TI2736-C Datamining
Assignment 3: Link Analysis

Delft University of Technology, February–April 2017

Thomas Abeel, Marcel Reinders
Zekeriya Erkin
Wouter Kouw, Tamim Abdelaal
Lucas van Dijk, Samuel Sital, Ravi Autar

Pattern Recognition and Bioinformatics Group

TU Delft

3 Link Analysis

In this exercise you will create the PageRank algorithm, named after Larry Page, co-founder of Google. PageRank was designed to combat the growing number of *term spammers*. For this exercise we will look at PageRank and some of its adaptations. Finally we will use PageRank to compute which airports in the US are most important.

Exercise 3.1. PageRank

We will start this exercise with a small network, simulating the entire internet with a few sites. Then we will simulate what a random surfer would do on this network and where the random surfer is most likely to end up.

Step 1. Investigate the data of transitions from one vertex to the other in the `data/example.txt` file. The data is of the form:

```
source|destination|weight
```

In this case, all weights are set to 1, meaning all transitions are equally likely to happen.

Question 1.1 Draw the directed graph based on this data.

Question 1.2 Write out the transition matrix for this network. Verify that all columns sum up to 1.

Question 1.3 If we would initialize a random surfer at a random location (ie. the chance to be spawned at a certain vertex is equal for all vertices), what are the chances for this random surfer to be at a certain location after one iteration? Manually calculate these probabilities.

Step 2. `PageRank.java` contains a class that can be used for calculating the PageRank values of a dataset with the form as described above. In `main.java`, create a `PageRank` object and import the data in `data/example.txt`. Print the `data` object and check to see how the data is stored.

Step 3. Next, a transition matrix has to be constructed, using the method in `PageRank.java`, `constructTransitionMatrix`. It is up to you to finish this method. For this you may use the `Matrix` class, which is basically a wrapper for an `ArrayList` with `Double` values. Note that in contrary to the lecture and the book, the edges have weights here. This means that if you want to calculate the probability of ending up in B when you are in A, you have to divide the weight of the edge from A to B with the sum of all edges *going out* of A.

Step 4. Run the method and verify that the transition matrix is equal to the matrix that you calculated in question 1.2.

Step 5. Finish the implementation of the `PageRank.getRandomSurfer` method. This method should create a row vector of length equal to the number of vertices in the data. Each element should have equal probability and the probabilities should sum up to one. In other words, it should construct the following vector:

$$\mathbf{v} = \begin{bmatrix} \frac{1}{n} \\ \frac{1}{n} \\ \vdots \\ \frac{1}{n} \end{bmatrix} \tag{1}$$

Where n is the number of vertices in the data.

Step 6. Next, complete the `PageRank.calculatePageRank` method. This method should calculate a transition matrix, get a random surfer vector and multiply these for a number of iterations. For the multiplication, you can make use of the `Matrix.dot` method, which calculates the matrix multiplication between two matrices. The iterative step is:

$$\mathbf{v}' = M\mathbf{v} \tag{2}$$

Where M is the transition matrix.

Step 7. Run the `PageRank.calculatePageRank` method on the `data/example.txt` dataset with 10 iterations. Verify that the result is approximately as follows:

$$\mathbf{v}_{10} = \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} 0.354 \\ 0.119 \\ 0.294 \\ 0.233 \end{bmatrix} \tag{3}$$

Step 8. Run the `PageRank.calculatePageRank` method on the `data/example2.txt` dataset with at least 10 iterations. This dataset is slightly different. The edge from C to A is replaced by an edge from C to C itself.

Question 8.1. Explain the results you now get from the PageRank algorithm.

Step 9. In order to make sure nodes like these do not corrupt our results, we can use taxation to allow the random surfer to randomly jump from one page to another. This comes down to changing our iterative step to:

$$\mathbf{v}' = \beta M\mathbf{v} + (1-\beta)\mathbf{e}/n \tag{4}$$

Where $\mathbf{e}$ is a vector of all ones, n is the number of vertices in the data and β is a constant.

The `TaxationPageRank.calculatePageRank` method should calculate this modified PageRank value using this iterative step. Implement this method setting β to 0,8.

Question 9.1. Check the results using this adapted version of PageRank, are the results better?

Question 9.2. What happens if we set beta to 0? What happens if we set beta to 1?

Step 10. Check out the `data/flight_data.txt` file. This file contains information regarding airports in the US and flights between them. Each line represents a connection from one airport to another with the weight equal to the number of flights in January 2013. Run the algorithm on this dataset.

Question 10.1. What is the most important airport according to the results? (You can use the `sortByValues` in `main.java` to sort by PageRank value)