УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Анализа линкова и алгоритам *PageRank*

МАСТЕР РАД

МЕНТОР:
проф. др Миодраг Живковић

СТУДЕНТ:
Душан Цемовић

БЕОГРАД, 2017.

Садржај

1 Увод

Свакодневни корисници интернета и професионални веб програмери често се питају зашто и како су неки сајтови први на *Google* листи, а зашто неки нису.

Приликом израде сајтова, један од кључних захтева клијената је да њихова интернет страница буде видљива на вебу. Данас је та тема изузетно атрактивна и постоји много радова на ту тему. Једна од главних грана маркетинга бави се баш тим проблемом.

Како је ова област привлачила све више пажње, тако је све теже било и високо позиционирање. Алати за претрагу и рангирање фаворизују веб странице чији програмерски код садржи структуру организовану на такав начин да буде лакши за претрагу, тј. да садржи посебне[1] атрибуте на елементима који су најважнији (слике, линкови, наслови). Поред структуре, битно је да веб страница садржи све пожељне *html* елементе, као што су *„h“ tag*-ови, атрибути у *head* тагу и слично. Број фактора који утичу на рангирање страница из године у годину расте (преко 200 фактора). Последњих година, све више пажње привлачи и машинско учење које постаје саставни део рангирања.

Глобална светска мрежа „*World Wide Web*“ је једна од ствари које су обележиле двадесет први век. Незамислив је један дан а да се не користи интернет у било које сврхе. Поред друштвених мрежа које се масовно користе, људи приликом решавања неког проблема (ако је потребна нека свежа информација, како нешто скувати, направити, написати) најпре користе информације које се могу наћи на интернету. Не постоји проблем за који се не може да нађи (адекватно) решење.

Интернет полако прелази из статуса „корисно“ у статус „обавезно“, што даље имплицира да функционисање претраге информација треба бити све једноставније. Модеран свет приморава кориснике да све брже решавају проблеме чиме се и захтева веће коришћење интернета.

До ваљаних информација долази се помоћу алата за претрагу, који „разврставају“ велики број информација на вебу. Алати добијају упит записан на људском језику, преводе га у запис који је подложан претрази и враћају резултате кориснику.

У мору различитих претраживача и великој конкуренцији, глобални претраживач *Google* донео је иновативни алгоритам претраге *PageRank*, који га је учинио једним од најпопуларнијих.

[1] *Itemprop* је глобални атрибут који се састоји од пара (*html* елемент, вредност) којим претраживач лакше индексира стране.

Концепт алгоритма је осмишљен да буде лако проширив, што омогућује алгоритму да се избори са експоненцијалним растом броја корисника, веб страница и упита.

Google је место где се појављују злонамерни кориници, покушавајући да рангирају свој сајт „нерегуларним“ методама. Међутим, алгоритам је робустан и прилагодив, тј. лако се врше побољшања у одређеним ситуацијама, чиме *Google* успешно излази на крај са злонамерним корисницима.

У раду је алгоритам *PageRank* реализован у оквиру софтвера за анализу линкова. При томе је скуп веб страница представљен усмереним графом. У оквиру програма реализована су побољшања алгоритма која решавају могуће аномалије таквог графа, као што су чворови који немају излазну грану или графови који садрже циклус.

За уводни део користе се књиге [1, 2, 8] са списка литературе.

1.1 Алати за претрагу

Претраживање представља можда и најважнији део интернета јер би без претраге коришћење информација било практично немогуће. Паралелно са развојем самог интернета, развијали су се и алати за претрагу који омогућују да се добију информације које постоје на светској мрежи.

Први алат за претрагу се појавио 1990. и звао се *Archie*. Алат је имао могућност да индексира само наслове страница без прегледа икаквог садржаја на страници јер је проблем био у расположивом простору.

Појавом претраживача *AltaVista* (1994) први пут је постојала могућност да се упише упит (*query*) на људском језику. У тој години појавио се и претраживач *Yahoo*.

Један од најпознатијих претраживача *Google* је основан 1998. године од стране Ларија Пејџа (*Larry Page*) и Сергеја Брина (*Sergey Brin*). Име *Google* потиче од имена „*googol*“, што значи 10^{100}. Иницијално је започет 1996. као пројекат на Универзитету Станфорд у Калифорнији. Претрага је користила тада иновативан приступ рангирања сајтова по популарности - алгоритам *PageRank*[2]. У међувремену, *Google* претрага је постајала све паметнија и преузела примат од алата *Yahoo*. Оно што је додатно интересатно за *Google* у поређењу са другим алатима за претрагу је што је *Google* увек форсирао добро организован и квалитетан код (квалитетна структура *html* тагова, која није видљива крајњем

[2] Име алгоритма PageRank потиче од аутора Ларија Пејџа (*Larry Page*).

кориснику) испред било какве рекламе и плаћања (Програмерска етика), што се испоставило као врло важно јер данас се много улаже у *SEO* (објашњено у одељку 2.3) и системе који имају квалитетан и добро организован код.

Microsoft-ов алат за претрагу *Bing* је такође један од алата који води битку за најактивнији алат. Основан је 2009. као поновно брендирање некада познатог *MSN/live*.

1.2 Проблем

Упити који се данас користе су веома разнолики. Од „колико је данас степени?“ до „где да нађем најближу продавницу која продаје поједине ствари?“. Поставља се питање како претрага рангира резултате упита с обзиром да је типично да број пронађених страница јако велики. Оно што се данас тражи је најбржа и најквалитетнија информација. Претпоставља се да квалитетнији извор информација (сајт) чешће даје ваљане информације.

Ако данас постоје информације од великог броја различитих извора, како се зна да су оне тачне и како ће се прво излистати странице која су имају квалитетнији садржај?

2 Алгоритми претраге и оптимизација

Веб је омогућио невероватно брзу дистрибуцију информација. Омогућио је људима да брзо дођу до неке жељене информације. За разлику од нпр. библиотеке која је организована по неким категоријама и правилима, веб је „самоорганизован“, тј. свако може да постави страницу и да у њу стави линкове ка било којој другој. Уз могућност да сви објављују информације на разне начине, јавио се проблем ваљаних информација. Веб нема стандардну или шаблонску структуру, већ је организован хаотично (једна страница показује на другу страницу). Таква организација омогућила је да се на лак и јефтин начин заради преко веба. Као логична последица, јавили су се и спамери (*spammers)* који су масовно слали рекламни материјал на мејл (*e-mail*) адресе. Поред слања рекламног материјала, спамери су користили све могуће начине да свој сајт учине видљивијим другим корисницима.

Због свега овога се јавила потреба да се рангирају линкови тако да кориснику буду приказане релевантније информације. За обраду овог поглавља користе се књиге [1,2,4,6].

2.1 Први алгоритми претраге

Алгоритми коју су се први појавили су радили по принципу „*Crawling* [3]“ и листања термова (тј. речи или неког стринга без размака) у инвертованом индексу. Инвертован индекс је тип структуре који показује где се терм налази на страници. Када се пошаље упит (листа термова), страница са тим речима екстрактује се из инвертованог индекса и рангира их на одређени начин за ту страницу. Број појављивања у заглављу (по правилу релевантније), број појављивања у садржају као и друга места у *html* страници, утичу на ранг неке странице по упиту.

Лоша страна оваквог приступа је што рангирање страница зависи од садржаја који се налази на страници коју тај корисник поседује. Другим речима, неко лако може да обмане кориснике и да додаје неку одређену реч на страницу да би сајт имао већи ранг ако неки корисник пошаље упит са баш том речју. Нека, на пример, корисник тражи реч „кошарка“. Сајт за неку промоцију (углавном су то сајтови који продају рекламни простор) има неки тотално другачији садржај од онога што се очекује под темом „кошарка“, али има огроман број пута додату реч „кошарка“

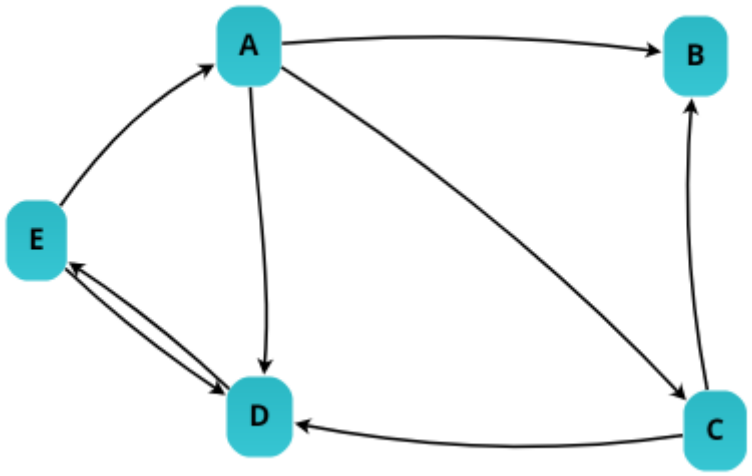
[3] *Crawling* – пузање, процес претраге сајта који је често описан у почетном фолдеру домена (*root*). Корисник који поседује сајт дефинише које странице претрага треба да индексира а које не.

која је обојена у исту боју као позадина странице, па корисник који долази на ту страницу преко упита, не види реч „кошарка“.

2.2 Алгоритми претраге по популарности

Као што је већ речено, круцијални проблем раних алгоритама је што рангирање зависи искључиво од информација које се баш налазе на сајту, па је лако манипулисати рангом. Идеја алгоритма претраге по популарности је да рангирање више зависи од оних који показују на сајт јер тим информацијама корисник, чији се сајт рангира, не може манипулисати. Према томе, ранг неке странице треба да буде велики ако постоји велики број линкова који показују на ту страницу.

Године 1998. независно су се појавила два алгоритма која функционишу по овом принципу. На Универзитету Станфорд развијен је алгоритам *PageRank*, а у Силиконској долини алгоритам *Hypertext Induced Topic Search* (*HITS*). Оба алгоритма представљају веб странице као чворове усмерених графова (слика 1). Гране графа су линкови који се налазе на једној страници, а показују на другу страницу.



Слика 1 – Пример усмереног графа који представља пет веб страница

2.3 *SEO* (*Search engine optimization*)

Оптимизација сајта за интернет претрагу и процес унапређења квалитета и количине саобраћаја од претраживача ка веб-сајту преко природних (нерекламних) резултата претраге, представља једну од најатрактивнијих стратегија подизања ранга веб сајта. Процес оптимизације се ослања на низ правила које треба бити испоштоване да би се повећао ранг веб странице.

Нека од правила оптимизације су:

- Наслов странице (*title tag*) би требало да садржи речи које ће се наћи у упиту претраживача
- Попуњавање тагова (*description tag*) који представљају кратак опис странице
- Формирање адреса страница (*URL*) која одговарају функцији саме странице (пракса је да су сличног садржаја као и наслов странице)
- Израда мапе сајта, као и интуитивне организације страница по категоријама
- Оптимизација слика, тј. смањивање величине фајлова
- Прилагођавање странице за мобилне уређаје редирекцијом на станице специјалне за мобилне уређаје које одговарају иницијалним страницама или прилагођавање исте странице тако да буду видљиве на мобилним уређајима (*fully-responsive*)

Поред наведених правила, модеран сајт би требало да оставља трагове на различитим друшвеним мрежама (*facebook*, *twitter*, *instagram*...). На сајту се остављају линкови ка друштвеним мрежама, а на друштвеним мрежама се остављају линкови ка сајту. Препоручује се и израда блога као пратећи низ информација које су неким делом повезане са темом самог сајта.

