LAB 23

SEARCH ENGINES

What You Will Learn

- How to write some components of a search engine
- How to calculate PageRank
- White–Hat Search Engine Optimization Techniques

Approximate Time

The exercises in this lab should take approximately 60 minutes to complete.

Fundamentals of Web Development, 2nd Ed

Randy Connolly and Ricardo Hoar

Textbook by Pearson
http://www.funwebdev.com

Date Last Revised: May 20, 2017

TITLE

PREPARING DIRECTORIES

1 If you haven't done so already, create a folder in your personal drive for all the labs for this book.

2 From the main `labs` folder (either downloaded from the textbook's web site using the code provided with the textbook or in a common location provided by your instructor), copy the folder titled `lab0X` to your course folder created in step one.

This Lab is far more compelling if you are optimizing your own website. While you can apply search engine optimization techniques to locally hosted websites, many techniques require a live site with a domain, including the calculation of PageRank.

Many of the search engine optimization techniques are easy to apply, and therefore do not require walkthroughs. Your instructor may assign additional Search Engine Optimization exercises to this lab.

SEARCH ENGINE COMPONENTS

Exercise 23.1 — WRITE A CRAWLER

1 This first exercise will have you write a simple crawler based on the code in Chapter 23.

2 Create a file named `lab23-exercise01.php` and paste the code below into the file.

```
<?php

class Crawler{
      private $URLList;
      private $nextIndex;
      function __construct($fileName){
             $this->nextIndex=0;
             $this->URLList = explode("\n",file_get_contents($fileName));
             print_r($this->URLList);
      }
      public function getNextURLToCrawl(){
        return $this->URLList[$this->nextIndex++];
      }
      public function scrapeHyperlinks($url){
      }
      public function printSummary(){
             echo count($this->URLList)." links. Index:".$this-
>nextIndex."<br>";
             foreach($this->URLList as $link){
                                       echo $link."<br>";
             }
      }
```

```
       public function robotsAllow($url){
              return false;
       }
       public function crawl(){
             $url = $this->getNextURLToCrawl();
             if($url=="")
                return;
             echo "<h2>Crawling ".$url."</h2>";
             if($this->robotsAllow($url)){
                 echo "$url is allowed<br>\n";
                 $this->scrapeHyperlinks($url);
                 echo "<br>";
             }
             else{
                 echo "$url is NOT allowed";
             }

             $this->crawl();
      }
}
```

3 Now complete the method which determines if you can crawl a particular URL according to the robots .txt directive:

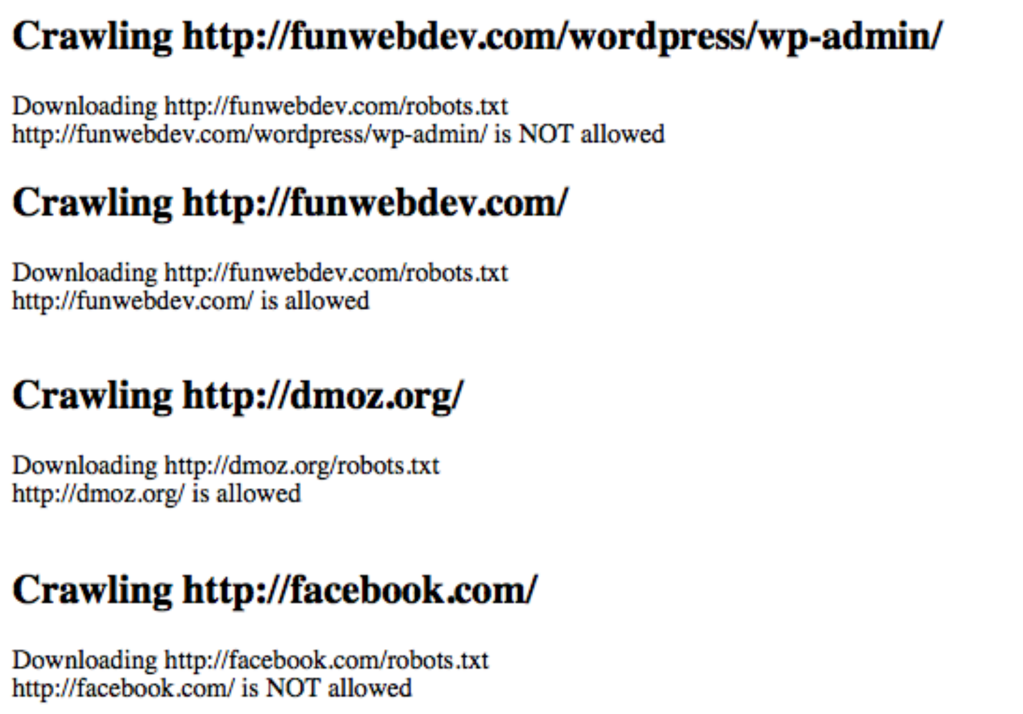
```
      public function robotsAllow($url){
/*Given a URL fetches robots,txt and checks to see if the
  crawl of the URL is allowed. */
          $urlParts = parse_url($url);
          $robots =
$urlParts["scheme"]."://".$urlParts["host"]."/robots.txt";
          echo "Downloading $robots<br>\n";
          $robots_txt = file($robots);
          echo "<pre>".print_r($robots_txt)."</pre><br>\n";
          //adhering to user-agent * rules.
          if(empty($robots_txt)) return true;//no rules against crawling.
          foreach($robots_txt as $line) {
               if(!$line = trim($line)) continue; //skip blanks

               if(preg_match('/^\s*Disallow:(.*)/i', $line, $regs)) {
                  if(!$regs[1]) return true;
                  // add rules that apply to array for testing
                  $rules[] = preg_quote(trim($regs[1]), '/');
               }
          }

          foreach($rules as $rule) {
           // check if page is disallowed to us
           if(preg_match("/^$rule/", $urlParts['path']))
              return false;
           }
           // page is not disallowed return true;
           return true;
       }
```

