Lucrare de disertație

Motor de căutare Web

https://www.text-mining.ro/

Student: ing. Ștefan-Mihai MOGA
Facultatea de Cibernetică, Statistică și Informatică Economică
Academia de Studii Economice din București

Coordonator: prof. univ. dr. Titus Felix FURTUNĂ

Cuprins

1. Introducere

Convergența procesării informației cu tehnicile de comunicație, dovedită de dezvoltarea exponențială a Internet-ului, a determinat apariția unui volum mare de date, în cele mai diverse forme. Această cantitate mare de informații se datorează nu doar dezvoltării Web-ului, cât și apariției unor tehnologii emergente precum sistemele dedicate, sistemele mobile și sistemele omniprezente. Devine clară necesitatea extragerii de informații și cunoștințe din aceste masive de date puternic distribuite. În acest scop au fost dezvoltate metodele de **Data Mining**, reprezentând tehnicile de extracție a informațiilor din date, necunoscute anterior și de un folos potențial pentru utilizatori.

Altfel spus, "mineritul datelor" constă în descoperirea *pattern*-urilor structurale din date. Algoritmii dezvoltați în acest sens trebuie să fie suficient de robuști pentru a putea prelucra date imperfecte, afectate de zgomot, dar și pentru a putea extrage reguli și corelații folositoare. Aceste reguli pot fi ulterior utilizate în predicția, explicarea și înțelegerea evoluției structurii datelor analizate. De o mare actualitate sunt tehnicile de **Text Mining** și Web Mining, dezvoltarea acestora fiind accelerată de dublarea cantității de date existente pe Internet la fiecare 6 luni, precum și de caracterul anarhic și entropic al acestor date.

Consider că tehnicile de **Web Mining** sunt încă într-o primă fază a dezvoltării, complexitatea lor fiind una uriașă ținând cont de faptul că vizează simultan procesarea structurii și conținutului paginilor Web, precum și a *pattern*-urilor de acces la paginile Web. Eficiența regăsirii informațiilor relevante pe Web reprezintă deci o provocare majoră în informatică. Clasificarea și clusteringul paginilor Web, abordate în această lucrare științifică, reprezintă sper o soluție la această problemă. Îmi propun să prezint gradual implementarea unui **motor de căutare Web** cu tehnici de Text Mining.

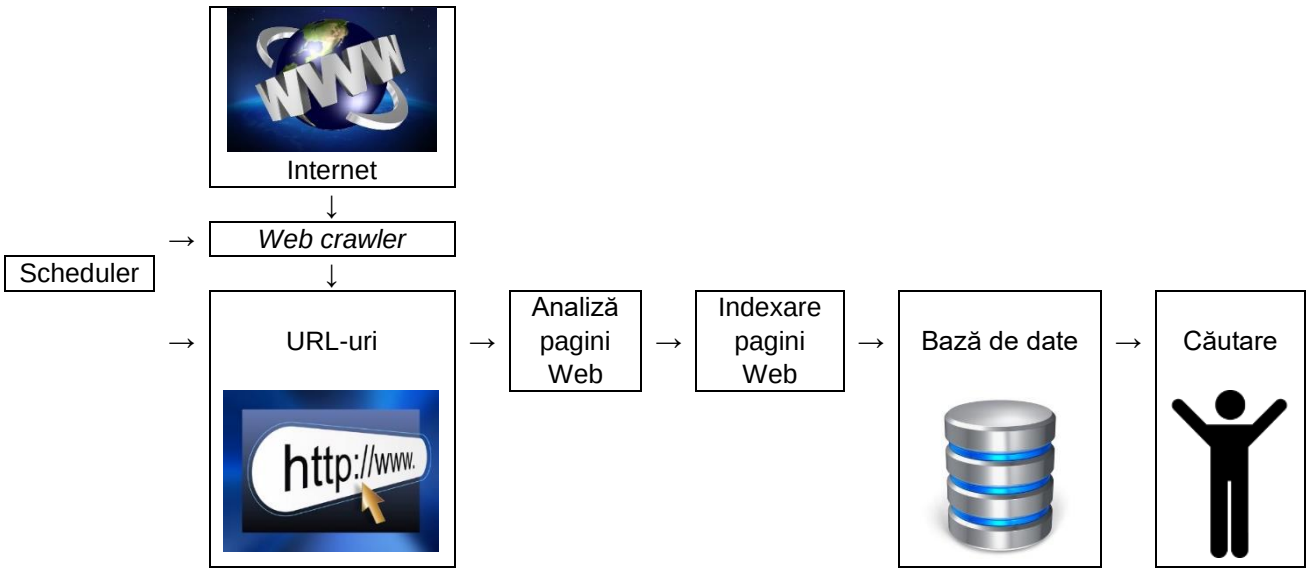
2. Stadiul cunoașterii

Un motor de căutare Web este un sistem software care este proiectat pentru a căuta informații pe World Wide Web. Rezultatele căutării sunt prezentate sub forma unor adrese de pagini generate de motoarele de căutare SERPs. Informațiile pot fi un amestec de pagini Web, imagini și alte tipuri de fișiere. Unele motoare de căutare extrag informații din bazele de date sau directoarele deschise. Spre deosebire de directoarele Web, care sunt întreținute numai de către editori umani, motoarele de căutare mențin informații în timp real rulând un algoritm pe un *Web crawler*.

Un motor de căutare Web este format din următoarele procese (care rulează în timp real):

- *Web crawling*;
- Indexare;
- Căutare.

Motoarele de căutare Web își obțin informațiile prin parcurgerea fiecărui website, pagină cu pagină. Mai întâi, este verificat fișierul `robots.txt`, înainte ca paginile Web să fie indexate în funcție de mai mulți factori precum titlul, conținutul paginii, JavaScript, Cascading Style Sheets (CSS) sau metadatele sale în metatag-uri HTML.



Indexarea înseamnă asocierea cuvintelor cheie definite la paginile Web ale fiecărui website. Asocierile sunt realizate într-o bază de date publică, pusă la dispoziție pentru căutare pe Web. O interogare din partea unui utilizator poate fi chiar un singur cuvânt. Indexul ajută la găsirea informațiilor referitoare la interogare, cât mai repede posibil. Unele dintre tehnicile de indexare și punerea în memoria cache sunt secrete comerciale, în timp ce *Web crawling* este un proces simplu de vizitare a tuturor websiturilor pe o bază sistematică. O versiune a fiecărei pagini indexate este păstrată în memoria cache, astfel încât aceasta să fie furnizată la cerere dacă pagina originală se încarcă greu sau s-a pierdut, motorul de căutare acționând ca un proxy Web.

În mod obișnuit, un utilizator introduce o interogare de câteva cuvinte cheie într-un motor de căutare Web. Baza de date are deja numele websiturilor care conțin cuvintele cheie și, folosind indexul, se generează rezultatele căutării. Sarcina cea mai grea este clasificarea rezultatelor în funcție de informațiile din indexuri. Apoi, este necesară citarea pasajelor potrivite din fiecare pagină Web, care face parte din rezultatul căutării, astfel încât să prezinte un citat cu cuvintele cheie relevante. Acestea sunt doar o parte din procesarea pe care o necesită fiecare pagină cu rezultate ale căutării Web, fiind necesare mai multe postprocesări.

Dincolo de căutările cuvintelor cheie simple, motoarele de căutare oferă diferite interfețe, cu mulți parametri de căutare, pentru a îmbunătăți rezultatele; o parte din aceste motoare de căutare Web țin cont de alegerile utilizatorilor pentru a îmbunătăți rezultatele curente față de rezultatele oferite anterior. De exemplu, din 2007, motorul de căutare Google ne-a permis să filtrăm după dată, făcând clic pe „Arată instrumente de căutare" din coloana din stânga a paginii cu rezultatele căutării inițiale și apoi selectând intervalul de date dorit. De asemenea, cele mai multe motoare de căutare susțin utilizarea operatorilor booleeni AND, OR și NOT pentru a ajuta utilizatorii finali să îmbunătățească rezultatele căutării. Operatorii booleeni permit utilizatorului să îmbunătățească și să extindă cuvintele interogării, motorul căutând cuvintele sau expresiile exact așa cum s-au introdus. Unele motoare de căutare oferă chiar și o funcție avansată numită căutare de proximitate, care permite utilizatorilor să definească distanța dintre cuvintele cheie. Există, de asemenea, căutarea bazată pe concept, unde cercetarea implică utilizarea analizei statistice pe paginile care conțin cuvintele sau expresiile căutate. De asemenea, interogările de limbaj natural permit utilizatorului să introducă o întrebare în aceeași formă în care ar fi pusă unui om. Un astfel de site este Ask.com.

Utilitatea unui motor de căutare depinde de relevanța rezultatului pe care îl returnează. Deși pot exista milioane de pagini Web care includ un anumit cuvânt sau o frază, unele pagini pot fi mai relevante, populare sau demne de încredere decât altele. Cele mai multe motoare de căutare folosesc metode de clasificare a rezultatelor pentru a le oferi de prima dată pe „cele mai bune". Modul în care un motor de căutare decide care pagini sunt cele mai bune și în ce ordine trebuie prezentate rezultatele, variază foarte mult de la un motor la altul. Metodele se modifică în timp, pe măsură ce folosirea Internetului se modifică și noile tehnici evoluează. Există două tipuri principale de motor de căutare care au evoluat: unul este un sistem de cuvinte cheie predefinite și ordonate ierarhic, pe care oamenii l-au programat în mod considerabil. Celălalt este un sistem care generează un „index inversat" prin analiza textelor pe care le localizează. Această primă formă se bazează mult mai mult pe calculator în sine pentru a face cea mai mare parte a muncii.

Cele mai multe motoare de căutare pe Web sunt afaceri susținute de veniturile din publicitate și, prin urmare, unele dintre acestea permit agenților de publicitate ca listările acestora să fie clasate la un nivel superior în rezultatele de căutare contra cost. Motoarele de căutare care nu acceptă bani pentru rezultatele lor de căutare sunt finanțate prin difuzarea de anunțuri. Motoarele de căutare fac bani de fiecare dată când cineva face clic pe unul dintre aceste anunțuri.

Figură 1. Motoare de căutare Web; sursa: http://www.practicWeb.com/2015/10/Internet-search-engines.html

2.1. Istoric

Din punct de vedere istoric, motoarele de căutare pe Internet au apărut după decembrie 1990. Serviciul WHOIS a fost implementat pentru prima dată în 1982, iar *Knowbot Information Service* în 1989. Însă primul motor real de căutare în fișiere FTP a fost Archie, care a fost lansat în data de 10 septembrie 1990.

Înainte de septembrie 1993, World Wide Web era indexat în întregime manual. A existat o listă de Web servere întreținută de Tim Berners-Lee, care era găzduită pe un Web server CERN; însă de la un moment dat, când au fost activate noi servere, lista nu a mai fost întreținută. Pe site-ul NCSA, serverele noi au fost anunțate sub titlul "Ce este nou!"

Primul instrument folosit pentru căutarea de conținut pe Internet a fost Archie. Numele vine de la *archive* (i.e. în traducere "arhivă") fără "v". A fost creat de Alan Emtage, Bill Heelan și J. Peter Deutsch, studenți la informatică la Universitatea McGill din Montreal. Programul software descărca lista de directoare cu numele tuturor fișierelor localizate pe websituri FTP anonime publice; motorul de căutare Archie nu indexa conținutul acestor websituri deoarece cantitatea de date era limitată, încât ar fi putut fi prelucrată manual.

Dezvoltarea protocolului Gopher, creat în 1991 de Mark McCahill de la Universitatea din Minnesota, a condus la două noi programe de căutare, Veronica și Jughead. Veronica (i.e. *Very Easy Rodent-Oriented Net-wide Index to Computerized Archives*) permitea căutarea după cuvinte cheie în titlurile de listări Gopher. Jughead (i.e. *Jonzy's Universal Gopher Hierarchy Excavation And Display*) a fost un instrument pentru obținerea de informații din meniul de la anumite servere Gopher.

În vara anului 1993 nu exista propriu-zis nici un motor de căutare pentru Web, deși numeroase cataloage de specialitate au fost menținute manual. Oscar Nierstrasz, de la Universitatea din Geneva, a scris o serie de scripturi Perl, care oglindeau periodic aceste pagini, pe care le-a rescris într-un format standard. Acest lucru a constituit baza pentru W3Catalog, primul motor primitiv de căutare Web, lansat pe 2 septembrie 1993.

În iunie 1993, Matthew Gray de la MIT a produs practic primul robot Web, bazat pe Perl, numit World Wide Web Wanderer, folosit pentru a genera un index numit "Wandex". Scopul lui Wanderer a fost de a măsura dimensiunea World Wide Web-ului, pe care a făcut-o până la sfârșitul anului 1995. Al doilea motor de căutare de pe Web, AliWeb, a apărut în noiembrie 1993. AliWeb nu a folosit un robot Web pentru indexarea paginilor, însă depindea de notificarea manuală făcută de administratorii fiecărui website într-un fișier de index cu un format fix.

Browser-ul Mozaic de la NCSA nu a fost primul browser-ul Web, dar s-a diferențiat major de precursorii săi; versiunea 1.0 din noiembrie 1993 folosea iconițe, semne de carte și o interfață care au făcut aplicația ușor de utilizat și atrăgătoare pentru necunoscători (*non-geeks*).

JumpStation, creat în decembrie 1993 de Jonathon Fletcher, folosea un robot Web pentru a găsi și indexa paginile Web, iar interfața aplicației de interogare era formată dintr-un formular Web. Astfel, acesta a fost primul instrument WWW de descoperire care combina cele trei caracteristici esențiale ale unui motor de căutare Web (*Web crawling*, indexare și căutare). Însă, din cauza resurselor limitate de pe platforma care rula, indexarea s-a limitat doar la titlurile paginilor găsite de *Web crawler*.

Primul motor real de căutare Web a fost WebCrawler, care a apărut în 1994. Spre deosebire de predecesorii săi, acesta a permis utilizatorilor să caute orice cuvânt, în orice pagină Web, devenind standard pentru toate motoarele de căutare majore. Acesta a fost, de asemenea, primul cunoscut de către public. Tot în 1994, a fost lansat Lycos, care a început la Carnegie Mellon University, devenind foarte cunoscut.

La scurt timp după aceea, multe motoare de căutare au apărut și s-au luptat pentru popularitate. Acestea au fost Magellan, Excite, Infoseek, Inktomi, Northern Light și AltaVista. Yahoo! a fost printre cele mai populare moduri de a găsi pagini Web de interes, însă funcția sa de căutare opera pe un director Web, și nu pe copii full-text ale paginilor Web.

În 1998, Google a început să vândă cuvintele căutate pe Web, folosindu-se de websitul goto.com, devenind una dintre cele mai profitabile afaceri pe Internet. Motoarele de căutare Web au devenit cele mai profitabile investiții la sfârșitul anilor 1990. Mulți au fost prinși în bula dot-com, un boom de piață bazată pe speculații, care a atins punctul culminant în 1999 și s-a încheiat în 2001.

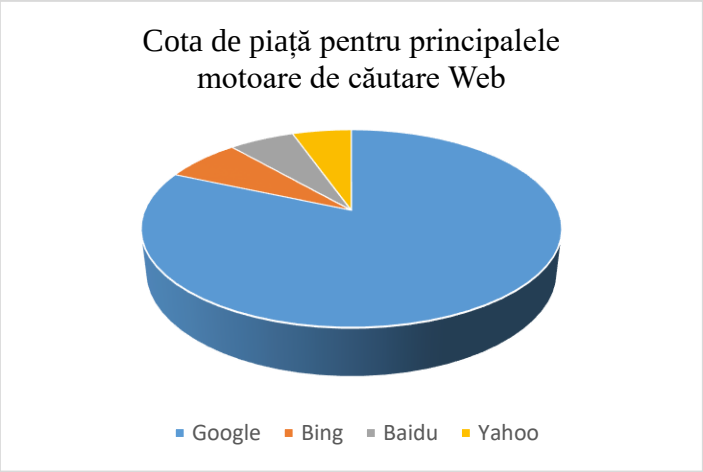
În jurul anului 2000, motorul de căutare Google a crescut în proeminență. Compania a obținut cele mai bune rezultate pentru căutări Web, cu o inovație numită PageRank, așa cum a fost explicat în "Anatomia unui motor de căutare" scrisă de Sergey Brin și Larry Page, fondatorii Google de mai târziu. Acest algoritm iterativ clasează paginile Web bazate pe numărul PageRank, plecând de la premisa că paginile bune sau dorite sunt legate mai mult decât celelalte.

Până în 2000, Yahoo! a furnizat servicii de căutare bazate pe motorul de căutare Inktomi. Yahoo! a achiziționat Inktomi în 2002, și Overture (care deținea AlltheWeb și AltaVista), în 2003. Yahoo! a apelat la motorul de căutare Google până în 2004, când a lansat propriul motor de căutare bazat pe tehnologiile combinate din achizițiilor sale.

Microsoft a lansat MSN Search în toamna anului 1998, folosind rezultatele de căutare de la Inktomi. La începutul lui 1999, websitul a început să afișeze listări de la Looksmart, amestecate cu rezultate de la Inktomi. Pentru o perioadă scurtă de timp, în 1999, MSN Search a folosit numai rezultatele de la AltaVista. În 2004, Microsoft a început o tranziție spre propria tehnologie de căutare, folosind propriul crawler Web (numit msnbot). Motorul de căutare Bing, rebrănduit de Microsoft, a fost lansat la data de 1 iunie 2009. La data de 29 iulie 2009, Yahoo! și Microsoft au finalizat o afacere în care Yahoo! Search este alimentat de tehnologia Bing.

2.2. Cota de piață

Conform cu Wikipedia, în martie 2017, Google avea cea mai mare cotă de piață – 80.52%, Bing avea 6.92%, Baidu avea 5.94%, iar Yahoo avea 5.35%. În unele țări din estul Asiei și în Rusia, Google nu este cel mai popular motor de căutare, întrucât filtrează anumite rezultate. În Rusia, cel mai popular motor de căutare Web este Yandex, care deține 61.9% cotă de piață, comparativ cu Google care are doar 28.3%. În China, cel mai populat motor de căutare Web este Baidu, în Korea de Sud este Naver 70%, iar în Japonia și Taiwan este Yahoo.



Figură 2. Cota de piață pentru principalele motoare de căutare Web

3. Analiza domeniului informatic

*Web crawler*ii, cunoscuți și ca "agenți" sau "roboți", sunt programe care descarcă automat paginile Web. De când informația pe Web este împrăștiată în miliarde de pagini furnizate de milioane de servere pe tot globul, utilizatorii care navighează pe Web pot urma ***hyperlink***-uri (i.e. legături Web) pentru a accesa informația, sărind virtual de la o pagină la alta. Un Web crawler poate vizita mai multe websituri pentru a colecta informația, care poate fi analizată într-o locație centrală, fie online (dacă este descărcată) fie offline (după ce este stocată).

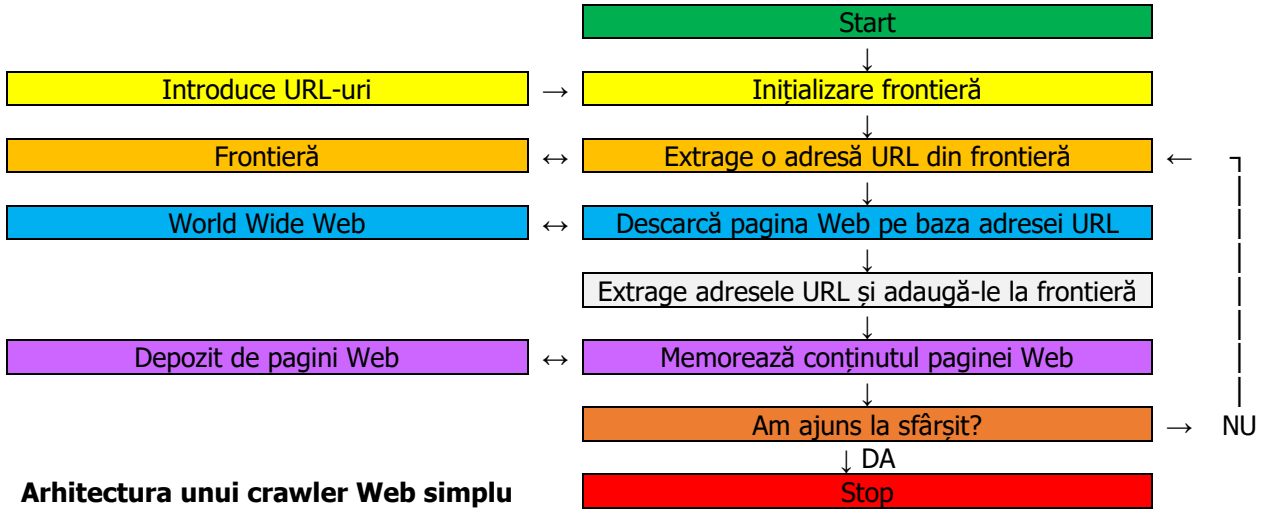
Dacă Web-ul ar fi o colecție statică de pagini, nu am avea suficient timp pentru accesarea cu crawlere. Odată ce toate paginile sunt aduse și salvate într-un depozit, am terminat. Cu toate acestea, Web-ul este o entitate dinamică, care evoluează rapid. Prin urmare, există o nevoie continuă de crawlere pentru a ajuta aplicațiile să rămână la curent cu paginile și legăturile adăugate, șterse, mutate sau modificate.

Există multe aplicații pentru Web crawlere. Una este în domeniul afacerilor inteligente, prin care organizațiile colectează informații despre concurenții lor și despre potențiali colaboratori. O altă utilizare este monitorizarea paginilor Web de interes, astfel încât un utilizator sau o comunitate să poată fi anunțat(ă) atunci când apar informații noi în anumite locuri. Există, de asemenea, aplicații malware, precum culegătorii de adrese de e-mail sau de alte informații personale, care vor fi utilizate în furturi de identitate. Cu toate acestea, utilizarea cea mai răspândită a crawlerelor este sprijinirea motoarelor de căutare. De fapt, crawlerele sunt principalii consumatori de lățime de bandă pe Internet. Ele colectează paginile Web pentru indexarea în motoarele de căutare. Binecunoscutele motoare de căutare Google, Yahoo! și Bing rulează **crawlere universale** foarte eficiente, concepute pentru a strânge toate tipurile de pagini, indiferent de conținutul lor. Alte crawlere, numite uneori **crawlere preferențiale**, sunt mai direcționate; ele încearcă să descarce numai anumite tipuri de pagini sau subiecte.

3.1. Cel mai simplu algoritm de tip crawler Web

Cel mai simplu crawler Web pornește de la un set de adrese URL și utilizează legăturile găsite în aceste pagini Web pentru a ajunge și la alte pagini. Legăturile din aceste pagini sunt, la rândul lor, extrase și paginile corespondente sunt vizitate. Procesul se repetă până când un număr suficient de pagini sunt vizitate sau este atins alt obiectiv. Această descriere simplă ascunde multe probleme delicate legate de conexiunile de rețea, capcane de agent de căutare, standardizarea URL-urilor, analiza gramaticală de pagină și etica crawler-lor. De fapt, fondatorii Google, Serghei Brin și Lawrence Page, în lucrarea lor de bază, au identificat crawlerul Web ca fiind componenta cea mai sofisticată dar și cea mai fragilă a unui motor de căutare [6].