3 *PageRank*

Студенти постдипломских студија на Универзитету Станфорд Сергеј Брин и Лари Пејџ увидели су потенцијал графова као презентацију највеће светске мреже, те као пројекат на факултету су развили алгоритам *PageRank*. Напустили су факултет и пројекат су интегрисали у *Google*. Идеју за алгоритам су презентовали као врсту препоручивања између сајтова. Један он најпознатијих примера је „Препорука од Доналда Трампа“. Наиме, ако вас препоручи неко као што је Доналд Трамп, та препорука сигурно има већу тежину него ако вас препоручи неко непознат. Са друге стране, Доналд Трамп је написао велики број препорука у свом животу па су оне често губиле на значају. Дакле веб страница је важна ако на њу показују друге важне веб странице.

PageRank је алгоритам који није зависан од упита (*query-independence*) јер се алгоритам не рачуна приликом упита већ се рачуна у одређеним временским интервалима независно од упита. По правилу, *Google* пушта своје истраживаче (*crawler*) периодично да обраде све сајтове. Дужина периода између два претраживања често зависе и од важности сајта. Поред тога, постоји начин да корисник који поседује сајт „регуларно“ захтева да се поново прегледа сајт.

За алгоритам *PageRank* се користи модел „случајног сурфера“ (*random surfer*) који насумице бира линкове и одлази на друге странице. Последица принципа је да чворови графа имају стабилне вероватноће са којима се сурфер задржава у њима након одређеног броја итерација. Овај процес се итерира више пута док се расподела вероватноћа страница у следећој итерацији не разликује много од претходне.

Кад би правило било да је најважније имати само више линкова ка неком сајту и да се на тај начин рачуна ранг, тада би било могуће да спамери направе фарму спама. Фарма спама је велики број страница са линковима који показују на неку „кључну страницу“. На тај начин се нарушава регуларно рангирање.

Мотивација да се користи принцип „важне странице“ су:

- Корисници веба чешће стављају на своју страницу важне и корисне линкове него непотребне и небитне.
- Корисници веба чешће користе важне и корисне странице него непотребне и небитне.

Приликом *Google* претраге, страница треба да има барем једну од речи које је корисник унео у претрагу (типично је да на страници морају бити све речи које се траже). Ако постоји барем једна реч онда се ранг рачуна као комбинација *PageRank-a* и других параметара као што су присуство речи на неким важним деловима на страници, нпр. *meta* тагу, заглављу и сл. Одређени проценат главног ранга претраживача чини *PageRank,* док остатак чине остали поменути параметри.

У овом одељку дефинисан је алгоритам, могући напади на алгоритам као и побољшања којима се ублажавају напади.

Алгоритам се добро понаша када се количина улазних података повећава јер постоје методи за повећање ефикасности. *PageRank* је настао исте године кад и алгоритам *HITS* који много лакше решава аномалије на графовима али се проблеми при повећавању количине улазних података много више повећавају него код алгоритма *PageRank*.

За обраду овог поглавља користе се књиге [1,2,4,5,7]. За обраду Марковљевих ланаца користе се књиге [2,3].

3.1 Марковљеви ланци

Посматрањем случајног сурфера, постављају се питања „ако је сурфер на некој страници, која је вероватноћа да ће сурфер после неког времена (броја понављања) бити поново на истој тој страници?“. Да ли сурфер са одређене странице може да дође до друге странице?

Сурфер на неку страницу X_2 може да дође само са странице X_1 са које има линк (грану) ка станици X_2. Са које странице је случајни суфер претходно дошао на X_1 уопште не утиче на даљи прелазак на страницу X_2. Случајни процеси који задовољавају то својство, добили су име по руском математичару Андрају Андрејевичу Маркову[4], Марковљеви ланци.

Дефиниција:

Ланац Маркова је случајан процес $\{X(t), t \in T\}$ код кога за свако $t \in T$, такво да $X(t)$: $\Omega \rightarrow S$, где је Ω простор вероватноћа, а S коначан или пребројив скуп стања, важи:
за свако $n \in N, t_1, t_2, \dots, t_{n+1} \in T$ где је $t_1 < t_2 < \cdots < t_{n+1}$ и било које $a_1, a_2, \dots$, $a_{n+1} \in S$ важи следеће:

$$P(X(t_{n+1}) = a_{n+1} | X(t_n) = a_n, \dots, X(t_1) = a_1) = \mathrm{P}(X(t_{n+1}) = a_{n+1} | X(t_n) = a_n)$$

[4] rus. Андрей Андреевич Марков, 14. jun 1856 — 20. jul 1922

Марковљев ланац је према томе описан матрицом реда $k = |S|$, чији је елемент (i, j) једнак вероватноћи $x_{ij} = \mathrm{P}\big(X(t_{n+1}) = a_j \big| X(t_n) = a_i\big)$. Параметар t најчешће означава време, односно t представља тренутак у којем се налази процес. T се зове скуп индекса и подскуп је скупа реалних бројева. У зависности од тога да ли је дискретан или непрекидан скуп , разликујемо процесе Маркова са дискретним и непрекидним временом. Ако је скуп индекса T коначан или пребројив, онда се користи ланац Маркова са дискретним временом.

Особине Марковљевих ланаца:

Ергодичност – ланац Маркова је ергодичан ако је могуће прећи из било код стања у било које друго стање (у једној или више итерација).

Регуларност – ланац Маркова је регуларан ако неки степен матрице има само позитивне елементе, тј. ако за неко n, могуће је прећи из било ког стања у било које друго у тачно n корака.

Важи тврђење да ако је ланац Маркова регуларан, онда је он и ергодичан.

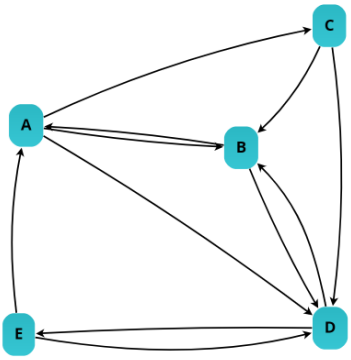
3.2 Дефиниција *PageRank-a*

PageRank је функција која сваком чвору, тј. страници додељује неки позитиван број. Што чвор има већи број, то је страница важнија.

Формира се матрица M која димензија $n \cdot n$, где n је број страница. Ако постоји линк који повезује два чвора, нпр. A и B (при чему су редни бројеви чворова A и B редом i и j), онда је елемент матрице M_{ij} већи од нуле. Његова вредност се дефинише као $1/z$, где је z број линкова који постоји на страници A. Пример матрице и одговарајућег графа се налази на слици 2.

Матрицу M одређује усмерени граф G. Може се претпоставити да је граф G јако повезан, тј. да из сваког чвора може да се дође до сваког другог чвора и да не постоје висећи чворови (*dangling nodes*), тј. чворови из којих не излази ни једна грана.

$$M = \begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 0 & 1/2 \\ 1/2 & 0 & 0 & 1/2 & 0 \end{bmatrix}$$



Слика 2 – Пример матрице и одговарајућег графа

Може се приметити да су збирови елемената у свакој врсти једнаки 1, тј. матрица M је стохастичка. Квадратна мартица је стохастичка матрица ако је збир елемената сваке њене врсте једнак 1. Висећем чвору би одговарала врста чији су сву елементи нуле.

Пошто се користи принцип слободног сурфера, иницијална вероватноћа да се сурфер налази на неком чвору једнака је $1/n$, где је n укупан број чворова у графу. Нека је v_0 вектор врста дужине n чији су елементи почетне вероватноће страница, тј. $1/n$. У свакој итерацији v_0 множи се са матрицом M. После прве итерације вектор вероватноћа x је v_0M, друге итерације вектор је v_0M^2, итд.

Уопште, ако је v вектор расподеле вероватноћа налажења сурфера у некој итерацији (v_j је вероватноћа да се он налази на страници са редним бројем j), а x вектор расподеле вероватноћа налажења сурфера у наредној итерацији (са елементима $x_j, j = 1,2, \dots, n$), онда је:

$$x_j = \sum_{i=1}^{k} v_i m_{ij} \ , j = 1,2, \dots, n$$

где је m_{ij} вероватоћа да сурфер са адресе i пређе на адресу j, а v_j је вероватноћа да је случајни сурфер претходно био на адреси j. Ово представља пример Марковљевог ланца.

Итерације се извршавају док не се не добије вектор вероватноћа x који се „мало“ разликује у односу на вектор v из претходне итерације.

$$v_{n+1} = v_n \lambda M$$

где је граница v сопствени вектор матрице M, а λ је сопствена вредност матрице M једнака „проближно“ 1. У пракси на вебу, потребно је 50-75 итерација да би се добила прецизност на две децимале.

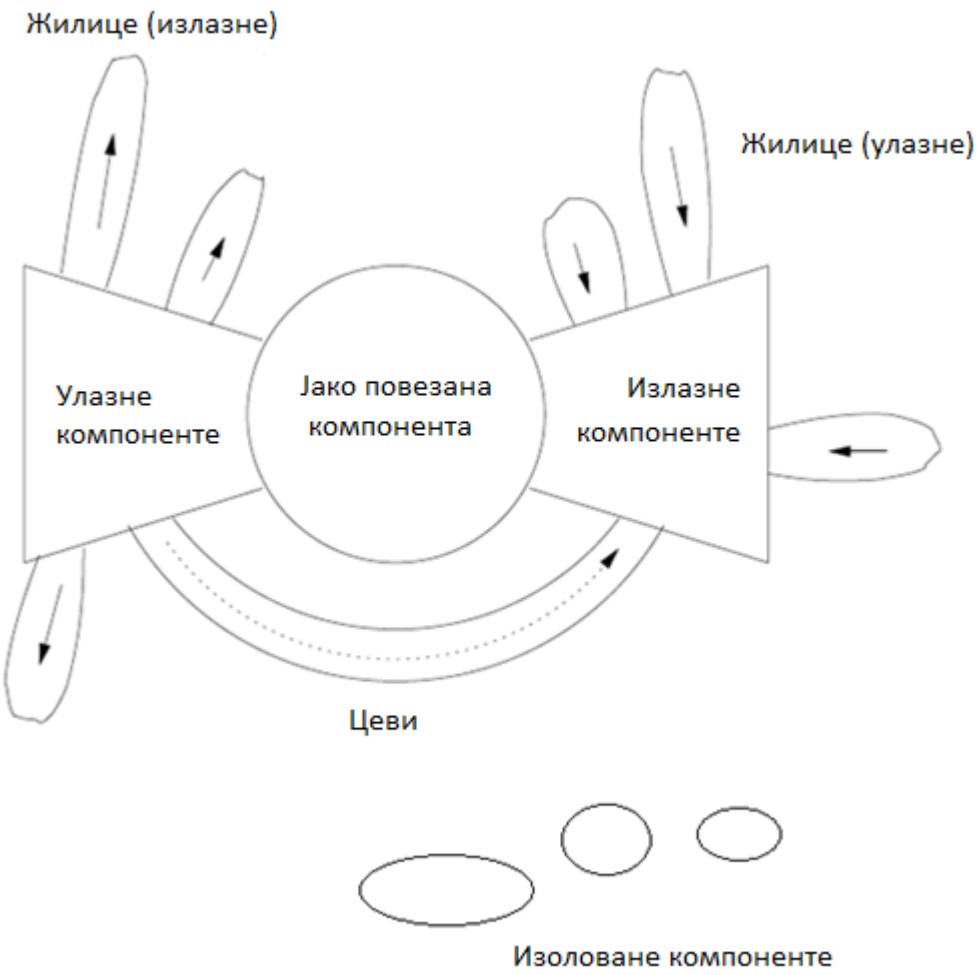
Резултат алгоритма *Pagerank* након 50 итерација , графа са слике 2, износи:

$$Pagerank(A, B, C, D, E) = (0.19, 0.23, 0.06, 0.28, 0.14)$$

Поставља се питање зашто се не користи Гаусов алгоритам за решавање система линеарних једначина који је мање сложености $O(n^3)$ уместо приближног итеративног метода код кога је сложеност само једне итерације множења матрице и вектора $O(n^2)$. Наиме, у пракси матрица веба има велики број нула, тј. велики је број чворова који нису повезани и матрица је ретка. Број линкова на страници је у просеку око 10, па је сложеност једне итерације множења матрице и вектора $O(n)$, што имплицира да је итеративни метод много ефикаснији од Гаусовог метода.

