# Project 1: PageRank algorithm (Due Sep. 11 11:59 p.m.)

# Deliverables

You are required to turn in the following items in a zip file (**username\_HadoopPageRank.zip**) in this assignment:

1. The source code of Hadoop PageRank you implemented.
2. Technical report (**username\_HadoopPageRank\_report.docx**) that contains:
   1. The description of the main steps and data flow in your program.
   2. The output file (**username\_HadoopPageRank\_output.txt**) which contains the first 10 URLs along with their ranks.

Points will be reduced (maximum 1 point) if the filename or directory structure are different from instructed above.

# Evaluation

The total points for Project #1 is 10, where the distribution is as follows:

* 1. Completeness of your code and output (6 points)
  2. Correctness of written report (3 points)
  3. Readability and clarity of README.txt (1 point)

# PageRank Introduction

## PageRank

The web search engine is a typical distributed system on the Internet. It is designed to search for information on the World Wide Web. Search results are generally presented in a list of results and are often called hits. PageRank is a well-known web graph ranking algorithm that helps Internet users sort hits by their importance.

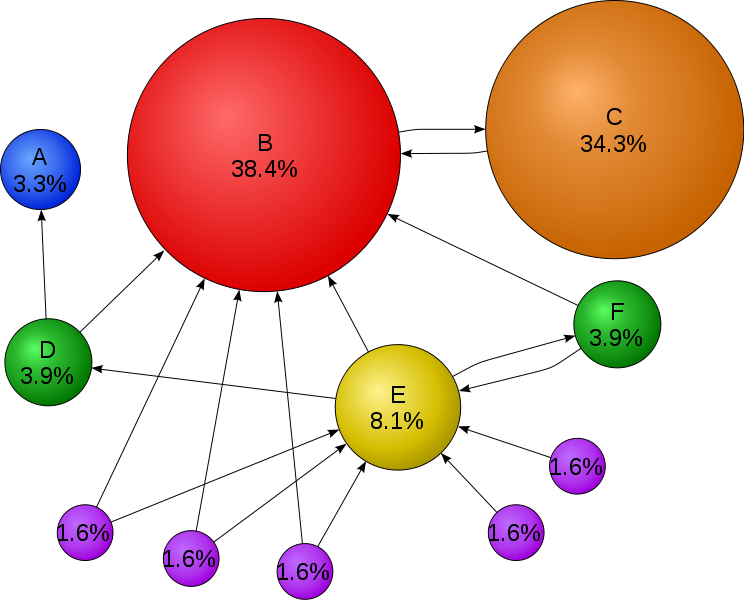
PageRank calculates a numerical value for each element of a hyperlinked set of webpages, which reflects the probability that a random surfer will access that page. The process of PageRank can be understood as a Markov Chain which requires iterative calculations to converge. An iteration of PageRank calculates the new access probability for each webpage based on values calculated in the previous iteration. The process will repeat until the number of current iterations is bigger than predefined maximum iterations, or the Euclidian distance between rank values in two subsequent iterations is less than a predefined threshold that controls the accuracy of the output results. 

Fig.1 Mathematical PageRank for a simple network in Wikipedia

## Figure 1 shows a web graph consisting of 11 vertices {A, B, C, D, E, F, G1, G2, G3, G4, G5}. Each vertex refers to a unique webpage, and the directed edge means there is one link from the source webpage to the target webpage. The percentage on each vertex represents the rank value of each webpage.

## Notes:

You can implement a sequential PageRank that can run on desktops or laptops. But when processing larger input data, like web graphs containing more than a million webpages, you need to run the PageRank application in parallel so that it can aggregate the computing power of multiple compute nodes. Currently, in both industry and academia, the study of large-scale web or social graphs has become increasingly popular. In one published paper, the job execution engines that claim to support large-scale PageRank include: MPI, Hadoop, Dryad, Twister, Pregel.

In project #1, you need implement the sequential version of PageRank. In a later project, you will implement a parallel version of PageRank by using the programming interfaces of MPI and Hadoop MapReduce job execution engine.

