Motoare de căutare
Algoritmi Web Crawling

Profesor coordonator:
Prof. Dr. Ing. Stăncescu STEFAN

Masterand:
Mănăilă George,
Master II - IISC

Contents

1 Introducere:

Un **motor de căutare** este un apelabil program căutător, care accesează Internetul în mod automat și frecvent și care stochează titlul, cuvinte cheie și, parțial, chiar conținutul paginilor web într-o **bază de date**. În momentul în care un utilizator apelează la un motor de căutare pentru a găsi o informație, o anumită frază sau un cuvânt, motorul de căutare se va uita în această bază de date și, în funcție de anumite criterii de prioritate, va crea și afișa o listă de rezultate (hit-list) [1].

Problema nu este una minoră deoarece exista peste 100 de milioane de site-uri web care însumează milioane de pagini web pe tot globul. Faptul că acestea își schimbă conținutul în mod dinamic face dificilă returnarea rezultatelor. Răspunsul la o comandă trebuie sa fie rapid (în mai puțin de o secundă), chiar și atunci când lista de rezultate conține un număr mare de pagini returnate.

Cele mai utilizate motoare de căutare în iulie 2011 au fost:

- Google82,7%
- Yahoo!6,5%
- Baidu 4,7%
- Bing 3,7%
- Ask 0,5%
- Aol 0,4%
- Excite 0,0%

Modul de funcționare al motoarelor de căutare este bazat pe următoarele metode:

1. Chrawling (scotocire)
2. Indexing (indexare de conținut)
3. Searching (căutare)

Pentru crearea unor motoare de căutare este necesar un “robot” de căutare (boot, spider). Acesta este un program SW realizat într-un limbaj care poate fi Perl, Ruby, Java, Php, etc. Linkurile utile sunt extrase prin metode specifice într-o bază de date.

1.1 Mod de funcționare

Instrumentele de căutare folosesc următoarele metode de regăsire a informațiilor: căutarea după cuvinte cheie sau expresii, mecanisme booleene, proximitatea , trunchierea etc. Regăsirea unei resurse folosind adresa (URL = “universal resource locator”) este utilă și rapidă dar există posibilitatea modificării URL-ului datorită caracterului dinamic al Internetului.

Un motor de căutare returnează informați din linkul HTML. O bază de date Web este în esență o copie a fiecărei pagini înregistrată în listă (practic părți mici din acea pagină, cum ar fi titlul, antetul etc).

Crearea acestei liste cu surogate ale paginilor poartă numele de **indexare** și fiecare bază de date web o realizează în stilul său caracteristic. Pentru utilizatorul final, baza de date web funcționează ca o interfață ce are ca și caracteristică fie un "camp" special în care utilizatorul tastează cuvintele după care va efectua căutarea (ex google.ro), fie o lista de "directoare" din care utilizatorul poate alege legătura dorita (ex Yahoo directories).

Figura 1. Captarea datelor [2]

2 Metode de indexare

Există două modalități mai importante de indexare a informației în timpul generării bazelor de date web:

- Indexarea full –text
- Indexarea manuala

Indexarea full-text se caracterizează prin includerea tuturor cuvintelor dintr-o pagină în baza de date fiind disponibile în întregime pentru căutare, cu ajutorul *motoarelor de căutare* (roboți sau "spiders"). Acest mod de indexare permite să regăsim toate referirile la un anumit termen din documentul indexat.

Indexarea "manuală" se realizează cu ajutorul omului, care examinează paginile ce urmează a fi indexate și decide asupra câtorva cuvinte (fraze) cheie ce descriu mai bine informația conținută în respectiva pagină. Permite utilizatorului să regăsească mai multe legături utile în urma căutării, tocmai pentru că un om și nu o mașină a ales cuvintele cheie ce au fost incluse în indexul bazei de date. Această tehnică de indexare este folosită în cazul serviciilor de directoare de pe web (Yahoo directories sau Magellan).

3 Web crawling

Prin web crawling se transferă informațiile din paginile web în baza de date cu scopul de a le indexa după conținut. Obiectivul este de a salva cat mai rapid și eficient informațiile paginilor web accesate.