3.2.1 Структура веба

Основна верзија *PageRank*, односи се на графове коју су јако повезани, а данашњи веб садржи много одступања од таквог графа. Међутим, данашњи веб може да се подели на делове тако да један од њих (који садржи највећи број страница) буде јако повезан. За статус осталих страница постоји неколико могућности чија решења се посебно решавају.



Слика 3 – Структура веба

Подела веб компоненти уз приказ повезаности (слика 3):

1. **Јако повезана компонента (ЈПК)** – део веба који може да се класификује као јако повезани граф
2. **Улазне компоменте** – део веба који садржи линкове ка ЈПК али не постоје линкови у ЈПК који воде ка улазним компонентама
3. **Излазне компоменте** – део веба до којих постоје линкови из ЈПК али не постоји линк ка ЈПК
4. **„Жилице“** – део улазних компоменти из којих може да се изађе из основног система или део ван основног система које воде у излазне компоменте.
5. **Цеви** – део улазних компоменти из којих је могуће да се директно дође излазних компоменти, тј. да се не улази се у ЈПК
6. **Изоловане компоненте** – део ван основног система у који није могуће доћи из основног система нити је из тих делова могуће доћи у основни систем

Неки од ових делова нарушавају концепт међусобне повезаности и из неких делова не можемо да стигнемо до других. Ако случајни сурфер крене од улазних компоменти и крене кроз „жилице“ ван основног система, више не може да дође до ЈПК и тако ће нарушити рангове страница из ЈПК.

Претпоставља се да странице у ЈПК имају већи ранг. У случају да случајни сурфер пређе из ЈПК у излазну компоненту, неће више моћи да се врати у ЈПК, па ће на тај начин смањити ранг страница које се налазе у ЈПК, а повећати ранг страницама у излазној компоненти. Додатно, ако страница у излазној компоненти нема линкове а постоје линкови до ње, вероватноћа да ће случајни сурфер бити на њој је велика па на тај начин се нарушава рангирање страница алгоритмом *PageRank*. Чвор из кога не излази ни једна грана, зове се Висећи чвор (*dagling nodes*).

У ЈПК се може десити да нека страница направи линк ка самој себи или да мали број страница направи подциклус. У тим случајевима повећава се међусобни ранг, што такође нарушава рангирање страница и та аномалија се зове Паукова замка (*spider trap*).

Да би се пронашло решење за овакве специфичне ситуације, извршавају се промене графа на који се примењује алгоритам *PageRank*.

3.2.2 Висећи чворови (*dagling nodes*)

Усмерени граф који има висећи чвор није стохастичан, већ има чвор који је висећи (део излазних компоненти). Ако се рачуна ранг таквог графа, после одређеног броја итерација, за сваки чвор вероватноћа да се случајни сурфер налази баш на том чвору једнака је 0.

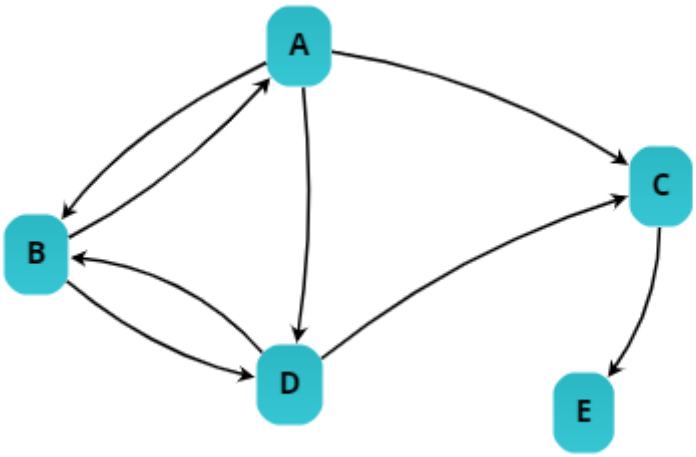
На пример, за матрицу:

$$M = \begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

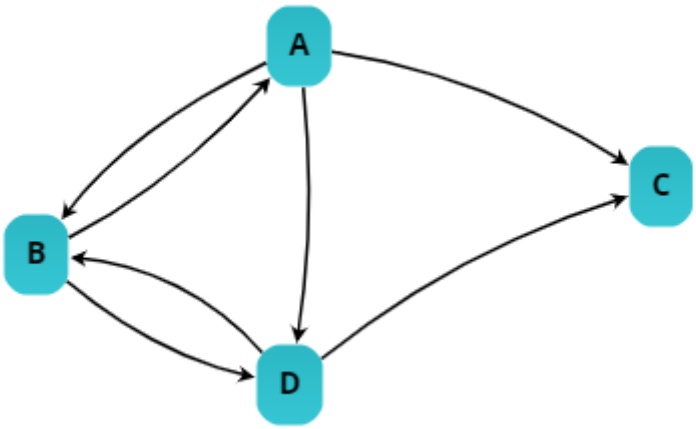
Алгоритам *Pagerank* би дао резултат:

$$Pagerank(A, B, C, D, E) = (0, 0, 0, 0, 0)$$

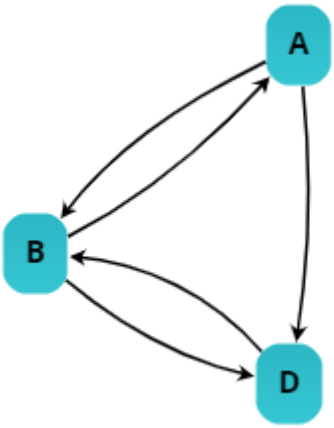
У случају да у нестохастичном графу постоје редови чији збир јесте 1, онда је подграф стохастичан. Уклањањем висећих чворова (што се у општем случају мора радити у неколико итерација) граф се може свести на неки стохастички граф за који може да се израчуна ранг. Приликом брисања висећег чвора, постоји вероватноћа да се дође у ситуацију да граф садржи нови висећи чвор чија грана се избрисала уклањањем претходног чвора. Чворови се уклањају док се не добије један или више јако повезани граф.



Слика 4 – Пример графа (који одговара матрици 3.2.2) са висећим чворовима



Слика 5 – Редукован пример графа који и даље има висећи чвор



Слика 6 – Редукован граф који нема висећи чвор

Поступак се може илустровати на наведеној матрици M. Редукција графа приказана је на сликама 4, 5 и 6. Добија се матрица редукованог графа:

$$M' = \begin{bmatrix} 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 \\ 0 & 1 & 0 \end{bmatrix}$$

Када се добије јак граф, рачуна се *PageRank* тог графа.

$$PageRank(A, B, D) = (2/9, 4/9, 3/9)$$

Након тога, рекурзивно се враћају чворови који су обрисани. Рангови чворова који су обрисани рачунају се на основу рангова чворова за које постоје вредности. Вредност (*PageRank*) се рачуна тако што се саберу вредности чворова из којих се долази до враћеног чвора подељених са бројем линкова које тај чвор (из ког се долази) има.

$$PageRank(C) = 1/3 \cdot PageRank(A) + 1/2 \cdot PageRank(D),$$

$$PageRank(E) = 1/1 \cdot PageRank(C)$$

На овај начин губи се својство слободног сурфера, али се добијају приближне вредности за ранг (*PageRank*) страница.

Осим овог начина, решавање висећих чворова могуће је извршити и „опорезивањем“ којом се решава и друга позната „аномалија“ графа звана „паукова замка“.

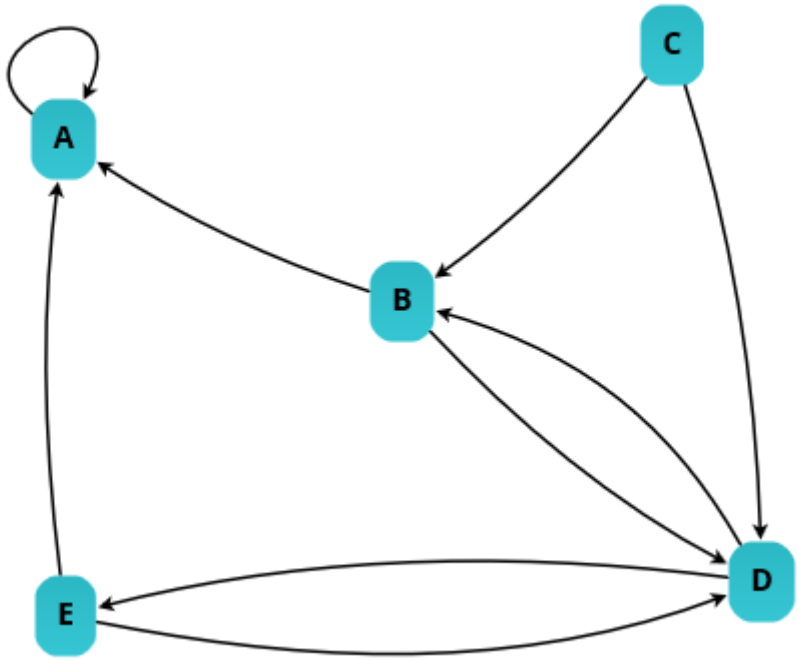
3.2.3 Паукова замка *(spider trap)*

Граф може имати чворове такве да из сваког постоји линк, али може имати линк који води до самог себе, односно може да постоји група страница које са својим линковима формирају циклус. На тај начин случајни сурфер може доћи до групе страница одакле не може да изађе и погрешно ће се рачунати ранг.

PageRank графа са слике 1 (после 50 итерација) износи:

$$PageRank(A, B, C, D, E) = (0.19, 0.23, 0.06, 0.28, 0.14)$$

Ако се граф измени, тако што се избришу гране из A и направи један чвор са граном ка самом себи, добија се граф:



Слика 7 – Пример графа са пауковом мрежом

После одређеног броја итерација добија се вектор који неће омогућити слободном сурферу да напусти „паукову замку“.

$$PageRank(A, B, C, D, E) = (1, 0, 0, 0, 0)$$

Да би се омогућила обрада оваквих графова уводи се позитивна вероватноћа да случајни сурфер пређе на било коју страницу у графу. Претпоставља се да он са

једнаком вероватноћом може да пређе на било коју страницу, без услова да постоји грана ка неком чвору.

Приликом сваке итерације производ матрице и вектора множи се са вероватноћом β (којом ће случајни сурфер да настави кретњу регуларним гранама) и додаје се вектор који је једнак јединичном вектору подељеним бројем чворова (уз неку вероватноћу, $1 - \beta$).

$$v' = v\beta M + (1 - \beta)e/n$$

Овде је β вероватноћа да ће случајни сурфер да настави линковима које постоје на страници, па први сабирак предстаља регуларни део алгоритма. Распон вероватноће је од 0,8 до 0,9. У супротном случајни сурфер користи могућност телепортовања на произвољну страницу. Овим методом такође се решава проблем „висећег чвора“.