Figura de mai jos arată fluxul celui mai simplu crawler secvențial. Acest crawler aduce doar o pagină la un moment dat, făcând ineficientă utilizarea resurselor sale. Mai târziu, în acest capitol discutăm despre modul în care poate fi îmbunătățită eficiența prin utilizarea proceselor multiple și a accesului asincron la resurse. Crawler-ul menține o listă de adrese URL nevizitate numită frontieră. Lista este inițializată cu adrese URL care pot fi furnizate de utilizator sau de un alt program. În fiecare iterație a buclei principale, crawlerul selectează următoarea adresă URL din frontieră, descarcă pagina Web care corespunde adresei URL prin protocolul HTTP, analizează conținutul pentru a extrage URL-urile acesteia, adaugă recent descoperitele adrese URL la frontieră și stochează pagina (plus alte informații extrase, eventual cuvinte de index) într-un depozit local de date. Procesul poate fi reziliat atunci când un anumit număr de pagini au fost extrase. Crawlerul poate fi forțat să se oprească și în cazul în care frontiera devine goală, deși acest lucru se întâmplă foarte rar în practică, datorită mediei ridicate a numărului de legături (de ordinul a zece legături pe pagina Web).



Arhitectura unui crawler Web simplu

Un crawler este, în esență, un algoritm de căutare orientat pe grafuri. Web-ul poate fi văzut ca un mare graf orientat unde paginile sunt nodurile sale iar legăturile sunt arcele sale. Un crawler pornește de la câteva noduri și inspectează marginile pentru a ajunge la alte noduri. Procesul de preluare a unei pagini Web și extragerea legăturilor din cadrul acesteia sunt similare cu extinderea unui nod în căutarea bazată pe grafuri.

Frontiera este structura principală de date, care conține adresele URL ale paginilor nevizitate. De obicei, crawlerele încearcă să stocheze frontiera în memoria principală pentru eficiență. Având în vedere prețul scăzut al memoriei RAM și răspândirea procesoarelor pe 64 de biți, devine din ce în ce mai fezabilă o frontieră de dimensiune mare. Cu toate acestea, cei care proiectează crawlere trebuie să decidă care adrese URL au o prioritate scăzută, pentru ca acestea să fie ignorate când frontiera s-a umplut. Rețineți faptul că frontiera este limitată și se va umple rapid, datorită gradului ridicat de pagini externe. Chiar mai important, algoritmul de crawler trebuie să decidă ordinea în care noile adrese URL sunt extrase din frontieră pentru a fi vizitate. Aceste mecanisme determină algoritmul de căutare orientat pe grafuri implementat de crawler.

3.1.1. Algoritmul de căutare în lățime

Frontiera poate fi implementată ca o coadă de tip **FIFO** – prima adresă URL intrată este prima analizată (en: *first in first out*) – care corespunde unui crawler cu căutare în lățime. Algoritmul extrage adresele URL din capul cozii de căutare și adaugă adresele URL noi găsite la finalul cozii. Odată ce frontiera atinge dimensiunea maximă, crawlerul de căutare în lățime poate adăuga în coadă o singură adresă URL nevizitată pentru fiecare pagină Web nou accesată prin crawler.

Strategia de căutare în lățime nu implică faptul că paginile Web sunt vizitate în ordine aleatoare. Intuitiv, putem înțelege de ce paginile Web populare au atât de multe legături către ele, încât atrag crawlerele de căutare în lățime. Prin urmare, nu este surprinzător faptul că ordinea în care paginile Web sunt vizitate de crawler este dată de valorea lor de ***PageRank***. Astfel, este ușor de înțeles de ce crawlerele au predilecția de a indexa în primul rând paginile Web cu multe legături către acestea (***backlink***-uri).

Un alt motiv important pentru care putem spune că vizitarea paginilor Web nu este aleatoare este alegerea paginilor de pornire (en: ***seed pages***) pentru algoritmii de căutare în lățime. Astfel, paginile care au legături directe din paginile de pornire vor fi primele indexate față alte pagini Web aleatoare. Acest aspect stă la baza oricărui crawler universal.

După cum am menționat mai devreme, numai adresele URL nevizitate trebuie adăugate la frontieră. Acest lucru impune necesitatea unei structuri de date care să memoreze adresele URL vizitate,

împreună cu data și ora la care s-a făcut analiza paginei Web respective. Acest control este necesar pentru a evita revizitarea paginilor Web sau pierderea de spațiu în frontiera de dimensiuni reduse. De obicei, o tabelă de tip *HashMap* este potrivită pentru a obține o performanță bună la operațiile de inserare și căutare care au ordin de complexitate O(1). Căutarea presupune identificarea a două adrese URL care să indice către aceeași pagină Web. În acest punct ne dăm seama de necesitatea standardizării adreselor URL.

3.1.2. Algoritmul de căutare preferențial

O altă strategie pentru un crawler Web este implementarea unei frontiere pe baza unei **cozi cu priorități**. Un astfel de crawler va atribui o prioritate fiecărei pagini Web nevizitate încă. Această prioritate este calculată în funcție de anumite proprietăți precum PageRank-ul, sau conținutul paginii Web în raport cu căutarea făcută de utilizatorul motorului de căutare. În acest caz, alegerea paginilor de pornire este cu mult mai importantă decât în cazul algoritmilor de căutare în lățime.

Coada de priorități poate fi un vector dinamic care este întotdeauna sortat după prioritatea adreselor URL. Astfel, adresa URL cu cea mai ridicată prioritate este selectată pentru analiză în fiecare pas. După analiza paginii Web corespunzătoare, fiecărui URL extras îi este asignat o nouă prioritate. Aceste adrese URL sunt adăugate apoi la frontieră în așa fel încât este menținută ordinea de sortare a priorității din coadă. Ca și algoritmii de căutare în lățime, aceste crawlerele trebuie să evite introducerea adreselor URL duplicate în frontieră. Menținerea unei tabele separate de tip HashMap pentru căutare este o modalitate eficientă de a realiza acest lucru. Ordinul de complexitate pentru introducerea unei adrese URL în coada cu priorități este O(logF), unde F este mărimea frontierei. Pentru a extrage o adresă URL, trebuie mai întâi să fie eliminată din coada de priorități – operație cu ordinul de complexitate O(logF) – și apoi din HashMap (din nou O(1)). Prin utilizarea paralelă a celor două structuri de date, costul total este logaritmic per URL. Odată ce dimensiunea maximă a frontierei este atinsă, numai cele mai bune adrese URL sunt păstrate; frontiera trebuie să fie ajustată după adăugarea fiecărui set nou de adrese URL.

3.2. Probleme de implementare

3.2.1. Extragerea paginilor Web

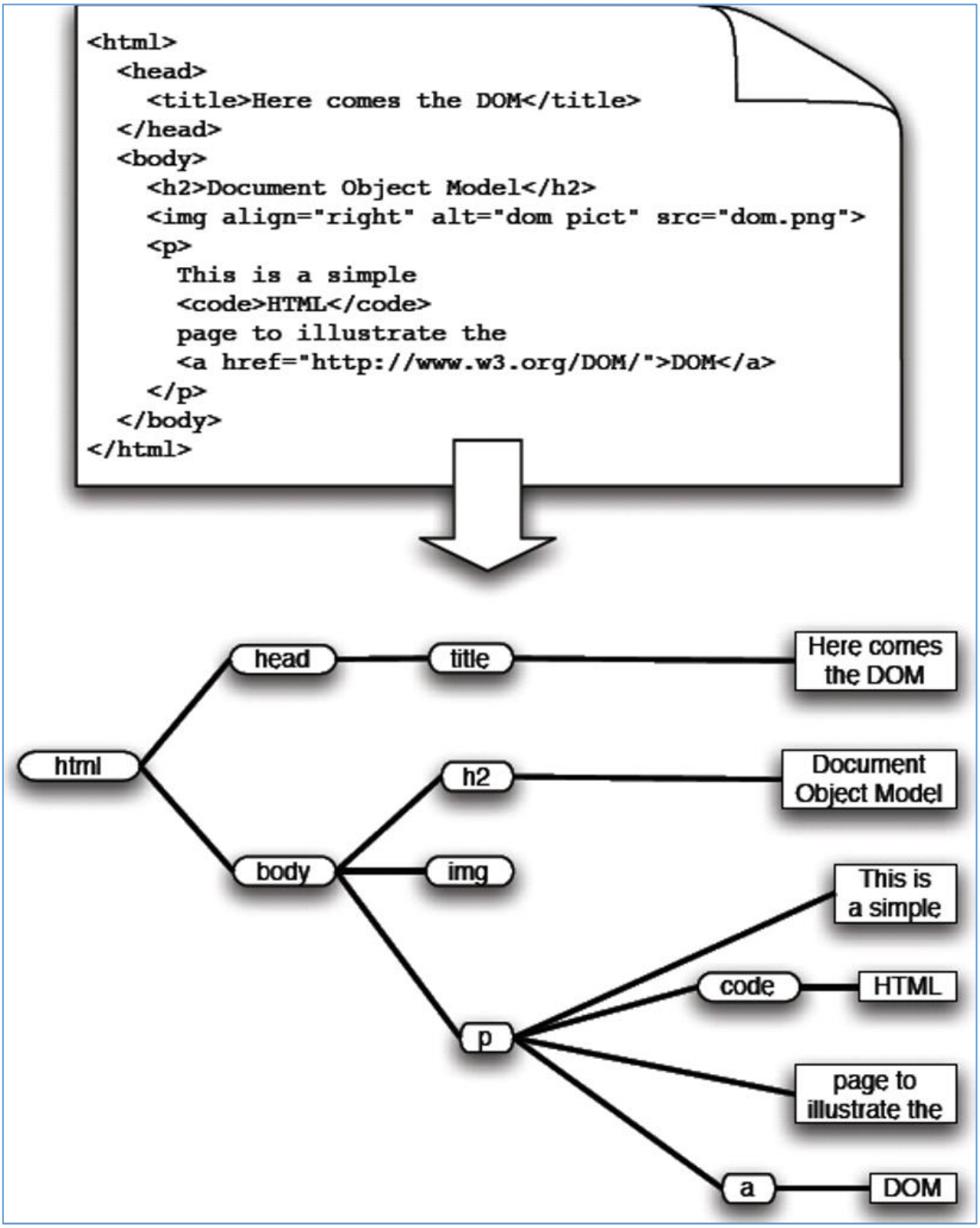
Pentru a extrage paginile Web, un crawler acționează ca un client Web; el trimite o cerere HTTP către serverul Web care găzduiește pagina și interpretează răspunsul. Clientul Web are nevoie să limiteze timpul de așteptare pentru a preveni întârzierile generate de serverele lente sau citirea unor pagini uriașe. De fapt, se obișnuiește restricționarea descărcărilor la primii 10-100 KB de date din fiecare pagină Web.

Crawlerul Web analizează codurile de stare din răspunsul HTTP și redirecționările. Buclele de redirecționare trebuie detectate și întrerupte, pentru a evita extragerea și analiza unei adrese URL de două ori. De asemenea, se poate analiza și memora ultimul antet de răspuns HTTP pentru a determina vârsta documentului, deși se știe că aceste informații sunt nesigure.

Verificarea erorilor și tratarea excepțiilor sunt importante în timpul procesului de extragere a paginii Web. Limbaje de programare precum Java, Python și Perl oferă interfețe API simple pentru extragerea paginilor Web. Cu toate acestea, trebuie alese cu atenție interfețele API de nivel înalt, unde este dificilă detecția problemelor la nivel inferior. De exemplu, un crawler robust în Perl ar trebui să utilizeze modulul Socket – pentru a trimite cereri HTTP – în locul bibliotecii LWP de nivel superior (biblioteca World Wide Web pentru Perl, care nu permite setarea *timeout*-ului pe conectare).

3.2.2. Analiza paginilor Web

După ce (sau în timp ce) o pagină Web este descărcată, crawlerul îi analizează conținutul (de ex. indexează pagina, dacă crawlerul susține un motor de căutare) și extrage toate adresele URL pe care le adaugă la frontieră. Toate paginile HTML au o structură arborescentă de tag-uri HTML, în format DOM (en: **Document Object Model**) [7, 14], precum este prezentat în Figură 3.



Figură 3. Ilustrarea arborelui DOM (i.e. de tag-uri) construit dintr-o pagină HTML simplă. Nodurile interne (reprezentate prin forme ovale) reprezintă tag-urile HTML, cu tagul <html> ca rădăcină. Nodurile frunze (reprezentate prin dreptunghiuri) corespund bucăților de text.

Spre deosebire de o aplicație executabilă unde codul sursă trebuie să se compileze corect, fără greșeli de sintaxă, browserele Web sunt foarte permisive din punct de vedere al corectitudinii sintaxei HTML. Pe lângă acest aspect, numărul mare de autori de pagini Web, care nu cunosc suficient de bine standardul HTML, impune o complexitate mare crawlerelor care fac analiza de conținut Web. Multe pagini Web sunt publicate cu taguri obligatorii lipsă, taguri imbricate necorespunzător, taguri scrise greșit sau cu nume de atribute și valori lipsă, ș.a.m.d. De exemplu, ghilimelele duble în HTML sunt rezervate pentru sintaxa de taguri și este interzisă folosirea lor în text. Entitatea specială HTML **`&quot;`** trebuie folosită în locul său. Cu toate acestea, doar un număr mic de autori sunt conștienți de acest lucru, iar o mare parte din paginile Web au conținut incorect în raport cu sintaxa HTML. La fel ca browserele, crawlerele trebuie să treacă cu vederea aceste cazuri, pentru a nu renunța la multe pagini importante, așa cum o analiză strictă ar cere-o. În general, crawlerele apelează la un tool ca **tidy** pentru a curăța conținutul HTML înainte de analiză (https://www.w3.org/People/Raggett/tidy/). Astfel, analizatorii HTML disponibili în limbajele Java și Perl sunt din ce în ce mai sofisticați.

O parte din conținutul Web este scris în alte formate decât HTML. Crawlere care susțin principalele motoare de căutare indexează și documente scrise în formate proprietare precum Adobe PDF, Microsoft Word, Microsoft Excel și Microsoft PowerPoint. Unele websituri comerciale utilizează animații grafice în Adobe Flash, care sunt dificil de analizat pentru un crawler. Alte exemple asemănătoare sunt paginile Web care fac abuz de JavaScript pentru interacțiune. Noi provocări apar datorită noilor standarde introduse, precum *Scalable Vector Graphics* (SVG), *Asynchronous Javascript and XML* (AJAX) și alte limbaje bazate pe XML, care capătă popularitate.

3.2.3. Standardizarea adreselor URL

Analiza HTML permite identificarea etichetelor și a perechilor atribut-valoare asociate într-o pagină Web. Pentru a extrage o legătură (en: *hyperlink*) dintr-o pagină Web, putem căuta tagul ancoră **`<a>`** și să luăm valoarea asociată atributului **`href`**. Dar chiar și așa, în anumite cazuri această valoare mai necesită niște prelucrări. Un crawler clasic folosește *"liste albe"* – pentru a identifica conținutul urmărit (de exemplu, fișiere text și HTML) – și *"liste negre"* – pentru a identifica conținutul ignorat (de exemplu, fișiere binare de date, precum Adobe PDF). În general, identificarea tipului de conținut se face pe baza extensiilor de fișiere, dar această modalitate este nesigură; nu ne putem permite să descărcăm un document și apoi să decidem dacă îl vrem sau nu. O alternativă mai bună este de a trimite o solicitare de antet HTTP și a inspecta tipul conținutului din antetul de răspuns.

Altă filtrare necesară pentru un crawler are de-a face cu natura statică sau dinamică a paginii Web. O pagină dinamică (de exemplu, generată de un script CGI) poate indica o interfață de interogare pentru o bază de date, care nu prezintă interes pentru crawler. În primii ani de la apariția Internetului, astfel de pagini erau ușor de recunoscut întrucât adresa URL conținea directorul **`/cgi-bin/`** și folosea caracterele speciale **`?`** și **`&`**. Însă, cu timpul, a devenit din ce în ce mai greu de recunoscut paginile dinamice prin inspecția adresei URL. Din aceste motive, cele mai multe crawlere nu mai fac distincție între conținutul static și cel dinamic. Există o excepție importantă în această abordare, *capcana de păianjen* (en: ***spider trap***), care este discutată mai jos, în următoarea secțiune.

Încă o prelucrare suplimentară este necesară înainte de a adăuga legături la frontieră: **adresele URL relative** trebuie convertite în **adrese URL absolute**. De exemplu, adresa URL relativă stiri/azi.html din pagina http://www.domeniu.com/index.html trebuie transformată în forma absolută http://www.domeniu.com/stiri/azi.html. Sunt mult mai multe reguli de a converti adresele URL relative în cele absolute, care vor fi discutate ulterior. De exemplu, o adresă URL relativă poate fi exprimată ca o cale relativă sau absolută față de documentul director rădăcină al serverului Web. **Adresa URL de bază** poate fi specificată în antetul HTTP sau de o meta-etichetă într-o pagină HTML sau deloc – caz în care directorul paginii sursă a hyperlink-ului este utilizat ca adresă URL de bază.

Conversia adreselor URL relative în adrese URL absolute este doar un pas necesar pentru standardizarea adreselor URL. Crawlerele trebuie să aplice însă mult mai multe reguli. De exemplu, un host poate specifica întotdeauna numărul de port din cadrul adresei URL (de exemplu, http://www.domeniu.com:80/), în timp ce altul poate specifica numărul de port numai atunci când nu este 80 (portul HTTP implicit). Este important ca un crawler să fie consistent în aplicarea acestor reguli. Unele limbaje de programare, precum Perl, oferă module dedicate pentru prelucrarea adreselor URL, inclusiv metode de conversie absolută/relativă și de standardizare. Uneori este necesar aplicarea euristică a acestor reguli pentru a detecta când două URL-uri indică spre aceeași pagină, pentru a minimiza probabilitatea ca aceeași pagină să fie analizată de mai multe ori.

Descrierea și *transformarea*	**Exemplu și forma standard**
Numărul portului implicit *Eliminare*	http://bdsa.ase.ro:80/ http://bdsa.ase.ro/
Directorul rădăcină *Adăugare backslash*	http://bdsa.ase.ro http://bdsa.ase.ro/
Ghicesc directorul *Adăugare backslash*	http://bdsa.ase.ro/People http://bdsa.ase.ro/people/
Fragmente *Eliminare*	http://bdsa.ase.ro/faq.html#3 http://bdsa.ase.ro/faq.html
Directorul curent sau părinte *Rezolvă calea*	http://bdsa.ase.ro/a/./../b/ http://bdsa.ase.ro/b/
Numele de fișier implicit *Eliminare*	http://bdsa.ase.ro/index.html http://bdsa.ase.ro/
Caractere codificate inutil *Decodificare*	http://bdsa.ase.ro/%7Efil/ http://bdsa.ase.ro/~fil/
Caracterele nepermise *Codificare*	http://bdsa.ase.ro/My File.htm http://bdsa.ase.ro/my%20File.htm
Nume de domeniu mixt (litere mari) *Litere mici*	http://BDSA.ASE.RO/People/ http://bdsa.ase.ro/people/

Tabel 1. Unele transformări necesare pentru standardizarea adreselor URL

3.2.4. Capcana de păianjen

Un Web crawler trebuie să evite ***capcana de păianjen***. Acestea sunt websituri unde adresele URL sunt modificate dinamic în funcție de secvența acțiunilor efectuate de utilizator (sau de crawler). Unele websituri de comerț electronic, precum amazon.com, utilizează adrese URL pentru a codifica secvența de produse pe care fiecare utilizator o vizualizează. În acest fel, de fiecare dată când un utilizator dă clic pe un link, serverul adăugă în adresa URL informații detaliate despre comportamentul cumpărătorului, pentru analize ulterioare. Ca o ilustrare, luați în considerare o pagină dinamică pentru produsul x, a cărui cale URL este **`/x`** și care conține o legătură cu produsul y. Calea URL-ului pentru această legătură ar fi **`/x/y`** pentru a indica faptul că utilizatorul merge de la pagina x la pagina y. Acum să presupunem că pagina pentru y are un link înapoi la produsul x. Calea URL creată dinamic pentru această legătură ar fi **`/x/y/x`**, astfel încât un crawler ar crede că aceasta este o pagină nouă când, de fapt, este o pagină deja vizitată dar cu o altă adresă URL. Ca efect secundar al capcanei de păianjen, serverul poate crea o intrare într-o bază de date de fiecare dată când utilizatorul (sau crawlerul) face clic pe anumite legături dinamice. Un exemplu relevant sunt blogurile și forumurile, unde utilizatorii pot posta comentarii. Aceste websituri par infinite pentru un crawler, deoarece cu cât sunt urmate mai multe linkuri, cu atât sunt create mai multe adrese URL noi. Însă aceste noi linkuri false nu conduc la conținut nou, ci pur și simplu la pagini care au fost deja vizitate. În acest fel, un crawler s-ar pierde înăuntrul capcanei păianjen, fără a mai extrage conținut nou.

În practică, capcanele de păianjen nu sunt dăunătoare numai pentru crawler, care irosește lățime de bandă și spațiu pe disc pentru a descărca și a indexa datele duplicate/inutile. Acestea pot fi la fel de dăunătoare și pentru serverul care găzduiește domeniul respectiv. În cazul unui crawler prins în capcana de păianjen, serverul consumă din ce în ce mai multe resurse pentru a răspunde solicitărilor primite, ajungând în situația de *respingere a serviciilor* (en: ***denial of service***).

În unele cazuri, capcana de păianjen apare doar în cazul în care serverul necesită trimiterea unui cookie, pe baza căruia se formează adresele URL dinamice. Deci, problema nu apare dacă crawlerul evită acceptarea sau trimiterea cookie-urilor. Cu toate acestea, în cele mai multe cazuri este necesară o abordare pro activă pentru a proteja un crawler de capcanele de păianjen. Dat fiind faptul că adresele false devin adesea din ce în ce mai mari, pe măsură ce crawlerul se încurcă într-o capcană de păianjen, o abordare euristică comună este limitarea dimensiuni adreselor URL la un număr maxim de caractere, de exemplu 256. Dacă se întâlnește o adresă URL mai lungă, crawlerul trebuie să o ignore pur și simplu. O altă modalitate este limitarea numărului de pagini pe care crawlerul le solicită dintr-un anumit domeniu. Codul asociat cu frontiera se poate asigura că fiecare secvență consecutivă, de exemplu 100 de adrese URL preluate de crawler, conțin cel mult o adresă URL de la fiecare nume de gazdă complet calificat. Această abordare este, de asemenea, legată de problema etichetei de crawler, discutată mai târziu.

