovlivní výstup BibTeXu.[2] Mnohem důležitější jsou však standardní položky pro definici bibliografických záznamů jako např. `@article`, `@book`, `@conference`, ... Ty mohou podle typu dále obsahovat pole `address`, `author`, `editor`, `journal`, `title`, ap. Nejedná se sice o tak vyčerpávající seznam jako např. u standardu UNIMARC, na druhou stranu BibTeX je využíván i „neknihovníky“ a je bohatší než Dublin Core.

3.2.2. Ukázka záznamu BibTeX

```
@article{Gettys90,
    author = {Jim Gettys and Phil Karlton and Scott McGregor},
    title = {The {X} Window System, Version 11},
    journal = {Software Practice and Experience},
    volume = {20},
    number = {S2},
    year = {1990},
    abstract = {A technical overview of the X11 functionality.
This is an update of the X10 TOG paper by Scheifler \& Gettys.}
}
```

1 V obou případech je po překladu (překladech) zdrojového textu vytvořen na konci dokumentu oddíl citací a v textu uvedené citace jsou nahrazeny odkazy do tohoto oddílu.

2 Zpravidla není specifikováno. BibTeX produkuje implicitně výstup v podobě maker LaTeXu.

Základní dělení vyhledávacích služeb na WWW

1. Kritéria pro posouzení kvality vyhledávací služby

Vyhledávací systémy na WWW používají k získávání nových informací různé techniky. Jejich kvalita se často proměřuje z hlediska dvou základních kritérií, a to sice kritéria přesnosti (precision) a kritéria úplnosti (recall) odpovědi. Kritérium přesnosti udává poměr relevantních informací vzhledem ke všem informacím vrácených vyhledávačem na daný dotaz. Kritérium úplnosti udává poměr relevantních informací vrácených vyhledávačem na daný dotaz vzhledem ke všem na WWW potencionálně existujícím informacím odpovídajícím dotazu. Zlepšení vyhledávací techniky ve výsledcích proměřovaných jedním z těchto kritérií má bohužel zpravidla za následek zhoršení výsledků vzhledem ke kritériu druhému.

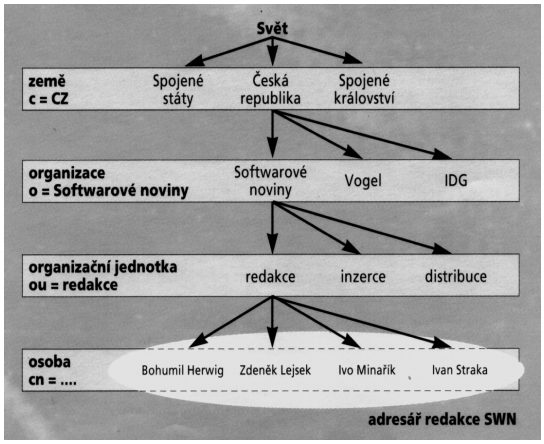
2. Katalogy a adresáře

Adresáře prostoru WWW jsou patrně historicky starším nástrojem, sloužícím k jeho prohledávání, než indexy WWW (první významný robot World Wide Web Worm se objevil v roce 1994). Jedná se zpravidla o ručně vytvářené tematické hierarchie se stromovou strukturou, jejichž listy ukazují na konkrétní dokumenty dostupné z WWW. Nejvyšší úroveň adresáře pokrývá mnoho oblastí lidské činnosti (kultura, sport, ...). Ze známých světových adresářů jmenujme např. Yahoo! (`www.yahoo.com`), z tuzemských adresář Seznam (`www.seznam.cz`).

V ranných dobách webu, kdy se počet WWW stránek počítal v tisících, byly ručně vytvářené adresáře (někdy též katalogy, pojmem adresář bývá v dnešní době označována hlavně služba specializovaná na vyhledávání lidí – viz dále) dostatečným a přesným nástrojem pro nalezení požadované informace. V současné době (miliony WWW stránek) je však jejich samostatná existence prakticky nemožná a často bývají v rámci tzv. portálů provozovány společně s indexovacím vyhledávacím nástrojem.

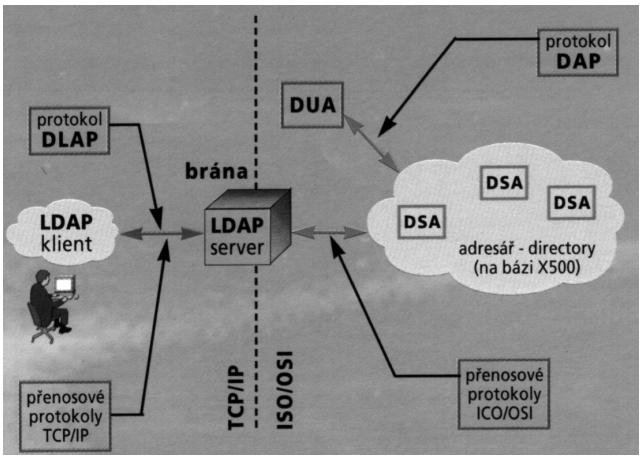
Než se začnu zabývat indexovacími vyhledávacími nástroji WWW, chtěl bych se ještě zmínit o standardech vztahujících se k adresářovým službám pro vyhledávání lidí na Internetu.

V souvislosti s evidencí lidí na Internetu vznikly tzv. adresářové služby [4]. Tento nástroj nahrazuje původní globálně spravované databáze uživatelů Internetu (např. Whois), které začaly být s nárůstem Internetu nedostatečné. Vznik adresářových služeb je úzce spojen se standardem X.500. Podle X.500 mají adresáře hierarchickou stromovou strukturu (viz obr.). Jednotlivé organizace spravují svoji dílčí část adresáře, navenek však vše splývá do jediného logického celku.



Adresářová struktura X.500 (DIT – Dir. Inf. Tree), převzato ze [4]

Dílčí adresáře jsou spravovány servery DSA (Directory System Agent), které mezi sebou komunikují protokolem DSP (Directory Service Protokol). Dotaz může být vznesen ze speciálního klienta DUA (Directory User Agent), který s DSA komunikuje protokolem DAP (Directory



Princip zpřístupnění adresářových služeb X.500 pro platformu TCP/IP, převzato z [4]

Access Protocol). Pokud DSA nezná odpověď na otázku DUA, zeptá se kompetentního DSA a vrátí zprostředkovanou odpověď.

V současné době byly některé součásti standartu X.500 nahrazeny jednoduššími variantami pro prostředí TCP/IP. Např. protokol DAP byl nahrazen protokolem LDAP – LightDAP. Konkrétním případem podpory tohoto protokolu je např. standardní adresář systému MS Windows používaný např. poštovním programem MS Outlook.

3. Indexy WWW

Alternativou k mapování prostoru pomocí adresářů (katalogů) webu je použití autonomně pracujících vyhledávacích systémů nazývaných též indexy WWW. Indexy WWW vznikly z potřeby automatizovat mapování informací extrémně rychle rostoucího prostoru webu. Ve srovnání s katalogy obsahují zpravidla mnohem více informací, na druhou stranu bývají tyto informace vzhledem k principu práce indexů méně přesné[1].

Vyhledávací systém založený na indexaci sestává z několika hlavních komponent. Jednak z robota, který automaticky prohledává web a stahuje z něj dokumenty (fáze označovaná jako web crawling), dále z indexovače, který stažené dokumenty analyzuje (parsing) a indexuje (zpravidla nejvýznamnější části dokumentu), a konečně rozhraní, přes které je možné informace uložené v databázi indexu získat.[2]

Indexovací vyhledávací stroje bývají často primárním vyhledávacím nástrojem tzv. webových portálů. Jsou pravděpodobně jedinou rozumnou cestou, jak zpracovat (více či méně dostatečně) současnou nabídku informací dostupných na webu.

4. Metavyhledávače

Metavyhledávače jsou v ryzí podobě jen rozhraní, přes která je možné rozeslat dotaz na více primárních vyhledávacích služeb současně. Tyto systému však zpravidla získané odpovědi dále zpracovávají – například vyřazují duplicity nebo na získaná data aplikují pro dosažení lepších výsledků různé heuristiky. Mezi metavyhledávače patří například systém Ahoy! pro vyhledávání domovských stránek uživatelů webu, který je popsán v části Specializované vyhledávací systémy.

Není vyloučeno (naopak, je to častý případ), že si metavyhledávač buduje vlastní databázi. To může být výhodné například pro urychlení odpovědi. Databáze bývá (podle zaměření metavyhledávače) úzce specializovaná a její význam je z hlediska pokrytí informací dostupných na webu samozřejmě zanedbatelný.

1 Poslední aplikace indexů tento rozdíl stírají – viz popis systému Google v části Univerzální vyhledávací systémy.
2 Výčet komponent je neúplný.

Dotazy na vyhledávače

1. Klíčová slova, masky, fráze, pole a limity

1.1. Klíčová slova

Nejjednodušší a při vyhledávání na WWW také nejčastějším typem dotazu je dotaz složený ze seznamu několika slov[1]. Vyhledávací stroj pak prohledá svoji databázi a na základě svých interních technik (četnost slov, PageRank) nabídne nejrelevantnější dokumenty. Seznam nabídnutých odpovědí bývá velmi rozsáhlý.

1.2. Masky

Mnoho vyhledávacích systémů podporuje maskování klíčových slov pomocí speciálních znaků, někdy též označovaných jako žolíkové znaky (wildcards). Jsou známé i z jiných aplikací, například shellů OS nebo nástrojů pro zpracování textu. Často používaným speciálním znakem je ? (někdy %) pro vyjádření jednoho libovolného znaku a * pro vyjádření libovolného řetězce znaků. Např. klíčové slovo maskované jako *hled* vyhovuje slovům vyhledávání, hledat, ap.

1.3. Fráze

V případě, že seznam klíčových slov v dotazu tvoří frázi a vyhledávací systém fráze podporuje, je samozřejmě vhodné dotaz jako frázi položit. Fráze se ve většině systémů vytváří tak, že se posloupnost klíčových slov uzavře do uvozovek. V odpovědi jsou pak vráceny jen ty dokumenty, které vykázaly alespoň jeden výskyt celé fráze.

1.4. Pole a limity

Limity jsou dalším možným rozšířením dotazu, které může zúžit množinu vrácených odpovědí. Příkladem specifikace limitu je určení maximálního „stáří“ hledaného dokumentu. Jiným příkladem limitování dotazu je specifikace jazyka, ve kterém je dokument napsán. Limity jsou často vyjadřovány ve formě omezujících polí. Mnoho vyhledávacích systémů podporuje pole url, title, link a image (význam je zřejmý). Dotaz omezený pomocí polí by mohl vypadat např. takto:

```
"Linux 6.2" url:www.redhat.cz link:www.redhat.com
```

Tento uměle vykonstruovaný dotaz se ptá na stránky, které obsahují frázi `"Linux 6.2"`, jsou uloženy na `www.redhat.cz` a vede z nich odkaz na `www.redhat.com`.

2. Booleovské dotazy

Specifikace dotazů uvedené v části 1 lze ve většině vyhledávacích systémů kombinovat pomocí logických operátorů. Mezi často podporované operátory patří AND, OR, NOT. Mezi méně časté operátory patří např. NEAR, který vyjadřuje vztah termů „být blízko od sebe“.

Kromě logických operátorů bývá k dispozici kvalifikátor (ne)existence, vyjadřovaný pomocí znaménka `(-)+`. Například dotaz `+Linux -SUSE` vrátí stránky, které obsahují slovo Linux a neobsahují slovo SUSE.

1 Některé vyhledávací systémy rozlišují velká a malá písmena (case sensitive).

3. W3QL a W3QS

WWW Query Language (W3QL) [22] je strukturovaným dotazovacím jazykem podobným SQL, který vytvořili David Konopnicky a Oded Shmueli z Izraele. W3QL je vykonáván strojem W3QS a umožňuje tvořit dotazy na obsah i strukturu webu. Mezi další vlastnosti jazyka patří například schopnost tvořit pohledy v podobném významu jako v SQL.

W3QS používá při zpracování dotazu služeb existujících vyhledávacích systémů. Je implementován pod OS Unix a chová se jako běžné unixové nástroje. Nabízí aplikacím API rozhraní v podobě knihovny objektů napsaných v jazyce Perl. Obsahuje nástroje, které se pokoušejí pomocí technik strojového učení vyplňovat formuláře a tím získávat i dokumenty pro stroje vyhledávající nad WWW obvykle nedostupné.

Příklad: W3QL dotaz[1]

```
Select
From n1,l1,(n2,l2),l3,n3
Where
n1 in {http://www.cs.technion.ac.il};
l2 in {/technion\.ac\.il/};
l3 in {/technion\.ac\.il/};
n3: PERLCOND 'n3.format=~/image/'
Using ISEARCHd -d 5 -l 1000
```

Uvedený příklad ukazuje použití systému pro vyhledání dokumentů, které vyhovují podmínce dané výrazem n1, l1, (n2, l2), l3, n3 ve významu:

- n1 hledáme dokumenty z domény `http://www.cs.technion.ac.il`,
- l2 odkaz na dokument musí obsahovat řetězec `technion.ac.il`,
- n3 dokument musí být obrázek.

Pro hledání se použije `ISEARCHd`, který položí maximálně 1000 HTTP dotazů a maximální zanoření ve WWW struktuře dokumentů bude 5 úrovní.

1 Převzato z [22].

Používané techniky a technologie

1. Roboti

Roboti (též spiders, crawlers) jsou programy[1], které automaticky procházejí hypertextovou strukturu Webu, načítají dokumenty a rekurzivně následují v nich obsažené odkazy. Mohou být použiti ke stahování dokumentů, k validaci odkazů ap. Roboti jsou součástí vyhledávacích systémů.

Pro roboty je definován *Robots Exclusion Standard* – soubor pravidel určující, jak by se měl robot na navštíveném serveru chovat. Pravidla lze použít na úrovni adresáře serveru vytvořením souboru `robots.txt`, např:

```
User-agent: webcrawler
Disalow: /tmp
```

nebo na úrovni jednotlivých HTML stránek pomocí tagu `META`:

```
<META name="ROBOTS" content="NOINDEX">
```

2. Databázové technologie

Dnešní informační systémy (IS) pracují s velkými objemy strukturovaných dat, které tvoří databázi (vyhledávací systémy na WWW nejsou výjimkou). Jejich správu zajišťuje zvláštní subsystém – systém řízení báze dat (SŘBD). SŘBD obsahuje jazyk pro definici databáze (Data Definition Language – DDL), jazyk pro manipulaci s daty (Data Manipulation Language, DML) a případně další jazyky. Snahou projektantů nových SŘBD je integrovat tyto dílčí jazyky do jednoho databázového jazyka, jakým je např. dnes velmi rozšířený standardizovaný jazyk SQL [12] pro relační databáze.

Relační databázový model [13] je dnes nejrozšířenější (kromě něj jsou známé ještě modely hierarchický a z něj odvozený síťový) a nejčastěji je realizován architekturou klient-server. Na serveru běží vlastní databázový stroj, jehož služeb využívá klient.

Kromě logického pohledu na data, vyjádřeného použitým databázovým modelem, lze na data pohlédnout i z hlediska fyzického uložení. Fyzicky jsou data ukládána obvykle do souborů [14]. Podle použité technologie může být soubor (index) implementován pomocí B-stromů, jako hashovaný index apod.

3. Strojové učení

Strojové učení je informační věda, která se zabývá aplikací strojů na problémy, k jejichž rozhodnutí je třeba jistá znalost. Vzhledem ke složitosti se zpravidla jedná o problémy úzce specializované, např. rozpoznávání písma.

Znalost potřebnou k rozhodnutí problému získá algoritmus ve fázi učení na tzv. trénovacích datech. Učící data mohou mít např. podobu vyplněné tabulky s alespoň jedním klasifikačním atributem. V tomto případě je úkolem algoritmu na základě ostatních dat tabulky správně určovat klasifikační atribut. K tomu může algoritmus například vybudovat tzv. rozhodovací strom (jedna z technik strojového učení). Jakmile je rozhodování algoritmu dostatečně přesné, řekneme, že je algoritmus naučený. K ověření úspěšnosti algoritmu se používají tzv. testovací data. Příkladem algoritmu strojového učení je např. algoritmus[2] C4.5.

1 Ukázku jednoduchého robota `inetf` lze nalézt v nepoužitých zdrojových textech vyhledávacího systému implementovaného v praktické části.
2 Popis algoritmu lze nalézt např. na adrese `www.kdnuggets.com`.

4. Datamining

4.1. Vysvětlení pojmu

Datamining je informační technologie, která se zabývá získáváním znalostí z již existujících naplněných databází informací. Často se mluví také o dolování znalostí. Datamining je velmi rozsáhlé odvětví, v němž se využívá strojové učení a další obory umělé inteligence. Ke klíčovým problémům patří nalezení zajímavého vzorku (clustering), popis vzorku (clasification) a hledání závislostí. Často se říká, že klasické technologie slouží k vyhledávání toho, co víme, že nevíme. Pomocí nástrojů datamining se můžeme posunout směrem k nalézání toho, co nevíme, že nevíme.

4.2. Kontextová analýza dokumentů InfoRating

Nástroje pro datamining jsou použity např. v komerční metodě Inforating [7]. Inforating je založen na fulltextové technologii firmy Verity a slouží k dolování znalostí z nestrukturovaných textových dokumentů. Na vstupu přijímá seznam analyzovaných dokumentů a soubor tzv. profilů tvořených jednoduchými dotazy (klíčová slova) nebo strukturovanými dotazy (topiky). Výstupem je kontextová matice, kde řádky odpovídají jednotlivým dokumentům a sloupce zadaným profilům. Prvky matice pak mají význam relevance dokumentu vzhledem k danému profilu a nabývají hodnot od 0 do 100.

4.2.1. Vyhodnocení obsahu dokumentu

Základem při ohodnocování obsahu dokumentů technologií Verity jsou fuzzylogické operátory a zejména pak operátor „acrue“. Operátor acrue pracuje na principu čím více, tím lépe. Tím jsou odstraněny potíže spojené s používaním operátorů „and“ (zvyšuje přesnost, ale snižuje úplnost odpovědi) a „or“ (naopak). Díky těmto operátorům lze se zadanou mírou rizika provádět i vyřazování duplicitních dokumentů a shlukování podobných dokumentů do tzv. „clusterů“.

4.2.2. Sestavování profilu dotazu

Kromě jednoduchého dotazu tvořeného klíčovými slovy může být dotaz i strukturovaný v podobě tzv. topiku. Topik je stromová struktura, jejíž uzly jsou tvořeny pomocí klíčových slov, jejich vah a operátorů (např. může existovat uzel [produkty; 0,5; and]). Operátor se aplikuje na potomky. Výběr slov je ovlivněn jejich příslušností do jedné z následujících kategorií:

- determinující – jednoznačně identifikují téma (např. název konkrétního produktu)
- asociující – mohou se vyskytovat i v jiné souvisloti (např. obecný název produktu)
- indikující – v určitých aspektech zvyšují důležitost dokumentu (např. ekonomické údaje)

Pomocí topiku lze realizovat tematické vyhledávání.

4.2.3. Zhodnocení technologie Verity InfoRating

Uvedená technologie může vhodně doplnit existující vyhledávací systém. Umožňuje zhodnocení relevance dokumentu pomocí fuzzylogických operátorů, tematické vyhledávání prostřednictvím dotazu v topiku, nalezení podobných dokumentů (clustery) a vyřazení duplicitních dokumentů. Kromě těchto hlavních vlastností umožňuje systém klást dotazy podle vzorového dokumentu.

5. XML

5.1. Definice pojmu

XML [20, 21] je značkovací jazyk (eXtensible Markup Language) odvozený od obecného standartu SGML (Standard Generalized Markup Language). Pomocí značkování vnáší jazyk do dokumentu strukturu. Na rozdíl od HTML nejsou značky předdefinovány. Podle mnohých prognóz je to konečně značkovací jazyk vhodný pro WWW. Dosud používaný jazyk HTML není rozšiřitelný (jeho syntaxe je pevně dána) a jeho poslední verze je pro nově nastupující platformy kapesních PC příliš složitá. Díky své univerzálnosti nalézá XML uplatnění jako jazyk pro popis publikací, databází, faktur ap.

5.2. Syntaxe a schémata XML

Dokumenty XML jsou výhradně textové. Stavebním kamenem jazyka XML jsou elementy, ze kterých se XML dokumenty skládají. Každý element má svůj název a obsah:

```
<telefon prednostni='true'>553321</telefon>
```

Obsah elementu je ohraničen počáteční a koncovou značkou (obě jsou povinné). Element může obsahovat v podobě atributů další informace. Celý XML dokument má stromovou strukturu, v níž musí být právě jeden kořenový element. Dále lze v dokumentech XML specifikovat deklaraci, entity, komentáře a instrukce pro zpracování.

