České vysoké učení technické v Praze – Fakulta elektrotechnická
Katedra řídicí techniky

Diplomová práce

Předzpracování dokumentů pro WEB

2003 **Libor Beneš**

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne ………………………. ……………………………………

podpis

Poděkování

Rád bych na tomto místě poděkoval především svému vedoucímu diplomové práce ing. Richardu Šustovi za čas, který mi věnoval a za cenné připomínky a rady, které přispěly k vylepšení této práce. Dále bych chtěl poděkovat Lence a dalším za morální podporu.

Abstrakt

Tato diplomová práce řeší problém automatického vytváření a aktualizace odkazů v HTML dokumentech podle dat (adres a textů odkazů) uložených v databázi. Tato data lze získat z dokumentů, které mají tyto odkazy již vyznačeny.

Z analýzy jednotlivých úkolů vyplynul postup pro úpravu HTML dokumentů, aby bylo možné jejich přímé zpracování. Úlohu vytváření odkazů se povedlo převést na zpracování textu jednoduchým automatem, pro který byla navržena generující gramatika. Úloha ukládání odkazů řeší problém s nalezením textu a adresy odkazu, zpracováním adres v relativním tvaru a možností, že jeden odkaz tvoří více elementů. Aktualizace odkazů, která je spojením obou předešlých úloh, již řeší pouze problém s tvorbu nového dokumentu. Práce zahrnuje i návrh databáze a sestavení algoritmů pro jednotlivé úlohy. Navržená řešení byla ověřena experimenty.

Abstract

This diploma thesis solves the problem of automatic creating and updating hyperlinks in the HTML documents according to the data (addresses and texts of hyperlinks) which are saved in the database. It is possible to obtain these data from the documents, where the hyperlinks have already been created.

From the analysis of each individual problem the procedure for adaptation of HTML documents emerges so as the direct processing of them is possible. The task of creating hyperlinks was successfully transferred to procession of text by a simple automatic machine for which generation grammar was designed. The task of depositing the hyperlinks solves the problem with finding the text and address of the hyperlinks, the problem with processing the address in a relative form and the problem that hyperlinks composite more than one element. The task of updating hyperlinks, which is the combination of both previous tasks, solves then the problem with creating a new document. This diploma thesis also includes the proposal of database and compiling of algorithms for individual tasks. All the designed solutions were attested by experiments.

Obsah

1.ÚVOD **1**

2. HTML – HYPERTEXT MARKUP LANGUAGE **3**

2.1 CO JE HTML? 3
2.2 SYNTAXE HTML 3
2.2.1 Element *3*
2.2.2 Tag *4*
2.2.3 Atribut *4*
2.2.4 Znakové entity *5*
2.3 HYPERTEXTOVÝ ODKAZ 5
2.3.1 Vytváření odkazu *6*
2.4 HTML V PRAXI 7

3. ROZBOR ZADÁNÍ **9**

3.1 VYTVÁŘENÍ ODKAZŮ 9
3.1.1 Vstupy a výstupy *9*
3.1.2 Specifikace problému *9*
3.1.3 Zachování původní struktury *10*
3.1.4 Estetické hledisko *10*
3.1.5 Hledání textů odkazů *11*
3.1.6 Kategorizace textů *11*
3.1.7 Kategorizace slov *12*
3.1.8 Generování tvarů *13*
3.1.9 Zkracování a porovnávání slov *13*
3.1.10 Shrnutí *15*
3.2 UKLÁDÁNÍ ODKAZŮ 15
3.2.1 Vstupy a výstupy *16*
3.2.2 Specifikace problému *16*
3.2.3 Lokalizace elementů A a BASE *16*
3.2.4 Zpracování adresy odkazu *17*
3.2.5 Zpracování děleného odkazu *17*
3.2.6 Získání textu odkazu *18*
3.2.7 Shrnutí *19*
3.3 AKTUALIZACE ODKAZŮ 19
3.3.1 Vstupy a výstupy *19*
3.3.2 Specifikace problému *19*

4. PŘEDZPRACOVÁNÍ HTML DOKUMENTU **21**

4.1 PRINCIP PŘEDZPRACOVÁNÍ 21
4.2 ROZPOZNÁNÍ KÓDU DOKUMENTU 22
4.2.1 Rozpoznání tagů *22*
4.2.2 Rozpoznání entit *23*
4.2.3 Rozpoznání textů *24*
4.2.4 Realizace rozpoznávání *25*
4.3 SLUČOVÁNÍ FRAGMENTŮ – TVORBA SLOV 26
4.3.1 Princip slučování *27*
4.3.2 Reprezentace slov *27*

5. NÁVRH GRAMATIK **29**
5.1 CO JE TO GENERATIVNÍ GRAMATIKA? 29
5.2 GENEROVÁNÍ TEXTŮ 29
5.2.1 Terminály *29*
5.2.2 Neterminály *30*
5.2.3 Přepisovací pravidla *30*
5.3 POUŽITÍ GRAMATIK 31
5.3.1 Název a zkratkový název *31*
5.3.2 Jméno osoby *31*
6. DATABÁZE **33**
6.1 SQL A MYSQL 33
6.2 NÁVRH DATABÁZE 33
6.2.1 Konstrukce E-R modelu *34*
6.2.2 Relační schéma databáze *37*
7. ALGORITMY **40**
7.1 ALGORITMUS PRO SPRÁVU ADRES 40
7.1 ALGORITMUS UKLÁDÁNÍ ODKAZŮ 41
7.2 ALGORITMUS AKTUALIZACE ODKAZŮ 42
7.3 ALGORITMUS VYTVÁŘENÍ ODKAZŮ 44
7.4 OPTIMALIZACE ALGORITMŮ 48
8. EXPERIMENTY **50**
8.1 IMPLEMENTACE 50
8.2 TESTY 51
9. ZÁVĚR **52**
LITERATURA **54**

Seznam obrázků

2.1. Příklad elementu. 3
2.2. Tvar počátečního a koncového tagu. 4
2.3. Příklad elementu s počátečním tagem s atributy. 4
2.4. Tvar zápisu znakových entit (a) pomocí jejich jména, (b) pomocí kódu znakové sady Unicode. 5
2.5. Příklad URI složené ze tří částí. 5
2.6. Ukázka zápisu odkazu v absolutním tvaru (v prohlížeči i ve zdrojovém kódu). 6
2.8. Možnosti zápisu stejného textu. Ukázka rozdílu mezi výsledným zobrazením v prohlížeči a zdrojovým kódem dokumentu. 8
3.1. Ukázka stejně vypadajících odkazů (s výpisem zdrojového kódu) tvořených (a) třemi částmi, (b) jednou částí. Tučně je ve zdrojovém kódu vyznačen počátek a konec odkazu. 18
4.1. Princip předzpracování. 22
4.2. Rozdělení věty na jednotlivé texty. 24
4.3. Rozdělení vět oddělených pouze znakem, (a) původní tvar, (b) rozdělení na texty, (c) rozdělení na texty, které tvoří buď znaky, číslice nebo symboly. 24
4.4. Předzpracování složitější věty (a), větu rozdělíme na jednotlivé fragmenty (b) a rozpoznáme je (c). Označíme první fragment v dokumentu a fragmenty, které předcházel bílý znak – začátky slov (d) a z fragmentů vytvoříme slova (e). 25
6.1. Základní E-R model. 34
6.2. Základní E-R model s vyznačenou kardinalitou a členstvím ve vztahu. 34
6.3. Základní E-R model s integritním omezením. 34
6.4. M:N diagram výskytů (a) pro vztah „*Tvoří odkaz s*“, (b) pro rozložený vztah na dva vztahy 1:N „*Tvoří*“ a „*Odkazuje na*“. 35
6.5. Dekomponovaný E-R model. 35
6.6. E-R model s vyznačenými atributy entitních typů (tučně jsou vyznačeny klíčové atributy). 36
6.7. Výsledný E-R model. 36
6.8. Grafický zápis schématu databáze. 39
7.1. Označení slov, která databáze obsahuje. 44
7.2. Obsah databáze po uložení odkazů. 45
7.3. Označení slov a jejich zkratek uložených v databázi, spolu s připojeným seznamem čísel adres, ke kterým se jednotlivá slova vážou. 46
7.4. Příklad označení slov u jména „*ing. J. Novák*“ čísly adres. Připojené tabulky obsahují seznam možností před setřídění a po setřídění podle počáteční pozice a délky textu. V druhé tabulce jsou zvýrazněny možnosti, které mohou tvořit přípustnou kombinaci. 46
7.5. Rozdělení textu na úseky. 47
8.1. Hlavní okno aplikace s popisem jednotlivých částí. 50

Seznam tabulek

1. Příklady převodů adresy v relativním tvaru na absolutní podle místa uvedení. 6
2. Vlastnosti skupin slov. 13
3. Příklady porovnání dvou slov v databázi a v textu. 14
4. Označení skupin slov terminálními symboly. 29
5. Relační schéma - entitní typy s příslušnými atributy. 37
6. Datové typy všech atributů (N – celočíselná hodnota, X – pole znaků) i pro MySQL a rozlišení klíčových atributů. 37
7. Nastavení příznaku *form* podle typu a tvaru fragmentu a dostupnosti základní adresy. 40

1.Úvod

Tato diplomová práce si klade za cíl vytvořit postupy, které usnadní vytváření nebo aktualizaci hypertextových odkazů (dále jen odkazů) v dokumentech prezentovaných prostřednictvím internetu.

V těchto dokumentech obvykle autor odkazy vyznačí při jejich vytváření. Texty, které takto nevznikly (například elektronická pošta, zápisy z úředních jednání atd.), nemají odkazy vyznačeny a pokud nechceme, aby přišly o největší výhodu této prezentace, musí projít úpravou.

Druhým typem dokumentů vyžadující úpravu tvoří ty, jejichž odkazy již nemají adresáta (nefunkční odkazy). U těchto odkazů stačí aktualizovat adresu.

Vytvoření (případně aktualizace) jednoho odkazu není žádný problém. Postačí nám k tomu obyčejný textový editor. Vyhledáme příslušný text a vytvoříme odkaz s odpovídající adresou. Celá operace nám zabere krátkou dobu avšak, máme-li takto zpracovat stovky dokumentů s desítkami různých odkazů v různých tvarech a podobách, stane se tato činnost naší noční můrou. Práci komplikuje hned několik skutečností:

1. *Různé tvary textů*. Texty odkazů nemusí být uvedeny právě v tom tvaru, v jakém ho budeme hledat. Například hledáme text:

   „*František Ptáček*“

   a v dokumentu může být uvedeno například:

   „*F.Ptáček*“,
   „*Ptáček František*“,
   „*PTÁČEK, F.*“.

2. *Podobnost jmen*. Může se objevit více podobných jmen, stejná příjmení nebo stejná křestní jména.
3. *Fragmenty jazyka HTML*. V případě, že upravujeme HTML dokument, nám všechno mohou zkomplikovat fragmenty jazyka HTML.
4. *Estetické hledisko*. Nezanedbatelným faktorem je estetické hledisko, velké množství stejných odkazů (stejný text v jakémkoliv tvaru i adresa) působí spíše rušivým dojmem.
5. *Zápis adresy*. Zapsat adresu odkazů také nemusí být úplně triviální. Pod pojmem „internetovská adresa“ si většinou představíme tvar:

   „*http://www.adresa.cz/*“.

   Zápis takové krátké adresy nečiní žádný problém. Odkazy mají většinou konkrétního adresáta. Obsahují proto více upřesňujících a doplňujících informací, nejenom typ protokolu a adresu hostitelského počítače. Můžeme mít například takové:

   „*http://dir.yahoo.com/Computers_and_Internet/Information_and_Documentation/ Data_Formats/HTML/Validation_and_Checkers/index.dhtml#W3C*“

   „*http://validator.w3.org/check?uri=http%3A%2F%2Fwww.feld.cvut.cz%2F*“

   Neudělat chybu při přepisování takové adresy je téměř nadlidský úkol. Navíc u adres obecně záleží na velikosti písmen, což nám práci ještě více ztíží.

Rozvoj internetu vedl i k rozvoji vizuálních prostředků (například HomeSite, Coffee HTML, HoTMetaL atd.), které nám sice umožní rychlou úpravu textu, ale s vytvořením nebo aktualizací odkazů výrazně nepomohou. Text, který má být označen jako odkaz, musíme opět najít a také musíme znát a zadat příslušnou adresu odkazu.

Novější komerční produkty, například FontPage 2000, již dovolují vytváření odkazů podle databáze. Za výhodu lze považovat to, že si při úpravě dokumentů nemusíme pamatovat všechny texty a adresy. Máme ovšem malý vliv na proces vytváření odkazů, který vyznačí pouze texty ve tvaru uvedeném v databázi. Také tvorba a aktualizace databáze vyžaduje stejné úsilí jako při vytváření odkazů za pomoci zmíněných vizuálních prostředků.

Projekty, které se zabývaly automatickým formátováním (například [12], [14]), řeší pouze druhou část tohoto problému – platností odkazů a jejich případnou aktualizací. Jejich úkolem je takové odkazy v dokumentu najít, otestovat a podle centrální databáze buď v dokumentu aktualizovat nebo z dokumentu odstranit. Existují internetoví agenti (například [17]), kteří umí ohlásit neplatnost odkazu. Testování platnosti adres odkazů v této práci neuvažujeme, ale může to být námět na její další rozšíření.

Další projekty (například [13]) se zabývají vytvořením univerzální distribuované databáze odkazů, která umí na dotaz vracet příslušnou adresu k textu odkazu nebo spíše k médiu (obrázek, video, dokument atd.) pomocí speciálních značek HYPER-G [15]. Formátování ponechává systém na straně uživatele. Jeho výhodou je jednotnost pro všechny registrované uživatele a také to, že při změně adresy v databázi automaticky upozorní na změnu i uživatele.

V této práci navrhneme postupy, které umožní vytvářet a aktualizovat odkazy v dokumentech podle dat uložených v lokální nebo vzdálené databázi MySQL. Tyto data získáme z dokumentů, které mají tyto odkazy již vytvořené. Při vytváření odkazů budeme hledět i na estetické hledisko. Všechny cíle a požadavky, které vyplývají z předchozího textu, si nyní stručně shrneme. Doplněním dalších bodů sestavíme rozšířené zadání této práce (dále jen rozšířené zadání):

- získávání dat (adresy a texty odkazů) z odkazů v již existujících dokumentech,
- vytváření nových odkazů podle dat z databáze,
- aktualizace adres v existujících odkazech podle dat z databáze,
- nevytvářet stejný odkaz v rámci většího celku, například v rámci odstavce,
- umožnit vytvoření nebo aktualizaci u textů ve všech přípustných tvarech,
- zachovaní původní struktury dokumentů,
- umožnit připojení na lokální nebo vzdálenou databázi.

Hodnocení, jak se podařilo splnit jednotlivé body rozšířeného zadání, je uvedeno v poslední 9. kapitole. Následující kapitola se věnuje jazyku HTML a jeho problematice. V 3. kapitole analyzujeme základní úkoly této práce, zjistíme jaké mohou nastat problémy a navrhneme jejich řešení. Předzpracování dokumentů se věnuje 4. kapitola, půjde nám o přípravu dokumentů pro další zpracování a sjednocení přístupu k textu dokumentu. V 5. kapitole provedeme návrh generujících gramatik pro tvorbu přípustných tvarů jednotlivých textů. Kapitola 6 se věnuje návrhu databáze. Algoritmy pro jednotlivé úlohy navrhneme a ucelíme v 7. kapitole a uvedeme možnosti jejich optimalizace. Kapitola 8 se zmiňuje o implementaci a o provedených testech.

2. HTML – Hypertext Markup Language

Dokumenty uvažované v této práci jsou téměř výhradně psané v jazyce HTML. Problematika tohoto jazyka nemusí být zcela jasná, proto si v této kapitole nejdříve představíme základy a principy jazyka HTML a ukážeme z jakých částí se skládá. Seznámíme se blíže s odkazy, se způsoby a možnostmi jejich vytváření. Nakonec si ukážeme, jaké mohou nastat rozdíly mezi zdrojovým kódem dokumentu a tím, jak jej zobrazí internetovský prohlížeč (dále jen prohlížeč).

2.1 Co je HTML?

HTML je jedna ze tří základních technologií služby WWW (World Wide Web). Jde o značkovací jazyk založený na SGML (Standard Generalized Markup Language) pro popis struktury dokumentu pro službu WWW, který je nezávislý na platformě a definovaný také pomocí SGML.

První formální specifikací byla HTML 2.0, která zahrnovala základní formátování a strukturování dokumentu, obrázky a formuláře. Tvůrci prohlížečů si vytvářely vlastní nestandardní formátovací značky, a proto vzniklo konsorcium W3C (World-Wide Web Consorcium). Jde o sdružení velkých softwarových firem, které se stará o standardy WWW, tedy i o standard HTML.

V roce 1996 konsorcium W3C schválilo HTML 3.2 [3], které přineslo velkou řadu rozšíření. Zatím poslední verze HTML 4.01 [4] z roku 1999 opravuje drobné chyby ve specifikaci HTML 4.0 [2]. Poslední vývoj v této oblasti vede na XHTML (Extended HTML) a prvním krokem je právě HTML 4.01. Takto tvořené dokumenty striktně dodržují normu, obsahují více možností (XML) a přitom zahrnují i původní HTML.

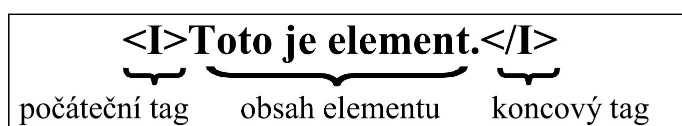
2.2 Syntaxe HTML

Formátování HTML dokumentů definuje norma a není závislé na platformě. To znamená, že například počet mezer mezi slovy nebo přechod na nový řádek (pokud není vynucen nebo zakázán) závisí pouze na velikosti okna, ve kterém je text zobrazen. Původní formátování textu se ignoruje, s výjimkou případu, kdy pomocí odpovídajících značek jeho použití vynutíme (například u výpisu zdrojového kódu programu).

Norma dále rozděluje dokument na několik částí (například hlavička a tělo dokumentu) a určuje, co musí a mohou obsahovat jednotlivé části. Každá taková část (i celý dokument) tvoří element, který může obsahovat další (vnořené) elementy. Elementy spolu s tagy, atributy a znakovými entitami patři mezi základní stavební kameny jazyka HTML.

2.2.1 Element

Elementem se rozumí taková část dokumentu, která se skládá z počátečního tagu, obsahu elementu a ukončovacího tagu. Výsledkem příkladu uvedeného na obr. 2.1 bude zobrazení textu (obsahu elementu) v prohlížeči kurzívou.

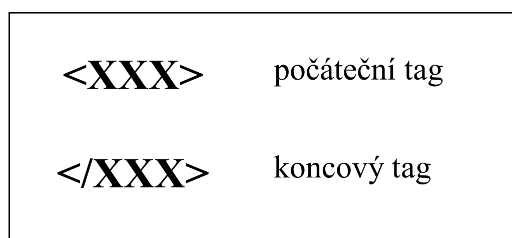


Obr.2.1. Příklad elementu.

Některé elementy nemají koncový tag. Většinou proto, že nevymezují žádný text nebo jiný prvek. Mají platnost pouze v místě svého uvedení (například vložení obrázku, přechod na nový řádek apod.). U některých elementů není uvedení koncového tagu povinné (například označení konce odstavce před začátkem nového).

2.2.2 Tag

Všechny tagy začínají symbolem ‚<‘ a končí symbolem ‚>‘. Zapisují se ve tvaru podle obr. 2.2. Za ***XXX*** dosadíme klíčové slovo příslušného elementu. U těch elementů, které nemají koncový tag, tento tvar neexistuje.



Obr.2.2. Tvar počátečního a koncového tagu.

Tagy vymezují jednotlivé elementy a určují způsob jejich úprav. Podle významu se rozdělují do několika skupin:

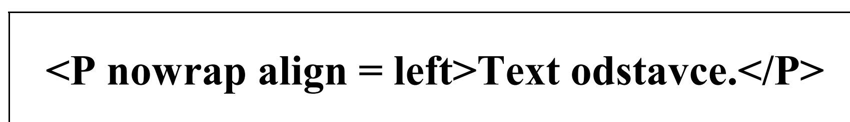
- strukturování dokumentu,
- formátování a úprava textu,
- tvorba tabulek,
- tvorba seznamů,
- tvorba odkazů,
- vkládání ovládacích prvků,
- vkládání multimediálních prvků,
- komentáře.

Pokud potřebujeme nějak upřesnit význam elementu, můžeme použít atributy. Uvádějí se vždy za klíčovým slovem počátečního tagu a lze jich u jednoho tagu použít více.

2.2.3 Atribut

Každý atribut má jméno. Některé atributy se nastaví pouze svým uvedením, jiným musíme přiřadit hodnotu. Pokud přiřazovaná hodnota obsahuje pouze písmena (velká i malá), číslice, pomlčku a tečku, pak nemusíme hodnotu přiřazovanou atributu uzavírat do uvozovek. V opačném případě musíme hodnotu atributu uzavřít do uvozovek nebo do apostrofů.

Příklad elementu začínajícím tagem s atributy je na obr. 2.3 (nový odstavec se zakázaným zalamováním řádků a s textem zarovnaným vlevo). V počátečním tagu jsou uvedeny atributy ***nowrap*** a ***align***, atributu ***align*** přiřazujeme hodnotu ***left***.



Obr.2.3. Příklad elementu s počátečním tagem s atributy.

2.2.4 Znakové entity

Některé symboly (například „<“ a „>“) mají v jazyce HTML speciální význam. Pokud chceme tyto symboly vložit do našeho dokumentu, musíme použít znakové entity (dále jen entity). Tyto entity mají svoje názvy odvozené od anglických a zapisují se ve tvaru podle obr. 2.4(a).

&název_entity; &#číselný_kód_entity;

(a) (b)

Obr.2.4. Tvar zápisu znakových entit (a) pomocí jejich jména, (b) pomocí kódu znakové sady Unicode.

Postupným rozšiřováním nyní můžeme pomocí entit zapsat všechny znaky ze znakové sady Unicode, ve tvaru podle obr. 2.4(b). Symbol **‚<‘** lze zapsat nejenom ve tvaru: ***<***, ale také ve tvaru: ***<***.