Ако се за дати граф уведе вероватноћа за нетелепортовање β = 0,8 добија се

$$v' = v \cdot 0.8 \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 0 & 1/2 \\ 1/2 & 0 & 0 & 1/2 & 0 \end{bmatrix} + 0.2 \cdot \begin{bmatrix} 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \end{bmatrix}$$

Након одређеног броја итерација добија се вектор:

$$PageRank(A, B, C, D, E) = (0.61, 0.11, 0.04, 0.14, 0.09)$$

И даље је чвор A најзначајнији по рангу, а добијени су и други резултати.

3.3 Ефикасно израчунавање *PageRank*-а

Данашњи веб може да се представи као огроман граф или матрица за коју једна итерација множења узима огромну процесорску и меморијску снагу. На све то се додаје да је потребно између 50 и 70 итерација да би итерација постала релативно непромењена (на две децимале).

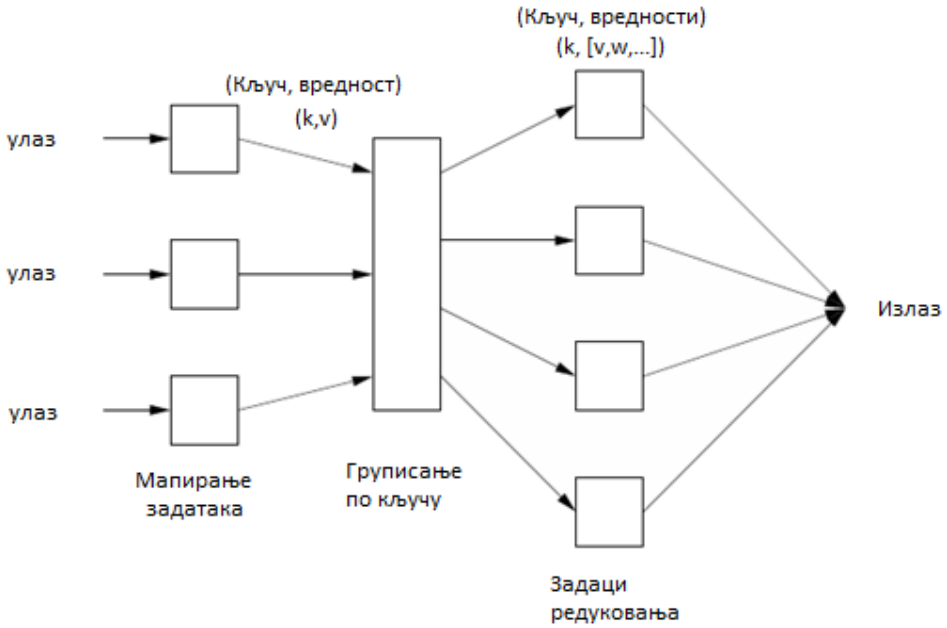
Да би се убрзао процес израчунавања ранга (алгоритмом *PageRank*), сам процес се дели на што више нити, које се рачунају одвојено и након тога спајају. Нити су независни делови неког процеса који добијају одређене улазне податке и избацују решења која се даље употребљавају.

3.3.1 Редуковање матрица – *MapReduce*

Фајл системи садрже велику количину података и захтева се да се операције на њима извршају врло брзо. Због тога се уводи систем паралелног израчунавања. Идеја алгоритма *MapReduce* је да се систем подели на мање делове који су независни. Таква архитектура је отпорна на грешке јер ако се деси проблем на једно нити, понавља изршавање само те нити.

Имплементација алгоритма на матрицу омогућава да операције на матрицама буду толерантне на хардверске грешке. Овакве имплементације се налазе у системима као што су *Google* и фајл систем сервера *Apache*.

За имплементацију је потребно да се напишу функције мапирања и редуковања (које се објашњене у следећем пасусу, са сликом 9) док фајл систем управља задацима уз координацију и бригу да ли нека од тих нити није успела да се изврши.



Слика 8 – Пример „*MapReduce*“ редуковања

Мапирање задатака је разврставање података на начин који је лакши за рачунање. Улазни подаци се разврставају по паровима кључ-вредност. (слика 8) Кључеви не морају бити јединствени. Пример су речи у документу који се у овој фази праве као пар (реч, 1).

Груписање по кључу је део који обавља систем тако што за сваки кључ формира листу вредности (слика 8). Добија информацију у колико задатака треба да их подели и прави функцију која ће их поделити.

Задаци редуковања добијају један или више парова кључ-вредност(и) од система извршава их и прослеђује једном истом фајлу где се спајају са осталим вредностима (слика 8). Пример су речи у документу који у овој фази постају (реч, број појављивања)

Комбинација је случај када фукцију редукције можемо искористити у сврху груписања по кључу. Пример су речи у документу јер груписање у фази редуковања су могли да се ураде већ у фази груписања.

3.3.1.1 PageRank итерација користећи Редуковање матрица

Претпоставља се постоји матрица M са димензијама $n \cdot n$.

$$x_j = \sum_{i=1}^{k} v_i \, M_{ij}$$

Ако је матрица димензије до $n = 100$, онда нема потребе за редуковање матрица. Веб је далеко већи па редуковање матрица повећава ефикасност у таквим ситуацијама. Претпоставља се да информације могу да се чувају у (i, j, M_{ij}).

У случају да вектор v може да стане у меморију онда функција мапирања задатка разврстава улазне податке у пар кључ-вредност $(j, v_i \cdot M_{ij})$, Функција редукције ће једноставно сабрати све вредности који имају исти кључ j односно компонента x_j и резултат ће бити пар (j, x_j).

Ако пак вектор v не може да стане у меморију онда се матрица дели хоризонтално у траке, док се вектор дели на исто толико вертикалних делова. На тај начин j-та трака се множи само са j-том компонентом вектора v. Функције мапирања и редукције се праве на исти начин. Дакле, матрица се разбија на блокове, а вектори на подвекторе:

$$[v'_1 \quad v'_2 \quad v'_3 \quad v'_4] = [v_1 \quad v_2 \quad v_3 \quad v_4] \cdot \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix}$$

Да би се избегло да се делови матрице више пута довлаче у меморију, матрицу се дели и по вертикали па се добија матрицу $k \cdot k$ (у случају наведене матрице, $k = 4$). На тај начин добија се могућност да матрица израчуна са једним позивом квадрата у матрици и k позива вектора v.

3.3.2 Листе повезаности

Матрица која повезује веб је ретка. У просеку има 10 линкова по страници, док линкова постоји око 10 милијарди, што значи да је вероватноћа да је ненула вредност у матрици $1/10^9$.

Да би се рангирање извршило ефикасније, формира се листе повезаности која дефинише за сваки чвор ка којим другим чворовима има линкове тј. гране.

Пример матрице M и њена листа повезаности:

$$M = \begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 \\ 1/2 & 0 & 0 & 1/2 \\ 1 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 \end{bmatrix}$$

Почетна тачка	Степен	Завршне тачке
A	3	B,C,D
B	2	A,D
C	1	A
D	2	B,C

Овакве матрице су корисне само ако је матрица ретка. У овом случају није корисна, али приказује пример пресликавања.

Ако се овај принцип користи и код редуковања матрица у тренутку када је матрица M разбијена на k^2делова, за сваку M_{ij} рачуна се одвојено табела. На пример, горња матрица може да се разбије на 4 дела:

$$\begin{bmatrix} \mathrm{M}_{11} & \mathrm{M}_{12} \\ \mathrm{M}_{21} & \mathrm{M}_{22} \end{bmatrix}$$

При томе се блокови могу представити следећим табелама:

Почетна тачка	Степен	Завршне тачке
A	3	B
B	2	A

Почетна тачка	Степен	Завршне тачке
A	3	C,D
B	2	D

Почетна тачка	Степен	Завршне тачке
C	1	A
D	2	B

Почетна тачка	Степен	Завршне тачке
D	2	C

У последњој табели избачен је чвор C као почетна тачка јер не постоји линк из тог чвора. Веб је ретка матрица, па табела као што је последња M_{22} на вебу има велики број. Овим принципом се растерећује меморијски простор.

3.4 Тематски осетљиви *(Topic-Sensitive) PageRank*

Резултати претраге би могли да се приближе конкретном кориснику ако би свако од корисника имао своје рангирање страница. Пошто би захтевало огроман меморијски простор, није реално изводљиво. Не може се да постићи тотална персонализованост али се може приближити групи специфичних корисника.

Додавањем тежина неким страницама даје се већа шанса неким страницама у односу на друге. Додавањем различитих вероватноћа за неке линкове, нарушава се концепт случајног сурфера. Таква кретање сурфера се назива „пристрасна случајна шетња“.

Постоје речи које имају вишеструко значење и могу бити из различитих области. На пример, верзије оперативног система *Android* увек имају називе неких слаткиша и ако би корисник уписао назив неку од тих слаткиша, добио би концептуално различите странице. Нека би можда показивала на Википедију са тим слаткишом, а нека би можда водила на сајт Андроида. Корисници углавном имају нека правила претраге и на основу претходних резултата, може се предвидети на шта су тачно мислили (са већем вероватноћом).

Тематски осетљив *PageRank* омогућује да се израчуна ранг за различите теме, а да се сваком кориснику одреди коју тему највише воли да претражује. На тај начин, сваки корисник има свој вектор рангирања теми. То није прецизно као да свако има своје резултате рангирања, али захтева мање простора.

Странице које су у вези са неком темом, имају велику вероватноћу да имају линкове на странице које су такође из исте теме.

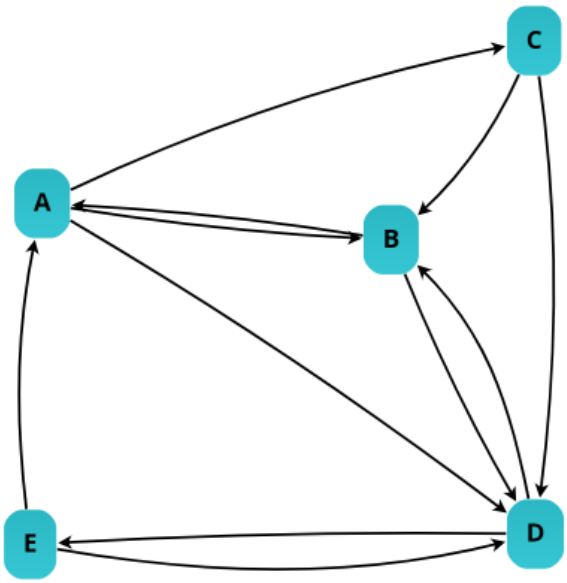
Поред генералне дефиниције *PageRank*-а уз неку вероватноћу, додаје се и вектор који одређује које странице припадају одређеној теми.

$$v' = v\beta M + (1 - \beta)e_s/|S|$$

У овом изразу је:

β – вероватноћа између 0.8 и 0.9
S – који садржи странице из оређене теме
e_s – вектор који има 1 ако је елемент у S, а 0 ако није.
$|S|$ – број страница у S

Као илустрација може да послужи граф са слике:



Слика 9 – Пример јако повезаног графа који се користи за *Topic-Sensitive PageRank*

Вероватноћа је 0.8, док су чворови који припадају одређеној теми $S = \{A, E\}$, према томе, израз за једну итерацију графа са слике 9, гласи:

$$v' = v \cdot 0{,}8 \cdot \begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 0 & 1/2 \\ 1/2 & 0 & 0 & 1/2 & 0 \end{bmatrix} + 0{,}2 \cdot [1/2 \quad 0 \quad 0 \quad 0 \quad 1/2]$$

После одређеног броја итерација (20), добија вектор:

$$PageRank(A, B, C, D, \text{E}) = (0.26, 0.20, 0.07, 0.25, 0.20)$$

Да се није користио „телепорт“ за *PageRank,* вредност странице D би била највећа:

$$PageRank(A, B, C, D, \text{E}) = (0.19, 0.23, 0.06, 0.28, 0.14)$$

3.4.1 Истраживање корисника

Претпоставља се да постоји дефинисан низ тема (са својим јединственим вектором који одређује конкретну тему) за које се извршава специјализовано рангирање. Да би се знало који специјализовани *PageRank* се обрађује приликом упита, потребне су информације о кориснику који је поставио упит.