Každý, z hlediska XML syntaxe dobře utvořený dokument nazýváme správně strukturovaný. Kromě toho však může být dokument XML ještě ověřován (validován) proti tzv. DTD (Document Type Definition). DTD je nástroj XML, který nám umožňuje definovat vlastní třídu XML dokumentů (např. faktury firmy X). V DTD se specifikují názvy elementů a jejich atributů, povolené hodnoty elementů, jejich vzájemné zanořování a podobně. Dokument vyhovující danému DTD nazýváme platný.

5.3 Nástroje XML

5.3.1. XSL, QXL

XSL (eXtended Stylesheet Language) [20, 21] je deklarativní jazyk, který slouží k transformaci XML dokumentů. Deklarace spočívají v definici stylů, které jsou tvořeny návěštími a odpovídajícími akcemi. Návěští vyjadřují cesty ve stromové struktuře XML dokumentu. Pokud je při procházení dokumentem taková cesta nalezena, pak se provede akce – generování výstupního kódu. XSL dále obsahuje několik užitečných předdefinovaných funkcí. Konkrétní příklad využití XSL lze nalézt v praktické části diplomové práce, kde je použit ke transformaci XML dokumentů do formátu HTML.

XQL [20] (eXtended Query Language) je nástroj, který rozšiřuje možnosti jazyka XSL při dotazování na obsah XML dokumentu. Pro tyto úlohy se v praxi často používají i tzv. parsery XML (SAX a DOM XML parsery – viz dále). Zatímco parsery jsou tvořeny rozhraním pro parsování XML dokumentů, XQL je deklarativní jazyk a poskytuje při dotazování na XML vyšší míru abstrakce.

Podobně jako XSL pracuje i XQL s cestami v XML dokumentech, jeho syntaxe je však pro vyjádření kontextu o něco bohatší. Následující dotaz například hledá v aktuálním kontextu element `book`, jehož atribut `style` má stejnou hodnotu jako atribut `specialty` elementu `bookstore` umístěném na nejvyšší úrovni stromové struktury XML dokumentu:

```
book[/bookstore/@specialty = @style]
```

Množina funkcí XQL je také bohatší. Oproti XSL jsou zde navíc relační operátory a specielně booleovské operátory. Následující dotaz hledá element `author`, který má mezi svými potomky alespoň jeden element `degree` a jeden element `award`:

```
author[degree $and$ award]
```

QXL je dostupný např. jako GMD–IPSI XQL Engine [19]. Jedná se o volně šiřitelný nástroj založený na třídách jazyka Java, které staví na XML DOM parseru. Kromě tříd jsou součástí balíku i programy umožňující klást dotazy z příkazové řádky.

5.3.2. Analyzátory XML

Analyzátory jsou nástroje XML, které ocení především vývojáři aplikací na XML založených. Analyzátor je softwarová komponenta umístěná mezi vlastní aplikaci a soubory XML. Jejím úkolem je oddělit programátora od záludností syntaxe XML. Podle způsobu práce rozlišujeme analyzátory DOM (budují v paměti strom XML dokumentu) a SAX (generování událostí při parsování XML dokumentu). Analyzátory (též parsery) jsou dostupné v podobě knihoven pro různé programovací jazyky, mezi nimiž však vyniká jazyk Java.

5.3.3. Transformace XML

Pro potřeby zobrazení lze přidat ke XML dokumentu formátovací objekty XSLFO nebo styly CSS [20, 21]. Generátory XML jsou pomůcky, pomocí nichž se XML dokumenty vytvářejí. Kromě nich existuje ještě celá řada nástrojů usnadňujících práci s XML. Patří mezi ně editory, rozšířené odkazy (XLink) a v neposlední řadě také prohlížeče XML. V současné době podporují XML jen poslední verze prohlížečů WWW (Mozilla, MS IE 5, Netscape 6).

Univerzální vyhledávací systémy

1. Google

1.1. Úvod

Google (`www.google.com`) byl vyvinut studenty Sergey Brinem a Lawrence Pagem na univerzitě ve Stanfordu. Jejich cílem bylo implemetovat vyhledávací systém, který by byl dostatečně adaptabilní pro vyhledávání na neustále rostoucím webu a obsahoval inovativní prvky využívající jeho hypertextovou strukturu. Následující přehled shrnuje požadavky autorů na vlastnosti implementovaného systému:

- maximální a co možná nejaktuálnější pokrytí WWW
- rychlá odezva (miliony dotazů za den)
- snadná škálovatelnost (nutné vzhledem k růstu prostoru WWW)
- maximálně relevantní odpovědi (uživateli stačí projít několik prvních nabídnutých odpovědí a nalezne hledanou informaci)
- otevřený systém (technologie věřejně dostupná, např. pro akademické využití)

V dalších částech textu popisuji architekturu a techniky používané vyhledávacím systémem Google. Text čerpá z volně dostupných materiálů [2, 3] autorů systému.

1.2. Používané techniky

1.2.1. PageRank (hodnota stránky)

Systém Google řadí v odpovědi odkazy na relevantní stránky velmi efektivně. Pokud je dotaz dostatečně specifikován, jsou informace, které nás zajímají, ve většině případů nabídnuty v odpovědi mezi prvními (viz statistiky v části 1.4.). Této skutečnosti systém dosáhuje zejména díky zpracování informace uložené ve struktuře odkazů prostoru WWW, ze které počítá tzv. PageRank (ohodnocení stránky).

Odkazy na WWW stránky lze přirovnat k citacím v akademické literatuře. PageRank z této abstrakce vychází, zároveň ji však podstatně rozšiřuje. PageRank stránky je přímo úměrný počtu odkazů na tuto stránku, jednotlivé odkazy však nepřispívají pro skóre stránky stejným dílem. Např. odkaz z velmi odkazované stránky bude mít pro PageRank stránky vyšší přínos než odkaz ze stránky odkazované jen velmi málo. Dále je přínos každé na danou stránku se odkazující stránky tím nižší, čím více odkazů obsahuje. Nyní již můžeme uvést zjednodušenou[1] formální definici PageRanku stránky.

Definice

Nechť $T_1, \ldots, T_n$ jsou stránky ukazující na stránku A („citace stránky A“). Nechť $C(T_i)$ je počet odkazů jdoucích ze stránky T_i. Nechť d je faktor útlumu z intervalu (0, 1). Pak PageRank stránky A lze vypočítat jako:

$$PR(A) = (1-d) + d(PR(T_1)/C(T_1) + \ldots + PR(T_n)/C(T_n))$$

1 V systému, pro který je PageRank počítán, se může vyskytnout dvojice webových stránek, které na sebe vzájemně ukazují a zároveň neukazují na jiné webové stránky. V tomto případě by se použitím uvedené jednoduché metody výpočet zacyklil (řešení viz [3]).

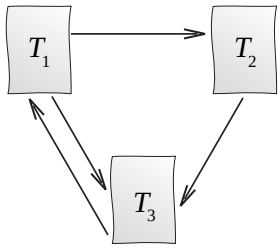
PageRank lze vypočítat jednoduchou iterační metodou – je to v podstatě vlastní vektor normalizované matice odkazů prostoru WWW (viz příklad dále). Před výpočtem je třeba shromáždit odkazy všech indexovaných stránek. Pro několik set miliónů odkazů trvá na současné běžné stanici výpočet jen několik hodin.

Alternativní intuitivní vysvětlení PageRanku

O PageRanku můžeme uvažovat jako o modelu chování surfujícího uživatele. Nechť tento uživatel navštíví náhodnou počáteční stránku. Nechť se tento uživatel nikdy nevrací zpět a občas, nezajímá-li ho oblast stránek, ve které se právě nachází, přejde na novou počáteční stránku. Pravděpodobnost návštěv jednotlivých WWW stránek odráží právě PageRank. Faktor útlumu d vyjadřuje pravděpodobnost, že uživatel zadá novou výchozí stránku.

Příklad: výpočet PageRanku

Přesný popis výpočtu PageRanku lze nalézt např. v [3], já jej zde budu demonstrovat na jednoduchém příkladu.



Mějme systém webových stránek T_1, T_2, T_3 vzájemně propojených odkazy (viz obr.). Sestrojíme normalizovanou čtvercovou matici sousednosti A stránek T_1, T_2, T_3:

$$A = \begin{pmatrix} 0 & 0 & 1 \\ 0,5 & 0 & 0 \\ 0,5 & 1 & 0 \end{pmatrix}$$

Nechť $u, v \in \{T_i\}$. Pokud vede ze stránky u odkaz na stránku v, pak je prvek a_{uv} matice A roven hodnotě $1/N_u$, kde N_u je počet odkazů jdoucích ze stránky u. V ostatních případech je $a_{uv} = 0$.

Pagerank R vypočítáme jako vlastní vektor matice A, tj. $R = cAR$, kde c je vlastní číslo matice A. Výpočet probíhá iterativně, přičemž za „startovní vektor“ volíme libovolný nedegenerativní vektor. V našem případě by mohl výpočet probíhat například takto:

$$\begin{pmatrix} 0 & 0 & 1 \\ 0,5 & 0 & 0 \\ 0,5 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1,5 \\ 1,5 \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 1 \\ 0,5 & 0 & 0 \\ 0,5 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1,5 \\ 1,5 \end{pmatrix} = \begin{pmatrix} 1,5 \\ 0 \\ 0,5 \end{pmatrix}, \ \ldots \ , \begin{pmatrix} 0 & 0 & 1 \\ 0,5 & 0 & 0 \\ 0,5 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0,45 \\ 0,95 \end{pmatrix} = \begin{pmatrix} 0,95 \\ 0,5 \\ 0,95 \end{pmatrix}$$

Po normalizaci vektoru (0,95; 0,5; 0,95) získáme výsledný vektor PageRanku $R = (0,4; 0,2; 0,4)$. PageRank stránky T_1 je tedy 0,4; T_2 0,2 a T_3 0,4.

1.2.2. Text odkazu (Anchor Text)

Ve většině vyhledávacích systémů je text odkazu asociován se stránkou, ve které se odkaz nachází. Google asociuje text odkazu se stránkou (jejím URL), na kterou odkaz ukazuje.[1] Hlavní důvod použití této techniky tkví v tom, že texty odkazů na danou stránku zpravidla poskytují o této stránce více informací, než stránka samotná. Navíc je takto možné indexovat dokumenty, které nejsou textové povahy (např. obrazová data).

1.2.3. Ostatní používané techniky

Kromě PageRank a Anchor Text používá Google ještě další techniky pro zlepšení poskytovaných výsledků. Jedná se například o uchovávání pozice, velikosti písma a dalších atributů každého slova (dohromady hitu) asociovaného s danou stránkou.

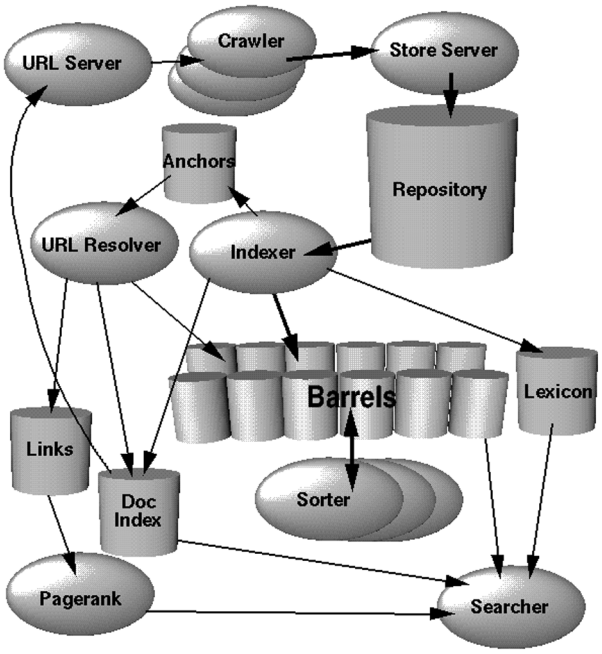
1 Idea propagace textu odkazu na odkazovanou stránku byla původně implementována ve World Wide Web Wormu (jeden z prvních roborů na WWW), jednak z důvodů možnosti indexovat netextové dokumenty a také proto, že takto bylo k dispozici více informací o více stránkách.

1.3. Architektura systému

1.3.1. Přehled architekury Google

Většina kódu Google byla z důvodu efektivity napsána v C nebo C++. Kód je provozovatelný (zřejmě odzkoušený) na Solaris a Linux.

Stahování WWW stránek (web crawling) je realizováno několika paralelními ‚stahovači' (crawlers). URL požadovaných stránek je dodáván URLserverem. Stránky jsou staženy a zaslány ukládacímu serveru (store server). Ten je zkomprimuje a uloží do skladiště stránek (repository). Každá stažená stránka je uložena s URL a indexovačem (indexer) přiděleným unikátním číslem, tzv. docID. DocID je rovněž ihned přiděleno i všem dokumentům (jejich URL), na které ukazují odkazy právě staženého dokumentu. Vzniká tak index dokumentů (document index), z něhož čerpá např. URLserver URL ještě nestažených stránek.



architektura Google

V následujícím kroku je provedeno indexování, které zajišťují moduly indexovač (indexer) a třídič (sorter). Indexovač načítá stránky ze skladiště, dekomrimuje je a parsuje. Z každého dokumentu je extrahována množina slov a z každého slova pak informace označené jako hit. Hit obsahuje pozici slova v dokumentu, aproximaci jeho velikosti a příznak udávající, zda je slovo napsáno velkými písmeny. Slovo je dále zařazeno do lexikonu a nahrazeno jeho wordID. Indexer poté ukládá wordID slov s jejich hitem do tzv. barelů (barrels), vzniká částečně setříděný tzv. předběžný (forward) index (klíčem je zde docID). Indexer vykonává ještě další důležitou činnost. Extrahuje z dané stránky všechny její odkazy do souboru odkazů (anchors file). Položky souboru obsahují všechny důležité informace o odkazu, tj. odkud a kam odkaz směřuje a text odkazu.

Před aplikací modulu třídič je ještě vyvolán modul pro zpracování URL (URLresolver). Čte soubor odkazů a mění relativní odkazy na absolutní. Text odkazů spolu s docID dokumentu, na který odkaz ukazuje, ukládá do předběžného indexu. Kromě toho odkazy asociuje s docID dokumentu, ve kterém se nacházejí, a tyto informace ukládá do databáze odkazů (links). Tato databáze se použije pro výpočet skóre PageRank jednotlivých stránek.

Třídič zpracovává předběžný index uložený v barelech a vytváří k němu index inverzní setříděný podle wordID.

Dalším modulem, který využívá uživatel systému Google, je hledač (searcher). K zodpovězení dotazu využívá lexikon, který ke slovům zadaným ve formuláři najde jejich wordID. Poté vyvolá dotaz s wordID na invertovaný index. Z vrácených docID položí dotaz na PageRank modul. Ten vrátí docID dokumentů s nejvyšší hodnotou PageRank. Nakonec je docID převedeno pomocí indexu dokumentů na URL a je vrácena odpověď.

1.3.2. Datové struktury systému Google

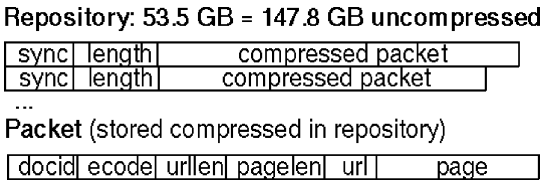
Datové struktury Google jsou optimalizovány tak, aby mohly být i přes svoji velikost zpřístupňovány rychle. Na disk jsou například ukládány tak, aby se při hledání provedlo co nejméně (pokud možno jedno) vystavení hlav na danou stopu. Na rychlost má samozřejmě vliv i vývoj HW technologií – přístupová doba disků je dnes běžně 10 ms, kapacity se pohybují v desítkách GB.

BigFiles (velké soubory)

Nabídka systémů, na kterých byl Google vyvíjen, byla z hlediska maximální možné velikosti souboru nedostatečná. Proto byl vyvinut systém BigFiles. BigFiles jsou virtuální soubory s 64bitovým adresováním. Jsou realizovány jako několik skutečných souborů s nižším rozsahem adres, které použité systémy (Linux, Solaris) v současné době nabízejí (32bitový přístup). Systém BigFiles zajišťuje automatickou alokaci a dealokaci deskriptorů použitých skutečných souborů, stejně tak jako volitelnou kompresi virtuálního souboru.

Skladiště (repository)

Skladiště obsahuje úplný HTML text každé stránky. Stránky se ukládají komprimované. Ke komprimaci byl použit algoritmus zlib (viz RFC1950). Z hlediska stupně dosahované komprese je sice lepší algoritmus bzip dosahující poměru 4 : 1, zlib však při komprimaci dosahuje vyšší rychlosti (kompresní poměr zlib je průměrně 3 : 1). Ve skladišti jsou dokumenty uloženy „jeden za druhým“, každý dokument s prefixem údajů docID, délka a URL (viz obr.).



Struktura dat v repository
(údaje o kapacitě z roku 1999)

Index dokumentů

Index dokumentů obsahuje informace o každém dokumentu. Primárním klíčem tohoto indexu je docID. Index je tvořen dvěma soubory.

Dotazem na docID do prvního souboru lze získat aktuální status dokumentu, ukazatel do skladiště dokumentů na data dokumentu (odtud lze získat např. URL dokumentu), kontrolní součet a několik statistických údajů vedených pro každý dokument.

Druhý soubor je určen pro konverzi URL na odpovídající docID. Jedná se o seznam jednoznačných součtů pořízených z jednotlivých URL a odpovídajících docID. Při hledání docID k URL se tedy z URL vypočítá kontrolní součet a z něj se prohledáním souboru určí docID.

Lexikon

Lexikon obsahuje několik miliónů slov (původní implementace Google měla lexikon s 14 miliony slov) a pro každé z nich uchovává jednoznačný identifikátor – wordID.[1] Pro rychlý běh systému je vhodné, aby byl celý lexikon v operační paměti (pro lexikon s 14 miliony slov je potřeba 256 MB RAM). Lexikon je implementován odděleně ve dvou různých datových strukturách. Jedna z nich je implementována jako soubor, ve kterém jsou jednotlivá slova oddělena nulami. Ve druhém případě jsou slova uložena v hashované databázi.

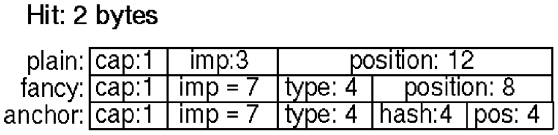
Datové struktury realizované v barelech

a) Seznam hitů

Seznam hitů byl pro svůj rozsah oddělen od předběžného i invertovaného indexu (z obou jsou do něj odkazy). Z důvodu rozsahu bylo také nutné vyvinou jeho co možná nejefektivnější implementaci. Autoři se v podstatě rozhodovali mezi třemi možnostmi – jednoduchou reprezentací hitů pomocí 3 čísel typu integer, kompaktní reprezentací (vlastní systém alokace bitů pro jednotlivé atributy) a konečně Huffmanovým kódováním. Nakonec se jako nejlepší jevila kompaktní reprezentace – vzhledem k jednoduché reprezentaci znamená mnohem lepší využití paměti a vzhledem k Huffmanovu kódování zase lepší přístup k jednotlivým atributům.

1 Každé slovo je opatřeno ještě dalšími informacemi.

Kompaktní kódování používá pro jeden hit jen dva bajty paměti. Hity jsou rozděleny na dva druhy –jednoduchý (plain) pro atributy běžného slova textu a rozšířený (fancy) pro spec. případy jako atributy META tagu, atributy textu odkazu apod. Speciálním případem rozšířeného hitu je hit pro text odkazu (anchor hit). Rozdělení bitů jednotlivých 2bajtových hitů ukazuje obr.



