Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Michal Vykypěl

Testovací platforma pro Webové roboty

Katedra softwarového inženýrství
Vedoucí diplomové práce: *RNDr. Leo Galamboš, Ph.D.*
Studijní program: *Informatika*

Na tomto místě bych rád poděkoval vedoucímu své diplomové práce RNDr. Leo Galambošovi, Ph.D., za rady, připomínky a náměty.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 5. srpna 2008

Michal Vykypěl

Obsah

Název práce: *Testovací platforma pro Webové roboty*

Autor: *Michal Vykypěl*

Katedra: *Katedra softwarového inženýrství*

Vedoucí diplomové práce: *RNDr. Leo Galamboš, Ph.D.*

e-mail vedoucího: *leo.galambos@mff.cuni.cz*

Abstrakt: *Webový robot je důležitou částí pro získávání informací na Internetu. V této práci je navrženo a implementováno testovací prostředí pro webového robota, sloužící k ověření efektivity stahovacích strategií. Rozhraním je skriptovací jazyk, ve kterém je popsán jeho průchod webovým prostorem.*

Klíčová slova: *Web, Robot, Simulace webu, Testování robota*

Title: *Test bed for Web robots*

Author: *Michal Vykypěl*

Department: *Department of Software Engineering*

Supervisor: *RNDr. Leo Galamboš, Ph.D.*

Supervisor's e-mail address: *leo.galambos@mff.cuni.cz*

Abstract: *Web robot is important part of mining information from the Internet. The aim of this work is to design and implement test bed for web robot, to verify effectiveness of downloading strategies. Interface is script language, where user can define instructions for web robot.*

Keywords: *Web, Crawler, Web simulation, Testing of robot*

Kapitola 1

Úvod

Webový robot je program, který stahuje stránky z webu, aby zjistil jejich obsah. Potřeba webových robotů vznikla v době rozrůstání webového prostoru. Se zvětšujícím se počtem stránek na Internetu vznikl požadavek mít přehled o jeho obsahu, aby bylo možné uspokojit uživatele, hledající určitou informaci.

Robot pracuje nad protokolem http, má určitou množinu URL, které navštíví a stáhne jejich obsah. Ze stránek extrahuje linky a poté se rozhoduje, které další uzly (webové stránky) navštíví. Vzhledem k velikosti Internetu a k jeho růstu v čase, není možné aby navštívil všechny stránky, proto klíčovou roli hrají algoritmy pro výběr relevatních stránek. Jednou z částí této práce je umožnit tvůrci webového robota otestovat si svojí stahovací strategii, tedy algoritmus výběru stránek jejichž obsah robot stánhne.Roboti jsou software těžko testovatelný v reálném provozu, vzhledem k tomu, že potřebují zabírat značnou síťovou kapacitu. Toto vadí zejména provozovatelům webů, kteří snadno poznají, že jejich stránky byly navštěvovány robotem a mohou mu případně i zakázat další stahování.

1.1 Cíl práce

Cílem této práce je vytvořit pro programátora webového robota prostředí, které bude simulovat dostatečně veliký webový prostor a poskytnout mu jednak rozhraní, přes které může robota testovat a za druhé rozhraní, které slouží k ověření určité stahovací strategie (bez účasti robota).

Aby takovéto testování mělo smysl, je potřeba jej provádět nad velkými daty. Toto množství dat není potřeba stahovat, v části diplomové práce, která se zabývá generováním webového grafu jsou rozebrány vlastnosti webového prostoru, které byly zjišteny nad reálnými staženými daty. Základním požadavkem na simulaci je aby tyto požadavky splňovala. Ke generování jsou použity známé modely a v experimentálná části práce jsou testovány, zda mají vlastnosti, které má reálný webový prostor.

Pro vytvoření pro robota transparentího testovacího prostředí bylo potřeba implementovat simulovaný DNS a HTTP server, který odpovídá na DNS dotazy a požadavky na stažení stránek, které robot posílá. Robota není pro běh nad simulovaným webovým prostorem nutné nijak upravovat a na rozdíl od reálného stahování obsahu webu nezatěžuje síťovou kapacitu a veškerá komunikace se odehrává v rámci jedné stanice.

Druhým z rozhraní je skriptovací jazyk, ten slouží k ověření efektivity vybrané stahovací strategie - programátorovi proto stačí, když si v tomto jazyce napíše příkazy pro průchod webovým prostorem a na výstup dostane seznam stránek a pořadí, v jakém byly staženy.

Kapitola 2

Webový graf

Webový graf $G = (V, E)$ je orientovaný graf, kde jednotlivé uzly představují webové stránky a hrany mezi nimi představují odkazy mezi stránkami. Jak bude ukázáno dále (podle měření provedených na reálných datech), webový graf neodpovídá modelu náhodného grafu podle Erdős–Rényi. Proto pokud nemáme k dispozici velká reálná data, musíme webový graf vytvořit jiným způsobem. Na grafu budou zkoumány vlastnosti jako vstupní a výstupní stupň uzlu, počet uzlů s danými stupni, bipartitní jádra, kliky a souvislé komponenty.

2.1 Struktury sledované na webovém grafu

Na úvod budou uvedeny definice některých pojmů:

Definice 2.1.1. *Náhodným grafem $G_{n,p}$ rozumíme graf na n vrcholech, ve kterém se hrana e vyskytuje s pravděpodobností p (jedná se o model náhodného grafu, který popsali autoři Paul Erdős, Alfréd Rényi v [PE59]).*

Vstupní a výstupní stupeň uzlu.

