June 2nd, 2018

Lab 5 Report

Devon Martin and Shiv Sulkar

Implementation Overview:

The entirety of this algorithm was implemented using Python 3.5.2 with two external modules matplotlib and numpy.

The first method we used to calculate the PR was the traditional PageRank formula, the web surfer model. It is represented by the following equation:
PR(A) = (1-d) / N + d (PR(T1)/C(T1) + ... + PR(Tn)/C(Tn)).
The variable N represented the number of pages. PR(Ti) represented the pagerank of A and C(Ti) represented the number of outgoing links on page Ti. The damping value was initialized to .85. There are two stoppage conditions: a set maximum amount of iterations, 50 in our case, and an epsilon value that determines the threshold of minimum change between two iterations.

The alternate method used the stochastic probability matrix to converge onto a reasonable result that was less than the provided epsilon value. Unlike the first method, this method did not depend on an iteration limit. We found this method to have better performance than the first as far as calculations, because it is more suited to working with vectors. However, both required a similar number of iterations to converge on each dataset.

Results - Small Datasets

1. dolphins.csv
    a. N/A
    ```
    read_dur: 0.0019
    calc_dur: 0.0308
    iter_count: 31
    rank: 0, page: Grin, value: 1.9782
    rank: 1, page: Jet, value: 1.9556
    rank: 2, page: Trigger, value: 1.9283
    rank: 3, page: Web, value: 1.8543
    rank: 4, page: SN4, value: 1.8388
    ```
    b. Based off the results Grin is the most relevant dolphin out of all the dolphins that were studied.
2. karate.csv
    a. N/A
    ```
    read_dur: 0.0034
    calc_dur: 0.0334
    ```

```
iter_count: 29
rank: 0, page: 34, value: 3.399
rank: 1, page: 1, value: 3.2675
rank: 2, page: 33, value: 2.4148
rank: 3, page: 3, value: 1.9217
rank: 4, page: 2, value: 1.7807
```

   b. The member with ID 34 has had the most interactions with other members in the karate group.

3. lemis.csv
   a. N/A

```
read_dur: 0.0054
calc_dur: 0.0405
iter_count: 32
rank: 0, page: Valjean, value: 5.7782
rank: 1, page: Myriel, value: 3.2853
rank: 2, page: Gavroche, value: 2.7361
rank: 3, page: Marius, value: 2.3633
rank: 4, page: Javert, value: 2.3193
```

   b. The pagerank algorithm seems to have worked correctly for this dataset. It makes sense that the name of the protagonist would have the highest rank since he is likely mentioned in every chapter.

4. NCAA_football.csv
   a. N/A

```
read_dur: 0.0362
calc_dur: 0.5436
iter_count: 38
rank: 0, page: Montana, value: 2.6395
rank: 1, page: "Campbell, value: 2.3926
rank: 2, page: "Idaho State, value: 2.2067
rank: 3, page: Richmond, value: 2.1669
rank: 4, page: "St. Francis (PA), value: 2.1313
```

   b. Based off these results Montana competed and won against the most teams.

5. stateborders.csv
   a. N/A

```
read_dur: 0.0021
calc_dur: 0.0257
iter_count: 30
rank: 0, page: MO, value: 1.6004
rank: 1, page: KY, value: 1.5777
rank: 2, page: TN, value: 1.5766
rank: 3, page: MA, value: 1.4568
rank: 4, page: PA, value: 1.3642
```

   b. Montana has the most incoming edges so it is surrounded by the most states.

