CS542200 Parallel Programming
Homework 5: PageRank

1st Due: 1/14, 2018
2nd Due (with 5 pts off): 1/ 17, 2018
Final Due (with 10 pts off): 1/ 21, 2018

1 GOAL

This assignment helps you get familiar with Apache Hadoop MapReduce API by implementing PageRank algorithm.

2 PROBLEM DESCRIPTION

In this assignment, you are required to implement PageRank algorithm using Hadoop. PageRank is a link analysis algorithm previously used by Google Search to rank websites in their search engine results, so that more important pages have higher ranks. It works by counting the **number** and **quality** of links to a page to determine a rough estimate of how important the website is. A page is considered more important if it has more incoming links and is also pointed by other important pages.
PageRank assigns a numerical weighting to each element E of the set of pages. The weighting is referred to as the *PageRank of* E, denoted by $PR(E)$.
We can think of pages as vertices, and links between pages as edges. A set of hyperlinked pages can form a directed multigraph.
Given page x with n direct predecessor $\{t_1, t_2, \dots, t_n\}$, then

$$PR(x) = (1-\alpha)\left(\frac{1}{N}\right) + \alpha \sum_{i=1}^{n} \frac{PR(t_i)}{C(t_i)} + \alpha \sum_{j=1}^{m} \frac{PR(d_j)}{N}$$

where

- $C(t)$ denotes the out degree of page t
- α (set to 0.85) is the *damping factor*, $(1-\alpha)$ is the probability of *random jump*
- N is the total number of pages (nodes)
- d denotes a *dangling node*, which has no outgoing links
- m is the total number of dangling nodes

We will use power iteration method to compute the correct weight for all pages. Let $PR^{(k)}(x)$ denotes the PageRank of x at k-th iteration. We define:

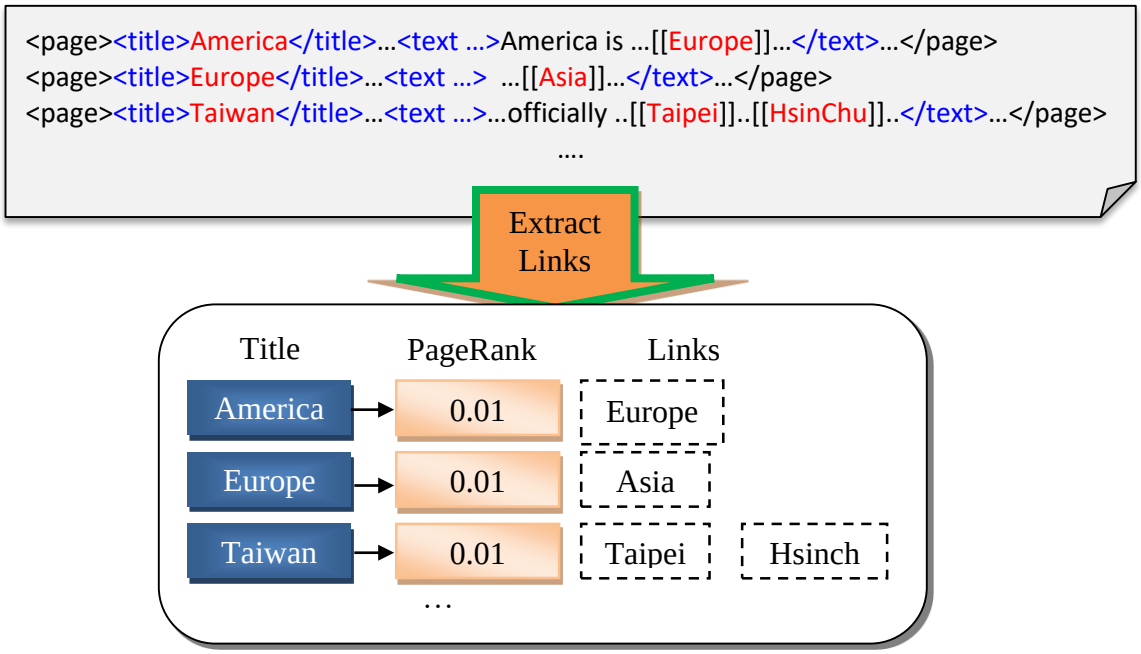
$$PR^{(0)}(x) = \frac{1}{N} \; \forall x$$