## Formula

Eqn.1 is the formula to calculate the rank value for each webpage. We will learn this formula by applying it to the case in Fig.1. There are 11 webpages in Fig.1, which include: {A, B, C, D, E, F, G1, G2, G3, G4, G5}. Assuming the probability distribution for a web surfer accessing all these 11 pages in current iteration is {PR(A), PR(B), PR(C), … PR(G5)}, then the probability for the surfer to access Page B in the next iteration is:   
PR(B) = PR(D)/2 + PR(E)/3 + PR(F)/2 + PR(C) + PR(G1)/2 + PR(G2)/2 + PR(G3)/2   
In general, the PageRank value for any page u can be expressed as:

Eqn.1:

The vertices seen in the right of the formula contain all the webpages that point to target webpage ‘u’. L(v) refers to the out degree of each webpage in the vertices set. The initial rank values of each webpage, like PR’(u), can be any double value. After several iteration calculations, the rank values converge to the stationary distribution regardless of what their initial values are (Markov Chain [2]).

## Damping factor

The PageRank theory holds that even an imaginary surfer who is randomly clicking on links will eventually stop clicking. The probability, at any step, that the person will continue is a damping factor **d**. Various studies have tested different damping factors, but it is generally assumed that the damping factor will be around 0.85. The formula considering damping factor is shown in Eqn.2. N refers to the total number of unique URLs.

Eqn.2:

# Execution guide for sample PageRank code

To help you understand the process of PageRank computation better, we offer an executable PageRank java class file in an assignment package. Its usage is:

**Java SequentialPageRank [input file name] [output file name] [iteration count] [damping factor]**

We also provide a small PageRank input file which is based on the web graph shown in Fig.1. You can evaluate the rank values in Fig. 1 by running SequentialPageRank with the following parameters.

**Java SequentialPageRank pagerank.input pagerank.output 10 0.85**

To check the correctness of your output rank values, you can open the pagerank.output file and see whether the output results are the same as the rank values in Fig. 1. Generally speaking, the larger the number of iterations, the more accurate the computed rank values will be.

# Programming guide for PageRank

## Input Data format

The input data for PageRank application is the web graph in adjacency matrix format [3]. In our sample program, it transfers the web graph into a simplified adjacency matrix. Following are the steps we used to construct an adjacency matrix for the web graph in Fig.1:

1. Construct a set of tuples that describe the web graph structure: WebG = {(A,null), (B, C), (C, B) (D, A, B), (E, B D F), (F, B E), (G1, B E), (G2, B E), (G3, B E), (G4, E), (G5, E)
2. Map letters to numbers. A->0, B->1, C->2, D->3, E->4, F->5, G1->6, G2->7, G3->8, G4->9, G5->10
3. Construct the simplified adjacency matrix based on information in step 1,2.

0

1 2

2 1

3 0 1

4 1 3 5

5 1 4

6 1 4

7 1 4

8 1 4

9 4

10 4

In the program, you are not supposed to construct the adjacency matrix from the web graph yourself. Instead, we will provide each group with different adjacency matrix input files (pagerank.input.1000.urls.groupid) which contains one thousand unique URLs. Besides, you can use HashMap<Integer, ArrayList<Integer>> to store the adjacency matrix.

## Implementation

You should use Java to implement the algorithm. There is a SequentialPageRank.java file included in the project. You should fill the empty methods listed in the Java file.

## Rank values table

In the program, you need a rank value table that stores intermediate rank values within one iteration. The process of PageRank [4] computation is the process of updating the rank values table by applying to Eqn.2. The number of elements in the rank values table is the number of unique URLs in the web graph you will study, which is 11 in Fig.1.

## Output

Final ranks are sorted in decreasing order and URLs are stored in an output file. Number of Iterations and top 10 URLs are displayed on the screen.

# References

1. Sergey Brin and Lawrence Page, [The Anatomy of a Large-Scale Hypertextual Web Search Engine](http://infolab.stanford.edu/~backrub/google.html), Stanford University, WWW7 Proceedings of the seventh international conference on World Wide Web 7 , 1998
2. <http://en.wikipedia.org/wiki/Markov_chain>
3. <http://en.wikipedia.org/wiki/Adjacency_matrix>
4. <http://en.wikipedia.org/wiki/PageRank>