CMDA 3634 Fall 2017 Homework 02

Domenic Riccardi

September 27, 2017

You must complete the following task by 5pm on Monday 09/25/17.

Your write up for this homework should be presented in a LaTeX formatted PDF document. You may copy the LaTeX used to prepare this report as follows

1. Click on this link
2. Click on Menu/Copy Project.
3. Modify the HW02.tex document to respond to the following questions.
4. Remember: click the Recompile button to rebuild the document when you have made edits.
5. Remember: Change the author

Each student must individually upload the following files to the CMDA 3634 Canvas page at https://canvas.vt.edu

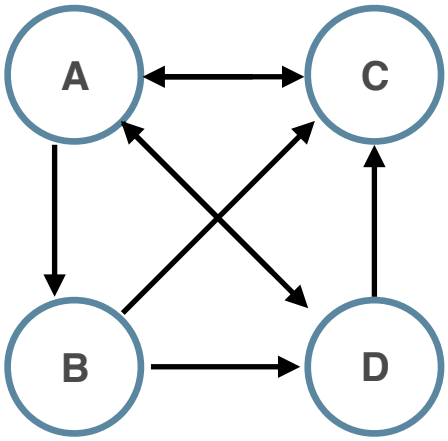
1. `firstnameLastnameHW02.tex` LaTeX file.
2. Any figure files to be included by `firstnameLastnameHW01.tex` file.
3. `firstnameLastnameHW02.pdf` PDF file.
4. `network.c` text file with student code.

You must complete this assignment on your own.

150 points will be awarded for a successful completion.
Extra credit will be awarded as appropriate.

In the last assignment, we learned how to represent a directed graph with a two dimensional array. Now we will examine how to naively compute PageRank. We will then fine tune the algorithm so we can consistently compute PageRank[1].

Q1 *(30 points) Computing the naive PageRank with example.*
Lets consider an example graph and compute the PageRank. Consider the graph:



In array representation, this network is represented with the following:

			To	
	A	B	C	D
A	0	1/3	1/3	1/3
B	0	0	1/2	1/2
C	1	0	0	0
D	1/2	0	1/2	0

Suppose we initially guess all the websites are of equal importance. We assign each of them an importance value of $1/n$ where n is the number of nodes in the network. (This way the sum of the importance of all nodes will equal one). The idea behind PageRank is to let this importance flow between websites that are linked (following directional arrows in our network diagrams) in steps until an equilibrium is reached. But how should we let the importance flow? We will first define a natural, but somewhat limited, set of rules for how importance flows.

Naive PageRank Algorithm: Envision that we are stepping forward in time. Each time we take a step forward, the importance a node has is distributed evenly among the nodes it directly relies on. For instance, in the above network: Every time we step forward, node B will distribute all of its current importance among nodes C and D evenly. On the other hand, B receives one third of the importance that A has. So the importance of B at step $k+1$ is always one third the importance of A at step k. We can derive similar formulas for the importance of A, C, and D at step $k+1$.

For the example above, we assign each node an importance of 1/4 to begin. Now lets take a few steps forward:

Node	Step 0	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6
A	0.25	0.375	0.4375	0.3542	0.3958	0.3906	0.3819
B	0.25	0.0833	0.125	0.1458	0.1181	0.1319	0.1302
C	0.25	0.3333	0.2708	0.2917	0.2951	0.2865	0.2917
D	0.25	0.2083	0.1667	0.2083	0.1910	0.1910	0.1962

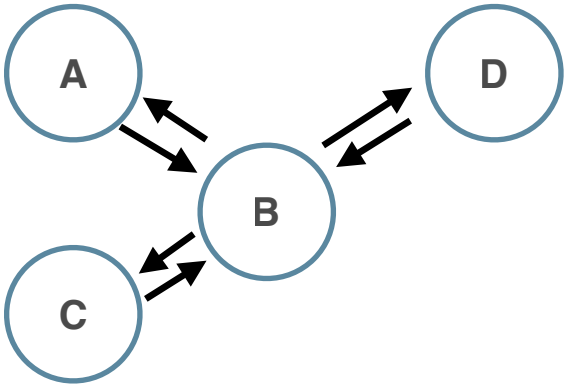
It seems when we let the importance flow in this sense it begins to settle to some constant distribution among the nodes. In fact, lets look at the extreme long term:

Node	Step 0	Step 10	Step 100	Step 1000
A	0.25	0.3876	0.3871	0.3871
B	0.25	0.1289	0.1290	0.1290
C	0.25	0.2902	0.2903	0.2903
D	0.25	0.1933	0.1935	0.1935

Table 1: PageRank importance values after 0, 10, 100, and 1000 steps.

It appears that after just 100 steps we have reached an equilibrium. The equilibrium values associated with the nodes in the network are referred to collectively as the PageRank vector. The higher the PageRank value for a node, the more important we say the node is. Interpreting the results in Table 1 we say node A is the most important, node C is second most important, node D is third most important, and finally node B is least important. Will this algorithm always work so nicely?