3.2.5. Depozitul de pagini Web

Odată ce o pagină Web este descărcată, aceasta poate fi stocată și indexată pentru aplicația principală (de exemplu, un motor de căutare). Cel mai simplu mod este a memora conținutul fiecărei pagini Web într-un fișier pe disc. În acest caz, fiecare pagină Web trebuie mapată la un nume unic de fișier. O modalitate simplă de a face acest lucru este de a folosi o funcție hash (de exemplu, algoritmul MD5). Astfel, valoarea hash rezultată este un nume unic de fișier. Deficiența acestei soluții apare în cazul unui crawler care încearcă să stocheze un număr foarte mare de pagini Web, întrucât sistemul de operare trebuie să gestioneze un număr imens de fișiere mici.

O soluție mai eficientă este combinarea mai multor pagini Web într-un fișier. O abordare naivă este de a concatena pur și simplu un număr de pagini (de exemplu, 1000) în fiecare fișier, cu un marcaj special pentru a separa și identifica paginile în dosar. Acest lucru necesită un tabel separat de căutare pentru a mapa adresele URL la numele fișierelor și ID-urile din fiecare fișier.

O metodă mai bună este utilizarea unei baze de date pentru stocarea paginilor Web, indexate pe baza adresei URL standardizată. Multe sisteme relaționale de baze de date sunt accesibile din limbaje de nivel înalt, precum Java, Perl și PHP, astfel încât operațiile de gestionare devin transparente pentru codul de crawler, care poate trata depozitul de pagini Web ca o structură de date în memorie.

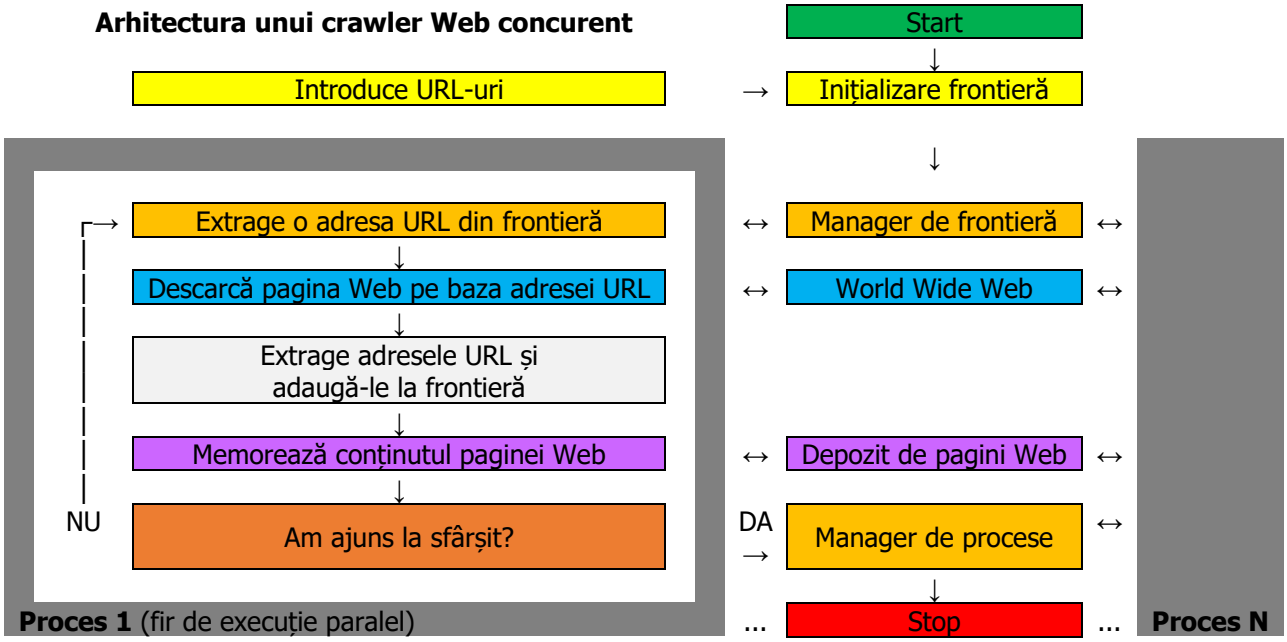
3.2.6. Concurența

Un crawler dispune de trei resurse principale: rețea, CPU și HDD. Toate acestea sunt limitate de lărgimea de bandă, viteza microprocesorului și timpul de citire/scriere al discului. Cel mai simplu crawler descris anterior utilizează ineficient aceste resurse, deoarece la un moment dat două dintre ele sunt inactive, în timp ce crawlerul o folosește pe a treia.

Cel mai simplu mod de a accelera un crawler este prin intermediul proceselor concurente (sau fire de execuție paralele), astfel încât fiecare fir sau proces funcționează ca un crawler independent, cu excepția faptului că accesul la structurile de date partajate (în principal frontiera și depozitul de pagini Web) trebuie să fie sincronizate.

Este un pic mai complicat pentru un crawler concurent să se ocupe de o frontieră goală decât un crawler secvențial. O frontieră goală nu mai implică faptul că crawlerul a ajuns la final, deoarece alte procese

pot descărca pagini Web și adăuga noi adrese URL în viitorul apropiat. Managerul de procese (sau fire de execuție paralele) poate face față unei astfel de situații prin trimiterea unui semnal temporar de suspendare a proceselor care raportează o frontieră goală; managerul de proces trebuie să țină evidența numărului de procese de suspendate, iar când toate procesele sunt suspendate, crawlerul trebuie să se oprească.



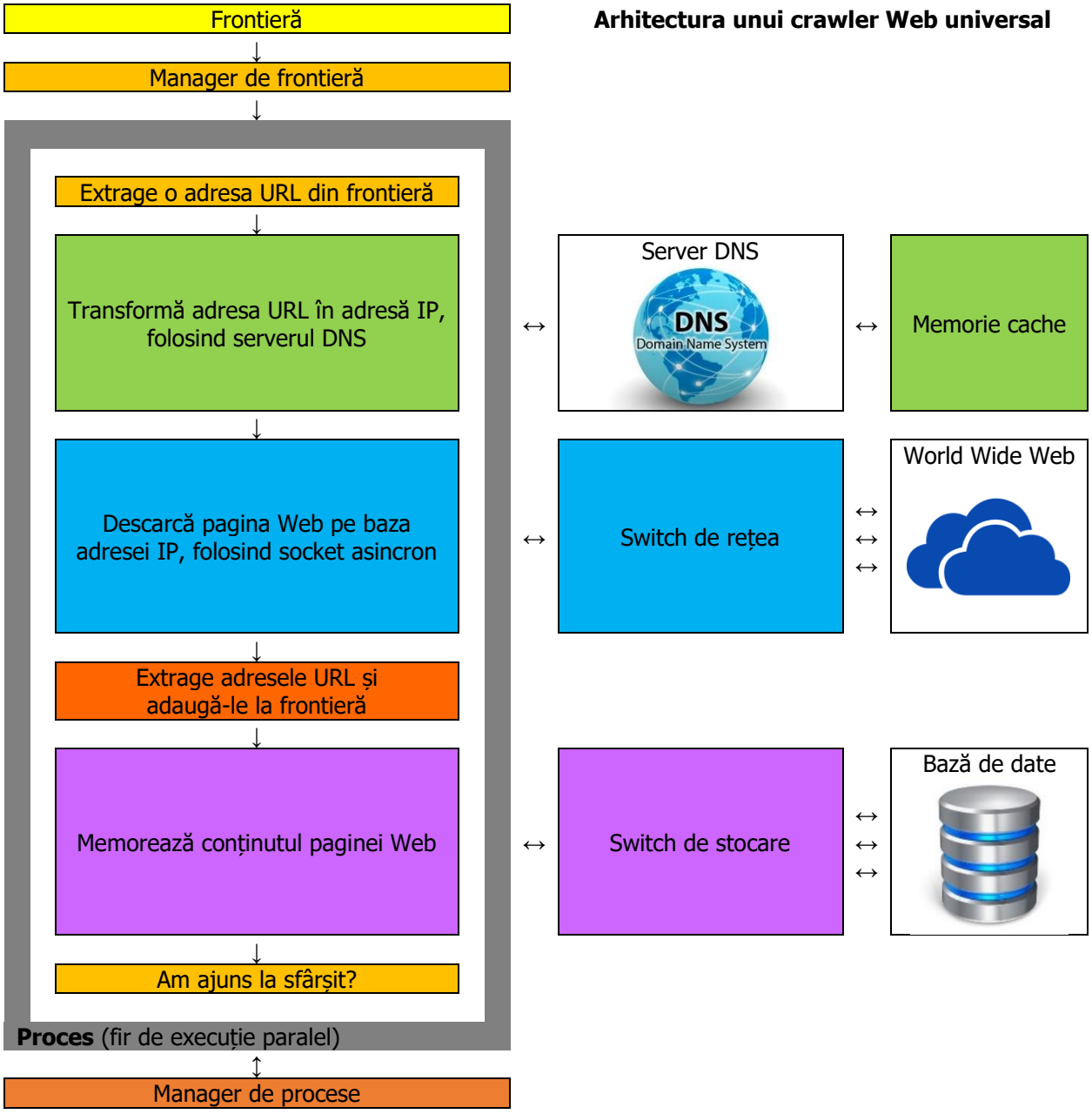
3.3. Crawlere universale

Motoarele de căutare cu scop general folosesc crawlere universale pentru a-și menține baza de date la zi [8], care indexează miliarde de pagini Web, fiind în paralel disponibile pentru diverse interogări [9]. Aceste crawlere universale diferă de cele concurente cu căutare în lățime prin două caracteristici esențiale:

1. *Performanța* – este important ca acestea să extragă și să prelucreze milioane de pagini Web pe secundă; acest aspect este realizabil prin anumite modificări arhitecturale;
2. *Politica* – este necesar ca acestea să indexeze cele mai relevante pagini Web pentru a menține baza de date la zi.

Figura de mai jos ilustrează arhitectura unui crawler universal performant. Cea mai importantă deosebire față de modelul concurent discutat mai devreme este folosirea ***socket***-urilor *asincroni*. Față de cei sincroni, acești socketi sunt non-blocanți, astfel încât un singur proces (sau fir de execuție paralel) poate păstra sute de conexiuni simultan deschise și folosește eficient lățimea de bandă a rețelei. În acest model, nu mai este necesară sincronizarea accesului la resursele de date partajate. În plus, managerul de frontieră poate fi îmbunătățit prin menținerea mai multor cozi de adrese URL, astfel încât fiecare coadă se referă la un singur server Web.

Crawlerul trebuie să rezolve numele de gazdă din adresa URL în adresă IP. Acest lucru este posibil prin interogarea serverului DNS (en: ***Domain Name System***). Accesul la DNS pentru rezolvarea fiecărei adrese URL întârzie considerabil un crawler naiv. O posibilă soluție este folosirea conexiunilor UDP, în locul celor TCP, deși este cunoscut faptul că acestea nu garantează transferul pachetelor în rețea. În al doilea rând, ar fi indicată utilizarea unui server DNS local cu o memorie cache mare, persistentă și rapidă. În al treilea rând, este necesar accesul la rețea prin mai multe routere, folosind furnizori diferiți de Internet. Nu în ultimul rând, păstrarea paginilor indexate se poate face printr-un switch de stocare folosind fibra optică ca mediu de transmisie.

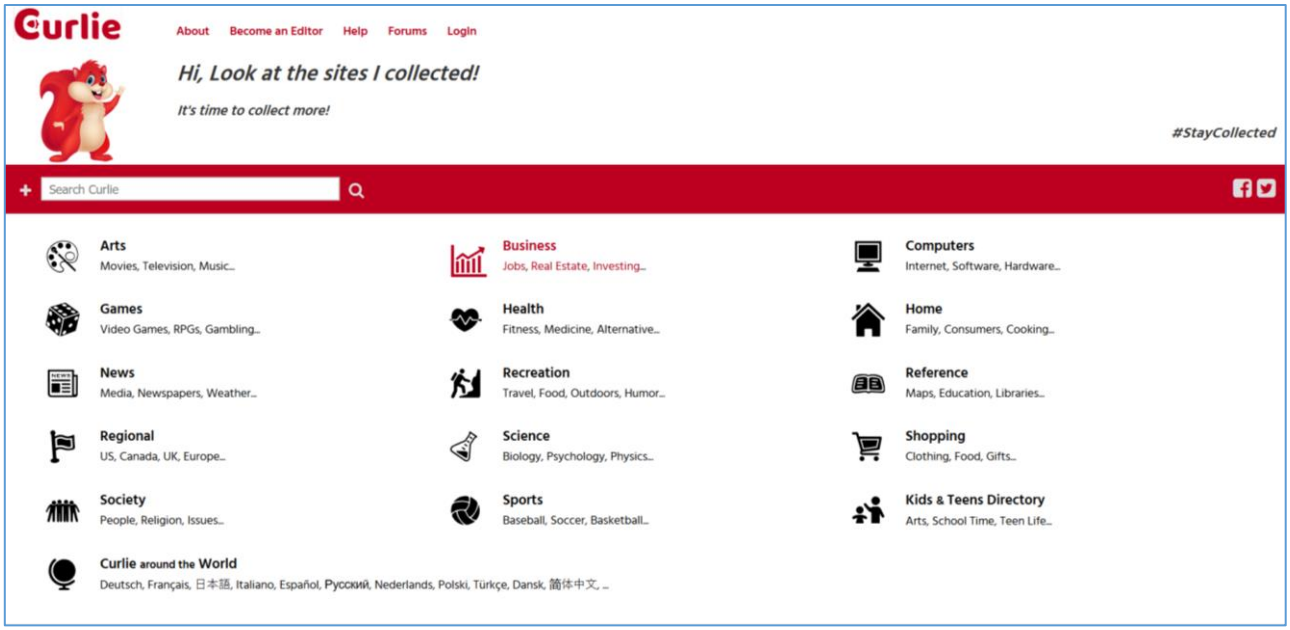


Având în vedere dimensiunea Internetului, nu este fezabil să fie indexate toate paginile Web disponibile. De aceea, principalele motoare de căutare încearcă să se concentreze pe cele mai "importante" pagini, unde importanța este evaluată pe baza unor factori precum PageRank-ul [9, 10]. În prezent, principalele motoare de căutare comerciale au indexate aprox. 10^{10} pagini Web, deși numărul paginilor disponibile este cu cel puțin un ordin de mărime mai mare.

Obiectivul unui motor de căutare de a indexa cât mai multe pagini Web (adică pe cele mai "importante" pagini) este în contradicție cu necesitatea de a ține baza de date actualizată. Datorită naturii dinamice a Internetului, în care paginile Web sunt zilnic adăugate, șterse sau modificate, un crawler trebuie să reviziteze toate paginile deja indexate pentru a ține baza de date actualizată. Au fost efectuate mai multe studii cu scopul de a analiza dinamica Internetului, adică proprietățile statistice ce conduc la modificarea structurii și conținutului paginilor Web. Conform acestora, rata de adăugare a paginilor noi este de 8% pe săptămână și numai 62% din conținutul acestor pagini este cu adevărat nou [11]. Însă structura de legături a Internetului este mult mai dinamică: săptămânal sunt adăugate aprox. 25% ***hyperlink***-uri noi. Odată create, paginile Web tind să se schimbe foarte puțin, astfel încât majoritatea modificărilor observate pe Internet se datorează mai mult adăugărilor și ștergerilor. De aceea, gradul de modificare a unei pagini este un predictor mai bun decât frecvența schimbării [12].

3.4. Crawlere preferențiale

O strategie mult mai bună este analiza anumitor categorii de pagini Web decât analiza tuturor paginilor disponibile, folosind un crawler preferențial. O aplicație pentru un astfel de crawler este menținerea unei taxonomii Web, precum directorul ***Curlie*** (https://curlie.org/), succesorul lui ***Open Directory Project*** (http://www.dmoz.org/). În acest caz, un crawler preferențial ar menține taxonomia Web la zi întrucât are grijă doar de paginile Web din anumite categorii.



Figură 4. Directorul Web *Curlie - The Collector of URLs*

Conform [15], un crawler preferențial poate fi implementat pe baza unui clasificator de Data Mining. Astfel, clasificatorul va ghida crawlerul preferențial să selecteze din frontieră acele pagini care sunt cel mai apropriate de anumite categorii de interese, conform previziunilor clasificatorului. Pentru a antrena clasificatorul, paginile de tip exemplu sunt extrase din categorii diferite de taxonomie, după cum se arată în Figură 5 (stânga). Algoritmul de clasificare folosit este ***metoda Bayesiană***.

Pentru fiecare categorie c din taxonomie putem construi un clasificator Bayesian pentru a calcula probabilitatea $Pr(\mathrm{c} \mid \mathrm{p})$ ca pagina p să aparțină lui c (prin definiție, probabilitatea $Pr(\mathrm{top} \mid \mathrm{p}) = 1$ pentru categoria de bază). Utilizatorul poate selecta un set c^* de categorii de interes. Fiecărei pagini accesate prin crawlere i se atribuie un scor de relevanță: $R(p) = \sum_{c \in c^*} Pr(\mathrm{c} \mid \mathrm{p})$.

Au fost explorate două strategii. În strategia *"soft"*, crawlerul utilizează scorul $R(p)$ al fiecărei pagini p accesate prin crawlere ca valoare prioritară pentru toate adresele URL nevizitate extrase din pagina p. Adresele URL sunt apoi adăugate la frontieră care este tratată ca o coadă de priorități (vezi secțiunea 3.1.2). În strategia *"hard"*, pentru o pagină p accesată prin crawlere, clasificatorul găsește mai întâi categoria $\hat{c}(p)$ cea mai probabilă să includă pagina p: $\hat{c}(p) = \arg max_{c' \in c} \big(Pr(\mathrm{c} \mid \mathrm{p})\big)$. Ambele strategii au dat rezultate bune în experimentele făcute.

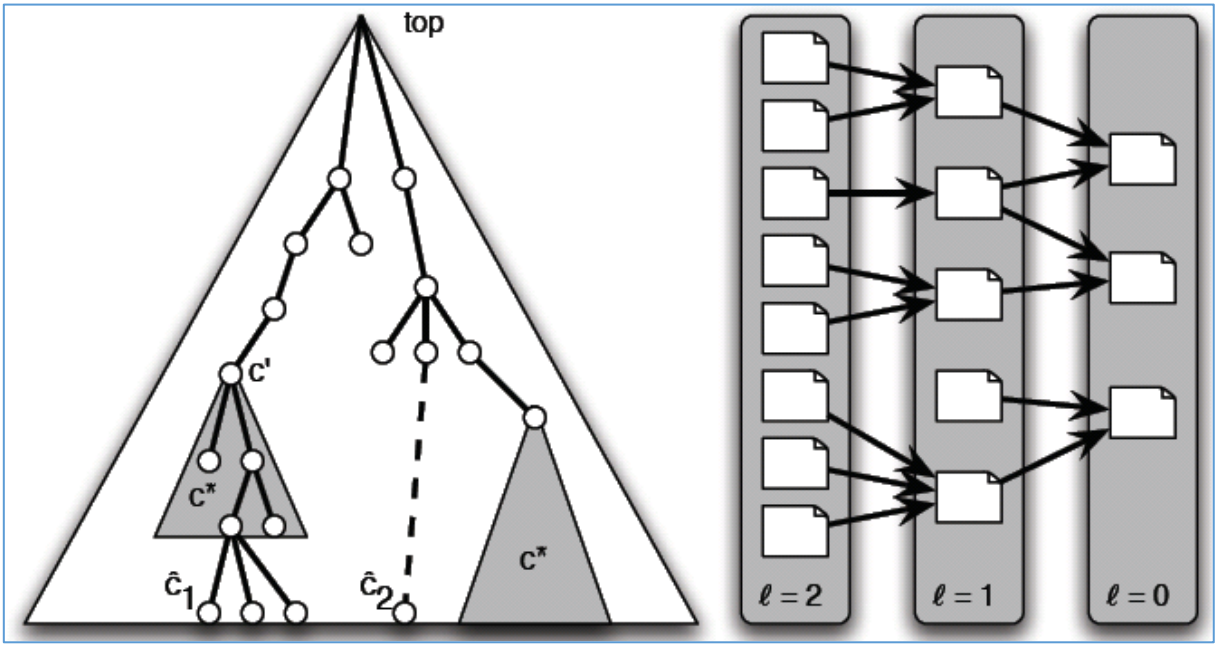
Conform [16], ***crawlere orientate pe context*** sunt un alt tip de crawlere preferențiale. Acestea folosesc tot metoda Bayesiană pentru selecție, dar clasificatorul este instruit pentru a estima distanța legăturii Web dintre o pagină accesată prin crawlere și un set de pagini țintă relevante. Să luăm, ca exemplu de căutare, cuvintele "învățare automată" (en: ***machine learning***). Acestea conduc de obicei la paginile departamentelor de informatică și de acolo la pagina principală a facultății, apoi la pagini Web și documente publice relevante. Însă pagina departamentului nu conține de obicei aceste cuvinte cheie. Un crawler preferențial tipic ar da unei astfel de pagini o prioritate scăzută și, probabil, nu ar urma

niciodată legăturile sale Web. Cu toate acestea, dacă un crawler ar putea estima că paginile despre "învățare automată" sunt la doar două legături distanță de o pagină conținând cuvintele-cheie "departamentul de informatică", atunci ar fi dat paginii departamentului o prioritate mai mare.

Crawlerul orientat pe context este antrenat folosind un grafic de context cu L straturi, ca în Figură 5 (dreapta). Paginile țintă formează stratul $l = 0$ din grafic. Paginile cu legături spre paginile țintă formează stratul $l = 1$. Paginile cu legături spre paginile din stratul 1 formează stratul $l = 2$ ș.a.m.d. Paginile țintă din stratul 0 (și posibil cele din stratul 1) sunt apoi concatenate într-un singur document mare, iar primele cuvinte sunt selectate ca vocabular pentru clasificare. Clasificatorul Bayesian este construit pentru fiecare strat din graficul de context. Inițial, o probabilitate $Pr(l) = \frac{1}{L}$ este atribuită fiecărui strat. Pentru toate paginile dintr-un strat, se calculează $Pr(\mathrm{t} \mid l)$, adică probabilitatea apartenenței unui termen *t* la stratul (clasa) *l*. În momentul analizei, acestea sunt folosite pentru a calcula $Pr(\mathrm{p} \mid l)$ pentru fiecare pagină *p*. Astfel, probabilitatea ca pagina *p* să aparțină stratului (clasei) *l* se calculează cu regula Bayer; stratul *l** cu cea mai ridicată probabilitatea posterioară este selectat: $l^*(p) = \arg max_l\big(Pr(l \mid \mathrm{p})\big)$. Dacă $Pr(l^* \mid \mathrm{p})$ este mai mică decât o valoare prag, atunci pagina *p* rămâne neclasificată, iar dacă $Pr(l^* \mid \mathrm{p})$ depășește pragul, pagina *p* este clasificată în stratul (clasa) *l**.

Setul de clasificatori care corespund graficului de context furnizează un mecanism pentru a estima distanța unei pagini accesată cu crawlere de o pagină relevantă. Dacă mecanismul funcționează, pagina departamentului de informatică din exemplul nostru va fi clasificată în stratul 2. Crawler-ul păstrează o coadă de priorități separate pentru fiecare strat, care conține legăturile extrase din paginile vizitate și clasificate în acel strat. Fiecare coadă este ordonată după scoruri $Pr(l \mid \mathrm{p})$. Următoarea adresă URL selectată pentru analiză este luată din coadă, care are cel mai mic *l*. Deci, crawlerul dă prioritate legăturilor care par a fi cel mai aproape de paginile relevante.