Then, we are able to compute PageRank iteratively as follows

$$PR^{(k)}(x) = (1-\alpha)\left(\frac{1}{N}\right) + \alpha \sum_{i=1}^{n} \frac{PR^{(k-1)}(t_i)}{C(t_i)} + \alpha \sum_{j=1}^{m} \frac{PR^{(k-1)}(d_j)}{N}$$

In order for you to understand this algorithm better, the execution flow of PageRank is illustrated step by step as below:

1. **[Build a graph]** First, you need to extract links in the input file and build a graph. (N = 100 for example)



- **Notice**: Remove missing links from the graph.
  In the example below, there is an out-link which points to nowhere! We need to remove it before moving on to the next step.

2. **[Calculate PageRank]** Initialize the PageRank of all pages to $1/N$ where N is total number of pages. Compute the PageRank of pages iteratively until the values *converge*.

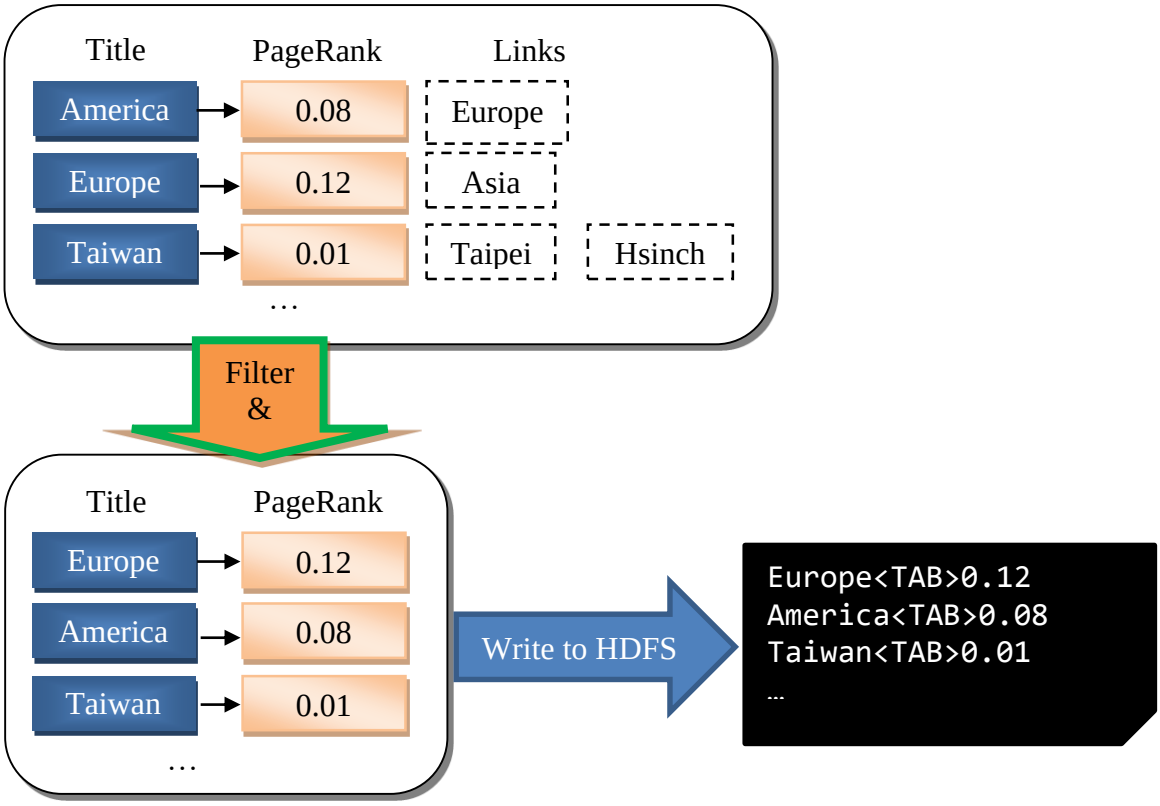
   Please use double precision numbers while calculating the PageRank weights, your result might otherwise be considered incorrect.