2.3 Hypertextový odkaz

Každý dokument, obrázek, program apod. (dále jen zdroje) dostupný prostřednictvím internetu má svojí adresu, která jednoznačně určuje jeho umístění. Adresy jsou kódované pomocí URI (Universal Resource Identifier [2] je obecnější schéma zahrnující i URL – Uniform Resource Locators) a do jisté míry připomínají jméno souboru. Většinou obsahují ještě další informace, například určení typu síťového protokolu (HTTP, FTP atd.). Typicky se URI skládá ze tří částí:

1. *název schématu* – jednak se jím určuje typ síťového protokolu, neboli jakým mechanismem se budou požadovaná data přenášet, ale také, jak se budou vytvářet a z čeho se budou skládat další části URI,
2. *adresa počítače* – adresa hostitelského počítače, na kterém jsou požadovaná data umístěna,
3. *cesta k datům* – poslední část obsahuje cestu k těmto datům v adresářové struktuře hostitelského počítače.

http://dce.felk.cvut.cz/orr/index2.html

název schématu adresa počítače cesta k datům

Obr.2.5. Příklad URI složené ze tří částí.

Pokud URI obsahuje alespoň první dvě části, pak nezáleží na umístění odkazu a jedná se o absolutní tvar. Takto zapsaná URI bude vždy odkazovat na stejnou adresu, ať jí uvedeme v jakémkoliv dokumentu. Příklad odkazu složeného ze všech tří části je na obr. 2.5.

Je-li uvedena pouze třetí část, jde o relativní tvar odkazu. Tento tvar se vždy převádí na absolutní a jako výchozí (nebo také základní) se pro tento převod použije adresa dokumentu, ve kterém se tento odkaz nachází. Z tohoto důvodu u odkazu v relativním tvaru záleží na jeho umístění, protože se k převodu používá adresa dokumentu, ve kterém je odkaz uveden. Příklady převodů jednoho odkazu v relativním tvaru jsou uvedeny v tab. 1.

Tab.1. Příklady převodů adresy v relativním tvaru na absolutní podle místa uvedení.

Adresa dokumentu (dokument obsahuje odkaz: *../naz/main.htm*)	**Převedená adresa odkazu**
http://dce.felk.cvut.cz/orr/index2.html	*http://dce.felk.cvut.cz/naz/main.htm*
http://www.feld.cvut.cz/index.html	*http://www.feld.cvut.cz/naz/main.htm*
http://www.seznam.cz/lide/adresy/a.html	*http://www.seznam.cz/lide/naz/main.htm*

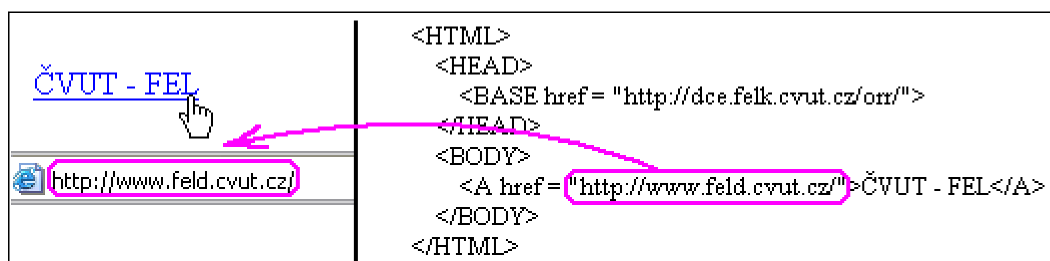
Jak relativní tak absolutní tvar má své opodstatnění. Použití relativního tvaru má výhodu, pokud jde o odkaz na místní zdroj (například na obrázky ve stejném adresáři). Když celý obsah adresáře přesuneme jinam, odkazy budou stále funkční. Absolutní tvar odkazuje vždy na danou adresu a využívá se jich u odkazů na vzdálené zdroje.

Vzhledem k tomu, že se budou zpracovávat soubory s různým umístěním, které nemusí být vždy známé, budeme upřednostňovat absolutní tvar odkazů. Na tento tvar se budeme snažit (pokud to půjde) převádět i relativní tvary.

2.3.1 Vytváření odkazu

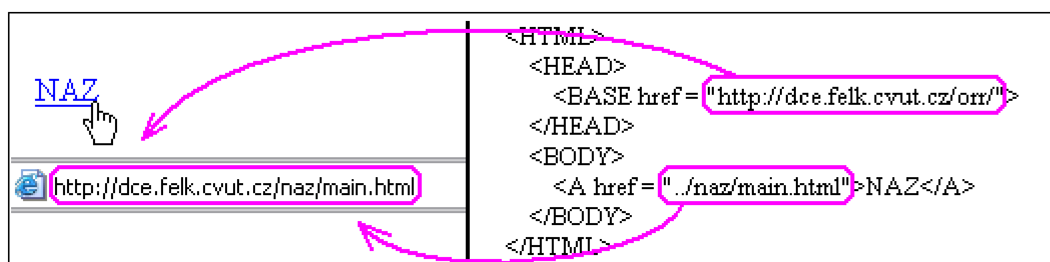
Odkaz lze vytvořit pouze pomocí elementu ***A***. Případně můžeme pomocí elementu ***BASE*** určit základní adresu pro celý dokument. Oba elementy obsahují několik atributů, které ovlivňují chování nebo vzhled odkazu, ale pouze atributem ***href*** lze přiřadit adresu odkazu.

Podle normy se element *BASE* skládá pouze z jednoho tagu. Může být v dokumentu maximálně jeden a musí být umístěn v hlavičce dokumentu. Adresa přiřazovaná atributu *href* musí být v absolutním tvaru. Pokud uvedeme tento element v dokumentu, ovlivňuje pouze odkazy v relativním tvaru (jak vidíme na obr. 2.6 a na obr. 2.7) a jeho adresa se používá pro jejich převod na absolutní tvar.



Obr.2.6. Ukázka zápisu odkazu v absolutním tvaru (v prohlížeči i ve zdrojovém kódu).

Elementy *A* se mohou vyskytovat pouze v těle dokumentu a jejich počet není nijak omezen. Jediné omezení dané normou zakazuje, aby element *A* obsahoval jiný element *A* (a to jak vnořený, tak zkřížený). Chceme-li, aby byl text v prohlížeči vyznačen jako odkaz, musíme přiřadit atributu *href* adresu. Adresa může být v absolutním nebo relativním tvaru. Pokud



Obr.2.7. Ukázka zápisu odkazu v relativním tvaru (ve zdrojovém kódu) a převodu na absolutní tvar podle nastavené základní adresy (v prohlížeči).

tento atribut element *A* neobsahuje, není v prohlížeči vyznačen jako odkaz. Textem odkazu se stává celý obsah elementu. Na obr. 2.6 je uveden příklad absolutního odkazu (jeho zobrazení v prohlížeči i původní zdrojový kód). Příklad relativního odkazu je uveden na obr. 2.7. Tento odkaz se převede na absolutní tvar podle základní adresy nastavené elementem *BASE*.

2.4 HTML v praxi

Při testování, co všechno můžeme vytvořit v HTML, jsem dospěl k závěru, že není nikdo nucen k tomu, aby dodržel normu jazyka. Můžeme vytvořit HTML dokument se syntakticky nesprávným zdrojovým kódem a prohlížeče ho akceptují. Pomocí speciálních programů – validátorů (například [1]), lze kontrolovat jejich syntaktickou správnost a program sám určí o jak závažnou chybu jde. Některá nedodržení normy jsou pouze kosmetická a na výsledný dokument mají malý vliv. Bohužel jsou tolerovány i takové věci, které přímo ovlivňují oblast zájmu této práce - odkazy a text dokumentu.

Nedodržení normy, která ovlivňují tuto práci, lze shrnout do čtyř bodů (testováno v prohlížečích Microsoft Internet Explorer 5.0 a Netscape Communicator 4.2):

1. *Nesprávné uvedení elementu BASE*. Element *BASE* může být podle normy uveden v dokumentu maximálně jednou v jeho hlavičce. Realita je ovšem jiná. Uvedeme-li tento element kdekoliv v dokumentu, prohlížeč ho akceptuje a použije pro převod všech relativních adres, které jsou za ním uvedeny.
2. *Neuvádění koncových tagů.* U některých elementů neuvedení koncového tagu dovoluje norma, často se používá tento postup i u elementů, u kterých to dovoleno není.
3. *Křížení elementů.* Tuto možnost norma vůbec nepřipouští. V praxi jde o celkem běžný jev, můžeme například najít zápisy typu:

   <A href = „adresa odkazu“><B>Text odkazu.</A></B>,

   místo správného zápisu:

   <A href = „adresa odkazu“><B>Text odkazu.</B></A>.

4. *Neukončování znakových entit symbolem ‚;‘*. Entity jsou podle normy ukončeny tímto symbolem. Za ukončení entit se v testovaných prohlížečích považuje i mezera, Microsoft Internet Explorer 5.0 dokonce rozpoznává entity bez jakéhokoliv ukončení.

Všechny tyto odlišnosti od normy mohou mít za následek jiné zobrazení dokumentu v různých prohlížečích. Při rozhodování, jakou cestou směřovat aplikaci, byla vybrána cesta akceptování dokumentů neodpovídající normě, aby jejich zpracování nekončila pouze zprávou „*o nedodržení normy jazyka HTML*“.

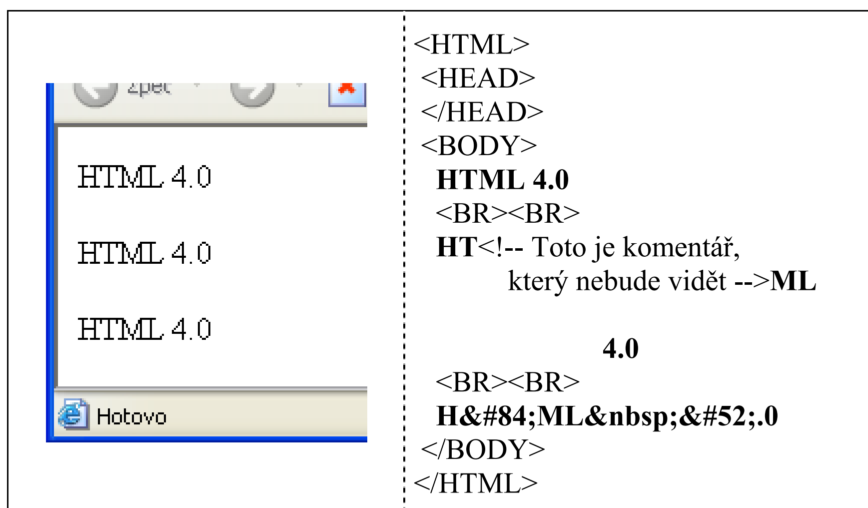
Dalšími experimenty[1] bylo zjištěno, že text ve tvaru:

<X… >

je považován za tag (pro zpětnou kompatibilitu prohlížečů s novějšími normami). Všechny tři části jsou povinné. Za ***X*** můžeme dosadit všechna písmena (velká i malá) z ASCII tabulky a symboly ‚?‘, ‚!‘ a ‚/‘.

---

[1] Experimenty obsahovaly testy (v aplikacích Microsoft Internet Explorer 5.0 a Netscape Communicator 4.2) jednotlivých možností zápisu tagu: bez prvního znaku, bez posledního znaku a postupným dosazování všech znaků z ASCII tabulky za klíčové slovo elementu.



Obr.2.8. Možnosti zápisu stejného textu. Ukázka rozdílu mezi výsledným zobrazením v prohlížeči a zdrojovým kódem dokumentu.

Poměrně velký rozdíl, tentokrát v souladu s normou, může nastav mezi zdrojovým kódem a výsledkem zobrazeným v prohlížeči. Například text „*HTML 4.0*“ lze zapsat mnoha způsoby a všechny povedou k témuž cíly. V prohlížeči budou zobrazeny naprosto shodně, jak vidíme v levé části na obr. 2.8. Elementy *
* ve zdrojovém kódu značí odřádkování a stejně jako zvýraznění kódu tvořícího text, byly použity pouze pro zvětšení přehlednosti.

Z uvedeného příkladu tří možností zápisu jednoho textu na obr. 2.8 vyplývá, že nelze úlohu řešit metodou přímého hledání jednoho řetězce v druhém. Jedním z řešených problémů v dalších částech této práce bude oddělení textu dokumentu od fragmentů jazyka HTML. Následným rozborem úloh zjistíme, jakým způsobem bude nutné toto oddělení provést a jaké informace zachovat.

3. Rozbor zadání

Rozšířené zadání obsahuje několik bodů, které doplňují informaci o tom, jak by aplikace měla reagovat v určitých situacích. Tyto body nemají vliv na celkovou koncepci, neboť pouze upřesňují chování za určitých situací.

Základní úkoly této práce lze rozdělit do tří bodů, které jsou do jisté míry na sobě nezávislé:

- nalezení odkazů v dokumentu a uložení do databáze (dále jen ukládání odkazů),
- aktualizace adres odkazů v dokumentu podle dat z databáze (dále jen aktualizace odkazů),
- nalezení textu v dokumentu a jeho označení jako odkaz podle dat v databázi (dále jen vytváření odkazů).

Tyto úlohy můžeme pro každý dokument různě kombinovat, od použití pouze jedné z nich, až po aplikaci všech třech.

V dalších částech této kapitoly provedeme rozbor problému jednotlivých úloh a budeme se snažit určit jaké postupy a principy budou použity. Ujasníme si, jaké informace budou jednotlivé úlohy potřebovat a kde je získají. V neposlední řadě budeme hledat společné dílčí postupy a určíme podmínky pro vytvoření jednotného přístupu k dokumentům.

3.1 Vytváření odkazů

Analýzu začneme od nejobtížnější úlohy. Tento zdánlivě nelogický krok má jednoduché vysvětlení: vytváření odkazů tvoří stěžejní část této práce a více či méně ovlivňuje ostatní úkoly. Pokud bychom dosáhli skvělých výsledků při ukládání odkazů, ale tato data dále neuměli použít, jakého efektu bychom dosáhli?

K vytváření odkazů se budou používat data uložená v databázi podle jednoduchého principu. K textu odkazu z databáze se nalezne příslušná část ve zpracovávaném dokumentu, která bude tvořit obsah elementu odkazu. V počátečním tagu elementu *A* se atributu *href* přiřadí odpovídající adresa, která bude také uložena v databázi.

Jednoduchost principu neukazuje nic o složitosti vlastního provedení. V následujícím textu podrobně rozebereme celou úlohu vytváření odkazů. Nejprve určíme vstupy a výstupy této úlohy, následně provedeme analýzu možných problémů a navrhneme jejich řešení. Nakonec shrneme získané poznatky a navržená řešení.

3.1.1 Vstupy a výstupy

Nejprve definujme vstupy tohoto procesu. Zpracovávaný obecný HTML dokument se stává prvním vstupem. V tomto souboru budeme hledat a vytvářet nové odkazy. Druhý vstup tvoří data z databáze (texty a adresy odkazů).

Výstupem procesu je HTML dokument, který bude obsahovat nově vytvořené odkazy. Tento nový dokument musí být vytvořený jako původní s nově vyznačenými elementy odkazů, aby byla splněna podmínka zachovaní původní struktury dokumentu. Pokud nebudou v původním dokumentu nalezeny žádné nové odkazy, vytvářený dokument bude stejný jako původní. V tomto případě jej není nutné vytvářet.

3.1.2 Specifikace problému

Před samotnou analýzou úlohy vytváření odkazů ještě jedna poznámka. Ať už použijeme jakýkoliv postup při hledání textů, nevyhneme se porovnání řetězců. Obecný HTML dokument tedy není vhodný pro přímé zpracování. Z tohoto důvodu musíme vložit na začátek

procesu předzpracování, které dokument převede do unifikovaného tvaru. Zbylé dvě úlohy (aktualizace a ukládání odkazů) také obsahují práci s textem, a proto bude předzpracování předcházet všechny úlohy. Jednotlivé úlohy budou mít na předzpracování vlastní požadavky, které vyplynou z jejich analýzy. Nyní se spokojíme s tvrzením, že umíme upravit soubor na vstupu tak, aby bylo možné jeho přímé zpracování. Až provedeme analýzu všech tří úloh, navrhneme v kapitole 4 předzpracování HTML dokumentů.

Všechny úkoly, které musí řešit proces vytváření odkazů, jsou dány rozšířeným zadáním. Dají se shrnout do těchto bodů:

- *zachovat původní strukturu dokumentu* – nový dokument vytvořit jako původní s nově vyznačenými odkazy,
- *estetické hledisko* – nevytvářet stejný odkaz vícekrát ve větším celku, jako je například odstavec nebo kapitola,
- *hledat texty odkazů v různých tvarech* – umět najít text odkazu i v tom případě, že je uveden v databázi a v dokumentu v různých tvarech.

Nyní si tyto úkoly postupně rozebereme a budeme se snažit najít řeší problémů s nimi spojených.

3.1.3 Zachování původní struktury

Jak jsme již zmínili v části 3.1.1, abychom mohli zachovat původní strukturu, musí být nový dokument (výstup procesu) vytvářen z původního. Pokud nalezneme nějaký text odkazu v unifikovaném textu, musíme znát doplňující informaci o umístění v původním dokumentu (přesněji ve zdrojovém kódu původního dokumentu).

Řešení tohoto problému bude součástí unifikace, kde se pro každou část textu vygeneruje struktura obsahující: *část textu - počátek v původním dokumentu - konec v původním dokumentu*. Při vytváření nového dokumentu použijeme tyto informace na získání původního zdrojového kódu. Tím dostaneme text dokumentu v podobě, která nám umožní porovnávat řetězce i s vazbou na původní dokument a zachovat tak jeho strukturu.

3.1.4 Estetické hledisko

Z estetického hlediska je omezeno vytváření odkazů ve větším celku. To znamená, že odkaz, který se váže ke stejnému textu, se v takovém celku vyznačí pouze jednou a to při jeho prvním výskytu. Z kapitoly 2 víme, že všechny způsoby formátování dokumentu (tedy i vytváření jednotlivých celků) se provádí pomocí elementů. Nejprve musíme určit, jak velké celky budeme uvažovat, a pak zjistit pomocí kterých elementů se vytvářejí. Tyto elementy bude nutné při předzpracování zachovat nebo jinak označit.

Vizuální rozdělení textu HTML dokumentu na samostatné celky lze provést několika způsoby. Podle rozřazení elementů do skupin v části 2.2.2 budeme uvažovat elementy z těchto tří skupin:

1. *Tvorba seznamu*. Seznam jako celek považujeme za oddělený od ostatního textu. Jazyk HTML zná tři druhy seznamů, nečíslovaný seznam vytvořený prostřednictvím elementu ***UL*** (nebo starším elementem ***DIR***), číslovaný seznam vytvořený elementem ***OL*** a seznam pojmů tvořený elementem ***DL***.
2. *Tvorba tabulek*. Tabulka sama o sobě tvoří část textu oddělenou od okolí a vytvoří se pomocí elementu ***TABLE***. Oproti seznamu většinou neprocházíme celou tabulku, ale vyhledáme příslušný řádek případně sloupec. Z tohoto důvodu bude pro nás každý řádek (případně každá buňka) tvořit samostatnou část textu. Řádek vymezuje element ***TR*** a jednotlivé buňky lze vymezit buď elementem ***TH*** nebo elementem ***TD***.

3. *Formátování a úpravu textu*. Formátování a úprava textu nabízí širší paletu možností. Jako první zmíníme nadpisy různé velikosti, které vymezuje element ***H1*** až ***H6***. Další formátování textu provádíme na odstavce, které lze vytvořit pomocí elementu ***P*** nebo elementu ***DIV***. Vizuálně lze text oddělit také linkou vytvořenou elementem ***HR***. Poslední možnost jak oddělit text je vynucené odřádkování elementem ***BR***.

Těchto 18 elementů se používá na logické členění a oddělení textu HTML dokumentu. Nám stačí umět tyto elementy (případně příslušné tagy) rozpoznat a budeme tak mít informaci o členění dokumentu.

3.1.5 Hledání textů odkazů

Než se začneme zabývat hledáním textů odkazů, musíme se zamyslet nad tím, na jaké texty se zaměříme. Někdy se jako text odkazu používají například texty: „*klikněte zde*“, „*pošlete mi email sem*“, „*dostupné na našem FTP* “ a další. Tento typ textu odkazu není doporučován [7]. Také není vždy v lidských silách, natož v možnostech výpočetní techniky rozeznat, která adresa by k textu typu: „*klikněte sem*“ patřila. Z tohoto důvodu nebudeme tyto texty dále uvažovat.

Při hledání textu odkazu musíme počítat s tím, že se hledaný text může vyskytovat v dokumentu v jiném tvaru (zkráceném nebo upraveném) než je uložen v databázi. Například v databázi budou uloženy texty:

„ing. Jan Novák“ a ***„Dynamické systémy“***

a v dokumentu budou texty uvedeny v tomto tvaru:

„ing. Novák, J.“ a ***„Dyn. systémy“***.

Z příkladu vidíme, že u textů může docházet k různým úpravám. První text se změnil na zcela jiný tvar a u druhého došlo pouze ke zkrácení jedné části.

Budeme-li chtít sami určit, jestli se může jednat o stejné texty, použijeme více svých znalostí. Umíme rozlišovat o jaký text jde. Pokud bude na knize uvedeno:

„ing. Jan Novák – Dynamické systémy“,

je jasné, že jde o dvojici jméno autora – název knihy. Takto rozdělujeme texty do kategorií. Ke každému textu pak podle příslušných jazykových pravidel dokážeme vytvořit přípustné tvary. Případně umíme porovnat upravený a původní tvar, a proto musíme zajistit, aby i počítač měl podobné informace.

Půjde nám o to, řadit texty se stejnou možností úprav do jedné skupiny – *kategorizovat texty*. Pro každou skupinu vytvoříme snadněji aparát, který umí upravené tvary kontrolovat nebo vytvářet.

3.1.6 Kategorizace textů

Rozdělení textů do skupin budeme tedy provádět podle možností jejich úprav. Tyto úpravy lze shrnout do tří bodů:

- zkrácení jedné nebo více částí,
- změna pořadí jednotlivých částí,
- vynechání jedné nebo více částí.

Celkem 8 kombinací neboli 8 skupin. Další možnosti, které nejsou v seznamu, jsou změna velikosti písmen a změna v diakritice. Tento problém bude řešen v jiné části práce, protože ani jedna z nich nemění tvar textu, ale pouze jednotlivá písmena.

Podívejme se na problém z druhé strany. Jaké texty odkazů se mohou v dokumentech vyskytovat? Neboli, je 8 skupin opravdu reálných?