3.1 Caracteristicile unui "web crawler"

Caracteristicile unui *"web crawler"* se încadrează în două categorii: Caracteristici care trebuie îndeplinite și care ar trebui îndeplinite.

Caracteristici care trebuie îndeplinite:

c: Pe internet se găsesc servere care creează așa zisele capcane "spider traps", care sunt de fapt generatori de pagini web care induc in eroare "crawlere-le" și intră într-un număr infinit de pagini pentru un anumit domeniu. De aceea ele trebuie să fie gândite să reziste acestor tipuri de capcane. Nu toate aceste capcane sunt rău intenționate unele dintre ele sunt pur și simplu *buguri* de la dezvoltarea web-ului.

Politica: Servere de web au atât politici implicite cât și explicite de reglementare referitor la rata de timp la care un *crawler* le pot vizita. Aceste politici trebuie să fie respectate.

Caracteristici care ar trebui îndeplinite

Distributivitate: Un *crawler* ar trebi să aibă abilitatea să ruleze într-un mod elegant pe sisteme cu resurse distribuite.

Scalabilitate: Arhitectura *crawler-ului* ar trebui să permită extinderea ratei de acces prin adăugarea de mașini suplimentare și lățime de bandă.

Performanță și eficiență: Sistemul *crawler-ului* trebuie să folosească eficient un sistem variat de resurse incluzând procesorul, spațiul de pe disc si lățimea de bandă a conexiuni internet.

Calitatea: Având în vedere că o parte semnificativă a tuturor paginilor web nu deservesc nevoile de căutare a utilizatorilor, *crawler* trebuie să fie bazat spre identificarea paginilor utile în primul rând.

Prospețime: În multe aplicații *crawler* trebuie să funcționeze în mod continuu; ar trebui să obțină copii noi ale paginilor preluate anterior. Un motor de căutare crawler, de exemplu poate asigura astfel că indicele motorului de căutare conține o reprezentare destul de curentă a fiecărei pagini web indexate. Pentru asta un crawler trebuie să poată să acceseze o pagină care aproximează rata de schimbare a paginii respective.

Extensibil: Crawler ar trebui să fie conceput pentru a fi extensibil ân multe feluri pentru a face față cu noile formate de date, noile protocoale de preluare și așa mai departe. Asta necesită ca arhitectura să fie modulară.

3.2 Operațiile unui Web Crawler

Obiectivul principal pentru un motor de căutare este de a returna cât mai multe rezultate relevante într-un timp cât mai scurt. Sunt trei operații importante pe care un motor de căutare trebuie să le efectueze.

- Crawling
- Indexind
- Searching

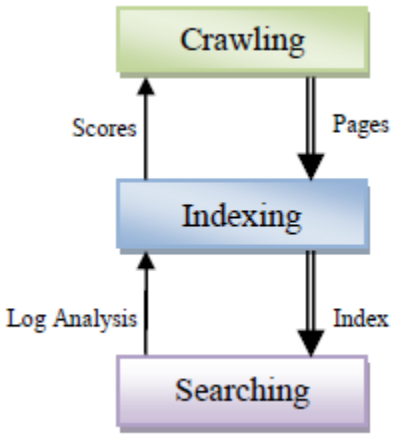

Figura 2. Taskuri pentru web crawler [7]

Un web crawler este folosit în scopuri multiple: descărca pagini web pentru indexare, validarea de pagini web, analiza structurală, vizualizare, notificări, replicare, spam, captarea de adrese email etc.

Dintre acestea scopul principal este acela de a vizita pagini web și de a le descărca într-un mod sistematic pentru motoarele de căutare. Un crawler pornește împreuna cu un URL (Universal Resource Locator), explorează toate hyperlink-urile din acea pagină web, vizitează aceste pagini și le descarcă conținutul. Aceste pagini downloadate sunt indexare și folosite pentru motoarele de căutare. Un motor de căutare este evaluat în funcție de performanțele sale, de calitatea rezultatelor și de abilitarea de a căuta eficient date pe web.

Principalele cerințe de la un web crawler sunt:

- Descărcarea paginilor web.

- Căutarea și identificarea tuturor hyperlink-urilor din pagină.
- Pentru fiecare link identificat, se va repeta procesul.