Како се интернет користи за велики спектар информација до којих се долази, успут се остављају информација о кориснику. Појавом друштвених мрежа, створена је могућност да се интернет у великој мери прилагоди корисницима и њиховим интересовањима, па је зато једно од могућности да алат за претраживање изабере одређену тему на основу корисникових интересовања које на друштвеним мрежама оставља.

Алат за претрагу садржи информације шта је корисник раније претраживао (преко истог претраживача или другог). Сама конкретна реч која је раније претраживана је чешћа у некој теми, а у некој ређа. Различите речи у различитим језицима се често користе за различиту сврху па је алгоритам решавања класификације зависан од језика.

Поред наведених начина, претрага може добити информације и од списка линкова које корисник користи (*bookmarked*).

3.5 *LinkSpam*

У тренутку кад је *Google* учинио спамовање термовима немогућим, појавиле су се технике које су нарушавале и алгоритам *PageRank*. Пошто се нарушава јачина линка, техника нарушавања названа је *LinkSpam*. Иако неке странице нису доступне неким корисницима, ипак постоје начини да се оставе информације на њима. Данашњи веб је све интерактивнији и велики број сајтова има могућност да се остави неки траг као што су коментари. Такође, поред „дозвољеног“ деловања, постоји и „недозвољено“ као што је хаковање нечијег сајта и остављање трагова у виду линкова који могу водити на нежељени сајт.

3.5.1 Спам фарме (*SpamFarm*)

Када злонамерни корисник успе на некој екстерној (високорангираној) страници да постави линк ка својој страници, он прави фарму спама којима покушава да „исцрипи“ што је више могуће вредности линка који је поставио. Фарма спама (*SpamFarm)* је колекција страница којом се повећава рангирање.

Са стране спамера, странице на вебу делимо на:

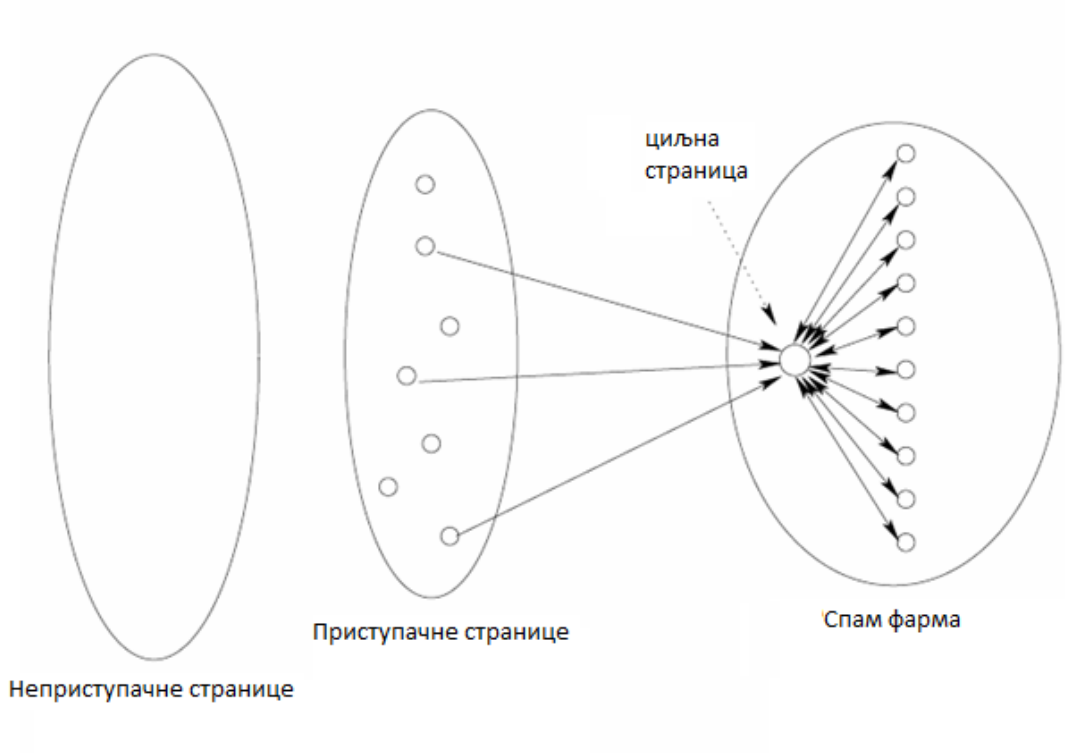
- **Неприступачне странице** – странице на које злонамерни корисник не може да делује. Ове странице су често веома затворене или имају високе безбедоносне мере. Немају могућности за кометаре. Ипак хаковањем могу постати приступачна страница, барем у неком кратком периоду.
- **Приступачне странице** – странице на које злонамерни корисник може да делује. Углавном су то сајтови као што су блогови, вести и слични сајтови на којима могу да се остављају коментари. Углавном се поставља линк ка страници за коју корисник хоће повећа ранг.

  На пример може се оставити овакав коментар:

  ```
  Dell optiplex 790 dt driver update and drivers installation dvd disk
  http://gloomates.260mb.net/qoracuz/super-mario-games-collection-
  englishpc-gamestorrents.html


  Hi, great post. Please check my post at randomsite.com
  ```

- **Спам странице** – странице које поседује злонамерни корисник и са којима има потпуну слободну за прављење фарме. Постоји једна циљна (*target*) страница којој корисник хоће да повећа ранг, док остале странице користи као помоћне. На циљној страници, корисник прави линкове ка свим помоћним страницама, док на помоћним прави линкове као циљној страници. На тај начин се спречава расипање резултата *PageRank* алгоритма који је „украден“ од приступачне странице.



Слика 10 – Пример изгледа веб-а из угла злонамерног корисника

Ако се користи принцип „опорезивања“, онда постоји нека вероватноћа β са којом слободни сурфер излази из фарме спама.

- t је „*target*“, тј. циљна страница
- y је *PageRank* циљне странице
- x је *PageRank* који долази од пристипачне странице
- m је број страница који чине фарму спама

PageRank помоћне странице (z) се рачуна према изразу:

$$PageRank(z) = \frac{(\beta \cdot y)}{m} + \frac{(1-\beta)}{n}$$

Први сабирак је *PageRank* циљне странице, док се други *PageRank,* који се добија од стране свих страница на вебу, може се занемарити јер је изузетно мали за даља рачунања.

PageRank у се добија од приступачне странице и од помоћних страница

$$y = x + \beta \cdot m \cdot \frac{(\beta \cdot y)}{m} + \frac{(1-\beta)}{n} = x + \beta^2 y + \beta(1-\beta) \cdot \frac{m}{n}$$

Ако се изврши смена и рачуна y:

$$c = \frac{\beta(1-\beta)}{(1-\beta^2)} = \frac{\beta}{1+\beta}$$

Добија се израз за y:

$$y = \frac{x}{1-\beta^2} + c \cdot \frac{m}{n}$$

што представља ранг циљне странице. Наравно, да не постоји страница из приступачне зоне, ни помоћне странице не би имале икакву сврху. Наведени пример (архитектура) представља само једну од варијација фарми спама.

3.5.2 Решавање проблема *LinkSpam* – а

Како су се развијали начини нарушавања регуларног рачунања рангирања алгоритмом *PageRank*, тако су се развијали и начини решавања таквих проблема. Алати за претрагу су примећивали овакве шаблонске кретње и успевали су да „насилно“ сруше ранг сајтова са оваквим архитектурама. Како постоје низ варијација архитектуре фарми спама, тако постоји могућност да се поново збуни претрага. Овакав начин „ратовања“ прављења нове архитектуре и решавање тог проблема се води већ дуго.

Осим могућности примећивања одређене архитектуре, постоји могућност и да се утиче на сам алгоритам *PageRank*-а који смањује нежељене ефекте.

3.5.2.1 Trust Rank

Идеја је да се дефинишу странице које су сигурно од поверења и да се њихови линкови вреднују више него остали. Постоји велика тенденција да ће такви сајтови имати линкове само ка регуларним сајтовима а не спам сајтовима. *TrustRank* је *topic-sensitive PageRank* где „тема“ представља сајтове који су од поверења.

Такви сајтови морају бити „безбедни“ да не би додатно нарушавали ранг. Углавном се не бирају по аутоматизму већ сам човек дефинише такве сајтове. Најчешћа пракса је да се бирају сајтови који су под неким заштићеним доменом, као што су *.edu .gov* или други. Такви домени постоје под различитим националним доменима.

Имплеметација се извршава тако што се прави „телепорт“ од стране важних (од поверења) страница.

3.5.2.2 Spam Mass

Spam Mass је алгоритам који користи и пореди вредност алгоритма *PageRank* и вредност *TrustRank* алгоритма и на основу резултата одређује да ли је нека страница спам или није. Ако се *r, t, s* редом означавају

r – PageRank

t – TrustRank

s – SpamMass

Онда се *s* рачуна:

$$s = \frac{(r - t)}{r}$$

Ако је вредност негативна или мало позитивна реч је о регуларној страници. Иначе, страница је врло могуће спам страница.

Добра страна овог алгоритма је што није потребно истраживање архитектуре неког сајта да би се дошло до закључка да је сајт спам.

3.6 *HITS* као пандан *PageRank*-u

У тренутку кад је настајао *PageRank*, у истом граду независно је развијен алгоритам *Hypertext Induced Topic Search* (*HITS)*. Млади научник Џон Клејнберг (Jon Kleinberg) развио је алгоритам који је такође користио хиперлинкове да би се одредили резултати претраге на Интернету. За разлику од ктитора алгоритма *PageRank*-а, Џон је остао на факултету и није комерцијалозовао алгоритам. Касније су други предузетници основали компаније које су развијале алате за претрагу са алгоритмом *HITS*.

Алгоритам је за разлику од *PageRank*-а користи линкове који показују на неки чвор, али и линкове коју су кретали из тог истог чвора. Алгоритам није претпроцесорски већ се извршава у тренутку кад се постави упит.

HITS користи два вектора за рангирање. Компоненте вектора су бројеви који за сваки сајт показују у колико мери су они власт, односно чвориште.

- *Authorities* (власти) – странице које садрже информације које корисник тражи
- *Hubs* (чворишта) – странице које приказују на којим страницама се налазе информације које корисник тражи

Страница је добро чвориште ако линкује на добре власти и страница је добра власт ако је линкована од стране добрих чворишта.