Exercises: Q1.1 (10 points) For the following networks, try preforming the naive PageRank Algorithm by hand (or with MATLAB). Assign each node an initial importance of 1/4. Do four steps forward. Does anything interesting happen? Can you guess what the PageRank values will be after step 100 and after step 101?



Solution: The PageRank importance values oscillate, completing an entire oscillation on every other step. From this we can determine that the PageRank importance values on step 100 will match those of step 0, while the values of step 101 will match those of step 1. This pattern can be observed in the table below.

Node	Step 0	Step 1	Step 2	Step 3	Step 4
A	0.25	0.0833	0.25	0.0833	0.25
B	0.25	0.75	0.25	0.75	0.25
C	0.25	0.0833	0.25	0.0833	0.25
D	0.25	0.0833	0.25	0.0833	0.25

Table 2: PageRank importance values computed by hand for **Q1.1**

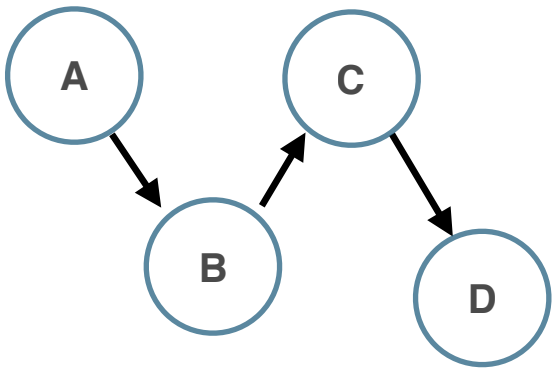
Q1.2 (10 points) For the following array representation of a network, try preforming the naive PageRank Algorithm by hand (or with MATLAB). Assign each node an initial importance of 1/4. Do three steps forward.

			To	
	A	B	C	D
A	0	1	0	0
B	1/2	0	1/2	0
C	0	1	0	0
D	0	0	1	0

Solution: The PageRank importance values for the Q1.2 network after 3 time steps are as follows...

Node	Step 0	Step 1	Step 2	Step 3
A	0.25	0.125	0.25	0.25
B	0.25	0.5	0.5	0.5
C	0.25	0.375	0.25	0.25
D	0.25	0	0	0

Q1.3 (10 points) For the following networks, try preforming the naive PageRank Algorithm by hand (or with MATLAB). Assign each node an initial importance of 1/4. Do an appropriate number of steps. What problem arises?



Solution: The PageRank importance values for the Q1.3 network quickly shift all importance to node D in exactly 3 time steps, after which the importance values do not change. This can be observed below...

Node	Step 0	Step 1	Step 2	Step 3
A	0.25	0	0	0
B	0.25	0.25	0	0
C	0.25	0.25	0.25	0
D	0.25	0.5	0.75	1

Q2 *(30 points) Refining the PageRank Algorithm.*
We have already seen what can go wrong when attempting to compute the PageRank. Now we examine an alternate PageRank update strategy posed by Page and Brin[2] (the founders of Google). But first we must set up some notation and machinery. Notation:

1. Let N be the number of nodes in our network.
2. Let i denote the $i-$th node in our network.
3. Let $PR(i)$ be the PageRank of node i-th node.
4. Let $M(i)$ be the set of all nodes that are connected to i.
5. Let $L(i)$ be the number of outbound connections i has.

We will also need to develop sum notation. Many will be comfortable with the following:

$$\sum_{j=1}^{9} j = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 45.$$

This reads "compute the sum of all values j where $j = 1, 2, 3, ..., 9$." So we sum the first 9 natural numbers. But we can sum over any collection of things. For instance, we could do something like the following:

$$\sum_{j \in M(i)} 1$$

This means "add one to the sum for every node that is connected to node i". (For those that have never seen $\in$ before, it just means "in the set". So $j \in M(i)$ is fancy notation for node j is in the set of nodes connected to i.)

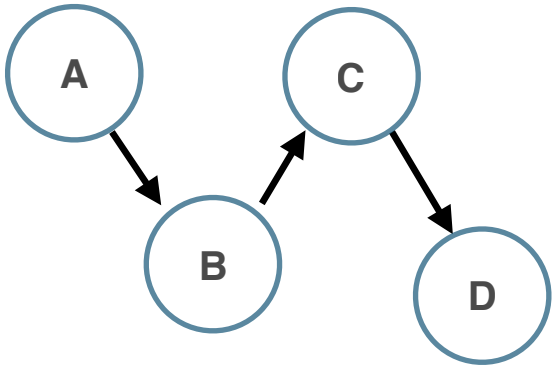
We introduce a constant d which can be any value between 0 and 1. (A common choice is to let $d = 0.85$.) Page and Brin proposed updating the PageRank of each node with the following formula:

$$PR(i)_{\text{new}} = \frac{1-d}{N} + d\left(\sum_{j \in M(i)} \frac{PR(j)_{\text{old}}}{L(j)}\right)$$

The formula finds the updated PageRank value for a single node i, so we would use this formula once for each of the N nodes to compute a new array of PageRank values. The formula uses only the information in step k to compute the PageRank at step $k + 1$. Imagine we doing the update for each node at the same time. This means if we update the PageRank for node j and then go to compute the next PageRank of node i, we should *not* use the updated value of $PR(j)$.

