Web Retrieval and Mining

Report for Hw2

資工二 B00902064 宋昊恩

1. Implementation (`PageRank`):

I referred several articles about sparse-matrix calculation on Google, and found out that I could understand little about them. After thinking for a while, I got the point that the time-complexity of each-time calculation should not be `O(N^2)`, but should be `O(E)`, if I traversed only the out-links of every node.

My evaluation algorithm is as following:

a. Create two arrays, one is for old data, and the other one is for new data.
b. Do calculate and then store the value in array for new data.
c. Calculate the difference between old and new data, if bigger than ϵ, go to b.

Frankly speaking, my first-version-program's performed very slowly. After using one of my classmates' tips, my program sped up about three hundred times. The key reason was, for every zero-out-link nodes, I used `O(N)` time to deliver the value to all the other nodes. However, it was really stupid to do so separately. The solution was to pre-calculate the sum of values of these no-out-link nodes and then deliver the value at once. This manipulation can actually reduce the time-complexity from `O(N^2)` to `O(E)`.

2. Implementation (`LexRank`):

I first used `std::map` in `c++` to store all the value from file `idf`. Then, I read in the input data and began to parse each single line. I used `std::map` again to store information for each line as individual vector space. After that, I calculated the euclidean distance between two different lines. If the distance between them exceeded threshold (initial as `0.1`), I then created double direction edge across them.

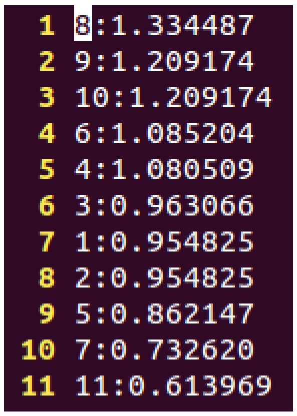
After analyzing the similarity between lines, I can get a whole new graph. Then, I performs `pageRank` algorithm on the graph just as above-said.

Though the difference between each sentence's `pageRank` seemed small, `L1` norm still showed a quite big number between `standford-08-03.pagerank` and my answer. This really interested me, and I desired to know the reason why.

I was very surprised to find out that once I change the threshold from `0.1` to `0.2`, not only the `lexRank`'s value changed, but also the relatively order. I believed that it is caused by edge-links changing in the graph.

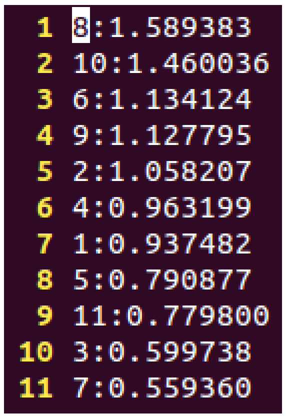
I will added some testing results for `sample.sen` with different threshold value.

3. Some other resulting graphs that I tried with thresholds:



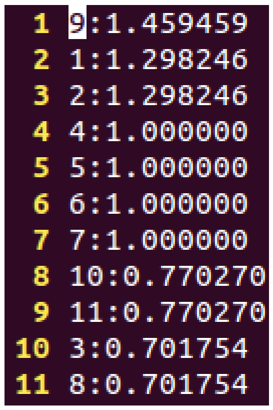
```
8:1.334487
9:1.209174
10:1.209174
6:1.085204
4:1.080509
3:0.963066
1:0.954825
2:0.954825
5:0.862147
7:0.732620
11:0.613969
```

(Threshold with `0.1`)



```
8:1.589383
10:1.460036
6:1.134124
9:1.127795
2:1.058207
4:0.963199
1:0.937482
5:0.790877
11:0.779800
3:0.599738
7:0.559360
```

(Threshold with `0.2`)



```
9:1.459459
1:1.298246
2:1.298246
4:1.000000
5:1.000000
6:1.000000
7:1.000000
10:0.770270
11:0.770270
3:0.701754
8:0.701754
```

(Threshold with `0.3`)

When the limit threshold was enhanced, there were more legal edges existing in the resulting graph. Though the differences between the thresholds were not quite big, the order of the rank still changed.

4. Reference:

The whole program was implemented by my own self. I got some key tips from my classmate `B00902053`.