Definice 2.1.2. *Vstupním stupněm uzlu u rozumíme počet hran, které končí v uzlu u.* $deg_+(u) = |\{e \in E \mid \exists\, v \in V,\, (v, u) \in E\}|$

Definice 2.1.3. *Výstupním stupněm uzlu u rozumíme počet hran, které začínají v uzlu u.* $deg_-(u) = |\{e \in E \mid \exists\, v \in V,\, (u, v) \in E\}|$

Podle měření se ukazuje, že ve webovém postoru platí:

$P(deg_+(u) = i) = \frac{1}{i^\alpha}$

$P(deg_-(u) = i) = \frac{1}{i^\beta}$

Toto rozložení uzlů matematicky upřesňuje intuitivně zřejmý fakt, že nejvíce je stránek s malým počtem odkazů. V [RK99] se uvádí, že webový graf splňuje uvedené rovnice pro $\alpha = 2.1$, $\beta = 2.38$. Dále je v [RA02] také uvedeno, že průměrný výstupní stupeň vrcholu je přibližně 8.

Požadavkem na generovaný webový graf je tedy splnění uvedených rovností pro nějaké α a β.

Bipartitní jádra.

Definice 2.1.4. *Graf $K_{i,j} = (V, E)$ je bipartitní klika, pokud platí $V = V_1 \bigcup V_2$, $V_1 \bigcap V_2 = \emptyset$ a $E = V_1 \times V_2$. Bipartitní jádro $C_{i,j}$ je graf na $i + j$ vrcholech, který obsahuje $K_{i,j}$ jakožto podgraf.*

Důvod zkoumání bipartitních jader je ten, že ve webovém grafu se vyskytují na rozdíl od náhodného grafu ve větší míře – tedy jejich zvýšený výskyt je vlastnost pro webový prostor charakteristická. Tato vlastnost pro graf vyjadřuje na webu obvyklý jev, kdy existují skupiny stránek, které se věnují jednomu tématu. Tato skupina stránek se odkazuje na stránky se stejným zaměřením mnohem pravděpodobněji, než na jiné náhodné stránky.

Shlukovací koeficient.

Definice 2.1.5. *Shlukovací koeficient C_u je pro uzel u definován jako $\frac{2E_u}{deg(u)\cdot(deg(u)-1)}$, kde E_u je počet hran mezi sousedy vrcholu u.*

Toto číslo vyjadřuje "propojení" sousedů vrcholu u. Pokud je u součástí kliky, potom $C_u = 1$ (všichni sousedi u jsou navzájem propojeni, takže $E_u = \frac{deg(u)\cdot(deg(u)-1)}{2}$.

Kapitola 3

Modely webových grafů a jejich srovnání.

Naším cílem je vytvoření vytvoření grafu, který bude splňovat parametry sledované v předchozí kapitole. Hlavní nevýhodou náhodného modelu je jeho statičnost, to znamená že vzniká v jednom okamžiku, o každé hraně můžeme říci jestli je v grafu $G_{n,p}$, bez ohledu na existenci ostatních hran. Naproti tomu v modelech, podle kterých jsou generovány webové grafy, vznikají hrany a vrcholy v čase. To odráží i typický vznik webové stránky na Internetu. Autor ji vytvoří a přidá linky směrující na již existující stránky. Často tyto linky míří na stránky, které se zabývají stejnou tematikou, jako nově vzniklá webová stránka. Na úvod budou představeny některé modely z hlediska vhodnosti pro použití ke generování grafu webového prostoru.

3.1 Náhodný graf podle ER

Nejdříve bude ukázáno splnění vlastností z přechozí kapitoly v ER grafu. Pokud bychom se blíže podívali na rozložení stupňů uzlů u náhodného grafu, dostaneme vztah $P(deg(u) = i) = \binom{N-1}{i} p^i (1-p)^{N-1-i}$, kde $\binom{N-1}{i}$ je počet všech vrcholů, které lze spojit s u hranou, p^i je pravděpodobnost výskytu i hran a $(1-p)^{N-1-i}$ je pravěpodobnost, že u nebude spojen s více než p hranami. Dostáváme tedy, že distribuce stupňů uzlů není exponenciální ale je polynomiální.

Jednoduchým výpočtem se dále přesvědčíme, že v náhodném grafu je i pravděpodobnost vzniku bipartitních jader velmi malá. Vezměme graf $G_{n,p}$ a jeho bipartitní jádro $C_{i,j}$ na $i+j$ uzlech. Takových jader může být v tomto grafu až $\binom{n}{i+j}$. Mezi těmito uzly musí být $i \cdot j$ hran, vedoucích z uzlů I do uzlů J. Pravděpodobnost existence těchto hran je $p^{i \cdot j}$. Tedy dostáváme vztah $P(C_{i,j}) = \binom{n}{i+j} \cdot p^{(i \cdot j)}$. Pokud bychom tedy chtěli, aby v grafu o velikosti $8 \cdot 10^7$ bylo alespoň jedno jádro velikosti $C_{2,3}$, musela by být pravděpodobnost

$p > 10^{-7}$ (přibližně), pro jádra o velikosti $C_{3,3}$ už by tato pravděpodobnost musela být alespoň 10^{-5} (přibližně). Druhé z čísel ale dává graf s průměrným výstupním stupněm vrcholů 800, což neodpovídá webovému prostoru.

Co se týče shlukování, je situace z hlediska výpočtu jednoduchá, náhodný graf má $C = \frac{avg(deg(u))}{N} = p$. Proto pokud bychom chtěli C např. = 0.1, což je číslo reálné na webu, musel by mít graf $\frac{n \cdot (n-1)}{20}$ hran, což pro graf o $8 \cdot 10^7$ vrcholech dává stupeň vrcholu $4 \cdot 10^6$.

3.2 Barabási-Albert model

Barabási-Albert model je jedním z modelů, které splňují exponencionální distrubuci stupňů uzlů v grafu. Do generovaného grafu jsou postupně v čase přidávány vrcholy a hrany. Hlavní myšlenkou toho modelu je spojování nových uzlů s uzly s větším stupněm – což odpovídá tomu, že autor webové stránky na ni přidá linky odkazující na "populárnější" dokumenty. Postup je následovný:

- generování začíná s m_0 uzly.
- v cyklu se přidávají uzly a hrany následovně:
- je vytvořen vrchol v a poté spojen s $m < m_0$ uzly stávajícího grafu
- uzel u bude koncovým uzlem nové hrany (tedy bude spojen s v) s pravděpodobností $p(u) = \frac{deg(u)}{\sum (degw)}$.