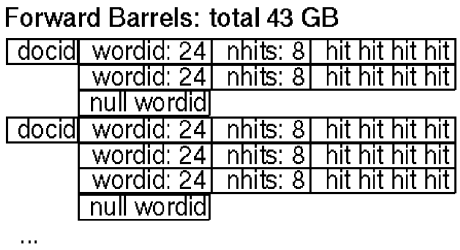
Rozdělení bitů v jednotlivých 2bajtových hitech

Jednoduchý hit používá jeden bit na příznak velkých písmen (cap – capitalisation), dále 3 bity pro označení velikosti textu (imp – importance, hodnoty 1–6) a konečně 12 bitů na označení pozice v dokumentu (pokud pozice překročí 4096, pak je nastaveno 4096). Rozšířený hit má 3 bity pro označení textu nastaveny na 1 (tj. hodnota 7), následují 4 bity pro označení typu rozšířeného hitu a 8 bitů pro označení pozice slova, k němuž se hit vztahuje (v textu). Konečně hit pro text odkazu má posledních 8 bitů alokováno jinak než standardní rozšířený hit. Rozděluje je na dvě části – první čtyři bity indikují pozici slova v odkazu a zbylé 4 bity jsou určeny pro hashování docID dokumentu, ve kterém se text odkazu vyskytuje (slova textu odkazu jsou vázána na docID dokumentu, na který odkaz ukazuje!). To může být užitečné např. při hledání frází, ovšem funkční jen v případě, že pro dané slovo neexistuje příliš mnoho odkazů (vzhledem ke 4 bitům alokovaným pro hash), které toto slovo obsahují.

b) Dopředný index

Dopředný index je částečně setříděný. Toho je dosaženo jeho rozdělením do několika skupin – dopředných barelů (forward barrels). Každý barel slouží pro uchovávání určitého rozsahu slov zastoupených jejich wordID.

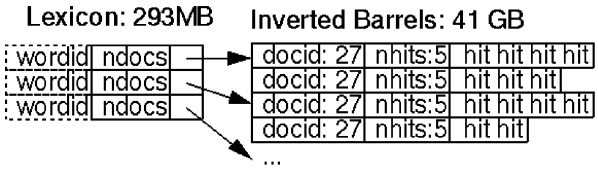
Pokud je v dokumentu indexovačem nalezeno slovo k uchování, pak se opatří jeho hit-em a identifikátorem wordID. Jako struktura sestávající z docID (podle kterého je dopředný index sestaven), wordID[1], nhits (počet hitů) a seznamu hitů se poté uloží do daného (podle wordID) barelu dopředného indexu. Ukládaná struktura je zobrazena na obrázku.



Struktura položek v barelech dopředného indexu

c) Inverzní index

Obsah inverzního indexu je z hlediska množství uchovávané informace totožný s indexem dopředným. Data každého dopředného barelu jsou přepracována modulem sorter do odpovídajícího invertovaného barelu.



Struktura dat v inv. barelu

K realizaci indexu je využit lexikon. Každé platné wordID je totiž v lexikonu svázáno s ukazatelem do invertovaného barelu, do kterého wordID spadá. Cílem odkazu je seznam položek invertovaného barelu. Ty sestávají z docID dokumentu, ve kterém se slovo vyskytuje, následovaného seznamem hitů slova. Takto je možné k danému slovu z lexikonu (jeho wordID) zjistit jeho výskyty ve všech dokumentech, a to včetně atributů tento výskyt provázejících.

Důležitou otázkou je, jak položky v invertovaných barelech uspořádat, aby bylo jejich prohledávání co nejefektivnější. Nejjednodušší uspořádání podle docID vede k efektivnímu zodpovídání

1 Z důvodů úspory místa v barelech se wordID v barelech ukládá jako relativní přírůstek k minimálnímu wordID daného barelu. Pak je možné kódovat wordID slov v barelech pomocí 24 bitů.

víceslovních dotazů – seznamy docID k jednotlivým slovům dotazu lze při tomto uspořádání rychle slučovat. Jiná varianta – uspořádání podle četnosti výskytu odpovídajícího slova ve všech dokumentech – vede k rychlému zodpovídání jednoslovních dotazů, slučování seznamů docID k jednotlivým slovům je však mnohem náročněší. Konečné řešení bylo kompromisem mezi oběma variantami. Invertované barely sestávají ze dvou sad. Jedna uchovává jen položky, jejichž hity jsou typu titulek (title) nebo text odkazu (anchor hits). Druhá sada uchovává všechny položky. Při dotazování je nejdříve prohledávána první sada, druhá (rozsáhlá) pak jen v případě nedostatečného počtu shod v první sadě.

1.3.3. Mapování prostoru WWW

Systém pro stahování webových dokumentů se skládá z jednoho URLserveru, který distribuuje URL stránek určených ke stažení několika paralelním stahovačům (crawlers, typicky běží 3). URLserver i stahovače jsou implementovány ve skriptovacím jazyku Python.

Každý stahovač má obvykle otevřeno kolem 300 spojení najednou. Ve špičkových intervalech systém pomocí 4 stahovačů získává 100 stránek za sekundu, což odpovídá průměrnému datovému toku 600 kB/s. Z důvodů omezení dotazů na DNS servery[1] (úspora času) si každý stahovač udržuje vlastní DNS vyrovnávací paměť (DNS cache).

Každou transakci stahovače lze popsat ve 4 krocích: dotaz na DNS, napojení na server, specifikace požadovaného dokumentu, přenos odpovědi. Z hlediska řízení je stahovač asynchronní systém, který k řízení přechodů ze stavu do stavu používá několik front.

Stahovače dodržují protokol o přístupu robotů ke zdrojům WWW (robot exclusion protokol) a řeší i některé obskurdní problémy, které při stahování rozličných WWW stránek mohou nastat[2].

1.3.4. Indexování WWW

Indexování probíhá v několika krocích (viz dříve přehled architektury Google a datové struktury Google). V tomto oddíle jsou podrobněji popsány některé fukční části indexovače – parser, primární indexovač budující předběžný index a třídič, který vytváří invertovaný index.

Parser

Vzhledem k tomu, že převážná většina HTML dokumentů na webu nedodržuje syntaxi jazyka, je implementace parseru pro HTML dokumenty náročný úkol. Z důvodů dosažení co nejvyšší rychlosti parsování bylo upuštěno od implementace CFG parseru pomocí nástroje YACC. Parser je napsán s využitím lexikálního analyzátoru (pravděpodobně FLEX), který byl z důvodu nutnosti rozpoznávat CFG jazyk HTML rozšířen o zásobník.

Primární indexování

Každé indexované slovo je pomocí lexikonu převedeno na odpovídající wordID. Lexikon je z důvodů rychlosti systému udržován v operační paměti a je sdílen několika paralelně pracujícími moduly. To přináší určité problémy s jeho aktualizací. Pokud je například zpracováváno slovo, které není v lexikonu obsaženo (málo pravděpodobné), pak je zapsáno do speciálního malého souboru slov a ten je v případě potřeby zpracováván speciálním indexovačem bez paralelizace.

Mnoho událostí, vznikajících při indexaci, je logováno.

Třídění

Generování inverzního indexu je paralelizováno. Je použito tolik počítačů, kolik jich je právě k dispozici. Každý přitom zpracovává jeden barel, přesněji řečeno třídí položky jednoho předběžného barelu podle wordID do jednoho invertovaného barelu.

1 DNS server slouží k převodu IP adresy na doménovou adresu a naopak.
2 Autoři např. uvádějí stahování stránek online provozované hry – ty je samozřejmě stahovat nevhodné.

1.4. Výsledky Google, srovnání s jinými systémy

Systém Google nabízí ve srovnání s jinými vyhledávacími službami WWW podobného zaměření inovativní techniky pro zvýšení kvality hledání, jako např. PageRank, Anchor Text a zpracování mnoha informací každého indexovaného slova. Je dobře škálovatelný, takže má budoucnost i vzhledem ke stále rostoucímu prostoru WWW. Na položené dotazy poskytuje relevantní odpovědi zařazené mezi prvními. Zkoušel jsem zjistit také něco o záporech systému Google. Databáze systému v současné době obsahuje více než 1 miliardu odkazů. Zajímal mě zejména podíl duplicitních, neaktuálních a „mrtvých“ odkazů obsažených v databázi. Tyto údaje lze společně se srovnáním s dalšími vyhledávacími systémy nalézt v následujících tabulkách. Uvedené hodnoty jsou z roku 2000 a byly získány ze statistik uvedených na adrese `http://www.notess.com/search/stats/`.

Test na duplicity

Vyhledávací systém	unikátní odkazy
Fast Search	27 %
AltaVista	18 %
Excite	23 %
Northern Light	8 %
Google!	7 %

Test na „mrtvé“ a nedostupné odkazy

Vyhledávací systém	mrtvé	nedostupné
Fast	2,3 %	1,8 %
Google!	4,3 %	3,3 %
Northern Light	5,7 %	2,0 %
Excite	8,7 %	5,7 %
AltaVista	13,7 %	9,3 %

Další statistiky

Vyhl. systém	počet odkazů v milionech	a relevance odpovědi
Google	470	Ve 13 z 25 odpovědí zařadil relevantní odpovědi mezi prvními, z toho 2 vazby na jiné vyhledávače.
All the Web	–	V 10 z 25 odpovědí zařadil relevantní odp. mezi prvními, bez vazeb.
iWon Advanced	365	Ve 3 z 25 odpovědí zařadil relevantní odpovědi mezi prvními, z toho 1 vazba na Google.
Northern Light	380	V 1 z 25 odpovědí zařadil relevantní odpovědi mezi prvními, jde ale o vazbu na Google.
AltaVista	294	Ani v jedné z 25 odpovědí nezařadila relevantní odpovědi mezi první.

2. WebWatcher

Webwatcher [8, 10] je vyhledávač, který byl vyvíjen jako univerzitní projekt na Carnegie Mellon University. Jedná se o starší projekt a v současné době již není veřejnosti přístupný (v provozu 1995–7) , nicméně díky použití neobvyklých a zajímavých technik stojí jistě za zmínku. Vyhledávač patří do kategorie tzv. adaptivních vyhledávačů. Adaptivní vyhledávače používají při zpracování dotazu speciálních heuristik založených na strojovém učení, umělé inteligenci ap. To, co však činí WebWatcher zajímavým, je technika zvaná *reinforcement learning* (stimulované, zpětnovazebné učení). Systém při zodpovídání dotazů sleduje reakce uživatele. Získané poznatky ovlivňují budoucí rozhodování systému, např. při určování relevance dokumentů vrácených v odpovědi.

Specializované vyhledávací systémy

1. Cora

Portál Cora (`cora.whizbang.com`), určený pro hledání vědeckých článků, je aplikací výsledků výzkumného projektu *Just Research*[1] [16]. Cílem Just Research je pomocí technik umělé inteligence a strojového učení zautomatizovat budování specificky zaměřených portálů. Na výzkumu pracuje Andrew McCallum spolu se studenty Carnegie Mellon University (Kamal Nigam, Kristie Seymore, Jason Rennie, Jason Reed a Huan Chang). Systém je stále ve vývoji.

Portál je místo na Internetu, kde jsou shromažďovány informace určitého typu (například `www.linuxstart.com` je potrál o linuxových zdrojích). Často je budován s vynaložením velkého úsilí lidí. Na základě znalosti o zaměření portálu a s využitím pokročilých technik lze však tuto činnost v mnohém zautomatizovat.

Na portálu Cora jsou články automaticky zatřiďovány do témat. K tomu se využívá skrytých Markovových modelů. Pro efektivní vyhledávání dokumentů se používá reinforcement learning.

Cora umožnuje hledání podle klíčových slov. Lze použít operátory +, –, title:, reference:, author:, url: a slovní fráze. Výsledek je vrácen ve formátu název, autor a abstrakt, automaticky je generován záznam ve formátu BibTeX. Relevantní dokumenty jsou seřazeny podle vah a četností klíčových slov z dotazu a jsou u nich poskytnuty i odkazy na dokumenty, které je citují a které jsou z nich citovány.

1.1. Efektivní vyhledávání dokumentů

Každý autonomně pracující systém, který získává na Internetu informace, využívá pro tuto činnost robota. V případě systému Cora využívá robot při hledání nových dokumentů znalosti o zaměření portálu (též znalosti o doméně). Tím se odlišuje od robotů univerzálních vyhledávačů, kteří se snaží najít dokumentů co nejvíce. Jeho schopnosti najít relevantní dokumenty se navíc neustále zlepšují, jelikož pro svoji činnost používá techniku reinforcement learning[2] (učení se zpětnou vazbou).

Některé systémy pro efektivní vyhledávání tematicky zaměřených dokumentů obsahují algoritmy, které se nejdříve naučí rozpoznat dokumenty podobné dokumentům z učící sady. Na základě získané znalosti poté rozhodují o relevanci nalezených dokumentů. Nevýhoda této metody spočívá v závislosti na učící množině, která nemusí pokrýt různorodost relevantních dokumentů.

1.1.1. Reinforcemenet learning

Roboti začínají na domovských stránkách oddělení a laboratoří zabývajících se informačními technologiemi. Relevantní dokument musí být postscriptový, musí mít části abstrakt a reference. Takový dokument budeme dále nazývat odměna. Podstatou učení robota je naučit se zvolit takový postup, aby počet odměn byl v co nejkratší době co nejvyšší.

Za tímto účelem zavedeme množinu stavů $s \in S$, množinu akcí $a \in A$, přechodovou funkci T: $S \times A \rightarrow S$ a funkci odměny R: $S \times A \rightarrow \mathbf{R}$. Stavu při učení robota odpovídá počet již nalezených a počet ještě nenalezených dokumentů[3], akci odpovídá volba některého odkazu. Cílem učení je najít takovou *politiku* π: $S \rightarrow A$, která maximalizuje množství v čase získaných odměn. Nyní již můžeme zavést pojem *hodnota stavu*:

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t r_t$$

1 JustResearch je výzkumná organizace v USA pracující v letech 1996–2000 pro korporaci Justsystem z Japonska. Hlavní náplní práce byly aplikace strojového učení v různých oblastech IT.

2 Reinforcement learning – Kaelbling, Littman, Moore, 1996.

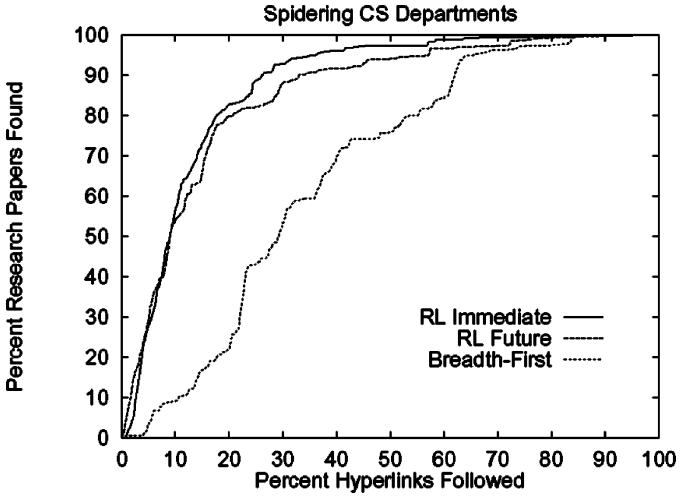
3 Ve skutečnosti je použito zjednodušení.

r_t vyjadřuje odměnu získanou v t krocích při startu ve stavu s, $0 \square \quad 1$ jsou srážky odměn (s rostoucím časem rostou). Naší snahou je najít optimální politiku $\square^*$, která pro všechny stavy s maximalizuje hodnotu funkce $V^{\square}(s)$, kterou v tomto případě označíme $V^*(s)$. V praxi však místo ní často používáme její korelovanou hodnotu Q (viz např. [16]).

K naučení optimální politiky $\square^*$ předložíme algoritmu učící data. Ta jsou tvořena mapou dokumentů a z nich vedoucích odkazů nějaké konkrétní webové sítě. Po nalezení $\square^*$ následuje hledání aproximační fuknce F:

$$Q(s, a) = F(text(a)) \quad \text{pro stav } s \text{ a akci } a.$$

Funkce F pro každý nalezený dokument aproximuje jeho textový obsah na hodnotu funkce Q. Textovým obsahem dokumentu se v systému Cora rozumí dvojice 1) text odkazu a 2) text celého dokumentu, ve kterém je odkaz obsažen. Díky funkci F jsme v každém stavu procházení webové struktury schopni zvolit optimální odkaz vedoucí k získání maximálního počtu relevantních dokumentů v minimálním čase.



Srovnání naučeného a nenaučeného robota (převzato z [16])

1.2. Extrakce informací

Ze získaných dokumentů je nejdříve získána úvodní část (předchází tělu dokumentu) a část s referencemi. Pro potřeby dalšího zpracování jsou z nich extrahovány některé specifické informace, například název díla, jméno autora ap. V systému Cora je pro tento úkol použita statistická technika strojového učení nazývaná skryté Markovovy modely (Hidden Markov Models, HMM) [23].

1.2.1. Skryté Markovovy modely

Cílem HMM je vytvořit konečný automat, který pracuje jako klasifikátor při extrakci informací. Učící algoritmus pro naučení vyžaduje specifikaci topologie učících dat a dále rozsáhlou množinu označkovaných dat. Pro dosažení dobrých výsledků je bohužel vhodné tato data označkovat ručně.

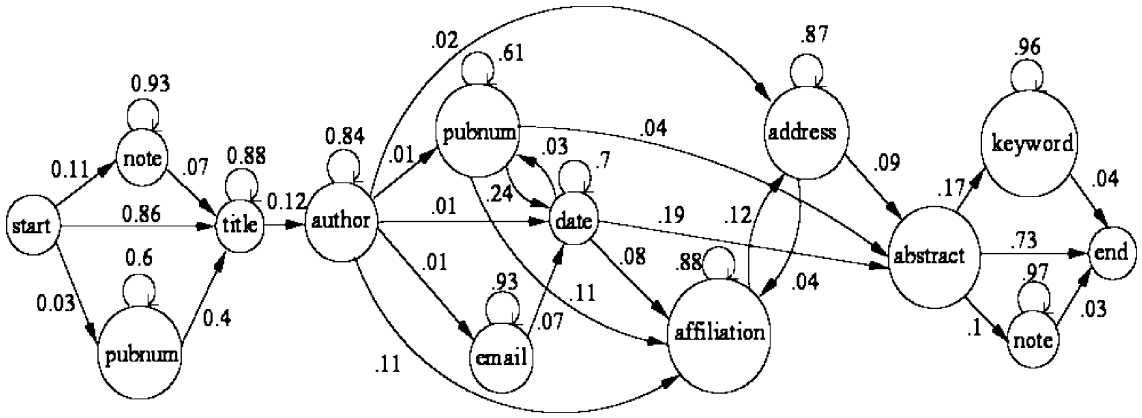
Topologie učících dat je vyjádřena grafem, jehož uzly mají význam stavů konečného automatu a hrany přechodů mezi nimi. Při přechodu ze stavu do stavu je dále emitován tzv. výstupní symbol. Označme Q množinu všech stavů q_i mezi nimiž musí existovat alespoň jeden počáteční a alespoň jeden konečný stav, symbolem $\square$ množinu všech výstupních symbolů w_i a symbolem $\rightarrow$ relaci přechodu mezi stavy. Při práci automatu je generován výstupní řetězec $w_1 w_2 \ldots w_k$. Dále zaveďme pravděpodobnostní funkci $P(q \rightarrow q')$, která vyjadřuje pravděpodobnost přechodu mezi stavy q_i, q_j a pravděpodobnostní funkci $P(q \uparrow s)$ vyjadřující pravděpodobnost generování symbolu s při přechodu ze stavu q.

