

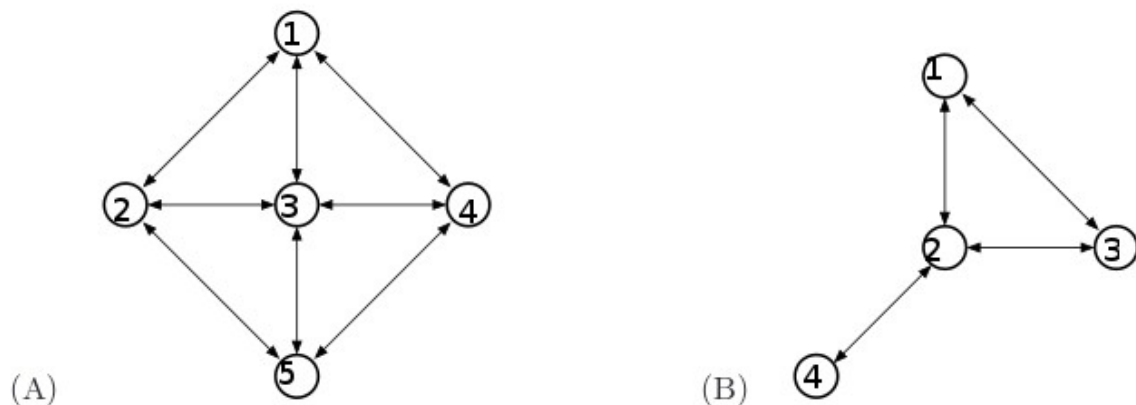
Homework 4

Problem 1

The code used to compute the PageRank results can be found in:

pagerank.py

First of all, let us give some etiquettes to the nodes in the picture.



Let's examine graph A:

The transition matrix (A) will be:

	Node 1	Node 2	Node 3	Node 4	Node 5
Node 1	0	1/3	1/4	1/3	0
Node 2	1/3	0	1/4	0	1/3
Node 3	1/3	1/3	0	1/3	1/3
Node 4	1/3	0	1/4	0	1/3
Node 5	0	1/3	1/4	1/3	0

There is a clearly noticeable simmetry between the nodes that have the same number of edges connected, and the one that really is different from the others is the node number 3, since it has four edges and not just three like the others.

So, let's start the algorithm for computing the PageRank assigning to each node the same exact weight:

$$v=[1/5,1/5,1/5,1/5,1/5]$$

The first iteration will compute $A \cdot v$, with A as the transition matrix. The result of this is:

$$A \times v=[0.18888889,0.18888889,0.24444444,0.18888889,0.18888889]$$

Needless to say, we obtained the same exact result for all the numbers with the same number of edges.
Next iteration: $A^2 \times v$

$$A^2 \times v = [0.18765432, 0.18765432, 0.24938272, 0.18765432, 0.18765432]$$

For a more refined result, we can iterate $A^n \times v$ until the result stabilizes to a certain vector, that will be the vector of the PageRank values of all our nodes. In this case, this will be:

$$[0.1875, 0.1875, 0.25, 0.1875, 0.1875]$$

graph B:

Transition matrix:

	Node 1	Node 2	Node 3	Node 4
Node 1	0	1/3	1/2	0
Node 2	1/2	0	1/2	1
Node 3	1/2	1/3	0	0
Node 4	0	1/3	0	0

Here, the only nodes that have the same number of edges are node 1 and 3.

$$v = [\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]$$

$$A \times v = [0.27083333, 0.29166667, 0.27083333, 0.16666667]$$

$$A^2 \times v = [0.26215278, 0.32986111, 0.26215278, 0.14583333]$$

...

And the resulting pagerank values will be:

$$[0.25, 0.375, 0.25, 0.125]$$

My conjecture are that, if $\beta = 1$, then the PageRank values can be calculated by computing:

$$PageRank = \frac{\text{number of edges of a node}}{2 \times \text{total number of edges}}$$

Let's try this on the nodes of graph A:

The total number of edges in the graph is 8.