În timp ce majoritatea crawlerelor preferențiale din literatură folosesc metoda Bayesiană ca algoritm pentru clasificarea adreselor URL nevizitate, există dovezi puternice că clasificatorii bazați pe SVM sau rețele neuronale pot produce îmbunătățiri semnificative ale calității paginilor accesate cu crawlere, conform [17].



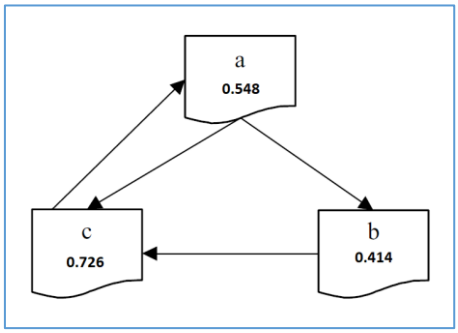
Figură 5. *Stânga:* o taxonomie pentru un crawler Web preferențial.
Dreapta: Un graf cu 3 nivele necesar pentru a antrena un crawler preferențial.

3.5. Algoritmul PageRank

Legăturile către o pagină Web sunt foarte importante, întrucât acestea sunt un indiciu despre autoritatea pe care pagina respectivă o deține, așa cum mențiunile unei lucrări academice atrag atenția asupra sa cu cât este mai citată de alți autori. Autoritatea (sau "popularitatea") unei pagini Web este evaluată de motorul de căutare Web atunci când este indexată. Astfel, Internetul poate fi comparat cu o rețea socială, în care paginile/documentele formează nodurile sale, iar legăturile Web formează arcele unui graf orientat.

Să notăm cu A matricea de legături Web, astfel încât $A(u, v) = 1$ dacă există o legătură de la pagina u la pagina v sau $A(u, v) = 0$ altfel. Fiecare nod u are un ***scor de prestigiu*** notat cu $p(u)$, care este definit ca fiind suma scorurilor de prestigiu ale tuturor nodurilor care citează pagina u: $p(u) = \sum_v A(v, u)p(v)$.

În notație matriceală, scorul de prestigiu $p(u)$ al tuturor paginilor u poate fi scris ca vectorul coloană P. Dat fiind vectorul de prestigiu inițial P, noul vector de prestigiu este $P' = A^T P$ (are o natură recursivă), unde A^T este matricea transpusă a lui A. După un anumit număr de iterații găsim P, astfel încât să putem calcula soluția ecuației $\lambda P = A^T P$.



Figură 6. O rețea de pagini Web cu scoruri de prestigiu.

Să luăm exemplul din Figură 6, pentru care matricea $A = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$, iar $A^T = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}$.

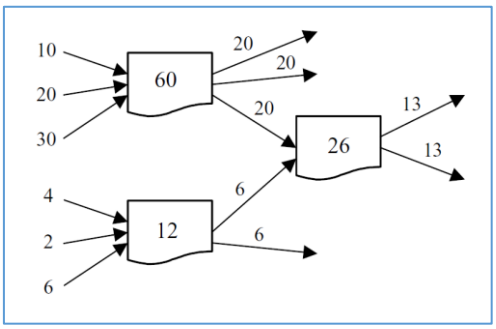
Soluția este valoarea proprie $\lambda = 1.325$ și vectorul propriu $P = (0.548 \quad 0.414 \quad 0.726)$, care confirmă ceea ce puteam intui încă de la început: pagina c are un scor de prestigiu mai mare pentru că are două legături către ea; paginile a și b au scoruri de prestigiu mai mici, deoarece fiecare au o singură legătură către ele.

$$1.325 \begin{pmatrix} 0.548 \\ 0.414 \\ 0.726 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0.548 \\ 0.414 \\ 0.726 \end{pmatrix}$$

Un algoritm simplu pentru rezolvarea ecuației este:

- $P \leftarrow P_0$
- iterează
  - $Q \leftarrow P$
  - $P \leftarrow A^T Q$
  - $P \leftarrow \frac{1}{\|P\|} P$
- cât timp $\|P - Q\| > \varepsilon$, unde ε este un parametru care controlează acuratețea soluției.

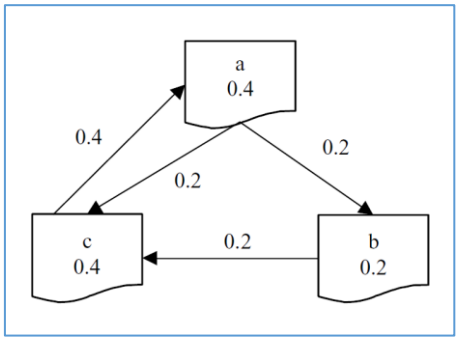
În realitate, lucrurile sunt mult mai complexe. Să luăm cazul în care o pagină u are legături către N_u pagini Web, iar v este doar una dintre ele. Putem concluziona că pentru un utilizator normal probabilitatea de a vizita pagina v în momentul în care se află pe pagina u este $\frac{1}{N_u}$. Intuitiv realizăm că acest algoritm duce la o schemă mult mai complexă de propagare a scorului de prestigiu, care implică atât legăturile spre, cât și dinspre paginile Web analizate. Adică scorul de prestigiu al paginei v este $\frac{1}{N_u}$ din scorul de prestigiu al paginei u. Conform [18], chiar acest concept stă la baza ***algoritmului PageRank*** folosit de Google (vezi Figură 7).



Figură 7. Propagarea PageRank-ului folosit de Google.

Revenind la exemplul nostru inițial, vom rescrie astfel matricea $A = \begin{pmatrix} 0 & 0.5 & 0.5 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$. Pentru fiecare pagină u autoritatea sa va fi $R(u) = \lambda \sum_v \frac{A(v,u)R(v)}{N_v}$, unde N_v este numărul de legături externe din pagina v (vezi Figură 8). Algoritmul PageRank folosește aceeași ecuație pentru calculul scorurilor de prestigiu: $\lambda P = A^T P$, iar soluția de mai jos este calculată pentru $\lambda = 1$.

$$\begin{pmatrix} 0.4 \\ 0.2 \\ 0.4 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0.5 & 0 & 0 \\ 0.5 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0.4 \\ 0.2 \\ 0.4 \end{pmatrix}$$



Figură 8. O rețea de pagini Web folosind algoritmul PageRank.

Întrucât Internetul conține și bucle formate din legăturile paginilor Web, cei de la Google au introdus noțiunea de *rank sursă* E, care definește probabilitatea ca un utilizator să sară la o pagină Web la întâmplare. Astfel, ecuația de calcul PageRank devine $R(u) = \lambda \left[\sum_v \frac{A(v,u)R(v)}{N_v} + E(u) \right]$.

Un algoritm simplu pentru rezolvarea ecuației este:

- $R \leftarrow R_0$
- iterează
  - $Q \leftarrow R$
  - $R \leftarrow A^T Q$

- $d \leftarrow \|Q\|_1 - \|R\|_1$
- $R \leftarrow R + dE$

- cât timp $\|R - Q\|_1 > \varepsilon$, unde ε este un parametru care controlează acuratețea soluției.

3.6. Indexarea paginilor Web

Înainte de a indexa o pagină Web, un crawler transformă textul în litere mici, elimină semnele de punctuație și prepozițiile, după care conținutul rămas poate fi indexat folosind 3 metode distincte: metoda booleană, frecvența cuvintelor (en: ***term frequency***) și inversul frecvenței documentului (en: ***term frequency – inverse document frequency***).

Fie n documente, notate cu $d_1, d_2, \dots, d_n$ și m cuvinte, notate cu $t_1, t_2, \dots, t_m$, iar n_{ij} numărul de apariții al termenului t_i în documentul d_j. Astfel, în ***metoda booleană***, documentul d_j este reprezentat ca un vector cu m componente $d_j = \left(d_j^1 d_j^2 \dots d_j^m\right)$, unde $d_j^i = \begin{cases} 0, \ dacă\ n_{ij} = 0 \\ 1, dacă\ n_{ij} > 0 \end{cases}$. Această metodă este foarte bună pentru clasificare și clustering de documente, dar nu este potrivită pentru căutarea unor cuvinte cheie, ca în cazul unui motor de căutare Web.

În metoda bazată pe calculul *frecvenței cuvintelor* (TF) vom avea frecvența acestora raportată la lungimea documentului respectiv:

- Folosind frecvența raportată la numărul total de cuvinte: $TF\left(t_i, d_j\right) = \begin{cases} 0, dacă\ n_{ij} = 0 \\ \frac{n_{ij}}{\sum_{k=1}^{m} n_{kj}}, dacă\ n_{ij} > 0 \end{cases}$;
- Folosind numărul maxim de apariții al unui termen: $TF\left(t_i, d_j\right) = \begin{cases} 0, dacă\ n_{ij} = 0 \\ \frac{n_{ij}}{max_k n_{kj}}, dacă\ n_{ij} > 0 \end{cases}$;
- Folosind o scară logaritmică, conform [19]: $TF\left(t_i, d_j\right) = \begin{cases} 0, dacă\ n_{ij} = 0 \\ 1 + log\left(1 + \log n_{ij}\right), n_{ij} > 0 \end{cases}$.

În metoda bazată pe *inversul frecvenței documentului* (IDF) vom avea frecvența acestora raportată la numărul de apariții din toate documentele indexate. Fie $D = \bigcup_1^n d_j$ colecția de documente indexate, iar D_{t_i} documentele în care apare termenul t_i, avem $IDF(t_i) = \log \frac{1+|D|}{\left|D_{t_i}\right|}$, iar $d_j^i = TF\left(t_i, d_j\right) IDF(t_i)$.

Dat fiind un vector de căutare q se poate calcula distanța Euclidiană $\left\|q - d_j\right\| = \sqrt{\sum_{i=1}^{m} \left(q^i - d_j^i\right)^2}$.

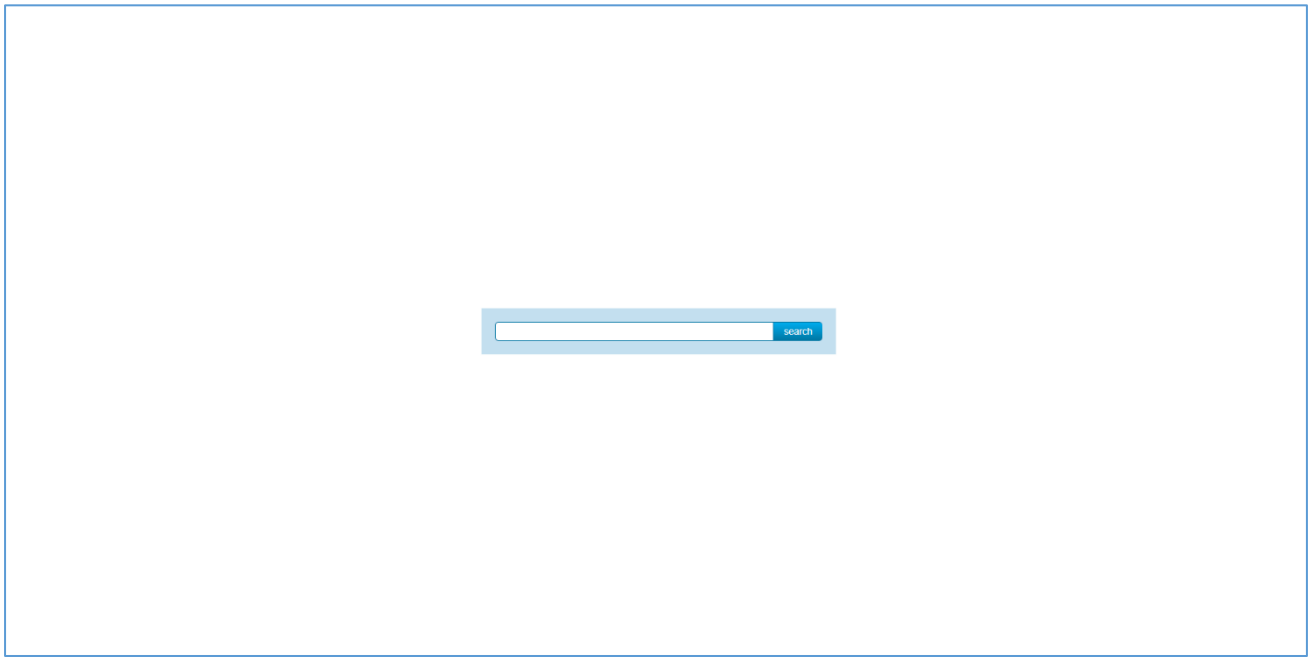
Însă motoarele de căutare Web folosesc similaritatea cosinus între cei doi vectori $q \cdot d_j = \sum_{i=1}^{m} q^i d_j^i$.

4. Proiectarea și implementarea sistemului informatic

4.1. Proiectarea sistemului informatic

Conform cu Wikipedia, un sistem informatic este un sistem care permite introducerea de date prin procedee manuale sau prin culegere automată de către sistem, stocarea acestora, prelucrarea lor și extragerea informației (rezultatelor) sub diverse forme; componentele sistemului informatic sunt: calculatoarele, programele, rețelele de calculatoare și utilizatorii. În cazul de față, vom avea două programe diferite pentru acest sistem informatic: un robot de căutare și o interfață Web pentru acces la rezultatele căutării.

Robotul de căutare este o aplicație implementată în Microsoft Visual C++ 2015, iar interfața Web este realizată în PHP, cu bază de date MySQL. Acest robot are rolul de a indexa în mod automat paginile din Internet și de a stoca (parțial) conținutul acestora într-o bază de date MySQL; acesta este cea mai fragilă componentă din tot sistemul. Interfața Web pentru motorul de căutare va folosi algoritmul descris în secțiunea Indexarea paginilor Web pentru a returna ca rezultat cele mai relevante pagini Web indexate pentru căutarea noastră.



Figură 9. Pagina de pornire pentru https://www.text-mining.ro/

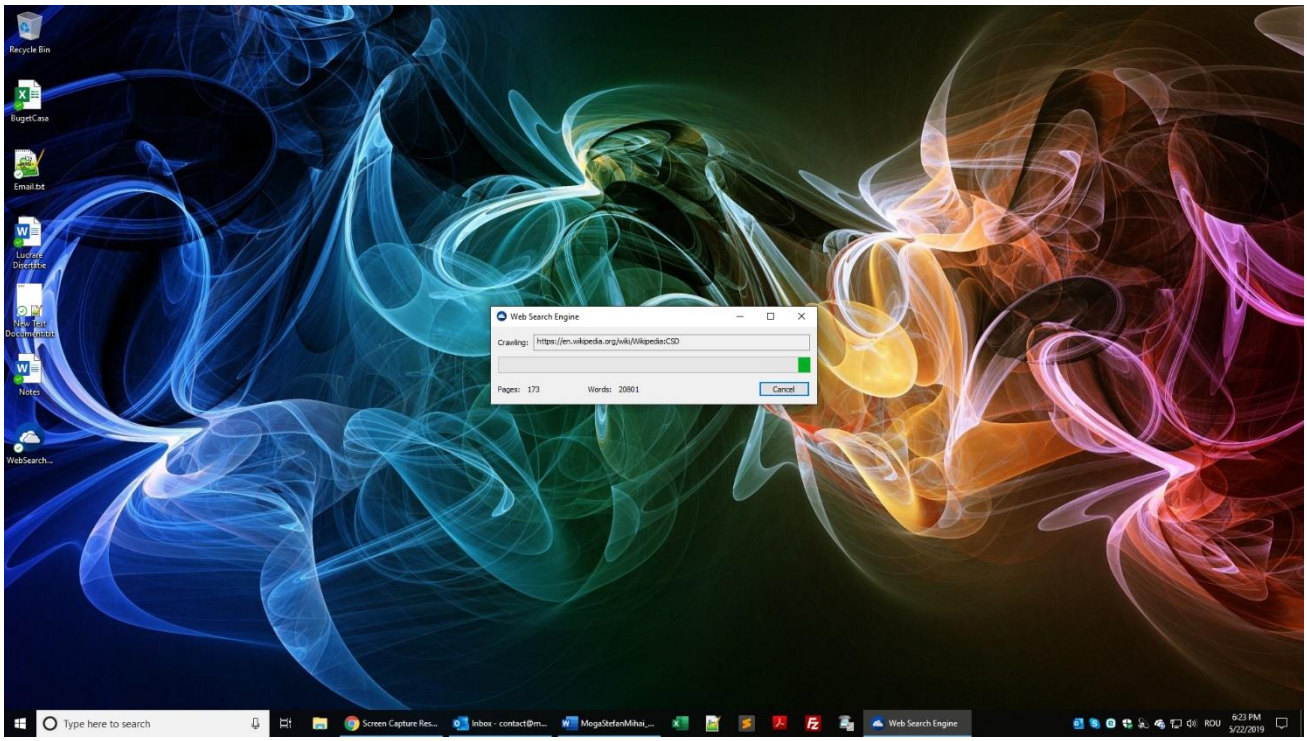
4.2. Proiectarea robotului de căutare Web

Așa cum am menționat anterior, robotul de căutare Web folosește o coadă de priorități pentru gestiunea adreselor URL pe care le va vizita. Algoritmul de adăugare a unei adrese URL în frontieră este: verifică dacă adresa URL a fost deja vizitată; dacă da, stop, altfel verifică dacă ea este deja în coada de priorități; dacă da, incrementează prioritatea acesteia, iar dacă nu, o adaugă în coadă folosind cea mai mică prioritate. Algoritmul de extragere a unei adrese URL din frontieră este chiar mai simplu: caută adresa URL cu prioritatea cea mai mare, iar dacă toate adresele URL au aceeași prioritate folosește principiul *First In First Out*, adică prima adăugată va fi prima adresă URL vizitată.

Pentru fiecare adresă URL extrasă din frontieră se vor aplica următorii pași:

- se descarcă local pagina Web din Internet care urmează să fie indexată;
- se caută titlul acestei pagini HTML, care este memorat într-o variabilă temporară;

- se caută toate adresele URL din această pagină HTML, care vor fi adăugate în frontieră conform cu algoritmul descris mai sus; (în frontieră sunt adăugate doar adrese URL absolute, adică dacă pagina HTML conține adrese relative acestea sunt transformate în adrese complete);
- pe conținutul paginii HTML se aplică mai multe procedee care au ca scop curățarea conținutului paginii Web: se elimină toate tagurile HTML pentru a păstra textul pur, se face conversia din formatul de caractere UTF8 la Unicode pentru analiză unitară, după care se elimină toate caracterele de punctuație cunoscute: `\t\n\r\"\' !?#$%&|(){}[]*/+-:;<>=,.`
- textul pur rezultat din procesarea paginii HTML, împreună cu titlul și adresa URL corespunzătoare este adăugat în tabela `WEBPAGE`; fiecare cuvânt unic este adăugat în tabela `KEYWORD`, iar fiecare legătură dintre pagina HTML și cuvintele conținute de aceasta în tabela `OCCURRENCE`, care are rolul de a cuantifica numărul de apariții al unui cuvânt într-o pagină.

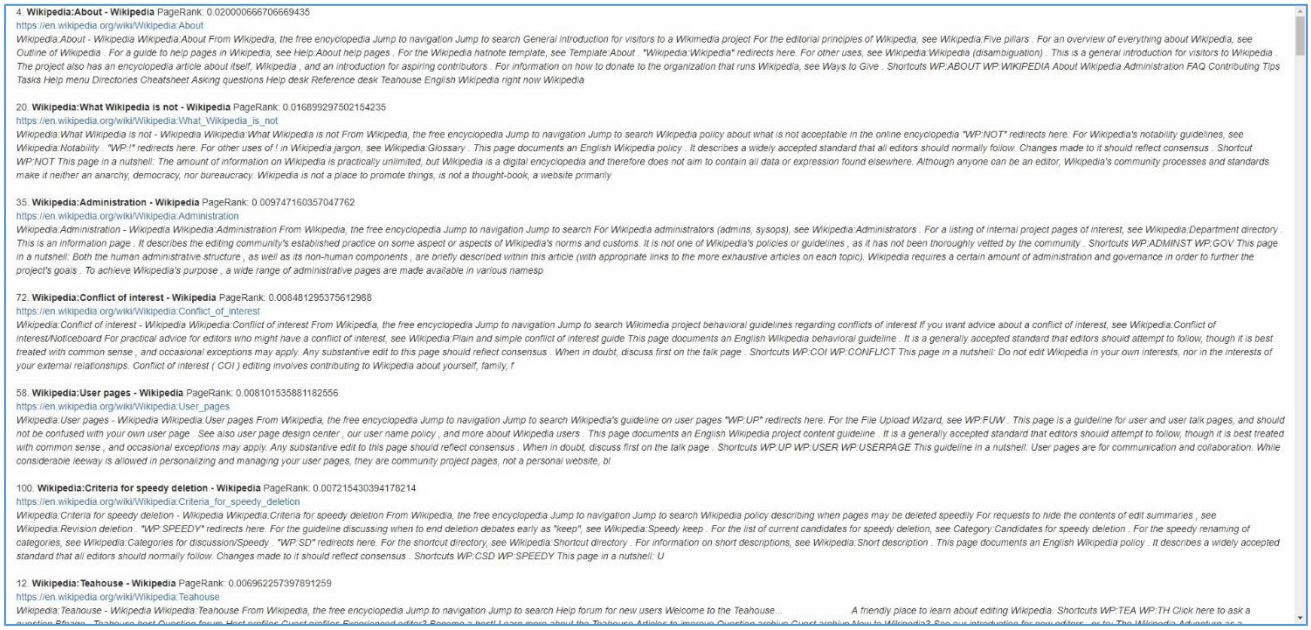


Figură 10. Captură cu robotul de căutare Web în acțiune

4.3. Proiectarea interfeței Web de căutare

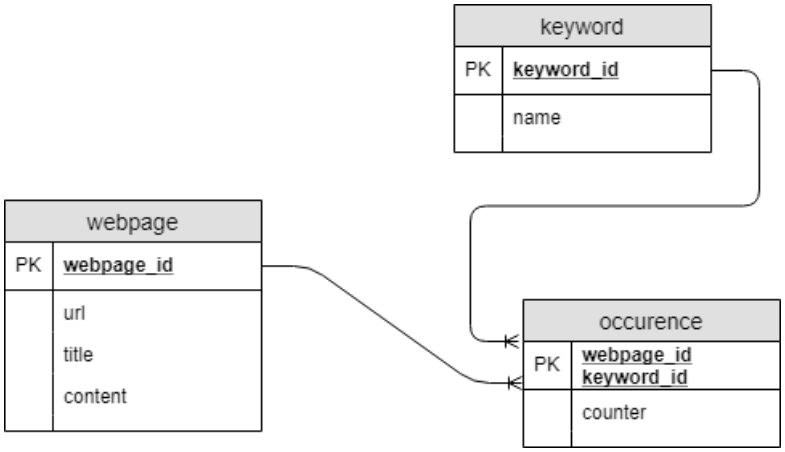
Ca și robotul de căutare Web, interfața Web este proiectată să descompună în cuvinte unice folosind caracterele de punctuație cunoscute: `\t\n\r\"\' !?#$%&|(){}[]*/+-:;<>=,.` Aceste cuvinte vor fi folosite pentru a selecta din baza de date MySQL doar pe acele pagini Web indexate care conțin absolut toate cuvintele unice folosite la căutare. Tot pe baza acestor cuvinte unice se compune un scor pentru fiecare pagină Web indexată, folosind un algoritm de Data Mining, descris în secțiunea Indexarea paginilor Web. Conform acestuia, scorul fiecărei pagini Web va fi dat de formula $PageRank = TF_{cuvânt} * IDF_{cuvânt}$, unde $TF_{cuvânt}$ (en: *term frequency*) reprezintă frecvența termenului raportată la numărul maxim de apariții, iar $IDF_{cuvânt}$ (en: *inverse document frequency*) reprezintă frecvența documentelor raportată la numărul de apariții din toate documentele indexate. Pentru a calcula $TF(t_i, d_j) = \begin{cases} 0, dacă\ n_{ij} = 0 \\ \frac{n_{ij}}{max_k n_{kj}}, dacă\ n_{ij} > 0 \end{cases}$ vom folosi funcția MySQL `no_of_words(token VARCHAR(256))`. Pentru a calcula $IDF(t_i) = \log \frac{1+|D|}{|D_{t_i}|}$, unde $D = \bigcup_1^n d_j$ este colecția de documente indexate, iar D_{t_i} sunt documentele în care apare termenul t_i, vom folosi funcțiile MySQL `no_of_pages(token`

VARCHAR(256)) și total_pages(). Rezultatul final este dat de funcția data_mining(webpage_no BIGINT, token VARCHAR(256)).