Cílem učení je nalézt hodnoty pravděpodobnostní funkce $P(w \mid M)$, která udává pravděpodobnost generování řetězce w pomocí HMM M:

$$P(w \mid M) = \sum_{q_1,\ldots,q_l \in Q'} \prod_{k=1}^{l+1} P(q_{k+1} \rightarrow q_k)\, P(q_k \uparrow w_k)$$

a nalezení ohodnocení řetězce w v HMM M:

$$V(w \mid M) = \arg\max_{q_1,\ldots,q_l \in Q'} \prod_{k=1}^{l+1} P(q_{k+1} \rightarrow q_k)\, P(q_k \uparrow w_k)$$



Ukázka naučeného HMM (převzato z [16])

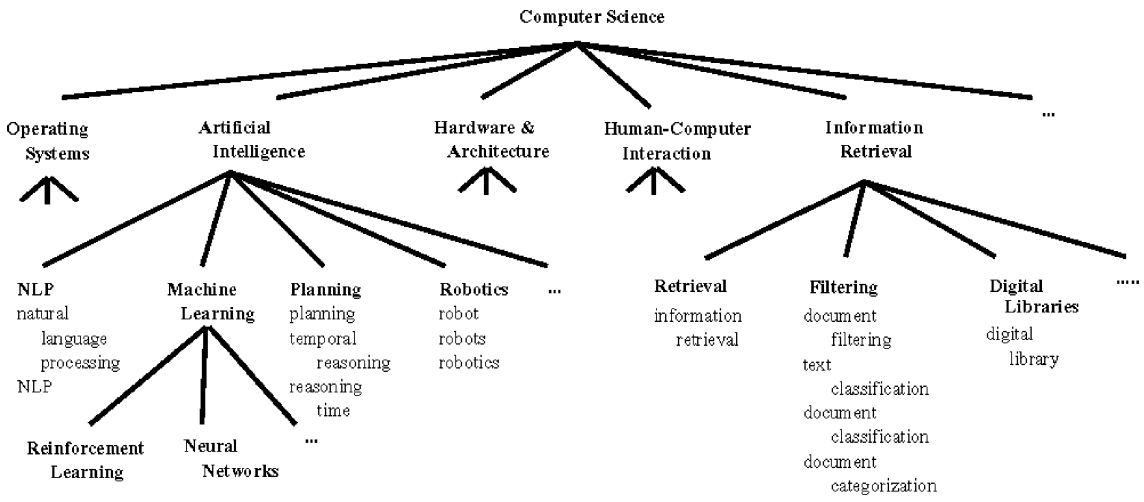
1.3. Klasifikace dokumentů

Systém Cora má ručně vypracovanou tematickou hierarchii pro oblast vědeckých článků. Tato stromová struktura obsahuje 70 listů a maximálně 3 úrovně v jednotlivých větvích. Dokumenty jsou do struktury zařazovány automaticky pomocí techniky strojového učení zvané *bootstraping*. Využívají se přitom informace získané při extrakci. Prostřednictvím citací se hledají vazby mezi dokumenty, podobně jako např. při výpočtu PageRanku v systému Google. Proti univerzálně pracujícímu systému Google lze využít i znalost struktury citací.

1.3.1. Bootstrapping

Vstupem pro algoritmus *bootstrapping* je ručně vypracovaná tematická hierarchie (stromová struktura, topik). Listy zastující jednotlivé třídy dokumentů mají asociovanou informaci o několika klíčových slovech pro třídu typických.

V první fázi jsou technikou naivního Bayesovského klasifikátoru (zjednodušeně podle četností klíčových slov) dokumenty rozděleny do tříd. Jádro *bootstrappingu* tvoří opakované provádění tzv. E a M kroků. Krok E zpracuje informace o názvech dokumentů (extrakce informací z dokumentů viz předchozí oddíl) a sestaví nejpravděpodobnější názvy tříd v topiku. Krok M přehodnotí příslušnost dokumentů k jednotlivým třídám podle analýzy slov v dokumentech obsažených. Pokud například 99 % dokumentů ve třídě X obsahuje slovo parsing, pak všechny dokumenty, které toto slovo neobsahují, jsou ze třídy vyloučeny. Po konečném počtu E a M kroků konverguje počet změn k 0 a topik je sestaven.



Ukázka části topiku systému Cora (převzato z [16])

1.4. Zhodnocení systému Cora

Začátkem roku 2000 obsahoval systém asi 50 000 článků. Bylo zjištěno, že robot používající reinforcement learning je asi 3× úspěšnější při hledání specificky zaměřených dokumentů než „neučící se“ robot (resp. najde informace následováním 3× menšího množství odkazů, tj. 3× rychleji). Přesnost extrakce informací z dokumentů překračuje 90 %. Rovněž efektivita začleňování dokumentů do topiků se blíží efektivitě lidského zpracování. Systém Cora a podobné portály vybudované pomocí technik Just Research mohou výrazně pomoci při automatickém budování specificky zaměřených portálů.

Název systému Cora je utvořen z prvních dvou písmen sousloví Computer Science, zbylá dvě písmena Ra jsou jménem egyptského boha slunce.

2. ResearchIndex

ResearchIndex [15] (formálně CiteSeer, `http://citeseer.nj.nec.com/cs`) je autonomně pracující systém sloužící k vyhledávání vědecké literatury. Vyvíjejí ho Steve Lawrence, Kurt Bollacker a C. Lee Giles v NEC Research Institute. Systém je pro nekomerční využití volně dostupný ve formě zdrojového kódu. Na adrese `majordomo@research.nj.nec.com` existují také diskusní skupiny pro vývojáře a uživatele ResearchIndexu.

2.1. ResearchIndex na FI

Využil jsem nabídky autorů a požádal je o zaslání kopie ResearchIndex. Ta mi opravdu byla spolu s dalšími informacemi obratem zaslána (jednalo se o aktuální verzi 0.2.1). Součástí byl i podrobný návod obsahující seznam vyžadovaných balíků a postup instalace a zprovoznění systému.

Software jsem nainstaloval do svého domovského adresáře na stroji `aurora`. Díky administrátorům strojů `aurora` a `charon` (oba patří NLPlab FI) byly nainstalovány i potřebné balíky. Po úpravě konfiguračních souborů již bylo možné se systémem experimentovat. Později se bohužel ukázalo, že vše neběží „na 100 %“, díky dalším lidem z NLPlab se však podařilo zprovoznit téměř všechny části systému.

2.2. Algoritmy ResearchIndexu

Kód ResearchIndexu je rozdělen na dvě části. Programy pro fulltextovou indexaci jsou napsány v jazyce C++, zbytek systému tvoří skripty v Perlu.

Hlavním nástrojem Research Indexu je autonomní vyhledávání a indexace citací (Autonomous Citation Indexing, ACI). ACI nejdříve na Internetu nalezne a stáhne dokumenty (ps) týkající se zadaného tématu. Poté se pokouší v získaných dokumentech najít citace. Pomocí heuristik jsou rozeznány různě zapsané citace téhož díla. Díky tomu systém nevrací v odpovědi duplicitní informace. Kromě citací je indexován i jejich kontext. Ze získaných informací se odvozují další analýzy.

2.2.1. Vyhledávání dokumentů

ACI systém vyhledává články prohledáváním Webu, monitorováním diskusních skupin a hledáním přímo u autorů. Dále je možné systému poskytnout soubory „ručně“. Pro prohledávání Webu používá služeb vyhledávacích systémů jako jsou Altavista nebo Excite. Pomocí heuristik jsou vybrána vhodná místa (např. se hledají slova publication, paper, ...). V současné době se hledají pouze postscriptové a pdf dokumenty, které se pro další zpracování konvertují do textu.

2.2.2. Parsování dokumentů

V získaných dokumentech se ACI pokouší nalézt oblast referencí. Pokud je sekce nalezena, je dokument považován za vědecký článek a je dále zpracováván. Pomocí heuristik ACI jsou jednotlivé citace z dokumentu extrahovány a dále analyzovány. Hledají se pole název, autor, rok vydání, počet stran a identifikátor citace.

Pomocí metody *invariants first* se systém pokouší citace převést do uniformní podoby. Díky tomu lze např. sledovat a analyzovat tytéž citace i v ostatních článcích. Metoda k určení polí citací používá:

- databáze jmen autorů, názvů článků ap.
- znalosti, např. předcházení jména autora názvu článku
- proměřování vzdáleností mezi řetězci

- zjišťování údajů TF/TFIDF (Term Frequency vyjadřující četnost slova; Term Frequency – Inverse Document Term Frequency vyjadřující zde poměr četnosti slova v dané citaci ku četnosti slova ve všech citacích).

Algorimy ACI pro normalizaci citací jsou sice prakticky velmi dobře použitelné, autoři však uvažují o jejich zlepšení použitím metod strojového učení

2.2.3. Zpracování dotazů

Ze získaných informací systém odpovídá na dotazy specifikované pomocí klíčových slov. Systém může vrátit seznam citací odpovídajících dotazu nebo seznam dokumentů, které klíčová slova obsahují. Citace v dokumentech mohou být dále procházeny.

U citací ResearchIndex zobrazuje i další informace, např. počet indexovaných článků, v nichž je citace obsažena, nebo kontext citace v článku obsažené. Dále je zobrazen graf vyjadřující počet citací v článcích v daném roce. Díky ACI jsou v odpovědi limitovány duplicity citací.

2.3. Práce s ResearchIndexem

Prvním krokem při práci se systémem je vytvoření nové databáze. Za tímto účelem je třeba v souboru `$ROOTDIR/databases` (`$ROOTDIR` je kořenový adresář ResearIndexu) vytvořit nový řádek se jménem databáze. K vlastnímu vytvoření databáze slouží skript `createdb`, kterému se předává povinný parametr číslo databáze.

Po vytvoření databáze je možné pomocí skriptu `start-citeseer` ResearchIndex spustit. Nastartuje se několik démonů, kteří se starají o vyřizování dotazů, zamykání souborů (software lze provozovat i na více strojích zároveň), přístup k databázím ap. Se systémem jsou dodávány i skripty pro restartování démonů v případě „zatuhnutí“.

Pokud je systém v provozu, je možné se na Internetu pokusit najít dokumenty, které nás zajímají. K tomu slouží skript `findps`, kterému se předávají parametry dotazu jako např. klíčová slova (lze použít i logické spojky), maximální počet dokumentů ap. Skript se pokusí najít url dokumentů (ps), které jsou relevantní našemu dotazu. V dalším kroku je možné tyto dokumenty pomocí příkazu `getps` stáhnout. Skriptu `getps` je třeba zadat název databáze a případně další parametry specifikující stroje, které se o stažení postarají. Po stažení jsou soubory automaticky konvertovány do holého textu (zde byly s kopií provozovanou na FI problémy).

Posledním krokem je zaindexování nyní již textových dokumentů do databáze ResearchIndexu. K tomu slouží příkaz `adddocd` s parametry určujícími cílovou databázi. Indexace může běžet i paralelně na více strojích. K informacím získaným indexací lze následně přistupovat přes WWW rozhraní systému nebo z příkazové řádky pomocí skriptu `query` (například `query 'dbnum=1, citation:neural'`).

2.4. Zhodnocení ResearchIndexu

ResearchIndex pracuje autonomně. Lze ho využít ke sledování vazeb mezi publikacemi, vývoji zvoleného tématu v čase. Systém zároveň vyřazuje duplicitní informace a tím šetří uživatelův čas. Na druhou stranu systém nemá nástroje pro nalezení článků konkrétního autora (viz oddíl 2.3., příkaz `findps`). Na rozdíl od podobného systému Cora také nejsou automaticky budovány tematické hierarchie, které by mohly usnadnit přístup k informacím zvoleného tématu.

3. Ahoy!

Ahoy! [1] je volně šiřitelný vyhledávač domovských stránek. Během psaní této diplomové práce byl jistou dobu dostupný na adrese `http://ahoy.cs.washington.edu:6060/`. Jeho výsledky byly natolik dobré, že jsem plánoval jeho integraci do mého vyhledávacího systému. Bohužel byla koncem roku 2000 jeho činnost z důvodu ukončení studia jeho autorů ukončena.

Ahoy! patří do kategorie metavyhledávačů a je primárně vázán zejména na vyhledávač Metacrawler. Jím poskytnuté výsledky filtruje pomocí techniky nazvané Dynamic Reference Sifting (DRS, dynamické prosévání odkazů). V případě, že Metacrawler (ani jiná využívaná primární služba) nevrátí dostatečně relevantní odpověď, Ahoy! se pokouší na základě vyplněného formuláře spekulativně odvodit URL domovské stránky požadované osoby. Na relevanci přitom mají vliv tato kritéria:

- Stránka je umístěna v instituci, ve které hledaná osoba pracuje.
- Stránka je umístěna v požadované zemi.
- Stránka má charakteristické znaky domovské stránky, tj. např. obsahuje fráze typu „Home page“[1] ap.
- Stránka obsahuje odkazy na danou osobu.

3.1. Dynamic Reference Sifting

3.1.1. Přehled klíčových součástí DRS

DRS ke své práci potřebuje *seznam odkazů* poskytnutých na daný dotaz některým primárním vyhledávacím systémem, např. Altavistou. Získané odkazy třídí pomocí *křížového filtru* na základě ortogonální informace, např. e-mailu. V dalším kroku jsou dokumenty staženy a podrobeny *heuristické filtraci*, která může zvýšit přesnost odpovědi . Heuristika vychází ze znalostí doposud získaných o dané doméně. Poté jsou dokumenty rozděleny do skupin (*buckets*), např. podle země.

Pokud je po provedení výše popsaných kroků relavance odpovědi stále nedostatečná, použije se *generátor URL*, která se pokusí URL domovské stránky spekulativně odvodit. K tomu používá tzv. *URL Patern Extractor* – generátor vzorků URL, učící se z předchozích úspěšných hledání.

3.1.2. Nasazení DRS

DRS je možné nasadit pro jakékoliv hledání na webu, existují však oblasti, ve kterých je jeho použití zvláště efektivní. DRS se hodí pro vyhledávací systémy, které pracují s určitým druhem stránek. Může zpracovat i stránky, které jsou pro indexaci příliš rozsáhlé. Na druhou stranu potřebuje, aby velké množství z nich bylo indexováno některou primární vyhledávací službou. DRS je také efektivní v případě, že výsledkem dotazu je jeden nebo několik málo dokumentů.

3.2. Aplikace DRS v systému Ahoy!

3.2.1. Zdroj odkazů

Zdrojem odkazů je pro systém Ahoy! vyhledávací systém MetaCrawler[2]. Ahoy! se MetaCrawler zeptá na jméno osoby, jejíž domovskou stránku hledáme. Ten vrátí seznam potencionálně vyhovujících URL. Dále je odeslán dotaz na dvě e-mailové adresářové služby – WhoWhere? a IAF. Výsledky tohoto dotazu jsou použity ve fázi křížové filtrace podle e-mailu.

1 Vyhledávač Ahoy! je zaměřen zejména na stránky v anglickém jazyce.
2 MetaCrawler je metavyhledávač dotazující se více primárních vyhledávacích služeb s následným vyřazením duplicit.

3.2.2. Křížový filtr

Křížový filtr pro instituci

Pokud uživatel specifikuje ve vstupním formuláři instituci, Ahoy! se s využitím lokální databáze pokouší odvodit její URL. S využitím informace o instituci mohou být zavrženy některé irelevantní odkazy. Například lze jednoznačně rozhodnout, zda půjde o domovské stránky uživatele Erik Anderson z univerzity ve Stanfordu nebo domovské stránky Erika Andersona z univerzity ve Washigntonu. Pokud je informace o instituci neúplná, pokusí se systém pomocí lokální databáze URL instituce odvodit. Po aplikaci křížového filtru už je relevance odpovědi přibližně 74%.

Křížový filtr pro e-mail:

Výsledek hledání v e-mailových adresářových službách je použit pro odvození potencionálního přihlašovacího jména osoby (login). Znalost této informace je potřeba v případě, kdy se reálné jméno osoby velmi odlišuje od jejího přihlašovacího jména.

3.2.3. Heuristika

Ahoy! disponuje databází jmen a přezdívek, kterou využívá při heuristice vycházející z typické podoby domovských stránek. Objeví-li se například na stránce text „About Erik Andersoni“ nebo „Erik Anderson Home Page“, půjde pravděpodobně o domovskou stránku Erika Andersona. Heuristika analyzuje titulek stránky, několik prvních řádků, URL stránky a odkazy, které stránka obsahuje. Skoře stránky se zvyšuje, pokud odkazy ukazují na stránky s podobným URL.

3.2.4. Buckets

Systém Ahoy! rozděluje nalezené stránky do 3 skupin. Relevance stránky se zvyšuje, pokud náleží do většího počtu z nich. Následující seznam popisuje jejich stručnou charakteristiku:

1. stránky, které mají znaky domovské stránky
2. stránky, jejichž URL je v souladu se specifikovanou institucí a zemí
3. stránky, u nichž část URL specifikující jméno, je v souladu se zadaným jménem

3.2.5. Extrakce a generování URL

Extrakce URL

Ahoy! v případě, že byl dotaz úspěšně zodpovězen pomocí kroků 1. – 4. DRS, uloží znalosti o daném URL do své lokální databáze pro budoucí použítí (např. pro spekulativní generování URL). Konkrétně se ukládá trojice jméno osoby, název instituce a URL. Takto má Ahoj! k dispozici URL dané organizace, dále cestu k domovským stránkám na tomto serveru a z URL může extrahovat vzorky pro případné pozdější spekulace při generování URL.

Spekulativní generování URL

Pro každé úspěšné hledání ukládá systém Ahoy! získané znalosti do své lokální databáze. Ta obsahuje informace o jménu osoby, instituci a URL domovské stránky. Poslední jmenovaná položka se použije pro spekulativní generování URL prováděné jen v případě, že předchozí kroky DRS selžou. Postup generování URL vysvětlíme na příkladu.

Předpokládejme, že domovská stránka Roba Jacoba z organizace Tufts nebyla aplikací komponent 1. – 4. DRS nalezena.[1] Ahoy! proto přistoupí ke generování URL. V předchozích úspěšných hledáních byly uloženy URL organizací obsahujících v názvu řetězec Tufts – `www.cs.tufts.edu` a `emerald.tufts.edu`. Rovněž bylo zjištěno, že cesta k domovským stránkám je v prvním případě `~[username]` a ve druhém `people/[username]`. V případě, že dotazy na e-mailové adresáře (krok 1) byly úspěšné, lze rovněž usoudit na login uživatele (`username`). Z těchto

1 Příklad je převzat z [1].

znalostí vygeneruje Ahoy! několik URL a po analýze cílů se pokusí odvodit URL domovské stránky hledaného uživatele.

`www.cs.tufts.edu/~rjacob/`	`emerald.tufts.edu/people/rjacob/`
`www.cs.tufts.edu/~robjacobs/`	`emerald.tufts.edu/people/robjacobs/`
`www.cs.tufts.edu/~rob/`	`emerald.tufts.edu/people/rob/`
`www.cs.tufts.edu/~rj/`	`emerald.tufts.edu/people/rj/`
`www.cs.tufts.edu/~robj/`	`emerald.tufts.edu/people/robj/`
`www.cs.tufts.edu/~jacob/`	`emerald.tufts.edu/people/jacob/`

3.3. Zhodnocení výsledků služby Ahoy!

3.3.1. Srovnání s jinými vyhledávacími službami

Během provozu systému Ahoy! jsem zkoušel položit dotazy na URL domovských stránek lidí z naší fakulty. Jeho úspěšnost byla téměř 100%. Následující statistiky jsou převzaty ze studie [1], ze které čerpám i při popisu architektury DRS.

Pro porovnání systému Ahoy! s konkurencí byl vybrán vzorek několika set URL domovských stránek, z nichž byly dvěma nezávislými cestami vygenerovány 2 sady vzorků pro naplnění vstupního formuláře Ahoy!. Z těchto údajů byly vygenerovány dotazy na konkurenční vyhledávací služby, přičemž pro každou službu byl dotaz vytvořen s ohledem na využití všech dostupných možností dané služby.

Systém v první fázi používá reference poskytnuté primárními vyhledávacími službami, jako je například Altavista (zprostředkovaně přes MetaCrawler). Altavista nalezne průměrně 58 % cílů a 23 % z nich umístí ve výstupu na první místa. Ahoy! díky DRS v 74 % případů nalezne správné cíle a označí je jako nejpravděpodobnější. V 9 % jsou cíle určeny pomocí spekulativní metody odvození URL. Systém Ahoy! poskytuje odpověď průměrně za 9 s od zadání dotazu a nejčastěji nabídne 2 URL potenciálních domovských stránek hledaného uživatele.

4. Collection of Computer Science Bibliographies

Collection of Computer Science Bibliographies (`liinwww.ira.uka.de`) je ručně udržovaná kolekce bibliografií vědeckých titulů z oblasti informačních technologií. Většinu obsahu tvoří bibliografie článků z časopisů, materiálů z konferencí a technických zpráv. Záznamy jsou vedeny ve formátu BibTex. 660 MB databáze v současné době obsahuje více než 1,1 miliónu záznamů, z nichž přes 100 000 odkazuje na online přístupné materiály. Měsíčně je podle údajů vydavatele aktualizováno přibližně 1400 záznamů. U více než 16 000 záznamů je uvedena informace o citacích z jiných materiálů i citacích jiných materiálů. Součástí portálu je i seznam více než 2 000 odkazů na další podobně zaměřené služby (např. katalogy IEEE dostupné z `ieeexplore.ieee.org` nebo digitální knihovna ACM dostupná na url `www.acm.org`).

Pokud má být do této kolekce zařazen nový titul, je třeba o něm informovat provozovatele databáze. Kromě toho mají autoři už vybudovánu síť trvalých zdrojů informací, ze které databázi pravidelně aktualizují. Kolekce neobsahuje autonomně pracující systém, který by zajišťoval automatický přísun nových záznamů.