Pentru a reduce timpul de procesare, unele web crawlere descarcă în paralel paginile web. Odată ce un s-a identificat URL-ul, crawler-ul urmărește toate link-urile pe care le găsește în pagină și le parcurge după un anumit algoritm până ce epuizează lista de link-uri și se înregistrează un mesaj de oprire.

Un crawler parcurge link-urile cub formă arborescentă precum am să arăt în figura de la jos:

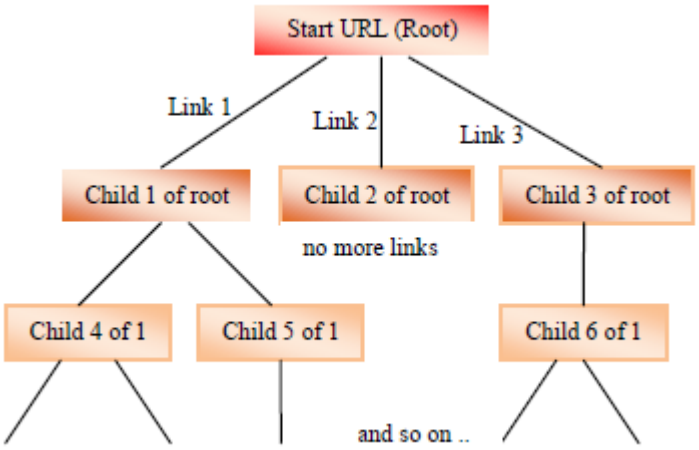

Figura 3. Structura arborescentă de parcurgere a linkurilor.

Aici URL-ul de start este nodul rădăcină al arborelui și copii acestuia sunt link-urile sale. Unii copii au la rândul lor alți copii, deci alte link-uri descendente. Copii care nu au nici un descendent se numesc *copii marionetă*.

3.3 Tipuri de web crawlere

Există cinci tipuri principale de motoare de căutare Acestea sunt:

1. Crawler-Based Search Engines.

Motoarele de căutare bazate pe robot sau păianjeni sunt programe care rulează automat în buclă și caută informații în paginile web, de descarcă, le indexează și le stochează în baza de date. Când un utilizator efectuează o căutare, acesta interoghează baza de date și returnează paginile web relevante pentru cuvântul/cuvintele cheie folosite. Acest tip de program *robot* rulează continu pe internet întreținând baza de date. Exemplele sunt motoarele de căutate Google și Yahoo!

2. Directory Based Search Engines

Un Crawler bazat pe directoare este ajutat de om pentru a sorta paginile descărcate în categoriile de directoare de care aparțin. Se formează astfel această bază de date a directoarelor. Exemplele sunt motoarele de căutare Yahoo! Directory

3. Meta Search Engines

Motoarele de căutare *meta* sunt cunoscute ca și "motoare de căutate multi-thread" Meta Search Enginess nu caută paginile web și nu întreține nici o baza de date. În schimb acestea au rol de agent intermediar pasând interogarea către motoarele de căutare puternice returnând direct rezultatele. Exemplele sunt Dogpile și Vivisimo.

4. Hybrid Search Engines

Motoarele de căutare hibrid folosesc o combinație atât de crawler-based cât și bazat pe directoare. Spre exemplu MSN Search de la Microsoft are un motor de căutare bazat pe directoare populate manual dar este prezent și un crawler care returnează rezultatele.

5. Concept Based Search Engines

Acest crawler încarcă să înțeleagă ceea ce dorești să cauți, nu doar ceea ce cauți efectiv. În circumstanțe favorabile acesta returnează indicii (hint-uri) care au legătură cu subiectul căutat. Acest sistem de hint-uri nu are nevoie de o căutare foarte precisă, el funcționează chiar dacă cuvintele din document nu se încadrează perfect pe texul căutat. Exemple: Excite, Essie Compass.

4 Arhitectura unui crawler

Schema simplificată a unui "robot crowler" necesită mai multe module care să se compună la final.