Tímto postupem je zaručeno, nový uzel bude spojen s uzly s větším stupněm. Podrobná analýza tohoto modelu je uvedena například v [RK99]. Z hlediska použitelnosti tohoto modelu je důležité, že není možné ovlivnit exponencionální distribuci stupňů vrcholů. Zachovává tuto distribuci, ale pouze pro $a = 3$. Je tedy nezávislá na jediném vstupním parametru, kterým je m_0 – počet počátečních vrcholů. Proto bude představen další model, ve kterém lze distribuci parametricky ovlivnit.

3.3 (α, β) model

Dalším modelem je (α, β) model (uveden v [RK99]), který splňuje vlastnost rozložení uzlů s vstupním a výstupním stupněm tak, jak je ve webovém prostoru. Vstupem jsou reálná čísla α a β, která udávají pravděpodobnosti. V každém kroku konstrukce grafu budeme přidávat uzel u a hranu e. α udává pravděpodobnost, že nová hrana bude končit v uzlu u a (1–α) je tedy pravděpodobnost, že e bude končit v náhodně zvoleném cíli libovolné hrany (tedy $e = (u, w)$, pokud je v grafu hrana (q, w)). Dále β

udává pravděpodobnost, že e bude začínat v novém uzlu u a $(1–\beta)$ je pravděpodobnost, že e začíná v libovolném začátku jiné hrany. Označme $P_{i,t}$ počet uzlů, které mají v čase t vstupní stupeň i a $Q_{i,t}$ počet uzlů, které mají v čase t výstupní stupěň i. Pro tento model platí:

$$P_{i,t} = i^{-\frac{1}{1-\alpha}}$$

$$Q_{i,t} = i^{-\frac{1}{1-\beta}}$$

Důkaz, že daný model splňuje distribuci vstupních a výstupních uzlů podle uvedených vztahů (1) je v [RK99].

Pokud tedy chceme vygenerovat webový graf, který splňuje dané rozložení, je potřeba na daném vzorku toto rozložení spočítat a aproximovat nějakou funkcí (najít potřebné koeficienty). Tyto koeficienty dát jako vstupní parametry modelu a poté dostaneme očekávaný výsledek co se týče rozložení stupně uzlů. Model, ve kterém je přidávána pouze jedna hrana k novému uzlu ale jistě nesplňuje požadavky na vznik dostatečného počtu bipartitních jader v grafu. Pokud by ale bylo v každém kroce přidáno k novému uzlu k hran, které by vznikly kopií již existujících hran, situace by byla následující. Pravděpodobnost, že dva nové uzly dostanou stejný vzor u pro kopírování hran je $\binom{n}{2}^{-1}$, tedy přibližně n^{-2}. Pokud navíc oba tyto uzly okopírují stejných l uzlů ($l < k$), vznikne bipartitní jádro $C_{3,l}$.

Kapitola 4

Implementace

V následující kapitole bude popsáno, co bylo v rámci práce implementováno. Nejdříve část věnovaná reprezentaci webového grafu, jeho uložení na disku a dále skriptovací jazyk, který slouží k definování průchodu webem pro robota. Práce byla implentována v jazyku C++, je určená pro unixové prostředí. Vývoj probíhal na linuxu s překladačem gcc. Byla použita externí knihovna Berkeley DB a programy flex a bison.

4.1 Struktura webového adresáře, práce s webovým grafem.

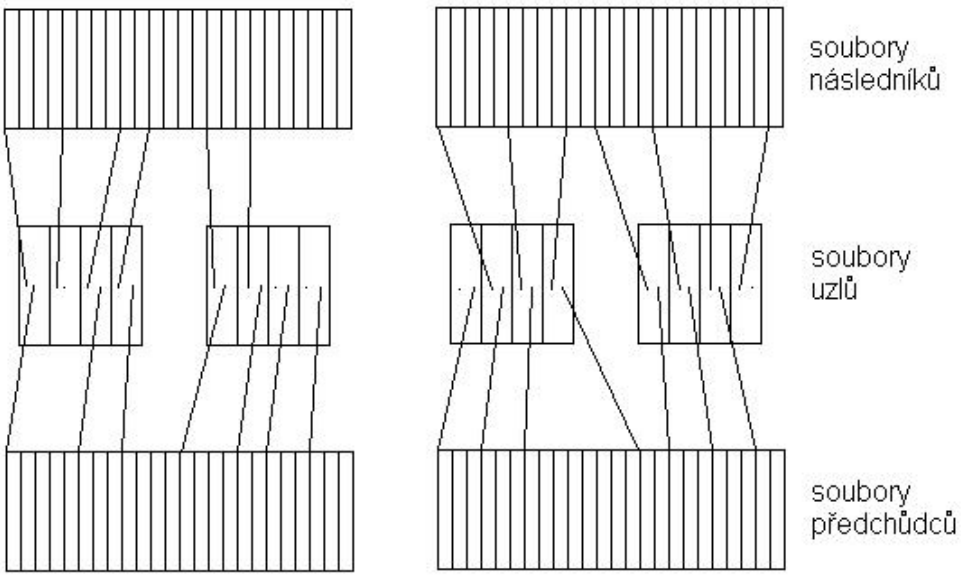
Nejdříve bude popsána struktura souborů potřebných k uložení simulovaného webu a práce s nimi. Informace vztažené k uzlu jsou umístěny ve třech souborech. V prvním souboru je uložena informace o čísle uzlu, počtu následníků, předchůdců a o pozici v souborech předchůdců a následníků, na které jsou tyto uzly uloženy. Záznamy v prvním souboru jsou setřízeny podle čísel uzlů, kvůli snadnějšímu vyhledání konkrétního vrcholu.