Pro zadávání dotazů obsahuje portál jednoduché i rozšířené rozhraní. V obou lze v dotazu použít operátory and, or, sdružování do závorek, lze nastavit citlivost na velká písmena a také to, zda mají být zobrazeny jen ty dokumenty, které jsou dosažitelné online.

5. DL New Zealand

New Zealand Digital Library (`www.nzdl.org`) je online dostupnou digitální knihovnou provozovanou na University of Waikato. Jádrem knihovny je nástroj Greenstone, jehož prostřednictvím knihovna nabízí mnoho různých sbírek v mnoha jazycích.

Greenstone je software budovaný na univerzitě ve spolupráci s dalšími organizacemi (UNESCO, Humanities Library Project). Umožňuje vytvářet a prostřednictvím CD-ROM nebo Internetu distribuovat multimediální kolekce. Jedná se o Open Source software, který podléhá GNU General Public Licence. Většina kódu je napsána v C++, část v Perlu. Jako uživatelské rozhraní je použit WWW prohlížeč. Pro svůj běh potřebuje Apache Webserver.

Greenstone je navržen tak, aby byl snadno rozšiřitelný. Prostřednictvím tzv. pluginů (existují např. moduly pro pdf, ps, html, ...) lze do kolekcí přidávat dokumenty nových formátů, pomocí tzv. klasifikátorů lze přidávat podporu pro nové metadatové struktury. Kromě přístupu k digitální knihovně NZDL lze pomocí Greenstone vytvářet vlastní kolekce.

NZDL je typickým příkladem digitální knihovny. Umožňuje efektivně vytvářet rozsáhlé kolekce multimediálních elektronických sbírek. Má uživatelsky velmi vyspělé rozhraní, takže lze předpokládat, že si získá oblibu mezi širokou veřejností. Jak je však u systémů digitálních knihoven zvykem, nejedná se o autonomní systém, nové dokumenty je tedy nutné systému explicitně předložit.

Elektronické informační zdroje na MU

Podle zprávy Miroslava Bartoška z Knihovnicko-informačního centra MU má v současné době univerzita předplacené následující informační zdroje:

1. Web of Science (Institute for Scientific Information)

obsah	retrospektiva	aktualizace	přírůstek při aktualizaci
Články z více než 8.500 vědeckých časopisů ze všech oborů; bibliografie, reference, citace.	20 let	týdně	25.000 záznamů; 400.000 citačních odkazů

2. ProQuest 500 (Bell+Howell)

obsah	retrospektiva	aktualizace	přírůstek při aktualizaci
Plné texty více než 5.000 periodik; záznamy z dalších 3.000 časopisů; humanitní a spol. obory, obchod, medicína, přírodní vědy, výpočetní a telekomunikační technika.	od r. 1993	max. 48 h po vydání	

3. PCI – Periodicals Contents Index (Bell+Howell)

obsah	retrospektiva	aktualizace	přírůstek při aktualizaci
Obsahy a bibliografie článků z 3.388 časopisů z oblasti humanitních a sociálních věd (11 miliónů článků).	1770 – 1990	ročně?	1.000.000 záznamů

4. EIFL Direct (EBSCO Publishing)

obsah	retrospektiva	aktualizace	přírůstek při aktualizaci
Metadatabáze plných textů článků z 3.300 časopisů, novin a zpravodajství; především sociální a humanitní vědy. Navíc mediccínské databáze. V projektu je sdruženo 39 zemí, které tak tvoří největší světové informační konsorcium.	od r. 1990		

5. JSTOR – Journal Storage (konsorcium JSTOR)

obsah	retrospektiva	aktualizace	přírůstek při aktualizaci
Plné texty z více než 117 amerických vědeckých časopisů (antropologie, ekologie, ekonomika, filosofie, finance, historie, literatura, matematika, politické vědy, sociologie, statistika, vzdělávání).	od minulého století do současnosti		

6. Biological Abstracts, Zoological Records (BIOSIS)

obsah	retrospektiva	aktualizace	přírůstek při aktualizaci
Biological Abstracts – reference na články z 6.000 časopisů z oblasti life scicences (včetně bio..., botaniky, ekologie, neurologie, farmakologie, zdravotnictví, a zemědělství). Zological Records – 4.500 periodik ze 100 zemí světa.	Biological Abstracts od 1997; Zological Records od 1978		

7. Springer – LINK (Springer – Verlag)

obsah	retrospektiva	aktualizace	přírůstek při aktualizaci
400 vědeckých periodik vydávaných nakl. Springer–Verlag.			

8. Springer – LNCS (Springer – Verlag)

obsah	retrospektiva	aktualizace	přírůstek při aktualizaci
LCNS – Cecture Notes in Computer Science; DB elektronických verzí sborníků z konferencí z oblasti computer science a umělé inteligence.		ročně?	200 publ.

9. Digitální knihovna ACM (ACM)

obsah	retrospektiva	aktualizace	přírůstek při aktualizaci
30 časopisů ACM z oblasti computer science; plné texty sborníků konferencí pořádaných ACM od r. 1985 (stovky sborníků z > 130 sérií konferencí).	od r. 1985		200 publ.

10. ETRDL (ERCIM)

obsah	retrospektiva	aktualizace	přírůstek při aktualizaci
Technické zprávy z oblasti computer science a matematiky v rámci evropského konsorcia ERCIM; napojeno na obdobný americký celosvětový systém NCSTRL.			

11. KnowEurope

obsah	retrospektiva	aktualizace	přírůstek při aktualizaci
Rozsáhlý zdroj informací o EU – instituce, legislativa, lidé, politika, fakta, čísla, analýzy, adresář.			

Předplacení uvedených zdrojů bylo možné zejména díky finanční podpoře programu MŠMT LI – Informační zdroje pro vědu a výzkum. V letech 2000-2003 bude z tohoto programu na nákup informačních zdrojů formou grantů vynaloženo téměř půl miliardy korun. Řada projektů má konsorciální charakter a univerzita zapojením do těchto projektů získala cenné informační zdroje jiným způsobem prakticky nedostupné (předplatné každého jednoho takového zdroje by pro MU představovala náklady v řádu mnoha miliónů Kč ročně).

Přístup k uvedeným službám sjednotilo Knihovnicko-informační centrum MU na portálu `http://www.muni.cz/library/sluzby.html`. Uvedené zdroje jsou dostupné prostřednictvím sítě z kteréhokoliv počítače MU v doméně `muni.cz` pomocí WWW prohlížeče.

Uvedené informační zdroje nejsou jediné, k nimž má v současnosti MU přístup, další jsou však již dostupné jen určitým omezujícím způsobem: buď jen pro odborníky daného oboru (např. chemie, filosofie) a/nebo pomocí speciálních klientských programů.

Integrace IS na WWW

1. Architektura Stanford InfoBus

1.1. Důvody vzniku architektury InfoBus

S růstem Internetu vzniká mnoho problémů, z nichž patrně ten největší je organizace dat na Internetu se nacházejících. Důležitost tohoto problému lze vystihnout větou: „To, co nelze najít, jako by neexistovalo.“ Problém organizace úzce souvisí s problémem integrace a lze ho chápat i jako snahu o to, aby se uživatel přes uniformní rozhraní dostal k libovolné informaci na Internetu uložené.

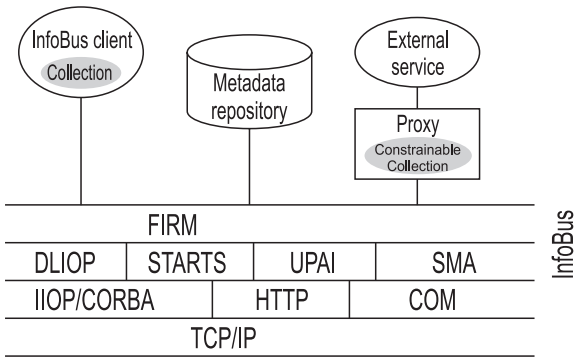
K integraci informačních zdrojů Internetu lze přistupovat mnoha způsoby. Některé z nich nepožadují po uživatelích ani po poskytovatelích informací (služeb) na Internetu mnoho – totiž jen to, aby zde své informace vystavili. Touto filosofií se řídí vyhledávací systémy, které se snaží automaticky prohledávat a indexovat informace dostupné na Internetu. Jejich kvalita se neustále zlepšuje (používají se techniky umělé inteligence a strojového učení), avšak Internet roste příliš rychle a spektrum informací je velmi různorodé.

Další integrační nástroje požadují při publikování nových informací dodržování jistých standardů. (např. standard Dublin Core). Dodržováním těchto standardů by se situace na Internetu velmi zjednodušila, ale běžní uživatelé se jakékoli „nadbytečné“ práci brání.

S poněkud jinou myšlenkou pro integraci informačních zdrojů dostupných na Internetu přichází projekt Stanford Infobus [11]. (Jedná se o jeden z 6 projektů, které vyhrály konkurz několika komerčních společností v USA v roce 1994, zaměřených právě na integraci Internetu. Projekty byly dotovány částkou asi 24 mil. dolarů.) Tento projekt se v podstatě snaží přenést problémy integrace na poskytovatele informačních služeb Internetu tím, že je vybízí k vybudování rozhraní mezi jimi poskytovanými službami a novým standardem InfoBus. InfoBus je architektura, která rozšiřuje stávající protokoly Internetu o novou množinu protokolů (a s tím související služby), které umožní klientům uniformní přístup k heterogenním informacím a informačním službám Internetu.

1.2. Rozbor architektury InfoBus

Jedním ze základních kamenů architektury InfoBus je tzv. „InfoBus Proxy“. Jedná se o oddělovač nativního prostředí konkrétní informační služby od standardního prostředí InfoBus. Realizace Proxy je na provozovateli informační služby. Proxy musí být schopna předávat informace ze služby pomocí standardních protokolů InfoBusu, se službou potom komunikuje pomocí jejího nativního protokolu (telnet, Z39.50, HTTP, ...). Protokolů Infobusu bylo vyvinuto celkem 5 (5 služebních vrstev InfoBusu). Nyní se budu věnovat jednotlivým služebním vrstvám InfoBusu, jejichž stručný náhled lze nalézt na uvedeném obrázku.



Architektura Stanford InfoBus

1.2.1. DLIOP

Digital Library InterOperability Protocol je vrstva InfoBusu, která poskytuje základní prostředky pro práci s položkami (Item, obecný objekt InfoBusu, např. dokument) a jejich kolekcemi (Collection) v síťovém prostředí. Umožňuje klientům prostřednictvím Proxy komunikovat s informačními zdroji. Proxy typicky na žádost klienta naplní tzv. Constrainable Collection a klient si pomocí několika na proxy vhodně směřovaných dotazů (viz vrstva START) doplní svoji vlastní Collection „na požadované množství informací".

1.2.2. SMA

Stanford digital library Metadata Architecture (SMA) definuje metadatové struktury pro popis služeb InfoBusu (služba vrátí popis sebe sama) a metadata popisující přenášené informace. SMA je specifikována pomocí 4 základních tříd: atributový model pro Proxy, atributový model pro „překladače" (Translators), metadatové struktury pro vyhledávání přes Proxy a metadata „repositářů" (Repository, skladiště). Každý atribut (položka jedné ze 4 uvedených tříd) je popsán trojicí název modelu (Model Name), jméno atributu (Attribute Name) a typ hodnoty (Value Type). Typy jednotlivých položek trojice jsou popsány pomocí objektové třídy AttributeItem, ta využívá IDL (Interface Definition Language, část CORBA). Takto je každý atribut určen jednoznačně.

Atributový model pro Proxy je implementován pomocí InfoBus Collection a může jít např. o model „Dublin Core". Překladače jsou třídy SMA, které slouží k převodu specifikací mezi jednotlivými modely. Metadatové struktury pro vyhledávání přes Proxy umožňují zjistit vlastnosti (schopnosti) o informační službě, pro kterou Proxy pracuje jako rozhraní. Konečně metadata pro repositáře (Repository) umožňují ukládat do těchto skladišť konkrétní výsledky hledání z různých Proxy.

1.2.3. STARTS

STAnford protocol proposal fot internet ReTrieval and Search je vrstva InfoBusu sloužící k organizaci vyhledávání. Hlavními úkoly STARTS je najít nejlepší zdroje (Proxy) pro vyhodnocení dotazu klienta, dále vyhodnotit odpovědi z těchto zdrojů a v odpovědi je uspořádat. STARTS definuje jednoduchý dotazovací jazyk, který musí informační zdroje (Proxy) podporovat. Je založen na standartu Z39.50. Obsahuje booleovskou logiku, je možno specifikovat různé modifikátory. Kromě toho musí být zdroj schopen poskytovat různé statistiky, např. koeficient kvality odpovědi (rank).

1.2.4. UPAI

Universal Payment Aplication Interface je vrstva řešící mechanismus placení za poskytování informací. Podobně jako u Proxy je filosofie této vrstvy založena na zachování původních nativních protokolů (např. DigiCash) a definuje rozhraní pro standart InfoBus. Při transakci si klient i služba (přes U-PAI rozhraní) otevřou monitory transakce. Důležitými složkami komunikace je přenos PCR (Payment Control Record) obsahující položky odkud, kam a jaká částka je zasílána a dále přenosy různých stavových informací (např. Complette, InProgress, Failed).

1.2.5. FIRM

Stanford Framework for Interoperable Rights Management slouží k definici toho, „kdo co může". Návrh opět zachovává původní fungující modely (např. UNIXová přístupová práva) a definuje rozhraní pro InfoBus. Pro ověřování práv používá techniku posílání lístků (Tickets), kromě toho ještě existuje technika trvalého spojení klienta a služby při autentikaci. FIRM zavádí několik pojmů důležitých pro principy ověřování autentikace, z nichž nejdůležitější je pojem Compact (také smart contract, jednoduchá smlouva). Compact je definován rozhraním mezi oběma stranami a to včetně předávání událostí a stavů. Compact je při komunikaci mezi klientem a službou udržován správcem smlouvy (Compact manager).

1.3. Zhodnocení architektury InfoBus

InfoBus je architektura, která se pomocí 5 protokolů rozšiřujících stávající protokoly Internetu snaží o jeho integraci. Jedná se o zcela nový přístup, který sice přináší nutnost dodržování jistých standardů, ale ty neklade na uživatele Internetu nýbrž na poskytovatele již existujících služeb, přitom návrh InfoBusu zachovává jejich funkčnost a implementaci. Myslím si, že je to jedna z těch lepších cest, jak zvládnout informační explozi, kterou pozorujeme na současném Internetu.

Zadání

Implementujte systém pro vyhledávání osob pracujících v určité oblasti výzkumu, shromažďování jejich publikací dostupných na WWW a vyhledávání v takto vzniklé databázi.

PRAKTICKÁ ČÁST

Realizace vyhledávacího systému

1. Použité nástroje

1.1. OS Linux

Vývoj celého systému byl realizován v prostředí OS Redhat Linux, v počátcích ve verzi 6.0 s jádrem 2.2, postupně až ve verzi 7.0 s jádrem 2.4. Systém jsem rovněž bez problémů odzkoušel na distribuci Debian, která je v současné době instalována na počítačích NLPlab FI MU Brno.

1.2. Kompilátor C++, BASH a podpůrné nástroje

Převážná většina kódu je napsána v C++ a přeložena pomocí volně dostupného GNU kompilátoru C++ známého pod jménem G++. Často je využívána standardní šablonová knihovna pro C++ (Standard Template Library, STL) a samozřejmě knihovny dodávané s OS Linux. Některé úlohy jsou řešeny v podobě skriptů unixového shellu BASH, přičemž jsou často použity nástroje standardně dodávané s OS Linux, jako např. grep, wc a další. Použitý kompilátor se velmi blíží standardu ISO pro C++, proto by měl být kód dobře přenostitelný na jiné UN*Xové platformy.

1.3. XML nástroje

Jedním ze základních kamenů systému je použití nástrojů technologie XML, která je popsána v teoretické části. Bibliografické informace o dokumentech dostupných na WWW jsou ukládány do XML dokumentů – jeden dokument získaný na WWW je popsán právě jedním XML dokumentem. Pro generování a parsování XML dokumentů jsem použil balík Xerces-C, což je kolekce volně šiřitelných XML parserů pro jazyk C++ od IBM. Obsahuje parser Xerces-C DOM realizující DOM parser první úrovně a parser Xerces-C SAX realizující SAX parser specifikace SAX 1.0. Xerces-C pracuje s mezinárodním kódováním UNICODE.

Pro zobrazení XML dokumentů vrácených vyhledávacím systémem jsem použil volně šiřitelný nástroj LotusXSL od IBM, pomocí něhož jsou dokumenty transformovány do formátu HTML.[1] LotusXSL je naprogramován jako třída v jazyce Java, pro jeho běh je tedy nutné mít nainstalován alespoň tzv. runtime jazyka Java. Výše uvedené nástroje společnosti IBM lze získat na adrese `http://www.alphaworks.ibm.com/`, prostředí pro spouštění i vývoj programů v jazyce Java lze získat na adrese `http://www.sun.com/`.

1.4. Relační databáze

Další velmi důležitou částí systému je spolupráce s relační databází Postgres. Postgres je volně dostupnou implementací relační databáze pro Linux. XML dokumenty získané v prostoru WWW samozřejmě nelze prohledávat sekvenčně, a proto jsou zaindexovány do tabulek spravovaných v databázi Postgres. Pro přístup k postgresové databázi jsem použil knihovnu `libpq`, která realizuje API Postgresu pro jazyk C.

1.5. HTML, Javascript, CGI

Jednotlivé komponenty systému lze použít buď přímým voláním příslušných programů z příkazové řádky, pro většinu z nich však existuje WWW rozhraní. Vizuální stránka rozhraní je realizována technologií `HTML`. Pro odlehčení serveru jsou vstupy z formulářů před odesláním

1 Původně jsem uvažoval o použítí webového prohlížeče podporujícího XML. V současné době však podporuje XML jen prohlížeč Mozilla a poslední verze prohlížečů Netscape a MS Explorer a to v podobě, jak se ukázalo, pro mé potřeby nedostatečné.

2 Knihovna `libpq` je zároveň základem implementací rozhraní pro další programovací jazyky.

verifikovány skripty Javascriptu. CGI rozhraní pak slouží k volání na serveru uložených programů tvořících funkcionalitu systému.

1.6. Další použité nástroje

Pro některé další dílčí úlohy byly v systému použity tyto nástroje:

- programy pstotext a pdftotext pro konverzi ps a pdf dokumentů do textu
- slovník ecd
- hashovaná databáze gdbm
- program wget pro stahování souborů z www (ftp) serverů
- prostředí jdk1.3 společnosti Sun pro spouštění a vývoj programů v jazyce Java

2. Instalace a spuštění systému

V této části se zmíním o instalaci nástrojů potřebných k běhu systému a o instalaci vyhledávacího systému samotného. Nebudu zde zmiňovat instalaci ani konfiguraci OS Linux, kompilátorů a dalších nástrojů (jako např. http serveru) standardně dodávaných se systémem.

2.1. Instalace XERCES-C a LotusXSL

Součástí balíku Xerces-C je bohatá dokumentace, popisující jak API implementovaných parserů, tak popis instalace. Nástroj lze přeložit pod mnoha platformami. V případě OS Linux jsem se setkal s problémem v nastavení parametrů v souboru Makefile.incl. V sekci Linux Specific Options jsem v položce PLATFORM_COMPILE_OPTIONS musel zrušit volbu –instances=static. Po konfiguraci a přeložení je ještě nutné nastavit proměnnou prostředí XERCESCROOT na kořenový adresář Xerces-C a do proměnné LD_LIBRARY_PATH přidat adresář $XERCESCROOT/lib.

Instalace balíku LotusXSL se provádí rozbalením do zvoleného adresáře. V případě problémů s předkompilovanými knihovnami je lze pomocí make znovu přeložit. Runtime prostředí jazyka Java, které LotusXSL potřebuje pro svůj běj, lze získat v podobě rpm balíků (instalace provedením rpm -i soubor.rpm[1]).

2.2. Instalace Postgres a vytvoření databáze

Instalace relační databáze Postgres (7.0) je v prostředí OS Linux snadná, jelikož jsou její nástroje šířeny i v podobě binárních rpm balíků. Po nainstalování[1] je potřeba pomocí utility initdb inicializovat novou databázovou kolekci (např. initdb ~/db) a dále pomocí utility createdb vytvořit databázi (createdb marianek).