4 Write a second file `testCrawler.php` test your crawler to see if it knows whether or not to crawl a particular URL with code like the following:

```
<?php

require_once("Crawler.php");

$boris = new Crawler("URLS.txt");
$boris->crawl();

?>
```

Put the URL's (1 per line) you want to test in a file named URLS.txt. Try testing some that you know it should succeed on and other you know should fail. You might consider using PHPUnit to build a more fulsome test.

5 You should see output similar to Figure 23.1 that says whether or not the crawler was allowed to crawl the URL. Your crawler is now ready to start scarping URLs.



Crawling http://funwebdev.com/wordpress/wp-admin/

Downloading http://funwebdev.com/robots.txt
http://funwebdev.com/wordpress/wp-admin/ is NOT allowed

Crawling http://funwebdev.com/

Downloading http://funwebdev.com/robots.txt
http://funwebdev.com/ is allowed

Crawling http://dmoz.org/

Downloading http://dmoz.org/robots.txt
http://dmoz.org/ is allowed

Crawling http://facebook.com/

Downloading http://facebook.com/robots.txt
http://facebook.com/ is NOT allowed

Figure 23.1 – Exercise 23.1 complete

Exercise 23.2 —Scrape Out URLs

1 The crawler from Exercise 1 needs to be able to identify URLs from pages it is requesting. This exercise enhances the last one by adding a scraping component to your crawler. Copy your completed `lab23-exercise01.php` to `lab23-exercise02.php` to begin.