Všechny námi uvažované odkazy mají konkrétního adresáta. Ať už se jedná o dokument, aplikaci, úřad, firmu, soukromou osobu apod., zápisy vždy odpovídají konkrétnímu jménu, názvu nebo označení. Podle [5] jde o vlastní jména a názvy živých bytostí a neživých věcí, ustálenou zkratku, značku nebo zkratkové slovo. Pokud budeme zkoumat jejich vlastnosti, zjistíme, že se dají rozdělit do tří skupin. Použitá označení skupin jsou pouze formální.

První skupinou jsou **zkratkové názvy** (dále jen zkr. názvy). Patří sem značka, ustálená zkratka a zkratkové slovo (například *ČVUT*, *Čedok* atd.). Mohou být tvořeny jednou nebo více částmi, ale nelze je dále upravovat. Zkrácení, změna pořadí nebo vynechání některé z částí vytvoří jiný zkr. název. Zkrácením *ČVUT* na *ČT* dostaneme dva naprosto rozdílné zkr. názvy.

Druhou skupinou jsou **názvy**. Do této skupiny lze zařadit všechna vlastní jména a názvy neživých věcí (například *České vysoké učení technické*, *Kutná Hora* atd.). Všechny tyto názvy mají právě jeden oficiální název a jedinou možností pro jejich úpravu je zkrácení jedné nebo více částí. Lze napsat:

České vys. uč. tech.*,*

ale není možné napsat:

České učení vysoké technické nebo jenom ***České učení technické****.*

Někdy se označuje jedna věc více názvy. Například *Česká republika* a *Česko, Univerzita Karlova* a *Karlova univerzita*, *Pražský hrad* a *Hrad* atd. Vždy pouze první z uvedených názvů je oficiální. Tento jev se nedá nijak zobecnit, protože se jedná o singulární případy.

Poslední třetí skupinou jsou **jména osob**. Patří sem vlastní jména a názvy živých bytostí (například *ing. Jan Novák*, *Sněhurka* atd.). Obecně můžeme jména osob upravovat všemi způsoby, ale jiná pravidla platí pro příjmení, jiná pro křestní jméno (případně jména) a další části (tituly apod.). Z tohoto důvodu musíme rozdělit do skupin i jednotlivá slova.

3.1.7 Kategorizace slov

Texty lze rozdělovat do skupin bez ohledu na jejich umístění, protože vždy pracujeme pouze s jedním textem. Slova takto rozdělovat nemůžeme, protože jeden text může být (a často bude) složen z více slov, a proto záleží na jejich pořadí.

Využijeme toho, že se text skládá z jedné nebo více souvislých částí, které obsahují slova se stejnými vlastnostmi (například křestní jména a příjmení). Tyto slova sloučíme do skupin, na které budeme pohlížet jako na celek. Pro větší přehlednost vytvoříme skupiny pro každý typ textu zvlášť. Jako jejich názvy použijeme obecně známá označení.

Slučování slov do skupin je nutné pouze u jmen osob. Názvy i zkr. názvy se skládají ze slov, která tvoří vždy jednu skupinu. Typ textu tedy přímo určuje skupinu. Skupinu pro název označíme **části názvu** a skupinu pro zkr. název označíme **části zkratkového názvu** (dále jen části zkr. názvu). Vlastnosti jsou také dány typem textu. U částí názvu můžeme pouze zkracovat a u částí zkr. názvu nejsou možné žádné úpravy.

Pro označení skupin slov jména osoby použijeme označení – titul, křestní jméno a příjmení. Protože tituly mohou být jak před jménem tak za jménem, bude konečné rozdělení slov jména osoby na **tituly před jménem**, **křestní jména**, **příjmení** a **tituly za jménem**. Vlastnosti jednotlivých skupin jsou uvedeny v tab. 2.

Vlastnost se vždy váže ke skupině jako celku. Pokud mluvíme o změně pořadí, myslíme tím změnu pořadí skupin. Pořadí slov pro daný text je ve všech skupinách neměnné. Stejně tak vynechání se týká celé skupiny. Nelze vynechat jen některá slova, ale vždy celou skupinu. Pouze zkrácení se nemusí týkat všech slov ve skupině najednou.

Tab.2. Vlastnosti skupin slov.

Typ textu	Typ slova	Vlastnost		
		Zkrácení	**Vynechání**	**Změna pořadí**
Název	*Část názvu*	ANO	NE	NE
Zkr. název	*Část zkr. názvu*	NE	NE	NE
Jméno osoby	*Tituly před jménem*	NE	ANO	NE
	Křestní jména	ANO	ANO	ANO
	Příjmení	NE	NE	ANO
	Tituly za jménem	NE	ANO	NE

Sloučením slov do skupin ztrácíme informaci o původním pořadí. Aby bylo možné na základě znalosti skupin slov vytvořit opět původní nebo pozměněný tvar textu, potřebujeme znát tuto informaci. Pořadí slov ve skupině se nemění, ale pořadí skupin se měnit může, a proto uložíme tuto informaci v rámci každé skupiny zvlášť.

Když umíme rozlišit typ textu a jednotlivých slov, můžeme přistoupit k řešení problému týkající se návrhu aparátu pro generování jednotlivých přípustných tvarů textů.

3.1.8 Generování tvarů

Nyní víme jaká data budou ukládána do databáze. Vedle adresy to bude text odkazu a jeho typ a typ jednotlivých slov s pořadím ve skupině . Tyto informace se jeví jako dostačující pro vytváření odkazů z textů v různých tvarech.

Chceme-li zjistit, zda text uvedený v dokumentu tvoří přípustný tvar některého z textů uložených v databázi, potřebujeme je umět porovnat. Jsou-li oba tvary stejné, je porovnání triviální. V opačném případě musíme z textu z databáze, u kterého máme všechny potřebné informace (typ textu, typ a pořadí slov) uložené, umět vygenerovat přípustné tvary a následně porovnat s nalezeným textem.

Zamysleme se nyní nad tím, jak lze také na rozdělení textů a slov do skupin pohlížet, pokud si odmyslíme možnost zkrácení (tvorbu zkratek). Typ textu jednoznačně určuje z jakých skupin slov se text může skládat. Skupiny tvoří slova, jejichž pořadí se nemění, což nám umožňuje označit skupiny různými symboly. Dostaneme tak pro každý typ textu konečnou abecedu symbolů, kterou lze zpracovat jednoduchým automatem. Podaří-li se nám pro něj navrhnout gramatiku, dostaneme nástroj pro tvorbu přípustných tvarů jednotlivých textů.

Gramatiky by tak řešily problém generování tvarů. Jak se ovšem vypořádat se zkratkami? Nejdříve je nutné vědět podle jakých pravidel se zkratky vytvářejí. Následně určíme jak zjišťovat, že právě tato zkratka patří k tomuto slovu.

3.1.9 Zkracování a porovnávání slov

Podle [5] obvykle písemně zkracujeme tak, že vypisujeme pouze první písmeno nebo počáteční skupinu písmen, někdy i slabiku nebo i více slabik, vždy však tak, aby končila zkratka souhláskou. Za zkratkou píšeme tečku. Druhou možnost používáme méně často. U zkracovaných slov vypisujeme první a poslední písmeno nebo písmena (většinou ohýbací koncovku). Za těmito zkratkami nepíšeme tečku. Existují ovšem výjimky, které nelze skloňovat a píšeme za nimi tečku.

První způsob se také týká křestních jmen. Bohužel tyto pravidla jsou nejednoznačná. Zkratka *J.* může být od křestních jmen *Jan*, *Jana*, *Josef*, *Jiří* a další. Na druhou stranu křestní jméno *Josef* lze zkrátit *J.* nebo *Jos.*, *František* lze zkrátit *F.*, *Fr.*, *Frant.* Také podmínka, aby zkratka končila vždy souhláskou není úplně striktně dodržována, protože například křestní jméno *Alois* lze zkrátit na *Al.*, ale také jen na *A.*

Druhý popsaný způsob zkracování je ještě více vágní. Dokonce umožňuje, že některé zkratky nemusí být ukončeny tečkou. Patří sem například *fa*, *fy*, *fě* jako zkratky slov *firma*, *firmy*, *firmě* atd.

Zkracování nelze zobecnit, protože pro každé slovo mohou platit jiná pravidla. Jak tedy alespoň přibližně určit, jestli tato zkratka je od tohoto slova? Když se podíváme na uvedené příklady slov a jejich zkratek, každá dvojice má společné dvě věci. První je počáteční písmeno, obě slova (původní i zkrácené) mají stejné počáteční písmeno. Druhou společnou věcí je to, že zkratku výhradně tvoří písmena z původního slova a to v nezměněném pořadí. Podle těchto dvou vlastností budeme zkratku definovat.

Zkratkou se tedy stává slovo, které: 1.začíná stejným písmenem jako původní slovo a 2. obsahuje pouze písmena z původního slova a to v nezměněném pořadí.

Toto zjednodušení má několik výhod. Lze použít gramatiky pro generování přípustných tvarů textů. Zkratky není nutno složitě analyzovat a testovat. Když se bude porovnávací funkce držet výše uvedených pravidel, na všechna porovnání stačí pouze jedna. Porovnání může zahrnovat i inverzní způsob zkrácení, kdy zkrácené slovo není v dokumentu, ale uložené v databázi a dokument obsahuje slovo v nezkráceném tvaru. Například v dokumentu bude uvedeno „*František Novák*“ a v databázi bude uloženo jméno „*Fr. Novák*“ apod. Pravidla zahrnují i triviální případ dvou stejných slov, jako například *Novák – Novák*. Obě slova začínají stejným písmenem a obsahují stejná písmena v nezměněném pořadí. Na všechna porovnání pak stačí jediná funkce a my toho využijeme.

Na druhou stranu i něco ztrácíme. Zjednodušující pravidla postihují i velký počet tzv. „planých poplachů“. To znamená slov, která nejsou zkratky, ale podle pravidel tak jsou označeny. Na první pohled to může vypadat jako velký problém, ale úkolem porovnání bylo pouze zjistit, jestli se jedná o stejná slova nebo o zkratku daného slova. Každá zkratka bude následně testována v kontextu celého textu odkazu.

Dalším problémem společného porovnání je, že některá slova, tak jak jsme je rozdělili v kapitole 3.1.7, nemohou být zkrácena. Ke zkrácení tedy nemohlo dojít ani inverzním způsobem. Výstupem porovnávací funkce proto musí být ještě informace o typu podobnosti. Typ podobnosti dvou slov rozlišíme na **různá**, **stejná**, **zkratka** a **inverzní zkratka**. Příklady porovnání slov i s uvedeným typem podobnosti jsou uvedeny v tab. 3.

Tab.3. Příklady porovnání dvou slov v databázi a v textu.

Slovo z textu odkazu v databázi	**Slovo z textu odkazu v dokumentu**	**Typ podobnosti**
Novák	Komenský	*různá*
Novák	Novák	*stejná*
Jiří	J.	*zkratka*
J.	Jiří	*inverzní zkratka*

Tím, že umožníme tvorbu zkratek, se nevyhneme ani té situaci, kdy jednomu textu lze přiřadit dvě nebo více adres. Mohou nastat dva případy:

1. *Jednomu textu můžeme přiřadit více adres*. Například k názvu instituce můžeme přiřadit emailovou adresu a adresu jejich internetových stránek.
2. *Dva různé texty mají stejný přípustný tvar*. Například *„Jan Novák"* a *„Jiří Novák"* mají oba stejný tvar *„J.Novák"*.

V druhém případě jde o podobný případ jako u textu typu „*klikněte zde*". Je zde nutný dotaz na uživatele, který vybere odpovídající adresu. U první možnosti půjdeme stejnou cestou dotazování z důvodů, které nejsou na první pohled zřejmé.

Pokud bychom chtěli, aby počítač tyto případy rozhodoval, musíme určit podle jakého klíče. Možností máme mnoho, například můžeme přiřadit váhy jednotlivým adresám, můžeme ke každé adrese do databáze ukládat kolikrát již byla použita atd. Všechny způsoby mají jedno společné, pokud upřednostníme jednu adresu před druhou, bude ji počítač používat stále a uživatel se ani nedozví, že byla jiná možnost. Budeme-li chtít tento stav změnit a upřednostnit jinou adresu, musíme změnit například hodnoty jednotlivých vah, a to znamená opět zásah uživatele.

Vždy, když bude více možností, bude uživatel dotazován. Vytváření odkazů tedy nebude plně automatické. Použití rozhodovacího klíče není ani v tomto případě bez užitku. S jeho pomocí můžeme řadit jednotlivé adresy od těch, které budou pravděpodobně použity až po ty nejméně pravděpodobné.

3.1.10 Shrnutí

Rozborem úlohy vytváření odkazů jsme zjistili, které úkoly bylo nutno řešit. Nyní si tyto úkoly zopakujeme se stručným popisem jejich řešení.

1. *Zachování původní struktury dokumentu*. Řešení tohoto problému jsme zahrnuli jako součást předzpracování, kde se pro každou část textu vygeneruje struktura obsahující *část textu – počátek v původním dokumentu – konec v původním dokumentu*
2. *Estetické hledisko*. Nejprve jsme určili jaké celky uvažujeme (seznam, tabulka, kapitola, odstavec atd.) a jakými elementy je možné tyto celky v HTML dokumentu vytvořit. Řešení jsme opět zahrnuli jako součást předzpracování, kde tyto elementy ponecháme nebo jinak označíme.
3. *Hledaní textů odkazů v různých tvarech*. Nejprve jsme si ukázali, že u textů může docházet k různým úpravám. Podle těchto úprav jsme texty rozdělili do skupin. Dále se ukázalo nutné rozdělit do skupin i jednotlivá slova textů. Z důvodů uvedených v částech 3.1.7, 3.1.8 a 3.1.9 můžeme navrhnout pro vytváření přípustných tvarů gramatiky, a tak dokážeme porovnat texty v různých tvarech.

Vyřešili jsme všechny úkoly spojené s vytvářením odkazů. Návrh gramatik pro jednotlivé typy textů bude proveden v kapitole 5.

3.2 Ukládání odkazů

Ukládání odkazů je do jisté míry inverzní proces k vytváření odkazů. Data se nezískávají z databáze, aby se použili na hledání a úpravu v dokumentu, ale naopak. Odkazy hledáme v dokumentu a ukládáme do databáze.

Princip je opět jednoduchý. Hledáme všechny aktivní elementy *A*, neboli ty, které tvoří odkaz. Z jejich počátečního tagu získáme adresu a obsah elementu tvoří text odkazu. Adresy

v relativním tvaru převedeme do absolutního, pokud byl před odkazem uveden element *BASE* nebo uživatel nastavil základní adresu.

Některé problémy již byly zmíněny v kapitole 2. V následujícím textu podrobně rozebereme úlohu ukládání odkazů. Nejprve definujeme vstupy a výstupy této úlohy, následně analyzujeme možné problémy a navrhneme jejich řešení. Nakonec provedeme shrnutí získaných poznatků.

3.2.1 Vstupy a výstupy

Opět jako první definujeme vstup procesu ukládání odkazů. Oproti vytváření odkazů je jeho jediným vstupem HTML dokument.

Výstupem procesu budou data následně ukládaná do databáze. Zatím nevíme, jak bude databáze vypadat. Ani neuvažujeme, jestli jsou nalezená data už v databázi nebo ne. Chceme pouze, aby nám proces zpřístupnil všechny texty a adresy odkazů, které dokument obsahuje. Možný problém s konzistencí dat přenecháme databázi, kde před uložením nových adres a textů otestujeme jejich existenci v databázi.

3.2.2 Specifikace problému

Stejně jako u vytváření odkazů předchází tuto úlohu předzpracování, pracujeme tedy s předzpracovaným dokumentem. Zvýšené nároky této úlohy na předzpracování se týkají pouze odkazů. V rámci předzpracování musí být zachovány nebo jinak označeny počáteční a koncové tagy elementů *A* a počáteční tagy elementů *BASE*. Opět se spokojíme s tvrzením, že máme předzpracovaný dokument s unifikovaným textem a umíme rozlišit výše zmíněné elementy.

V části 3.1 jsme zjistili jaké informace potřebujeme při vytváření odkazů získat a uložit do databáze. Z uvedeného principu ukládání odkazů plyne, že přímo z dokumentu se v obecném případě dají získat pouze tři informace: adresa odkazu, text odkazu a pořadí slov v textu odkazu. Nelze v něm získat označení typu textu ani jednotlivých slov. Tyto informace získáme od uživatele. I proto proces ukládání odkazů nebude plně automatický.

Úkoly ukládání odkazů vycházejí ze zápisu odkazu podle normy, kdy počáteční tag obsahuje adresu a text odkazu tvoří celý obsah elementu. Nejprve je tedy nutné najít samotný element, ať už se jedná o element *A* nebo *BASE*. Dále musíme získat adresu a text odkazu. Všechno může komplikovat odkaz, který je tvořen z více elementů – **dělený odkaz**.

Úkoly lze shrnout do těchto bodů:

- *lokalizace elementů A a BASE* – nalezení těchto elementů,
- *zpracování adresy odkazu* – získání adresy s případným převodem adres v relativním tvaru na absolutní,
- *zpracování dělených odkazů* – zpracování odkazu, který je tvořen více než jedním elementem *A*,
- *získání textu odkazu* – získání adresy odkazu, a to jak z neděleného, tak z děleného odkazu.

Nyní postupně provedeme analýzu těchto úkolů a budeme se snažit najít řeší problémů s nimi spojených.

3.2.3 Lokalizace elementů A a BASE

Začneme elementem *BASE*. Tento element je tvořen pouze jedním tagem, který jsme si označili při předzpracování, a proto jeho nalezení nečiní žádný problém. Z tohoto tagu získáme základní adresu. Tato adresa se používá k případnému převodu adres v relativním tvaru, které jsou uvedeny za tímto elementem. Jak bylo uvedeno v kapitole 2, element *BASE* akceptují

prohlížeče kdekoliv v HTML dokumentu. Z těchto důvodů musíme umět zpracovat element *BASE* vícekrát a to v kontextu jeho umístění.

U elementu *A* je situace složitější. Můžeme se setkat s tím, že nejsou uváděny jeho ukončovací tagy. Například při vytváření více odkazů za sebou, se někdy neuvádí ukončovací tag a za ukončení elementu se považuje začátek dalšího elementu *A* (jeho počáteční tag). Je-li poslední element *A* bez ukončovacího tagu, za obsah elementu je považován celý zbytek dokumentu.

Za element odkazu tak budeme považovat tu část kódu, která začíná počátečním tagem elementu *A* a končí buď počátečním nebo koncovým tagem elementu *A* anebo na konci HTML dokumentu. Element odkazu nesmí obsahovat žádný jiný element odkazu ani počáteční nebo koncový tag elementu *A*.

3.2.4 Zpracování adresy odkazu

Zpracování adresy musí předcházet její získání, které je pro element *BASE* i pro element *A* stejné. Když si uvědomíme, že se adresa přiřazuje stejnému atributu – *href*, pak postačí pouze jediná funkce pro oba elementy na její získání. Této skutečnosti lze využít při samotné implementaci procesu ukládání odkazů.

Po získání adresy z elementu *BASE* není nutné její další zpracování, pouze ji musíme uložit pro další použití. Následně se tato uložená adresa použije pro převod adres z elementů *A*, které budou v relativním tvaru.

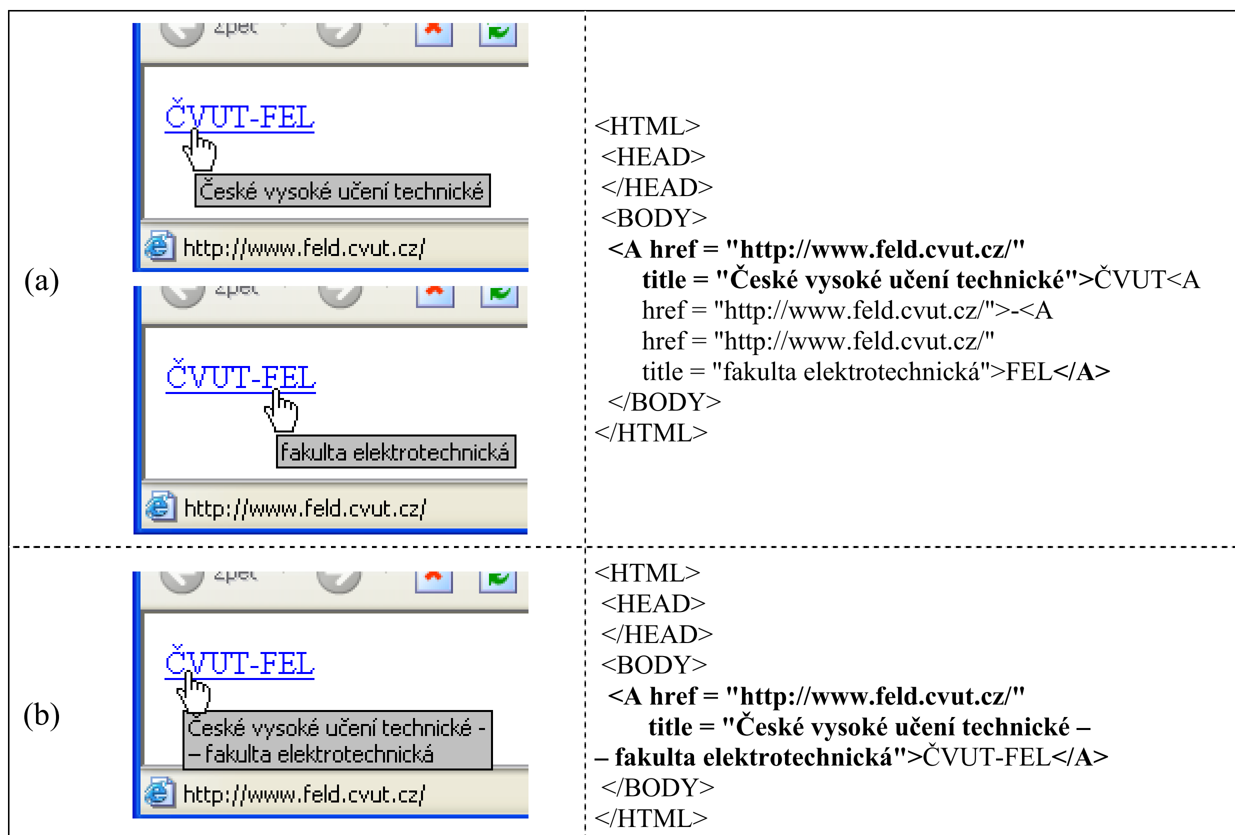
Adresy z elementů *A* chceme získat v absolutním tvaru. U adres uvedených v tomto tvaru není nutná žádná další úprava. Jiná situace nastane u adres v relativním tvaru, kdy mohou nastat dva případy. V prvním byl před odkazem uveden a zpracován element *BASE*, potom se musí adresa převést na absolutní tvar. V druhém případě nebyl před odkazem uveden tento element. Tyto odkazy nemusíme dále zpracovávat nebo umožníme uživateli, aby sám uvedl umístění (základní adresu) dokumentu a adresu opět převedeme na absolutní tvar. Samotný převod z relativního do absolutního tvaru realizujeme podle [9].