2.3. Instalace vyhledávacího systému

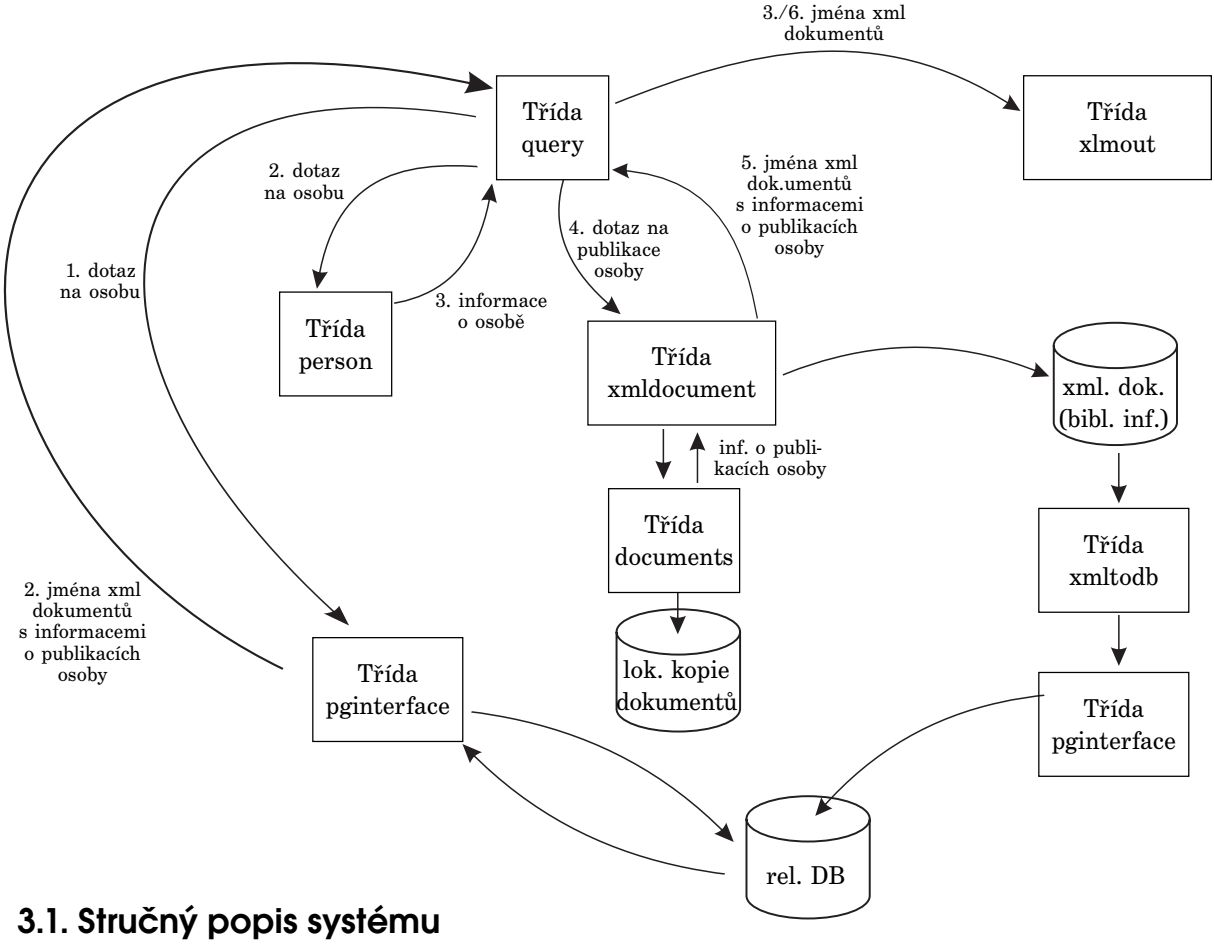
Instalace samotného vyhledávacího systému spočívá v rozbalení archivu (gunzip proj.gz) do kořenového adresáře systému (dále ROOTDIR), překladu[2] (make clean && make && make install) a umožnění zápisu do adresářů $ROOTDIR/tmp a $ROOTDIR/xml pro HTTP server. Pokud HTTP server používá modul suExec, pak stačí správně nastavit práva vlastníka souborů. Pokud tomu tak není, lze použít např. chgrp nobody $ROOTDIR/tmp && chmod g+a $ROOTDIR/tmp[1]. HTTP server musí mít rovněž přístup do kořenového adresáře systému (chmod o+rx $ROOTDIR)[3].

1 Jsou potřeba administrátorská práva.
2 V případě problémů je v komentářích na začátku zdrojových textů popsán postup „ruční“ kompilace.
3 Často je HTTP server nastaven tak, že v domovském adresáři uživatele vidí jen adresář public_html. Pak je systém nutno instalovat do něj nebo požádat administrátora o zpřístupnění i jiného adresáře než public_html.

2.4. Spuštění systému

Po spuštění HTTP serveru[1] (v OS Redhat Linux zpravidla `/etc/rc.d/init.d/httpd start`) a spuštění databázového serveru Postgres (v OS Redhat Linux příkazem `postmaster`) už lze nasměrovat WWW prohlížeč na hlavní stránku vyhledávacího systému, která je uložena v adresáři `$ROOTDIR/index.html`[2].

3. Základní schema systému[3]



3.1. Stručný popis systému

Na začátku práce systému uživatel specifikuje dotaz. Může jít o dotaz na dokumenty vypracované konkrétní osobou (zadáním jejího jména a dalších upřesňujících údajů), nebo dotaz na zvolené URL HTML stránky. Ve druhém případě je daná HTML stránka podrobena analýze, jejímž výsledkem je získání jmen a s nimi spojených dalších údajů, které uživatel může specifikovat v dotazu na konkrétní osobu. Aktivací jednoho takového jména se druhý typ dotazu převede na typ první.

Poté se provede dotaz na relační databázi, zda-li již nebyla specifikovaná osoba někdy systémem hledána. Pokud ano, pak jsou vráceny XML dokumenty obsahující bibliografické údaje nejvíce odpovídající dotazu.

Pokud byl dotaz na relační databázi neúspěšný, jsou provedeny dotazy na různé WWW vyhledávače a jsou zkompletovány informace o osobě (případně více osobách stejného jména), z nichž nejpodstatnější je získání URL domovské stránky (domovských stránek).

1 Jsou potřeba administrátorská práva.
2 Pokud je systém nainstalován v adresáři `public_html` uživatele `marianek`, pak odpovídající url je `http://adresa.serveru/~marianek/$ROOTDIR/`.
3 Nezachycuje všechny fáze vyhledávání, chybí např. zpracování dotazu na URL.

V následujícím kroku se systém pokusí na domovské stránce najít odkazy na dokumenty. Hledání probíhá rekurentně do několika úrovní odkazů. Pokud byly nějaké relevantní odkazy nalezeny, jsou odkazované dokumenty staženy a analyzovány. Relevantní dokumenty jsou uloženy do XML souborů a převedeny na výstup. Získané XML soubory jsou připraveny pro vložení do relační databáze.

Indexace je zpravidla prováděna po stažení dostatečného množství dokumentů. Během ní jsou zároveň vyřazeny duplicitní XML dokumenty (duplicity XML dokumentů popisujících jeden dokument na WWW mohou vzniknout v době, kdy ještě nejsou zaindexovány do relační databáze).

Uživatel může při práci se systémem využít i dalších pomocných nástrojů. Je to nástroj pro měření odezvy serverů na WWW a dále záznamník, kde je možné specifikovat problémy při práci se systémem.

V dalším textu popisuji jednotlivé části systému, a to sice metodou „zdola nahoru“, tzn., že daná část systému by neměla být popsána dříve, než jsou popsány ty součásti systému, které využívá.

4. Implementace pomocných nástrojů

4.1. HTML parser

```
class htmlpars
{   ifstream infile;
    vector <string> src;                  // kontejner pro zdrojovy HTML kod
    vector <string> :: iterator src_it;   // pracovni iterator kont. src
    vector <string> :: iterator src_pos;  // iterator src - ukazatel pozice
    string base_href;
    bool elements_upcased;
    bool href_set_abs;
public:
    htmlpars(char* fname, int& code); // konstruktor, fname je nazev souboru;
    htmlpars();                       // prazdny konstruktor
    ~htmlpars();                      // destruktor
    int load(char* fname);   // nacte obsah souboru fname do kontejneru src
    int init();              // nastavi src_pos_it na 1. polozku
    int step();              // nastavi src_pos_it na nasledujici polozku
    int step_to_elem(string elem);  // jdi na nejblizsi element elem
    int step_to_value(string txt);  // jdi na nejblizsi hodnotu txt elementu
    string get();   // vrati element/hodnotu z aktualni pozice src_pos
    int get_attr(map <string, string> &attr); // vrati atributy akt. elementu
    int remove();            // opdstrani polozku na aktualni pozici src_pos
    int insert(string s);   // vlozi na aktualni pozici src_pos retezec s
    int upcase_elements();  // nazvy elementu zmeni na velka pismena...
    int set_abs_href(string url);  // zmeni vsechny rel. odkazy na absolutni
    friend ostream& operator << (ostream& output, htmlpars& p); // vystup
};
```

V raných stádiích projektu jsem HTML soubory zpracovával bez použití analyzátoru. Zdrojový kód díky tomu rostl a stával se nepřehledným.[1] Po těchto zkušenostech jsem přesně věděl, co potřebuji, a rozhodl jsem se vytvořit vlastní jednoduchý analyzátor HTML kódu. Je implementován jako třída v jazyce C++ a jeho použití je zřejmé z uvedeného výpisu[2]. V adresáři `$ROOTDIR/src/htmlpars` se nachází i příklad `test_htmlpars` demonstrující použití parseru.

4.2. Parser pro QUERY_STRING

Po vyplnění a odeslání dat z formuláře HTML stránky se v případě dotazu typu GET informace na straně serveru zakódují do proměnné prostředí s názvem `QUERY_STRING`. Pro analýzu jejího obsahu jsem implementoval program `quextr` (`$ROOTDIR/src/quextr/quextr.C`):

```
./quextr $QUERY_STRING name
```

Parametr `name` specifikuje položku formuláře, jejíž obsah je vrácen na standardní výstup.

4.3. Program pro měření odezvy WWW serveru

Pokud systém přestane na delší dobu reagovat, může to být způsobeno špatnou průchodností sítě. Tu lze detekovat pomocí programu `resp` (`$ROOTDIR/src/resp/resp.C`):

```
./resp -u url -t time
```

Program se pomocí `wget` pokusí navázat spojení se serverem zadaným parametrem `url`. V případě překročení času specifikovaného v parametru `time` je `wget` ukončen a vypsána zpráva o nedostupnosti serveru v daném časovém intervalu. Pro tento program existuje i WWW rozhraní dostupné z hlavní stránky systému.

4.4. Poznámkový blok

Pokud vyhledávací systém neodpoví na zadaný dotaz dostatečně relevantně, lze v poznámkovém bloku (`$ROOTDIR/src/locdbm/`) specifikovat data dotazu, která mohou být použita pro další vývoj systému.

Poznámkový blok je realizován jako hashovaná databáze a staví na GNU rozhraní `gdbm`. Tato databáze měla být původně nosnou databází celého vyhledávacího systému, nakonec se však ukázala jako nevyhovující[3]. Poznámkový blok lze volat z příkazové řádky (`./locdbm ...`) a existuje pro něj i WWW rozhraní dostupné z hlavní stránky systému.

5. Implementace subsystému pro vyhledávání lidí

5.1. Datová struktura persdata

Datová struktura `persdata` je využívána třídou `person` a deklaruje položky, které chceme získat o osobě, na jejíž články a on-line přístupná díla se dotazujeme. Význam jednotlivých položek je zřejmý z výpisu. Jméno (`name`) je vždy specifikováno v dotazu. Položka `name_by` udává, zda-li bylo jméno zadáno explicitně nebo získáno analýzou HTML dokumentu (dotaz na URL HTML stránky).

Informace o domovské stránce (`hp`, `hp_by`) a emailu (`email`, `email_by`) mohou být specifikovány již v dotazu nebo získány pomocí třídy `person`. Položku `seq`, která obsahuje URL dotazů na danou osobu na další vyhledávací služby, naplňuje vždy třída `person`.

1 Některé zdrojové texty jsem díky tomu dokonce odložil a v konečném systému nejsou využity (programy `conv`, `qpars`, `ldbprotc`, `protc`, `docc`).
2 V případě potíží lze nahlédnout do zdrojového textu, který je bohatě komentován.
3 Na hashovanou databázi např. nelze klást dotaz pomocí tzv. žolíkovách znaků.

```
struct persdata
{   string name;    // jmeno osoby
    int name_by;     // 0: jmeno zadano explicitne, 1: jmeno ziskano analyzou
    string hp;       // url domovske stranky
    int hp_by;       // 0: zadana jako parametr, 1: ziskana sluzbou hpsearch
                      // 2: domovska stranka ziskana vyhl. systemem Google
    string email;   // email osoby
    int email_by;   // 0: zadan jako parametr, 1: email ziskan analyzou hp
    map <string, string> seq;  // Search Engines Queries na danou osobu
};

class person
{   vector <persdata> pers;    // osoby vyhovujicí danemu jmenu
    vector <persdata> :: iterator pers_it;
    persdata qpers;            // dotazovana osoba (naplni konstruktor)
    bool hpsearch_success;
    int check_hp();    // vyradi duplicitni domovske stranky
public:
    person(string name, string hp, string email, int &code); // konstruktor
    ~person();         // destruktor
    int gethp();       // nalezne domovske stranky osob, zavola check_hp()
    int getseq();      // sestavi seznam dotazu na dalsi vyhledavace (seq)
    int collect_email();  // pokusi se doplnit email vlastnika domov. stranky
    int show();        // zobrazi obsah kontejneru pers

    int get(vector <persdata> &p);     // vrati kontejner pers
};
```

5.2. Třída person

Třída `person` slouží pro sestavení kontejneru `pers`, obsahujícího informace o osobě (osobách) s daným jménem. Každá osoba je v kontejneru reprezentována právě jednou datovou strukturou `persdata`.

Konstruktoru je nutné předat povinný parametr `name` obsahující jméno osoby, případně volitelné parametry pro specifikaci domovské stránky (`hp`) a emailu (`email`). Pokud proběhne konstruktor v pořádku, je parametr `code` nastaven na 0 a lze pokračovat v naplňování kontejneru `pers` voláním metody `gethp()`.

Metoda `gethp()` naplní kontejner `pers` tolika strukturami `persdata`, kolik URL domovských stránek osob s daným jménem se jí podaří získat. Domovské stránky mohou být získány buď z explicitního zadání v dotazu (`hp_by=0`), z odpovědi vyhledávacího systému `HPSearch`[1] (`hp_by=1`) nebo z odpovědi vyhledávacího systému `Google`[2] (`hp_by=2`). Dotazy na vyhledávače `HPSearch`[1] a `Google` jsou položeny pomocí programu `wget` a získané odpovědi parsovány pomocí parseru `htmlpars`. Získané údaje o URL domovských stránek jsou ve strukturách `persdata` uloženy v položce `hp`.

1 http://HPSearch.uni-trier.de
2 http://www.google.com

Metoda gethp() po získání URL domovských stránek ještě zavolá privátní metodu check_hp(), která vyřadí případné duplicity (např. http://www.domain/~who a http://www.domain/~who/bookmarks.html).

Po úspěšném provedení get_hp() (návratový kód 0) je možné zavolat metodu collect_email(), která projde kontejner pers a ke každé jeho položce (záznam o osobě ve struktuře persdata) se pokusí doplnit informace o emailu (email). Za tímto účelem stáhne pomocí programu wget jednotlivé domovské stránky a podrobí je analýze pro hledání emailu jejich vlastníka. Pokud je analýza úspěšná, je kromě položky email, obsahující url emailu, nastavena i položka email_by=1.

Poslední nevyplněnou položkou ve strukturách persdata kontejneru pers je seq – kontejner obsahující pro danou osobu URL dotazů na další vyhledávací služby. Ten je naplněn voláním metody getseq().

Po získání všech údajů o osobách s daným jménem lze naplněný kontejner pers vyvézt pomocí metody get() třídy person. Získané údaje mohou být použity k dalšímu zpracování, v našem případě k hledání on-line dokumentace osob v kontejneru pers obsažených.

Funkcionalitu třídy person lze vyzkoušet pomocí programu test_getpers, případně z WWW rozhraní přístupného z hlavní stránky systému. Zdrojové texty třídy person jsou uloženy v adresáři $ROOTDIR/src/getpers.

6. Implementace subsystému pro hledání dokumentů

6.1. Datová struktura docdata

```
struct kwdata
{   string value; // vlastni retezec
    string type;  // typ (PLAIN, META, H1, ...)
};

struct docdata
{   string url;          // umisteni souboru na Internetu
    int depth;           // uroven odkazu z hp, ze ktere byl ziskan
    string fname;        // lokalni nazev souboru dokumentu (po stazeni)
    string ftype;        // typ souboru (ps, ps.gz, pdf, html, txt)
    string get_time;     // casove udaje o ziskani souboru
    string create_time;  // casove udaje o vytvoreni souboru
    string header;       // prvnich nekolik znaku textu
    string title;        // nazev clanku (dokumentu)
    vector <kwdata> kw;  // klicova slova
    bool is_coauthor;    // je person spoluautor? (vyuziva trida documents)
};
```

Datová struktura docdata obsahuje položky, které popisují získaný článek nebo publikaci. Jejich význam je vysvětlen v komentářích uvedeného výpisu. Tuto strukturu používá třída documents, která se pro danou osobu popsanou strukturou persdata snaží získat informace o jejích publikacích, které jsou dostupné on-line.

1 Původně bylo pro hledaní domovských stránek plánovano využití služby Ahoy!, jejíž popis je uveden v teoretické části. Služba Ahoy! však koncem roku 2000 bohužel ukončila svoji činnost.

6.2. Třída documents

```
#define HEADER_SIZE 2048
#define MAX_TITLE_LENGTH 128
#define MAX_KEYWORDS_COUNT 5
#define TMPDIR "./tmp/"

class documents
{   persdata person;        // udaje o osobe, ke ktere naplnujeme kont. docs
    vector <docdata> docs;   // dokumenty majici souvislost k dane osobe
    int depth;               // pocet urovni z dom. stranky, ktere projdeme
    list <string> docs_url;  // url depth urovni odkazu z domovske stranky
    bool docs_url_checked;
    string suf;              // povolene typy souboru; nastavi konstruktor
public:
    documents(persdata p, int init_depth, string init_suf, int &code);
    ~documents();
    int get_docs_url(); // z person.hp stahne do docs_url depth urovni odkazu
    int check_url();    // odstrani z docs_url nezadouci cile
    int get_docs();     // stahne dokumenty z url specifikovanych v docs_url
                        // a naplni kontejner docs
    int get_title();   // pro kazdy dok. v kont. docs vyplni polozku title
    int get_kw();      // pro kazdy dokument v kont. docs vyplni polozku kw
    int check_is_coauthor();  // pro dokumenty v docs vyplni is_coauthor
    int show();        // zobrazi obsah person a docs
    int get(vector <docdata> &d);   // vraci kontejner docs
    friend int get_header(docdata &doc1); // prevede doc1.fname do txt a
                        // ziska HEADER_SIZE znaku, kterymi naplni doc1.header
};
```

Třída `documents` slouží pro sestavení kontejneru `docs` obsahujícího popis dokumentů, které mají souvislost s osobou předanou konstruktoru v datové struktuře `persdata`. Datovou strukturu `persdata` naplňuje v oddílu 5.2. popsaná třída `person`. Položky kontejneru `docs` jsou struktury `docdata`, které jsou popsány v 6.1.

Po bezproblémové inicializaci konstruktorem (`code=0`) je naplněna struktura `person` a můžeme volat metodu `get_docs_url()`, která stáhne z domovské stránky `person.hp` `depth` úrovní URL odkazů[1] a uloží je do kontejneru `docs_url`. Tyto odkazy následně filtrujeme pomocí metody `check_url()`, která vyřadí cíle nevyhovující povoleným typům souborů specifikovaných v `suf` (v současné verzi se zpracovávají soubory typu `ps`, `ps.gz`, `pdf`, `html` a `txt`). Tím je vše připraveno pro stahování a následnou analýzu souborů potencionálních vědeckých dokumentů.

Stahování souborů dokumentů realizuje metoda `get_doc()`. Prochází kontejner `docs_url` a pokouší se stáhnout jím odkazované cíle. K tomu používá program `wget`. Pro každé úspěšné stažení dokumentu je v kontejneru `docs` vytvořena nová položka – datová struktura `docdata`, v níž jsou vyplněny údaje `url` (umístění dokumentu na Internetu), `fname` (název souboru lokální kopie dokumentu), `ftype`, `depth`, časové údaje a `header`.