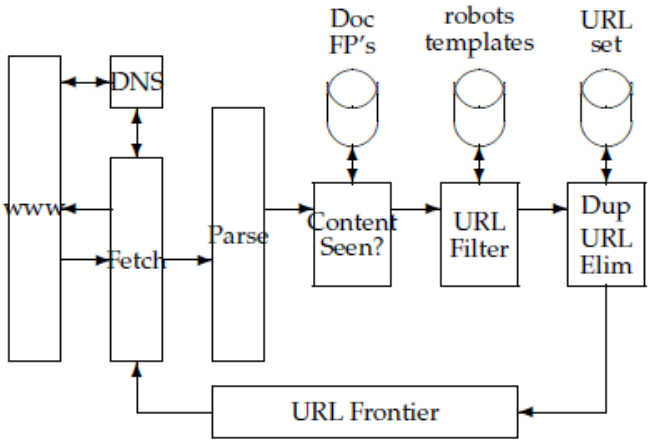

Figura 4. Arhitectura web crawler simplificată [3].

1. Frontiera URL (The URL Frontier) conține adresele URL care urmează să fie preluate (în cazul unui sistem continu, o adresă URL poate fi preluată anterior prin re-fetching)
2. Un modul de rezoluție DNS care determină pagina de la care să încarce.
3. Un modul de încărcare care folosește protocolul http pentru a prelua pagina web la un URL.
4. Un modul de parsare care extrage textul și setul de link-uri de la un web încărcat.
5. Un modul care elimină duplicatele, el determină dacă un link extras există sau nu în lista URL sau dacă a fost încărcat de curând.

Operațiunea se realizează prin unu sau până la sute de thread-uri, fiecare realizând bucle logice. Aceste thread-uri pot fi rulate într-un singur proces, sau să fie împărțite între mai multe procese care rulează la diferite noduri ale unui sistem distribuit. Un thread crowler începe prin preluare unui URL de la frontiera și încărcarea paginii web prin protocolul http. Pagina încărcată este apoi scrisă într-o locație temporară, asupra căruia se efectuează un număr de operații. Urmează ca pagina să fie parsată și textul corespunzător și link-urile să fie extrase. Urmează indexarea informațiilor bazându-se și pe informațiile existente in tag. Fiecare link extras trece printr-o serie de teste pentru a se determina dacă link-ul va fi adăugat în lista de URL-uri.

Prima dată se verifică dacă o pagină web cu același conținut se regăsește în lista sub un alt URL. Cea mai simpla implementare pentru această verificare este să se folosească un *marker* precum un *checksum* (stocat in sistem sub numele de “DocEP’s din Figura 2”). În continuare se folosește un filtru URL pentru a determina dacă adresele extrase trebuie excluse sau nu bazându-se pe testele anterioare. Spre exemplu dacă am dori să excludem un anumit domeniu web (ex: adresele cu .com). În acest caz testul ar trebui sa filtreze adresele URL care corespund domeniului .com. Pentru acesta operațiune exista și un protocol de excludere numit *Robots Exclusion Protocol*. Acest lucru se face prin plasarea unei fișier protocol cu numele *robots.txt* la baza ierarhiei URL.

Aici este un exemplu de fișier *robots.txt* care specifică faptul că robotul nu ar trebui să viziteze adresele URL a căror poziție în ierarhia fișierului începe cu / yoursite / temp /, cu excepția celor denumite "motor de căutare".

```
User-agent: *
Disallow: /yoursite/temp/
User-agent: searchengine
Disallow:
```

5 Algoritmi Web Crawler

Unii dintre algoritmii folosiți de roboții de căutare (*web crawler*) pe care îi voi prezenta sunt următorii:

5.1 Breadth First Search

Cel mai simplă metodă de crawling. *Breadth First Search* folosește frontiera de crawl ca și o coadă FIFO parcurgând linkurile în ordinea în care sunt găsite. Acest algoritm parcurge arborele sau structurile de date arborescente. Pornește de la nodul rădăcină, sau poate porni de la oricare alt nod, uneori un nod referit drept nod cheie și explorează prima dată nodurile vecine, apoi trece la următorul nivel și repetă procesul până când explorează toate nodurile. Se folosește pentru structurile arborescente cu topologia în lățime.

