CSCI 340
Data Structures and Algorithms
Spring 2019
Project 1 – PageRank

Due: Tuesday, February 19, at the beginning of class.

Points: 30 points

Background
This section is borrowed in a large part from Wikipedia and from the book, *Nine Algorithms that Changed the Future*, by John MacCormick.

Google began in 1996 as a research project by Larry Page and Sergey Brin, Ph.D. students at Stanford University. As a research project, it was hosted on the university servers. It very quickly outgrew the bandwidth that the university could supply and in 1998 incorporated as a company and moved to a garage in Menlo Park. That same year it was named by *PC Magazine* as one of the top 100 websites for 1998. According to *PC Magazine*, Google's elite status was awarded for its "uncanny knack for returning extremely relevant results." Search engines weren't new, Lycos, AltaVista, Hotbot, Yahoo and Excite had all been around for years, but Google very quickly overtook them. Their innovative algorithm for ranking search results was a major reason.

The name PageRank is a play on words, since it ranks web pages, but is also the ranking algorithm of Larry Page, its chief inventor. Page rank refers to the search engine's ability to place the most relevant pages at the top of the search results. For example, your instructor recently did a Google search for Erica Eddy. The search returned 9,240,000 hits, but Google was smart enough to place our Ms. Eddy's homepage as #1 out of all of them. That's pretty impressive.

The PageRank algorithm assigns every page a numeric value, or rank. The search results are sorted in descending order based on the rank, and the list is returned. The key to success is making the best rank assignments possible.

Here's how the algorithm works:
We assign every page, p, an initial rank, PR(p) = 1/N, where N is the number of web pages. We then iteratively go through all the pages updating their ranks based on the following formula:

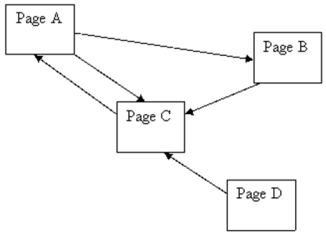
$$PR(p) = \frac{1-d}{N} + d \sum_{p' \in M(p)} \frac{PR(p')}{L(p')}$$

PR(p) = the page rank of page p
d = a small numeric constant. We will use d = 0.15
N = the number of pages
M(p) = the set of all pages that refer to p
PR(p') = the page rank of p', where p' ϵ M(p), that is, p' has a link to p
L(p') = the total number of outgoing links from page p'

This formula looks a lot more complicated than it is. The first term is a constant term that guarantees a minimum page rank for every page.

The second term is based on all the other pages that link to our page. So if 10 other pages contain links to our page, then each of them contributes to our page's new rank. Intuitively this makes sense, because if there are lots of links to our page, then our page is popular and should have a higher rank. Their contribution is based on their own page rank, PR(p') divided by their number of outgoing links. We multiply this result by d to get the overall contribution.

Let's consider a very simple example. Consider the following figure containing 4 pages:



To begin with we will assign all the pages the same rank. Since there are 4 of them, we will give each one a value of 0.25

PR(A) = 0.25
PR(B) = 0.25
PR(C) = 0.25
PR(D) = 0.25

We want to update each of them in turn.

new PR(A) = (1-0.15)/4 + 0.15 * PR(C) /1 = 0.2125 + 0.15 * 0.25 / 1 = 0.25

Note here that we only use PR(C) in the calculation, since C is the only page that links to A.

new PR(B) = (1-0.15)/4 + 0.15 * PR(A) /2 = 0.23145

First note that we use the old PR(A). Yes, I know it hasn't changed, but it is the old 0.25. The next iteration through the algorithm we will use the updated PR values. Note, too, that we divide PR(A) by 2, since A has 2 outgoing edges.

new PR(C) = (1-0.15)/4 + 0.15 * (PR(A)/2 + PR(B)/1 + PR(D)/1) = 0.30625

new PR(D) = (1-0.15)/4 = 0.2125
PR(D) has a minimal page rank, since no pages link to it.

After one iteration, page C has the highest page rank, which makes sense, since three other pages refer to it, so it is probably quite important.

One iteration through the process is not enough, because the page ranks will continue to change with more iterations. A second iteration gives the values:

PR(A) = 0.2584375
PR(B) = 0.23125
PR(C) = 0.2978125
PR(D) = 0.2125

After a few iterations the page ranks stabilize to the values:
PR(A) = 0.257266
PR(B) = 0.231795
PR(C) = 0.298439
PR(D) = 0.2125

If we run any more iterations, the values don't change.

> ***Aside #1:*** This isn't important to understanding or doing the assignment, but if you were in math class, we would do the re-ranking using a matrix multiplication, and the resulting stabilized values would be called the Eigenvalues of the matrix.

Your assignment:
You are to write a Java program named `PageRank.java` that implements the PageRank algorithm.

It may be fun to let you write your own web crawler to create the data to use, but it is beyond the scope of this project. Instead, your instructor has created three data files for you to test your code on. They are located in the course directory on the lab machines:
`/home/student/Classes/Cs340/PageRank/`
and on D2L.

`Simple.txt` contains data for the four web pages described above. This file is strictly for testing purposes.

