PageRank Project

Project Due: Thursday, March 30 (by 5:00 pm ET)

In this assignment you will implement the PageRank algorithm used for ranking web pages. You will be provided with a small and a large web graph for running PageRank. You will then analyze the performance and stability of the algorithm as you vary its parameters. Finally, for extra credit, you can attempt to come up with a provable strategy that can increase the PageRank for a new website.

Instructions

- You must complete this assignment on your own, not in a group.
- You can use any of the following programming languages: C, C++, Java or Python 2.7.*.
- Your code must obey the specified input-output format, accepting input from a text file and writing output to stdout (standard/console output).
- If you write C or C++ code, you must supply a Makefile with 2 targets (`compile` and `run`) such that your program can be compiled (if necessary) and executed as follows:

```
make compile
make run input.txt alpha > output.txt
```

- Other than built-in functions such as a random number generator, and numerical computing libraries like NumPy or SciPy, you cannot use any libraries, or code from any other source, esp. related to hashing or Bloom filters.
- You need to turn in a report (PDF, 1 page, single-sided), a text file called “output.txt” containing a PageRank vector, source code, and the Makefile (for C/C++) via T-Square. The contents of the report are described in part 3 below.

Tasks

In this project you will need to write an efficient implementation of the PageRank algorithm in order to rank a very large web dataset in real time.

1. Implement PageRank

You will need to implement PageRank that takes input from a text file and outputs to stdout. You can use input redirection (the ">" on the command line) to output to a file. PageRank will take the parameter alpha from the command line. **Your PageRank vector should be initialized to a uniform probability vector (each entry is 1/n initially).**

Important Note 1: You need to represent the adjacency matrix in adjacency list form. Moreover, you need to implement your pageRank iteration such that it take O(m) time, not $O(n^2)$. You will lose points if you do an $O(n^2)$ implementation. For reference, my code takes under 5 minutes and about 70 iterations on the big data set.

Important Note 2: Some vertices will not have any out-edges, which can cause some problems. To avoid this, **you must add a self-loop to each vertex**. If you do not do this, you will get a different answer. You can make this modification when you read in the input.

Input:

Your PageRank program will receive a text file as input and a number alpha. The alpha will correspond to the alpha used for PageRank. The text file will encode a graph. In the text file, the k^{th} line will look like this:

```
k:a1,a2,a3,...,an
```

That is, it will be the number k, followed by a colon, followed by a sorted comma-separated list of numbers, where each number corresponds to a vertex that vertex k has edges to. For example, the text file might look like this:

```
0:2
1:2
2:0,1,3
4:0
```

In this example, vertex 0 has an edge to vertex 2, vertex 1 has an edge to vertex 2, vertex 2 has edges to vertices 0, 1, and 3, vertex 3 has no outgoing edges, and vertex 4 has an edge to vertex 0.

Your program will need to store a representation of the graph, which should fit in your RAM. To do so, you will store the graph as a 1D array of linked lists. The i^{th} entry of the

array contains a linked list of all of the vertices that vertex i has an edge to. Do not store the graph explicitly as an *n* x *n* matrix.

Note: The input is given such that if there is an entry (u,v), this means there is an edge from u to v. In other words, this is a matrix of out-edges. For the purposes of implementing the PageRank algorithm, it may be helpful to store the matrix of in-edges. That is, for each vertex v, you will store a list of vertices that have an edge to v. Make sure you store this matrix as a 1D array of linked lists.

Note: For Java, you can store your adjacency matrix as an ArrayList of ArrayLists. For python, you can store the adjacency matrix as a list of lists. If you're using C/C++, you may need to find out *n* from the text file before you initialize your array (unless you have some implementation of dynamic arrays). You can find out *n* by simply by making one pass where you find the largest vertex in the graph, followed by a pass where you actually process the graph.

Note: The input file has fewer rows than the number of vertices. This is because some vertices have no out-edges.

Output:

Your code will compute the PageRank vector, which is a length-n vector corresponding to the stationary distribution of the random walk on the input graph. The output should be n lines, one for each vertex, where the k^{th} line outputted contains the value of the PageRank vector for the k^{th} vertex (where we index the first line outputted with 0)

For example if your PageRank vector is the length-n vector **p**, where the i^{th} entry is **p**(i), then the output is:

Output:

p(0) **p**(1) **p**(2) .. **p**(n)

Since each **p**(i) will be some decimal, we further require that each line be formatted in scientific notation to 10 decimal places.