Obrázek 4.1: Soubor obsahující informace o uzlech

1 - 64bitové číslo udávající číslo uzlu
2 - 32-bitové číslo udávající počet následníků
3 - 32-bitové číslo udávající počet předchůdců

4 - 64-bitové číslo udávající pozici v souboru následníků
5 - 64-bitové číslo udávající pozici v souboru předchůdců

Celá struktura souborů vypadá následovně:



Obrázek 4.2: Stuktura souborů s informacemi o grafu

Pro načtení uzlu i je potřeba přístup do tří souborů (pokud má i nenulový počet následníků a předchůdců). Nejdříve je nalezen v souboru uzlů, kde je také zjištěno z jaké pozice a kolik 64-bitových čísel musí být načteno ze souboru předchůdců (analogicky pro následníky). Kvůli optimalizaci přístupu na disk nejsou uzly načítány do vnitřní paměti jednotlivě, ale po větším množství. Předpokládá se, že když je potřeba načíst uzel n, bude třeba načíst i vrcholy, které následují hned za ním. Tím však vzniká problém - kdy načtené uzly uvolňovat z paměti? Pokud byly načteny aniž by o ně bylo zažádáno, potom také nikdo nebude volat jejich uvolnění. Nejdříve budou ukázány dvě hlavní třídy - CindexedBufferedFile a Node. Instance třídy CindexedBufferedFile spravuje všechny soubory, které byly doposud popsány:

```
class CindexedBufferedFile {
    ...
    Node* getNode(long long int llNode);
    ...
};
```

Metoda getNode() vrací ukazatele na uzel s číslem llNode. Třída Node má čítač referencí – pokud je tedy zavolána metoda getNode(), je instanci Node nejdříve zvýšen tento čítač a až poté je vrácen ukazatel. Po skončení práce s ukazatelem je potřeba zavolat metodu Node::freeNode(), která sníží hodnotu čítače referencí. Všechny ukazatele na uzly načtené do vnitřní paměti jsou uchovány ve třídě CindexedBufferedFile a pokud je alokováno větší množství paměti než N bajtů, je jich několik uvolněno (pokud je čítač referencí roven jedné, tedy jediný kdo na ně má odkaz je práve instance CindexedBufferedFile). Dalším problémem, který bylo potřeba vyřešit při implementaci načítání uzlů byla fragmentace paměti. Třída pro reprezentaci uzlů uchovává následníky a předchůdce v dynamicky alokovaném poli. Proto při postupném načítání uzlů a uvolňování nepoužívaných poměrně rychle došla paměť právě kvůli fragmentaci. Nakonec byl napsán vlastní alokátor, který obsahuje bloky různých délek a pokud uzel paměť potřebuje, je mu přidělena tímto alokátorem. Naopak dealokace probíhá tak, že uzel pouze oznámí alokátoru, že daný blok je opět volný.

4.2 Skriptovací jazyk.

Aby bylo možné simulovat průchod webového robota webovým prostorem a hlavně testovat různé možnosti jak jím procházet, byl navržen a implementován skriptovací jazyk, který to umožnuje. Jazyk byl navržen tak, aby byl co nejvíce jednoduchý, ale zároveň aby v něm bylo možné pracovat se stránkami, s linky vedoucími z resp. do dané stránky a aby bylo možné tyto stránky ukládat do kontejnerů (front nebo zásobníků). Vychází z pascalovské syntaxe. Na úvod si ukážeme jednoduchý příklad zdojového kódu a poté detailněji rozebereme možnosti jazyka.

Průchod grafem do šířky:

```
attributes
    visited : boolean;
var
    webFile(/tmp/webFileDir/) file;
    webFileSize: integer;
    q: queue of node;
    qSize: integer;

    n, tmpNode: node;
    nOutlinksSize, tmpNodeOutlinksSize: integer;
    nOutlinks : queue of node;
begin
```

```
    webFileSize := webFile.size();
    while (webFileSize > 0) do
    begin
       n := webFile.top();
    webFile.pop();
    if (n.visited = 0) then
    begin
          q.push(n);
          qSize := q.size();
          while (qSize > 0) do
          begin
             n := q.top();
             n.visited := 1;
             q.pop();
             nOutlinks := n.Outlinks;
             nOutlinksSize := nOutlinks.size();
             // zápis uzlu n na výstup - uzel n byl zpracován
             output(n);
             while (nOutlinksSize > 0) do
             begin
                //do fronty vkládam všechny uzly, na které se n odkazoval a
                //které jsem ještě nenavštívil
                tmpNode := nOutlinks.top();
                nOutlinks.pop();
                if (tmpNode.visited = 0) then
                begin
                   q.push(tmpNode);
                end;
                nOutlinksSize := nOutlinksSize - 1;
             end;
             qSize := q.size();
          end;
       end;
       webFile.pop();
       webFileSize := webFileSize - 1;
    end;
end.
```

Ve skriptovacím jazyce je možné používat několik datových typů:

```
    integer
    boolean
    node
```

queue
queue(parametr)
stack
file
aoq

Atributy.

Dále u datového typu node můžeme používat tzv. atributy - které jsou podobné atributům tak, jak je známe z objektových programovacích jazyků. Máme-li proměnnou typu node (tedy webovou stránku), potom můžeme přes tečku přistupovat k jejím atributům. Tyto atributy jsou perzistentní, jejich hodnota je tedy uložena na disku a můžeme se k ní vrátit kdykoliv budeme pracovat s daným uzlem. Hlavní motivací zavedení atributů do jazyka bylo, aby bylo možné uchovávat si k uzlům informace, které jsme dosud spočítali. Například pagerank. Pokud by jej bylo v jazyce možné pouze spočítat pro aktuální stránku a po jejím zpracování by se zapomněl, neměli bychom tuto informaci, pokud bychom na stejnou stránku narazili příště (například by z jiné stránky vedl odkaz na původní). To by znamenalo, že by se příště nemohl pagerank upřesnit, ale počítal by se znova. Pro implementaci atributů byla použita externí knihovna Berkley DB. Atributy jsou uloženy v této databázi jakožto dvojice klíč hodnota. Pokud pro uzel u chceme uložit například atributy *pagerank* a *statistika*, jsou pod klíčem u uložena tato dvě čísla. Používání atributů výrazně spomaluje práci s celým grafem, pro smysluplný výpočet však jsou potřebné.

