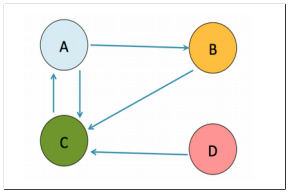
Implementing PageRank and Intro to PySpark

Due: 11:59 pm Friday, April 5

Learning Objectives:

In this assignment you will explore the PageRank algorithm via simulation and writing an mrjob program. As we've discussed in class the PageRank algorithm was Google's secret sauce in winning dominance in the competition amongst search engines. It is an excellent, non-trivial case study in big data processing in how to express a computation with the Map-Reduce paradigm. As a warmup to the next assignment, I've also included a simple PySpark exercise.

1. **Simulate PageRank**. Below is a graph representing four web pages. If a page has a link to another page, then it is represented by an edge from one node of the graph to the other. In the adjacent graph, A has links to B and C, which are known as the neighbors of A. We will represent web pages using an *incidence vector representation*, where for each node we list its neighbors (the order of the neighbors does not matter). The incidence vector representation of the sample graph, to be read from a file, is also given. As discussed in class, we can get to a page in one of two ways (a) either we directly jump to the page or (b) we navigate to the page by clicking a link. β is the probability that we will click on a link when on a page. Hence (1-β) is the probability that we will jump to a random page. *The PageRank of a web page is the probability that one will land on that page after a random "walk" across the web.* The process of the random walk is described below.




```
A B C
B C
C A
D A
```

   i. Randomly select a page, `p`, to start with and call this the current page
   ii. Generate a random number, `r`, from [0,1). (a) If `r` is less than or equal to β then simulate a click by randomly selecting a page from amongst all the neighbors of the current page and make it the current page. (b) If the random number is greater than β then simulate a jump by randomly selecting from amongst all the pages in the graph and make it the current page
   iii. Repeat step ii `walk_len` times. Whichever page you are on at the end, increment a counter for that page

To simulate PageRank itself, repeat the above 3 steps `N` times. Finally divide the counter associated with each page by `N` to determine the PageRank. Report the page rank of the various nodes in alphabetical order.

Write your code in a Jupyter notebook called `pagerank_simulation.ipynb`. Within that express your solution using the following functions:

- `read_graph(fname)` which takes the name of a file with the incidence vector representation of the graph and returns some python representation. *You are free to choose whichever representation for a graph you prefer (dictionary, list etc).*
- `random_walk(graph, walk_len=1000, beta=0.85)` which performs the random walk described above in steps i—iii. `random_walk` should return the final node it lands on after performing the random walk.
- `simulate_pagerank(fname, walk_len=1000, N=1000, beta=0.85)` is the main driver routine that calls `read_graph`, and `random_walk` and calculates the relative frequency

at which the random walk process ends on a particular node. Your execution on the sample graph above will be along the lines of and will produce the output given below:

```
simulate_pagerank("graph-1.txt", walk_len=1000, N=1000, beta=0.85)

A 0.379
B 0.206
C 0.370
D 0.045
```

If the first thing you do inside the body of `simulate_pagerank` is to seed the random number generator with `random.seed(1)` you should get the same numbers I've shown.

- Create a file called `graph-2.txt` with an incidence vector representation for the graph in the second tab of the spreadsheet exercise. Run `simulate_pagerank` on that file and check for correctness against what you got on the spread sheet.
- Create a file called `wikipedia-example.txt` which is an incidence vector representation of the graph at the beginning of the Wikipedia article on PageRank https://en.wikipedia.org/wiki/PageRank

2. Implementing PageRank using MrJob. Review slides from class and ensure you understand the logic summarized in the file QQQ . The focus of this question is to convert the logic expressed in that page to executable MrJob code. I have provided started code in `mrjob-pagerank-start.py`. Copy this into a file with the name `mrjob-pagerank-<andrewid>.py` and complete the code. Test your code along the lines of:

```
python -u mrjob-pagerank-solution.py graph-1.txt --nodes 4 --beta 0.85 -N
10 -q
```

In problem-1 you created a `wikipedia-example.txt` file. Insert appropriate print statements in the reducer_1 to produce a trace along the lines of the sample `graph-1-mrjob.trace` I have provided. Test your code with `graph-2.txt` and `wikipedia-example.txt`

3. **Spark warmup**. The file `tallest-buildings.txt` is a textual representation of Wikipedia markup used to produce this table . Write a PySpark NB, called `spark-benford.ipynb` to read the contents of the file and produce a frequency count of the number of times the first number of the height in feet occurs. Note that columns in this data set are separated by `!`. Counting from 0, the 5th column is the height of buildings in feet. As we did for the anagram exercise in class. Develop your notebook two ways (i) assign to a series of intermediate RDDs and immediately do `.take(3)` to show the intermediate values (ii) write a single parenthesized pyspark expression to compute the result. The expected final answer is:

```
[('1', 17),
 ('2', 6),
 ('3', 4),
 ('4', 7),
 ('5', 13),
 ('6', 5),
```

```
 ('7', 2),
 ('9', 1)]
```

What to submit:

Create a single zip file `a7.zip` with the following:

1. The NB `pagerank_simulation.ipynb` with the functions `read_graph(fname)`, `random_walk(graph, walk_len=1000, beta=0.85)`, `simulate_pagerank(fname, walk_len=1000, N=1000, beta=0.85)`
2. `graph-2.txt`
3. `wikipedia-example.txt`
4. `mrjob-pagerank-andrewid.py`
5. `spark-benford.ipynb`

and submit to the Canvas assignment box.