Anthony Boccadoro
Professor Fisher
446 Search Engines
18 February 2018

PageRank Report

1. I implemented the assignment in java using several helper classes. The main class is PageRank and utilizes the help of MapUtil and Tuple. MapUtil offers a generic comparator perfect for sorting number data types in a map in either ascending or descending order. The Tuple class provides four data types to be paired together. I used this to pair four Map data types, two of which mapped Strings to link counts respective to that String's page to store information about inlinks and outlinks. The other two Maps mapped Strings to ArrayList<Strings> to store information about the actual pages that are linked to the page referenced by the String key for inlink and outlinks. These, along with a ArrayList<String> called pages allowed me to implement the main algorithm. Utilizing these data structures I was able to calculate the necessary values. The fact that the file was large influenced my decision making early on in the design process. I realized that the best way to deal with a file of this magnitude was to open a FileReader and wrap it in a BufferedReader to extract data line by line and not consume too much of the computer's resources. One of the key issues that I have been dealing with is the PageRank values themselves, as a few immediately become infinite after one ranking cycle, and others not too far behind. This seems inaccurate to me but appears to make some sense.
2. I made use of Java's I/O and Util libraries for general file handling and overall calculation. Other classes include the Tuple and MapUtil class which offer a rather nice layout to the main algorithm. As mentioned above, the MapUtil class is very convenient for sorting values associated with keys in a Map. I was able to order the top 50 results by using this feature and was able to keep the associated String page key to print to the file as well. The Tuple class allowed me to manipulate a handful of data structures within the same iteration in the populate() method. This offered a more efficient runtime as fewer looping protocols needed to be executed.
3. For the most part both lists correlate well; those with high inlink count have high PageRanks. Some notable aspects include pages like Biography, where it has 7,234 inlinks yet is second for PageRank. A lot of links could be repeated links and would therefore be discarded when handling PageRank. For example, the index page has 118,979 inlinks, far more than any other page, and yet it is fifth for PageRank. For the most part, however, these lists are quite similar.
4. If the PageRanks were ill proportioned with respect to the number of pages, The nature of the algorithm influences the equations to behave according to the sum of those randomized PageRanks divided by their respective outlink counts, which holds an essentially random value. It would more than likely converge given valid enough PageRanks but could take a lot longer. If the PageRanks were set to 0, the value of the right have of the equation would be 0 on all input and every PageRank would remain constant with proportion to λ and the number of pages. If the random-surfer component is removed from the algorithm, the randomness with which pages are selected is negated, and the idea of PageRank is skewed. It would no longer represent a ranking system with a degree of accuracy. The values would become meaningless as there would be no convergence.