Atributy mohou být dvou typů: boolean a integer. Existují i tři implicitní atributy jsou popsány v sekci věnované datovému typu node.

Popis jednotlivých datových typů.

Kontejnery.

Sem patří tři datové typy – zásobník, fronta a prioritní fronta. Do kontejneru můžeme skladovat celá čísla (integery), nebo uzly (datový typ node). Mají společné rozhraní, jsou nad nimi definovány operace:

push(prvek) – vloží prvek do kontejneru
pop() - odstraní prvek z kontejneru
top() - návratová hodnota je první prvek v kontejneru
size() - návratová hodnota je velikost kontejneru.

stack

Jedná se o zásobník, prvek naposeld přidaný je prvkem, který se odebere jako první.

queue

Fronta, prvky se z ní odebírají ve stejném pořadí v jakém byly vloženy.

queue(parametr)

Prioritní fronta. Prvky v tomto kontejneru jsou setřízeny podle parametru. Pokud jsou ve frontě integery, přítomnost parametru pouze říká, že má být fronta setřízena. V případě uzlů (node) musí být parametr jméno atributu (definované dříve v sekci attributes), podle kterého se prvky budou třídit. Příklad: Chceme-li odebírat z fronty nejdříve prvky s největší pagerankem, deklarujeme frontu takto:

```
attributes:
    pagerank: integer;
var
    q: queue(pagerank) of node;
    ...
```

Všechny kontejnery jsou implementovány tak, aby dokázaly pojmout množství prvků, které se nemusí vejít do vnitřní paměti. Každý má nějaký vnitřní buffer, do kterého jsou vkládány prvky (a z kterého jsou také odebírány) a pokud je tento buffer naplněn, je zapsán na disk. Tím je umožněno pracovat s "libovolně" velkými kontejnery, omezení je pochopitelně velikost místa na disku a počet otevřených souborů.

node

Reprezentuje v programu webovou stránku. Má tři implicitní atributy: outlinks, což je kontejner uzlů, do nichž vede odkaz z daného uzlu, inlinks, kontjner uzlů z nichž vede odkaz na daný uzel a dále visited. Visited je typu boolean a slouží k označení, zda robot daný uzel již zpracoval. Pokud chceme zpracovat následníky uzlu node, vložíme je do libovolného typu kontejneru a poté na něm voláme operace které byly popsány dříve.

Příklad, ve kterém je použita prioritní fronta, atributem, podle kterého se prvky řadí je pagerank - říkame tím, že chceme jako první zpracovávat následníky s nejvyšším pagerankem:

```
var
    n: node;
    q(pagerank) : queue of node;
    ...
begin
    ...
    q := n.outlinks
    while (q.size() > 0) do
    begin
       n := q.top();
       q.pop();
       output(n);
    end;
    ...
end.
```

file

Reprezentuje v programu web. Má jeden povinný parametr, který říká, kde je umístěn adresář, v němž jsou uloženy datové soubory popisující web. Pracuje se s ním obdobně jako s kontejnerem, s tím rozdílem, že nelze přidávat prvky.
Příklad:

```
var
    webFile(/tmp/webFileDir/) file;
begin
    n := webFile.top();
    webFile.pop();
    //zpracování uzlu
```

Operace pop() nemá význam smázání uzlu z disku, pouze se ukazatel na aktuální uzel v souboru přesune na následující.

Aoq (array of queue)

Pro vysvětlení zavedení tohoto datového typu do jazyka je potřeba vysvětlit možný průchod robota webovým prostorem. Robot při stahování "neskáče" ze stránky na stránku, ale při zpracovávání webové stránky ukládá jednotlivé odkazy do front odpovídajícím webovým serverům. Když už má v dané frontě určité množství odkazů, připojí se na daný server a stáhne stránky, které má v odpovídající frontě. Pro lepší představu je uveden příklad:

Mějme webové servery A, B a C se stránkami a1, a2, a3 ,a4 (stránky na serveru A), b1, b2, b3, b4 (stránky serveru B) a analogicky c1,..,c4 na serveru C. Dále nechť je definovaný následující webový graf: vrcholy jsou stránky, hrany jsou { (a1,a3) (a1,c4), (a1,c3), (a3,b1), (a3,b2),(a3,b4),(a3,c2),(a3,c3)}. Jako první je stažena stránka a1, procházíme její následníky a2, a3, c4 a c3 a ukládáme je do pole front takto:

queue1 = {a3}
queue2 = {}
queue3 = {c4,c3}

Poté zpracujeme uzel a3 a pole se změní:

queue1 = {a3}
queue2 = {b1,b2,b4}
queue3 = {c4,c3,c2,c3}

Fronty queue1 až queueN jsou ve skriptovacím jazyce skryty právě za datový typ aoq. Jsou nad ním definovány tyto operace:

push(uzel) – vloží uzel do pole (zařadí jej do příslušné fronty)
pop(i) - odstraní i-tou frontu z pole (vyprázdní ji)
top(i) - návratová hodnota je první i-tá fronta v poli
size() - návratová hodnota je velikost pole (počet front s nenulovou velikostí).