CS.txt contains data for about 175 web pages on the old CS website.
`http://green.uwp.edu/departments/computer.science/`
This is the file against which your program will be graded.

`UWP.txt` contains data for 5000 pages from UW-Parkside's website. Note that this file is over 60 Megabytes, so will take a bit to load. This file is also strictly for testing purposes.

Each file contains data about a sequence of web pages. Each page's data is formatted as follows:
It begins with a line that says: PAGE
The next line contains the URL for the page.
The third line contains the text of the page. Note that this is NOT html. It is just the text of the page.
Beginning on the fourth line are a list of the URLs (outgoing links) on the page. Some of these outgoing links may be broken. That is, the page they link to may not be part of the data set.

After the last URL for the current page, there will either be a line saying PAGE for the next webpage, or the end of the file.
For example:

```
PAGE
http://A.html
My dog has fleas.
http://B.html
http://C.html
PAGE
http://B.html
My cat has hairballs.
http://C.html
PAGE
http://C.html
My dog has earmites, but my cat has a tapeworm.
http://A.html
PAGE
http://D.html
I do not like pets.
http://C.html
```

Notes on processing the file:

1. You may hardcode the name of the data file ("`CS.txt`") into your program, but make certain that the string is the correct one to make your code run in the lab. E.g. if the string contains "C:\" it is not going to run in the lab. It is also okay to ask the user to enter the name of the data file.
2. Read the file in creating an appropriate data structures for it. While it is certainly up to you to decide what structure you want, a HashMap is probably a good choice. Use the page's URL as the key. The value is an object containing all the information for the web page.
    a. Again, it is up to you, but storing the text of the file as a HashSet of words, all in lowercase is probably a good idea. The easiest way to do this is to read the text in as a line, make it into lowercase then split it into an array of individual words, each of which gets added to the set.
    b. The links from a web page may also be stored as a HashSet of Strings.

After you have read in the file, calculate the page rank for every page.

Prompt the use to enter key words you wish to search for.

Locate all pages that contain these words and output up to 20 of them in descending page rank order. Include the page rank value you calculated with each. Make your search independent of the case of the words. The words do not need to be adjacent or in the same order on the page.

Here are the first few lines of a sample run of your instructor's code:

```
Enter your search terms:
communications
There were 7 hits
Rank      URL
```

```
0.0111313 http://green.uwp.edu/departments/computer.science/faq/faqdisplay.cfm
0.0110998 http://green.uwp.edu/departments/computer.science/faq/faqdisplay.cfm?category=email
0.0081621 http://green.uwp.edu/departments/computer.science/faq/faqdisplay.cfm?id=
0.0068788 http://green.uwp.edu/departments/computer.science/faq/faqdisplay.cfm?id=29
0.0068429 http://green.uwp.edu/departments/computer.science/cis/curriculum.cfm
0.0068429 http://green.uwp.edu/departments/computer.science/cis/preregs.cfm
0.0067819 http://green.uwp.edu/departments/computer.science/certs/cybersecurity/index.cfm
```

I haven't used this assignment for several years, and at some point I cleaned up CS.txt. I believe I eliminated the broken links, but I didn't record what I actually did. In any case, I reran this assignment with the CS.txt I gave you and came up with these results.

```
Enter your search terms
communications
The number if hits was: 7
0.0111904 http://green.uwp.edu/departments/computer.science/faq/faqdisplay.cfm
0.0111594
http://green.uwp.edu/departments/computer.science/faq/faqdisplay.cfm?category=email
0.0082326 http://green.uwp.edu/departments/computer.science/faq/faqdisplay.cfm?id=
0.0069354 http://green.uwp.edu/departments/computer.science/faq/faqdisplay.cfm?id=29
0.0068972 http://green.uwp.edu/departments/computer.science/cis/curriculum.cfm
0.0068972 http://green.uwp.edu/departments/computer.science/cis/preregs.cfm
0.0068355 http://green.uwp.edu/departments/computer.science/certs/cybersecurity/index.cfm
```

This is not a long assignment. Your instructor's entire program, with good documentation only runs 167 lines. It is a complicated assignment, however, so start early. You will not complete it if you wait until the day before it is due.

Grading Considerations:
Your program will be graded on the lab machines.
Use good software engineering. Use LOTS of comments, good indentation, lots of methods, etc.
Do not use packages, or more precisely place all your code in the default package.

How to Submit:

Name your program `PageRank.java`. Do NOT put a package statement in your code. Submit your code on the lab machines. Do not submit an entire project, just your java file. E.g. At a linux prompt type:

```
submit 340 PageRank.java
```

Aside #2: This information is not needed to complete the project, but is included just for interest.

How does this project compare with reality? Google's page rank algorithm has evolved over the years. They now use around 200 criteria to rank pages. A few of these are:

1. Ranking the page higher if the keyword is part of the page title.
2. Ranking the page higher if the keyword is part of the page's meta-description.
3. Ranking the page higher if the keywords appear close to each other in the text.

Why should we care? It turns out that there are companies that specialize in helping others improve their pages' rankings. This is because a better ranking generally means more people will click through to your site and this translates to higher sales.