K této části se váže ještě jedna připomínka. Text dokumentu tvoří v podstatě všechen viditelný text. Odkazy jsou tedy také součástí textu dokumentu. Řekněme, že pracujeme s touto aplikací. Chceme z dokumentu nejprve uložit odkazy do databáze, a pak v dokumentu vytvořit podle databáze nové odkazy. Co se stane? Uloží se stávající odkazy a začnou se v textu dokumentu hledat nové. Tedy nalezneme i ty, které byly v originálním dokumentu již vyznačeny a možná před tím uloženy do databáze. Proces ukládání odkazů proto musí odkazy po nalezení a zpracování „vyříznout“ nebo jinak označit, aby nebyly opětovně zpracovány.

3.2.5 Zpracování děleného odkazu

Problematiku děleného odkazu můžeme zařadit do dlouhého seznamu rozdílů mezi zdrojovým HTML kódem a tím, co je vidět v prohlížeči. Jednotlivé části (tzn. i písmena) lze každou zvlášť označit elementem *A* jako odkaz na stejnou adresu. V prohlížeči se takto vytvořený odkaz jeví naprosto stejně, jako kdyby šlo pouze o jeden odkaz.

Důvod, proč takto postupovat, může být například použití efektů, které nastavíme pro každou část jinak, jak vidíme na obr. 3.1 (i s výpisem HTML kódu). Jako efekt je v příkladu použit komentář (*ToolTip*). Pokud zůstane kurzor nad odkazem, zobrazí se text (v malém rámovaném okně) přiřazený atributu ***title***. U prvního odkazu na obr. 3.1(a) rozděleného na tři části, se takto zobrazí význam jednotlivých zkratek (nad textem „*ČVUT* “ se zobrazí „*České vysoké učení technické*“, pomlčka nemá přiřazený žádny komentář a nad textem „*FEL*“ se zobrazí „*fakulta elektrotechnická*“). U druhého neděleného odkazu na obr. 3.1(b) se zobrazuje stejný komentář ("*České vysoké učení technické – fakulta elektrotechnická*") pokud zůstane kurzor kdekoliv nad odkazem. Komentář byl v tomto případě rozdělen na dva řádky.



Obr.3.1. Ukázka stejně vypadajících odkazů (s výpisem zdrojového kódu) tvořených (a) třemi částmi, (b) jednou částí. Tučně je ve zdrojovém kódu vyznačen počátek a konec odkazu.

Na první pohled vypadají oba odkazy na obr. 3.1 v prohlížeči stejně. Tvořeny jsou stejným textem, který není nijak rozdělen, a odkazují na stejnou adresu, a proto je lze považovat za totožné. Výsledkem ukládání odkazů v tomto případě musí být nalezení shodných odkazů.

Z těchto důvodů nemůže proces ukládání odkazů nahlížet na odkazy jako na osamocené elementy, ale vždy v kontextu jejich umístění. Musí nejenom zpracovávat elementy *BASE* a *A*, ale také umět porovnat dva sousední elementy *A*, jestli neodkazují na stejnou adresu. Odkaz složený z více elementů ovlivňuje i získání textu odkazu.

3.2.6 Získání textu odkazu

V části 2.3.1 bylo uvedeno, že za text odkazu se považujeme celý obsah elementu *A*. Jak jsme zjistili v předchozí části, to platí pouze pro nedělené odkazy. U dělených odkazů se textem odkazu stává spojení všech obsahů elementů *A*, které tento dělený odkaz tvoří.

Získání textu by se tak lišilo podle toho, jestli se jedná o dělený nebo nedělený odkaz, a proto provedeme zobecnění pro oba typy odkazů. Zavedeme dva nové pojmy – **počátek odkazu** a **konec odkazu.** Za text odkazu pak budeme považovat text mezi počátkem a koncem odkazu.

U neděleného odkazu odpovídá počátku odkazu počáteční tag elementu *A* a konci odkazu ukončení elementu *A*, tak jak jsme ho určili v části 3.2.3. Pro dělený odkaz bude počátkem odkazu počáteční tag prvního elementu *A* děleného odkazu a koncem odkazu ukončení posledního elementu *A* tohoto děleného odkazu. Na obr. 3.1 je počátek a konec odkazu ve zdrojovém kódu vyznačen tučně.

Získání samotného textu, mezi počátkem a koncem odkazu, je triviální, protože jsme vycházeli z předpokladu, že pracujeme s předzpracovaným dokumentem. Výsledný text celého odkazu vytvoříme spojením textů jednotlivých částí tohoto odkazu.

3.2.7 Shrnutí

Rozborem úlohy ukládání odkazů jsme zjistili, které úkoly bylo nutné řešit. Nyní si tyto úkoly zopakujeme se stručným popisem jejich řešení.

1. *Lokalizace elementů A a BASE*. Tento úkol částečně řeší označení příslušných tagů v rámci předzpracování. Nalezení elementu *BASE* je potom triviální. Při hledání elementu *A* musíme počítat s třemi možnostmi jeho ukončení – ukončovacím tagem elementu *A*, počátečním tagem elementu A nebo koncem dokumentu.
2. *Zpracování adresy odkazu*. Řešení tohoto problému také nebylo obtížné. U obou elementů se adresa přiřazuje v počátečním tagu stejnému atributu *href* . Pouze u elementu *A* musíme počítat s možností uvedení relativní adresy a následným převodem do absolutního tvaru.
3. *Zpracování dělených odkazů*. Dělený odkaz ovlivňuje i následné získání odkazu, a proto jsme provedli zobecnění pro dělený i nedělený odkaz. To nám umožňuje zpracovávat text odkazu u obou typů odkazů shodně – spojením textů jednotlivých částí tvořící dělený odkaz.
4. *Získání textu odkazu*. Řešení tohoto úkolu také částečně řeší předzpracování. Texty jsou v unifikovaném tvaru, a proto nemusí být dále upravovány. Po provedeném zobecnění získáme text odkazu z obou typů odkazů shodným způsobem.

Všechny úkoly spojené s ukládáním odkazů jsme vyřešili. Po získání textu a adresy odkazu uživatel pouze označí typ textu a typ jednotlivých slov. Spolu s adresou a textem odkazu se tato data uloží do databáze.

3.3 Aktualizace odkazů

Poslední ze základních úloh této práce je aktualizace odkazů. Jde částečně o kombinaci předchozích dvou úloh, a proto v ní využijeme znalostí, postupů a závěry z nich vzešlých.

Už z principu aktualizace odkazů jsou patrné souvislosti. Hledáme všechny aktivní elementy *A* tvořící odkaz. Podle textu odkazu (neboli podle obsahu elementu) zjistíme, která adresa mu odpovídá v databázi. Pokud nejsou adresy shodné, adresou z databáze nahradíme stávající adresu tohoto odkazu v dokumentu. Jinak necháme odkaz v původním stavu.

Jak vidíme, některé části jsou naprosto stejné. Postup při hledání aktivních elementů můžeme použít z ukládání odkazů. Nalezení adresy v databázi (odpovídající textu odkazu) se shoduje s částí úlohy vytváření odkazů. Pouze tvorba nového HTML dokumentu se bude provádět jiným způsobem. Nevytvoříme celý nový element, ale pouze se v počátečním tagu přiřadí atributu *href* nová adresa.

3.3.1 Vstupy a výstupy

Vstupy a výstupy jsou stejné jako u vytváření odkazů. První vstup tvoří HTML dokument, ve kterém adresy odkazů aktualizujeme. Druhým vstupem jsou data z databáze. Za výstup považujeme nový HTML dokument s aktualizovanými odkazy. Pokud žádný odkaz v dokumentu neaktualizujeme, bude nový dokument stejný jako původní. V tomto případě jej není nutné vytvářet.

3.3.2 Specifikace problému

Jak už bylo řečeno, tato úloha je kombinací obou předešlých. Opět jí bude předcházet předzpracování, ale sama další nároky na předzpracování nemá. Pouze se více potvrzuje správnost volby jednotného přístupu k dokumentům. Všechny problémy, které se týkají této úlohy, tak byly vyřešeny u předchozích.

Jedinou odlišnou částí je tvorba nového dokumentu. Stejně jako u vytváření odkazů půjde o kopii původního dokumentu, tentokrát s aktualizovanými odkazy. Při jeho vytváření se nesmí zapomenout na možnost, že půjde o dělený odkaz a nová adresa se tedy musí přiřadit všem jeho částem.

Podobně jako při vytváření odkazů, i ze stejných důvodů, bude při výskytu více adres dotazován uživatel. Ani tento proces tak nebude plně automatický. Bude zde jedna výjimka, pokud mezi možnými novými adresami bude stejná jako v dokumentu, nebude se provádět aktualizace a tedy ani dotaz uživateli.

4. Předzpracování HTML dokumentu

V předchozí kapitole jsme zjistili, že první a společnou částí základních úkolů je předzpracování HTML dokumentu (dále jen předzpracování). Každá z úloh si pro svojí správnou funkci klade na předzpracování svoje vlastní požadavky, které se v mnohém liší. Dají se zobecnit na zachování významných elementů a úpravu textu dokumentu do unifikovaného tvaru. To nám zjednoduší jejich řešení. Také budeme mít nástroj, který uvede dokumenty do jednotného tvaru použitelného pro všechny tři úlohy. U textů tak budeme mít jistotu, že vždy projdou stejnou unifikací.

V dalších částech této kapitoly vytvoříme postup, který bude splňovat všechny požadavky a umožní tak další zpracování dokumentu. Ukážeme si z jakých fází se bude předzpracování skládat, vysvětlíme si použité postupy a provedeme návrh jejich realizace.

4.1 Princip předzpracování

Důvody, proč provádět předzpracování známe z předchozích kapitol. Z nich máme i mnoho informací pro jeho konečný návrh. Víme, že vstupem je dokument, který může obsahovat všechny fragmenty jazyka HTML. U výstupu nevíme přesně, jak má vypadat, ale víme, co je jeho cílem a co musí splňovat.

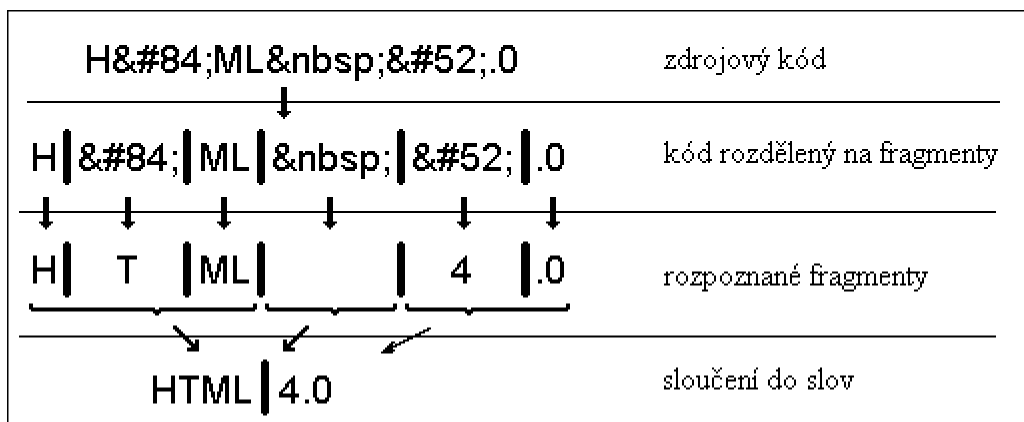
Principem předzpracování je převod HTML dokumentu do tvaru, který nám umožní pracovat buď pouze s textem dokumentu bez odkazů, se znalostí jeho struktury (pro určení odstavců, kapitol atd.) nebo pouze s odkazy. Jak toho docílíme?

Chceme předzpracovat HTML dokumentu, a proto budeme z tohoto jazyka vycházet. Každý takový dokument (viz. 2. kapitola) se skládá z elementů, tagů, entit a textů. My částečně toto rozdělení převezmeme. Dokument nejprve na tyto části rozdělíme, následně jim přiřadíme odpovídající význam a nakonec z nich vytvoříme slova. Předzpracování lze tedy rozdělit do tří bodů:

1. *Rozdělení dokumentu na fragmenty*. Dokument v této fázi rozdělíme na jednotlivé části – tagy, entity a texty (dále jen fragmenty), abychom poznali jeho strukturu. Elementy jako takové nepoužijeme, protože jsou jednoznačně dané počátečním, případně koncovým tagem.
2. *Přiřazení významu fragmentům*. Rozdělení nám umožní přiřadit jednotlivým fragmentů odpovídající význam. U tagů to znamená určit o jaký typ jde (odkaz, odstavec, tabulka atd.). K entitám přiřadíme odpovídající znak, pouze texty zatím ponecháme zatím bez úprav.
3. *Vytvoření slov z fragmentů*. Vytvoříme slova textu, což provedeme sloučením upravených fragmentů tvořících slovo. Dostaneme text dokumentu v unifikovaném tvaru, který bude složený pouze ze slov (případně dalších symbolů nebo číslic).

Na obr. 4.1 je uvedena ukázka postupu pro příklad z části 2.4 (text „*HTML 4.0*“). V první fázi rozdělíme zdrojový kód jednotlivé fragmenty, ve druhé fázi tyto fragmenty rozpoznáme a v poslední fázi spojíme rozpoznané fragmenty do slov.

V dalších částech této kapitoly jednotlivé fáze předzpracování rozebereme a navrhneme postupy k jejich realizaci. První a druhou fázi spojíme do rozpoznání fragmentů, protože rozdělením dostaneme fragment, který můžeme ihned identifikovat.



Obr.4.1. Princip předzpracování.

4.2 Rozpoznání kódu dokumentu

Rozpoznání provedeme rozdělením dokumentu na jednotlivé fragmenty, kterým přiřadíme odpovídající význam. Nejprve si ujasníme, jakým způsobem musíme rozdělení provést. Lze jednoznačně v následujícím příkladu (výřez zdrojového kódu) označit, jeho jednotlivé části?

…
**
**
Toto je testovací věta.
**
**
…

Nelze, protože nevíme, co bylo před touto částí kódu a jak kód pokračuje. Samozřejmě, může se jednat o trojici *tag – text – tag*, ale také nemusí. Stačí, aby před textem bylo „<!--“ a za textem „-->“ (nutné jsou obě části) a výsledkem není zmíněná trojice, ale pouze jeden komentář.

Na jednotlivé části musíme hledět v kontextu jejich umístění. Zpracování dokumentu (jeho zdrojového kódu) musí tedy probíhat od začátku, postupným rozpoznáváním částí. Tento postup nám vyhovuje, protože jedním z požadavků byla vazba na původní umístění ve zdrojovém kódu a postupným zpracováním budeme stále znát aktuální pozici ve zdrojovém kódu. Vždy tedy budeme rozpoznávat první neznámý fragment v kódu dokumentu.

Otázka, jakým způsobem provedeme rozdělení, už byla vlastně odpovězena. Řešením je postupné rozpoznávání a oddělovaní jednotlivých fragmentů. K tomu potřebujeme vědět, podle jakých pravidel lze určit, že se opravdu jedná o tag, entitu nebo text. Až budeme umět rozpoznat tyto fragmenty, přistoupíme k samotné realizaci rozpoznání.

4.2.1 Rozpoznání tagů

Při vytváření pravidel pro rozpoznání tagů, vyjdeme z praktických zkušeností s HTML. V části 2.4. byl popsán tvar, který je považován za tag. My toto pravidlo převezmeme a přidáme další doplnění.

Za tag budeme považovat takovou část kódu, která začíná symbolem ‚<‘, dále pokračuje písmenem (velkým i malým) z ASCII tabulky nebo symboly ‚?‘, ‚!‘ a ‚/‘ (dále jen druhý znak) a končí symbolem ‚>‘. Mezi druhým znakem a koncovým symbolem může být jakýkoliv počet různých znaků mimo symbolu ‚>‘. Z tohoto pravidla existují dvě výjimky.

Atributu můžeme přiřadit hodnotu, která je ohraničená uvozovkou nebo apostrofem. V rámci takové části mohou být jakékoliv jiné znaky, tedy i znak ‚>‘, který by jinak tag ukončoval, ale takto je považován za součást hodnoty přiřazené atributu. S tímto nepříliš

častým jevem musíme počítat. Když se v rámci tagu objeví takto ohraničená část, hledání ukončovacího symbolu musí pokračovat až za touto částí. Například u elementu odkazu může celý počáteční tag vypadat následovně:

*<A href=“http://www.feld.cvut.cz“ title=***“<ČVUT>“***>.*

Druhou výjimkou jsou komentáře. Tento element tvoří jeden speciální tag, který začíná čtveřicí znaků „<!--“ a končí trojicí znaků „-->“. Tag nemá žádné atributy, proto pro něj neplatí první výjimka. Komentář může vypadat například takto:

<!-- Toto je „celé <BR> komentář -->,

ale často se využívá jedinečnosti vytváření komentáře. Tag se jakoby rozdělí na tři části. Celý element pak vypadá jako složený z počáteční a koncového tagu a obsahu elementu:

<!-- >
Toto všechno je komentář
<-->.

Když víme, podle jakých pravidel rozpoznáme tagy, potřebujeme dále vědět, jestli se jedná o tag, který je pro nás důležitý. Element a tedy i tag jednoznačně určuje klíčové slovo. Potřebujeme znát pouze některé elementy, a proto není potřeba vytvářet kompletní seznam, ale stačí vytvořit tabulku, která bude obsahovat pro nás důležitá klíčová slova s uvedeným významem.

Porovnání klíčového slova v tabulce a klíčového slova v tagu určíme, o který tag (tedy element) se jedná. V normě jazyka HTML se doporučuje psaní těchto klíčových slov velkými písmeny. Samotný zápis elementů nerozlišuje velikost písmen (case insensitive), a proto *<HTML>* a *<hTmL>* jsou dva tagy se stejným významem. Samotné porovnání proto také nesmí rozlišovat velikost písmen.

4.2.2 Rozpoznání entit

U entit bylo podobné pravidlo jako u tagů popsáno v části 2.2.4. Bohužel univerzální pravidlo existuje pouze pro číselné entity. Rozpoznání entit určených jménem není úplně triviální a jde ruku v ruce s určením, jedná-li se o entitu.

Nejprve rozpoznání číselné entity. Číselnou entitou se stává taková část kódu, která začíná symbolem ‚*&*‘, dále pokračuje symbolem ‚*#*‘. Za touto dvojicí následuje nezáporná celočíselná hodnota zapsaná v desítkové soustavě. Zakončuje jí jakýmkoli jiným znakem mimo číslic. Uvedená celočíselná hodnota odpovídá kódu znaku ve znakové sadě Unicode.

Entity určené jménem mají podobný zápis. Začínají také symbolem ‚*&*‘, dále pokračují názvem entity a jsou zakončeny středníkem nebo bílým znakem[2]. Za entitu se považují pouze ty, jejichž názvy jsou uvedené v normě jazyka HTML. To znamená, že například *&aa;* je sice zapsáno jako entita, ale entita to není, protože jí nedefinuje norma. Oproti tomu *á* definuje norma a tak jde o entitu (‚*á*‘).

Tyto entity se dále dělí na rozlišující a nerozlišující velikost písmen. Nerozlišující většinou nahrazují symboly, které mají v HTML speciální význam. Jde například o nedělitelnou mezeru, pro kterou existují mimo jiné i tyto rovnocenné zápisy   . Entity rozlišující velikost písmen se většinou váží k zápisu různých písmen. Například *á* je entita pro ‚*á*‘ a *Á* je entita pro ‚*Á*‘, ale *&AAcute;* není entita.

---

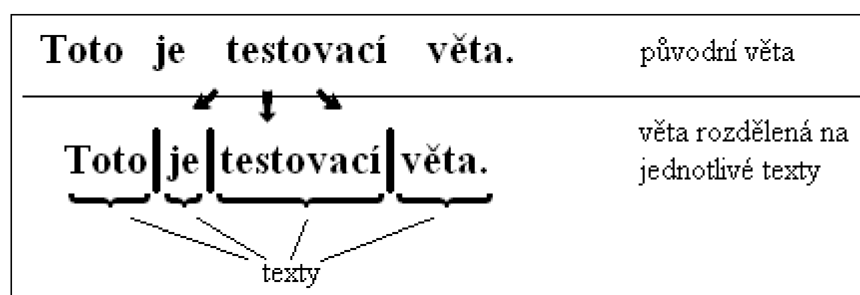
[2] *Bílý znak* je překlad anglického výrazu *white space* a zahrnuje důležité znaky, které nejsou na obrazovce vidět. Patří mezi ně znaky jako mezera, tabulátor, nový řádek, nová stránka, návrat na začátek řádku atd. Více informací je možné najít v [19] na str. 19, 20 a 29.

Rozlišující a nerozlišující entity proto musíme oddělit. Pro každou vytvoříme převodní tabulku, která bude obsahovat odpovídající znak. Pokud nebude potenciální entita odpovídat žádné entitě v tabulce, nebudeme jí považovat za entitu. Tato skutečnost nás nutí vytvořit tabulky s kompletním seznamem jednotlivých entit.

Takový postup má i svojí výhodu. Pokud detekujeme, že jde opravdu o entitu, přiřadíme jí ihned odpovídající znak. Také budeme vědět, jestli se jednalo o písmeno, symbol nebo bílý znak. U entit se jeví spojení prvních dvou fází předzpracování nejenom jako rozumné, ale téměř nutné.

4.2.3 Rozpoznání textů

K rozpoznání textu předem ještě jedna poznámka. V 2. kapitole jsme se dozvěděli, že formátování není závislé na předchozím formátu ve zdrojovém kódu dokumentu. Výsledný text dokumentu ovlivňuje pouze přítomnost bílých znaků ve zdrojovém kódů, ale ne jejich počet. Dále bylo v části 3.1 uvedeno, že porovnání celých textů bude ve výsledku převedeno na porovnání jednotlivých slov. My proto rozdělíme jednotlivé texty na části (většinou půjde o slova) oddělené nejenom tagy a entitami, ale také bílými znaky.