Pro ukázku práce s aoq bude ukázán výběr fronty s největším počtem prvků:

```
var
    i : integer;
    arrayOfqueue: aoq;
    q: queue of Node;
begin
    //...do arrayOfqueue vložíme určitý počet uzlů
    ....
    i := 0;
    maxIndex := 0;
    maxSize := 0;
    while ( i < arrayOfqueue.size() ) do
    begin
       q := arrayOfqueue.top(i);
       if (maxSize < q.size()) then
```

```
    begin
      maxSize := q.size();
      maxIndex := i;
    end;
    i := i + 1;
  end;
  //arrayOfqueue.top(maxIndex) vrátí frontu s největším počtem uzlů.
```

4.3 Simulátor DNS a WWW serveru.

Třetí části práce je tvorba zcela transparetního rozhraní pro testování robota nad daným webovým grafem. Ve zvoleném řešení robota tedy není potřeba nijak upravovat, aby mohl stahovat stránky ze simulovaného webu. Jak již bylo uvedeno v úvodu, robot dostane na vstupu seznam URL, stáhne dané stránky a poté se rozhoduje, které odkazy ze stažených stránek navštíví – takto funguje i simulace, s tím, že URL, s kterými pracuje jsou uměle vytvořená. To znamená, že parametrem je například www.aa.bb.cc.dd.cz/a.html (doménová jména, se kterými robot pracuje jsou popsána v příloze). Zde je dobré si uvědomit, že neexistující URL nejsou pro robota žádným omezením, důležité je, aby existoval jejich překlad na IP adresy o což se stará simulovaný DNS server. Tento překlad bude popsán v následující kapitole.

Překlad URL na čísla uzlů.

Z hlediska simulovaného DNS serveru bylo potřeba vyřešit následující problém: vytvoření množiny doménových jmen a jejich mapování na IP adresy. Umělá doménová jména jsou typu www.AA.BB.CC.DD.cz, kde AA.BB.CC.DD je hexadecimálním zápisem IP adresy, takže dané URL odpovídá adrese 170.187.204.221. Toto řešení bylo zvoleno z toho důvodu, aby ve webovém grafu nebylo nutné udržovat adresu stroje, na kterém je umístěna webová stránka – vytvoří se pouze zobrazení čísel uzlů na IP adresy a poté už se z IP adresy jednoduše zjistí doménové jméno (toto zobrazení je potřeba, protože ve webovém grafu je pro každou stránku uveden seznam následníků (opět čísel uzlů), takže pro dané číslo potřebujeme zjistit URL). Dále je potřeba také obrácené zobrazení – ze zadaného URL potřebujeme zjistit číslo vrcholu ve webovém grafu.

Předpokládejme, že maximimální číslo uzlu ve webovém grafu je MAX_NODE_NUMBER (může být například rovno 2^{42}). Dále počet IP adres označíme $IP\_ADDRESS\_COUNT$ (například rovno 2^{32}). Potom definujeme zobrazení *ipaddres* takto:

$ipaddress :< 0, MAX\_NODE\_NUMBER >$
$\rightarrow < 0, IP\_ADDRESS\_COUNT >$
$ipaddress(u) = u \cdot \frac{IP\_ADDRESS\_COUNT}{MAX\_NODE\_NUMBER}$

Pro každý uzel v našem webovém grafu tedy nyní známe jeho IP adresu. Dále je potřeba nějak identifikovat stránku v rámci daného webového serveru. Proto definujme další zobrazení *doc* následovně:

$doc :< 0, MAX\_NODE\_NUMBER > \rightarrow < 0, \frac{MAX\_NODE\_NUMBER}{IP\_ADDRESS\_COUNT} >$
$doc(u) = u \, mod \, \frac{MAX\_NODE\_NUMBER}{IP\_ADDRESS\_COUNT}$

Nyní už mužeme zavést zobrazení, které danému uzlu přiřadí URL:

$url(u) :< 0, MAX\_NODE\_NUMBER >$
$\rightarrow << 0, IP\_ADDRESS\_COUNT >, < 0, \frac{MAX\_NODE\_NUMBER}{IP\_ADDRESS\_COUNT} >>$
$url(u) =< ipaddres(u), doc(u) >$

Příklad:
uzel $u = 256987$
$url(u) =< \frac{256987}{2^{10}}, 256987 mod 2^{10} >=< 250, 987 >.$
Pokud budeme stránky pojmenovávat například aaaa.html až zzzz.html, potom nechť stránka 987 bude má označení abcd.html. URL uzlu 256987 bude www.00.00.00.FA.cz/ abcd.html (00.00.00.FA je pouze zápis čísla 250 v hexadecimální soustavě).
Když tedy robot chce stáhnout stránku s, stačí přeložit pomocí zobrazní url všechny odkazy vedoucí z s na URL a tyto odeslat robotovi ve stránce s, případně vrátit opověd, že stránka nebyla nalezena.
Obrácené zobrazení, kdy je ze zadaného URL potřeba zjistit číslo uzlu lze jednoduše zrekonstruovat ze zobrazení *url¿*

$node :<< 0, IP\_ADDRESS\_COUNT >, < 0, \frac{MAX\_NODE\_NUMBER}{IP\_ADDRESS\_COUNT} >>$
$\rightarrow < 0, MAX\_NODE\_NUMBER >$
$node(ip, docid) = ip * \frac{MAX\_NODE\_NUMBER}{IP\_ADDRESS\_COUNT} + docid.$

Pokud uvažujeme URL z přechozího příkladu (www.00.00.00.FA.cz/abcd.html), potom ip = FA16, docid = 987, takže node (FA,987) = 256987.

Simulace DNS.