It turns out this method for updating will guarantee the distribution of importance will settle to an equilibrium. Furthermore, we will settle to this equilibrium regardless of what initial distribution of importance we guess.

Q2 : Exercise (30 points) For the network the network below, use the PageRank Algorithm given above to step forward in time. Preform two iterations. (That is, update each nodes PageRank twice.) Suppose each node begins with an importance of 1/4.



Solution: The PageRank importance values of the network, computed using the Page and Brin algorithm where $d = 0.85$, are shown below...

Node	Step 0	Step 1	Step 2
A	0.25	0.0375	0.0375
B	0.25	0.25	0.0692
C	0.25	0.25	0.25
D	0.25	0.25	0.25

Q3 *(90 points) Implementing the PageRank Algorithm.*
Now it is finally time to implement the PageRank Algorithm. Edit your `network.c` code from assignment 1 and add the desired functionality.

Q3.1 *(5 points) Changing the Network struct.*
To implement PageRank, we need to slightly alter our network struct from assignment 1. We will need *two* pointers to arrays of doubles. These arrays will contain PageRank values of all nodes in the network. One array should contain old (or pre-update) PageRank values and the other should contain the new (or post-update) PageRank values. We need to store both so we can compute how much these PageRank values have changed during the step (discussed in **Q3.3**). Space for these arrays should be allocated in the `networkReader` method.

Q3.2 *(35 points) Update PageRank.*
Now write a method to simulate taking one step forward in the PageRank Algorithm. Use the update formula presented in **Q2**. Set $d = 0.85$ for now. Use the following prototype for the method:

```
double updatePageRank(Network net){

  /* Q3.2 Code */

  /* End Q3.2 Code */
```

```
}
```

You will (hopefully) notice the method is supposed to return a double. That is because we will measure how much the PageRank array has changed during the step. This brings us to the next method in need of implementation, `computeDiff` (discussed in **Q3.3**). So `updatePageRank` should call `computeDiff` and return the computed difference norm.

Q3.3 *(35 points) Compute Difference Norm in PageRank.*
In mathematics, we often measure the "closeness" of two objects with a norm. To measure the distance between the old PageRank array and the new PageRank array we will use the 2-norm (Euclidean norm). Let N denote the number of nodes in the network, let newPageRank_i denote the i-th entry of the new PageRank array. Similarly, let oldPageRank_i to be the i-th entry of the old PageRank array. We use the notation $\|x\|_2$ to indicate "compute the $2-$norm of x". The formula to compute the 2-norm of the difference between the arrays is as follows:

$$\text{diff} = \|\text{newPageRank} - \text{oldPageRank}\|_2 = \sqrt{\sum_{i=0}^{N-1} \left(\text{newPageRank}_i - \text{oldPageRank}_i\right)^2}$$

That is, sum the squares of the differences in the elements between new and old array and then take the square root. Write a method to compute the $2-$norm of the difference between the old and new PageRank arrays. The prototype of the method should be:

```
double computeDiff(Network net){


  /* Q3.3 Code */

  /* End Q3.3 Code */

}
```

Q3.4 *(10 points) Compute the PageRank.*
Now create a computePage rank method with the following prototype:

```
void computePageRank(Network net, double tol){

  /* Q3.2 Code*/


  /* End Q3.2 Code */

}
```

The method should call `updatePageRank` then `updateDiff` iterativey until the returned difference norm value dips below the desired tolerance (the variable input *tol*). If you would like to monitor the convergence of the process, feel free to print the norm of the difference in PageRank arrays as well. Important: Don't forget to initialize the PageRank values as 1 / (number of nodes) for all nodes.

Q3.4 *(5 points) Updating the Main Function.*
We would now like to change what the main function does. The method should now compute the PageRank and output the PageRank of node i (where i is the input the program takes). Set a tolerance of $tol = 1e{-}6$, call the computePageRank method and print the PageRank of node i neatly. Test the program on the file `numbers.csv` and a node of your choice.

Extra Credit: *(10 points)*
Alter the program to take *tol* and the damping factor d as a command line argument.

Solution: Refer to file `network.c` for my implementation of question 3 and the extra credit.

References

[1] PageRank Algorithm - The Mathematics of Google Search `http://www.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture3/lecture3.html`

[2] The PageRank Citation Ranking: Bringing Order to the Web `http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf`