ITCS 414 – Information Retrieval and Storage

Project 2

PageRank: Towards Ranking the WWW Documents

Deadline: Monday October 30, 2017 11:55pm

Update Logs

09/23/2017	Initial Version

1. GOAL & IMPORTANT NOTE

In this project, you will learn how PageRank algorithm works and implement a basic version of PageRank algorithm. Note that you are allowed to form a new team for this project, as long as the number of members does not exceed 3.

2. PAGE RANK ALGORITHM

In this project, you will develop an understanding on another non-query ranking algorithm used by Google search engine to rank websites based on their "importance". PageRank was named after Larry Page, a Google co-founder, and has been well studied in the literature. Variants of PageRank algorithm have been proposed to improve upon the existing ones and/or to extend the capability to handle different kinds of graph-like data (social networks, scientific papers, patents, etc.).

Conceptually, PageRank works by counting the number and quality of links to a particular page, with an assumption that more important websites tend to have more incoming links from other websites. In this project, you will be implementing a basic variant of PageRank algorithm, whose pseudo-code is given below:

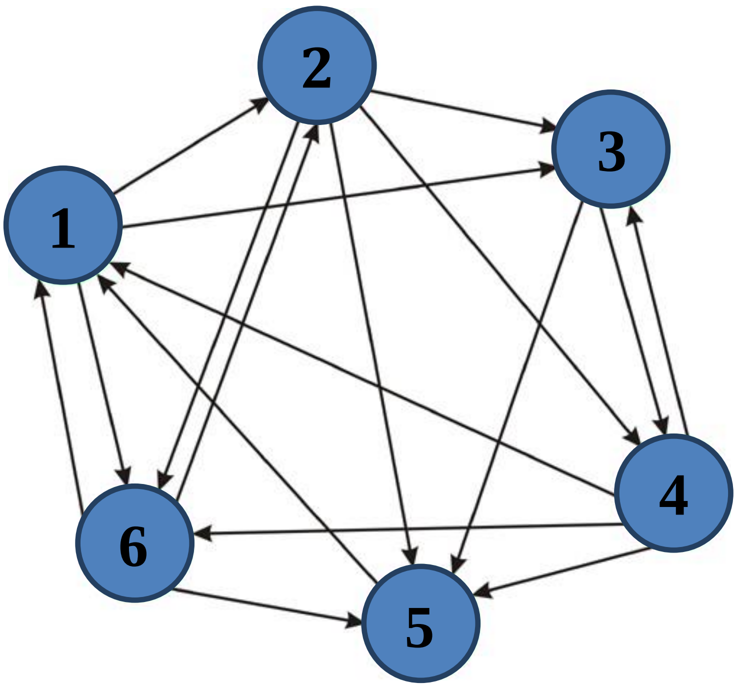
```
// P is the set of all pages; |P| = N
// S is the set of sink nodes, i.e., pages that have no out links
// M(p) is the set of pages that link to page p
// L(q) is the number of out-links from page q
// d is the PageRank damping/teleportation factor; use d = 0.85 as is typical

foreach page p in P
  PR(p) = 1/N                        /* initial value */

while PageRank has not converged do
  sinkPR = 0
  foreach page p in S                /* calculate total sink PR */
    sinkPR += PR(p)
  foreach page p in P
    newPR(p) = (1-d)/N               /* teleportation */
    newPR(p) += d*sinkPR/N           /* spread remaining sink PR evenly */
    foreach page q in M(p)           /* pages pointing to p */
      newPR(p) += d*PR(q)/L(q)       /* add share of PageRank from in-links */
  foreach page p in P
    PR(p) = newPR(p)

return PR
```

As an example, consider the following graph:



This graph can be represented as follows:

```
1 4 5 6
2 1 6
3 1 2 4
4 2 3
5 2 3 4 6
6 1 2 4
```

Where the first line indicates that page 1 has incoming links ***from*** pages 4, 5, and 6, and so on. PageRank algorithm is an iterative process. Before the algorithm starts, all pages have the same initial value (i.e. 1/|N|). The algorithm then keeps updating the value of each page until the stopping criteria is met.

3. PROGRAM AND INTERFACE

Download the project package (p2_pagerank.7z) and test case (p2_testcase.7z) from My Courses. In the package, you will find `PageRanker.java` and `citeseer.dat`. You will implement the rest of the code in `PageRanker.java`, and use citeseer.dat to test your code and generate the result files for submission.

Specifications

Your task is to implement the version of PageRank algorithm described above. In particular, you are to implement the six skeleton methods in `PageRanker.java`:

```
public void loadData(String inputLinkFilename)
public void initialize()
public double getPerplexity()
public boolean isConverge()
public void runPageRank(String perplexityOutFilename, String
                       prOutFilename)
public Integer[] getRankedPages(int K)
```

The description of each method can be found in `PageRanker.java` . You can use any efficient data structure to store the graph and immediate variables necessary for the computation. Since your algorithm may be tested against a large graph of up to 1 million nodes, it is important that you keep efficiency of the code and algorithm in mind. Your algorithm should run reasonably fast and avoid redundant/repetitive computation.

Stopping criteria

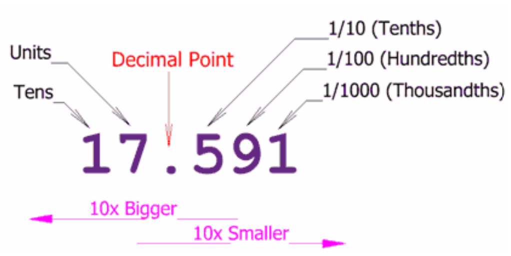
Run your iterative version of PageRank algorithm until your PageRank scores "converge". To test for convergence, calculate the ***perplexity*** of the PageRank score distribution, where perplexity is simply 2 raised to the (Shannon) entropy of the PageRank distribution:

$$perplexity(PR) = 2^{H(PR)} = 2^{-\sum_{p \in P} PR(p) \cdot log_2 PR(p)}$$

Where *P* is the set of all pages, and *PR(p)* is the current PageRank score of page *p*.

Perplexity is a measure of how "skewed" a distribution is -- the more "skewed" (i.e., less uniform) a distribution is, the lower its perplexity. Informally, you can think of perplexity as measuring the number of elements that have a "reasonably large" probability weight; technically, the perplexity of a distribution with entropy *h* is the number of elements *n* such that a uniform distribution over *n* elements would also have entropy *h*. (Hence, both distributions would be equally "unpredictable".)

Run your iterative PageRank algorithm, and output the perplexity of your PageRank distribution until the perplexity value no longer changes in the **units position** for ***at least four iterations***. (The units position is the position just to the left of the decimal point. See an illustration below.)



Implementation

You are allowed to use built-in Java libraries (e.g., `java.util.*`, `java.math.*`, `java.io.*`, etc.); however, you are not allowed to use third-party libraries. Your code should compile without having to install additional components.

4. DELIVERABLES

You must run your PageRank algorithm using the given input file `citeseer.dat`. Upon termination of the algorithm, generate two output files:

`perplexity.out` (lists the perplexity score after each iteration)
`pr_scores.out` (lists the PageRank score for each page)

Write your team members' names, IDs, and sections in a comment section on top of each java file. You will be penalized for any submitted code whose ownership cannot be determined. Do not write names or comments in any output files.

5. SUBMISSIONS

It is important that you follow the submission instructions to enable your code and output files to be graded by the auto grader. Failing to do so may result in a deduction and/or fault evaluation of your project.

1. Create a new directory and name it `P2_<id1>_<id2>_<id3>`, e.g. `P2_5888111_5888222_5888333`. Let's call this the submission directory.
2. Place `perplexity.out`, `pr_scores.out`, and `PageRanker.java` in the submission directory.
3. Use ***7zip*** to compress the submission directory and produce the ***submission package,*** e.g. `P2_5888111_5888222_5888333.7z`. Do not submit .zip or .rar files.
4. One of your team members then submits the submission package on My Courses.

6. GRADING

Students are expected to maintain standards of academic honesty and integrity. Violations of academic honesty and integrity in this assignment are unacceptable. Each group must work on their own project independently. A full or partial copy from other people's works automatically get zero and may result in a failing grade for this course.

Your code will be tested against an auto-grader with multiple test cases. 80% of the scores will be given towards the correctness of the output. The other 20% will be given based on the correctness, efficiency, and organization (including the ease of understanding and comments) of your code.

Late submission will suffer a penalty of 20% deduction of the actual scores for each late day. You can keep resubmitting your solutions, but the only latest version will be graded. *Even one second late is considered late.*

Extra credit (up to additional 10%):

Option01: PageRank Mini Literature Review

During the course of completing this project, you have learned how web search engines (such as Google, Bing, Yahoo, etc.) may use the PageRank algorithm to calculate the importance measure for each webpage. To get up to 10% bonus, find a research paper that is indexed by a reputable publisher (e.g. ACM, IEEE, AAAI, Springer, Elsevier) that either presents an extension, an application, an alternative, or the combination thereof, of the PageRank algorithm. The length of the paper must be at least 10 pages double column, or 18 pages single column. Then,

1. Summarize and explain the paper within 1 paragraph (not more than 1 page). You can assume that the readers will have some IT background, but not an expert in IR. So the language that you use must not be too technical, and avoid too many jargons.
2. Answer the following question. How could you apply the technique(s) you read in the paper to a novel problem that has not been solved before. Make sure that you first describe what the problem is and why this problem is worth solving, before suggesting how such a problem could be solved using the methods presented in the paper.