For an example of the desired output, see the given PageRank vector for the small test case.

Running your code:

We must be able to run your code in a specific way. Below are the specifications for each language:

C/C++:
Your code should run from the makefile as follows:

```
make compile
make run input.txt alpha > output.txt
```

Java:

Your code should run from your main method in a class called PageRank.

```
java PageRank input.txt alpha > output.txt
```

Python:
Your code should run as follows:

```
python PageRank.py input.txt alpha > output.txt
```

Your python code should be written in python 2.7.*. If you are not sure what version you are running, type this into the command line:

```
python -V
```

2. Find the PageRank vector

We will give you a large web dataset, and you will run PageRank on it (with **alpha = 0.85**) until your code converges, at which point you will output your PageRank vector. The code converges when each entry differs by less than some ε. In other words, if your PageRank vector in iteration t is $\mathbf{v}^{(t)}$, then you will stop at iteration T when for each entry i, $|\mathbf{v}^{(T-1)}(i) - \mathbf{v}^{(T)}(i)| < \varepsilon$. **We will set $\varepsilon = 10^{-10}$**.

You will be provided with a smaller test case of web data with its PageRank vector, so that you can verify that your code is working.

We will also provide you with a ranking of the nodes in order of decreasing PageRank. Your job is to find the actual PageRank values.

You will be required to save your PageRank vector in a text file called "output.txt" The format for this output is specified above. The easiest way to generate this output from the command line is to use stdout redirection as specified above. A python program will be provided that outputs the largest absolute entrywise difference between two vectors, where the two vectors are saved as text files with one line per entry.

3. Report: Analyze different alpha's

Your report will be entirely on this section.

The value of alpha changes the PageRank vector. You will vary alpha between .75 and .95 to see how the PageRank vector changes. The way in which you measure or describe this change is up to you. Some examples might be:

- How different are the top sites for each alpha? How different are the PageRanks of the top sites?
- What is the largest change in any site's PageRank as alpha changes?
- How does the PageRank vector change as a whole? What ways could you measure this? You might consider looking at the L1 or L2 norm of the difference between different PageRank vectors

You don't have to do any particular one of the above. If you think of a different way to measure the change, you are entirely free to use that method. For whatever method you choose for measuring the change in the PageRank vector, explain why you chose that metric and to what extent the PageRank vector changes using that metric. Write any interesting observations you find.

This section is purposely vague because the point is to let you figure out what makes sense for your data and give some justification.

Extra Credit

If you want to be found on the internet, one of the best ways is to be high in Google's search results. Getting a high PageRank value is one way to do that. One way to improve your PageRank value is to link your site to websites with high PageRank. Another way would be to make several new dummy websites and link them all to your website and to each other. Both of these methods cost money.

Let the score of a node be the index in the sorted PageRank vector. So the vertex with the fifth largest PageRank value will have score = 5.

There will be two variants of this problem, and you should solve both of them to receive extra credit.

For the extra credit, you will be trying to get in the top 1% of PageRank values by spending the least amount of money. The costs of the two above methods are as follows:

- To add a link from a vertex v with score *i* to your website (or to any of the dummy websites you create) will cost $(876000 - i + 1)^2$ dollars per link. Note we are using the position of v, not its actual PageRank value. So if Facebook has the 3rd-highest PageRank value, it will cost $(876000 - 3 + 1)^2$ dollars for Facebook to link to you. We give you a list of the rankings of vertices by PageRank.
- Adding a new dummy website will cost 1000 dollars per website
    - Adding links from your website or from dummy websites you create is free

For the first variant of this problem, you cannot create any dummy websites. You can only create links to and from other websites (for the appropriate cost). For the second variant, you can create dummy websites in addition to the links from the first variant.

For this extra credit, you will need to explain what your solution is doing in words and also why you picked your solution.

You website will be #875713. For your submission, provide the number of new websites that you make. These will be indexed starting from 875714. Then provide a list of edges, one line per edge.

In this example, we are making two new websites and adding 3 edges:

```
2
2001 875713
5 875714
875715 875713
```

In your extra credit report, you should mention how much your solution costs, what score you achieve for your website. Remember, the goal is to get into the top 1% of scores.

Only the best solutions will get extra credit. By best we mean minimum score but also your description of your approach in your report.

Piazza

No discussion of the extra credit on Piazza. Keep your questions on Piazza just about general questions for the project; e.g., don't post code or your PageRank vector etc. If you're unsure make it private.