Figură 11. Rezultatele căutării pentru cuvântul Wikipedia

4.4. Schema bazei de date MySQL



Figură 12. Schema bazei de date MySQL pentru motorul de căutare Web

4.5. Implementarea sistemului informatic

Implementarea bazei de date MySQL constă din următorul script SQL:

```
DROP TABLE IF EXISTS `occurrence`;
DROP TABLE IF EXISTS `keyword`;
DROP TABLE IF EXISTS `webpage`;

CREATE TABLE `webpage` (`webpage_id` BIGINT NOT NULL AUTO_INCREMENT, `url` VARCHAR(256)
NOT NULL, `title` VARCHAR(256) NOT NULL, `content` TEXT NOT NULL, PRIMARY
KEY(`webpage_id`)) ENGINE=InnoDB;
CREATE TABLE `keyword` (`keyword_id` BIGINT NOT NULL AUTO_INCREMENT, `name` VARCHAR(256)
NOT NULL, PRIMARY KEY(`keyword_id`)) ENGINE=InnoDB;
CREATE TABLE `occurrence` (`webpage_id` BIGINT NOT NULL, `keyword_id` BIGINT NOT NULL,
`counter` BIGINT NOT NULL, `pagerank` REAL NOT NULL, PRIMARY KEY(`webpage_id`,
```

```
`keyword_id`), FOREIGN KEY webpage_fk(webpage_id) REFERENCES webpage(webpage_id),
FOREIGN KEY keyword_fk(keyword_id) REFERENCES keyword(keyword_id)) ENGINE=InnoDB;

CREATE UNIQUE INDEX index_name ON `keyword`(`name`);

DELIMITER //
CREATE OR REPLACE FUNCTION no_of_words(token VARCHAR(256)) RETURNS REAL READS SQL DATA
BEGIN
	DECLARE retVal REAL;
	SELECT MAX(`counter`) INTO retVal FROM `occurrence` INNER JOIN `keyword`
USING(`keyword_id`) WHERE `name` = token;
	RETURN retVal;
END//
DELIMITER ;

DELIMITER //
CREATE OR REPLACE FUNCTION no_of_pages(token VARCHAR(256)) RETURNS REAL READS SQL DATA
BEGIN
	DECLARE retVal REAL;
	SELECT COUNT(`webpage_id`) INTO retVal FROM `occurrence` INNER JOIN `keyword`
USING(`keyword_id`) WHERE `name` = token;
	RETURN retVal;
END//
DELIMITER ;

DELIMITER //
CREATE OR REPLACE FUNCTION total_pages() RETURNS REAL READS SQL DATA
BEGIN
	DECLARE retVal REAL;
	SELECT COUNT(`webpage_id`) INTO retVal FROM `webpage`;
	RETURN retVal;
END//
DELIMITER ;

DELIMITER //
CREATE OR REPLACE FUNCTION data_mining(webpage_no BIGINT, token VARCHAR(256)) RETURNS
REAL READS SQL DATA
BEGIN
	DECLARE retVal REAL;
	SELECT SUM(`counter`)/no_of_words(token)*LOG((1+total_pages())/no_of_pages(token))
INTO retVal FROM `occurrence` INNER JOIN `keyword` USING(`keyword_id`) WHERE `name` =
token AND `webpage_id` = webpage_no;
	RETURN retVal;
END//
DELIMITER ;
```

Pentru inserarea unei pagini Web în baza de date MySQL se va folosi scriptul:

```
INSERT INTO `webpage` (`url`, `title`, `content`) VALUES (?, ?, ?);
```

Pentru inserarea unui cuvânt în baza de date MySQL se va folosi scriptul:

```
INSERT INTO `keyword` (`name`) VALUES (?);
```

Pentru inserarea unei legături între pagina Web și un cuvânt din baza de date MySQL se va folosi scriptul:

```
INSERT INTO `occurrence` (`webpage_id`, `keyword_id`, `counter`, `pagerank`) VALUES (?,
?, ?, ?);
```

Pentru actualizarea unei legături între pagina Web și un cuvânt din baza de date MySQL se va folosi scriptul:

```
UPDATE `occurrence` SET `counter` = `counter` + 1 WHERE `webpage_id` = ? AND
`keyword_id` = ?;
```

Pentru actualizarea scorului tuturor cuvintelor indexate în baza de date MySQL se va folosi scriptul:

```
UPDATE `occurrence` INNER JOIN `keyword` USING(`keyword_id`) SET `pagerank` =
data_mining(`webpage_id`, `name`);
```

4.6. Testarea sistemului informatic

Aplicațiile Web s-au dezvoltat într-o platformă de comunicare de bază pentru multe companii. Aplicațiile Web sunt cruciale pentru comerț, pentru schimbul de informații și pentru găzduirea activităților sociale. Din acest motiv, aplicațiile Web trebuie să ofere o înaltă performanță, fiabilitate și ușurință în folosire. Furnizarea unor aplicații Web de calitate pentru utilizatorii existenți și cei viitori reprezintă o provocare majoră pentru asigurarea calității. Testarea este una dintre cele mai importante măsuri de asigurare a calității. Metodele și tehnicile de testare tradiționale se concentrează în principal pe testarea cerințelor funcționale. Din păcate, acestea nu se concentrează suficient pe cerințele de calitate, importante pentru utilizatorii de aplicații Web, cum ar fi **performanța, ușurința în folosire, fiabilitatea** și **securitatea**. O provocare majoră în testarea aplicațiilor Web este dominanta schimbării. Cerințele utilizatorilor și așteptările, platformele și configurațiile, modelele de afaceri, dezvoltarea și testarea bugetelor sunt subiecte supuse unor modificări frecvente pe tot parcursul ciclului de viață al aplicațiilor Web. Prin urmare, este necesară dezvoltarea unui sistem eficient de testare care să acopere o gamă largă de caracteristici de calitate ale aplicațiilor Web, care să facă față la schimbări și care să ajute la implementarea și buna înțelegere a unei testări sistematice, complete și lipsite de riscuri. O astfel de schemă de test formează baza pentru construirea unei metode model și a instrumentelor aferente. Experiența practică a demonstrat că testarea metodică și sistematică, fundamentată pe o astfel de schemă, este realizabilă și utilă pe tot parcursul dezvoltării și evoluției aplicațiilor Web.

Aplicațiile Web sunt expuse unor noi provocări privind asigurarea calității și testarea. Aplicațiile Web sunt compuse din diverse componente software oferite în anumite cazuri de anumiți producători. Calitatea aplicațiilor Web este în principal determinată de calitatea fiecărei componente software implicate și de calitatea legăturilor dintre acestea. Testarea este una din cele mai importante instrumente folosite în dezvoltarea aplicațiilor Web pentru realizarea produselor de înalta calitate, care îndeplinesc așteptările utilizatorilor.

Testarea aplicațiilor Web merge dincolo de testarea software-ului din sistemele tradiționale. Deși se aplică cerințe similare la corectitudinea tehnică a unei aplicații, utilizarea unei aplicații Web de către grupuri eterogene de utilizatori, pe un număr mare de platforme, duce la cerințe speciale de testare. Deseori este greu de anticipat numărul viitor de utilizatori pentru o aplicație Web. Timpul de răspuns este unul din factorii de succes decisivi pe Internet și trebuie sa fie avut în vedere din timp, chiar dacă platforma hardware finală este, în general, disponibilă mult mai târziu. Alți factori importanți pentru succesul aplicațiilor Web sunt ușurința în folosire, disponibilitatea, compatibilitatea browserelor, securitatea, actualitatea și eficiența.

Abordările **Agile** (cum ar fi *Extreme Programming*) sunt din ce în ce mai folosite în proiectele Web. În timp ce abordările Agile se concentrează pe colaborare, abordările tradiționale se focalizează pe planificarea și managementul proiectului. În funcție de caracteristicile proiectului Web, poate fi necesară realizarea activităților de testare folosind atât abordările Agile cât și cele tradiționale pe parcursul proiectului.

4.7. Planul de audit pentru sistemul informatic

În urma dezvoltării motorului de căutare Web se va răspunde la următorul chestionar de audit:

1) Mediul IT din cadrul organizației
   a) Clientul folosește un sistem informatic integrat?
   b) Cât de critică este disponibilitatea sistemelor IT pentru afacerea clientului? (foarte critic – întrerupere tolerabilă < 1 zi, critic – întrerupere tolerabilă 1-3 zile, necritic – întrerupere tolerabilă > 3 zile)
   c) Ce părți din mediul IT sunt externalizate?
   d) Este IT-ul critic pentru atingerea obiectivelor clientului?
   e) Cum este formalizată relația dintre client și furnizorul de servicii externe?
   f) Cum se asigură clientul că IT-ul este parte a strategiei pe termen mediu și lung?
   g) Este IT-ul critic pentru atingerea obiectivelor clientului? Ce a întreprins clientul pentru a asigura securitatea datelor sale? (politici/proceduri)
   h) Cum se asigură clientul că proiectele pe care dorește să le inițieze sunt planificate corespunzător?
   i) Cum sunt monitorizate proiectele pentru a se asigura că își vor atinge obiectivele într-un mod eficient din punct de vedere al timpului și costului?
   j) Are clientul dezvoltat DRP (*disaster recovery plan*) / BCP (*business continuity plan*)?
   k) Planul (DRP/BCP) acoperă toate aplicațiile și funcțiile de infrastructură care suportă procesele? Care continuitate este critică pentru client? Cât de des și cât de riguros este testat planul?
   l) Este clientul conștient de date care îi sunt critice?
2) Tehnologia utilizată; diagrama de rețea
3) Organigrama departamentului IT
   a) La ce nivel din cadrul organizației raportează șeful departamentului de IT?
   b) Cum este IT-ul organizat astfel încât să asigure o delimitarea a responsabilităților și continuitatea activității? (cum ar fi pe perioada concediilor)
   c) Care sunt riscurile identificate de dumneavoastră din informațiile prezentate în secțiunea 2?
4) Analiza aplicației selectate
   a) Prezentați principalele setări de securitate ale sistemului și analizați completitudinea lor?
   b) Cum este monitorizat accesul în aplicație? (revizuire de loguri, revizuire de listă de utilizatori)
   c) Accesul utilizatorilor este autorizat și creat corespunzător? (cine face cererea, cine stabilește drepturile, cine aprobă accesul, cine creează contul, cine notifică plecarea angajatului din organizație)
   d) Cine și cum inițiază o modificare care să fie adusă aplicației?
   e) Cine aprobă modificarea pentru a fi dezvoltată?
   f) Cine și cum monitorizează modificările aduse aplicației?
   g) Cine și cum testează modificările dezvoltate? Cine aprobă migrarea dezvoltării în producție?
   h) Cum este asigurată delimitarea responsabilităților în cadrul procesului de gestionare a modificărilor aduse aplicației?
   i) Cum se realizează backup-ul informațiilor din aplicație? Cât de des este verificat backup-ul și cum?
   j) Cum sunt monitorizate și rezolvate deviațiile care apar în procesările programate? (transferuri, scheduled task)
   k) Care sunt riscurile identificate de dumneavoastră din informațiile prezentate în secțiunea 3?

5. Concluzie

Metoda de indexare a documentelor Web prezentată în această lucrare științifică este cu siguranță o soluție abordată de majoritatea motoarelor Web de căutare, însă înalta performanță impune ca, pe viitor, aceste metode computaționale să fie sprijinite de informații de profil al utilizatorului precum și de analiza semantică a textelor. Există opinii care susțin că o îmbunătățire radicală a căutării informațiilor relevante pe Web nu ar fi posibilă decât prin implementarea conceptului de *"Web semantic"*, după cum a fost numit de Tim Berners-Lee încă din 1999. Acesta presupune ca informațiile de pe Web să fie caracterizate de meta-informații, utile sistemelor de căutare. Aceste argumente mă determină să consider că acest subiect este o tematică realmente actuală, de interes major, fiind deosebit de oportună în contextul preocupărilor în știința și ingineria calculatoarelor pe întreg globul.

6. Bibliografie

1. Radu G. Crețulescu, *Text mining. Tehnici de clasificare și clustering al documentelor*, Ed. Albastră, 2012;
2. Smaranda Belciug, Marina Gorunescu, *Data mining. Modele predictive și de clasificare*, Ed. Albastră, 2012;
3. Florin Gorunescu, *Data Mining. Concepte, Modele și Tehnici*, Ed. Albastră, 2006;
4. Bing Liu, *Web Data. Exploring Hyperlinks, Contents, and Usage Data*, 2nd Edition, Springer, 2006;
5. https://en.wikipedia.org/wiki/Web_search_engine
6. Sergey Brin, Lawrence Page. *The anatomy of a large-scale hypertextual web search engine. Computer Networks*, 1998, pag. 107-117;
7. https://en.wikipedia.org/wiki/Document_Object_Model
8. Arvind Arasu, Junghoo Cho, Hector Garcia-Molina, Andreas Paepcke, Sriram Raghavan. *Searching the web. ACM Transactions on Internet Technology*, 2001, pag. 2-43;
9. Junghoo Cho, Hector Garcia-Molina, Lawrence Page. *Efficient crawling through URL ordering. Computer Networks*, 1998, pag. 161-172;
10. Monika R. Henzinger, Allan Heydon, Michael Mitzenmacher, Marc Najork. *Measuring index quality using random walks on the Web. Computer Networks*, 1999, pag. 1291-1303;
11. Alexandros Ntoulas, Junghoo Cho, Christopher Olston. *What's New on the Web? The Evolution of the Web from a Search Engine Perspective. In Proceedings of International Conference on World Wide Web*, 2004;
12. Dennis Fetterly, Mark Manasse, Marc Najork, Janet Wiener. *A large scale study of the evolution of Web pages. Software: Practice and Experience*, 2004, pag. 213-237;
13. Zdravko Markov, Daniel T. Larose. *Data Mining the Web: Uncovering Patterns in Web Content, Structure, and Usage*, Central Connecticut State University, 2006;
14. https://www.w3.org/DOM/
15. Soumen Chakrabarti, Martinvanden Berg, Byron Dom. *Focused crawling: a new approach to topic-specific Web resource discovery. Computer Networks*, 1999, pag. 1623-1640;
16. Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, Marco Gori. *Focused crawling using context graphs*. In *Proceedings of International Conference on Very Large Data Bases*, 2000;
17. Gautam Pant, Padmini Srinivasan. *Learning to crawl: Comparing classification schemes. ACM Transactions on Information Systems*, 2005, pag. 430-462;
18. Lawrence Page, Sergey Brin, Rajeev Motwani, Terry Winograd, *The PageRank Citation Ranking: Bringing Order to the Web*, Stanford University, 1998;
19. Chris Buckley, *Implementation of the SMART information retrieval system*, Technical Report 85-686, Cornell University, Ithaca, NY, 1985;
20. Andrew Oppel, *SQL Demystified*, McGraw-Hill/Osborne, 2005;
21. Andy Oppel, Robert Sheldon, *SQL – A Beginner's Guide*, McGraw-Hill, 3rd Edition, 2009;
22. David Sklar, Adam Trachtenberg, *PHP Cookbook*, 3rd Edition, O'Reilly Media, 2014;
23. Robin Nixon, *Learning PHP, MySQL & JavaScript with jQuery, CSS & HTML5*, 4th Edition, O'Reilly Media, 2015.

7. Anexă. Codul HTML pentru interfața Web

```
<!DOCTYPE html>
<html>
<head>
    <title>text-mining.ro</title>
    <meta name="ROBOTS" content="NOINDEX, NOFOLLOW" />
    <link rel="icon" type="image/png" href="https://www.mihaimoga.com/images/romania-
flag-square-icon-256.png">
    <!-- CSS styles for standard search box -->
    <style type="text/css">
        html, body, .container {
            height: 100%;
        }

        .container {
            display: -webkit-flexbox;
            display: -ms-flexbox;
            display: -webkit-flex;
            display: flex;
            -webkit-flex-align: center;
            -ms-flex-align: center;
            -webkit-align-items: center;
            align-items: center;
            justify-content: center;
        }

        #tfheader {
            background-color: #c3dfef;
        }

        #tfnewsearch {
            float: right;
            padding: 20px;
        }

        .tftextinput {
            margin: 0;
            padding: 5px 15px;
            font-family: Arial, Helvetica, sans-serif;
            font-size: 14px;
            border: 1px solid #0076a3;
            border-right: 0px;
            border-top-left-radius: 5px 5px;
            border-bottom-left-radius: 5px 5px;
        }

        .tfbutton {
            margin: 0;
            padding: 5px 15px;
            font-family: Arial, Helvetica, sans-serif;
            font-size: 14px;
            outline: none;
            cursor: pointer;
            text-align: center;
            text-decoration: none;
            color: #ffffff;
            border: solid 1px #0076a3;
```

```
            border-right: 0px;
            background: #0095cd;
            background: -webkit-gradient(linear, left top, left bottom, from(#00adee),
to(#0078a5));
            background: -moz-linear-gradient(top, #00adee, #0078a5);
            border-top-right-radius: 5px 5px;
            border-bottom-right-radius: 5px 5px;
        }

             .tfbutton:hover {
                   text-decoration: none;
                   background: #007ead;
                   background: -webkit-gradient(linear, left top, left bottom,
from(#0095cc), to(#00678e));
                   background: -moz-linear-gradient(top, #0095cc, #00678e);
             }

             /* Fixes submit button height problem in Firefox */
             .tfbutton::-moz-focus-inner {
                   border: 0;
             }

        .tfclear {
            clear: both;
        }
    </style>
</head>
<body>
    <!-- HTML for SEARCH BAR -->
    <div class="container">
        <div id="tfheader">
            <form id="tfnewsearch" method="get" action="search.php">
                <input type="text" class="tftextinput" name="q" size="50"
maxlength="120">
                <input type="submit" value="search" class="tfbutton">
            </form>
            <div class="tfclear"></div>
        </div>
    </div>
</body>
</html>
```

8. Anexă. Codul PHP pentru interogarea bazei de date MySQL

```
<?php
/* This file is part of Web Search Engine application developed by Mihai MOGA.

Web Search Engine is free software: you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by the Open
Source Initiative, either version 3 of the License, or any later version.

Web Search Engine is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
details.

You should have received a copy of the GNU General Public License along with
Web Search Engine.  If not, see <http://www.opensource.org/licenses/gpl-3.0.html>*/
```

```
echo "<!DOCTYPE html>\n";
echo "<html>\n";
echo "\t<head>\n";
echo "\t\t<title>" . $_GET['q'] . "</title>\n";
echo "\t\t<meta charset=\"utf-8\">\n";
echo "\t\t<link rel=\"icon\" type=\"image/png\"
href=\"https://www.mihaimoga.com/images/romania-flag-square-icon-256.png\">\n";
echo "\t\t<meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">\n";
echo "\t\t<link rel=\"stylesheet\"
href=\"https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css\">\n";
echo "\t\t<script
src=\"https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js\"></script>\n";
echo "\t\t<script
src=\"https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js\"></script>\n"
;
echo "\t</head>\n";
echo "\t<body>\n";
$search = strtolower($_GET['q']);
$counter = 0;
$mysql_clause = "";
$mysql_select = "";
$token = strtok($search, "\t\n\r\"\' !?#$%&|(){}[]*/+-:;<>=.,");
while ($token !== false) {
	if ($counter == 0) {
		$mysql_clause = "SELECT DISTINCT `webpage_id` FROM `occurrence` INNER JOIN
`keyword` USING (`keyword_id`) WHERE `name` = '$token'";
		$mysql_select = "(`name` = '$token')";
	}
	else {
		$mysql_clause = "SELECT DISTINCT `webpage_id` FROM `occurrence` INNER JOIN
`keyword` USING (`keyword_id`) WHERE `name` = '$token' AND `webpage_id` IN (" .
$mysql_clause . ")";
		$mysql_select = $mysql_select . " OR (`name` = '$token')";
	}
	$counter++;
	$token = strtok("\t\n\r\"\' !?#$%&|(){}[]*/+-:;<>=.,");
};
if ($counter > 0)
{
	// Create connection
	$conn = mysqli_connect($servername, $username, $password, $dbname);
	// Check connection
	if (!$conn) {
	    die("Connection failed: " . mysqli_connect_error());
	}

	$statement = "SELECT DISTINCT `webpage_id`, `title`, `url`, `content`,
AVG(`pagerank`) AS score FROM `occurrence` INNER JOIN `webpage` USING(`webpage_id`)
INNER JOIN `keyword` USING(`keyword_id`) WHERE  `webpage_id` IN (" . $mysql_clause . ")
AND (" . $mysql_select . ") GROUP BY `webpage_id` ORDER BY score DESC LIMIT 100;";
	$result = mysqli_query($conn, $statement);
	if (mysqli_num_rows($result) > 0) {
	    // output data of each row
	    while($row = mysqli_fetch_assoc($result)) {
	        echo "<div class=\"container-fluid\">" . $row["webpage_id"] . ". <b>" .
$row["title"] . "</b> PageRank: " . $row["score"] . "<br />";
	        echo "<a href=\"" . $row["url"] . "\">" . $row["url"] . "</a><br />";
```

```
            echo "<i>" . utf8_encode(substr($row["content"], 0, 1024)) .
"</i></div><br />\n";
        }
    } else {
        echo "0 results";
    }
    mysqli_close($conn);
}
echo "\t</body>\n";
echo "</html>\n";
?>
```