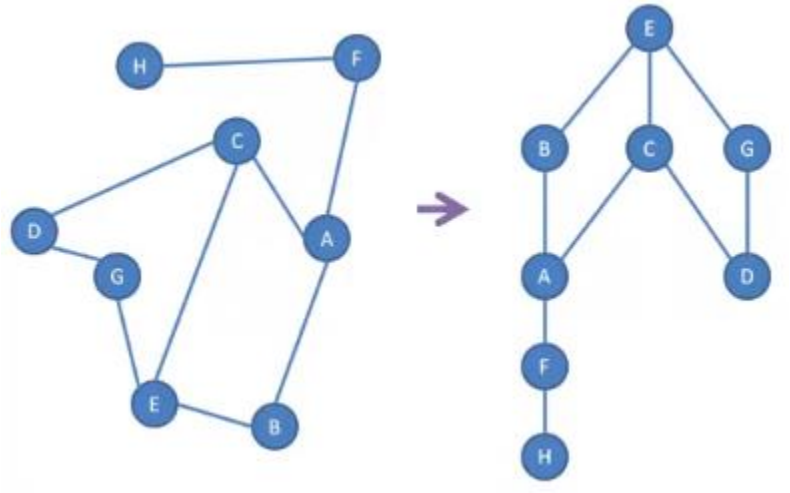

Figura 5. Ordinea de vizitare a nodurilor [4]

În Figura 3 este exemplificată ordinea de parcurgere a nodurilor pornind de la nodul rădăcină se vor explora toate nodurile disponibile.

Algoritmul de funcționare scris în *pseudocod* este scris în figura de mai jos.

```
Breadth-First-Search(Graph, root):
 2
 3      for each node n in Graph:
 4          n.distance = INFINITY
 5          n.parent = NIL
 6
 7      create empty queue Q
 8
 9      root.distance = 0
10      Q.enqueue(root)
11
12      while Q is not empty:
13
14          current = Q.dequeue()
```

```
15
16          for each node n that is adjacent to current:
17              if n.distance == INFINITY:
18                  n.distance = current.distance + 1
19                  n.parent = current
20                  Q.enqueue(n)
```

Acesta este o implementare non recursivă asemănătoare cu cea a algoritmului *Depth-first Search*, de parcurgere în adâncime a grafului și *Breadth First Search* folosește coadă (*buffer*) FIFO în loc de stivă (LIFO). Distanță față de următorul nod este necesară de ex: când se dorește căutarea celui mai scurt drum în graf. De aceea algoritmul, la prima rulare setează distanța cu infinit (`INFINITY`), ceea ce indică faptul că nu s-a descoperit încă nici un nod. Odată ce se găsesc noduri se determină și distanțele față de nodul rădăcină. Valoarea NIL reprezintă șirul vid care semnifică în acest caz absența unui nod părinte (sau nod predecesor).

5.2 Depth First Crawling Algorithm

Acesta este cel mai folositor algoritm atunci când se caută pornind de la rădăcină și se parcurge arborele în adâncime. Dacă există mai mulți copii vom începe cu cel mai din stânga și vom parcurge nivel după nivel în adâncime pană la ultimul copil. Aici se folosește becktraching pentru a ajunge la ultimul nod nevizitat. Este foarte eficient în ceea ce privește căutare în adâncime, dar pentru un arbore cu mulții copii pentru nodul rădăcină devine neeficient și poate intra ușor într-o buclă infinită.