2 Replace the stub in your class with the following function:

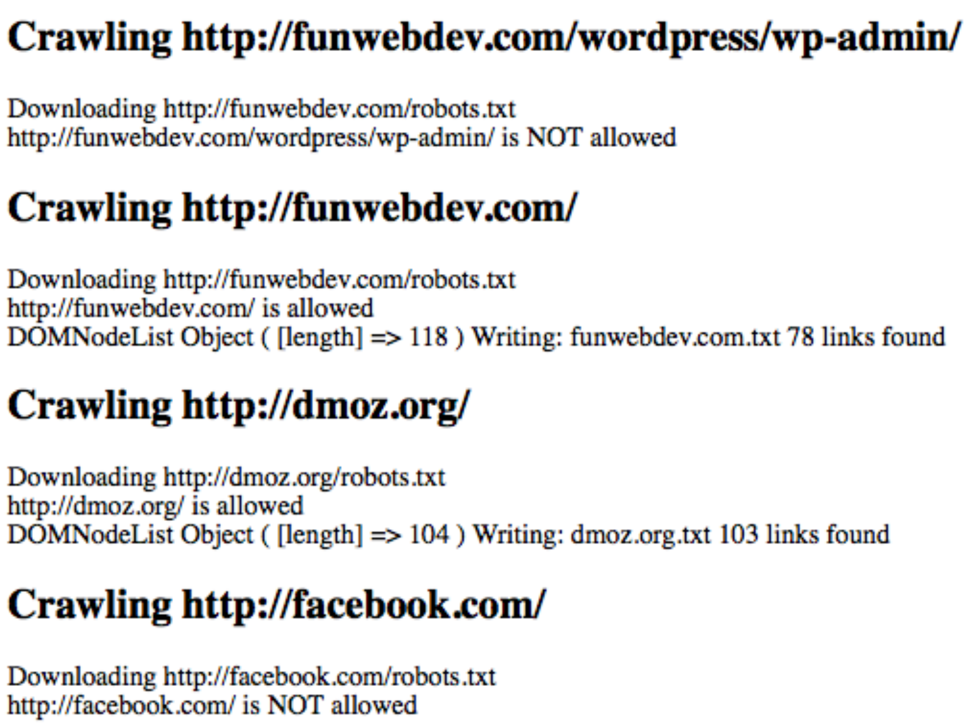
```
public function scrapeHyperlinks($url){
        $DOM = new DOMDocument();
        $contents = file_get_contents($url);
        $DOM->loadHTML($contents);
        $aTags = $DOM->getElementsByTagName("a");
        print_r($aTags);
        $i=0;
        $newLinks = array();
        foreach($aTags as $link){
                $newLinks[] = $link->getAttribute('href');
                $i++;
        }
        $urlParts = parse_url($url);
```

```
                $newLinks = array_unique($newLinks);

                echo " ".count($newLinks)." links found";
        }
```

3 To enhance your crawler consider adding code at the end of your function to print out all the links that were captured to a file (that can subsequently be crawled).

```
echo "Writing: ".$urlParts["host"].".txt";
file_put_contents($urlParts["host"].".txt",implode("\n",$newLinks));
```

4 Now your crawler, when run should output the count of links as shown in Figure 23.2, and the files should contain all the links.



```
Crawling http://funwebdev.com/wordpress/wp-admin/

Downloading http://funwebdev.com/robots.txt
http://funwebdev.com/wordpress/wp-admin/ is NOT allowed

Crawling http://funwebdev.com/

Downloading http://funwebdev.com/robots.txt
http://funwebdev.com/ is allowed
DOMNodeList Object ( [length] => 118 ) Writing: funwebdev.com.txt 78 links found

Crawling http://dmoz.org/

Downloading http://dmoz.org/robots.txt
http://dmoz.org/ is allowed
DOMNodeList Object ( [length] => 104 ) Writing: dmoz.org.txt 103 links found

Crawling http://facebook.com/

Downloading http://facebook.com/robots.txt
http://facebook.com/ is NOT allowed
```

Figure 23.2 Output from Exercise 23.2

5 In reality you need a more sophisticated architecture to manage links and URLs, since they are normally grouped together by URL and crawled in a distributed manner. Consider designing a distributed, polite crawling system if you decide to build a search engine.