9. Anexă: codul sursă pentru robotul de căutare Web

9.1. Fișierul HtmlToText.cpp

```
/* This file is part of Web Search Engine application developed by Mihai MOGA.

Web Search Engine is free software: you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by the Open
Source Initiative, either version 3 of the License, or any later version.

Web Search Engine is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with
Web Search Engine.  If not, see <http://www.opensource.org/licenses/gpl-3.0.html>*/

#include "stdafx.h"
#include "HtmlToText.h"
#include <algorithm>

CHtmlToText::CHtmlToText()
{
    _tags.SetAt(_T("address"), _T("\n"));
    _tags.SetAt(_T("blockquote"), _T("\n"));
    _tags.SetAt(_T("div"), _T("\n"));
    _tags.SetAt(_T("dl"), _T("\n"));
    _tags.SetAt(_T("fieldset"), _T("\n"));
    _tags.SetAt(_T("form"), _T("\n"));
    _tags.SetAt(_T("h1"), _T("\n"));
    _tags.SetAt(_T("/h1"), _T("\n"));
    _tags.SetAt(_T("h2"), _T("\n"));
    _tags.SetAt(_T("/h2"), _T("\n"));
    _tags.SetAt(_T("h3"), _T("\n"));
    _tags.SetAt(_T("/h3"), _T("\n"));
    _tags.SetAt(_T("h4"), _T("\n"));
    _tags.SetAt(_T("/h4"), _T("\n"));
    _tags.SetAt(_T("h5"), _T("\n"));
    _tags.SetAt(_T("/h5"), _T("\n"));
    _tags.SetAt(_T("h6"), _T("\n"));
    _tags.SetAt(_T("/h6"), _T("\n"));
    _tags.SetAt(_T("p"), _T("\n"));
    _tags.SetAt(_T("/p"), _T("\n"));
    _tags.SetAt(_T("table"), _T("\n"));
    _tags.SetAt(_T("/table"), _T("\n"));
    _tags.SetAt(_T("ul"), _T("\n"));
```

```
	_tags.SetAt(_T("/ul"), _T("\n"));
	_tags.SetAt(_T("ol"), _T("\n"));
	_tags.SetAt(_T("/ol"), _T("\n"));
	_tags.SetAt(_T("/li"), _T("\n"));
	_tags.SetAt(_T("br"), _T("\n"));
	_tags.SetAt(_T("/td"), _T("\t"));
	_tags.SetAt(_T("/tr"), _T("\n"));
	_tags.SetAt(_T("/pre"), _T("\n"));

	_ignoreTags.AddTail(_T("script"));
	_ignoreTags.AddTail(_T("noscript"));
	_ignoreTags.AddTail(_T("style"));
	_ignoreTags.AddTail(_T("object"));
}

CHtmlToText::~CHtmlToText()
{
}

const std::string& CHtmlToText::Convert(const std::string& html)
{
	// Initialize state variables
	bool selfClosing = false;
	_html = html;
	_pos = 0;

	// Process input
	while (!EndOfText())
	{
		if (Peek() == '<')
		{
			// HTML tag
			std::string tag = ParseTag(selfClosing);

			// Handle special tag cases
			if (tag.compare("body") == 0)
			{
				// Discard content before <body>
				_text.empty();
			}
			else if (tag.compare("/body") == 0)
			{
				// Discard content after </body>
				_pos = _html.length();
			}
			else if (tag.compare("pre") == 0)
			{
				// Enter preformatted mode
				_preformatted = true;
				EatWhitespaceToNextLine();
			}
			else if (tag.compare("/pre") == 0)
			{
				// Exit preformatted mode
				_preformatted = false;
			}

			_text.append(" ");
```

```
                if (_ignoreTags.Find(CString(tag.c_str())) != NULL)
                        EatInnerContent(tag);
        }
        else if (IsWhiteSpace(Peek()))
        {
                // Whitespace (treat all as space)
                _text += (_preformatted ? Peek() : ' ');
                MoveAhead();
        }
        else
        {
                // Other text
                _text += Peek();
                MoveAhead();
        }
    }

    return _text;
}

std::string CHtmlToText::ParseTag(bool& selfClosing)
{
    std::string tag;
    selfClosing = false;

    // Eat comments
    if (((_pos + 4) < _html.length()) &&
        (_html[_pos] == '<') &&
        (_html[_pos + 1] == '!') &&
        (_html[_pos + 2] == '-') &&
        (_html[_pos + 3] == '-'))
    {
        MoveAhead();
        MoveAhead();
        MoveAhead();
        MoveAhead();

        while (!EndOfText())
        {
                if (((_pos + 3) < _html.length()) &&
                        (_html[_pos] == '-') &&
                        (_html[_pos + 1] == '-') &&
                        (_html[_pos + 2] == '>'))
                        break;

                MoveAhead();
        }

        MoveAhead();
        MoveAhead();
        MoveAhead();
        EatWhitespace();
    }

    // Eat scripts
    if (((_pos + 7) < _html.length()) &&
        (_html[_pos] == '<') &&
```

```
        (_html[_pos + 1] == 's') &&
        (_html[_pos + 2] == 'c') &&
        (_html[_pos + 3] == 'r') &&
        (_html[_pos + 4] == 'i') &&
        (_html[_pos + 5] == 'p') &&
        (_html[_pos + 6] == 't'))
{
        MoveAhead();
        MoveAhead();
        MoveAhead();
        MoveAhead();
        MoveAhead();
        MoveAhead();
        MoveAhead();


        while (!EndOfText())
        {
                if (((_pos + 7) < _html.length()) &&
                        (_html[_pos] == '/') &&
                        (_html[_pos + 1] == 's') &&
                        (_html[_pos + 2] == 'c') &&
                        (_html[_pos + 3] == 'r') &&
                        (_html[_pos + 4] == 'i') &&
                        (_html[_pos + 5] == 'p') &&
                        (_html[_pos + 6] == 't'))
                        break;

                MoveAhead();
        }

        MoveAhead();
        MoveAhead();
        MoveAhead();
        MoveAhead();
        MoveAhead();
        MoveAhead();
        MoveAhead();
        MoveAhead();
        EatWhitespace();
}

if (Peek() == _T('<'))
{
        MoveAhead();

        // Parse tag name
        EatWhitespace();
        size_t start = _pos;
        if (Peek() == '/')
                MoveAhead();
        while (!EndOfText() && !IsWhiteSpace(Peek()) &&
                (Peek() != '/') && (Peek() != '>'))
                MoveAhead();
        tag = _html.substr(start, _pos - start);
        std::transform(tag.begin(), tag.end(), tag.begin(), ::tolower);

        // Parse rest of tag
```

```
		while (!EndOfText() && (Peek() != '>'))
		{
			if ((Peek() == '\"') || (Peek() == '\''))
				EatQuotedValue();
			else
			{
				if (Peek() == '/')
					selfClosing = true;
				MoveAhead();
			}
		}

		MoveAhead();
	}
	return tag;
}

void CHtmlToText::EatInnerContent(const std::string& tag)
{
	bool selfClosing = false;
	const std::string endTag = "/" + tag;

	while (!EndOfText())
	{
		if (Peek() == '<')
		{
			// Consume a tag
			if (ParseTag(selfClosing).compare(endTag) == 0)
				return;
			// Use recursion to consume nested tags
			if (!selfClosing && (tag[0] != '/'))
				EatInnerContent(tag);
		}
		else
			MoveAhead();
	}
}
```

9.2. Fișierul HtmlToText.h

```
/* This file is part of Web Search Engine application developed by Mihai MOGA.

Web Search Engine is free software: you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by the Open
Source Initiative, either version 3 of the License, or any later version.

Web Search Engine is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
details.

You should have received a copy of the GNU General Public License along with
Web Search Engine.  If not, see <http://www.opensource.org/licenses/gpl-3.0.html>*/

#pragma once

class CHtmlToText
{
```

```
public:
	CHtmlToText();
	~CHtmlToText();

public:
	const std::string& Convert(const std::string& html);
	std::string ParseTag(bool& selfClosing);
	void EatInnerContent(const std::string& tag);

	bool EndOfText() { return (_pos >= _html.length()); };

	char Peek() { return (_pos < _html.length()) ? _html[_pos] : (char)0; }

	void MoveAhead() { _pos = ((_pos + 1 < _html.length()) ? (_pos + 1) :
_html.length()); }

	bool IsWhiteSpace(char ch)
	{
		if ((ch == _T(' ')) || (ch == _T('\t')) || (ch == _T('\r')) || (ch ==
_T('\n')))
			return true;
		return false;
	}

	void EatWhitespace()
	{
		while (IsWhiteSpace(Peek()))
			MoveAhead();
	}

	void EatWhitespaceToNextLine()
	{
		while (IsWhiteSpace(Peek()))
		{
			char ch = Peek();
			MoveAhead();
			if (ch == _T('\n'))
				break;
		}
	}

	void EatQuotedValue()
	{
		char mark = Peek();
		if ((mark == _T('\"')) || (mark == _T('\'')))
		{
			// Opening quote
			MoveAhead();
			// Find end of value
			while (!EndOfText())
			{
				char ch = Peek();
				MoveAhead();
				if ((ch == mark) || (ch == _T('\r')) || (ch == _T('\n')))
					break;
			}
		}
	}
```

```
protected:
	std::string _text;
	std::string _html;
	size_t _pos;
	bool _preformatted;

	CMapStringToString _tags;
	CStringList _ignoreTags;
};
```

9.3. Fișierul UnquoteHTML.cpp

```
/* This file is part of Web Search Engine application developed by Mihai MOGA.

Web Search Engine is free software: you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by the Open
Source Initiative, either version 3 of the License, or any later version.

Web Search Engine is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
details.

You should have received a copy of the GNU General Public License along with
Web Search Engine.  If not, see <http://www.opensource.org/licenses/gpl-3.0.html>*/

#include "stdafx.h"

// https://brajeshwar.github.io/entities/

typedef struct
{
	const char * Name;
	unsigned int Value;
} EntityNameEntry;

static const EntityNameEntry StaticEntityNames[] =
/* list of entity names defined in HTML 4.0 spec */
{
	{ "nbsp", /*160*/32 },
	{ "iexcl", 161 },
	{ "cent", 162 },
	{ "pound", 163 },
	{ "curren", 164 },
	{ "yen", 165 },
	{ "brvbar", 166 },
	{ "sect", 167 },
	{ "uml", 168 },
	{ "copy", 169 },
	{ "ordf", 170 },
	{ "laquo", 171 },
	{ "not", 172 },
	{ "shy", 173 },
	{ "reg", 174 },
	{ "macr", 175 },
	{ "deg", 176 },
	{ "plusmn", 177 },
```

```
{ "sup2", 178 },
{ "sup3", 179 },
{ "acute", 180 },
{ "micro", 181 },
{ "para", 182 },
{ "middot", 183 },
{ "cedil", 184 },
{ "sup1", 185 },
{ "ordm", 186 },
{ "raquo", 187 },
{ "frac14", 188 },
{ "frac12", 189 },
{ "frac34", 190 },
{ "iquest", 191 },
{ "Agrave", 192 },
{ "Aacute", 193 },
{ "Acirc", 194 },
{ "Atilde", 195 },
{ "Auml", 196 },
{ "Aring", 197 },
{ "AElig", 198 },
{ "Ccedil", 199 },
{ "Egrave", 200 },
{ "Eacute", 201 },
{ "Ecirc", 202 },
{ "Euml", 203 },
{ "Igrave", 204 },
{ "Iacute", 205 },
{ "Icirc", 206 },
{ "Iuml", 207 },
{ "ETH", 208 },
{ "Ntilde", 209 },
{ "Ograve", 210 },
{ "Oacute", 211 },
{ "Ocirc", 212 },
{ "Otilde", 213 },
{ "Ouml", 214 },
{ "times", 215 },
{ "Oslash", 216 },
{ "Ugrave", 217 },
{ "Uacute", 218 },
{ "Ucirc", 219 },
{ "Uuml", 220 },
{ "Yacute", 221 },
{ "THORN", 222 },
{ "szlig", 223 },
{ "agrave", 224 },
{ "aacute", 225 },
{ "acirc", 226 },
{ "atilde", 227 },
{ "auml", 228 },
{ "aring", 229 },
{ "aelig", 230 },
{ "ccedil", 231 },
{ "egrave", 232 },
{ "eacute", 233 },
{ "ecirc", 234 },
{ "euml", 235 },
```

```
{ "igrave", 236 },
{ "iacute", 237 },
{ "icirc", 238 },
{ "iuml", 239 },
{ "eth", 240 },
{ "ntilde", 241 },
{ "ograve", 242 },
{ "oacute", 243 },
{ "ocirc", 244 },
{ "otilde", 245 },
{ "ouml", 246 },
{ "divide", 247 },
{ "oslash", 248 },
{ "ugrave", 249 },
{ "uacute", 250 },
{ "ucirc", 251 },
{ "uuml", 252 },
{ "yacute", 253 },
{ "thorn", 254 },
{ "yuml", 255 },
{ "fnof", 402 },
/* Greek */
{ "Alpha", 913 },
{ "Beta", 914 },
{ "Gamma", 915 },
{ "Delta", 916 },
{ "Epsilon", 917 },
{ "Zeta", 918 },
{ "Eta", 919 },
{ "Theta", 920 },
{ "Iota", 921 },
{ "Kappa", 922 },
{ "Lambda", 923 },
{ "Mu", 924 },
{ "Nu", 925 },
{ "Xi", 926 },
{ "Omicron", 927 },
{ "Pi", 928 },
{ "Rho", 929 },
{ "Sigma", 931 },
{ "Tau", 932 },
{ "Upsilon", 933 },
{ "Phi", 934 },
{ "Chi", 935 },
{ "Psi", 936 },
{ "Omega", 937 },
{ "alpha", 945 },
{ "beta", 946 },
{ "gamma", 947 },
{ "delta", 948 },
{ "epsilon", 949 },
{ "zeta", 950 },
{ "eta", 951 },
{ "theta", 952 },
{ "iota", 953 },
{ "kappa", 954 },
{ "lambda", 955 },
{ "mu", 956 },
```

```
{ "nu", 957 },
{ "xi", 958 },
{ "omicron", 959 },
{ "pi", 960 },
{ "rho", 961 },
{ "sigmaf", 962 },
{ "sigma", 963 },
{ "tau", 964 },
{ "upsilon", 965 },
{ "phi", 966 },
{ "chi", 967 },
{ "psi", 968 },
{ "omega", 969 },
{ "thetasym", 977 },
{ "upsih", 978 },
{ "piv", 982 },
/* General Punctuation */
{ "bull", 8226 },
{ "hellip", 8230 },
{ "prime", 8242 },
{ "Prime", 8243 },
{ "oline", 8254 },
{ "frasl", 8260 },
/* Letterlike Symbols */
{ "weierp", 8472 },
{ "image", 8465 },
{ "real", 8476 },
{ "trade", 8482 },
{ "alefsym", 8501 },
/* Arrows */
{ "larr", 8592 },
{ "uarr", 8593 },
{ "rarr", 8594 },
{ "darr", 8595 },
{ "harr", 8596 },
{ "crarr", 8629 },
{ "lArr", 8656 },
{ "uArr", 8657 },
{ "rArr", 8658 },
{ "dArr", 8659 },
{ "hArr", 8660 },
/* Mathematical Operators */
{ "forall", 8704 },
{ "part", 8706 },
{ "exist", 8707 },
{ "empty", 8709 },
{ "nabla", 8711 },
{ "isin", 8712 },
{ "notin", 8713 },
{ "ni", 8715 },
{ "prod", 8719 },
{ "sum", 8721 },
{ "minus", 8722 },
{ "lowast", 8727 },
{ "radic", 8730 },
{ "prop", 8733 },
{ "infin", 8734 },
{ "and", 8743 },
```

```
{ "or", 8744 },
{ "cap", 8745 },
{ "cup", 8746 },
{ "int", 8747 },
{ "there4", 8756 },
{ "sim", 8764 },
{ "cong", 8773 },
{ "asymp", 8776 },
{ "ne", 8800 },
{ "equiv", 8801 },
{ "le", 8804 },
{ "ge", 8805 },
{ "sub", 8834 },
{ "sup", 8835 },
{ "nsub", 8836 },
{ "sube", 8838 },
{ "supe", 8839 },
{ "oplus", 8853 },
{ "otimes", 8855 },
{ "perp", 8869 },
{ "sdot", 8901 },
/* Miscellaneous Technical */
{ "lceil", 8968 },
{ "rceil", 8969 },
{ "lfloor", 8970 },
{ "rfloor", 8971 },
{ "lang", 9001 },
{ "rang", 9002 },
/* Geometric Shapes */
{ "loz", 9674 },
/* Miscellaneous Symbols */
{ "spades", 9824 },
{ "clubs", 9827 },
{ "hearts", 9829 },
{ "diams", 9830 },
{ "quot", 34 },
{ "amp", 38 },
{ "lt", 60 },
{ "gt", 62 },
/* Latin Extended-A */
{ "OElig", 338 },
{ "oelig", 339 },
{ "Scaron", 352 },
{ "scaron", 353 },
{ "Yuml", 376 },
/* Spacing Modifier Letters */
{ "circ", 710 },
{ "tilde", 732 },
/* General Punctuation */
{ "ensp", 8194 },
{ "emsp", 8195 },
{ "thinsp", 8201 },
{ "zwnj", 8204 },
{ "zwj", 8205 },
{ "lrm", 8206 },
{ "rlm", 8207 },
{ "ndash", 8211 },
{ "mdash", 8212 },
```

```
	{ "lsquo", 8216 },
	{ "rsquo", 8217 },
	{ "sbquo", 8218 },
	{ "ldquo", 8220 },
	{ "rdquo", 8221 },
	{ "bdquo", 8222 },
	{ "dagger", 8224 },
	{ "Dagger", 8225 },
	{ "permil", 8240 },
	{ "lsaquo", 8249 },
	{ "rsaquo", 8250 },
	{ "euro", 8364 },
	{ NULL, 0 } /* marks end of list */
} /*StaticEntityNames*/;

typedef std::map<std::string, unsigned int> EntityNameMap;
typedef std::pair<std::string, unsigned int> EntityNamePair;
static EntityNameMap EntityNames;

/* writes Ch in UTF-8 encoding to Out. Note this version only deals with characters up
to 16 bits. */
static void WriteUTF8(std::string& Out, unsigned int Ch)
{
	if (Ch >= 0x800)
	{
		Out += (0xE0 | Ch >> 12 & 0x0F);
		Out += (0x80 | Ch >> 6 & 0x3F);
		Out += (0x80 | Ch & 0x3F);
	}
	else if (Ch >= 0x80)
	{
		Out += (0xC0 | Ch >> 6 & 0x1F);
		Out += (0x80 | Ch & 0x3F);
	}
	else
	{
		Out += Ch;
	} /*if*/
} /*WriteUTF8*/

/* copies In to Out, expanding any HTML entity references into literal UTF-8 characters.
*/
const std::string UnquoteHTML(const std::string& InBuffer)
{
	enum
	{
		NoMatch,
		MatchBegin,
		MatchName,
		MatchNumber,
		MatchDecimalNumber,
		MatchHexNumber,
	} MatchState;
	std::string MatchingName;
	unsigned int CharCode;
	bool ProcessedChar, GotCharCode;
	MatchState = NoMatch;
	std::string OutBuffer;
```

```
	for (int index = 0; index < InBuffer.length(); index++)
	{
		const unsigned char ThisCh = InBuffer[index];
		ProcessedChar = false; /* to begin with */
		GotCharCode = false; /* to begin with */
		switch (MatchState)
		{
			case MatchBegin:
			{
				if (ThisCh == '#')
				{
					MatchState = MatchNumber;
					ProcessedChar = true;
				}
				else if ((ThisCh >= 'a') && (ThisCh <= 'z')
					|| (ThisCh >= 'A') && (ThisCh <= 'Z'))
				{
					MatchingName.append(1, ThisCh);
					MatchState = MatchName;
					ProcessedChar = true;
				}
				else
				{
					OutBuffer += '&';
					MatchState = NoMatch;
				} /*if*/
				break;
			}
			case MatchName:
			{
				if ((ThisCh >= 'a') && (ThisCh <= 'z')
					|| (ThisCh >= 'A') && (ThisCh <= 'Z')
					|| (ThisCh >= '0') && (ThisCh <= '9'))
				{
					MatchingName.append(1, ThisCh);
					ProcessedChar = true;
				}
				else if (ThisCh == ';')
				{
					if (EntityNames.empty())
					{
						/* first use, load EntityNames from
StaticEntityNames */
						const EntityNameEntry* ThisEntry;
						ThisEntry = StaticEntityNames;
						for (;;)
						{
							if (ThisEntry->Name == NULL)
								break;

	EntityNames.insert(EntityNamePair(std::string(ThisEntry->Name), ThisEntry-
>Value));
							++ThisEntry;
						} /*for*/
					} /*if*/
					const EntityNameMap::const_iterator NameEntry =
EntityNames.find(MatchingName);
					if (NameEntry != EntityNames.end())
```

```
                {
                        CharCode = NameEntry->second;
                        ProcessedChar = true;
                        GotCharCode = true;
                } /*if*/
        } /*if*/
        if (!ProcessedChar)
        {
                OutBuffer += '&';
                for (unsigned int i = 0; i < MatchingName.size(); ++i)
                {
                        OutBuffer += MatchingName[i];
                } /*for*/
                MatchState = NoMatch;
        } /*if*/
        break;
}
case MatchNumber:
{
        if ((ThisCh == 'x') || (ThisCh == 'X'))
        {
                ProcessedChar = true;
                MatchState = MatchHexNumber;
                CharCode = 0;
        }
        else if ((ThisCh >= '0') && (ThisCh <= '9'))
        {
                CharCode = ThisCh - '0';
                MatchState = MatchDecimalNumber;
                ProcessedChar = true;
        }
        else
        {
                MatchState = NoMatch;
        } /*if*/
        break;
}
case MatchDecimalNumber:
{
        if ((ThisCh >= '0') && (ThisCh <= '9'))
        {
                CharCode = CharCode * 10 + ThisCh - '0';
                ProcessedChar = true;
        }
        else if (ThisCh == ';')
        {
                ProcessedChar = true;
                GotCharCode = true;
        }
        else
        {
                MatchState = NoMatch;
        } /*if*/
        break;
}
case MatchHexNumber:
{
        if ((ThisCh >= '0') && (ThisCh <= '9'))
```

```
                        {
                              CharCode = CharCode * 16 + ThisCh - '0';
                              ProcessedChar = true;
                        }
                        else if ((ThisCh >= 'a') && (ThisCh <= 'f'))
                        {
                              CharCode = CharCode * 16 + ThisCh - 'a' + 10;
                              ProcessedChar = true;
                        }
                        else if ((ThisCh >= 'A') && (ThisCh <= 'F'))
                        {
                              CharCode = CharCode * 16 + ThisCh - 'A' + 10;
                              ProcessedChar = true;
                        }
                        else if (ThisCh == ';')
                        {
                              ProcessedChar = true;
                              GotCharCode = true;
                        }
                        else
                        {
                              MatchState = NoMatch;
                        } /*if*/
                        break;
                  }
            } /*switch*/
            if (GotCharCode)
            {
                  WriteUTF8(OutBuffer, CharCode);
                  MatchState = NoMatch;
            }
            else if (!ProcessedChar && (MatchState == NoMatch))
            {
                  if (ThisCh == '&')
                  {
                        MatchState = MatchBegin;
                        MatchingName.erase();
                  }
                  else
                  {
                        OutBuffer += ThisCh;
                  } /*if*/
            } /*if*/
      } /*for*/
      return OutBuffer;
} /*UnquoteHTML*/
```