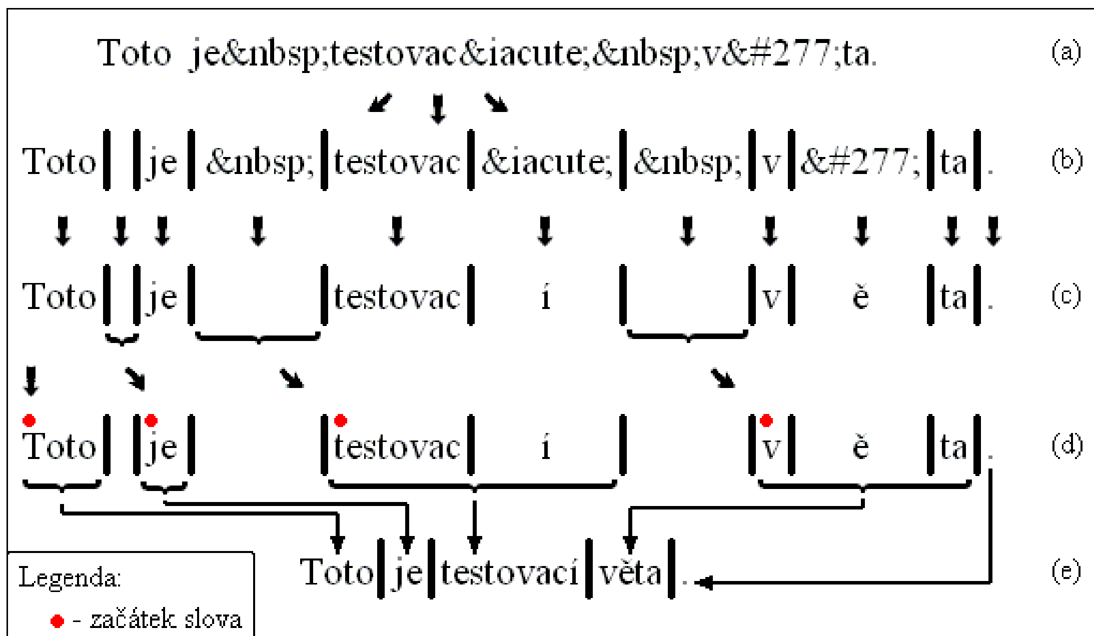
Obr.4.2. Rozdělení věty na jednotlivé texty.

Za text budeme považovat vše, co není entita nebo tag a neobsahuje bílé znaky. Rozdělení na texty provedeme jako na obr. 4.2. Z uvedeného příkladu vyplývá další zdroj možných problémů. Když tuto větu zapíšeme dvakrát za sebou bez mezery, jako na obr. 4.3(a), bude po rozdělení text „*věta.Toto*“ složen ze dvou slov, jak vidíme na obr. 4.3(b). Slova v tomto textu odděluje pouze znak. K porovnání potřebujeme pouze jednotlivá slova. Z tohoto důvodu budeme dále rozlišovat tři typy textů:

1. text tvořený pouze znaky (i s diakritikou) ‚a‘ – ‚ž‘ a ‚A‘ – ‚Ž‘,
2. text tvořený pouze číslicemi ‚0‘ – ‚9‘,
3. text tvořený pouze symboly, které nepatří do 1. ani do 2. skupiny.

Text „*věta.Toto*“ tak rozdělíme ještě na tři texty, jak vidíme na obr. 4.3(c).

Obr.4.3. Rozdělení vět oddělených pouze znakem, (a) původní tvar, (b) rozdělení na texty, (c) rozdělení na texty, které tvoří buď znaky, číslice nebo symboly.



Obr.4.4. Předzpracování složitější věty (a), větu rozdělíme na jednotlivé fragmenty (b) a rozpoznáme je (c). Označíme první fragment v dokumentu a fragmenty, které předcházel bílý znak – začátky slov (d) a z fragmentů vytvoříme slova (e).

Tím, že jsme vynechali bílé znaky, ztrácíme důležitou informaci pro případ, kdy výsledné slovo netvoří pouze jeden text. Obecně se může skládat z více částí. Když bude ve zdrojovém kódu uvedena věta z předchozího příkladu ve tvaru podle obr. 4.4(a), po rozdělení nebude například poslední slovo „*věta*“ možné bez této dodatečné informace znovu vytvořit. U každého fragmentu, tak musíme přidat informaci o tom, zda-li ho předcházel bílý znak nebo nikoliv.

4.2.4 Realizace rozpoznávání

Nyní víme na jaké části budeme dělit zdrojový kód HTML dokumentu a jaké informace každá část potřebuje. Z důvodu jejich uvedení v různých částech této práce, si je před návrhem vlastní realizace shrneme, aby bylo patrnější, co která struktura musí obsahovat.

Struktura pro **tag**:

- samotný tag
- typ tagu
- počáteční pozice v původním kódu
- indikátor, byl-li před tagem bílý znak (dále jen indikátor bílého znaku)
- indikátor zakázání nebo povolení dalšího zpracování (dále jen indikátor dalšího zpracování)

Struktura pro **entitu**:

- znak odpovídající entitě
- typ entity
- počáteční a koncová pozice v původním kódu
- indikátor bílého znaku
- indikátor dalšího zpracování

Struktura pro **text**:

- samotný text
- typ textu
- počáteční pozice v původním kódu
- indikátor bílého znaku
- indikátor dalšího zpracování

Z uvedených obsahů struktur vyplývá, že všechny mají společnou část a liší se pouze v tom, čím jsou prezentovány navenek (text a typ). Zapouzdřením těchto dat s příslušnými operacemi (například podle [10], [11]) dostáváme třídy (objekty), které nazveme ***Text***, ***Tag*** a ***Entita***.

Tímto způsobem rozdělíme zdrojový kód do datové struktury a vytvoříme pole těchto objektů. Při dalším zpracování bychom se ovšem museli stále dotazovat na typ třídy a pracovat s objekty odděleně. Třída *Tag* bude jistě používat jiný proces zpracování než třída *Text*. Využijeme toho, že všechny třídy mají společná data. Jejich zapouzdřením vytvoříme základní třídu, od které třídy *Text*, *Entita* a *Tag* zdědí svoje společné vlastnosti.

Základní třídu nazveme ***Fragment***. Obsahuje pouze společné datové prvky (atributy) a operace nad nimi (jednotný interface). Společné atributy všech tří tříd jsou:

- typ
- počáteční pozice v původním kódu
- indikátor bílého znaku
- indikátor dalšího zpracování

a dále základní třída obsahuje minimálně jeden řetězec pro uložení textu v třídě *Text*, uložení tagu v třídě *Tag* nebo uložení odpovídajícího znaku ve třídě *Entita*.

Použitím polymorfismu, využití virtuálních funkcí a realizací pozdní vazby (více informací například v [10]), dosáhneme toho, že můžeme s rozděleným kódem dokumentu pracovat bez nutnosti dalšího zjišťování a rozlišování o jakou třídu jde. Toto rozhodování zahrnuje použití polymorfismu a virtuálních funkcí.

Výsledkem rozpoznání bude tedy pole objektů, které nám umožní provést třetí fázi předzpracování jednoduše a jednotně, bez nutnosti dalšího třídění nebo rozlišování. Stačí pouze implementovat společné operace pro každou třídu zvlášť.

4.3 Slučování fragmentů – tvorba slov

Rozdělili jsme si text na fragmenty, které jsme uložili do pole objektů. Nyní navrhneme poslední část předzpracování – slučování fragmentů do slov. V předchozí části byl návrh realizace směrován prostřednictvím OOP (objektově orientované programování), a proto budeme i nadále využívat jeho výhody.

Slučování fragmentů v podstatě pouze rozšiřuje předchozí fázi předzpracování. Pole objektů, které je jejich výstupem, se zároveň stává vstupem slučování. Toto pole objektů tak tvoří data všech fází předzpracování. Zapouzdřením s příslušnými operacemi jednotlivých fází dostaneme třídu, která bude tvořit kompletní předzpracování.

Zůstávají nám pouze poslední dva problémy tykající se předzpracování:

- určení pravidel pro tvoření slov z fragmentů,
- reprezentace takto vytvořených slov.

Nejprve musíme určit pravidla, podle kterých se budou fragmenty do slov slučovat případně, které fragmenty budou slova rozdělovat. Když budeme umět slova vytvořit, můžeme přistoupit k řešení, jak vytvořená slova (sloučené fragmenty) reprezentovat.

4.3.1 Princip slučování

Na vstupu máme pole objektů typu *Text*, *Entita* nebo *Tag*. Slova mohou vzniknout pouze sloučením objektů typu *Text* a *Entita*. Tag jako část elementu, tedy ani objekt typu *Tag*, sám o sobě nemůže tvořit žádným způsobem text nebo část textu. Pro samotnou tvorbu slov jsou důležité dva parametry:

1. *Text fragmentu*. Při realizaci rozpoznání jsme s výhodou využili polymorfismu, abychom nemuseli rozlišovat mezi typy jednotlivých objektů. Z tohoto důvodu musíme vytvořit, jako součást společného rozhraní, jednu operaci, jejímž výstupem bude „viditelný“ text daného objektu. Výstupem pro třídu Text bude samotný text, pro třídu Entita odpovídající znak a pro třídu Tag bude prozatím výstupem prázdný řetězec.

2. *Oddělení slov*. Oddělení slov je neméně důležitým parametrem při tvorbě slov. Každý objekt nese informaci o tom, byl-li před ním v původním kódu bílý znak, což je první způsob oddělení slov. Druhou možnost oddělení skrývají některé entity (obr. 4.4) a většina pro nás zajímavých tagů (určeny v části 3.1.4). Oba způsoby oddělování slov jsou rovnocenné a nezáleží na tom, který byl použit.

Entita může být zařazena do první i do druhé. K jejich rozlišení použijeme odpovídající znak entity. Dalším možným řešením by bylo tyto entity speciálně označit v převodní tabulce entit.

Tagy mohou patřit pouze do druhé skupiny, ale jenom některé, a proto rozlišení tagů bude podobné. Každý takový tag v tabulce označíme. Následně stačí zjistit, o jaký tag jde a podle označení upravit výstup operace na získání „viditelného“ textu. S výhodou použijeme znak přechodu na nový řádek, protože ten už se jinak vyskytnou nemůže. Umožní nám to použít stejný princip – test na přítomnost bílého znaku u tagů i u entit.

Za slovo budeme považovat takovou skupinu objektů typu *Text* a *Entita*, kterou tvoří minimálně jeden takový objekt (s výjimkou *Entit* oddělující slova). Pouze prvnímu objektu ve skupině musí předcházet alespoň jeden bílý znak (realizován oběma způsoby), jak vidíme na obr. 4.4(c) a 4.4(d) například u slova „*věta*“. Ostatní objekty ve skupině nesmí předcházet žádný bílý znak ani takový znak nesmí tvořit. Výjimkou je první slovo v textu, u kterého nemusí první objekt ve skupině předcházet bílý znak jak vidíme na obr. 4.4(d) u slova „*Toto*“.

Parametry takto vytvořeného slova jsou jeho text, počátek v původním kódu, který odpovídá počáteční pozici prvního objektu ve skupině a konec v původním kódu, kterému odpovídá koncová pozice posledního objektu ve skupině. Takto vytvořená slova nám umožní převést porovnání slov z dokumentu a z databáze na přímé porovnání řetězců.

4.3.2 Reprezentace slov

Vytvořili jsme z fragmentů slova a nyní se rozhodneme, jakou pro ně vybereme vhodnou reprezentaci. Na výběr máme nesporně mnoho možností, ale všechny se dají rozdělit do dvou skupin, které se liší umístěním reprezentace:

1. *Umístit označení jednotlivých slov do původního pole objektů*. Například zvlášť označit první a poslední objekt ve skupině. Mezi výhodu tohoto postupu patří skutečnost, že stačí přidat jeden atribut základní třídě a výrazně se tak nezvýší paměťové nároky. Naopak mezi nevýhody můžeme zařadit nutnost opakovaně vytvářet každé slovo znovu, když budeme potřebovat jeho text.

2. *Vytvořit novou datovou strukturu*. Tato datová struktura by obsahovala všechny potřebné informace (text slova, počáteční a koncovou pozici). Výhody a nevýhody jsou inverzní vůči první skupině. Vytváříme novou datovou strukturu, která minimálně

zdvojnásobí paměťové nároky. Nemusíme ovšem při každém použití znovu a znovu skládat z objektů ve skupině jednotlivá slova.

Vzhledem k tomu, že HTML dokumenty v průměru nedosahují velikosti přes 1MB, nebudou paměťové nároky obou postupů oproti dostupnému adresovému prostoru (u novějších počítačů dosahuje velikost operační paměti minimálně 64MB) nijak velké. Reprezentovat jednotlivá slova proto budeme pomocí jiné datové struktury, než jakou jsme použili pro fragmenty HTML dokumentu.

Z důvodů, které nejsou zatím zřejmé, použijeme na reprezentaci jednotlivých slov také objekty. K dosavadním datovým prvkům (text slova, počáteční a koncovou pozici) přibudou další, které vyplynou z použitého algoritmu hledání textů odkazů v kapitole 7.

5. Návrh gramatik

V této kapitole provedeme návrh generujících gramatik pro vytváření přípustných tvarů jednotlivých textů. Výsledné gramatiky nejsou složité a jejich návrh není obtížný. I proto zde neuvádíme celou teorii. Použité věty a definice ponecháme bez důkazů. Lze je najít například v [6], [8] nebo [15]. Podle této literatury byl návrh gramatik proveden.

5.1 Co je to generativní gramatika?

Gramatiky spolu s automaty tvoří nejčastější způsob reprezentace jazyků. Aparát formálních gramatik byl původně navržen pro popis přirozeného jazyka. Nově našel použití i pro popis vyšších programovacích jazyků.

Podle [6] je generativní gramatika uspořádaná čtveřice $\zeta = (\Pi, \Sigma, S, P)$, kde Π a Σ jsou dvě disjunktní konečné abecedy, $S \in \Pi$, a P je konečná množina přepisovacích pravidel. Π se nazývá množinou neterminálů (proměnných), Σ se nazývá množinou terminálů a S je počáteční symbol. Přepisovací pravidla se zapisují ve tvaru $\alpha \to \beta$, kde α, β jsou slova z abecedy $\Pi \cup \Sigma$, přičemž α obsahuje alespoň jeden symbol z abecedy Π.

Jazyk $L(\zeta)$ generovaný generativní gramatikou ζ je definován:

$$\mathrm{L}(\zeta) = \{w, w \in \Sigma^* \ a \ \ S \Rightarrow_{\zeta}^{*} w\},$$

kde Σ^* je množina posloupností tvořena sjednocením prázdné posloupnosti e a všech konečných neprázdných posloupností vytvořených z prvků jazyka Σ. Jazyk tvoří všechny slova v terminální abecedě, která lze odvodit z počátečního symbolu S.

5.2 Generování textů

Generující gramatika je podle definice uspořádaná čtveřice. Abychom vyhověli podmínce $S \in \Pi$, přiřadíme do množiny neterminálů počáteční symbol S a bude to jediný počáteční symbol, který budou tyto úlohy mít. Pro návrh gramatiky potřebujeme určit ještě obsah množiny terminálních a neterminálních symbolů a navrhnout přepisovací pravidla.

5.2.1 Terminály

Nejprve určíme prvky množiny terminálů. Generujeme tvary textů, které se skládají z jednotlivých skupin slov. Jak bylo uvedeno v části 3.1.5, každou skupinu můžeme nahradit symbolem a dostaneme pro každý text skupinu terminálů.

Tab.4. Označení skupin slov terminálními symboly.

Název skupiny	Terminální symbol
Titul před jménem	t
Křestní jména	k
Příjmení	p
Titul za jménem	u
Části názvu	n
Části zkr. názvu	z

Označení jednotlivých skupin je uvedeno v tab. 4. Terminály se značí malými písmeny. Pro jméno osoby dostáváme množinu terminálních symbolů $\Sigma_1 = \{t, k, p, u\}$, pro název a zkr. název dostáváme množiny o jednom terminálu $\Sigma_2 = \{n\}$ a $\Sigma_3 = \{z\}$.

5.2.2 Neterminály

Neterminály nebo také proměnné (označujeme je velkými písmeny) použijeme jen pro ty skupiny slov, které mohou mít více stavů. Jedná se o ty skupiny, které mají jako jednu z vlastností možnost vynechání. Jsou tedy dvoustavové, text buď obsahují nebo neobsahují. Z tohoto důvodu přidáme neterminály pouze u jména osoby. Skupinu titul před jménem označíme symbolem A, křestní jméno označíme B a titul za jménem označíme písmenem D. U ostatních typů textů není žádná skupina s touto vlastností.

Množina neterminálů u všech skupin obsahuje počáteční stav S. Výsledné podoby jednotlivých množin jsou:

- pro jméno osoby: $\Pi_1 = \{S, A, B, D\}$,
- pro název: $\Pi_2 = \{S\}$,
- pro zkr. název: $\Pi_3 = \{S\}$.

5.2.3 Přepisovací pravidla

Přepisovací pravidla pro název (P_2) a zkr. název (P_3) jsou triviální. Oba typy obsahují pouze jedno přepisovací pravidlo ve tvaru: $X \rightarrow x$. U jména osoby musí přepisovací pravidla splnit jednu nutnou podmínku, kdy každý přípustný tvar jména osoby musí vždy obsahovat příjmení. Ostatní části obsahovat může.

Výsledná přepisovací pravidla pro název mají tvar:

$$P_2 = \{ S \rightarrow n \}$$

a pro zkr. název:

$$P_3 = \{ S \rightarrow z \}.$$

Přepisovací pravidla P_1 pro jméno osoby nejprve zapíšeme ve tvaru přípustném pro vypouštěcí gramatiku, aby bylo patrné na první pohled jaké tvary gramatika generuje:

(1) $S \rightarrow ABpD$,
(2) $Bp \rightarrow pB$,
(3) $A \rightarrow t \mid e$,
(4) $B \rightarrow k \mid e$,
(5) $D \rightarrow u \mid e$.

Jako příklad odvodíme jméno „*ing. Novák Jan*“ neboli slovo ‚*tpk*‘ (pro lepší orientaci jsou v závorkách uvedena čísla použitých pravidel):

$$\mathrm{S} \Rightarrow_{(1)} \mathrm{ABpD} \Rightarrow_{(2)} \mathrm{ApBD} \Rightarrow_{(3)} \mathrm{tpBD} \Rightarrow_{(4)} \mathrm{tpkD} \Rightarrow_{(5)} \mathrm{tpk}.$$

Nyní tuto vypouštěcí gramatiku přepíšeme na nevypouštěcí:

(1) $S \rightarrow ABpD \mid ABp \mid BpD \mid ApD \mid Ap \mid Bp \mid pD \mid p$,
(2) $Bp \rightarrow pB$,
(3) $A \rightarrow t$,
(4) $B \rightarrow k$,
(5) $D \rightarrow u$,

a odvodíme stejné slovo jako v předchozím příkladě:

$$S \Rightarrow_{(1)} ABp \Rightarrow_{(2)} ApB \Rightarrow_{(3)} tpB \Rightarrow_{(4)} tpk.$$

Když se podíváme na první řádek přepisovacích pravidel nevypouštěcí gramatiky, narazíme na tvar $S \rightarrow p$, který je shodný s pravidly pro název a zkr. název. Této informace lze využít při implementaci gramatik, neboť název a zkr. název se tvoří stejným způsobem jako jméno osoby tvořené pouze příjmením.

5.3 Použití gramatik

Gramatiky máme navrženy, ale jejich použití nemusí být zcela jasné. Jak se například zachováme, když není text tvořen všemi skupinami? Pro každý typ textu uvedeme použití příslušné gramatiky na jednom textu.

5.3.1 Název a zkratkový název

Jako příklad mějme název: „*České vysoké učení technické*“ (u zkráceného názvu je postup naprosto totožný). Tento název lze rozdělit na čtyři slova: „*České*“ – „*vysoké*“ – „*učení*“ – „*technické*“. Všechny slova patří do skupiny části názvu (pro zkr. název je to skupina části zkr. názvu), proto přidělíme slovu „*České*“ pořadové číslo 1, slovu „*učení*“ pořadové číslo 2 atd.

Známe pořadí ve skupině, proto můžeme celou skupinu nahradit jedním symbolem *f* a dostaneme:

$$n \approx \left[\frac{\text{České} \quad \text{vysoké} \quad \text{učení} \quad \text{technické}}{1 \qquad 2 \qquad 3 \qquad 4} \right]$$

Definujme nyní celou úlohu.

Nechť $\zeta_2 = (\Pi_2, \Sigma_2, S, P_2)$ je generující gramatika, kde $\Pi_2 = \{S\}$, $\Sigma_2 = \{n\}$ a pravidla $P_2 = \{S \rightarrow n\}$. Potom jazyk tvoří jednoprvková množina $L_2(\zeta_2) = \{n\}$, neboť gramatika generuje pouze jeden tvar. Po dosazení obsahu skupiny v daném pořadí za symbol *n* dostaneme opět stejný tvar.

5.3.2 Jméno osoby

U názvu nebo zkr. názvu nemůže dojít k situaci, kdy je skupina prázdná, ale u jména osoby to bude častý případ. Z tohoto důvodu jako příklad jména osoby použijeme jméno: „*Jan Ámos Komenský*“.

Jméno rozdělíme na tři slova: „Jan“ – „Ámos“ – „Komenský“. První dvě slova jsou křestní jména následována příjmením. Opět přiřadíme slovům ve skupině pořadí a nahradíme je symboly. Jednotlivé skupiny pro toto jméno vypadají následovně:

$$t \approx [\ ],$$
$$k \approx \left[\frac{Jan \quad \acute{A}mos}{1 \qquad 2} \right],$$
$$p \approx \left[\frac{Komensk\acute{y}}{1} \right],$$
$$u \approx [\ ].$$

Protože jméno neobsahovalo titul před jménem ani za jménem, jsou obě tyto skupiny prázdné. To nám ovšem nebrání v použití jejich odpovídajících symbolů, pouze s vědomím toho, že zastupují prázdnou skupinu. Opět definujeme celou úlohu.

Nechť $\zeta_1 = (\Pi_1, \Sigma_1, S, P_1)$ je gen.gramatika, kde $\Pi_1 = \{S, A, B, C\}$, $\Sigma_1 = \{t, k, p, u\}$ a $P_1 = \{S \rightarrow ABpD \mid ABp \mid BpD \mid ApD \mid Ap \mid Bp \mid pD \mid p;\ Bp \rightarrow pB;\ A \rightarrow t;\ B \rightarrow k; \mathrm{D} \rightarrow u\}$. Jazyk tvoří všechny přípustné tvary: $L_1(\zeta_1) = \{tkpu, tpku, tkp, tpk, tpu, kpu, pku, tp, pu, kp, pk, p\}$.