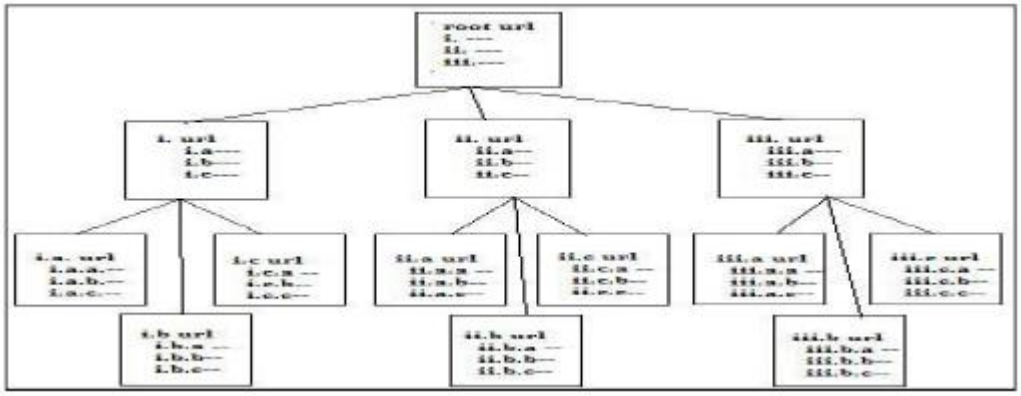

Figura 6. Deep First Crawling

5.3 Page Rank Algorithm

Algoritmul acesta determină importanța paginilor web din oricare website pe baza numărului total de back-links sau de citații pentru pagina web respectivă. Astfel se calculează scorul (rank-ul) paginii web pe baza relevanței paginilor procesare de algoritmul Page Rank. Paginile web al căror număr de linkuri de intrare este mare sunt considerate importante în ceea ce privește legătură cu alte pagini, mai exact gradul de interes al paginii în raport cu altele. Când numărul de link-uri către o pagină este mare gradul de interes este și el mare. Prin urmare, suma totală ponderată a legăturilor de intrare definește algoritmul page rank al unei pagini web. Calculul se face după o formulă de felul acesta:

PR(A) = (1-d) + d (PR(T1)/C(T1) + ... +PR(Tn)/C(Tn)), unde

PR(A) este rankul website-ului, d este factorul de amortizare și T1…Tn legăturile (links)

5.4 Online Page Importance Calculation Algorithm (OPIC)

În acest algoritm pentru a determina importanța paginilor web, fiecare pagină are o valoare unică numerică, care este distribuită în mod egal pentru toate linkurile de ieșire. Inițial toate paginile au aceeași valoare egală 1/n. Crawler-ul va începe să se descarce pagini web cu valori mai mari în fiecare etapă. După fiecare descărcare valorile se vor distribui apoi se va relua procesul. Dezavantajul acestei metode este că fiecare pagină web va fi descărcată de mai multe ori ceea ce face să crească timpul de execuție.

5.5 Crawling the large sites first

În 2005 *Ricardo Baeza Yates* a descoperit o strategie de crawling pentru algoritmul Breadth First în ceea ce privește ordinea de execuție al algoritmului pentru paginilor web. A experimentat pe aproximativ 100 milioane de pagini web și a descoperit că parcurgând prima dată website-urile mari este mult mai folositor față de importantă propriu zisă a unei pagini. Astfel că se caută prima dată în web site-urile mari cu multe pagini în așteptare (panding) pentru a găsi un număr mare de link-uri nedescoperite.

5.6 By the use of filter

Se folosește o abordare bazată pe filtru de date și interogarea datelor. Filtrul redirecționează paginile web reînnoite către crawler, iar acesta descarcă toate paginile reînnoite de la ultima sa vizită.

5.7 Best First Search

Acesta este un algoritm de căutare euristic. Calculează similaritatea dintre pagina și domeniul de interes si calculează un scor pentru URL-urile nevizitate de pe pagina pe baza acesteia. URL-urile sunt apoi adăugate la o frontieră care este menținută ca o coada cu priorități bazate pe scorul obținut.

Acest algoritm explorează graful prin extinderea nodului cel mai promițător ales conform unei reguli specificate. Se folosește o funcție de evaluare f(n) care depinde de descrierea n. Selecția eficientă a celui mai bun nod se face prin folosirea unui buffer de prioritate.