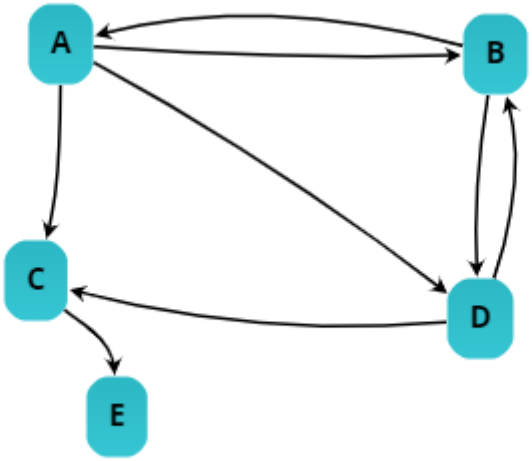
3.6.1 Дефиниција HITS-а

Посматрају се две вредности, тј. два вектора.

h – *hubbines* која садржи вредности страница рангиране по квалитету чворишта
a – *authorities* која садржи вредности странице рангиране по квалитету власти

Матрица повезаности L димензија $n \cdot n$, где је n број страница, састоји се од вредности L_{ij}, а L_{ij} једнака је 1 ако постоји грана од чвора са индексом *i* до чвора са индексом *j*, иначе 0. L^T је транспонована матрица која приказује одакле долазе линкови, а ${L^T}_{ij}$ је једнака јединици ако постоји грана од чвора са индексом *j* до чвора са индексом *i*, иначе 0.

На слици 11, приказан је пример графа и његове матрице повезаности.



$$L = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad L^T = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Слика 11 – Пример графа и матрица алгоритма *HITS*

Вектори h, a и матрица L повезани су следећим изразима:

$$h = a \cdot \lambda L$$

$$a = h \cdot \mu L^T$$

У изразима се појављују и фактори скалирања λ и μ. На основу тога се види да су h односно a решења следећих система једначина:

$$h = h \cdot \lambda\mu L L^T ,$$

$$a = a \cdot \lambda\mu L^T L .$$

Матрице LL^T нису ретке као кад L и L^T, па је ефикасније да се рачунају у итерацијама a и h, након чега се скалирају тако да највеће компоненте вектора буду 1. Итерације се понављају док не се не добију вредности вектора које се за малу вредност разликују од претходне.

На пример, за граф и матрицу са слике 11 добија се следећи низ вектора:

$h = [1 \quad 1 \quad 1 \quad 1 \quad 1]$

$hL^T = [1 \quad 2 \quad 2 \quad 2 \quad 1]$

$a = [1/2 \quad 1 \quad 1 \quad 1 \quad 1/2]$ - скалирање

$La = [3 \quad 3/2 \quad 1/2 \quad 2 \quad 0]$

$h = [1 \quad 1/2 \quad 1/6 \quad 2/3 \quad 0]$ - скалирање

$hL^T = [1/2 \quad 5/3 \quad 5/3 \quad 3/2 \quad 1/6]$

$a = [3/10 \quad 1 \quad 1 \quad 9/10 \quad 1/10]$ - скалирање

$La = [29/10 \quad 6/5 \quad 1/10 \quad 2 \quad 0]$

$h = [1 \quad 12/29 \quad 1/29 \quad 20/29 \quad 0]$

на крају итерација, резултат је приближно:

$h = [1 \quad 0{,}7168 \quad 0 \quad 0{,}7165 \quad 0]$

$a = [0{,}2087 \quad 1 \quad 1 \quad 0{,}7913 \quad 0]$

Резултати HITS алгоритма графа са слике 11

3.6.2 Поређење HITS и *PageRank* алгоритма

HITS алгоритам је за разлику од алгоритма *PageRank* имун на паукове замке и на висеће чворове. Није потребно примењивати „опорезивање“ да би се заобишли проблеми које алгоритам *PageRank* има. Алгоритам се извршава по сваком упиту за разлику од *PageRank*–а (*query-dependent*).

Највећа мана *HITS* алгоритма је што не могу да се користе транспоноване матрице или редуковање матрица да би се повећала ефикасност алгоритма.

4 Софтвер – Реализација алгоритма *PageRank*

Софтвер који симулира примену *PageRank* алгоритма развијен је као веб сајт, постављен је на адреси http://www.alas.matf.bg.ac.rs/~mi08189/pagerank

Сајт је комплетно отворен тако да корисник нема потребу за регистрацијом да би користио апликацију. Инцијално се уноси матрица, на основу које се рачуна граф. Постоје примери који помажу кориснику приликом креирања матрице, тј. може да се искористи већ дефинисане матрице. Након тога, корисник бира одређена својства пре него што рачуна *PageRank*. Углавном су то побољшања алгоритма или степен прецизности тј. број итерација.

4.1 Технологије израде

Програм је развијен на вебу са технологијама *PHP, HTML, CSS i JS.*

Као основа користи се *PHP framework Codeigniter*

https://www.codeigniter.com/

који је окружење ниског нивоа php –а. *Codeigniter* је организован по принципу модел-контролер-поглед (*model-view-contoller*). На апликацији модел је сведен на минимум јер се не користи база података, тако да се користи минимални део окружења.

Делови погледа, тј. изглед елемената у веб претраживач користе *Front-end* шаблон (*template*) *Metronic*.

https://themeforest.net/item/metronic-responsive-admin-dashboard-template/4021469

Metronic је модеран шаблон који представља скуп дефинисаних делова *html*-а као и *javascript*-а кода. Основ *Metronic*-а представља данас најпознатија *Front-end* библиотека *Bootstrap*.

http://getbootstrap.com/

Поред већ дефинисаних елемената *html*-а и *css*-а, део који је специјално развијен за потребе *PageRank* апликације написан је у *scss*-у. *Scss* представља угњеждено *css* програмирање.

Срж веб апликације се налази у контолеру где се користи објектно оријентисано *javascript* програмирање у *NodeJs*-у.

https://nodejs.org/en/

Главна функционалност као што је рачунање *PageRank*-а је написана у *javascript* и jquery

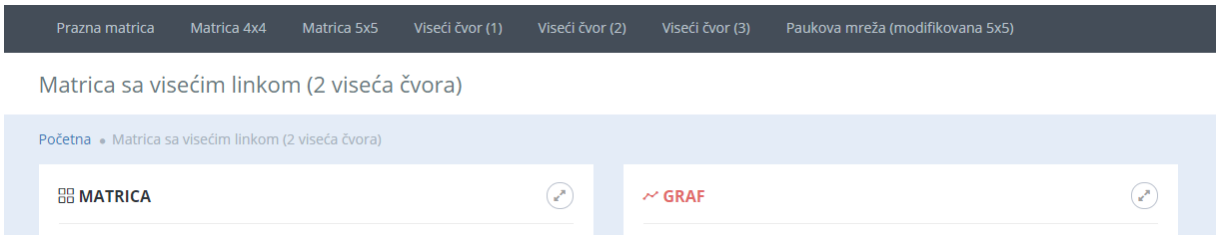
Исцртавање графа, као и могућност управљања графом постигнута је уз помоћ биоблиотеке *gojs*.

https://gojs.net/latest/samples/stateChart.html

Скрипта прихвата *json* запис чворова и страница, као и локацију чворова у координатном систему. Након тога, парсира *json* запис и исцритава граф са чворовима и гранама.

4.2 Делови апликације

Апликација је организована као једна страница са свим функционалностима. Различите странице имају другачије улазне податке тј. матрицу.



Слика 12 – Изглед заглавља програма за рачунање алгоритма *PageRank*

4.2.1 Матрице

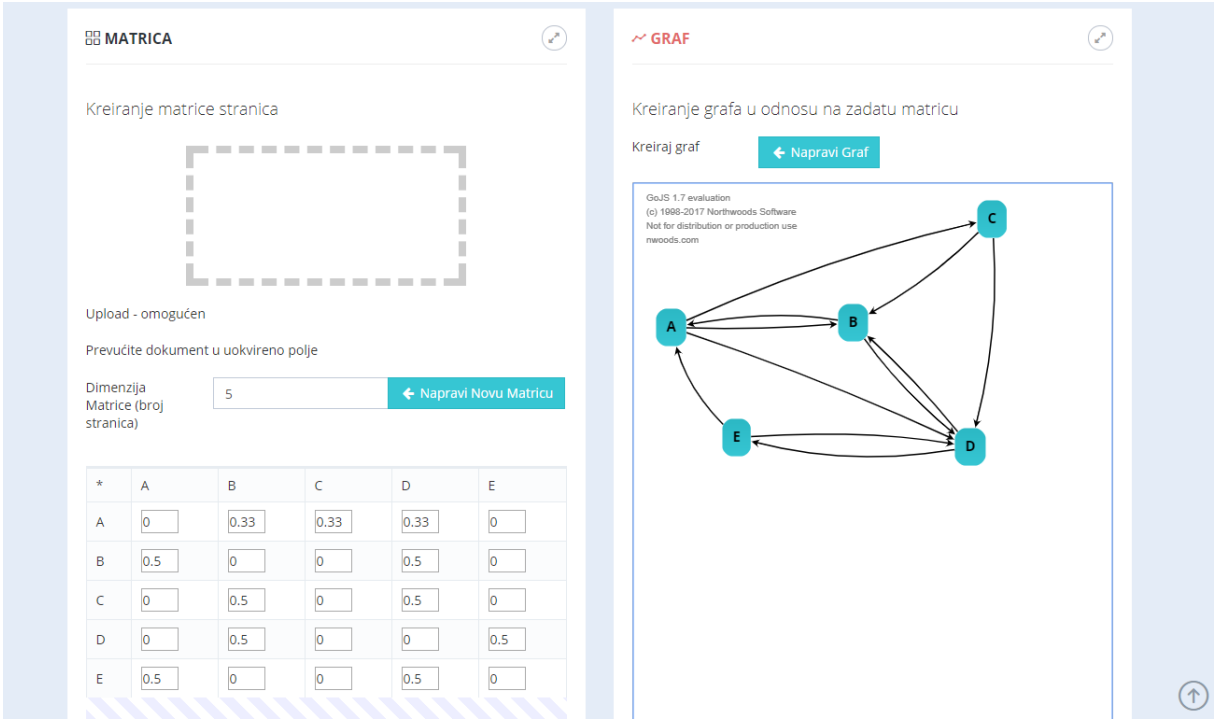
На самом почетку употребе програма, уноси се матрица. Осим предефинисаних матрица које могу да се унесу преко заглавља са слике 12, корисник може да постави своју матрицу. Уносом димензије матрице, тј. количине веб страница, прави се нула матрица (дугме у левом делу слике 13). Други начин да корисник унесе своју матрицу је преко екстерног документа. Документ треба да буде екстензије *.txt* и да саджи n – димензија матрице и уређен пар чворова од ког до ког чвора се простире грана. Између грана се налази размак. Формат документа је:

$$n$$
$$(A, B)\ (B, C)\ \ldots$$

Датотеку која садржи информације матрице, треба превући у уоквирено поље (слика 13).

Због лакшег прегледа користи се табеларни приказ матрице помоћу алата *datatables*

https://datatables.net/extensions/scroller/.



Слика 13 – Изглед уноса матрице и исцртавања графа програма

4.2.2 Граф

Након уноса нове матрице и вредности у њој, формира се матрица и њен садржај се парсира у *json* запис. Тај запис се прослеђује скрипти за исцртавање графа. Граф се исписује по колонама али корисник има могућност да измени приказ директно на радној површини где је исцритан граф, тј. у десној делу апликације (десни део слике 13)

Корисник има могућност померања чворова као и могућност брисања. Практичност може бити у томе што визуелно може лакше да се види који су чворови висећи и шта се дешава у рекурзивном ослобађању од висећих чворова.

