Search Engines 446 - PageRank Project Report - Spencer Moynihan

Project Flow:

The main method acts to direct the program flow, creating/defining the necessary variables and calling the methods to read in data (populate), calculate PageRank (page_rank), and then create the two output files (print100).

Description:

For the PageRank project, because we're processing many lines in a file, I decided to read in each line of a compressed version of the file (using GZIPInputStream) and deal with each line at once, then get rid of it. By adding data from the file one line at a time, the program does not require an excessive amount of memory. When adding the data I used a HashMap<String, Set<String>> for the links, which allowed every source to be associated with some number of links it points to. By using a set over a list, each link was guaranteed to be unique - which I thought would be ideal for this application. Although I had thought of a different way to represent the data, such that the PageRank algorithm executed faster, it ended up taking too much time to implement it and I scrapped it (although it certainly is do-able).

During the PageRank algorithm, I was having trouble getting it to run properly, so I switched from a while loop to a do-while loop and used the L1 norm rather than the L2 norm. The loop switch allowed me to execute the PageRank algorithm at least once before checking for convergence, and the L1 norm produced a larger result than the L2, allowing it to run more than just once and get better results. Furthermore, it was taking a long time to run so I switched to using an accumulator for when a page didn't point to any links. By updating the accumulator and then at the end adding the accumulated value to every page I reduced the algorithm from $O(n^2)$ to $O(n)$.

Finally, because my attempt to print the top results from a HashMap for project 1 was inefficient, I adapted the provided solution code (credit: supreethabs) for the print100 method.

Software Libraries:

I did not use any external libraries.

As for internal libraries, I used various io libraries to read from and write to files as well as various util libraries for representing various data structures including HashMaps and Sets.

Inlink count V. PageRank:

Index and United States are top 1 and 2 in both inlink count and PageRank.

The years 2006, 2007, 2008, etc. all rank lower in PageRank than inlink count.

Biography is 7th in inlink but 4th in PageRank, and All_Music_Guide is 100th in inlinks but 80 in PageRank.

Using https://molbiotools.com/listcompare.php I found that the two lists share 73 items. Out of the items only in inlinks January_31 (47), City (59), and UTC-5 (95) stuck out to me. Meanwhile, the top 100 PageRank included Personal_name (20), Romania (44), Film (68), and Rock Music (92) of note, which were not in the top 100 inlink counts.

From this, the algorithm seems to choose more specific pages to give a higher PageRank, even if it may not have the highest number of inlinks. This can be attributed to a few factors. First, is the random surfer component which is (as the name suggests) a random component that can help certain pages more than others. The second is that the source pages that point to the more specific pages likely have less links than those that point to the more generic pages. Finally, the generic pages are also more likely to point to a specific page (and pass on their PageRank) than that specific page is to point to directly point back to the generic page, rather than just referencing it (such as 1990 pointing to all the events that occurred, but the events only mentioning the year and not pointing back to the page 1990).

Experimentation: (baseline for convergence is 3 iterations)

Initializing PageRank scores: (random, all to 0)

Baseline: 3, 3, 3, 3

Rand score = 1: 4, 5, 5, 4

Rand score = 2: 78, 78, 78

Zeros: 19, 19, 19

Remove the random surfer component:

Set lambda = 0: 4, 4, 4, 4

Based on my experimenting, by setting the score randomly such that they represent a probability (rs = 1) it takes an extra iteration or two to converge. This is likely because the way the random numbers are assigned give a much higher initial score to the first element it sees (possibly all of the probability) which is statistically not a good choice (it is unlikely that this page indeed deserves such a high page rank).

By setting that score randomly to a number between 0 and 1 (rs = 2), it took 78 iterations to converge each time. The likely reason it took so long to converge is because the total initial score would be much higher than 1 (possibly higher than 1.4 million) and thus would have a larger norm for every iteration - taking longer to meet the 0.005 definition of convergence. One reason why it always took 78 runs is that the probability of selecting a random number between 0 and 1 for every entry in the dataset resulted in the same total 'probability' (~700,000). Just like how the baseline and rs = 1 took about the same number of iterations each time with a total probability of 1.

Finally, by setting lambda to 0, this dataset took an extra iteration to converge. This effectively removes the random surfer component such that the new PageRank is calculated only by the probability of clicking the link from its source. In general; however, doing this results in some pages not being accessible and thus it's possible the algorithm wouldn't converge on other data sets without the random surfer component.