[Project3 report] **Jeeho Shin**

In this project, I had to implement a site spidering functionality, calculating pagerank values for web pages. Also, I had to include the pagerank value of pages in calculating relevance with the query. Before starting, I looked on 'Graph' and 'Node' data structure to build a graph.

In the Node data structure, I added **'pagerank'** variable to save the current pagerank value of the Node. Also, I added **'pr_out'** variable to save the pagerank value that is conveyed to the nodes in edgesOut list. I added some methods to handle these variables in the Node class. In the project, I used **'edgesIn'** list to save the nodes that have an outgoing edge to the current page. Also, I used **'edgesOut'** list to save the nodes that the current page has an outgoing edge to. The 'pagerank' value of the current node is determined by adding the 'pr_out' value of nodes in its 'edgesIn' list. Then, the 'pr_out' value of the current node is calculated by dividing the 'pagerank' value by the size of 'edgesOut' list. Later, this 'pr_out' value is used for calculating the 'pagerank' value of nodes which include the current page in their 'edgesIn' list. With this mechanism in mind, I started making codes for this project.

PageRankSpider class extends Spider class. In PageRankSpider, there are two variables, which are **'graph'** and **'pageMap'**. The 'graph' is a Graph instance which is used for saving the link of the page and its corresponding node as an entry. The 'pageMap' is a HashMap instance used for saving the link of the page and the name of the file that the page information is saved. The page crawling process in the 'PageRankSpider' is very similar to that of 'Spider'. The difference is that some conditions are checked before making the edge between the current page and the links that it contains. I added the edge from current page to its links if the link is an html type, not a self-loop to the current page, and not previously included in the current page's edgesOut list. Also, after processing all links in the linksToVisit list, I iterated every node in the graph and deleted nodes in its edgesOut list, if the node doesn't have an outgoing edge. After building the graph, I printed it out and calculated the pagerank values of each page, using the 'pagerank' function.

I implemented the pagerank algorithm in **'pagerank'** function. I initialized the pagerank values of every nodes into 1 / (# of nodes). Also, I defined a 'random_jump' variable and set it to 0.15 (alpha) / (# of nodes). I ran the pagerank algorithm for 50 iterations. I used graph's **resetIterator method** and **nextNode method** to iterate nodes in the graph. After calculating pagerank values, I saved the filename and its pagerank value to the **'page_ranks.txt'** file in **'indexed'** directory.

The **PageRankInvertedIndex** class extends InvertedIndex class. It uses the 'page_ranks.txt' file to check the pagerank value of each document. It uses **'loadPageRank'** function to save filename – pagerank entry to the **pageRankMap**. In the **'getRetrieval'** function, the pagerank value of document is scaled by the weight variable (0.0, 1.0, 5.0) specified in the command line and is added

to the relevance score. Also, considering the relationship between 'Spider' class and 'SiteSpider' class, I overrode the **getNewLinks** method in the **PageRankSiteSpider** class to restrict the spidering accordingly.

To run the code, you first need to run PageRankSiteSpider and create indexed directory which saves crawled webpages and page_ranks.txt. Then, you can run PageRankInvertedIndex to try out some queries and see the relevance between documents and queries.

Q1. Does PageRank seem to have an effect on the quality of your results, as compared to the original retrieval code? Why or why not?

When I used 0.0 for weight value, all documents retrieved for the 'computer' query were webpages for students, which I think are not relevant to the query directly. However, when I used 1.0 for the weight value, two documents, P147 and P146, were retrieved compared to 0.0 weight value. These documents are webpages for 'CS439: Principles of Computer Systems' and 'Professor Alison N. Norman', respectively. They seem to be more relevant to 'computer'. I noticed that the relevance score did not increase much for other eight documents when I used 1.0 weight. However, considering that P147 and P146 have a relatively big pagerank values, 0.1182 and 0.066, respectively, I found out that incorporating pagerank to the retrieval increased two documents' scores remarkably.

Also, when I used 5.0 weight value for 'programming' query, I noticed that P149 is retrieved for the first time. I looked into its content and it was really related to 'programming'. I thought about the reason why this document is not retrieved for cases with 0.0 and 1.0. I think it has to with the document length. When the document is long, the relevance score is reduced. P149 is a long document, but its pagerank value was quite large (0.0429), I think 5.0 weight value was large enough to emphasize its pagerank and increase its relevance score.

Q2. How does varying weight change your results? Why do you think it changes in this way?

For all queries that I tried, there was no big difference for retrieved document when I used 0.0 and 1.0 for weight values. However, when 5.0 is used for the weight value, there was some difference in the result. For 'computer' query, P147 and P146 showed a huge relevance score compared to other documents. Also, P053 is retrieved newly in this case and it seems to be more relevant to 'computer' than other student pages. For both 'computer' and 'systems', P147 dominated because it has a really big pagerank value. For 'programming' query, the most relevant document was P083 for all weight values. This is because its pagerank value is small (0.0011). Therefore, I concluded that the value of weight has more impact on documents with big pagerank values. This is obvious because the weight is multiplied to pagerank value and directly added to score.