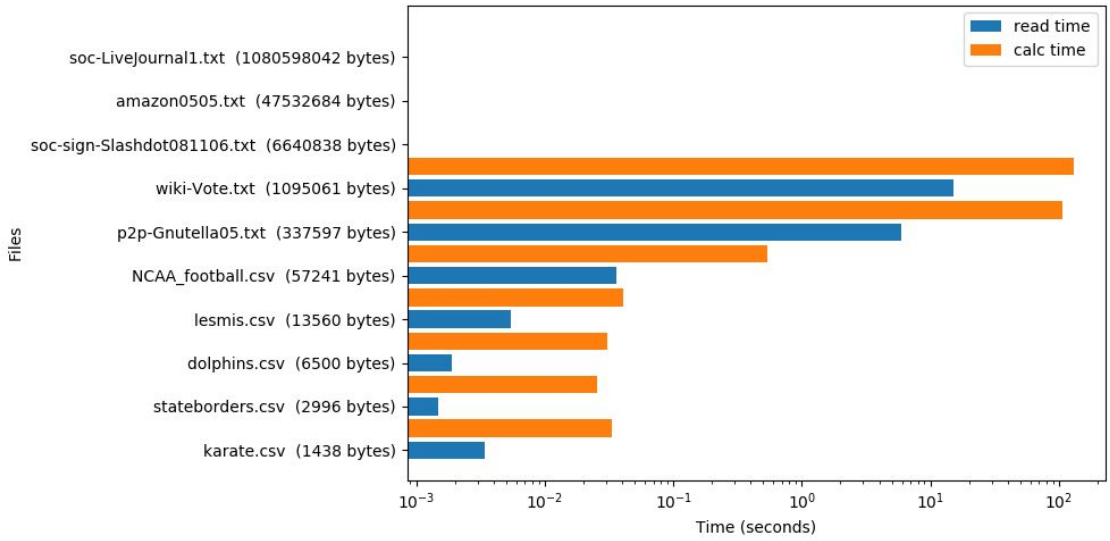
Overall Summary

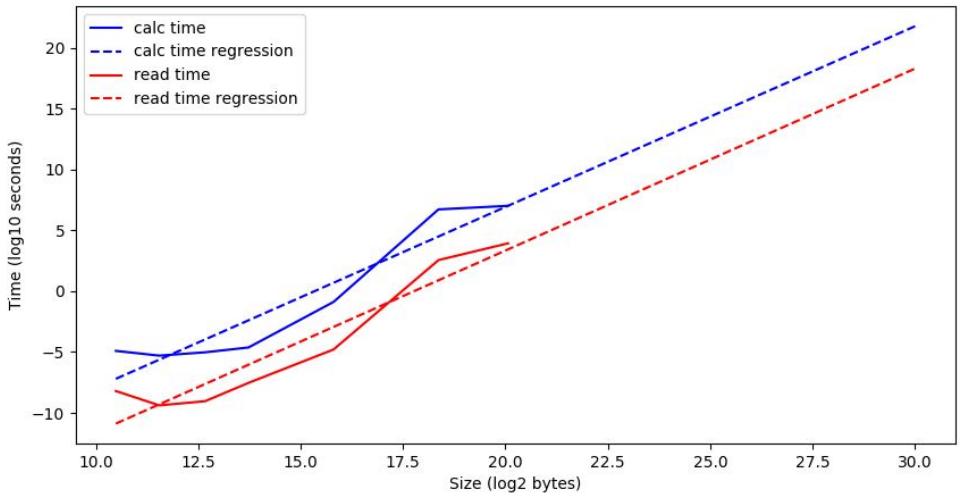
The pagerank algorithm accurately ranked items in the dataset. The pagerank algorithm worked especially well for the lesmis.csv and NCAA_football.csv since the results actually revealed valuable information about the data. The results for the other datasets just showed the interactions between the items in the dataset. Also the implementation that used an epsilon value usually had to go through more iterations to come to the correct value.

Our implementation was not as efficient as we would've liked it to be, so we were unable to run our program on the large datasets with the program running out of memory. Therefore, we opted to show a regression line indicating what that information might have looked like.

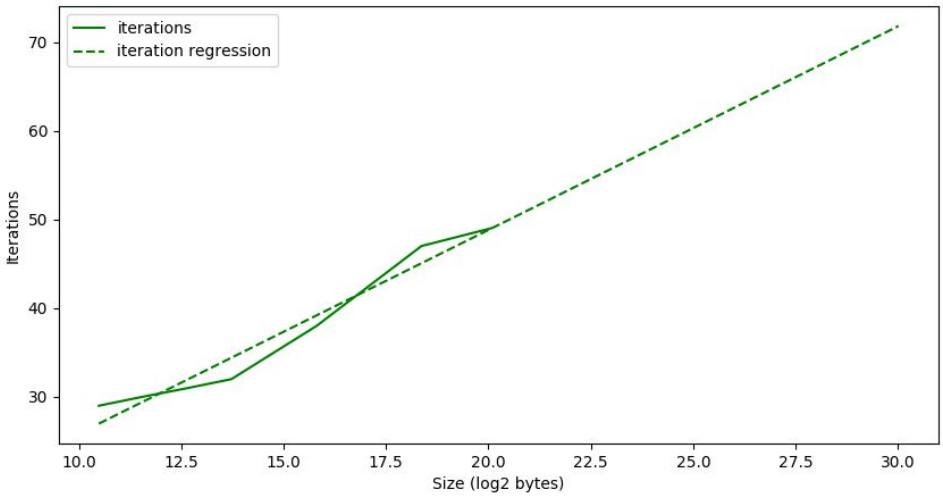
Performance - SNAP

To measure the performance of our program, we tracked how long the algorithm took to run on each dataset and graphed that against the size of the input file. We found that the time for the algorithm to run was linearly dependent of the input file size. This could mean that our complexity is on the order of O(n*n) where n is the number of nodes in the input graph. Unsurprisingly, because of this poor complexity, our algorithm failed to finish on some of the larger datasets, specifically past 10k nodes.





To add to the complexity problem, the number of iterations is also rising rapidly. The larger the dataset, the more iterations are needed to converge on a ranking, which creates a O(i*n*n) where i is the number of iterations necessary for convergence.



Extra Credit

Two different methods were used to calculate the pagerank which all resulted in varying accuracy and speed between the datasets. The stochastic probability matrix method was much faster at the calculations, but took about the same number of iterations to complete. The numbers shown in the graph of file size vs read time use the traditional method for computation time.