Robot vysílá protokolem UDP požadavky na překlad doménového jména na ip adresu (DNS dotazy). Simulátor webu musí umět na tyto DNS dotazy odpovídat. Proto byl implementován jednoduchý DNS server, který

zná předem danou množinu doménových jmen (www.AA.BB.CC.DD.cz) a v odpovědi pošle IP adresu podle převodu v miulé kaptiole. Různě náročných možností jak implementovat pro robota transparentní překlad je několik. Liší se podle toho, jestli simulátor sám odchytává všechny UDP pakety a pouze na ty jemu určené odpoví (a zároveň tyto pakety zahodí, nepošle je skutečnému DNS serveru, kterému byly adresovány). Protože práce je určená pro UNIXové systémy, ve kterých existuje nějaký paketový filtr (například v linuxu iptables), je tento filtr použit pro přesměrování paketů na localhost. Robot vyšle požadavek na překlad doménového jména www.AA.BB.CC.DD.cz reálnému serveru DNS (podle /etc/hosts), ten jej pomocí pravidla definovaného programu iptables přesměrován na localhost a dále je původní požadavek pomocí jiného pravidla zahozen. Simulovaný DNS server požadavek přijme, pokud existuje převod doménového jména na IP adresu tak jej provede a pošle robotovi tuto adresu v odpovědi. Pokud tento převod neexistuje, vrátí negativní odpověd. Implementace simulace DNS tedy obnáší naprogramování zjednodušené verze DNS serveru.

Simulace webového serveru.

Pokud robot zná IP adresu, z které chce stáhnout nějakou stránku, připojí se k serveru pomocí http protokolu jej stáhne. Používá dvě metody http protokolu – GET a HEAD. GET slouží ke stažení požadovaného dokumentu, pomocí HEAD robot pouze zjistí metadata k danému dokumentu (čas, datum modifikace, atd..). Proto druhá část simulátoru implementuje webový server. Požadavky robota jsou vysílány na IP adresu, kterou zjistil pomocí DNS dotazu (na simulovaný web server), proto je potřeba opět tyto požadavky odchytit pomocí paketového filtru a přesměrovat je na localhost. Protože je ale přepsáním cílové IP adresy ztracena informace o serveru, na který se daný požadavek posílá, je nutné aby robot v metodě GET poslal i doménové jméno serveru. Na základě znalosti doménového jména a názvu stránky simulátor najde daný uzel a může vrátit odpověd. Tou je webová stránka s libovolným textem a odkazy na další stránky, podle toho, kolik jich vychází z dané stránky v simulovaném webovém grafu.

4.4 Gramatika skriptovacího jazyka.

V následující části je popsána gramatika jazyka. Pro jeho parsování byly použity programy flex a bison.

$SOURCE \rightarrow ATTRIBUTES\_BLOCK \;\; PROGRAM$

$ATTRIBUTES\_BLOCK \rightarrow \lambda$

$| \ attributes \ ATTRIBUTES;$

$ATTRIBUTES \rightarrow ATTRIBUTES \ ; \ ATTRIBUTE$
$| \ ATTRIBUTE$

$ATTRIBUTE \rightarrow IDENTIFIERS \ : \ TYPE$

$IDENTIFIERS \rightarrow IDENTIFIERS \ : \ DECLARATION\_IDENTIFIER$
$| \ DECLARATION\_IDENTIFIER$

$DECLARATION\_IDENTIFIER \rightarrow IDENTIFIER \ SORT\_IDENTIFIER$
$| \ IDENTIFIER$

$IDENTIFIER \rightarrow [a-zA-Z]+$

$SORT\_IDENTIFIER \rightarrow (\ IDENTIFIER \ )$

$TYPE \rightarrow CONTAINER \ of \ TERM\_TYPE$
$| \ TERM\_TYPE$

$CONTAINER \rightarrow queue \mid stack$

$TERM\_TYPE \rightarrow node \mid integer \mid boolean$

$PROGRAM \rightarrow BLOCK \ .$

$BLOCK \rightarrow DECLARATION\_BLOCK \ STATEMENTS_BLOCK$

$DECLARATION\_BLOCK \rightarrow \lambda$
$| \ var \ DECLARATIONS \ ;$

$DECLARATIONS \rightarrow DECLARATIONS \ ; \ DECLARATION$
$| \ DECLARATION$

$DECLARATION \rightarrow IDENTIFIERS \ : \ TYPE$

$STATEMENTS\_BLOCK \rightarrow begin \ STATEMENTS \ end$

$STATEMENTS \rightarrow STATEMENTS \ ; \ STATEMENT$
$| \ STATEMENT$

$STATEMENT \rightarrow OPEN\_STATEMENT$
$| \ CLOSED\_STATEMENT$

$OPEN\_STATEMENT \rightarrow OPEN\_IF\_STATEMENT$
$\quad | \; OPEN\_WHILE\_STATEMENT$

$CLOSED\_STATEMENT \rightarrow ASSIGNMENT\_STATEMENT$
$\quad | \; CLOSED\_IF\_STATEMENT$
$\quad | \; CLOSED\_WHILE\_STATEMENT$
$\quad | \; STATEMENTS\_BLOCK$
$\quad | \; PROCEDURE\_STATEMENT$
$\quad | \; \lambda$

$OPEN\_IF\_STATEMENT \rightarrow if \;\; BOOLEAN\_EXPRESSION \;\; then$
$\quad STATEMENT$
$\quad | \; if \;\; BOOLEAN\_EXPRESSION \;\; then$
$\quad CLOSED\_STATEMENT \;\; else \;\; OPEN\_STATEMENT$

$OPEN\_WHILE\_STATEMENT \rightarrow while \;\; BOOLEAN\_EXPRESSION$
$\quad do \;\; OPEN\_STATEMENT$

$ASSIGNMENT\_STATEMENT \rightarrow VARIABLE\_ACCESS \;\; :=$
$\quad EXPRESSION$

$CLOSED\_IF\_STATEMENT \rightarrow if \;\; BOOLEAN\_EXPRESSION \;\; then$
$\quad CLOSED\_STATEMENT \;\; else \;\; CLOSED\_STATEMENT$

$CLOSED\_WHILE\_STATEMENT \rightarrow while$
$\quad BOOLEAN\_EXPRESSION \;\; do \;\; CLOSED\_STATEMENT$