   The error between k-th and $(k-1)$-th iteration is defined as

$$err(k) = \sum_{i=1}^{N} \left| PR^{(k)}(x_i) - PR^{(k-1)}(x_i) \right|$$

   Iterate until $err(k) < 0.001$ (s i.e. convergence is assumed).

3. **[Sort the result]** Sort the resulting ranks in a descending order and filter the result so that only the page title and its PageRank weight are listed in the final output.



   If there are multiple pages and values with the same PageRank value, please sort them lexicographically in ascending order.

3 INPUT / OUTPUT FORMAT

The input/output format requirements are specified as follows:

1. Your programs are required to read an input file from HDFS, and generate output in another file.

2. Your programs accept 2 input parameters. They are:

   i、 (String) the input file name on HDFS

   ii、 (String) the output directory which contains part-xxxxx files

   iii、 (Integer) number of iterations. (Default: iterate until converges)

3. The input file on HDFS is a normal text file in wikitext format with multiple lines. Each line contains one page which is enclosed in **<page>** and **</page>**. There are only two attributes we need to consider: **page title** and **page links**.

   - Page title will be placed between "**<title>**" and "**</title>**". The first character of a title is always in upper case (no need to capitalize it).

     Since title is part of an XML text, the real title text need to be un-escaped as follows:

input string in title text	un-escaped character
<	<
>	>
&	&
"	"
'	'

     For example, the title string `Ulmus &apos;Nire-keyaki&apos;` need to be converted to `Ulmus 'Nire-keyaki'`

   - A link will be placed between "**[[**" and "**]]**", which defines what page it points to and the shown text of the link. But there are some exceptions which make it not so trivial to parse.

     To simplify the processing and be more specific, we define a link to another page as follows:

     - "**[[**" means what follows is a target page title.

- The page title is case-sensitive **except the first character**. (The first character of page titles is always in upper case)
- Only capitalize the first character if it's from a - z
- The first "**]]**", vertical bar "**|**" or sharp sign "**#**" it meets afterwards means the end of the target page title.
- Note that links which points to a nonexistent page is not considered a valid link.

For instance, all the following links point to the page titled "Texas":

A. `[[Texas]]`

B. `[[texas]]`

C. `[[Texas|Lone Star State]]`

D. `[[Texas#Geography]]`

But note that the text `[[TEXAS]]` does NOT link to "Texas" since the target name of a page link is case-sensitive.

Also, since links are also part of the XML text, please un-escape them using the method we mentioned in title string processing part.
Check ParseMapper.java for example.

4. The output on HDFS naturally consists of one or several part-xxxxx or part-r-xxxxx files which represent the total view of the final output when combined in order.

   The output should contain *N* lines where *N* is the total number of pages in the input file. Each line has the **page title** and the corresponding **PageRank value** separated by a Tab character. These lines are sorted by PageRank weights in descending order.

   Sample Output:

```
Europe<TAB>0.000473036896878
America<TAB>8.72925041381e-05
Taiwan<TAB>3.97693378405e-05
…
```

   Merge your final output with

```
$ hdfs dfs -getmerge {hdfs_output_dir} homework/HW5/{student ID}_{DataSize}.out
```

4 PROVIDED TEST CASES

- Input files are collected from wiki-dump, there are many types of links. If you like to learn more about that, you can refer to:
  - http://en.wikipedia.org/wiki/Wikipedia:Database_download
  - http://en.wikipedia.org/wiki/Wikipedia:Free_links#Free_links
- Input files and sample output files are placed in **`hdfs:/// user/ta/PageRank`**
- There are 4 sizes of input: 100M, 1G, 10G and 50G
- **Please do not copy large input test cases (10G, 50G) into the local disk, we will run out of disk space!**