Dosazování skupin za jednotlivé symboly, dostaneme všechny přípustné tvary. Pro námi testované jméno vytvoříme tvary:

„Jan Ámos Komenský“
„Komenský Jan Ámos“
„Jan Ámos Komenský“
…
„Komenský“
„Komenský“

I z tohoto krátkého seznamu vidíme možnost vygenerování jednoho tvaru vícekrát, pokud jsou některé skupiny prázdné. Otázkou zůstává, jde-li o chybu nebo ne.

Generování používáme na vytvoření všech přípustných tvarů textu. Vygenerovaný tvar porovnáme s textem v dokumentu. Co se stane, když takto otestujeme stejný tvar několikrát? V případě, že se nebude shodovat jednou, nebude se shodovat ani podesáté, pouze tak ztratíme výpočetní čas. Pokud se bude shodovat jednou, bude se shodovat i podesáté. Opět ztratíme pouze výpočetní čas. Celé je to spíše na zamyšlení optimalizace při implementaci generování tvarů nebo jejich porovnávání.

Podařilo se nám navrhnout gramatiky pro generování přípustných tvarů všech typů textů. Jejich použití nám umožní porovnání dvou textů v různých tvarech.

6. Databáze

V této kapitole provedeme návrh databáze použité pro ukládání získaných dat. Nejprve se zmíníme o SQL a uvedeme několik důležitých parametrů použitého databázového serveru. Následně vytvoříme E-R konceptuální model (dále jen E-R model) a navrhneme strukturu databáze. Podobně jako u gramatik nebudeme uvádět celou problematiku návrhu a tvorby E-R diagramů a jejich převodu do relačního schématu. Více informací je možné najít například v [18].

6.1 SQL a MySQL

Počátky SQL neboli Structed Query Language sahají až do roku 1974 a postupně se stal jedním z nejrozšířenějších databázových jazyků. Nejde pouze o dotazovací (neprocedurální) jazyk, kterým popisujeme, co požadujeme od databáze, ale lze v něm definovat vlastní data, provádět aktualizace, definovat přístupová práva nejen k databázím, ale i k jednotlivým tabulkám atd. Základními rysy databázových modelů v SQL jsou shrnuty v následujících bodech:

- data v databázi jsou uložena a vždy prezentována ve formě tabulek, které jsou buď skutečné (odpovídající schématu databáze) nebo virtuální,
- tabulky a jejich sloupce jsou identifikovány jménem a jejich poloha a pořadí není důležité,
- řádky jsou identifikovány uloženými hodnotami ve sloupcích a jejich pořadí také není důležité,
- SQL umožňuje definice indexů pro rychlejší přístup k záznamům.

Jako databázový server byl v projektu použit MySQL - vícevláknový databázový server. Mezi jeho hlavní přednosti patří rychlost, robustnost, autorizovaný přístup k záznamům a také to, že jde volně dostupný software.

MySQL implementuje jazyk SQL a je založen na spojení *client-server*, kde server je typu *deamon* (například ve *Windows NT* spuštěn jako služba – service) a zpracovává všechny klientovi požadavky. Klientem mohou být libovolné aplikace nebo knihovny. Každý klient se připojí k serveru přes *TCP* (nebo *unix socket*) a server pro něj spustí vlastní komunikační vlákno.

V SQL dotazech plně podporuje funkce „*SELECT*“ jak v dotazové tak v podmínkové části, dále podporuje třídění a řazení. Tabulky vytváří jako rychlé B-stromy s kompresí. V každé tabulce mohou mít sloupce pevnou nebo proměnou délku záznamu. Podporuje všechny známé typy od znaků a jednoduchých přesných a aproximativních numerických typů, přes znakové řetězce s proměnnou délkou, až po časové údaje, výčtové typy a struktury. Pro rychlé přístupy k záznamům používá hashovací tabulky v operační paměti počítače.

MySQL podporuje řada operačních systému (např. Windows, Solaris, Red Hat Linux) a zároveň podporuje řadu programovacích jazyků (C, C++, Java, PHP atd.).

6.2 Návrh databáze

Databáze pro tuto úlohu není složitá, v podstatě ukládáme pouze texty a adresy odkazů spolu s dodatečnými informacemi. Nejprve postupně zkonstruujeme E-R model přidáváním

jednotlivých atributů, vazeb a entit[3]. Následně provedeme transformaci E-R modelu do relačního schématu databáze.

6.2.1 Konstrukce E-R modelu

Jak již bylo zmíněno, text a adresa tvoří základní data, které ukládaná do databáze. Ostatní (dodatečné) informace postupně přidáme. Model pro tento stav je uveden na obr. 6.1 a vyjadřuje vztah mezi textem a adresou, které spolu tvoří odkaz. Nyní určíme kardinalitu a členství v tomto vztahu.

Obr.6.1. Základní E-R model.

Členství ve vztahu označuje možnost, kdy každý výskyt entity je zapojen do vztahu (povinné členství), nebo připouští možnost existence i mimo vztah (nepovinné členství). K porovnání používáme texty. Když nalezneme nějaký text v dokumentu, potřebujeme k němu znát adresu. Texty proto mají povinné členství ve vztahu a označíme ho kolečkem uvnitř obdélníku entitního typu jako na obr. 6.2. Adresy nemusí být ve vztahu, jejich uvedení se nemusí vázat na žádný text. Nepovinné členství ve vztahu označíme kolečkem vně obdélníku (obr. 6.2).

Obr.6.2. Základní E-R model s vyznačenou kardinalitou a členstvím ve vztahu.

Kardinalitou vztahu vyjadřujeme poměr vztahů mezi entitními typy. V našem případě jde o vztah mezi entitou typu text a entitou typu adresa. Mohou nastat 3 případy poměrů:

- **1:1 -** jednomu textu odpovídá jedna adresa a jedné adrese odpovídá jeden text
- **1:N** - jednomu textu odpovídá několik adres a jedné adrese odpovídá pouze jeden text nebo naopak,
- **M:N** - jednomu textu odpovídá několik adres a jedné adrese několik textů.

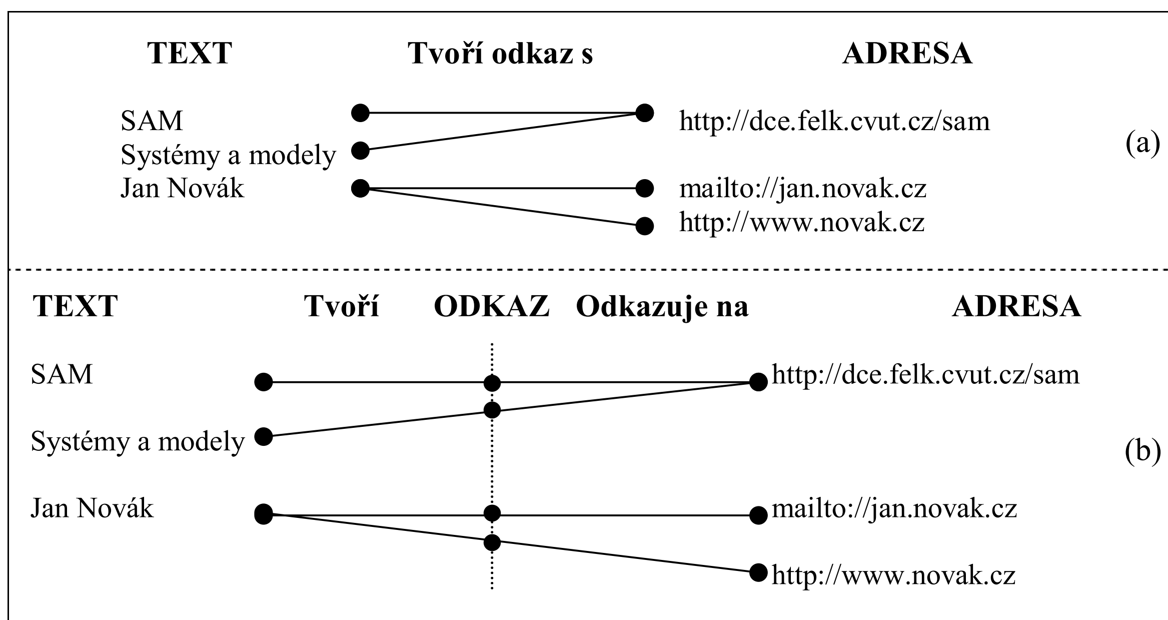
Z rozboru úloh vyplynula možnost, kdy u jednoho textu máme více adres (například internetovské stránky a emailová adresa). Adresa může být také svázána s více texty, například při existenci názvu i zkr. názvu. Výsledná kardinalita je proto typu **M:N** (obr. 6.2).

Obr.6.3. Základní E-R model s integritním omezením.

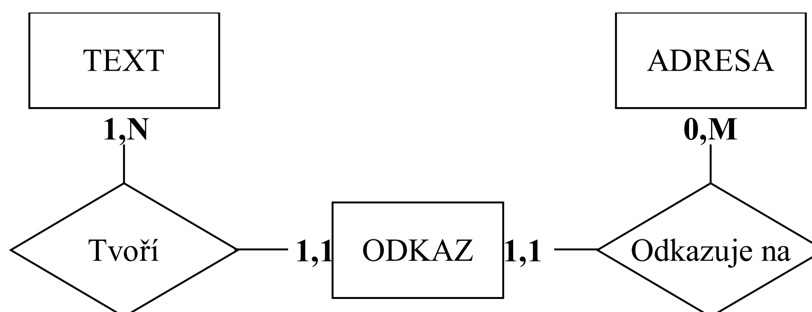
[3] *Entita* má v této kapitole jiný význam než v předešlých kapitolách. Používá se pro označení objektu reálného světa, který chceme modelovat. Terminologie k problematice databází byla převzata z [18].

Kardinalita i členství ve vztahu R(E_1, E_2) se dají vyjádřit integritním omezením ve tvaru: $\mathbf{E_i : (min_i, max_i)}$, které označuje minimální a maximální počet výskytů entity typu E_i ($i \in \{1, 2\}$) ve vztahové množině R. V našem případě, pro kardinalitu M:N a s povinným členstvím textu ve vztahu, dostáváme omezení: **TEXT:$\mathbf{E_1}$(1, n)** a **ADRESA:$\mathbf{E_2}$(0, m)**. Příslušný E-R model je na obr. 6.3

Poměr M:N nemusí být vždy na straně databázového systému (dále jen DBS) akceptovatelný. Některé DBS neumějí tento poměr přímo vyjádřit, proto provedeme jeho dekompozici na dva vztahy typu 1:N. Když se podíváme na obr. 6.4(a), kde je uveden jednoduchý M:N diagram výskytů pro vztah „*Tvoří odkaz s*“, vidíme, že každý výskyt odpovídá konkrétnímu odkazu. Budeme-li s těmito výskyty pracovat jako s novým entitním typem, dostaneme dva vztahy 1:N, jak je uvedeno na obr. 6.4(b). Nový průnikový entitní typ nazveme **ODKAZ**, vztah mezi ním a entitním typem TEXT nazveme „***Tvoří***“ a vztah s entitním typem ADRESA nazveme „***Odkazuje na***“. Odpovídající E-R model je na obr. 6.5.



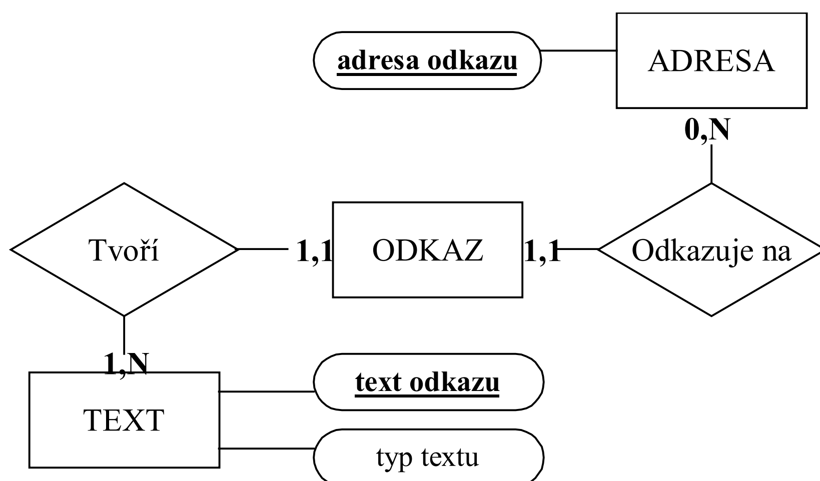
Obr.6.4. M:N diagram výskytů (a) pro vztah „*Tvoří odkaz s*“,
(b) pro rozložený vztah na dva vztahy 1:N „*Tvoří*“ a „*Odkazuje na*“.



Obr.6.5. Dekomponovaný E-R model.

Atributy entitních typů ADRESA a TEXT jsou dány rozborem úloh. Adresu jednoznačně určuje **adresa odkazu**. U textů ukládáme samotný **text odkazu** a **typ textu** (obr. 6.6). K atributu *text odkazu* musíme podle rozboru ukládat ještě typ jednotlivých slov a jejich pořadí. To by vedlo na tabulku s proměnným počtem sloupců a tedy na vícehodnotový

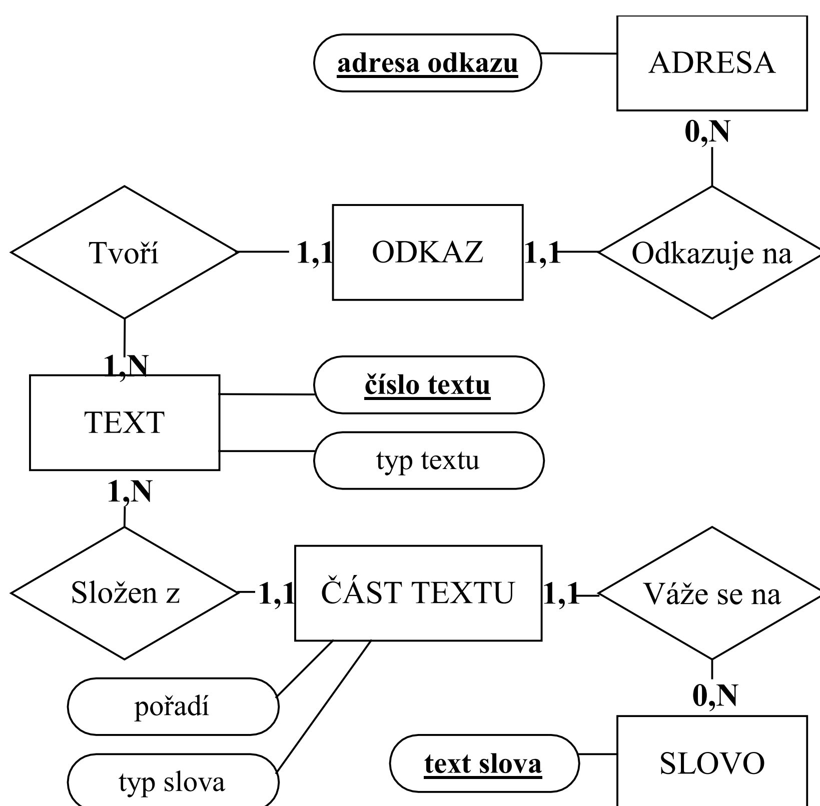
atribut. Vytvoříme proto nový entitní typ **SLOVO**, který bude jednoznačně určen atributem **text slova**.



Obr.6.6. E-R model s vyznačenými atributy entitních typů
(tučně jsou vyznačeny klíčové atributy).

Text odkazu může být tvořen více slovy a jedno slovo může být obsaženo ve více textech (dokonce i vícekrát v jednom textu, například u jména *Pavel Pavel*). Jde proto o vztah M:N. Opět provedeme dekompozici a průnikový entitní typ nazveme **ČÁST TEXTU**. Právě u každé takové části textu určíme typ slova a jeho pořadí.

Text měl opět povinné členství ve vztahu se slovem, protože každý text se skládá alespoň z jednoho slova. Pro jednoznačnou identifikaci textu přidáme atribut **číslo textu**, které bude pro každý text unikátní. Výsledný E-R model je na obr. 6.7.



Obr.6.7. Výsledný E-R model.

6.2.2 Relační schéma databáze

Chceme, aby výsledná databáze po transformaci z E-R modelu i její jednotlivé tabulky byly normalizované, tj. aby byly ve 3. normální formě (3NF). Náš konečný E-R model na obr. 6.7 neobsahuje žádné skupinové ani hromadné atributy, proto bude transformace snadná. Stačí vyřešit reprezentaci uvedených vztahů 1:N s povinným a nepovinným členstvím entitních typů.

Tab.5. Relační schéma - entitní typy s příslušnými atributy.

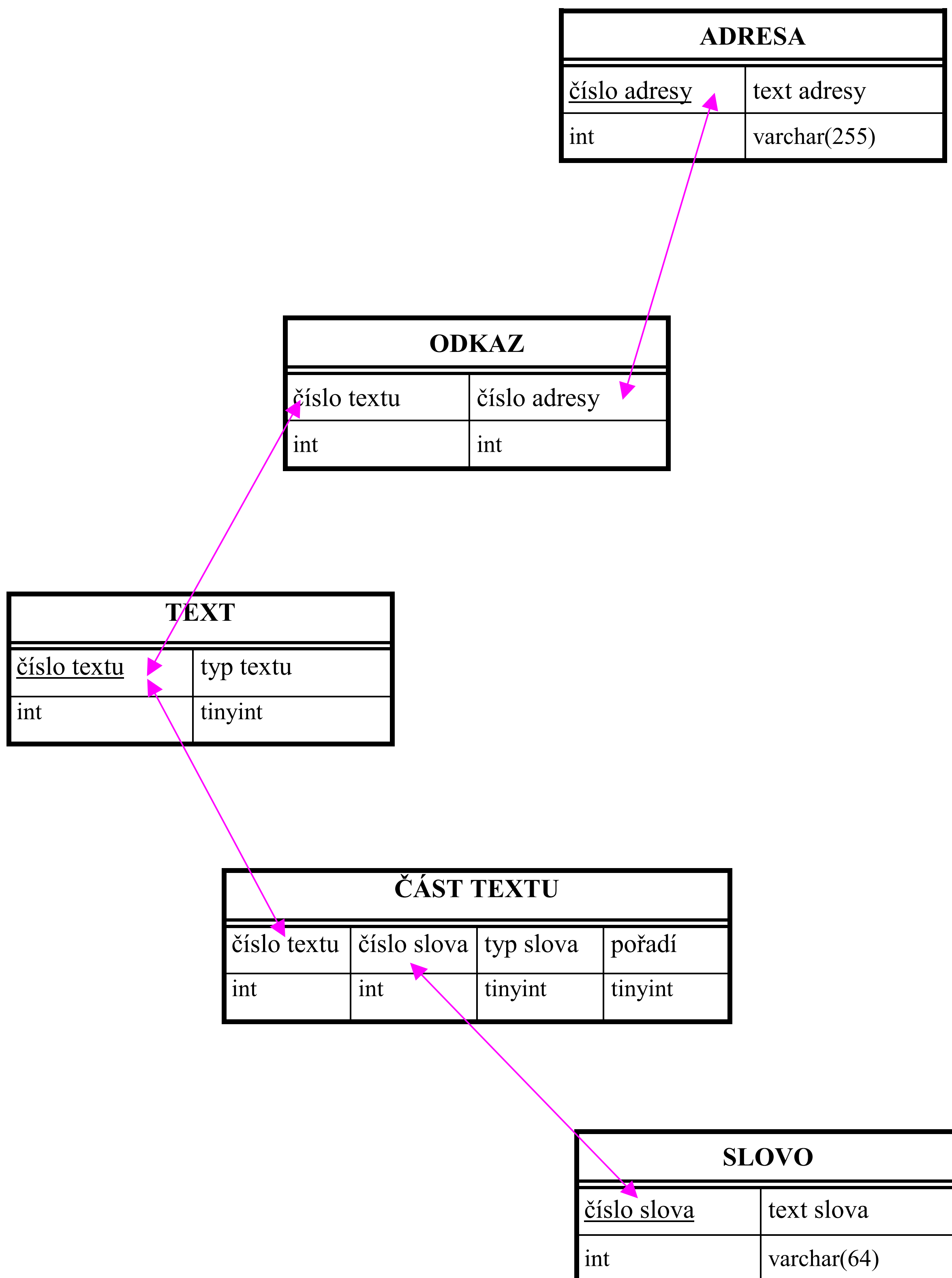
Entitní typ	Atributy
ADRESA	<u>číslo adresy</u>, text adresy
ODKAZ	číslo adresy, číslo textu
TEXT	<u>číslo textu</u>, typ textu
ČÁST TEXTU	číslo textu, číslo slova, typ slova, pořadí
SLOVO	<u>číslo slova</u>, text slova

V našem případě jsou vztahy pro prezentaci totožné. U všech má alespoň jeden entitní typ povinné členství ve vztahu 1:N. Podle [18] je nejjednodušší způsob reprezentace takového vztahu připojení klíčového atributu s kardinalitou N ke schématu s kardinalitou 1. Například u entitního typu ODKAZ by to znamenalo přidání klíčových atributů *text adresy* a *číslo textu*. Klíčový textový atribut *text adresy* nahradíme klíčovým číselným atributem **číslo adresy** (obdobně u entitního typu SLOVO). K entitnímu typu ODKAZ tak přidáme atributy *číslo adresy* a *číslo textu*. Výsledné relační schéma je v tab. 5. U každého atributu musíme ještě určit jeho datový typ. Pouze text slova a text adresy jsou řetězce (pole znaků), ostatní atributy lze vyjádřit celočíselným typem.

Tab.6. Datové typy všech atributů (N – celočíselná hodnota, X – pole znaků) i pro MySQL a rozlišení klíčových atributů.