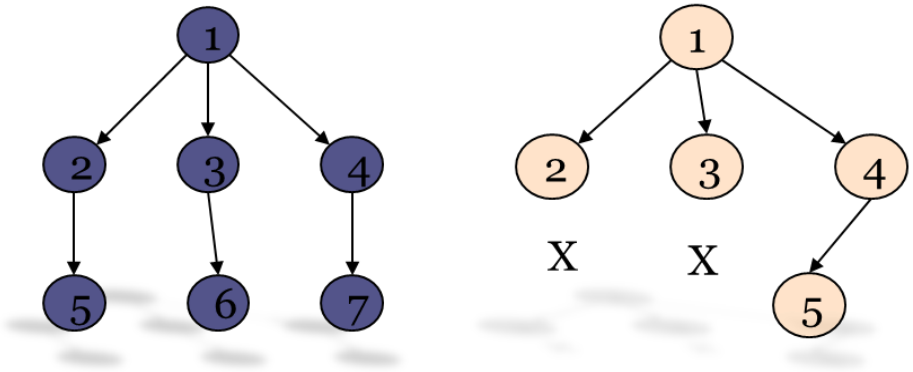

Figura 4. Ordinea de vizitare a nodurilor pentru Best First Search

5.8 Fish Search

Este un algoritm de căutare dinamică euristic. Se lucrează pe intuiția că link-urile relevante au vecini relevanți. Prin urmare se începe cu un link relevant și se parcurge graful în jos, căutarea oprindu-se sub linkurile relevante.

Punctul cheie al algoritmului constă în menținerea ordinii URL.

5.9 A* Search

Acest algoritm folosește căutarea Best Fit. Calculează relevanța fiecărui link și determină diferența între relevanța așteptată a paginii obiectiv și relevanța paginii web curente.

Pseudo codul pentru acest algoritm este prezentat mai jos:

```
/*Start with given Seed URLs as input*/
A_Star_Algo(Initial seed)

/*Insert Seed URLs into the Frontier*/
Insert_Frontier (Initial seed);

/*Crawling Loop*/
While (Frontier! = Empty)

      /*Pick new link from the Frontier*/
      Link: = Remove_Frontier (URL);

      Webpage: = Fetch (Link);

      Repeat For (each child_node of Webpage)

            /*Calculate Relevancy Value till that Page*/
            Rel_val_gn (child_node):= Rel_val(topic, node webpage);

            /*Calculate Relevancy Value from that Node till the Goal Page*/
            Rel_val_hn (child_node):=Rel_val(topic, goal webpage)-
            Rel_val(topic, node webpage);

            /*Calculate Total Relevancy Value of the Path to the Goal Page*/
            Rel_val_fn:= Rel_val_gn+Rel_val_hn;

            /*Add new link with Maximum Relevancy Value into Frontier*/
            Insert_Frontier (child_node_max, Rel_val_max);
End While Loop
```

Pe măsură ce se traversează graful, se formează un arbore cu căi parțiale. Frunzele acestui arbore sunt stocare într-o coadă de prioritate, care ordonează nodurile frunză printr-o funcție de cost. Această funcție combină o estimare euristică a costului.

$f(n) = g(n) + h(n)$

, unde g(n) reprezintă costul de la nodul inițial la nodul n. Valoarea acesta este monitorizată de algoritm. h(n) este un estimat euristic al costului necesar pentru a ajunge de la nodul n la nodul de referință (sau destinație).