SEARCH ENGINE OPTIMIZATION

Exercise 23.3 — RAGERANK CALCULATION

1 Calculate by hand the PageRank of the 5 Pages shown in Figure 23.3 assuming an initial weight of 1 each.

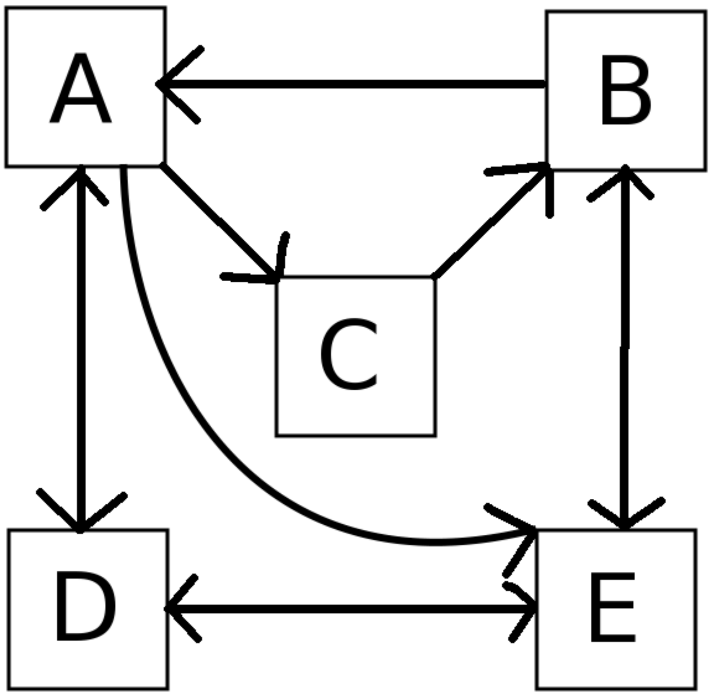


Figure 23.3 Links between 5 pages (each with an initial rank of 1

2 First, determine how many links out each page has.

$N_A=3$

$N_B=2$

$N_C=1$

N_D=2

N_E=2

3 Now calculate the 1st iteration of page rank.

$PR(A) = PR(D) / N_D + PR(B) / N_B$

$PR(A) = 1 / 2 + 1/ 2$

$PR(A) = 1$

$PR(B) = PR(C) / N_C + PR(E) / N_E$

$PR(B) = 1 / 1 + 1/ 2$

$PR(B) = 1\ 1/2$

$PR(C) = PR(A) / N_A$

$PR(C) = 1 / 3$

$PR(D) = PR(A) / N_A + PR(E) / N_E$

$PR(D) = 1 / 3 + 1/ 2$

$PR(D) = 5 / 6$

$PR(E) = PR(A) / N_A + PR(B) / N_B + PR(D) / N_D$

$PR(E) = 1 / 3 + 1 / 2 + 1/ 2$

$PR(E) = 1\ 1/3$

4 Complete the exercise for a second iteration, show your work, and confirm with your instructor that you have the correct answer.

.

Exercise 23.4 — =Write a PageRank Calculator

1 Create a file named `lab23-exercise04.php` where you will create code to calculate the page rank based and validate your solution to 23.3.

2 In that file, start by representing each page as nothing more than a list of links out. Considering the diagram in Figure 23.1, that would be

```
//Represent all the links through lists of lists.
```

```
$links = array("A"=> array("C","D","E"),
       "B" => array("A","E"),
       "C" => array("B"),
       "D" => array("A","E"),
       "E" => array("B","D"));
```

3 Build an array of backlinks from the set of links and print it out.

```
//build Backlinks list.
$backlinks=array();
foreach($links as $site => $linksOut){
  foreach($links as $blSite => $backL){
     foreach($backL as $L){
       if($L==$site){
    $backlinks[$L][]=$blSite;
       }
     }
  }
}
```

print_r($backlinks);

This list will contain a map of which pages link to which pages. With only 5 it's still relatively easy to visualize:

```
 Array
(
    [A] => Array
        (
            [0] => B
            [1] => D
        )

    [B] => Array
        (
            [0] => C
            [1] => E
        )

    [C] => Array
        (
            [0] => A
        )

    [D] => Array
        (
            [0] => A
            [1] => E
        )

    [E] => Array
        (
                [0] => A
                [1] => B
                [2] => D
            )

    )
```

4 Now create variables to hold the current and tempoary set of pageRanks.

```
//pagerankArrays;
$pageRank = array("A"=>1,"B"=>1,"C"=>1,"D"=>1,"E"=>1);
$tempPageRank= array("A"=>1,"B"=>1,"C"=>1,"D"=>1,"E"=>1);
```

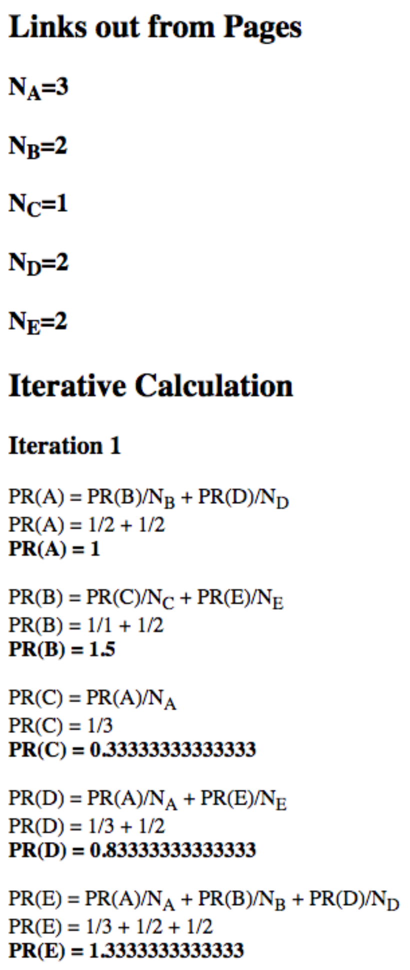
5 Build another array containing the number of backlinks to each Page.

```
//first output the number of links in each. (and store)
$linksFromPage = array();
echo "<h2>Links out from Pages</h2>";
foreach($links as $site => $linksOut){
       echo "<h3>N<sub>$site</sub>=".count($linksOut)."</he>";
       $linksFromPage[$site]=count($linksOut);
}
```