Pro naplnění položky `header` volá metoda `get_doc()` spřátelenou funkci `get_header()`, která do ní dosadí text ze začátku dokumentu. Soubory `ps`, `ps.gz`, `pdf` a `html` je před naplněním `header` nutné konvertovat, případně dekomprimovat. K tomu se použijí programy `pstotext`, `pdftotext`, `gunzip` a třída `htmlpars`.

1 K tomu využívá HTML parser `htmlpars` a program `wget`.

Pro vyplnění zbývajících položek struktur docdata v kontejneru pers slouží metody get_title(), check_is_coauthor() a get_kw(), které analyzují obsah header. Metoda get_title() se pokouší zjistit název dokumentu (title). Metoda check_is_coauthor() se pokusí zjistit, zda je vlastník domovské stránky, ze které vede odkaz na daný dokument, autorem (spoluautorem) tohoto dokumentu. Výsledek uloží do položky is_coauthor. Metoda get_kw() otypuje a dosadí do kw maximálně MAX_KEYWORDS_COUNT klíčových slov dokumentu. Klíčová slova jsou vybrána na základě četnosti technikou TF (Term Frequency) a filtrována seznamem zakázaných slov (stoplist).

Po získání všech údajů o dokumentech majících souvislost s osobou předanou konstruktoru třídy documents v datové struktuře persdata lze naplněný kontejner docs vyvézt pomocí metody get(). Získané údaje mohou být použity k dalšímu zpracování, v našem případě ke konverzi informací o dokumentech do XML souborů.

Třída documents je uložena v adresáři $ROOTDIR/usr/getdoc v souboru getdoc.C. V tomtéž adresáři je uložen i testovací program test_getdoc, jehož WWW rozhraní je přístupné z hlavní WWW stránky vyhledávacího systému.

7. Konverze informací o dokumentech do XML souborů

7.1. Třída xmldocument

V odstavci 6.2. byla popsána třída documents, která v podobě struktur docdata zkompletuje bibliografické informace o dokumentech majících za potencionálního autora osobu předanou konstruktoru ve struktuře persdata. Úkolem třídy xmldocuments je převést informace obsažené ve strukturách docdata a persdata na XML dokumenty podobné tomu z následujícího zkráceného výpisu:

```
<?xml version="1.0"?>
<document>
<person is_coauthor="yes">
     <name by="explicit">Jan Novak</name>
     <homepage by="hpsearch" url="http://www.domena.cz/~jnovak"/>
     <email by="analyze" url="jnovak@domena.cz"/>
     ...
</person>
<time get="Sat Mar 10 12:13:42 2001" create="Sat Mar 10 00:05:42 2001"/>
<location url="http://www.domena.cz/~jnovak/x.html" depth_from_homepage="1"/>
<type>html</type>
<local_copy fname="./tmp/fileVwbYGT.html"/>
<header>
Parsovani XML dokumentu
Pro parsovani XML dokumentu pouzivame parsery. Rozlisujeme dva zakladni...
</header>
<title>Parsovani XML dokumentu</title>
<keywords>
     <keyword value="XML" type="PLAIN,2"/>
     ...
</keywords>
</document>
```

XML dokumenty jsou vytvářeny podle DTD `document.dtd`, které lze nalézt např. v `$ROOTDIR/xml/`, kam jsou XML dokumenty ukládány. Vztah mezi položkami struktur `docdata`, `persdata` a částmi XML dokumentu je zřejmý – stačí si srovnat definici uvedeného DTD a definici struktur `docdata` a `persdata`.

Vlastní převod je zajištěn metodou `write_xml()`. Metoda převezme struktury `persdata` a `docdata` inicializované konstruktorem, a s využitím Xerces-C DOM parseru v paměti vybuduje strom odpovídajícího XML dokumentu. Využívá k tomu standardní metody rozhraní DOM první úrovně, které Xerces-C implementuje (viz `$ROOTDIR/src/getxmldoc/getxmldoc.C`).

```
class xmldocument
{   persdata p;   // struct persdata viz getpers.h
    docdata d;    // struct docdata viz getdoc.h
    public:
    xmldocument(persdata init_p, docdata init_d, int &code);
    ~xmldocument();
    int write_xml(const string fname);
          // z dat predanych konstruktoru vytvori xml soubor se jmenem fname
};
```

Po vytvoření DOM stromu vytvoří metoda `write_xml()` instanci třídy `XMLFormatter`, která zapíše XML dokument obsažený v DOM stromu do souboru specifikovaného v parametru `fname`. Třída `XMLFormatter` vznikla úpravou stejnojmenné třídy obsažené v příkladu `DOMPrint.C`, který je dodáván s balíkem Xerces-C.

Na tomto místě bych chtěl podotknout, že kromě automatického generování XML dokumentů při zodpovídání dotazů vyhledávacím systémem lze XML dokumenty generovat i externě. Stačí k tomu libovolný XML generátor[1], který je schopen vytvářený dokument zvalidovat proti DTD `document.dtd`. Takto lze vytvořit bibliografický popis dokumentu, který systém z nějakého důvodu nenašel, ale jehož údaje jsou známé.

7.2. Použití třídy xmldocument

Třídu `xmldocument` využívá program `test_getxmldoc` a bezprostředně s ní pracuje i program `query` pro obsluhu dotazu na vyhledávací systém. Program `test_getxmldoc` lze volat z příkazové řádky a existuje pro něj i rozhraní dostupné z hlavní WWW stránky systému.

```
./test_getxmldoc -n name -h hp -e email -d depth -p path -a suf
```

Při volání z příkazové řádky je nutné specifikovat alespoň parametry `name` pro jméno osoby (osob), `depth` pro počet úrovní při hledání odkazů na dokumenty z domovské stránky osoby (osob), `path` pro určení adresáře, kam se výsledné XML dokumenty uloží a `suf` pro specifikaci povolených typů souborů.

V první fázi program `test_getxmldoc` pomocí třídy `person` naplní kontejner `p` údaji `persdata` o osobách se jménem `name`. Poté je pro každou osobu z kontejneru `p` vytvořena instance třídy documents. Pomocí ní jsou v kontejneru `d` získány informace `docdata` o dokumentech, které osoba mohla potenciálně napsat (nebo být spoluautorem).

Nyní je sekvenčně procházen kontejner s dokumenty (`d`). Pro každou jeho položku je vytvořena instance třídy `xmldocument` voláním jejího konstruktoru s parametry `docdata` (položka kontejneru `d`, která je právě na řadě) a `persdata` (pro niž byl kontejner `d` vytvořen). Pro tuto instanci je nakonec zavolána metoda `write_xml()` s parametrem `fname`, který vznikne konkatenací `path` a pomocí funkce `tempnam()` vygenerovaného názvu souboru (v adresáři `path`).

1 Lze použít např. XMLGenerator od IBM dostupný na adrese `http://www.alphaworks.ibm.com`.

8. Indexace XML souborů do relační databáze

8.1. Relace vyhledávacího systému spravované Postgresem

Jak již bylo uvedeno ve 3. části, po stažení dostatečného množství dokumentů je vhodné je uložit do tabulek relační databáze Postgres. Tento krok je velmi důležitý z několika důvodů:

1. Vyhledávání v XML dokumentech by bylo možné i bez indexace jejich prostým sekvenčním procházením, z hlediska efektivity je však takový postup samozřejmě neúnosný.
2. Při indexaci se vyřazují případné duplicitní XML dokumenty. Duplicity se mohou objevit právě tehdy, když nejsou dokumenty zaindexovány.[1]

Indexace je realizována pomocí programu `xmltodb`. Ten ke své činnosti využívá Xerces-C SAX parser a třídu `pginterface`. SAX parser slouží k získávání údajů z ukládaných XML dokumentů a třída `pginterface` vytváří vysokoúrovňové rozhraní pro přístup k relacím vyhledávacího systému uložených v databázi Postgres.

Vyhledávací systém udržuje v relační databázi tabulky `authors`, `documents` a `keywords`. Jejich deklarace v jazyce SQL, integritní omezení a ukázky naplnění konkrétními hodnotami jsou uvedeny níže.

```
create table authors (name char(30) not null, name_by int not null,
     hp char(70) not null, hp_by int not null,
     email char(35), email_by int, xmlfname char(25) not null unique);
```

authors						
name	name_by	hp	hp_by	email	email_by	xmlfname
Jan Novak	0	http://.../~jnovak	2	jnovak@...	1	fileG1I1jF.xml

Tabulka `authors` uchovává informace o autorech dokumentů. Všechny atributy kromě `xmlfname` jsou získány z části XML dokumentu obklopené elementem `person`. Atribut `xmlfname` označuje název XML dokumentu, ze kterého byla data získána, a je velmi významný. Pomocí něj je v SQL dotazech možné provést propojení s ostatními tabulkami a tím zodpovědět dotazy typu „autor jménem Novák, který napsal dokument s názvem XML zabývající se SAX parsery“.

```
create table documents (location char(70) not null, type char(5) not null,
     get_time char(25) not null, create_time char(25), local_copy char(30)
     not null, title char(50) not null, xmlfname char(25) not null);
create unique index documents_index on documents (location, xmlfname);
```

documents						
location	type	get_time	create_time	local_copy	title	xmlfname
http://...	ps	Sat...	Nov...	./tmp/file...	XML	fileV2C1jA.xml

V tabulce `documents` jsou uchovávány informace o dokumentech. Tabulka je naplněna těmi informacemi z XML dokumentu, které se nacházejí mezi sekcemi `person` a `keywords`. Ze stejných důvodů jako v tabulce `authors` je zde navíc atribut `xmlfname` označující jméno souboru XML dokumentu, ze kterého byla data získána.

1 Tento fakt je vysvětlen v 10. části týkající se zpracování dotazů vyhledávacím systémem. To je prováděno tak, že je nejříve vznesen dotaz na relační databázi, zda v ní relevantní dokumenty již jsou obsaženy. Pokud ano, pak se nové XML dokumenty negenerují.

```
create table keywords (value char(30) not null, type char(10),
     xmlfname char(25) not null);
create unique index keywords_index on keywords (value, xmlfname);
```

keywords		
value	type	xmlfname
SAX	PLAIN	fileX5a1jb.xml

Poslední tabulkou, kterou vyhledávací systém v databázi Postgres udržuje, je tabulka klíčových slov `keywords`. Ta vzniká z části XML dokumentu ohraničené elementem `keywords`.

V tomto oddílu byly popsány tři tabulky, které si vyhledávací systém v Postgresu udržuje, nicméně např. prostřednictvím interaktivního klienta `psql` lze vytvářet další tabulky a selekce, z nichž mohou být vyvozovány různé statistické závěry o práci vyhledávacího systému.

8.2. Třída pginterface

Třída `pginterface` (`$ROOTDIR/src/xmltodb/pginterface.C`) je vysokoúrovňovým rozhraním pro přístup k relacím, které vyhledávací systém udržuje v databázi Postgres. Třída přitom sama staví na rozhraní `libpq`, což je rozhraní relační databáze Postgres pro jazyk C (je zároveň základem implementací rozhraní pro další programovací jazyky).

Připojení k databázi `marianek`, jejíž vytvoření je popsáno v oddílu 2.2., je provedeno voláním konstruktoru třídy `pginterface`. Tomu je nutné předat několik parametrů, jak je patrné z následujícího výpisu[1]:

```
class pginterface
{    char *pghost, *pgport, *pgoptions, *pgtty, *dbName;
#ifdef DEBUG
     FILE *debug;
#endif   /* DEBUG */
     PGconn *conn;
     PGresult *res;
public:
    pginterface(char *init_pghost, char *init_pgport, char *init_pgoptions,
                char *init_pgtty, char *init_dbName, int &code);
    ~pginterface();
    int inittab_authors();
    ...
    int instotab_authors(string name, int name_by, ..., string xmlfname);
    ...
    int showtab_authors();
    ...
    int open_cursor(PGresult &res, string query);
    int close_cursor(PGresult &res);
};
```

Konstruktor se pokusí připojit[2] na specifikovanou databázi (v našem případě `marianek`) a nastartovat nový transakční blok. Pokud je vše v pořádku, je po návratu z konstruktoru v `code` hodnota `0`. Poté je možné využít některou z metod pro práci s relacemi.

Pro inicializaci tabulek obsahuje třída metody `inittab_...()`, nové hodnoty lze do tabulek vložit pomocí metod `instotab_...()` a konečně zobrazení aktuálního stavu tabulek je implementováno v metodách `showtab_...()`.

Pokud je třeba na databázi položit jiný speciální dotaz, lze k tomu využít metod `open_cursor()` a `close_cursor()`. Metoda `open_cursor()` přijímá na vstupu v parametru `query` specifikaci dotazu, pro který se pokusí prostřednictvím referovaného parametru `res` otevřít nový kurzor. Pokud je kurzor správně otevřen (`PQresultStatus(res) == PGRES_COMMAND_OK`), může volající metoda provést s kurzorem požadované operace. Otevřený kurzor je poté potřeba korektně uzavřít metodou `close_cursor()`.

Uvedené metody vrací hodnotu `0`, pokud operace proběhla bez problémů. Po ukončení práce se třídou je automaticky volán destruktor, který korektně ukončí transakční blok a uzavře spojení s databází.

1 Výpis je zkrácen.
2 Musí běžet backend Postgresu.

8.3. Program xmltodb

Vlastní indexace je realizována pomocí programu `xmltodb` a pomocného unixového skriptu. Program `xmltodb` ke své činnosti využívá Xerces-C SAX parser pro získávání údajů z indexovaných XML dokumentů a instanci třídy `pginterface`, která vytváří vysokoúrovňové rozhraní pro přístup k relacím vyhledávacího systému uložených v databázi Postgres.

Pro parsování XML dokumentu program zavádí třídu `SAXGetHandlers`[1], která je odvozena od standardní třídy `HandlerBase` Xerces-C SAX parseru.

```
#define DBNAME "marianek"
class SAXGetHandlers : public HandlerBase
{   string xmlfname;   // naplnuje konstruktor
    persdata p;         // data potencionalniho autora ziskana parsovanim
    docdata d;         // data dokumentu ziskana parsovanim
    const static int START=0, DOCUMENT=1, PERSON=2, ...      // stavy
    int state;         // stav, udrzeni informace o aktualnim postaveni
     int instodb();     // privatni metoda volana po naplneni struktur p, d;
                        // zajisti ulozeni dat do db, nebo smaze soubor
                        // xmlfname, pokud uz jsou data v db obsazena;
public:
    SAXGetHandlers(string init_xmlfname);
    ~SAXInitHandlers();
     ...
    void startElement(const XMLCh* const name, AttributeList& attributes);
    void endElement(const XMLCh* const name);
    void characters(const XMLCh* const chars, const unsigned int length);
    void ignorableWhitespace(const XMLCh* const chars, const uint length);
     ...
};
```

Třída `SAXGetHandlers` předefinovává ty metody třídy `HandlerBase`, pomocí kterých jsou zachytávány události SAX parseru, které nás zajímají. Ve zkráceném výpisu deklarace třídy `SAXGetHandlers` je uvedena např. metoda `startElement()`, která je vyvolána vždy, když SAX parser narazí na počáteční tag elementu specifikovaného v parametru `name`.

Ve třídě jsou dále definovány konstanty `START`, `DOCUMENT`, `PERSON`, ... Ty jsou dosazovány do privátní proměnné `state`, ve které si třída udržuje informace o aktuálním postavení při parsování XML dokumentu. Pokud je např. `state="EMAIL"` a je zachycena událost `endElement` (stejnojmennou metodou) s parametrem `name="email"`, pak změníme stav proměnné `state` na `PERSON` a čekáme na další události, které pravděpodobně způsobí parsování elementů uvnitř elementu `person`.

Pokud je zachycena událost `endElement` s parametrem `name="document"`, pak již byly získány všechny informace o osobě i dokumentu a jsou prostřednictvím instance třídy `pginterface` vloženy do tabulek relační databáze.

Pro indexaci získaných XML dokumentů lze použít připravený unixový skript `make_index`, který prochází adresář s dokumenty a pro každý z nich zavolá program `xmltodb`:

```
./xmltodb [options] <XML file>
```

Ve volbách programu lze specifikovat, zda se mají dokumenty validovat proti DTD uvedenému v deklaraci XML dokumentu a zda se při parsování mají používat jmenné prostory.

1 Třída `SAXGetHandlers` vznikla úpravou třídy `SAXCountHandlers`, která je obsažena v příkladu `SAXCount` dodávaného s balíkem Xerces-C.

9. Transformace XML

9.1. Třída xmlout

Třída `xmlout` slouží ke konverzi XML souborů vyhovujících `document.dtd` do výstupního HTML souboru. Jména XML souborů jsou konstruktoru předána v kontejneru `init_xmlfnames`, jméno výstupního HTML souboru v parametru `init_htmlfname`. Konstruktor přijímá ještě další parametr `init_resinfo`. Ten může obsahovat informace od vyhledávacího systému, které se mají v odpovědi na zadaný dotaz, tj. vytvářeném HTML souboru, také objevit.

```
// na c333 (u mne doma):
// #define JAVA "/usr/java/jdk1.3.0_02/jre/bin/java"
// #define JAVA_BASE_CLASSES ""
// na faunus:
#define JAVA "/usr/lib/jdk1.1/bin/java "
#define JAVA_BASE_CLASSES "/usr/lib/jdk1.1/lib/classes.zip"
class xmlout
{   vector <string> xmlfnames;    // jmena souboru xml dokumentu
    string resinfo;               // informace od vyhl. systemu
    string htmlfname;             // jmeno vystupniho souboru
public:
    xmlout(vector <string> init_xmlfnames, string init_resinfo,
          string init_htmlfname);
    ~xmlout();
    int transform();
};
```

Vlastní konverze je realizována metodou `transform()`, která ke své prácí používá procesor `LotusXSL`. Jedná se o javovskou třídu pro transformaci XML dokumentů pomocí šablon XSL, které byly popsány v teoretické části. LotusXSL pro svůj běh potřebuje runtime Javy a soubor s XSL šablonami. Ten je v našem případě uložen v souboru `doc.xsl` a jeho začátek ukazuje následující výpis.

```
<?xml version="1.0"?>
<xsl:stylesheet  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
    <xsl:output method="html" indent="yes"/>
    <xsl:template match="/">
        <html>
            <head><title></title></head>
            <body  bgcolor="#ffffff" text="#000000">
                <xsl:apply-templates/>
            </body>
        </html>
    </xsl:template>
    ...
```

Třídu `xmlout`, která je uložena v souboru `xmlout.C` adresáře `$ROOTDIR/src/xmlout`, využívá program `query` při formátování odpovědi na dotaz. Ve stejném adresáři je také uložen unixový skript `xmltohtml`, který umožňuje převést do formátu HTML jeden XML soubor (vyhovující `document.dtd`) předaný skriptu jako parametr.