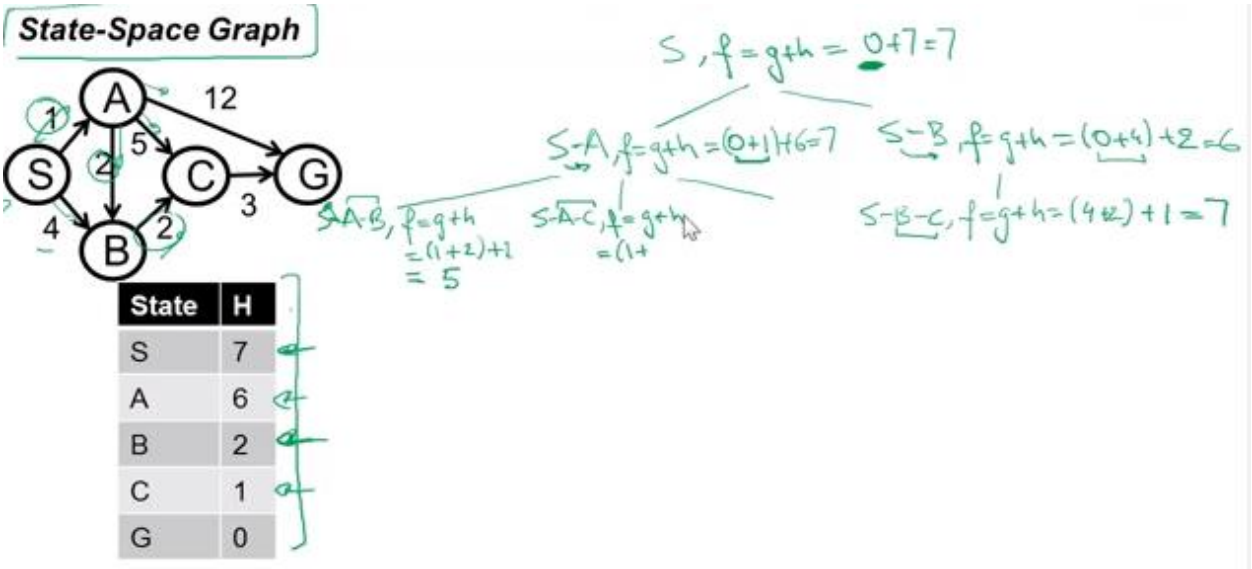

Figura 5.Mod calcul algoritm A*

5.10 Adaptive A* Search

Algoritmul Adaptive A* se bazează pe o funcție euristică adaptabilă pentru căutările sale. Cu fiecare iterație se calculează valoarea relevantă a paginii care se va folosi pentru următoarea traversare. Paginile sunt modificate pentru scară logaritmică

Algoritmul scris în pseudo cod este reprezentat mai jos:

```
/*Start with given initial Seed URL as input*/
Adaptive_A_Star_Algo(Initial seed, Graph Size)

/*No. of times relevancy has to be updated to get better results*/
b: = log(Graph Size);

Repeat For (b times)

      /*Insert Seed URLs into the Frontier*/
      Insert_Frontier (Initial seed);


      /*Crawling Loop*/
      While (Frontier! = Empty)

      /*Pick new link from the Frontier*/
      Link: =Remove_Frontier (URL);

      Webpage: = Fetch (Link);


      Repeat For (each child_node of Webpage)

      /*Calculate Relevancy Valuetill that Page*/
```

```
        Rel_val_gn (child_node):= Rel_val(topic, node webpage);

        /*CalculateRelevancyValuefrom that node till the Goal Page*/
        Rel_val_hn (child_node):=
        Rel_val(topic,goal webpage)-Rel_val(topic, node webpage);

        /*Calculate Total Relevancy Value of the Path to the Goal Page*/
        Rel_val_fn:= Rel_val_gn+Rel_val_hn;

        /*Add new link with Maximum Relevancy Value into Frontier*/
        Insert_Frontier (child_node_max, Rel_val_max);

End While Loop

/*After b times, A* Search more efficient on updated graphs*/
A_Star_Algo(seed URL, Graph (G));
```

6 Concluzii

Acest referat prezintă câteva metode folosite de către motoarele de căutare pentru descărcarea paginilor web in WWW. Toți algoritmii prezentați sunt eficienți în ceea ce privește căutarea paginilor web, cu mici avantaje și dezavantaje.

Eficiența crawler-ului se determină în funcție de precizie și rata de achiziție.

Precizie = găsite / (găsite+nefolositoare)

Rata achiziție = găsite / (nr total documente)

7 Bibliografie:

- [1] "Motor de căutare " https://ro.wikipedia.org/wiki/Motor_de_căutare
- [2] Asis Panda "Google search algorithm & SEOs "
- [3] Cambridge UP "Web crawling and indexes " 2009
- [4] "Breadth-first search" https://en.wikipedia.org/wiki/Breadth-first_search
- [5] "Best-first search" https://en.wikipedia.org/wiki/Best-first_search
- [6] Aviral Nigam "Web Crawling Algorithms"
- [7] K. Lengh Goh, Ashutosh K Singh "PyBot: An Algorithm for Web Crawling", Decembrie 2011
- [8] Rahul kumar, Anurag Jain și Chetan Agrawal "SURVEY OF WEB CRAWLING ALGORITHMS", Vol. 1, No.2/3,Septembrie 2014