У наставку је пример програмског уношења матрице:

1. `linkDataArray:Array(11) // линкови`

from:0 text:0.33 to:1	from:0 text:0.33 to:2
from:0 text:0.33 to:3	from:1 text:0.5 to:0
from:1 text:0.5 to:3	from:2 text:0.5 to:1
from:2 text:0.5 to:3	from:3 text:0.5 to:1
from:3 text:0.5 to:4	from:4 text:0.5 to:0
from:4 text:0.5 to:3	

2. `nodeDataArray:Array(5) // странице`

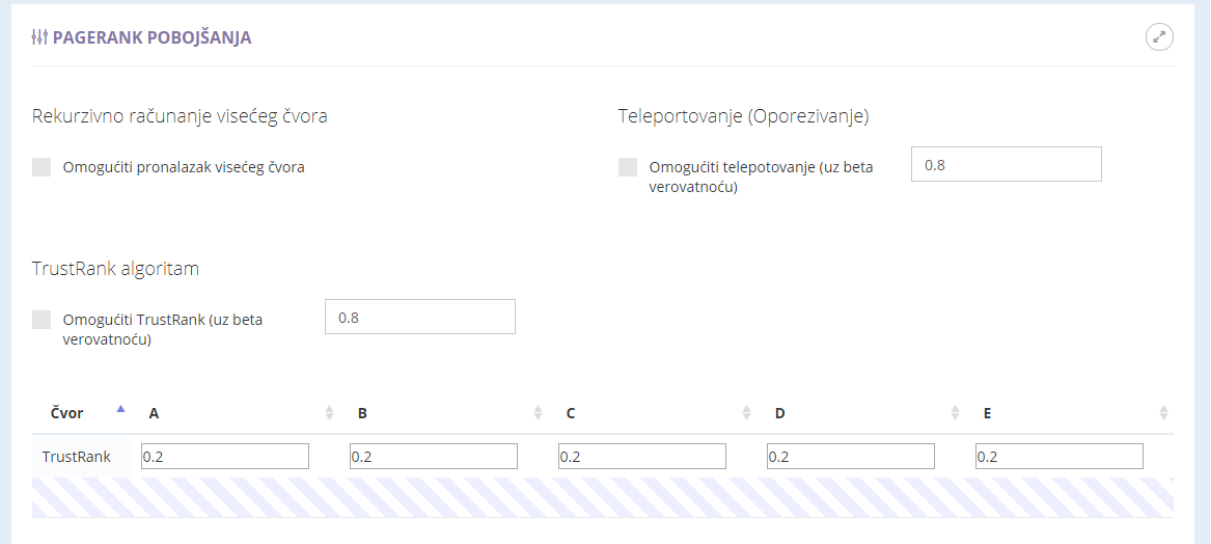
id:0 text:"A"	id:1 text:"B"
id:2 text:"C"	id:3 text:"D"
id:4 text:"E"	

Затим се ови подаци конвертују у *JSON* запис


```
{
  "nodeKeyProperty": "id",
  "nodeDataArray": [
    {
      "id": 0,
      "text": "A"
    },
    {
      "id": 1,
      "text": "B"
    },
    {
      "id": 2,
      "text": "C"
    },
    {
      "id": 3,
      "text": "D"
    },
    {
      "id": 4,
      "text": "E"
    }
  ],
  "linkDataArray": [
    {
      "from": 0,
      "to": 1,
      "text": 0.33
    },
    {
      "from": 0,
      "to": 2,
      "text": 0.33
    },
    {
      "from": 0,
      "to": 3,
      "text": 0.33
    },
    {
      "from": 1,
      "to": 0,
      "text": 0.5
    },
    {
      "from": 1,
      "to": 3,
      "text": 0.5
    },
    {
      "from": 2,
      "to": 1,
      "text": 0.5
    },
    {
      "from": 2,
      "to": 3,
      "text": 0.5
    },
    {
      "from": 3,
      "to": 1,
      "text": 0.5
    },
    {
      "from": 3,
      "to": 4,
      "text": 0.5
    },
    {
      "from": 4,
      "to": 0,
      "text": 0.5
    },
    {
      "from": 4,
      "to": 3,
      "text": 0.5
    }
  ]
}
```

4.2.3 Побољшања

Ако приликом рачунања ранга дође до неког специфичног случаја као што је висећи чвор, паукова мрежа или постоји потреба за побољшањем алгоритма *TrustRank*-ом, корисник може да искористи неко од побољшања. Побољшања се чекирају на одељку са слике 14.



Слика 14 – Изглед одељка програма са побољшањима алгоритма

4.2.3.1 Рекурзивно рачунање висећег чвора (имплементација)

Имплементација се састоји из три дела. Формирају се два стека.

Прва фаза почиње тако што се матрица претвори у 1/0 матрицу, где 1 означава да постоји страница, док 0 означава да не постоји страница (програмски код 1). Уписује се у стек за матрице, па се тражи висећи чвор. Кад се висећи чвор пронађе, ставља се на стек за избачене чворове, прави се нова матрица без тог чвора и ставља на стек за матрице. Итерација се понавља док се добије јако повезан граф. (претпоставка да постоји један јако повезан граф). На крају ове фазе, имамо два стека где је стек за матрице за један индекс дужи од стека за избачене чворове (програмски код 2.1 и 2.2).

```
// иницијална матрица 1/0
pageRankLinkingNodeMatrices[0] = [];
for(var i = 0; i < matrixdimension; i++) {
    pageRankLinkingNodeMatrices[0][i] = [];
    for(var j = 0; j < matrixdimension; j++) {
        pageRankLinkingNodeMatrices[0][i][j] = (matrix[i][j] !== 0) ? 1 : 0;
    }
}
```

Програмски код 1 – алгоритам попуњавања матрице у 1 или 0 вредност

```
var k = 0;
// број итерација
while (repeatFlag) {

    repeatFlag = false; // флег за прекид понављања
    for (var i = 0; i < varNewDimension; i++) {

        linkingNodeFound = true; // претпоставка да је нађен

        // ако у једном реду су свуда нуле
        for (var j = 0; j < varNewDimension; j++) {
            if(pageRankLinkingNodeMatrices[k][i][j] === 1) {
                linkingNodeFound = false;
            }
        }

        // ако је нађен висећи чвор, чвор се смешта у низ избачених
        if (linkingNodeFound) {
            pageRankLinkingNodeDeleted.push(i);
            repeatFlag = true;  // дозвољава се да поновна претрага
                                   матрица пошто је пронађен висећи чвор
            // deletedRow and deletedColumn помаже да се креира нову
               матрицу без врсте и колоне у којој се налази избачени чвор
            var deletedRow = false;

            pageRankLinkingNodeMatrices[k+1] = [];
            for(var p = 0; p < varNewDimension-1; p++) {
            // прелази се ред по ред и обраћа се пажња да ли прећен
               ред који је избачен

                pageRankLinkingNodeMatrices[k+1][p] = [];

                if(p === pageRankLinkingNodeDeleted
                   [pageRankLinkingNodeDeleted.length - 1]) {
                    deletedRow = true;
                }

                var deletedColumn = false;

                for(var q = 0; q < varNewDimension-1; q++) {
                 // прелазе се све колоне у реду и обраћа се пажња да ли је
                    прећена колону која је била избачена

                    if(q === pageRankLinkingNodeDeleted
                       [pageRankLinkingNodeDeleted.length - 1]) {
                        deletedColumn = true;
                    }
```

Програмски код 2.1 – алгоритам проналаска висећег чвора

```
// прелазе се све колоне у реду и обраћа се пажња на
// четири слзчаја. Није прећен ни ред ни колона. Јесте само ред, само колону или
ни једно ни друго.
                    if(deletedRow) {
                        if(deletedColumn) {
                            pageRankLinkingNodeMatrices[k+1][p][q] =
                            pageRankLinkingNodeMatrices[k][p+1][q+1];

                        }
                        else {
                            pageRankLinkingNodeMatrices[k+1][p][q] =
                            pageRankLinkingNodeMatrices[k][p+1][q];

                        }
                    }
                    else if (deletedColumn) {
                        pageRankLinkingNodeMatrices[k+1][p][q] =
                        pageRankLinkingNodeMatrices[k][p][q+1];

                    }
                    else {
                        pageRankLinkingNodeMatrices[k+1][p][q] =
                        pageRankLinkingNodeMatrices[k][p][q];

                    }

                }
            }

            k++;
            varNewDimension--;


            break;
        }
    }
}
```

Програмски код 2.2 – алгоритам проналаска висећег чвора

Друга фаза је рачунање *PageRank*-а за преостали граф.

Трећа фаза је процес скидања чворова са стека изабачених чворова и истовремено рачунање одговарајућих матрица. Приликом враћања чвора, посматра се матрица пред избацивање тог чвора. Преко ње се проналази колико и који чворови имају линк ка избаченом чвору. Процес се врши рекурзивно док се не прођу сви чворови (програмски код 3).

```
// креће се од краја. Низ ибрисаних чворова има један мање елемент него
редуковане матрице
for(var l = pageRankLinkingNodeDeleted.length - 1; l >= 0; l--) {
    var newPageRank = 0;

    // по редовима тражимо да ли је од стране неког чвора постоји линк ка
       избрисаном чвору.
    for(var i = 0; i < pageRankLinkingNodeMatrices[l].length; i++) {
        // не посматра се ред или колону адресе које су враћене
        if(i === pageRankLinkingNodeDeleted[l]) {
            continue;
        }
        // у тренутку када се наиђе до чвора из кога постоји линк, сабира се
          колико линкова има тај чвор и тај број се дели његовим израчунати
          ранком
        if(pageRankLinkingNodeMatrices[l][i][pageRankLinkingNodeDeleted[l]] === 1)
        {
            var counter = 0;
            for(var j = 0; j < pageRankLinkingNodeMatrices[l].length; j++) {
                if (pageRankLinkingNodeMatrices[l][i][j] === 1) {
                    counter++;
                }
            }


            if(counter !== 0) {
                if(i > pageRankLinkingNodeDeleted[l]) {
                    newPageRank += pageRankResult[i-1]/counter;
                }
                else {
                    newPageRank += pageRankResult[i]/counter;
                }
            }
        }
    }

    newPageRankResults = [];

// обраћа се пажња на локацију чвора у матрици.
    for(var i = 0; i < pageRankLinkingNodeMatrices[l].length; i++) {
        if(i < pageRankLinkingNodeDeleted[l]) {
            newPageRankResults[i] = pageRankResult[i];
        } else if(i === pageRankLinkingNodeDeleted[l]) {
            newPageRankResults[i] = newPageRank;
        } else {
            newPageRankResults[i] = pageRankResult[i-1];
        }

    }

    pageRankResult = newPageRankResults;

}

return pageRankResult;
```

Програмски код 3 – алгоритам одређивања *PageRank*-а висећим чворовима

4.2.3.2 Телепортовање (имплементација)

Имплементација се састоји само од читања задате вредности β као и формирања вектора случајног сурфера. Матрица се најпре модификује, па се додаје вектор случајних вредности (програмски код 4).