Entitní typ	Atribut	Typ:rozsah	MySQL typ	Klíč
ADRESA	číslo adresy	N:4	int unsigned	ano
	text adresy	X:255	varchar(255)	ne
ODKAZ	číslo textu	N:4	int unsigned	ano (cizí)
	číslo adresy	N:4	int unsigned	ano (cizí)
TEXT	číslo textu	N:4	int unsigned	ano
	typ textu	N:1	tinyint unsigned	ne
ČÁST TEXTU	číslo textu	N:4	int unsigned	ano (cizí)
	číslo slova	N:4	int unsigned	ano (cizí)
	typ slova	N:1	tinyint unsigned	ne
	Pořadí	N:1	tinyint unsigned	ne
SLOVO	číslo slova	N:4	int unsigned	ano
	text slova	X:64	varchar(64)	ne

Typy jednotlivých atributů a označené klíčové atributy (i cizí) jsou uvedeny v tab. 6. Žádný z atributů nemůže mít tzv. prázdnou hodnotu (NULL) a proto tento sloupec tabulka neobsahuje. Symbolem ***N*** je označen celočíselný typ a symbolem ***X*** znakový typ. Rozsah u celočíselných hodnot udává počet bytů, to znamená, že N:1 tvoří 1 byte a má rozsah – 127 až 128 nebo bez znaménka 0 až 255. U znaků rozsah udává jejich počet v poli (délka řetězce). Sloupec „**MySQL typ**“ v tabulce uvádí odpovídající datové typy v MySQL.



Obr.6.8. Grafický zápis schématu databáze.

7. Algoritmy

V této kapitole jsou uvedeny použité algoritmy pro jednotlivé úlohy. Nejprve si vytvoříme algoritmus pro správu adres, jehož použitím zjednodušíme návrh algoritmu pro ukládání a aktualizaci odkazů. Tyto algoritmy následně sestavíme podle analýzy jejich zadání, kterou jsme provedli v 3. kapitole a tím jsme v podstatě provedli první krok návrhu. Půjde nám o přesné určení jejich jednotlivých kroků. Algoritmus vytváření odkazů vytvoříme zcela nový, protože použití „hrubé síly“ počítače při velkém objemu dat v databázi nebo ve zpracovávaném souboru není úplně ideální. U všech algoritmů předpokládáme na vstupu předzpracovaný HTML dokument.

7.1 Algoritmus pro správu adres

Než přistoupíme k sestavení algoritmu ukládání a aktualizace, vytvoříme si nejprve doplňující algoritmus pro správu adres (***LinkManager***). Úkoly tohoto algoritmu lze shrnout do 4 bodů:

- získání adresy z elementu *BASE* a její uložení pro převod adres z relativního do absolutního tvaru,
- získávání adres z počátečních tagů elementů *A* s možným převodem na absolutní tvar
- umět porovnat poslední dvě získané adresy z elementů *A* a vyhodnotit jejich podobnost jako stejné nebo rozdílné,
- rozlišovat výsledný tvar adres.

Fragmenty jsme při předzpracování označili typem, a proto umíme rozlišit počáteční a koncový tag elementu *A* a element *BASE*. Vstupem tohoto algoritmu je vždy pouze jeden fragment, který na začátku uložíme do proměnné ***fragment*** a následně zpracujeme. Výstupem bude nejenom adresa, ale také dva příznaky: ***form*** a ***identical***.

Tab.7. Nastavení příznaku *form* podle typu a tvaru fragmentu a dostupnosti základní adresy.

Fragment	**Základní adresa**	**Nastavení příznaku *form***
počáteční tag elementu *A* bez atributu *href*	X	NONE
počáteční tag elementu *A* bez přiřazené adresy atributu *href*	X	NONE
počáteční tag elementu *A* s adresou v relativním tvaru	NE	RELATIVE
počáteční tag elementu *A* s adresou v relativním tvaru	ANO	ABSOLUTE
počáteční tag elementu *A* s adresou v absolutním tvaru	X	ABSOLUTE
koncový tag elementu *A*	X	END
element *BASE* bez atributu *href*	X	NONE
element *BASE* bez přiřazené hodnoty atributu *href*	X	NONE
element *BASE* s přiřazenou hodnotou atributu *href*	X	BASE_ADDRESS
ostatní fragmenty	X	NONE

Příznakem *form* rozlišíme o jaký tvar adresy se jedná. Jeho nastavení závisí na typu fragmentu a dostupnosti základní adresy. Možná nastavení jsou uvedeny tab. 7. Příznak *identical* nastavíme podle výsledku porovnání dvou posledně zpracovávaných adres. Tento příznak má pouze dvě hodnoty: YES při shodě adres a NO v ostatních případech.

Algoritmus musí mít pro porovnání a převod tři paměťové prvky na uložení nové adresy, posledně získané adresy a také na uložení základní adresy. Tyto proměnné nazveme ***newAddress***, ***lastAddress*** a ***baseAddress***. Celý algoritmus je sestaven v následujících bodech:

(1) fragment na vstupu ulož do proměnné *fragment*;
(2) if (*fragment* = počáteční tag elementu A) then pokračuj bodem (6);
(3) if (*fragment* = koncový tag elementu A)
then *form* := END a ukonči algoritmus;
(4) if (*fragment* = element BASE) then pokračuj bodem (13);
(5) *form* := NONE; ukonči algoritmus;
(6) if (*fragment* neobsahuje atribut href) then *form* := NONE a ukonči algoritmus;
(7) *newAddress* := hodnota přiřazená atributu href (adresa);
(8) if (*newAddress* není platná) then *form* := NONE a ukonči algoritmus;
(9) if (*newAddress* je v relativním tvaru and *baseAddress* je platná)
then převeď *newAddress* do absolutního tvaru a výsledek ulož do *newAddress*
else *form* := RELATIVE;
(10) if (*newAddress* je v absolutním tvaru) then *form* := ABSOLUTE;
(11) if (*newAddress* = *lastAdrress*) then *identical* := YES else *identical* := NO;
(12) *lastAddress* := *newAddress* a ukonči algoritmu;
(13) if (*fragment* neobsahuje atribut href) then *form* := NONE a ukonči algoritmus;
(14) *baseAddress* := hodnota přiřazená atributu href (základní adresa);
(15) if (*baseAddress* není platná) then *form* := NONE
else *form* := BASE_ADDRESS;
(16) ukonči algoritmus;

7.1 Algoritmus ukládání odkazů

Nyní můžeme přistoupit k sestavení samotného algoritmu ukládání odkazů. Použijeme předchozí algoritmus k rozpoznání tvaru a podobnosti adres. Odkazovat se na něj budeme jako na *LinkManager*. Pro uložení textu odkazu použijeme proměnnou ***linkText*** a pro uložení adresy odkazu proměnnou ***linkAddress***.

Před uložením obsahu těchto proměnných do databáze se otestuje jejich přítomnost v databázi. Uloží se pouze ty data, která databáze ještě neobsahuje. Například se může stát, že získáme text s adresou a tato adresa je již uložená v databázi. Nejprve tedy zjistíme číslo odpovídající této adrese v databázi a následně uložíme pouze text s tímto číslem adresy.

Algoritmus postupně zpracuje všechny fragmenty, které jsou uloženy v poli ***arrayFragment*** (indexovaný od 0). Aktuální pozici právě zpracovávaného fragmentu obsahuje proměnná ***i*** a k příslušnému fragmentu se dostaneme indexací – *arrayFragment[i]*.

Celý algoritmus lze rozdělit do čtyř částí. První část tvoří bod (1) a jde o inicializaci, kde se nastaví pozice, od které má algoritmus začít. V druhé části (body (2) až (6)) hledáme pozici prvního počátečního tagu aktivního elementu *A* (pomocí *LinkManageru*) od aktuální pozice. Pozici tohoto tagu si uložíme do proměnné ***beginText***. Třetí část algoritmu mezi body (7) až (11) hledá ukončení odkazu, které uloží do proměnné ***endText***, a těsně spolupracuje se čtvrtou částí algoritmu. V té se rozhoduje podle nastavení výstupních příznaků *LinkManageru* a přítomnosti textu mezi odkazy, jestli se jedná o dělený odkaz.

Proměnná ***space*** se nastaví pokud před fragmentem byl bílý znak. Pokud je tato proměnná nastavena, vložíme mezeru před přidání nového textu k neprázdnému obsahu proměnné *linkText*. Následně proměnnou *space* vynuluje.

Výsledný algoritmus je sestaven v následujících bodech:

(1) i := 0;
(2) if (*arrayFragment[i]* není platný) then ukonči algoritmus;
(3) if (*arrayFragment[i]* předcházel bílý znak) then *space* := true;
(4) vlož *arrayFragment[i]* do *LinkManageru*;
(5) if (*LinkManager.form* = RELATIVE or *LinkManager.form* = ABSOLUTE)
then pokračuj bodem (7);
(6) *i* := *i* + 1; pokračuj bodem (2);
(7) *beginText* := *i*; *linkText* := "";
(8) *i* := *i* + 1;
(9) if (*arrayFragment[i]* není platný) then *endText* := *i* a pokračuj bodem (14);
(10) if (*arrayFragment[i]* předcházel bílý znak) then *space* := true;
(11) vlož *arrayFragment[i]* do *LinkManageru*;
(12) if (*LinkManager.form* = RELATIVE or *LinkManager.form* = ABSOLUTE or *LinkManager.form* = END)
then *endText* := *i* a pokračuj bodem (14);
(13) pokračuj bodem (8);
(14) *linkAddress* := *LinkManageru.newAddress*;
(15) if (*linkText* <> "" and *space* = true) then *linkText* := *linkText* + " ";
(16) linkText := linkText + text mezi pozicemi uloženými v proměnných *beginText* a *endText*; *space* := false; *i* = *endText*;
(17) if (*arrayFragment[i]* není platný) then uložení obsahu proměnných *linkAddress* a *linkText* do databáze a ukonči algoritmus;
(18) if (*arrayFragment[i]* předcházel bílý znak) then *space* := true;
(19) vlož *arrayFragment[i]* do *LinkManageru*;
(20) if ((*LinkManager.form* = RELATIVE or *LinkManager.form* = ABSOLUTE) and *LinkManager.identical* = YES) then pokračuj bodem (8);
(21) if ((*LinkManager.form* = RELATIVE or *LinkManager.form* = ABSOLUTE) and *LinkManager.identical* = NO) then pokračuj bodem (23);
(22) if (*arrayFragment[i]* tvoří text) then pokračuj bodem (23)
else pokračuj bodem (8);
(23) ulož obsah *linkText* a *linkAddress* do databáze a pokračuj bodem (2);

7.2 Algoritmus aktualizace odkazů

Algoritmus aktualizace odkazů se téměř shoduje s algoritmem ukládání odkazů. Také u tohoto algoritmu použijeme správu adres (LinkManager) k rozpoznání tvaru a podobnosti adres. Fragmenty jsou opět uloženy v ***arrayFragment*** a proměnnou obsahující aktuální pozici právě zpracovávaného fragmentu označíme ***j***. Odlišnost spočívá pouze v tom., že je nutné si pamatovat počáteční (***beginLink***) a koncovou (***endLink***) pozici celého odkazu a v bodech (16) a (22), kde ukládání do databáze nahradíme aktualizací.

Počáteční a koncová pozice celého odkazu odpovídá, v případě odkazu tvořeného jedním elementem, pozici počátečního tagu a pozici před ukončením tohoto elementu odkazu. V případě, že se jedná o dělený odkaz, počáteční pozici odpovídá pozice počátečního

tagu prvního elementu a koncová pozice odpovídá pozici před ukončením posledního elementu tvořícího odkaz. Výsledný algoritmus aktualizace je sestaven v následujících bodech:

(1) i := 0;
(2) if (*arrayFragment[i]* není platný) then ukonči algoritmus;
(3) if (*arrayFragment[i]* předcházel bílý znak) then *space* := true;
(4) vlož *arrayFragment[i]* do *LinkManageru*;
(5) if (*LinkManager.form* = RELATIVE or *LinkManager.form* = ABSOLUTE)
then pokračuj bodem (7);
(6) *i* := *i* + 1; pokračuj bodem (2);
(7) *beginText* := *i*; *linkText* := "";
(8) *i* := *i* + 1;
(9) if (*arrayFragment[i]* není platný) then *endText* := *i* – 1 a pokračuj bodem (13);
(10) if (*arrayFragment[i]* předcházel bílý znak) then *space* := true;
(11) vlož *arrayFragment[i]* do *LinkManageru*;
(12) if (*LinkManager.form* = RELATIVE or *LinkManager.form* = ABSOLUTE or *LinkManager.form* = END)
then *endText* := *i* a pokračuj bodem (13);
(13) pokračuj bodem (7);
(14) *linkAddress* := *LinkManageru.newAddress*;
(15) if (*linkText* <> "" and *space* = true) then *linkText* := *linkText* + " ";
(16) linkText := linkText + text mezi pozicemi uloženými v proměnných *beginText* a *endText*; *space* := false; *i* = *endText*;
(17) if (*arrayFragment[i]* není platný) then ulož obsah proměnných *linkAddress* a *linkText* do databáze a ukonči algoritmus;
(18) if (*arrayFragment[i]* předcházel bílý znak) then *space* := true;
(19) Vložíme *arrayFragment[i]* do *LinkManageru*;
(20) if ((*LinkManager.form* = RELATIVE or *LinkManager.form* = ABSOLUTE) and *LinkManager.identical* = YES) then pokračuj bodem (8);
(21) if ((*LinkManager.form* = RELATIVE or *LinkManager.form* = ABSOLUTE) and *LinkManager.identical* = NO) then pokračuj bodem (23);
(22) if (*arrayFragment[i]* tvoří text) then pokračuj bodem (23)
else pokračuj bodem (8);
(23) ulož obsah *linkText* a *linkAddress* do databáze a pokračuj bodem (2);

Aktualizace se provádí pouze tehdy, odpovídá-li text z dokumentu jiné adrese než jaká je uložená v databázi. Samotná aktualizace se uskuteční nahrazováním hodnot přiřazovaným atributu *href* u každého počátečního tagu aktivních elementů *A*, které tvoří právě zpracovávaný odkaz. Tento postup nahrazuje v algoritmu volání ***UpdateLink***. Parametry této aktualizační funkce jsou: text (*linkText*) a adresa (*linkAddress*) odkazu z dokumentu, počáteční (*beginLink*) a koncová (*endLink*) pozice celého linku.

Postup při *UpdateLink* není složitý. Na začátku porovnáme adresu, která v databázi odpovídá textu odkazu v *linkText*, s adresou z dokumentu (*linkAddress*). Pokud se shodují, nedojde k aktualizaci a tato část algoritmu může být ukončena. Pokud se adresy neshodují, začneme procházet jednotlivé fragmenty od pozice *beginLink* do *endLink*. Pomocí *LinkManageru* může identifikovat počáteční tagy aktivních elementů. Když bude příznak *form* nastaven na hodnotu ABSOLUTE nebo RELATIVE, stačí nahradit hodnotu přiřazenou atributu href novou adresou z databáze.

7.3 Algoritmus vytváření odkazů

Pro tento algoritmus potřebujeme předzpracovaný dokument bez původních odkazů. Proto ho musí předcházet proces ukládání odkazů v upraveném tvaru, který místo ukládání do databáze bude pouze „vyřezávat“ odkazy.

Na vstupu algoritmu tak dostáváme předzpracovaný dokument bez odkazů a data z databáze. Pro generování přípustných tvarů textů potřebuje znát typ textu a typ jednotlivých slov. Tyto data jsou uložena pouze v databázi. Z tohoto důvodu není použití „hrubé síly“ počítače pro tento algoritmus ideální. Složitost takového algoritmu by byla závislá na počtu odkazů uložených v databázi. Tento počet může být (a také často bude) mnohonásobně větší než počet nevyznačených odkazů v dokumentu. Druhým nedostatkem takového postupu je nepoměr mezi počtem porovnání, která vedou a nevedou k nalezení nového odkazu. I kdyby nám někdo řekl, jestli odkaz v dokumentu nalezneme nebo ne, všechna porovnání až na jedno budou s negativním výsledkem. Nejde ani tak o možnost uvedení jiného tvaru textu odkazu, ale o to, že neoznačený text odkazu může být kdekoliv v dokumentu. Z tohoto důvodu navrhneme a sestavíme nový algoritmus, který bude hledat odkazy rychleji a hlavně účinněji.

Nejprve částečně odstraníme druhý nedostatek. V dokumentu můžeme najít pouze texty odkazy uložené v databázi. Tyto texty se skládají ze slov, která jsou také uložena v databázi. Proto si v prvním průchodu dokumentem označíme ty slova nebo jejich zkratky, která jsou uložena v databázi. Tím rozdělíme dokument na části, které mohou a nemohou obsahovat odkaz.

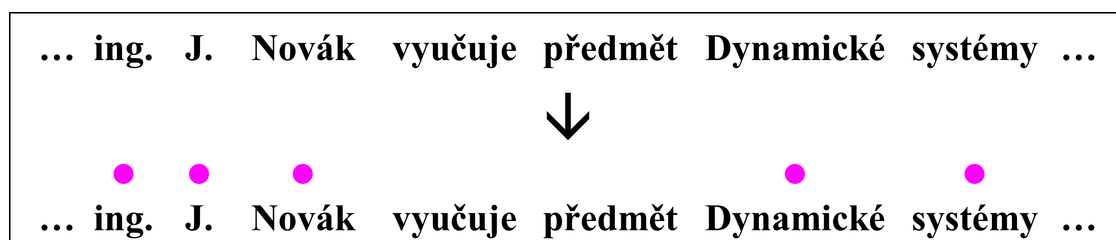
Mějme například v databázi uložena tato data:

Jiří Novák	*http://www.jiri.novak.cz/*,
ing. Jan Novák	mailto://jan@novak.cz,
Dynamické systémy	*http://www.ds.cz/*.

Situace v databázi po uložení těchto dat je znázorněna na obr. 7.2. Začneme hledat odkazy v dokumentu, který obsahuje mimo jiné i tento text:

ing. J. Novák vyučuje předmět Dynamické systémy.

Po označení slov nebo jejich zkratek, která jsou uložena v databázi, dostaneme situaci znázorněnou na obr. 7.1. V textu jsou zvýrazněna slova, která databáze obsahuje a pouze tyto označené části textu mohou tvořit text odkazu.



Obr.7.1. Označení slov, která databáze obsahuje.

ADRESA	
číslo adresy	**adresa**
1	http://www.jiri.novak.cz/
2	mailto://jan@novak.cz
3	http://www.ds.cz/

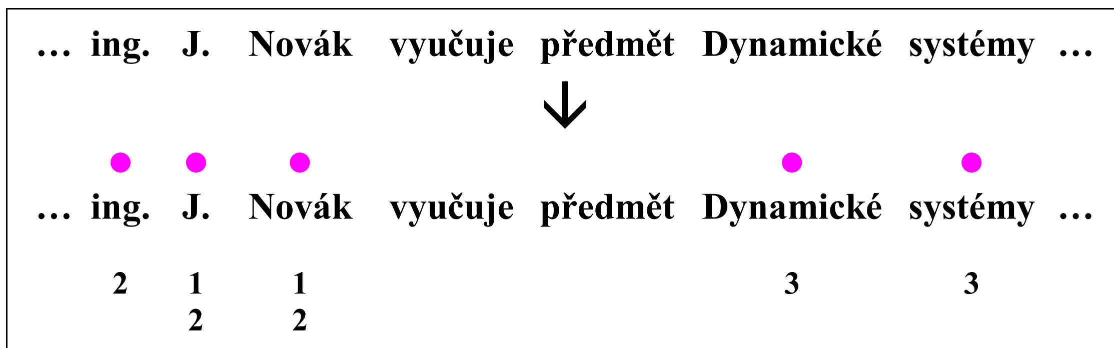
ODKAZ	
číslo textu	**číslo adresy**
1	1
2	2
3	3

TEXT	
číslo textu	**typ textu**
1	JMÉNO_OSOBY
2	JMÉNO_OSOBY
3	NÁZEV

ČÁST TEXTU			
číslo textu	**číslo slova**	**typ slova**	**pořadí**
1	1	KŘESTNÍ_JMÉNO	1
1	2	PŘÍJMENÍ	1
2	3	TITUL_PŘED_JMÉNEM	1
2	4	KŘESTNÍ_JMÉNO	1
2	2	PŘÍJMENÍ	1
3	5	ČÁST_NÁZVU	1
3	6	ČÁST_NÁZVU	2

SLOVO	
číslo slova	**slovo**
1	Jiří
2	Novák
3	ing.
4	Jan
5	Dynamické
6	systémy

Obr.7.2. Obsah databáze po uložení odkazů.



Obr.7.3. Označení slov a jejich zkratek uložených v databázi, spolu s připojeným seznamem čísel adres, ke kterým se jednotlivá slova vážou.

Samostatně toto rozdělení vede k minimální úspoře, protože jsme stále nuceni testovat obsah celé databáze. Provedeme proto druhou úvahu: každé slovo se váže k omezené skupině textů a každý text se váže k omezené skupině adres. Při prvním průchodu dokumentem slova nejenom označíme, ale u každého uložíme seznam obsahující čísla adres, ke kterým se váže. Dostaneme tak dokument rozdělený na části a u každé takové části známe jaké odkazy ji mohou tvořit.

Mějme stejnou situaci jako v předchozím příkladě. K jednotlivým slovům přiřadíme seznam čísel adres, jak vidíme na obr. 7.3. Při následném testování jednotlivých částí víme, že u první části stačí testovat dvě adresy a u druhé pouze jednu adresu.

Každá část může být tvořena žádným, jedním nebo více odkazy. Zatím jsme vůbec neuvažovali o tom, jestli záleží nebo nezáleží na pořadí v jakém texty testujeme. Když se podíváme na obr. 7.3. vidíme, že záleží. V tomto případě je jméno „*Jiří Novák*“ v databázi uloženo před jménem „*ing. Jan Novák*“. Nestanovili jsme žádná další kritéria, a proto se bude jako první testovat text „*Jiří Novák*“. Tento text se shoduje s textem „*J. Novák*“, a pokud bychom se s tímto výsledkem spokojili, došlo by k chybnému označení.