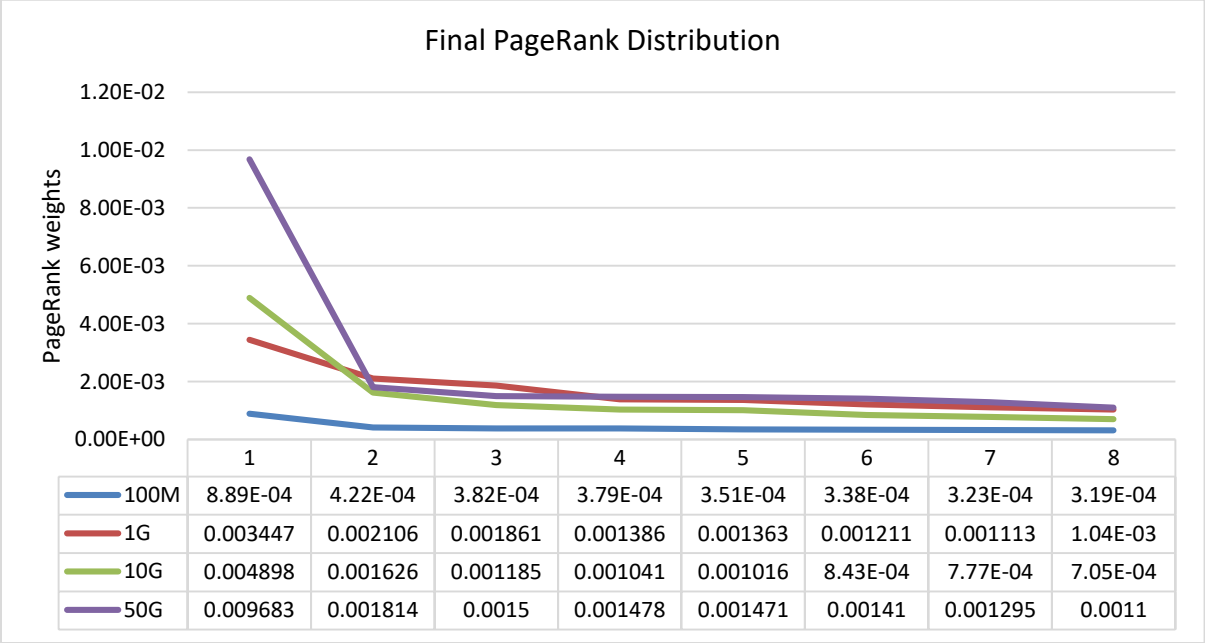
5 WORKING ITEMS

You are required to implement PageRank with *Hadoop MapReduce*.
Besides, you are required to write a report which contains all the following contents.

- **Title, name, student ID**
- **Instruction**
  Indicate how to compile & run your program.
- **Implementation**
  Describe your implementation **in detail** using diagrams, figures and sentences.
- **Experiment & Analysis**
  Try to use all the test cases as the input for all the following experiments.
  Please include the application ID for each of your successful experiments!
  You can look them up at
  http://140.114.91.183:19888/ and http://140.114.91.183:8088/