У програму се могу користити и вредности које одступају од стохастичке матрице.

```
// учитавање бета вредности
beta = taskPageRankTrustRankValue.val();

var normalSurfingVectorChanged = []

// множење матрице за вектором телепортације
for(var i = 0; i < matrixdimension; i++) {
    for(var j = 0; j < matrixdimension; j++) {
        newMatrix[i][j] = matrix[i][j] * beta;
    }
}

// рачунање TRUSTRANK вектора (за стандардно телепортовање додајемо случајне
вредности)
for(var j = 0; j < matrixdimension; j++) {
    normalSurfingVectorChanged[j] =
      $('.pagerank-trustrank-item-i1j'+(j+1)).val() * (1 - beta);
}

pageRankFixing[0] = normalSurfingVectorChanged;
// у случају да смо променили вектор
```

Програмски код 4 – алгоритам телепортовања (опорезивања)

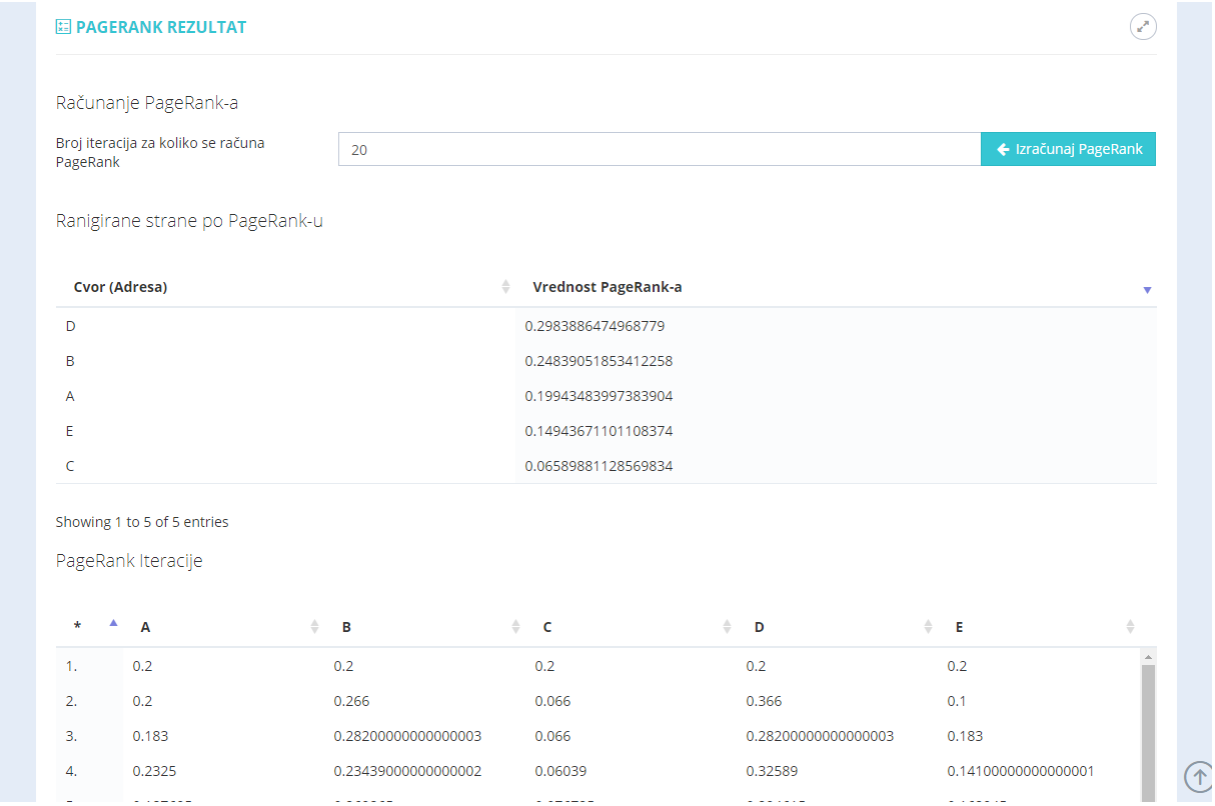
4.2.3.3. TrustRank Алгоритам (имплементација)

Приликом иницијалног креирања матрица, кориснику се одмах штампа и *TrustRank* вектор, који је на почетку једнак вектору случајних вредности.

Имплементација је базирана на принципу реализације телепортовања. За разлику од телепортовања, где вектор садржи једнаке вредности вероватноће за све компоненте, корисник има могућност да сам унесе вредности компоненти.

4.2.4 *PageRank* резултати

Након што корисник одреди колико итерација жели да се изврши и покрене поступак рачунања, штампају се резултати рангирања алгоритмом *PageRank*-а. Адресе су сортиране према опадајућим вредностима рангова. На слици 15, табела „Рангирање странице по *PageRank*-у“, налазе се сви чворови иако је можда дошло до неких одступања приликом рачунања.



Слика 15 – Пример резултата програма за рачунање алгоритма PageRank

Друга табела „*PageRank* итерације“ на слици 15 приказује кретање резултата по итерацијама. У овој табели се приказују само чворови који се налазе у матрици за коју се рачуна вредност алгоритма *PageRank*. Ако је пре тога извршено уклањање висећих чворова, онда рангови тих чворова неће бити рачунати у итерацијама, већ се оне тек на крају рачунају на основу постојећих резултата.

Поглед добија низ вектор за сваку итерацију. Последња итерација је приказана као резултат, док се све штампају у одељку за итерације.

4.3 Алгоритам *PageRank*

Главни део алгоритма је множење матрица уз могуће додатке. Пре самог множења вектора и матрице, оне су модификоване ако се користи неки корак „побољшања“, нпр множење вектором за телепортовање или редуковање матрице због висећих чворова. Након множења, додаје се вектор за телепортовање ако је активан, тј. ако се користи неко побољшање.

Операције се понављају у зависности од броја итерација. Кад се добију резултате *PageRank* рангирања, рачунају се рангови чворова који су избачени.

Резултати се шаљу у итерацијама „погледу“ где само последња итерација садржи обавезно све чворове.

```
// број итерација и множење матрица

numberOfIteration = taskPageRankIterationNumber.val();

for(var q = 0; q < numberOfIteration; q++) {
    pageRankOfMatrix = matrixMultiply(pageRankOfMatrix, newMatrix);

    // ако постоји било каква модификација
    if (beta !== 1) {
        for (var p = 0; p < matrixdimension; p++) {
            pageRankOfMatrix[0][p] = pageRankOfMatrix[0][p] +
                                     pageRankFixing[0][p];
        }
    }

    iterationResult[q+1] = pageRankOfMatrix[0];

}

// ако је мапа редукована, морају се вратити избачени чворови
if(iterationResult[0].length > iterationResult[numberOfIteration - 1].length)
{
    pageRankdcPrintRang(reverseLinkingNode(
                iterationResult[numberOfIteration - 1]));
}
else {
    pageRankdcPrintRang(
                iterationResult[numberOfIteration - 1]);
}


pageRankdcPrintResult(iterationResult);
```

Програмски код 5 – главни део алгоритма *PageRank*

4.4 Мале промене линкова – велике промене у рангу

Чести су упади на сајтове који су високо котирани на *Google* претрази. Злонамерни корисник оставља свој садржај у неком делу коју не може испрва лако да се примети. Нпр. ставља се садржај у подножје(*footer*). Такве ствари могу да наруше ранг, што се може илустровати следећим примером. Користи се матрицу $10x10$, са чворовима A,B, ... J :

$$\begin{bmatrix} 0 & 0.2 & 0 & 0.2 & 0.2 & 0 & 0.2 & 0 & 0 & 0.2 \\ 0 & 0 & 0.33 & 0 & 0.33 & 0 & 0 & 0.33 & 0 & 0 \\ 0.33 & 0 & 0 & 0 & 0 & 0 & 0.33 & 0 & 0 & 0.33 \\ 0.2 & 0 & 0.2 & 0 & 0 & 0.2 & 0.2 & 0.2 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0.2 & 0.2 & 0 & 0 & 0 & 0.2 & 0 & 0 & 0.2 & 0.2 \\ 0 & 0 & 0.33 & 0 & 0 & 0.33 & 0 & 0 & 0 & 0.33 \\ 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 0.2 & 0.2 & 0.2 & 0.2 & 0.2 & 0 \\ 0(0.25)^{5} & 0 & 0 & 0.33(0,25) & 0 & 0 & 0 & 0 & 0.33(0,25) & 0.33(0,25) \end{bmatrix}$$

Претпоставља се да је неко хаковао страницу „*E*“, окачио свој садржај у подножје и да је додао линк ка свом чвору. Претпоставља се да корисник поседује страницу „*A*“ и да хоће да јој повећа ранг. У последњој врсти матрице, добијамо друге вредности јер се додаје једна страница која креће са странице *E (*вредности у заградама*).*

Пре промене, чвор *J* је имао дупло већи ранг (*0,2*) од друге по реду странице, док је страница *A* седма на листи са *0,05*. Након промене и само једног додатог линка, чвор *A* прелази на друго место, тј. вредност ранга је скочио за дупло. Ако се пак обори страница *E*, тј. избаци се страница *E* (добија се матрица $9x9$), разлике ће бити огромне.

[5] Вредности вероватноће пре и после (у загради) модификације графа

PageRank пре промена:

J	0.20265733322497748
D	0.1007253483370075
I	0.09778467860513701
G	0.06999979412195105
C	0.05986063230120091
H	0.05630038878183945
F	0.05438558284279508
A	0.05102337160585606
B	0.04947164483367666
E	0.04631160394467139

PageRank након премена:

B	0.13435034034453724
C	0.13017842330895624
G	0.12637548486658526
A	0.11633101423772683
F	0.11442679908455335
D	0.11233683805601871
E	0.08309210827643629
H	0.07646716995741823
I	0.03821573625462615

Може се приметити да су вредности *PageRank*-а страница *I* и *D* драстично опале.

5 Закључак

Алгоритам *PageRank* је само основа данашњег најпознатијег претраживача. Сваким даном се појављује нови начин са нарушавање ранга, а паралелно са тим и нова правила и захтеви које корисници са својим сајтовима треба да поштују да би били уопште рангирани. Претпоставља се да оређени проценат рангирања зависи од *PageRank*, док остатак зависи од организације и садржаја кода на сајту.

Последњих година појавила се потпуно одвојена грана компјутерске науке која се бави искључиво видљивошћу на вебу. Том облашћу се често баве људи који се баве маркетингом, а не развојем софтвера.

Алгоритми који су описани могу да се искористе за једноставније графове. Веб је скуп више јако повезаних графова, тако би да требало да постоји додатак да се рачуна одвојено по графовима тј. подграфовима, после чега би се формирао заједнички ранг. Тренутна верзија софтвера ради са једним графом. Додатно се у програм може уградити аутоматско проналажење и обрада висећих чворова.

6 Литература

1. *A. Rajamaran, J. Leskovec, J. D. Ullman, Mining of Massive Datasets, Stanford, 2013*

2. *Amy N. Langville and Carl D. Meyer, Google's PageRank and Beyond: The Science of Search Engine Rankings, Princeton University Press, Princeton and Oxford, 2016*

3. Јована Марковић, Ланци Маркова са дискретним временом и примене, Мастер рад, Математички факултет Универзитет у Београду, фебруар 2014

4. Др Миодраг Живковић, Алгоритми, Математички факултет, Универзитет у Београду, 2001

5. *Sergey Brin and Lawrence Page, The Anatomy of a Large-Scale Hypertextual Web Search Engine, Computer Science Department, Stanford University*

6. *Google, Search Engine Optimization, Starter Guide*

7. *Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest ,Clifford Stein Introducion to Algorithms,* MIT Press, 2001.

8. *L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Stanford Digital Libraries Working Paper, 1998.*