Don't forget to put every member's name and ID on top of the report. Submit the report as a PDF file along with the submission package. Your report will be submitted to TurnItIn for similarity check. If your similarity index is >= 30%, you will be disqualified from this extra credit option.

Option02: PageRank Application

Though the original intent of PageRank algorithm is to quantify importance of web pages, its applications are limitless. If you choose to pursue this extra credit option:

1. Find a different graph-like dataset (i.e. web links datasets are not allowed). Your dataset should be reasonably large. So
2. Apply PageRank algorithm to the dataset. Collect results.
3. Analyze the results. Do not just copy/paste results on the report.
4. In the report:
    a. Explain how you can apply PageRank algorithm to the dataset.
    b. Explain what PageRank scores mean in terms of your dataset. I.e. What does it mean for a node to be more "important" than the other.
    c. Explain how you set up your dataset so that PageRank is applicable. (i.e. what is a node? what is an edge?) If you need to modify the current PageRank implementation (such as allowing edge weights, heterogenous edge types, etc.), explain your extra features here.
    d. Report the algorithm statistics and important results.
    e. Analyze the results. Provide some non-trivial insight.

Submit the extra credit report and the dataset along with the submission. If the dataset is huge, submit a downloadable link. Don't forget to put every member's name and ID on top of the report. Submit the report as a PDF file along with the submission package. Your report will be submitted to TurnItIn for similarity check. If your similarity index is >= 30%, you will be disqualified from this extra credit option.

Bug Report and Specs Clarification:

Though not likely, it is possible that our solutions may contain bugs. A bug in this context is not an insect, but error in the solution code that we implemented to generate the test cases. Hence, if you believe that your implementation is correct, yet yields different results, please contact us immediately so proper actions can be taken. Likewise, if the project specs contain instructions that are ambiguous, please notify us immediately.

Need help with the project?

If you have questions about the project, please first post them on the forum on My Courses, so that other students with similar questions can benefit from the discussions. If you still have questions or concerns, please come see one of the instructors or TAs during the office hours, or make appointments in advance. We do not debug your code via email. If you need help with debugging your code, please come see us. Consulting ideas among friends is encouraged; however, the code must be written by members in your own team without looking at other teams' code. (See next section.)

Academic Integrity

Don't get bored about these warnings yet. But please, please do your own work. Though students are allowed and encouraged to discuss ideas with others, **<u>the actual solutions must be written by themselves without being dictated or looking at others' code</u>**. Inter-team collaboration in writing solutions is not allowed, as it would be unfair to other teams. It is better to submit a broken program that is a result of your own effort than taking somebody else's work for your own credit! Students who know how to obtain the solutions are encouraged to help others by guiding them and teaching them the core material needed to complete the project, rather than giving away the solutions. **You can't keep helping your friends forever, so you would do them a favor by allowing them to be better problem solvers and life-long learners. ** Writing code is like writing an essay. If each of you writes your own code, then there is almost no chance that your code will appear similar to the others. Your code will be compared with other students' and online sources using state-of-the-art source-code similarity detection algorithms. If you get caught cheating, serious actions will be taken!

Frequent Qs/As

Q: Does the graph generated from the citeseer dataset have 716,800 pages (nodes)?

A: There can be more or fewer than 716,800 nodes. In the loadData(), you have to parse the file and find out exactly how many nodes are in the graph. Node IDs are not necessary increments of 1, and starting from 1. That is, if your graph has 3 nodes, for example, it does not necessarily means that the node IDs must be 1, 2, and 3. They could be 342, 8, and 908.

You cannot use the number of lines in the data file to determine the number of nodes, for a number of reasons. For example, there can be duplicate lines in the data file. Furthermore, if your graph file only contains one line:

4 6 7 8

Does this mean that the graph only has one node?

Q: Can you give some hint on the citeseer dataset?

A: As a last hint to help you debug, the perplexity score after the first iteration of the PR algorithm on the citeseer data should be around 657474.141528741

Note that your code will also be tested against other datasets, so please make sure it runs reasonably fast since efficiency is one of the criteria. (E.g. Running PR on the citeseer data should not take longer than 40 seconds to converge.)