Node 1 has 3 edges, so its PageRank value will be $3/16 = 0.1875$.

Same thing for all the other nodes, except for node 3, that will have a PageRank equal to $4/16 = 0.25$

$$v = [0.1875, 0.1875, 0.25, 0.1875, 0.1875]$$

And this is the same result we got before.

Let's apply the same procedure for graph B:

Total number of edges is 4.

Node 1 has 2 edges, same as node 3, so they have PageRank = $2/8 = 0.25$

Node 2 has 3 edges: $3/8 = 0.375$

Node 4 has 1 edge: $1/8 = 0.125$

$$v = [0.25, 0.375, 0.25, 0.125]$$

Problem 2

Assuming that X is:

$$X = \sum_{i=1}^l Y_i m(e_i)$$

The expected value of X can be calculated as following:

$$\begin{aligned} E[X] &= E\left[\sum_{i=1}^l Y_i m(e_i)\right] \\ &= \sum_{i=1}^l E[Y_i m(e_i)] \\ &= \sum_{i=1}^l E[Y_i] E[m(e_i)] \end{aligned}$$

At this point, let's concentrate only on $E[Y_i]$: Y is a vector composed by random values, 1 or -1, each one with probability $\frac{1}{2}$. Therefore, the expected value of Y_i will be:

$$E[Y_i] = (1) \times \frac{1}{2} + (-1) \times \frac{1}{2} = 0$$

This 0 is multiplied by the rest of the formula, therefore:

$$E[X] = 0$$

Let's now calculate $E[X^2]$:

$$\begin{aligned} E[X^2] &= E\left[\left(\sum_{i=1}^l Y_i m(e_i)\right)^2\right] \\ &= E\left[\sum_{i=1}^l Y_i^2 m(e_i)^2 + \sum_{i < j} Y_i Y_j m(e_i) m(e_j)\right] \\ &= \sum_{i=1}^l E[Y_i^2] E[m(e_i)^2] + \sum_{i < j} E[Y_i] E[Y_j] E[m(e_i)] E[m(e_j)] \end{aligned}$$

Now, as we have seen before:

$$E[Y_i], E[Y_j] = 0$$

So the second part of the sum is equal to zero. Instead,

$$E[Y_i^2] = (1)^2 \times \frac{1}{2} + (-1)^2 \times \frac{1}{2} = 1$$

Therefore:

$$E[X^2] = \sum_{i=1}^l m(e_i)^2$$

And this is the exact same formula used to calculate the second moment.

Algorithm used to estimate the second moment in part 2:

- I collect the streaming tweets
- For every tweet that arrives, I hash the name of the user with 1000 different hash functions.
- I keep a buffer of 1000 temporary values, corresponding to the 1000 different hash functions. For every hash function, if hash(username) is an even number, I increment the corresponding value, otherwise I subtract one.
- My resulting second moment is the sum of all the squared values in the temporary array, divided by how many hash functions I used.

To calculate X , I need to have a vector of random values (1 or -1) associated with every user. To do this without having to store all the user's names and values, I just hash the name of the user and see if the result is even or odd. This way, the same user will have always the same value, without any information to store.

The 1000 different hash functions are needed to simulate a random function, and only the average of their result is considered.

By running this algorithm while downloading the tweets in streaming I get these results:

16.00 – 23.00


Streaming:

```
----  
Number of tweets: 17430  
Second moment of this round: 1633284  
Average Second Moment: 811769  
Control value: 1  
----
```

From file:

```
total tweets: 17455  
number of users: 7495
```

second moment: 760131

user who tweeted the most:  with 605 tweets

5625 users tweeted only once

858 users tweeted twice

307 users tweeted thrice

To execute the program, run

twitterdatamining2.py

The program will output the calculated second moment every 10 tweets received. The relevant value to read is "Average Second Moment". The tweets will be stored in "tweets.txt". If you wish to check the second moment calculated with non streaming techniques, run

calculatesecondmomentfromfile.py