A. Analyze the distribution of PageRank weights

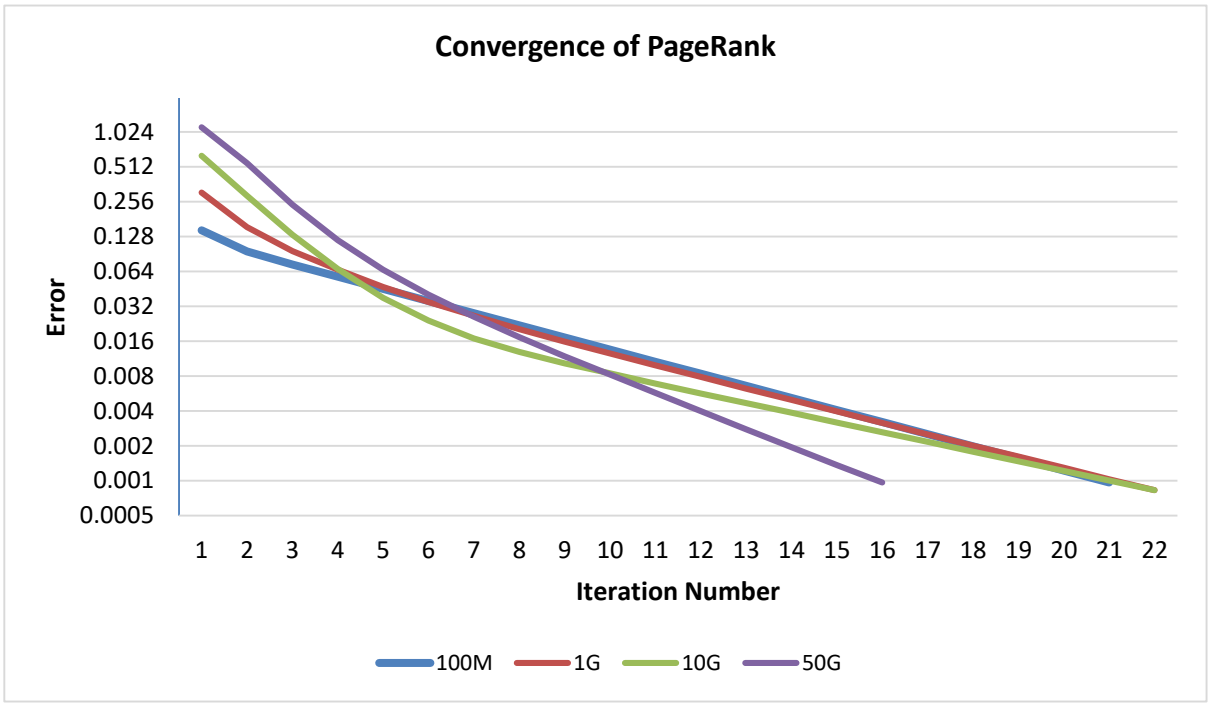
Draw a diagram to show your result. Like shown below:



	1	2	3	4	5	6	7	8
100M	8.89E-04	4.22E-04	3.82E-04	3.79E-04	3.51E-04	3.38E-04	3.23E-04	3.19E-04
1G	0.003447	0.002106	0.001861	0.001386	0.001363	0.001211	0.001113	1.04E-03
10G	0.004898	0.001626	0.001185	0.001041	0.001016	8.43E-04	7.77E-04	7.05E-04
50G	0.009683	0.001814	0.0015	0.001478	0.001471	0.00141	0.001295	0.0011

B. Analyze the converge rate

Draw a diagram to show how PageRank converges with number of iterations. Like shown below:



Please compute the error according this formula above.

C. **Performance analysis with different settings**
Compare the execution time in different number of reducers (1, 2, 4, 8, 16, 32), and the number of reducer is limit to 32. Briefly explain your result.

- **Experience & Conclusion**
- **Feedback (Optional)**
  What do you think about this assignment? Any feedback is welcome!

6 GRADING

1. **Correctness** (80%)

   i、 Hadoop MapReduce (80%)

   - Can handle 100M test case [20%]
   - Can handle 1G test case [20%]
   - Can handle 10G test case [20%]
   - Can handle 50G test case [20%]

2. **Report & Demo** (20%)

   - Run 10 iterations in Demo time.
   - Grading is based on your evaluation results, discussion and writing.

7 REMINDER

1. Please upload these files to our server

   - Source codes
   - Makefile
   - Execution scripts

   Make sure your compile script can execute correctly and your code has no compile error, and copy to **homework/HW5 directory under your home directory in our server before 1/14 (SUN) 23:59**

2. **HW5_{student ID}_report.pdf** upload to iLMS before **1/14 (SUN) 23:59**

3. **0 will be given to cheaters, do not copy & paste!**

4. Since we have limited resources for you guys to use, please start your work ASAP. Do not leave it until the last day!

5. Late submission penalty policy please refer to the course syllabus.

6. Asking questions through iLMS or email are welcome!

8 HINT

- Reference Java regex

- For title: `"<title>(.+?)</title>"`
- For link: `"\\[\\[(.+?)([\\|#]|\\]\\])"`

- StringBuffer for string concatenation
  - String class is immutable: slow and consumes more memory
  - StringBuffer class is mutable: fast and consumes less memory
- Counter API for recording some specific values
  - Counter interface:

    https://hadoop.apache.org/docs/stable/api/org/apache/hadoop/mapreduce/Counter.html