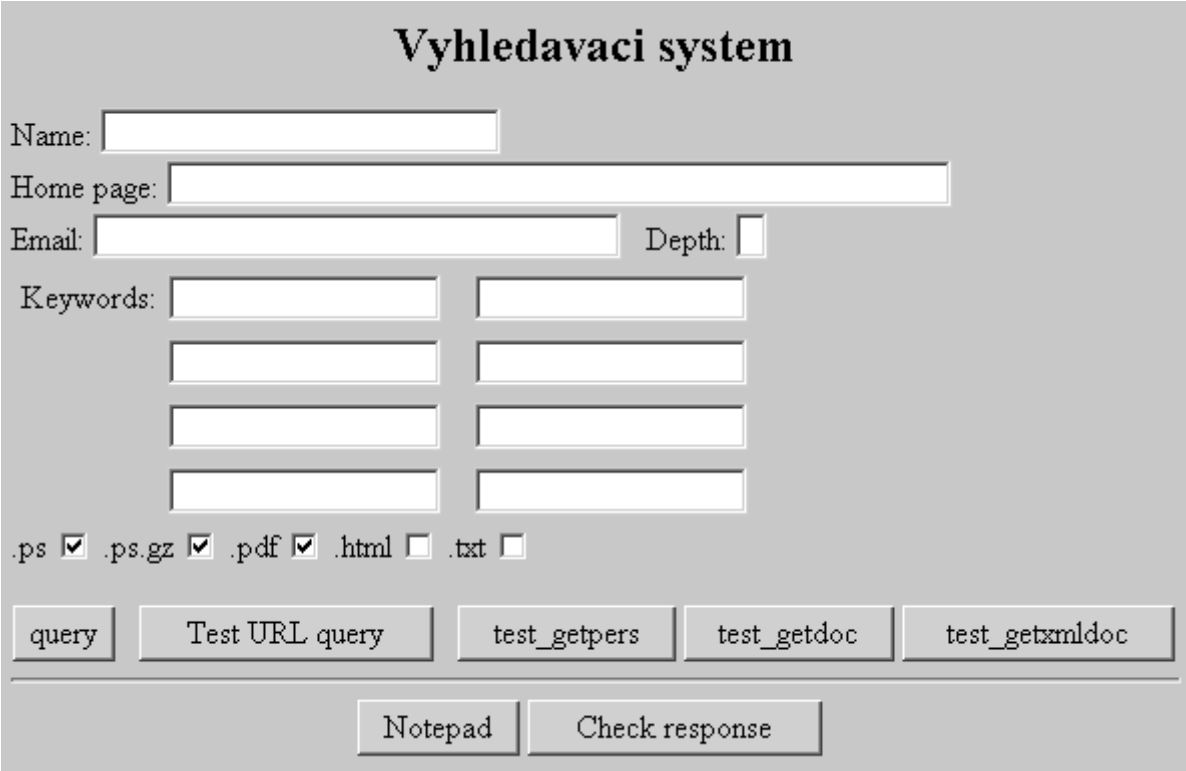
10. Zpracování jednoduchých dotazů

10.1. Program query

Primárním programem, sloužícím k zodpovězení dotazu položeného na vyhledávací systém, je program `query`. Ke své práci využívá třídu `xmldocument` (viz 7.1.), třídu `pginterface` (viz 8.2.) a třídu `xmlout` (viz 9.1.). Program lze volat z příkazové řádky:

```
./query -n name -h hp -e email -d depth -p path -a suf -k kw -b num
```

a je samozřejmě dostupný i z hlavní WWW stránky vyhledávacího systému. Parametry programu `query` mají stejný význam jako u programu `test_getxmldoc` (viz 7.2.). Navíc je zde parametr `-k` pro specifikaci seznamu klíčových slov a parametr `-b`, pomocí něhož lze určit, zda bylo `name` zadáno explicitně nebo získáno analýzou.



Program se prostřednictvím instance třídy `pginterface` nejdříve dotáže relační databáze, zda v ní existují data relevantní dotazu (viz oddíl 8.1.). Pokud tomu tak je, sestaví `query` kontejner s názvy odpovídajících XML dokumentů a pomocí třídy `xmlout` z nich vyrobí výstupní HTML dokument s odpovědí[1].

Pokud v relační databázi nejsou relevantní data nalezena, je proveden algoritmus v podstatě identický s algoritmem realizovaným programem `test_getxmldoc` (viz oddíl 7.2.). Program `query` se pro data dotazu pokusí v síti Internet nalézt relevantní dokumenty. K nim poté sestaví bibliografickou informaci, a tu v podobě XML dokumentů uloží do adresáře `$ROOTDIR/xml`. Ze jmen XML dokumentů sestaví kontejner a pomocí třídy `xmlout` je převede na výstup[1].

1 V části 12.3. je diskutován možný budoucí vývoj systému. Jedním z témat je řazení XML dokumentů ve výstupu podle relevance.

11. Zpracování dotazů na URL HTML stránky

Vzhledem k tomu, že informace o lidech publikujících své články on-line na Internetu se často vyskytují v HTML stránkách různých konferencí, byl vyhledávací systém rozšířen o zpracování dotazů nad URL HTML stránek.

V tomto případě je daná HTML stránka podrobena analýze, jejímž výsledkem může být získání jmen a s nimi spojených dalších údajů (email, URL domovské stránky, klíčová slova), které uživatel může specifikovat v jednoduchém dotazu na konkrétní osobu. Aktivací jednoho takového jména se dotaz na URL HTML stránky převede na jednoduchý dotaz na konkrétní osobu.

Realizace problému je implementována v programech `blmark` a `marknames`.

11.1. Program blmark

Program blmark (`$ROOTDIR/src/blmak/blmark.C`) lze volat z příkazové řádky a existuje pro něj i WWW rozhraní.

```
./blpars -u url -b string -l nn -h nn
```

Program pomocí `wget` stáhne HTML dokument specifikovaný parametrem `url` a pokusí se v něm najít logické celky. Nejdříve z HTML dokumentu vyextrahuje všechny tagy (holé, bez atributů). Poté se pokouší najít co nejvíce a co nejdelších sekvencí tagů, kde minimální délka sekvence je dána parametrem `-l nn` a maximální parametrem `-h nn`. Nalezené sekvence jsou vyhledány v původním HTML dokumentu, kde jsou odděleny řetězcem specifikovaným v parametru `string`.

Pokud algoritmus uspěje, vrátí hodnotu 0 a výsledný HTML dokument s bloky oddělenými řetězcem `string` vypíše na standardní výstup.

11.2. Program marknames, třída queries

Program `marknames` (`$ROOTDIR/src/marknames/marknames.C`) se používá pro vytvoření HTML souboru, v němž jsou jména prezentována jako odkazy na program `query` (viz 10.1.). Existuje pro něj WWW rozhraní a z příkazové řádky se volá s následujícími parametry:

```
./marknames -u url -b string -l nn -h nn -f nn > out.html
```

Význam parametrů `-u`, `-b`, `-l`, `-h` je stejný jako u programu `blmark`. Parametr `-f` slouží pro filtraci jmen pomocí slovníku `ecd`. Slovník `ecd` vrátí v odpovědi na dotazované potencionální jméno určitý počet položek. Parametr `-f` určuje horní mez, při které bude ještě dotaz na `ecd` považován za jméno.

Pro získání jmen a s nimi asociovaných dalších údajů je v programu `marknames` definována třída `queries`. Ta nejdříve zavolá program `blmark`, který stáhne HTML stránku specifikovanou v parametru `-u` a řetězcem `string` v ní označí logické celky. Poté se z jednotlivých bloků pokusí získat data, ze kterých se sestaví dotazy na program `query`.

Jména se hledají ve dvou krocích. V prvním je za jméno považována každá sekvence slov s velkými písmeny oddělených „bílým“ místem. V této fázi výpočtu je úspěšnost nalezení jmen obvykle pod hranicí 50 %. Nalezená potencionální jména jsou dále filtrována pomocí slovníkové filtrace s využitím slovníku `ecd`. Pomocí této spekulativní metody program `marknames` obvykle dosahuje přesnosti 70 %.

12. Výsledky a zhodnocení, návrhy na další vývoj

12.1. Jednoduché dotazy

Pro zhodnocení výsledků systému jsem mu prostřednictvím programu `query` položil dotazy na různá jména a tytéž dotazy jsem položil i systémům Google, ResearchIndex a Cora. Na dotaz

```
query -n "Petr Sojka" -h "http://www.fi.muni.cz/usr/sojka/"\
      -d 2 -p ./xml/ -a pdf -b 0
```

vrátil vyhledávací systém v odpovědi 26 položek zastupujících 26 nalezených `pdf` dokumentů. Z dotazu lze vypozorovat, že systému byla explicitně předána domovská stránka uživatele Petr Sojka. To proto, že domovská stránka `http://www.fi.muni.cz/~sojka/`, kterou systém nalezl sám, není udržována. Dále je vidět, že nebylo využito hledání do maximální hloubky odkazů, ale pouze do jejich druhé úrovně. Zobrazení začátku výsledku vrácených systémem ukazuje následující obrázek.

Search result

```
In the database relevant data were not found, continue to search in the Internet.
Time needed for get informations about persons: 5 s
Time needed for get persons documents and converting to text: 49 s
Time needed for get persons documents and converting to text: 1 s
Time needed for converting documents to XML and preparing HTML output: 0 s
Total response time: 55 s
```

Petr Sojka, name by explicit (see too with Computer Science Bibliography , Cora , Google(pdf) , HBP Bibliographies , ResearchIndex , XXX ,)
homepage http://www.fi.muni.cz/usr/sojka/ by explicit, email
is coauthor: yes of pdf document:

Notes on Compound Word Hyphenation in TEX
http://www.fi.muni.cz/usr/sojka/papers/tug95.pdf
Notes on Compound Word Hyphenation in TEX Petr Sojka Faculty of Informatics Masaryk University Brno Bure sova 20, 602 00 Brno Czech Republic Internet: sojka@muni.cz Abstract The problems of the automatic compound word and discretionary hyphenation in TEX are discussed. These hyphenation points have to be marked manually in the TEX source file so far. Several methods how to tackle with these problems are observed. The results obtained from experiments with German word-list are discussed. 1 Motivation only be

keywords: Brno (PLAIN,2), discussed. (PLAIN,2), hyphenation (PLAIN,2), problems (PLAIN,2), Abstract (PLAIN,1), Bure (PLAIN,1), Compound (PLAIN,1), Czech (PLAIN,1),
get Sun Apr 8 10:54:52 2001, create , local copy in ./tmp/filemT5TLK.pdf

---

Systém v odpovědi sděluje, že požadovaná data nejsou uložena v relační databázi a bylo proto provedeno hledání v síti Internet. Následují časové údaje o délce trvání jednotlivých pracovních fází, za nimiž už jsou seřazeny informace o nalezených dokumentech. Můžeme z nich vyčíst, že uživatel Petr Sojka, jehož jméno bylo zadáno explicitně, je autorem `pdf` dokumentu s názvem „Notes on Compound World Hyphenation in TEX“. V odpovědi jsou dále uvedeny údaje o domovské stránce a email[1] uživatele, dotazy na uživatele na další vyhledávače, klíčová slova vybraná z dokumentu, údaje o získání a vytvoření dokumentu a údaj o lokálním uložení kopie.

Po zaindexování dokumentů do relační databáze se maximální odezva systému pohybuje v jednotkách sekund.

Při dotazu na systém Google je vráceno více odpovědí, ale mezi nimi je méně `pdf` dokumentů. Pokud by byl dotaz omezen na `ps` dokumenty, vyhledávací systém stále vrací několik desítek dokumentů. Google `ps` dokumenty neindexuje.

1 Email v tomto případě nalezen nebyl, je to však vyjímečný případ. Zpravidla bývá získán analýzou domovské stránky.

V odpovědi Google chybí některé údaje vyhledávacím systémem poskytované jako např. email uživatele, dotazy na další vyhledávací systémy a klíčová slova dokumentu.

Dotaz na uživatele Petr Sojka jsem položil i systémům specializovaným podobně jako implementovaný vyhledávací systém. ResearchIndex vrací méně `pdf` dokumentů, ale všechny mají spojitost s osobou Petr Sojka. Implementovaný vyhledávací systém vrací i dokumenty, které s Petrem Sojkou nemají nic společného a označí je `"coauthor: maybe"`. ResearchIndex vrací také citace dokumentů, které mají spojitost s dotazovanou osobou, tyto dokumenty však nemusí být přístupné online.

Systém Cora nabízí jediný dokument, který má spojitost s osobou Petr Sojka (`http://www.loria.fr/tex/general/eurotex98/eurotex98-sojka.ps.gz`). Je to způsobeno tím, že systém Cora má zaindexovány dokumenty pouze z několika institucí v USA. K nalezenému dokumentu systém vrací i BibTeX záznam a další informace o dokumentu.

Vyhledávací systém jsem porovnával s dalšími vyhledávači i jinými dotazy. Na dotaz

```
query -n "Pavel Rychly" -d 3 -p ./xml/ -a ps,ps.gz -b 0
```

nachází vyhledávací systém několik desítek postscriptových dokumentů. Cora nevrací z výše uvedených důvodů ani jeden relevantní dokument, ResearchIndex také tak.

12.2. Dotazy na URL HTML stránky

Pro zhodnocení úspěšnosti dotazu na URL HTML stránky jsem systému předal ke zpracování HTML stránku s informacemi o konferenci IWPT 2000 uloženou lokálně v souboru `doc.html`.

After IWPT 2000

This page was created after the workshop. It contains links to online versions of the papers. Some are corrected or expanded versions of what appears in the printed proceedings.

Invited Talks

Automatic Grammar Induction: Combining, Reducing and Doing Nothing
Eric Brill
Guides and Oracles for Linear-time Parsing
Martin Kay
Parsing Techniques for Lexicalized Context-free Grammars (Powerpoint slides from the talk also available)
Giorgio Satta

Papers

A Bootstrapping Approach to Parser Development
Izaskun Aldezabal, Koldo Gojenola, Kepa Sarasola
New Tabular Algorithms for LIG Parsing
Miguel Alonso, Eric de la Clergerie, Jorge Graña, Manuel Vilares
Customizable Modular Lexicalized Parsing
Roberto Basili, Maria Teresa Pazienza, Fabio Zanzotto
Range Concatenation Grammars
Pierre Boullier
Automated Extraction of TAGs from the Penn Treebank
John Chen, K. Vijay-Shanker

Vyhledávací systém sestavil z předaných parametrů dotaz na program `marknames`:

```
marknames -u doc.html -b "<!--BLOCK-->" -l 2 -h 5 -f 5
```

Vyhledávací systém v textu HTML stránky rozpoznal naivní metodou (viz 11.2.) 157 potencionálních jmen. Po použití slovníkové filtrace se tento počet zmenšil na 89 potencionálních jmen. V 60 případech se skutečně jednalo o jména, úspěšnost analýzy je tedy v tomto případě 67%. Následující část výsledného HTML dokumentu ukazuje, jak program `marknames` sestavil dotazy na primární vyhledávací program `query`:

```
<HTML><HEAD><TITLE>Marknames</TITLE></HEAD><BODY>
<UL>
<LI><A HREF="query -n 'Eric Brill' -d 3 -p './xml/' -a 'ps,ps.gz,pdf,html,txt'
-b 1 -k 'Automatic(BLOCK),Grammar(BLOCK), Induction:(BLOCK),Combining,
(BLOCK),Reducing(BLOCK),and Doing Nothing(BLOCK),IWPT(H1),2000(H1),'">
Eric Brill</A>
...
```

12.3. Zhodnocení vyhledávacího systému

12.3.1. Kladné vlastnosti vyhledávacího systému

+ vyhledávací systém hledá, i když nemá data uložena v relační databázi
+ je možné klást dotazy na URL HTML stránky
+ zpracovává vektorové ps, ps.gz, pdf dokumenty
+ dobrý základ pro další vývoj systému
+ možnost klást dotazy i z psql klienta přímo do relační databáze

12.3.1. Nedostatky vyhledávacího systému

- nepoužívá vyspělé techniky strojového učení
- rychlost při hledání v případě, že data nejsou v relační databázi
- nedostatečná implementace řazení dokumentů podle relevance
- nedostatečná práce se znaky národních abeced
- chyby při parsování

12.4. Návrhy na další vývoj systému

1. Do datové struktury `docdata` by bylo možné přidat položku `citations` obsahující url citací odkazovaných z dokumentu. V případě `html` a `pdf` dokumentů lze informaci o citacích číst přímo ze zdrojového kódu dokumentů, v případě textových a `ps` souborů by byla analýza citací náročnější (viz např. systém ResearchIndex popsaný v teoretické části).
2. Po dostatečném naplnění relační databáze by bylo možné dolovat z ní jistý typ znalostí. Stejně tak by bylo možné provést statistické vyhodnocení dat.
3. Pro dosažení kvalitnějších odpovědí by bylo možné optimalizovat SQL dotazy kladené programem `query` na relační databázi.
4. V budoucí verzi vyhledávacího systému by bylo možné některé jeho úlohy realizovat s využitím technik strojového učení. Inspiraci lze nalézt například v systémech Cora a ResearchIndex.
5. Bibliografická informace obsažená v XML souborech generovaných systémem je dosti jednoduchá. Bylo by vhodné extrahovat a uchovávat informace o instituci, ve které autor dokumentu pracuje. V případě, že dokument pochází z konference, bylo by vhodné uchovávat i informace o ní. Existuje mnoho dalších položek, o které by bylo možné současnou verzi bibliografické informace uchovávané systémem rozšířit.

6. Internet je dynamický systém. Některá URL zanikají a nová vznikají, v současné verzi však systém umožňuje odstranění neexistujících URL jinak, než přímou prací s relační databází, například pomocí klienta `psql`.
7. Pomocí SQL dotazů lze z dat shromažďovaných systémem získat mnoho informací, například o oblastech, pro něž je počet nalezených dokumentů maximální apod. Tyto služby by bylo možné zpřístupnit i z webového rozhraní vyhledávacího systému.
8. V současné době je stále mnoho vědeckých dokumentů přístupných pouze v papírové podobě. Systém by mohl obsahovat nástroj, který by s využitím techniky OCR digitalizoval a evidoval i tyto papírové dokumenty.
9. Systém při své práci stahuje kopie dokumentů na lokální disk. Vzniklou sbírku souborů by bylo možné dále zpracovat do fulltextové podoby (tímto tématem se zabývá na FI reálně vyvíjený projekt).

Obsah přiloženého CD

Na přiloženém CD disku lze nalézt:

- zdrojové kódy a ostatní související části vyhledávajícího systému,
- text této diplomové práce,
- materiály, ze kterých jsem čerpal, přístupné v elektronické podobě,
- softwarové balíky vyžadované vyhledávacím systémem,
- testovací data využitá při implementaci vyhledávacího systému.

Literatura

[1] Shakes, j; Langheinruch, M.; Etzioni, O.: Dynamic Reference Sifting: a Case Study in the Homepage Domain, in Proceedings of the Sixth International World Wide Web Conference, pp. 189–200, 1997.

[2] Brin, S.; Page, L.: The Anatomy of a Large-Scale Hypertextual Web Search Engine, 1998.

[3] Page, L.; Brin, S.; Motwani, R.; Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web, 1998.

[4] Peterka, J.: Hledáte někoho na Internetu, SW noviny číslo 9/1999, strany 24–42.

[5] Sborník příspěvků ze semináře Knihovny budoucnosti, CASLIN, 8.–12. června 1997, Dlouhé stráně, Česká republika, vydala MU Brno, 1998.

[6] P070, Vybrané partie z KIV, přednáška na FI MU Brno, 2000.

[7] DATASEM '99, 19. ročník databázové konference, sborník přednášek, 24.–26. říjen 1999, Brno.

[8] Botík, R.: Inteligentní vyhledávání ve WWW. Diplomová práce FI MU, Brno, 1998.

[9] Boyan, J.; Freitag, D.; Thoriten, J.: A Machine Learning Architecture for Optimizing Web Search Engines, Carnegie Mellon University, 1996.

[10] Filip, J.: IVS – Inteligentní vyhledávací systém, Diplomová práce FI MU Brno 1999.

[11] Röscheisen, M.; Baldonado, M.; Chang, K.; Gravano, L.; Ketchpel, S.: Paepcke, A.: The Stanford InfoBus and Its Service Layers, 1997.

[12] Halaška, I.; Pokorný, J.: Databázové systémy, Cvičení (Jazyk SQL a systém ORACLE), ČVUT Praha, 1995.

[13] Pokorný, J.: Dotazovací jazyky, Science, 1994.

[14] Staudek, J.: Organizace souborů, přednáška na FI MU Brno, 2000.

[15] Lawrence, S.; Giles C., L.; Bollacker, K.: Digital Libraries and Autonomous Citation Indexing, NEC Research Institute, 1999.

[16] McCallum, A.; Knigam, K.; Rennie, J.; Seymore, K.: Automating the Construction of Internet Portals with Machine Learning, Carnegie Mellon University, 1999.

[17] Patashnik, O.: BibTeXing, 1988.

[18] Rybička, J.: LaTeX pro začátečníky, Konvoj, 1995.

[19] Robie, J.; Lapp, J.; Schack, D.: XML Query Language (XQL), 1999.

[20] Marschal, B.: XML v příkladech, Que, 1999, český překlad Computer Press, 2000.

[21] Materiály konsorcia W3C, `http://www.w3.org/`

[22] Konopnicky, D.; Shmueli, O.: Early Experience with W3QS – A WWW Information Gathering System, 1999.

[23] Schlesinger, Hlaváč: Deset přednášek z teorie strukturního rozpoznávání, ČVUT, 1999.

[24] P090, Unix – seminář ze správy systému, FI MU Brno, 2000.

[25] Harbison, S., P.; Steele G., L.: Referenční příručka jazyka C, Prentice Hall 1991, český překlad Science, 1996.

[26] Kernigham, B., W.,; Pike, R.: Programové prostředí operačního systému Unix, Bell Telephone Laboratories 1984, český překlad Science, 1996.

[27] Virius, M.:Programování v C++, skripta ČVUT, 1998.

[28] Stones, R.; Matthew, N.: Linux, začínáme programovat, Wrox Press 1999, český překlad Computer Press, 2000.

[29] Witten, I., H.; Moffat, A.; Bell, T., C.: Managing Gigabytes, Compressing and Indexing Documents and Images, Van Nostrand Reinhold, 1994.