6
4 Write a loop where inside you use the current page ranks to determine the future page ranks using the PageRank algorithm as follows.

```
$pr=0;
      foreach($backlinks[$site] as $BL){
    if($i>0) echo " + ";
        echo $pageRank[$BL]."/".$linksFromPage[$BL];
    $pr+=$pageRank[$BL]/$linksFromPage[$BL];
    $i++;
      }
```

7 Combine everything together to make a script that outputs the pagerank and related information as shown in Figure 23.4.



Links out from Pages

N_A=3

N_B=2

N_C=1

N_D=2

N_E=2

Iterative Calculation

Iteration 1

PR(A) = PR(B)/N_B + PR(D)/N_D
PR(A) = 1/2 + 1/2
PR(A) = 1

PR(B) = PR(C)/N_C + PR(E)/N_E
PR(B) = 1/1 + 1/2
PR(B) = 1.5

PR(C) = PR(A)/N_A
PR(C) = 1/3
PR(C) = 0.33333333333333

PR(D) = PR(A)/N_A + PR(E)/N_E
PR(D) = 1/3 + 1/2
PR(D) = 0.83333333333333

PR(E) = PR(A)/N_A + PR(B)/N_B + PR(D)/N_D
PR(E) = 1/3 + 1/2 + 1/2
PR(E) = 1.3333333333333

Figure 23.4 Output froma PageRank Calculator

Exercise 23.5 — Build a Site Map

1 Building a sitemap is something that can easily be automated if you are capturing proper logs for a real site with potentially thousands of links. If you have access to server logs, consider generating a sitemap using the Google's generator: https://code.google.com/p/googlesitemapgenerator/

Alternatively, if you are using WordPress try to generate a sitemap suing a freely available plugin.

If you would rather learn to make sitemaps manually, continue with this exercise.

2 Create a file named sitemap.xml and start by pasting in the following markup:

```
<?xml version="1.0" encoding="utf-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
</urlset>
```

3 Make a list of all the important URLs on your site and for each one add a node to your sitemap as follows (one example shown in red).

```
<?xml version="1.0" encoding="utf-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
   <url>
     <loc>http://funwebdev.com/</loc>
</url>
   <url>
     <loc>http://funwebdev.com/samples/chapters/</loc>
  </url>
</urlset>
```

4 Now for each URL consider the following attributes of it:

1. When was it last updated?
2. How often should it be recrawled?
3. How important is it (on a scale of 0-1)?

5 Now, using the answer to those questions fill in the additional attributes for each url as shown in our example below.

```
<?xml version="1.0" encoding="utf-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
   <url>
     <loc>http://funwebdev.com/</loc>
     <lastmod>2314-06-29</lastmod>
     <changefreq>weekly</changefreq>
     <priority>1.0</priority>
   </url>
   <url>
     <loc>http://funwebdev.com/samples/chapters/</loc>
     <lastmod>2314-04-29</lastmod>
     <changefreq>monthly</changefreq>
     <priority>0.5</priority>
  </url>
</urlset>
```

6 Finally, gzip the archive and place it on the root of your webserver. This archive will now potentially be used by search engines (and can be managed through various webmaster tools

for each search engine).

Exercise 23.6 — Levenshtein distance

1 Levenshtein distance is defined as follows.

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

2 Fill in the Levenshtein distance matrix for comparing help to hello.

		h	e	l	p
h					
e					
l					
l					
o					

3 Fill in the Levenshtein distance matrix for comparing **careful** to **careless**. Articulate the Levenshtein distance between the words.

		c	a	r	e	f	u	l
c								
a								
r								
e								
l								
e								
s								
s								

Exercise 23.7 — Levenshtein Calculator

1 Open the provided lab23-exerice07.php. Note that it compares an input strings against an array of potential matches and outputs the string with the shortest Levenshtein distance.

2 Will in the function fullLev(s,t) so that it outputs a table showing the Levenshetin calculation much like that from the previous exercise, done by hand.