Můžeme otestovat všechny možnosti a dát uživateli na výběr. I tak potřebujeme nějaké kritérium, podle kterého budou tyto možnosti seřazeny, aby si uživatel mohl vybrat od té nejpravděpodobněji správné možnosti. Dokonce můžeme jednotlivé možnosti seřadit ještě před samotným testováním. Stačí jako první kritérium použít počáteční pozici, na které text označený stejným číslem adresy začíná a jako druhé doplňující kritérium použít délku tohoto textu.

ing.	J.	Novák
1	2	1
3	5	3
5	6	6
7	8	9
9	10	12
12	12	13
	14	

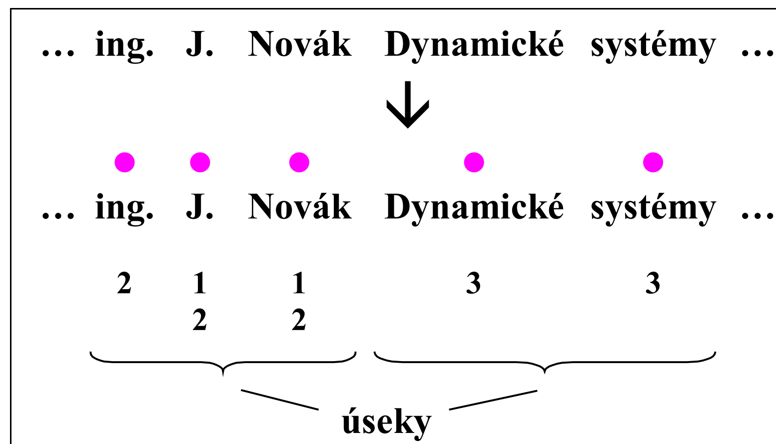
Seznam možností před setříděním

číslo adresy	1	3	5	7	9	12	2	6	8	10	14	1	3	9	13
poč. pozice	1	1	1	1	1	1	2	2	2	2	2	3	3	3	3
délka	1	1	2	1	1	3	1	2	1	1	1	1	1	1	1

Seznam možností po setřídění

číslo adresy	**12**	5	1	3	7	9	**6**	2	8	10	14	**1**	**3**	**9**	**13**
poč. pozice	**1**	1	1	1	1	1	**2**	2	2	2	2	**3**	**3**	**3**	**3**
délka	**3**	2	1	1	1	1	**2**	1	1	1	1	**1**	**1**	**1**	**1**

Obr.7.4. Příklad označení slov u jména „*ing. J. Novák*“ čísly adres. Připojené tabulky obsahují seznam možností před setřídění a po setřídění podle počáteční pozice a délky textu. V druhé tabulce jsou zvýrazněny možnosti, které mohou tvořit přípustnou kombinaci.



Obr.7.5. Rozdělení textu na úseky.

Celá situace je zobrazena na obr. 7.4. Uvedené seznamy slouží pouze jako příklad. Kroužkem je označena nejpravděpodobnější možnost a obdélníčkem ostatní možnosti s délkou větší než jedna. Připojená tabulka obsahuje seznam možností před setříděním a po setřídění podle počáteční pozice a délky textu. Celkem je v ní uvedeno 15 možností, ale pouze 6 z nich může tvořit přípustnou kombinaci pro jméno osoby, které musí obsahovat příjmení (v připojené tabulce s možnostmi po setřídění jsou vyznačeny tučným písmem).

Možnosti, které jsou opravdu přípustné, rozpoznáme až po otestování. Například se může stát, že některé z označených příjmení „*Novák*" na obr. 7.4 je ve skutečnosti pouze nalezená ‚zkratka' od příjmení „*Nováček*". Když začneme generovat a testovat jednotlivé přípustné kombinace od jména „*… Nováček* ", zjistíme, že „*Novák*" netvoří přípustnou kombinaci, a proto ji vyřadíme z dalšího zpracování.

Před sestavením algoritmu ještě jedna poznámka k označování slov a dělení textu na části. Na obr. 7.5 vidíme označenou část textu, kterou tvoří dva samostatné celky – úseky. Texty odkazů nebudeme hledat v celé označené části, ale vždy jen v jednom úseku. I tak může každý úsek obsahovat více odkazů, ale sníží se počet rozhodnutí, které bude muset udělat uživatel. Úsek definujeme jako skupinu slov, po která platí tato pravidla:

1) každé slovo ve skupině má neprázdný seznam čísel adres, ke kterým se váže,
2) dvě sousední slova patří do jedné skupiny (tvoří jeden úsek), pokud mají alespoň jedno stejné číslo adresy ve svých seznamech (seznamy mají neprázdný průnik),
3) první slovo ve skupině nemusí předcházet žádné slovo,
4) poslední slovo ve skupině nemusí následovat žádné slovo.

Nyní můžeme přistoupit k sestavení algoritmu. Slova máme uložená v proměnné *arrayWord*. Jedná se o pole objektů, které jsme částečně navrhli v části 4.3.2. Data těchto objektů (počáteční a koncová pozice v původním dokumentu a samotný text slova) rozšíříme o seznam adres, ke kterým se slovo váže. Aktuální pozici v poli *arrayWord* obsahuje proměnná *i*. K jednotlivým objektům budeme přistupovat indexací *arrayWord[i]*. U každého objektu pak můžeme přistoupit jednak k textu slova – *arrayWord[i].text* a také k seznamu adres – *arrayWord[i].addressList*.

Nejprve, v bodech (1) až (5), ke každému slovu přiřadíme seznam adres, ke kterým se váže. Následně začneme třídit a testovat možnosti v jednotlivých označených úsecích. Nejprve nalezneme počáteční (*beginSection*) a koncovou (*endSection*) pozici prvního úseku od aktuální pozice (body (7) až (13)). Jednotlivé seznamy slov v tomto úseku pak skombinujeme do pole *section*, které setřídíme podle zvolených kritérií (bod (14) a (15)). Pokud najdeme shodný text, označíme ho v textu jako odkaz a pokračujeme s tříděním a hledáním za jeho koncovou pozicí.

Není-li v úseku nalezen žádný text, pokračujeme zpracováním dalšího úseku. Algoritmus ukončíme, když jsme prohledali všechny úseky. Výsledný algoritmus vytváření odkazů je sestaven v následujících bodech:

(1) *i* := 0;
(2) if (*arrayWord[i]* není dostupný) then pokračuj bodem (6);
(3) *arrayWord[i].addressList* := všechna čísla adres, která se váží ke slovu *arrayWord[i].text*;
(4) *i* := *i* + 1;
(5) pokračuj bodem (2);
(6) *i* := 0;
(7) if (*arrayWord[i]* není dostupný) then ukonči algoritmus;
(8) if (*arrayWord[i].addressList* není prázdný)
then *beginSection* := *i* a pokračuj bodem (11);
(9) *i* := *i* + 1;
(10) pokračuj bodem (7);
(11) *i* := *i* + 1;
(12) if (*arrayWord[i]* není dostupný or *arrayWord[i].addressList* je prázdný
or *arrayWord[i]* má prázdný průnik s *arrayWord[i - 1]*)
then *endSection* := *i* - 1 a pokračuj bodem (14);
(13) pokračuj bodem (12);
(14) z jednotlivých seznamů *arrayWord[x].addressList* mezi pozicemi *beginSection* a *endSection* včetně vytvoř pole *section*;
(15) setřiď pole *section* podle počáteční pozice a délky textu.
(16) otestuj všechny možnosti uvedené v poli *section*, a pokud některá netvoří přípustnou kombinaci, vyřaď ji;
(17) if (*section* je prázdné) then pokračuj bodem (21);
(18) if (*section* obsahuje jednu možnost)
then označ text jako odkaz a pokračuj bodem (21);
(19) nabídni uživateli možnosti k vybrání;
(20) pokud vybral uživatel jednu možnost, označ text jako odkaz, *beginSection* nastav na poslední objekt tohoto textu a pokračuj bodem (14);
(21) *i* := *endSection* + 1; pokračuj bodem (7);

7.4 Optimalizace algoritmů

Algoritmy jsou navrženy, ale to nám nebrání v možnosti jejich optimalizace. Největší časová náročnost se dá předpokládat u dotazů do databáze. Ta může být ještě větší, pokud se jedná o vzdálenou databázi. Optimalizovat lze počet přístupů do databáze.

V procesu vytváření odkazů testujeme každé slovo i na to, jestli se nejedná o zkratku. Tento problém je možné řešit jedním dotazem, ale vždy jen pro jedno slovo. Před spuštěním procesu můžeme jedním dotazem z databáze získat všechny slova a čísla adres, ke kterým se vážou. Uložením do paměti vytvoříme *cache*, která bude rychlejší a můžeme jí použít pro zpracování více souborů. Pokud používáme stejnou databázi, může si tuto cache, uložit na disk a při dalším spuštění opět načíst.

Stejným principem můžeme vytvořit cache pro kterákoliv data z databáze. Pro všechny tři procesy lze například vytvořit cache pro adresy. V aplikaci je použita cache pouze pro slova s indexy adres a cache pro adresy.

Další možnost optimalizace se týká generování a následného testování přípustných tvarů v procesu vytváření odkazů. Přidáme-li ke každému seznamu adres u jednotlivých slov o jaký typ slova a jaký typ podobnosti se jedná, můžeme těchto informací využít a vytvořit filtr. Tím se dají před samotným testováním vyřadit například ty možnosti, které nemohou tvořit přípustný tvar (obr. 7.4). Dále můžeme vyřadit z dalšího zpracování ty úseky, které tvoří jedna ‚zkratka'. Tato možnost je implementována v aplikaci a výrazně tak snižuje počet porovnání.

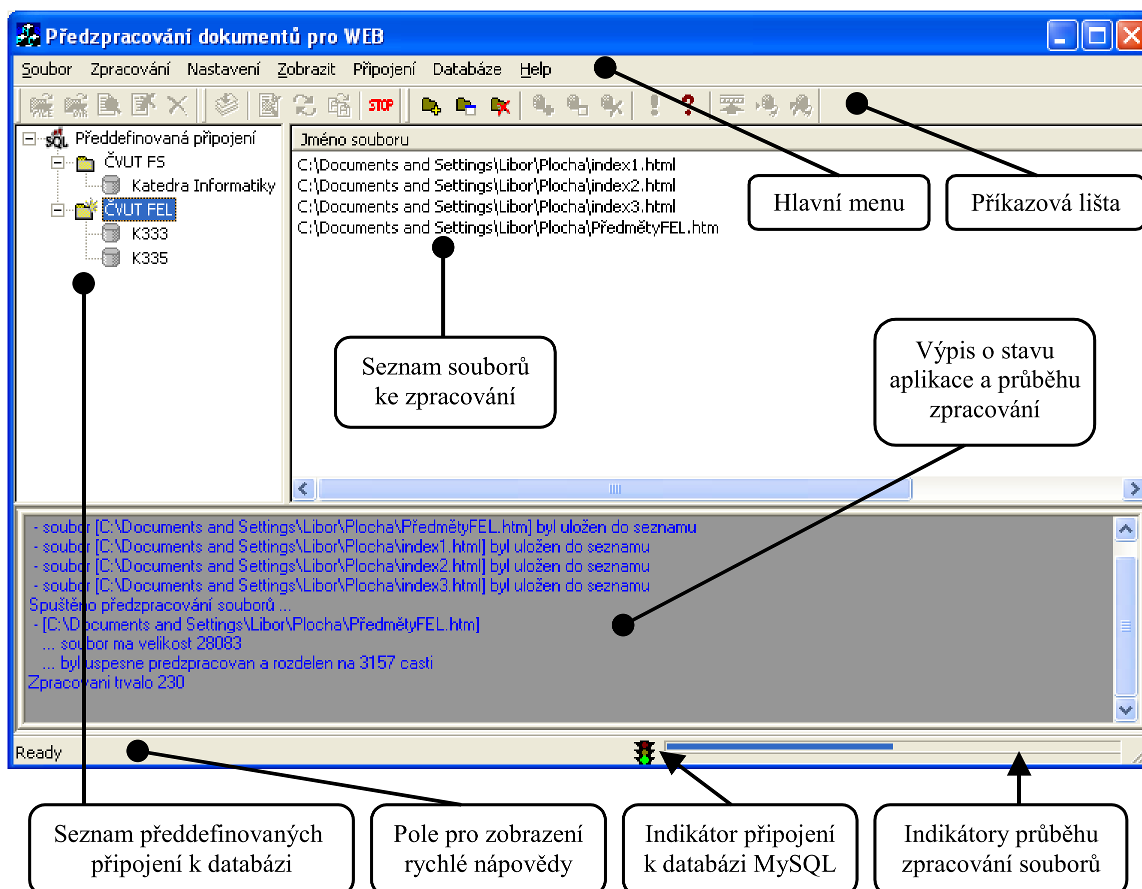
8. Experimenty

V této kapitole se nejprve zmíníme o samotné aplikaci, uvedeme její nejpodstatnější rysy. Dále se v kapitole zmíníme o provedených experimentech a jejich hodnocení.

8.1 Implementace

Aplikace byla vytvořena v prostředí Microsoft Visual C++ 6.0 s využitím knihovny MFC 6.0 (Microsoft Foundation Class) jako SDI aplikace (Single Document Interface) architekturou dokument - pohled. Hlavní okno aplikace je zobrazeno na obr. 8.1. Její hlavní rysy lze shrnout do těchto bodů:

- umožňuje otevření jednotlivých souborů nebo celých adresářů,
- seznam zpracovaných souborů lze dále upravovat,
- u každého souboru je možné vybrat typ zpracování,
- u nalezených odkazů je možné zpracovat a upravit jak texty, tak adresy odkazů,
- aplikace umožňuje připojit se na vzdálenou databázi,
- dále umožňuje vytvořit seznam předdefinovaných připojení k databázi,
- vytváří rychlou cache pro slova a adresy, které se po skončení práce automaticky uloží na disk a při spuštění akce se znovu nahrají z disku do paměti,
- uživatel je průběžně informován o stavu zpracování a jednotlivých krocích a rozhodnutích, které provedla aplikace nebo on sám,
- zprávy je možné ukládat do LOG souboru.



Obr.8.1. Hlavní okno aplikace s popisem jednotlivých částí

8.2 Testy

Aplikace byla testována na příkladech uvedených v této práci a na internetových stránkách Katedry řídicí techniky[4]. Z jejich hlavních stránek byla získána data (texty a adresy odkazů), která byla použita na úpravu dokumentů na lokálních počítačích. Testováno bylo i připojení na vzdálenou databázi. Všechny testy dopadly úspěšně, a proto lze konstatovat, že navržené algoritmy jsou funkční.

[4] http://dce.felk.cvut.cz/

9. Závěr

Úkolem této práce bylo vytvoření postupů, které by usnadnili vytváření nebo aktualizaci odkazů v dokumentech prezentovaných prostřednictvím internetu. Data (adresy a texty odkazů) pro tyto úpravy se měli získávat z dokumentů, které měli tyto odkazy vyznačeny, a ukládat do databáze.

Rozborem základních úkolů, spolu s uvedením několika příkladů a se znalostí jazyka HTML, jsme došli k závěru, že obecný HTML dokument není vhodný pro přímé zpracování. Z tohoto důvodu jsme před samotné zpracování dokumentu vložili předzpracování.

Předzpracování mělo za úkol umožnit přímé zpracování textu dokumentu a zachování pro nás významných elementů (např. elementy pro tvorbu odkazu, seznamu, tabulky atd.). Při jeho návrhu jsme vyšli z jazyka HTML. Zdrojový kód dokumentu jsme v první fázi rozdělili na tagy, entity a text. Souhrnně jsme tyto části nazvali fragmenty, které jsme reprezentovali jako objekty tříd *Tag*, *Entita* a *Word*. Bílé znaky v původním zdrojovém kódu jsme v těchto třídách nahradili indikátorem bílého znaku, který je nastaven, pokud byl před fragmentem jeden nebo více bílých znaků. Dále objekty obsahovaly atributy pro uložení počáteční a koncové pozice v původním dokumentu, samotný fragment, jeho typ a indikaci povolení dalšího zpracování. Takto vzniklým fragmentům jsme v druhé fázi přiřadili odpovídající význam (např. u entit jsme nalezli jejich odpovídají znak). V poslední fázi jsme fragmenty podle zvolených kritérií sloučili do slov.

Předzpracovaný dokument nám umožnil jeho přímé zpracování a stal se vstupem základních úkolů této práce. Jako první z těchto úkolů jsme analyzovali problémy spojené s vytvářením odkazů. Nejprve jsme určili, že pro tuto úlohu jsou důležité elementy pro tvorbu seznamů, tabulek a formátování a úpravu textu. Ke znalosti členění dokumentu nám stačí tyto elementy (jejich počáteční a koncové tagy) umět v rámci předzpracování rozpoznat. Dále jsme se zabývali, jak hledat texty v různých tvarech a zjistili jsme, že je můžeme rozdělit do tří skupin: na jména osoby, názvy a zkratkové názvy. Při zkoumání jejich vlastností jsme došli k závěru, že musíme rozdělovat do skupin i jejich jednotlivá slova. Tyto skupiny jsme nazvali: tituly před jménem, křestní jména, příjmení, tituly za jménem, části názvu a části zkratkového názvu. Pro takto rozdělené texty a slova jsme navrhli gramatiky pro generování přípustných tvarů, které lze následně použít při porovnávání. Nakonec jsme pro tuto úlohu navrhli algoritmus, který výrazně zmenšil počet porovnání.

Data z databáze, která se v úloze vytváření odkazů používají ke generování přípustných tvarů, se získávají v úloze ukládání odkazů. Tyto dvě úlohy jsou k sobě inverzní a navzájem se ovlivňují. Při analýze ukládání odkazů jsme nejprve určili, co považujeme za počáteční a koncový tag elementu *A*. Dále jsme stanovili, jakým způsobem budeme v předzpracovaném dokumentu odkazy hledat a zpracovávat texty a adresy a to i pro dělené odkazy (odkaz tvořený více elementy *A*). Pro tuto úlohu jsme také navrhli algoritmus, který se skládal ze dvou částí: algoritmu pro zpracování adres a algoritmu pro samotné nalezení a zpracování odkazu.

Poslední třetí úloha – aktualizace odkazů je částečným spojením obou předchozích úloh a je složena z částí, jejichž řešení jsme navrhli při jejich analýze. Z tohoto důvodu bylo nutné pouze stanovit, jakým způsobem se bude vytvářet nový soubor. Aktualizace odkazů musela totiž počítat s možností výskytu děleného odkazu a tedy i s jeho případnou aktualizací. I pro tuto úlohu byl navržen algoritmus, který se také skládá ze dvou částí: stejného algoritmu pro zpracování adres a algoritmu pro aktualizaci odkazů.

Všechna data se ukládají do databáze, která byla také navrhnuta. Data obsahují mimo textu a adresy odkazu ještě typ textu odkazu a pořadí a typ jeho jednotlivých slov. Výsledná databáze obsahuje 5 tabulek.

Jednotlivé body rozšířeného zadání tak byly splněny. Takto navržená aplikace umožňuje vytvářet a aktualizovat odkazy i v případě, že v databázi a v dokumentu jsou v různém

tvaru. Data se získávají z dokumentů, které tyto odkazy již mají vyznačené. Nové vytvořené dokumenty jsou tvořeny jako původní s nově vyznačenými nebo aktualizovanými odkazy, a proto je zachována původní struktura dokumentu. Také je možné připojení k lokální nebo vzdálené databázi.

Navržené algoritmy, například pro správu adres nebo předzpracování, je možné využít i v jiných aplikacích. Při práci s textem dokumentu (např. při textové analýze, porovnání obsahu dvou dokumentů atd.) potřebujeme znát pouze strukturu a text dokumentu. U HTML dokumentů nám tyto informace předzpracování umožní jednoduše a opakovaně získat.

Další rozšíření této aplikace je možné například v testování platnosti odkazu. To by mohlo vést na doplňující aplikaci typu *service* s nízkou prioritou, která by pouze průběžně testovala dostupnost adres uložených v databázi. Možnost rozšíření je i u jednotlivých procesů. Vložením jisté míry „inteligence“ například do procesu ukládání odkazů by mohlo vést až na plně automatické rozpoznání typu slov a textu.

Literatura

[1] *W3C HTML Validation Service*. URL:<http://www.validator.w3.org> [cit.2001-10-15].

[2] Dave Ragget, Amaud Le Hors a Ian Jacobs. *HTML 4.0 Specification*. W3C, 1998. URL:<http://www.w3.org/TR/REC-html40> [cit.2001-10-15].

[3] Dave Ragget. *HTML 3.2 Specification*. W3C, 1997 URL:<http://www.w3.org/TR/REC-html32.html> [cit.2001-10-15].

[4] Dave Ragget, Amaud Le Hors a Ian Jacobs. *HTML 4.01 Specification*. W3C, 1999. URL:<http://www.w3.org/TR/1999/REC-html401-19991224> [cit.2001-10-15].

[5] Zdeněk Hlavsa a kolektiv. *Pravidla českého pravopisu*. Academia Praha, 2001.

[6] Michal Chytil. *Automaty a gramatiky*. SNTL Praha, 1984, 336 stran.

[7] Jiří Kosek. *HTML – tvorba dokonalých WWW stránek*. Grada Publishing, 1998, 296 stran.

[8] Roman Barták. *Automaty a gramatiky*. URL:<http://ktiml.mff.cuni..cz/~bartak/automaty/> [cit.2002-02-26].

[9] *URL Moniker Reference*. URL:<http://msdn.microsoft.com/workshop/> [cit.2001-11-13].

[10] Stanislav Racek, Martin Kvoch. *Třídy a objekty v C++*. Nakladatelství Kopp, 1998, 224 stran.

[11] Richard Šusta. *Programování pro řízení ve Windows*. Vydavatelství ČVUT, Praha, 1999, 220 stran.

[12] James Pitkow, Kipp Jones. *Towards an Intelligent Publishing Environment*. Proceedings of the Third International World Wide Web Conference, April 1995. URL:<http://www.igd.fhg.de/archive/1995_www95/proceedings/papers/72/publish/publishing.html> [cit.2001-12-05].

[13] Kipp Jones, James Pitkow. *Atlas: Supporting the Web with a Distributed Hyperlink Database*. 1996 URL:<http://www.cc.gatech.edu/grads/j/Kipp.Jones/atlas/boston/atlas.html> [cit.2001-12-05].

[14] James Pitkow. *HTML_Analyzer*. 1995 URL:<http://www.cc.gatech.edu/pitkow/html_analyzer/index.html> [cit.2001-12-05].

[15] Bořivoj Melichar. *Gramatiky a jazyky*. Vydavatelství ČVUT, Praha, 1982, 185 stran.

[16] Frank Kappe a jiní. *Hyper-G: A New Tool for Distributed Hypermedia*. URL<:ftp://iicm.tu-graz.ac.at/pub/Hyper-G/doc/> [cit.2001-12-16].

[17] McGuire, J. *EIT-Link-Verifier-Robot*. URL:<http://wsk.eit.com/wsk/dist/doc/admin/Webtest/verify_links.html> [cit.2001-12-16].

[18] Jaroslav Pokorný, Ivan Halaška. Databázové systémy. Vydavatelství ČVUT, Praha, 1999, 146 stran.

[19] Pavel Herout. *Učebnice jazyka C*. Nakladatelství Kopp, 1997, 264 stran.