9.4. Fișierul WebSearchEngineDlg.cpp

```
/* This file is part of Web Search Engine application developed by Mihai MOGA.

Web Search Engine is free software: you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by the Open
Source Initiative, either version 3 of the License, or any later version.

Web Search Engine is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
```

```
or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
details.

You should have received a copy of the GNU General Public License along with
Web Search Engine.  If not, see <http://www.opensource.org/licenses/gpl-3.0.html>*/

// WebSearchEngineDlg.cpp : implementation file
//

#include "stdafx.h"
#include "WebSearchEngine.h"
#include "WebSearchEngineDlg.h"
#include "WebSearchEngineExt.h"
#include "ConnectionSettingsDlg.h"
#include "HtmlToText.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

DWORD WINAPI CrawlingThreadProc(LPVOID lpParam);

// CAboutDlg dialog used for App About

class CAboutDlg : public CDialogEx
{
public:
	CAboutDlg();

	// Dialog Data
#ifdef AFX_DESIGN_TIME
	enum { IDD = IDD_ABOUTBOX };
#endif

protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

// Implementation
protected:
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialogEx(IDD_ABOUTBOX)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
END_MESSAGE_MAP()

// CWebSearchEngineDlg dialog

CWebSearchEngineDlg::CWebSearchEngineDlg(CWnd* pParent /*=NULL*/)
	: CDialogEx(IDD_WEBSEARCHENGINE_DIALOG, pParent)
```

```
{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
	m_bThreadRunning = false;
	m_nThreadID = 0;
}

void CWebSearchEngineDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_CRAWLING, m_pCrawling);
	DDX_Control(pDX, IDC_PROGRESS, m_pProgress);
	DDX_Control(pDX, IDC_WEBPAGES, m_pWebpageCounter);
	DDX_Control(pDX, IDC_KEYWORDS, m_pKeywordCounter);
}

BEGIN_MESSAGE_MAP(CWebSearchEngineDlg, CDialogEx)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDCANCEL, &CWebSearchEngineDlg::OnBnClickedCancel)
END_MESSAGE_MAP()

// CWebSearchEngineDlg message handlers

BOOL CWebSearchEngineDlg::OnInitDialog()
{
	CDialogEx::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		BOOL bNameValid;
		CString strAboutMenu;
		bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
		ASSERT(bNameValid);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here
	CConnectionSettingsDlg pConnectionSettingsDlg(this);
	if (pConnectionSettingsDlg.DoModal() != IDOK)
		return FALSE;
```

```
	CWinApp* pWinApp = AfxGetApp();
	ASSERT(pWinApp != NULL);

	CString strHostName = pWinApp->GetProfileString(REGKEY_SECTION, REGKEY_HOSTNAME,
DEFAULT_HOSTNAME);
	CString strHostPort = pWinApp->GetProfileString(REGKEY_SECTION, REGKEY_HOSTPORT,
DEFAULT_HOSTPORT);
	CString strDatabase = pWinApp->GetProfileString(REGKEY_SECTION, REGKEY_DATABASE,
DEFAULT_DATABASE);
	// CString strFileName = pWinApp->GetProfileString(REGKEY_SECTION,
REGKEY_FILENAME, DEFAULT_FILENAME);
	CString strUsername = pWinApp->GetProfileString(REGKEY_SECTION, REGKEY_USERNAME,
DEFAULT_USERNAME);

	TCHAR lpszPassword[0x100] = { 0, };
	VERIFY(GetRegistryPassword(NULL, REGKEY_SECTION, REGKEY_PASSWORD, lpszPassword,
DEFAULT_PASSWORD));

	SQLRETURN nRet = m_pEnvironment.Create();
	ODBC_CHECK_RETURN_FALSE(nRet, m_pEnvironment);

	nRet = m_pEnvironment.SetAttr(SQL_ATTR_ODBC_VERSION, SQL_OV_ODBC3);
	ODBC_CHECK_RETURN_FALSE(nRet, m_pEnvironment);

	nRet = m_pEnvironment.SetAttrU(SQL_ATTR_CONNECTION_POOLING, SQL_CP_OFF);
	ODBC_CHECK_RETURN_FALSE(nRet, m_pEnvironment);

	nRet = m_pConnection.Create(m_pEnvironment);
	ODBC_CHECK_RETURN_FALSE(nRet, m_pConnection);

	_stprintf(m_sConnectionInString, _T("Driver={MySQL ODBC 3.51
Driver};Server=%s;Port=%s;Database=%s;Uid=%s;Pwd=%s;"),
		strHostName.GetBuffer(0), strHostPort.GetBuffer(0),
strDatabase.GetBuffer(0), strUsername.GetBuffer(0), lpszPassword);
	strHostName.ReleaseBuffer();
	strHostPort.ReleaseBuffer();
	strDatabase.ReleaseBuffer();
	strUsername.ReleaseBuffer();
	nRet = m_pConnection.DriverConnect(const_cast<SQLTCHAR*>(reinterpret_cast<const
SQLTCHAR*>(m_sConnectionInString)), m_sConnectionOutString);
	ODBC_CHECK_RETURN_FALSE(nRet, m_pConnection);

	m_pWebpageCounter.SetWindowText(_T("0"));
	m_pKeywordCounter.SetWindowText(_T("0"));

	CGenericStatement pGenericStatement;
	VERIFY(pGenericStatement.Execute(m_pConnection, _T("DROP TABLE IF EXISTS
`occurrence`;")));
	VERIFY(pGenericStatement.Execute(m_pConnection, _T("DROP TABLE IF EXISTS
`keyword`;")));
	VERIFY(pGenericStatement.Execute(m_pConnection, _T("DROP TABLE IF EXISTS
`webpage`;")));
	VERIFY(pGenericStatement.Execute(m_pConnection, _T("CREATE TABLE `webpage`
(`webpage_id` BIGINT NOT NULL AUTO_INCREMENT, `url` VARCHAR(256) NOT NULL, `title`
VARCHAR(256) NOT NULL, `content` TEXT NOT NULL, PRIMARY KEY(`webpage_id`))
ENGINE=InnoDB;")));
```

```
	VERIFY(pGenericStatement.Execute(m_pConnection, _T("CREATE TABLE `keyword` 
(`keyword_id` BIGINT NOT NULL AUTO_INCREMENT, `name` VARCHAR(256) NOT NULL, PRIMARY 
KEY(`keyword_id`)) ENGINE=InnoDB;")));
	VERIFY(pGenericStatement.Execute(m_pConnection, _T("CREATE TABLE `occurrence` 
(`webpage_id` BIGINT NOT NULL, `keyword_id` BIGINT NOT NULL, `counter` BIGINT NOT NULL, 
`pagerank` REAL NOT NULL, PRIMARY KEY(`webpage_id`, `keyword_id`), FOREIGN KEY 
webpage_fk(webpage_id) REFERENCES webpage(webpage_id), FOREIGN KEY 
keyword_fk(keyword_id) REFERENCES keyword(keyword_id)) ENGINE=InnoDB;")));

	m_hThread = ::CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)CrawlingThreadProc, 
this, 0, &m_nThreadID);

	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CWebSearchEngineDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialogEx::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CWebSearchEngineDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 
0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialogEx::OnPaint();
	}
}
```

```
// The system calls this function to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CWebSearchEngineDlg::OnQueryDragIcon()
{
	return static_cast<HCURSOR>(m_hIcon);
}

DWORD WINAPI CrawlingThreadProc(LPVOID lpParam)
{
	std::string lpszURL, lpszFilename;
	if (lpParam != NULL)
	{
		CWebSearchEngineDlg* pWebSearchEngineDlg = (CWebSearchEngineDlg*)lpParam;
		pWebSearchEngineDlg->m_bThreadRunning = true;
		pWebSearchEngineDlg->m_pProgress.SetMarquee(TRUE, 30);
		AddURLToFrontier("https://en.wikipedia.org/");
		while (pWebSearchEngineDlg->m_bThreadRunning)
		{
			if (ExtractURLFromFrontier(lpszURL))
			{
				pWebSearchEngineDlg-
>m_pCrawling.SetWindowText(CString(lpszURL.c_str()));
				if (DownloadURLToFile(lpszURL, lpszFilename))
				{
					if (!ProcessHTML(pWebSearchEngineDlg, lpszFilename,
lpszURL))
					{
						break;
					}
				}
			}
			else
				break;
		}

		pWebSearchEngineDlg->m_bThreadRunning = false;
		pWebSearchEngineDlg->m_pProgress.SetMarquee(FALSE, 30);
	}

	::ExitThread(0);
	return 0;
}

BOOL WaitWithMessageLoop(HANDLE hEvent, DWORD dwTimeout)
{
	DWORD dwRet;
	MSG msg;
	hEvent = hEvent ? hEvent : CreateEvent(NULL, FALSE, FALSE, NULL);

	while (true)
	{
		dwRet = MsgWaitForMultipleObjects(1, &hEvent, FALSE, dwTimeout,
QS_ALLINPUT);
		if (dwRet == WAIT_OBJECT_0)
			return TRUE;
		if (dwRet != WAIT_OBJECT_0 + 1)
			break;
		while (PeekMessage(&msg, NULL, NULL, NULL, PM_REMOVE))
```

```
			{
				TranslateMessage(&msg);
				DispatchMessage(&msg);
				if (WaitForSingleObject(hEvent, 0) == WAIT_OBJECT_0)
					return TRUE;
			}
		}
		return FALSE;
}

void CWebSearchEngineDlg::OnBnClickedCancel()
{
		if (m_bThreadRunning)
		{
			m_bThreadRunning = false;
			VERIFY(WaitWithMessageLoop(m_hThread, INFINITE));
		}
		CDialogEx::OnCancel();
}
```

9.5. Fișierul WebSearchEngineDlg.h

```
/* This file is part of Web Search Engine application developed by Mihai MOGA.

Web Search Engine is free software: you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by the Open
Source Initiative, either version 3 of the License, or any later version.

Web Search Engine is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
details.

You should have received a copy of the GNU General Public License along with
Web Search Engine.  If not, see <http://www.opensource.org/licenses/gpl-3.0.html>*/

// WebSearchEngineDlg.h : header file
//

#pragma once

#include "ODBCWrappers.h"
#include "afxwin.h"
#include "afxcmn.h"

// CWebSearchEngineDlg dialog
class CWebSearchEngineDlg : public CDialogEx
{
		// Construction
public:
		CWebSearchEngineDlg(CWnd* pParent = NULL);	// standard constructor

// Dialog Data
#ifdef AFX_DESIGN_TIME
		enum { IDD = IDD_WEBSEARCHENGINE_DIALOG };
#endif
		CEdit m_pCrawling;
		CProgressCtrl m_pProgress;
```

```
	CStatic m_pWebpageCounter;
	CStatic m_pKeywordCounter;

protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

// Implementation
public:
	bool m_bThreadRunning;
	HICON m_hIcon;
	CODBC::CEnvironment m_pEnvironment;
	CODBC::CConnection m_pConnection;
	CODBC::String m_sConnectionOutString;
	TCHAR m_sConnectionInString[0x100];
	DWORD m_nThreadID;
	HANDLE m_hThread;

protected:
	// Generated message map functions
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnBnClickedCancel();
	DECLARE_MESSAGE_MAP()
};
```

9.6. Fișierul WebSearchEngineExt.cpp

```
/* This file is part of Web Search Engine application developed by Mihai MOGA.

Web Search Engine is free software: you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by the Open
Source Initiative, either version 3 of the License, or any later version.

Web Search Engine is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
details.

You should have received a copy of the GNU General Public License along with
Web Search Engine.  If not, see <http://www.opensource.org/licenses/gpl-3.0.html>*/

#include "stdafx.h"
#include "WebSearchEngineExt.h"
#include "HtmlToText.h"
#include "ODBCWrappers.h"
#include <string>
#include <vector>
#include <map>
#include <codecvt>
#include <iostream>
#include <fstream>
#include <algorithm>
#include <cctype>
#include <Windows.h>

#include <Urlmon.h>
```

```
#pragma comment(lib, "Urlmon")

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

FrontierArray gFrontierOlder; // visited URLs
FrontierArray gFrontierArray; // enque/deque URLs
FrontierScore gFrontierScore; // the score of each URL
WebpageIndex gWebpageID; // map of each webpage ID
KeywordIndex gKeywordID; // map of each keyword ID
KeywordArray gWordArray; // list of all keywords

std::vector<std::wstring> gDataMiningTerms;

static __int64 gCurrentWebpageID = 0;
static __int64 gCurrentKeywordID = 0;

#define DELIMITERS _T("\t\n\r\"\' !?#$%&|(){}[]*/+-:;<>=.,")

const std::string UnquoteHTML(const std::string& InBuffer);

// convert UTF-8 string to wstring
std::wstring utf8_to_wstring(const std::string& str)
{
	std::wstring_convert<std::codecvt_utf8<wchar_t>> myconv;
	return myconv.from_bytes(str);
}

// convert wstring to UTF-8 string
std::string wstring_to_utf8(const std::wstring& str)
{
	std::wstring_convert<std::codecvt_utf8<wchar_t>> myconv;
	return myconv.to_bytes(str);
}

// adds a new URL to frontier
bool AddURLToFrontier(const std::string& lpszURL)
{
	bool found = false;
	for (auto it = gFrontierOlder.begin(); it != gFrontierOlder.end(); it++)
	{
		if (lpszURL.compare(it->c_str()) == 0)
		{
			return true; // URL already visited
		}
	}
	for (auto it = gFrontierArray.begin(); it != gFrontierArray.end(); it++)
	{
		if (lpszURL.compare(it->c_str()) == 0)
		{
			found = true;
			gFrontierScore[lpszURL]++;
			break;
		}
	}
	if (!found)
	{
```

```
			gFrontierArray.push_back(lpszURL);
			gFrontierScore[lpszURL] = 1;
		}
		return true;
}

// gets the next URL from frontier
bool ExtractURLFromFrontier(std::string& lpszURL)
{
		lpszURL = "";
		int score = 0;
		if (gFrontierArray.size() > 0)
		{
			for (std::string it : gFrontierArray)
			{
				if (lpszURL.empty()) // select the first element
				{
					lpszURL = it;
					score = gFrontierScore[it];
				}
				else
				{
					if (score < gFrontierScore[it]) // update the selection if
necesary
					{
						lpszURL = it;
						score = gFrontierScore[it];
					}
				}
			}

			// remove selected element from frontier
			for (auto it = gFrontierArray.begin(); it != gFrontierArray.end(); it++)
			{
				if (lpszURL.compare(it->c_str()) == 0)
				{
					gFrontierArray.erase(it);
					break;
				}
			}
			for (auto it = gFrontierScore.begin(); it != gFrontierScore.end(); it++)
			{
				if (lpszURL.compare(it->first.c_str()) == 0)
				{
					gFrontierScore.erase(it);
					break;
				}
			}

			gFrontierOlder.push_back(lpszURL);
			return true;
		}
		return false;
}

// function to download a Web page
bool DownloadURLToFile(const std::string& lpszURL, std::string& lpszFilename)
{
```

```
	char lpszTempPath[MAX_PATH + 1] = { 0, };
	char lpszTempFile[MAX_PATH + 1] = { 0, };

	const DWORD dwTempPath = GetTempPathA(sizeof(lpszTempPath) - 1, lpszTempPath);
	if (dwTempPath > 0)
	{
		lpszTempPath[dwTempPath] = '\0';
		if (GetTempFileNameA(lpszTempPath, "map", 0, lpszTempFile) != 0)
		{
			if (URLDownloadToFileA(NULL, lpszURL.c_str(), lpszTempFile, 0, NULL)
== S_OK)
			{
				lpszFilename = lpszTempFile;
				return true;
			}
		}
	}
	return false;
}

// finds and replaces all occurences from a given string
int findAndReplaceAll(std::wstring& data, std::wstring toSearch, std::wstring
replaceStr)
{
	int counter = 0;
	// Get the first occurrence
	size_t pos = data.find(toSearch);

	// Repeat till end is reached
	while (pos != std::string::npos)
	{
		counter++;
		// Replace this occurrence of Sub String
		data.replace(pos, toSearch.size(), replaceStr);
		// Get the next occurrence from the current position
		pos = data.find(toSearch, pos + replaceStr.size());
	}
	return counter;
}

// trim from start (in place)
static inline std::wstring ltrim(std::wstring s) {
	s.erase(s.begin(), std::find_if(s.begin(), s.end(), [](int ch) {
		return !iswspace(ch);
	}));
	return s;
}

// trim from end (in place)
static inline std::wstring rtrim(std::wstring s) {
	s.erase(std::find_if(s.rbegin(), s.rend(), [](int ch) {
		return !iswspace(ch);
	}).base(), s.end());
	return s;
}

// trim from both ends (in place)
static inline std::wstring trim(std::wstring s)
```

```
{
	return ltrim(rtrim(s));
}

// string to lower
static inline std::wstring to_lower(std::wstring s)
{
	std::transform(s.begin(), s.end(), s.begin(), std::tolower);
	return s;
}

// string to upper
static inline std::wstring to_upper(std::wstring s)
{
	std::transform(s.begin(), s.end(), s.begin(), std::toupper);
	return s;
}

// process HTML page and extract plain text and hyperlinks
bool ProcessHTML(CWebSearchEngineDlg* pWebSearchEngineDlg, const std::string&
lpszFilename, const std::string& lpszURL)
{
	CString strMessage;
	CHtmlToText pHtmlToText;
	std::ifstream pHtmlFile(lpszFilename);
	std::string pHtmlContent;
	if (pHtmlFile.is_open())
	{
		pHtmlFile.seekg(0, std::ios::end);
		pHtmlContent.reserve(pHtmlFile.tellg());
		pHtmlFile.seekg(0, std::ios::beg);

		pHtmlContent.assign((std::istreambuf_iterator<char>(pHtmlFile)),
			std::istreambuf_iterator<char>());
		pHtmlFile.close();

		std::wstring pTitle;
		std::size_t found = pHtmlContent.find("<title>", 0);
		if (std::string::npos != found)
		{
			found += 7;
			const std::size_t last_char = pHtmlContent.find("</title>", found);
			if (std::string::npos != last_char)
			{
				pTitle =
trim(utf8_to_wstring(UnquoteHTML(pHtmlContent.substr(found, last_char -
found)))).substr(0, 0x100 - 1);
			}
		}
		if (pTitle.length() == 0)
			return true;

		found = pHtmlContent.find("<a href=\"", 0);
		while (std::string::npos != found)
		{
			found += 9;
			const std::size_t last_char = pHtmlContent.find('\"', found);
			if (std::string::npos != last_char)
```

```
                {
                        std::string hyperlink = pHtmlContent.substr(found, last_char -
found);
                        std::size_t diez = hyperlink.find('#');
                        if (std::string::npos != diez)
                        {
                                hyperlink = hyperlink.substr(0, diez);
                        }
                        // OutputDebugString(CString(hyperlink.c_str()) + _T("\n"));
                        TCHAR lpszRelativeURL[MAX_URL_LENGTH] = { 0 };
                        TCHAR lpszAbsoluteURL[MAX_URL_LENGTH] = { 0 };
                        TCHAR lpszDomainURL[MAX_URL_LENGTH] = { 0 };
                        DWORD dwLength = MAX_URL_LENGTH;
                        wcscpy_s(lpszRelativeURL, _countof(lpszRelativeURL),
CString(hyperlink.c_str()));
                        wcscpy_s(lpszDomainURL, _countof(lpszDomainURL),
CString(lpszURL.c_str()));
                        if (CoInternetCombineUrl(lpszDomainURL, lpszRelativeURL, 0,
lpszAbsoluteURL, MAX_URL_LENGTH, &dwLength, 0) == S_OK)
                        {
                                lpszAbsoluteURL[dwLength] = '\0';
                                // OutputDebugString(CString(lpszAbsoluteURL) +
_T("\n"));
                                hyperlink = CStringA(lpszAbsoluteURL);
                                if (hyperlink.length() < 0x100)
                                        AddURLToFrontier(hyperlink);
                        }
                }
                found = pHtmlContent.find("<a href=\"", found);
        }

        const std::wstring& pURL = utf8_to_wstring(lpszURL);
        OutputDebugString(CString(pURL.c_str()) + _T("\n"));
        // OutputDebugString(CString(pTitle.c_str()) + _T("\n"));
        std::wstring& pPlainText =
trim(utf8_to_wstring(UnquoteHTML(pHtmlToText.Convert(pHtmlContent))));
        findAndReplaceAll(pPlainText, _T("\t"), _T(" "));
        findAndReplaceAll(pPlainText, _T("\n"), _T(" "));
        findAndReplaceAll(pPlainText, _T("\r"), _T(" "));
        while (findAndReplaceAll(pPlainText, _T("  "), _T(" ")) > 0);
        pPlainText = pPlainText.substr(0, 0x10000 - 1);
        OutputDebugString(CString(pPlainText.c_str()) + _T("\n"));

        SQLRETURN nRet = 0;
        CWebpageInsert pWebpageInsert;
        if (!pWebpageInsert.Execute(pWebSearchEngineDlg->m_pConnection, pURL,
pTitle, pPlainText)) // add webpage to database
        {
                pWebSearchEngineDlg->m_pProgress.SetMarquee(FALSE, 30);
                do {
                        ::MessageBeep(0xFFFFFFFF);
                        nRet = pWebSearchEngineDlg->m_pConnection.Disconnect();
                        ::Sleep(60 * 1000);
                        nRet = pWebSearchEngineDlg-
>m_pConnection.DriverConnect(const_cast<SQLTCHAR*>(reinterpret_cast<const
SQLTCHAR*>(pWebSearchEngineDlg->m_sConnectionInString)), pWebSearchEngineDlg-
>m_sConnectionOutString);
                } while (!SQL_SUCCEEDED(nRet));
```

```
                  if (!pWebpageInsert.Execute(pWebSearchEngineDlg->m_pConnection, pURL,
pTitle, pPlainText))
                  {
                        pWebSearchEngineDlg->MessageBox(_T("Cannot insert webpage into
the database"), _T("Error"), MB_OK);
                        return false;
                  }
                  pWebSearchEngineDlg->m_pProgress.SetMarquee(TRUE, 30);
            }
            gWebpageID[pURL] = ++gCurrentWebpageID;
            pWebSearchEngineDlg-
>m_pWebpageCounter.SetWindowText(std::to_wstring(gCurrentWebpageID).c_str());

            const std::wstring pLowerCaseText = to_lower(pPlainText);
            // Skip delimiters at beginning.
            std::size_t lastPos = pLowerCaseText.find_first_not_of(DELIMITERS, 0);
            // Find first "non-delimiter".
            std::size_t pos = pLowerCaseText.find_first_of(DELIMITERS, lastPos);

            while ((std::string::npos != pos) || (std::string::npos != lastPos))
            {
                  // Found a token, add it to the vector.
                  const std::wstring pKeyword = pLowerCaseText.substr(lastPos, pos -
lastPos);
                  // Skip delimiters.  Note the "not_of"
                  lastPos = pLowerCaseText.find_first_not_of(DELIMITERS, pos);
                  // Find next "non-delimiter"
                  pos = pLowerCaseText.find_first_of(DELIMITERS, lastPos);

                  if (pKeyword.length() == 0)
                        continue;

                  if (pKeyword.find_first_not_of(_T("abcdefghijklmnopqrstuvwxyz")) !=
std::string::npos)
                        continue;

                  if ((pKeyword.compare(_T("of")) == 0) ||
                        (pKeyword.compare(_T("with")) == 0) ||
                        (pKeyword.compare(_T("at")) == 0) ||
                        (pKeyword.compare(_T("from")) == 0) ||
                        (pKeyword.compare(_T("into")) == 0) ||
                        (pKeyword.compare(_T("during")) == 0) ||
                        (pKeyword.compare(_T("against")) == 0) ||
                        (pKeyword.compare(_T("among")) == 0) ||
                        (pKeyword.compare(_T("throughout")) == 0) ||
                        (pKeyword.compare(_T("despite")) == 0) ||
                        (pKeyword.compare(_T("towards")) == 0) ||
                        (pKeyword.compare(_T("upon")) == 0) ||
                        (pKeyword.compare(_T("concerning")) == 0) ||
                        (pKeyword.compare(_T("to")) == 0) ||
                        (pKeyword.compare(_T("in")) == 0) ||
                        (pKeyword.compare(_T("for")) == 0) ||
                        (pKeyword.compare(_T("on")) == 0) ||
                        (pKeyword.compare(_T("by")) == 0) ||
                        (pKeyword.compare(_T("about")) == 0) ||
                        (pKeyword.compare(_T("like")) == 0) ||
                        (pKeyword.compare(_T("through")) == 0) ||
                        (pKeyword.compare(_T("over")) == 0) ||
```

```
                        (pKeyword.compare(_T("before")) == 0) ||
                        (pKeyword.compare(_T("between")) == 0) ||
                        (pKeyword.compare(_T("after")) == 0) ||
                        (pKeyword.compare(_T("since")) == 0) ||
                        (pKeyword.compare(_T("without")) == 0) ||
                        (pKeyword.compare(_T("under")) == 0) ||
                        (pKeyword.compare(_T("within")) == 0) ||
                        (pKeyword.compare(_T("following")) == 0) ||
                        (pKeyword.compare(_T("across")) == 0) ||
                        (pKeyword.compare(_T("behind")) == 0) ||
                        (pKeyword.compare(_T("beyond")) == 0) ||
                        (pKeyword.compare(_T("plus")) == 0) ||
                        (pKeyword.compare(_T("except")) == 0) ||
                        (pKeyword.compare(_T("but")) == 0) ||
                        (pKeyword.compare(_T("up")) == 0) ||
                        (pKeyword.compare(_T("out")) == 0) ||
                        (pKeyword.compare(_T("around")) == 0) ||
                        (pKeyword.compare(_T("down")) == 0) ||
                        (pKeyword.compare(_T("off")) == 0) ||
                        (pKeyword.compare(_T("above")) == 0) ||
                        (pKeyword.compare(_T("near")) == 0))
                        continue;

                OutputDebugString(CString(pKeyword.c_str()) + _T("\n"));
                bool already_added = false;
                for (auto it = gWordArray.begin(); it != gWordArray.end(); it++)
                {
                        if (pKeyword.compare(it->c_str()) == 0)
                        {
                                already_added = true;
                                break;
                        }
                }

                if (!already_added)
                {
                        gWordArray.push_back(pKeyword);

                        CKeywordInsert pKeywordInsert;
                        if (!pKeywordInsert.Execute(pWebSearchEngineDlg-
>m_pConnection, pKeyword)) // add keyword to database
                        {
                                pWebSearchEngineDlg->m_pProgress.SetMarquee(FALSE, 30);
                                do {
                                        ::MessageBeep(0xFFFFFFFF);
                                        nRet = pWebSearchEngineDlg-
>m_pConnection.Disconnect();
                                        ::Sleep(60 * 1000);
                                        nRet = pWebSearchEngineDlg-
>m_pConnection.DriverConnect(const_cast<SQLTCHAR*>(reinterpret_cast<const
SQLTCHAR*>(pWebSearchEngineDlg->m_sConnectionInString)), pWebSearchEngineDlg-
>m_sConnectionOutString);
                                } while (!SQL_SUCCEEDED(nRet));
                                if (!pKeywordInsert.Execute(pWebSearchEngineDlg-
>m_pConnection, pKeyword))
                                {
                                        pWebSearchEngineDlg->MessageBox(_T("Cannot insert
keyword into the database"), _T("Error"), MB_OK);
```

```
                                    return false;
                                }
                                pWebSearchEngineDlg->m_pProgress.SetMarquee(TRUE, 30);
                            }
                            gKeywordID[pKeyword] = ++gCurrentKeywordID;
                            pWebSearchEngineDlg-
>m_pKeywordCounter.SetWindowText(std::to_wstring(gCurrentKeywordID).c_str());

                            COccurrenceInsert pOccurrenceInsert;
                            if (!pOccurrenceInsert.Execute(pWebSearchEngineDlg-
>m_pConnection, gCurrentWebpageID, gCurrentKeywordID, 1))
                            {
                                pWebSearchEngineDlg->m_pProgress.SetMarquee(FALSE, 30);
                                do {
                                    ::MessageBeep(0xFFFFFFFF);
                                    nRet = pWebSearchEngineDlg-
>m_pConnection.Disconnect();
                                    ::Sleep(60 * 1000);
                                    nRet = pWebSearchEngineDlg-
>m_pConnection.DriverConnect(const_cast<SQLTCHAR*>(reinterpret_cast<const
SQLTCHAR*>(pWebSearchEngineDlg->m_sConnectionInString)), pWebSearchEngineDlg-
>m_sConnectionOutString);
                                } while (!SQL_SUCCEEDED(nRet));
                                if (!pOccurrenceInsert.Execute(pWebSearchEngineDlg-
>m_pConnection, gCurrentWebpageID, gCurrentKeywordID, 1))
                                {
                                    pWebSearchEngineDlg->MessageBox(_T("Cannot insert
occurrence into the database"), _T("Error"), MB_OK);
                                    return false;
                                }
                                pWebSearchEngineDlg->m_pProgress.SetMarquee(TRUE, 30);
                            }
                        }
                        else
                        {
                            const __int64 nKeywordID = gKeywordID[pKeyword];
                            COccurrenceInsert pOccurrenceInsert;
                            if (!pOccurrenceInsert.Execute(pWebSearchEngineDlg-
>m_pConnection, gCurrentWebpageID, nKeywordID, 1))
                            {
                                COccurrenceUpdate pOccurrenceUpdate;
                                if (!pOccurrenceUpdate.Execute(pWebSearchEngineDlg-
>m_pConnection, gCurrentWebpageID, nKeywordID))
                                {
                                    pWebSearchEngineDlg-
>m_pProgress.SetMarquee(FALSE, 30);
                                    do {
                                        ::MessageBeep(0xFFFFFFFF);
                                        nRet = pWebSearchEngineDlg-
>m_pConnection.Disconnect();
                                        ::Sleep(60 * 1000);
                                        nRet = pWebSearchEngineDlg-
>m_pConnection.DriverConnect(const_cast<SQLTCHAR*>(reinterpret_cast<const
SQLTCHAR*>(pWebSearchEngineDlg->m_sConnectionInString)), pWebSearchEngineDlg-
>m_sConnectionOutString);
                                    } while (!SQL_SUCCEEDED(nRet));
```

```
                                        if
(!pOccurrenceUpdate.Execute(pWebSearchEngineDlg->m_pConnection, gCurrentWebpageID,
nKeywordID))
                                        {
                                              pWebSearchEngineDlg->MessageBox(_T("Cannot
update occurrence into the database"), _T("Error"), MB_OK);
                                              return false;
                                        }
                                        pWebSearchEngineDlg->m_pProgress.SetMarquee(TRUE,
30);
                                  }
                            }
                      }

                      already_added = false;
                      for (auto it = gDataMiningTerms.begin(); it !=
gDataMiningTerms.end(); it++)
                      {
                            if (pKeyword.compare(it->c_str()) == 0)
                            {
                                  already_added = true;
                                  break;
                            }
                      }
                      if (!already_added)
                            gDataMiningTerms.push_back(pKeyword);
               }

               if ((gCurrentWebpageID % 1000) == 0)
               {
                      for (auto it = gDataMiningTerms.begin(); it !=
gDataMiningTerms.end(); it++)
                      {
                            strMessage.Format(_T("applying data mining for '%s'..."), it-
>c_str());
                            pWebSearchEngineDlg->m_pCrawling.SetWindowText(strMessage);

                            CDataMiningUpdate pDataMiningUpdate;
                            if (!pDataMiningUpdate.Execute(pWebSearchEngineDlg-
>m_pConnection, *it))
                            {
                                  pWebSearchEngineDlg->m_pProgress.SetMarquee(FALSE, 30);
                                  do {
                                        ::MessageBeep(0xFFFFFFFF);
                                        nRet = pWebSearchEngineDlg-
>m_pConnection.Disconnect();
                                        ::Sleep(60 * 1000);
                                        nRet = pWebSearchEngineDlg-
>m_pConnection.DriverConnect(const_cast<SQLTCHAR*>(reinterpret_cast<const
SQLTCHAR*>(pWebSearchEngineDlg->m_sConnectionInString)), pWebSearchEngineDlg-
>m_sConnectionOutString);
                                  } while (!SQL_SUCCEEDED(nRet));
                                  if (!pDataMiningUpdate.Execute(pWebSearchEngineDlg-
>m_pConnection, *it))
                                  {
                                        pWebSearchEngineDlg->MessageBox(_T("Cannot apply
data mining to the database"), _T("Error"), MB_OK);
                                        return false;
```

```
                                }
                        }
                }
                gDataMiningTerms.clear();
        }
        return true;
    }
    return false;
}
```