$STATEMENTS\_BLOCK \rightarrow begin STATEMENTS end$

$PROCEDURE\_STATEMENT \rightarrow VARIABLE\_ACCESS \;\; .$
$\quad PROCEDURE\_NAME \;\; PARAMS$
$\quad | \; VARIABLE\_ACCESS \;\; . \;\; PROCEDURE\_NAME \;\; (\;\;)$

$VARIABLE\_ACCESS \rightarrow IDENTIFIER$
$\quad | \; FIELD\_DESIGNATOR$

$PROCEDURE\_NAME \rightarrow pop \mid top \mid push \mid size$

$PARAMS \rightarrow (\;\; ACTUAL\_PARAMETER\_LIST \;\;)$

$FIELD\_DESIGNATOR \rightarrow VARIABLE\_ACCESS \;\; . \;\; IDENTIFIER$

$ACTUAL\_PARAMETER\_LIST \rightarrow ACTUAL\_PARAMETER\_LIST\ ,$
$\quad ACTUAL\_PARAMETER$
$\quad \mid ACTUAL\_PARAMETER$

$ACTUAL\_PARAMETER \rightarrow EXPRESSION$

$BOOLEAN\_EXPRESSION \rightarrow EXPRESSION$

$EXPRESSION \rightarrow SIMPLE\_EXPRESSION$
$\quad \mid SIMPLE\_EXPRESSION\ RELOP\ SIMPLE\_EXPRESSION$

$SIMPLE\_EXPRESSION \rightarrow TERM$
$\quad \mid SIMPLE\_EXPRESSION\ ADDOP\ TERM$

$TERM \rightarrow FACTOR$
$\quad \mid TERMMULOPFACTOR$

$FACTOR \rightarrow SIGN\ FACTOR$
$\quad \mid PRIMARY$

$PRIMARY \rightarrow VARIABLE\_ACCESS$
$\quad \mid UNSIGNED\_CONSTANT$
$\quad \mid (\ EXPRESSION\ )$
$\quad \mid not\ PRIMARY$
$\quad \mid FUNCTION\_DESIGNATOR$

$FUNCTION\_DESIGNATOR \rightarrow PROCEDURE\_STATEMENT$

$UNSIGNED\_CONSTANT \rightarrow [0-9]+$

$RELOP \rightarrow <\ \mid\ <=\ \mid\ <>\ \mid\ >=\ \mid\ >\ \mid\ =$

$ADDOP \rightarrow\ +\mid\ -\ \mid OR$

$MULOP \rightarrow * \mid / \mid AND$

$SIGN \rightarrow\ +\mid -$

Kapitola 5

Závěr

Úkolem práce bylo vytvořit prostředí pro testování webového robota. Generování webového prostoru byla věnována řada předchozích prací, proto byl v rámci této práce implementován jeden z možných modelů. Dalším možným rozšířením je implementování dalších modelů - s malou náročností na úpravu kódu.

Možnost popsat stahovací strategii ve skriptovacím jazyce a takto ji otestovat nebyla představena v žádné jiné autorem nalezené práci. Současná implementace splňuje zadání práce - vstupem je popis průchodu robota webem, výstupem je posloupnost navštívených uzlů. Navržený jazyk je možné dále rozvíjet, například přidáním funkcí pro práci s jednotlivými stránkami - pro testování jestli daná stránka obsahuje určité slovo, stáří stránky atd. Dalším možným rozšířením může být například podpora funkcí nebo procedur. Transparetní testovací prostředí nebylo naimplementováno zcela, proto jediným plně funkčním rozhraním je skriptovací jazyk.

Literatura

[AAR01] Hector Garcia-Molina Andreas Paepcke Arvind Arasu, Junghoo Cho and Sriram Raghavan, *Searching the web*, ACM Transactions on Internet Technology **1** (2001), no. 1, 2–43.

[DC06] Christos Faloutsos Deepayan Chakrabarti, *Graph mining: Laws, generators, and algorithms*, ACM Computing Surveys **38** (2006), no. 2, 84–89.

[DD04] Stefano Leonardi-Stefano Millozzi Debora Donato, Luigi Laura, *Simulating the webgraph: A comparative analysis of models*, Computing in science & engineering **6** (2004), 84–89.

[DD05] Stefano Leonardi-Stefano Millozzi Debora Donato, Panayiotis Tsaparas, *Mining the inner structure of the web graph*, Eighth International Workshop on the Web and Databases (Baltimore, Maryland), 2005.

[DD06] Stefano Leonardi-Stefano Millozzi Debora Donato, Luigi Laura, *Algorithms and experiments for the webgraph*, Journal of Graph Algorithms and Applications **10** (2006), no. 2, 219–236.

[JT01] Ph. Nain-E. G. Coffman J. Talim, Z. Liu, *Controlling the robots of web search engine*, 2001.

[PE59] Alfréd Rényi Paul Erdős, *On random graphs*, Publ. Math. (1959), 290–297.

[RA02] Albert-László Barabási Réka Albert, *Statistical mechanics of complex networks*, Reviews of modern physics **74** (2002), no. 1.

[RK99] Sridhar Rajagopalan-Andrew Tomkins Ravi Kumar, Prabhakar Raghavan, *Extracting large-scale knowledge bases from the web*, Proceedings of the 25th VLDB Conference (Edinburgh, Scotland), 1999.

[WA00] Linyuan Lu William Aiello, Fan Chung, *A random graph model for massive graphs*, 2000.

Dodatek A

Obsah DVD-ROM

Obsah DVD-ROM

DVD v příloze je součástí práce. Obsahuje její text, zdrojové kódy a testovací data. Pro kompilaci je potřeba mít na počítači nainstalován flex, bison a Berkeley DB.

- text - text práce ve formátu pdf
- src\WebGraphReader - zdrojový kód generátoru webového grafu
- src\WebInterpret - zdrojový kód skriptovácího jazyka
- data - testovací data, webový graf ve formátu popsaném v kapitole implementace