9.7. Fișierul WebSearchEngineExt.h

```
/* This file is part of Web Search Engine application developed by Mihai MOGA.

Web Search Engine is free software: you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by the Open
Source Initiative, either version 3 of the License, or any later version.

Web Search Engine is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
details.

You should have received a copy of the GNU General Public License along with
Web Search Engine.  If not, see <http://www.opensource.org/licenses/gpl-3.0.html>*/

#ifndef __WEBSEARCHENGINE_H__
#define __WEBSEARCHENGINE_H__

#pragma once

#include "stdafx.h"
#include "ODBCWrappers.h"
#include "WebSearchEngineDlg.h"

typedef std::vector<std::string> FrontierArray;
typedef std::map<std::string, int> FrontierScore;
typedef std::map<std::wstring, __int64> WebpageIndex;
typedef std::map<std::wstring, __int64> KeywordIndex;
typedef std::vector<std::wstring> KeywordArray;

bool AddURLToFrontier(const std::string& lpszURL);
bool ExtractURLFromFrontier(std::string& lpszURL);
bool DownloadURLToFile(const std::string& lpszURL, std::string& lpszFilename);
bool ProcessHTML(CWebSearchEngineDlg* pWebSearchEngineDlg, const std::string&
lpszFilename, const std::string& lpszURL);

class CGenericStatement // execute one SQL statement; no output returned
{
public:
    // Methods
    BOOLEAN Execute(CODBC::CConnection& pDbConnect, LPCTSTR lpszSQL)
    {
        // Create the statement object
        CODBC::CStatement statement;
        SQLRETURN nRet = statement.Create(pDbConnect);
        ODBC_CHECK_RETURN_FALSE(nRet, statement);
```

```
            // Prepare the statement
#pragma warning(suppress: 26465 26490 26492)
            nRet = statement.Prepare(const_cast<SQLTCHAR*>(reinterpret_cast<const
SQLTCHAR*>(lpszSQL)));
            ODBC_CHECK_RETURN_FALSE(nRet, statement);

            // Execute the statement
            nRet = statement.Execute();
            ODBC_CHECK_RETURN_FALSE(nRet, statement);
            return TRUE;
      }
};

class CWebpageInsertAccessor // sets the data for inserting one row intro WEBPAGE table
{
public:
      // Parameter values
      TCHAR m_lpszURL[MAX_URL_LENGTH];
      TCHAR m_lpszTitle[0x100];
      TCHAR m_lpszContent[0x10000];

#pragma warning(suppress: 26429)
      BEGIN_ODBC_PARAM_MAP(CWebpageInsertAccessor)
            SET_ODBC_PARAM_TYPE(SQL_PARAM_INPUT)
#pragma warning(suppress: 26446 26485 26486 26489)
            ODBC_PARAM_ENTRY(1, m_lpszURL)
            ODBC_PARAM_ENTRY(2, m_lpszTitle)
            ODBC_PARAM_ENTRY(3, m_lpszContent)
      END_ODBC_PARAM_MAP()

      DEFINE_ODBC_COMMAND(CWebpageInsertAccessor, _T("INSERT INTO `webpage` (`url`,
`title`, `content`) VALUES (?, ?, ?);"))

            // You may wish to call this function if you are inserting a record and
wish to
            // initialize all the fields, if you are not going to explicitly set all of
them.
            void ClearRecord() noexcept
      {
            memset(this, 0, sizeof(*this));
      }
};

class CWebpageInsert : public CODBC::CAccessor<CWebpageInsertAccessor> // execute INSERT
statement for WEBPAGE table; no output returned
{
public:
      // Methods
      BOOLEAN Execute(CODBC::CConnection& pDbConnect, const std::wstring& pURL, const
std::wstring& pTitle, const std::wstring& pContent)
      {
            ClearRecord();
            // Create the statement object
            CODBC::CStatement statement;
            SQLRETURN nRet = statement.Create(pDbConnect);
            ODBC_CHECK_RETURN_FALSE(nRet, statement);

            // Prepare the statement
```

```
            nRet = statement.Prepare(GetDefaultCommand());
            ODBC_CHECK_RETURN_FALSE(nRet, statement);

            // Bind the parameters
#pragma warning(suppress: 26485)
            _tcscpy_s(m_lpszURL, _countof(m_lpszURL), pURL.c_str());
            _tcscpy_s(m_lpszTitle, _countof(m_lpszTitle), pTitle.c_str());
            _tcscpy_s(m_lpszContent, _countof(m_lpszContent), pContent.c_str());
            nRet = BindParameters(statement);
            ODBC_CHECK_RETURN_FALSE(nRet, statement);

            // Execute the statement
            nRet = statement.Execute();
            ODBC_CHECK_RETURN_FALSE(nRet, statement);
            return TRUE;
      }
};

class CKeywordInsertAccessor // sets the data for inserting one row intro KEYWORD table
{
public:
      // Parameter values
      TCHAR m_lpszName[0x100];

#pragma warning(suppress: 26429)
      BEGIN_ODBC_PARAM_MAP(CKeywordInsertAccessor)
            SET_ODBC_PARAM_TYPE(SQL_PARAM_INPUT)
#pragma warning(suppress: 26446 26485 26486 26489)
            ODBC_PARAM_ENTRY(1, m_lpszName)
      END_ODBC_PARAM_MAP()

      DEFINE_ODBC_COMMAND(CKeywordInsertAccessor, _T("INSERT INTO `keyword` (`name`)
VALUES (?);"))

            // You may wish to call this function if you are inserting a record and
wish to
            // initialize all the fields, if you are not going to explicitly set all of
them.
            void ClearRecord() noexcept
      {
            memset(this, 0, sizeof(*this));
      }
};

class CKeywordInsert : public CODBC::CAccessor<CKeywordInsertAccessor> // execute INSERT
statement for KEYWORD table; no output returned
{
public:
      // Methods
      BOOLEAN Execute(CODBC::CConnection& pDbConnect, const std::wstring& pKeyword)
      {
            ClearRecord();
            // Create the statement object
            CODBC::CStatement statement;
            SQLRETURN nRet = statement.Create(pDbConnect);
            ODBC_CHECK_RETURN_FALSE(nRet, statement);

            // Prepare the statement
```

```
            nRet = statement.Prepare(GetDefaultCommand());
            ODBC_CHECK_RETURN_FALSE(nRet, statement);

            // Bind the parameters
#pragma warning(suppress: 26485)
            _tcscpy_s(m_lpszName, _countof(m_lpszName), pKeyword.c_str());
            nRet = BindParameters(statement);
            ODBC_CHECK_RETURN_FALSE(nRet, statement);

            // Execute the statement
            nRet = statement.Execute();
            ODBC_CHECK_RETURN_FALSE(nRet, statement);
            return TRUE;
      }
};

class COccurrenceInsertAccessor // sets the data for inserting one row intro OCCURRENCE
table
{
public:
      // Parameter values
      __int64 m_nWebpageID;
      __int64 m_nKeywordID;
      __int64 m_nCounter;
      double m_rPageRank;

#pragma warning(suppress: 26429)
      BEGIN_ODBC_PARAM_MAP(COccurrenceInsertAccessor)
            SET_ODBC_PARAM_TYPE(SQL_PARAM_INPUT)
#pragma warning(suppress: 26446 26485 26486 26489)
            ODBC_PARAM_ENTRY(1, m_nWebpageID)
            ODBC_PARAM_ENTRY(2, m_nKeywordID)
            ODBC_PARAM_ENTRY(3, m_nCounter)
            ODBC_PARAM_ENTRY(4, m_rPageRank)
      END_ODBC_PARAM_MAP()

      DEFINE_ODBC_COMMAND(COccurrenceInsertAccessor, _T("INSERT INTO `occurrence`
(`webpage_id`, `keyword_id`, `counter`, `pagerank`) VALUES (?, ?, ?, ?);"))

            // You may wish to call this function if you are inserting a record and
wish to
            // initialize all the fields, if you are not going to explicitly set all of
them.
            void ClearRecord() noexcept
      {
            memset(this, 0, sizeof(*this));
      }
};

class COccurrenceInsert : public CODBC::CAccessor<COccurrenceInsertAccessor> // execute
INSERT statement for OCCURRENCE table; no output returned
{
public:
      // Methods
      BOOLEAN Execute(CODBC::CConnection& pDbConnect, const __int64& nWebpageID, const
__int64& nKeywordID, const __int64& nCounter)
      {
            ClearRecord();
```

```
            // Create the statement object
            CODBC::CStatement statement;
            SQLRETURN nRet = statement.Create(pDbConnect);
            ODBC_CHECK_RETURN_FALSE(nRet, statement);

            // Prepare the statement
            nRet = statement.Prepare(GetDefaultCommand());
            ODBC_CHECK_RETURN_FALSE(nRet, statement);

            // Bind the parameters
#pragma warning(suppress: 26485)
            m_nWebpageID = nWebpageID;
            m_nKeywordID = nKeywordID;
            m_nCounter = nCounter;
            m_rPageRank = 0.0;
            nRet = BindParameters(statement);
            ODBC_CHECK_RETURN_FALSE(nRet, statement);

            // Execute the statement
            nRet = statement.Execute();
            ODBC_CHECK_RETURN_FALSE(nRet, statement);
            return TRUE;
      }
};

class COccurrenceUpdateAccessor // sets the data for updating one row intro OCCURRENCE
table
{
public:
      // Parameter values
      __int64 m_nWebpageID;
      __int64 m_nKeywordID;

#pragma warning(suppress: 26429)
      BEGIN_ODBC_PARAM_MAP(COccurrenceUpdateAccessor)
            SET_ODBC_PARAM_TYPE(SQL_PARAM_INPUT)
#pragma warning(suppress: 26446 26485 26486 26489)
            ODBC_PARAM_ENTRY(1, m_nWebpageID)
            ODBC_PARAM_ENTRY(2, m_nKeywordID)
      END_ODBC_PARAM_MAP()

      DEFINE_ODBC_COMMAND(COccurrenceUpdateAccessor, _T("UPDATE `occurrence` SET
`counter` = `counter` + 1 WHERE `webpage_id` = ? AND `keyword_id` = ?;"))

            // You may wish to call this function if you are inserting a record and
wish to
            // initialize all the fields, if you are not going to explicitly set all of
them.
            void ClearRecord() noexcept
      {
            memset(this, 0, sizeof(*this));
      }
};

class COccurrenceUpdate : public CODBC::CAccessor<COccurrenceUpdateAccessor> // execute
UPDATE statement for OCCURRENCE table; no output returned
{
public:
```

```
	// Methods
	BOOLEAN Execute(CODBC::CConnection& pDbConnect, const __int64& nWebpageID, const
__int64& nKeywordID)
	{
		ClearRecord();
		// Create the statement object
		CODBC::CStatement statement;
		SQLRETURN nRet = statement.Create(pDbConnect);
		ODBC_CHECK_RETURN_FALSE(nRet, statement);

		// Prepare the statement
		nRet = statement.Prepare(GetDefaultCommand());
		ODBC_CHECK_RETURN_FALSE(nRet, statement);

		// Bind the parameters
#pragma warning(suppress: 26485)
		m_nWebpageID = nWebpageID;
		m_nKeywordID = nKeywordID;
		nRet = BindParameters(statement);
		ODBC_CHECK_RETURN_FALSE(nRet, statement);

		// Execute the statement
		nRet = statement.Execute();
		ODBC_CHECK_RETURN_FALSE(nRet, statement);
		return TRUE;
	}
};

class CDataMiningUpdateAccessor // applying Data Mining
{
public:
	// Parameter values
	TCHAR m_lpszName[0x100];

#pragma warning(suppress: 26429)
	BEGIN_ODBC_PARAM_MAP(CDataMiningUpdateAccessor)
		SET_ODBC_PARAM_TYPE(SQL_PARAM_INPUT)
#pragma warning(suppress: 26446 26485 26486 26489)
		ODBC_PARAM_ENTRY(1, m_lpszName)
	END_ODBC_PARAM_MAP()

	DEFINE_ODBC_COMMAND(CDataMiningUpdateAccessor, _T("UPDATE `occurrence` INNER JOIN
`keyword` USING(`keyword_id`) SET `pagerank` = data_mining(`webpage_id`, `name`) WHERE
`name` = ?;"))

		// You may wish to call this function if you are inserting a record and
wish to
		// initialize all the fields, if you are not going to explicitly set all of
them.
		void ClearRecord() noexcept
	{
		memset(this, 0, sizeof(*this));
	}
};

class CDataMiningUpdate : public CODBC::CAccessor<CDataMiningUpdateAccessor> // execute
UPDATE statement for Data Mining; no output returned
{
```

```
public:
	// Methods
	BOOLEAN Execute(CODBC::CConnection& pDbConnect, const std::wstring& pKeyword)
	{
		ClearRecord();
		// Create the statement object
		CODBC::CStatement statement;
		SQLRETURN nRet = statement.Create(pDbConnect);
		ODBC_CHECK_RETURN_FALSE(nRet, statement);

		// Prepare the statement
		nRet = statement.Prepare(GetDefaultCommand());
		ODBC_CHECK_RETURN_FALSE(nRet, statement);

		// Bind the parameters
#pragma warning(suppress: 26485)
		_tcscpy_s(m_lpszName, _countof(m_lpszName), pKeyword.c_str());
		nRet = BindParameters(statement);
		ODBC_CHECK_RETURN_FALSE(nRet, statement);

		// Execute the statement
		nRet = statement.Execute();
		ODBC_CHECK_RETURN_FALSE(nRet, statement);
		return TRUE;
	}
};

#endif
```