EECS 485 Project 5: PageRank and HITS

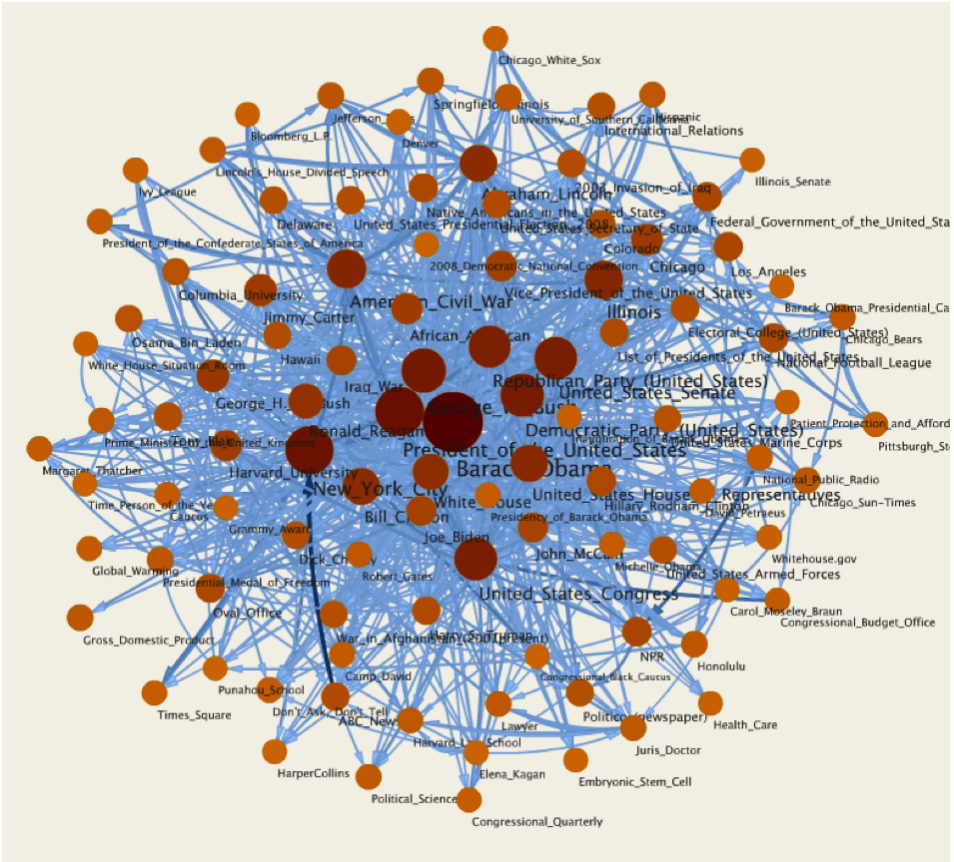
Due 9pm Monday, April 4, 2016

In this assignment you will compute PageRank and HITS values for a portion of a Wikipedia hyperlink graph. This assignment will be standalone: it will not build on your previous work and you do not need to integrate it with your previous inverted index work (at least, not yet). You should focus exclusively on generating an accurate set of PageRank and HITS scores for the hyperlink graph that you are given.

Very Important Note: For this assignment, please use your own laptop or CAEN computers to run your programs. Do not use the EECS servers for this project due to the high computational overhead. Make sure it works on CAEN computers before submitting.

An even bigger note: Do NOT commit input files we give you to Github (they are huge). We give you a .gitignore file just for that reason.

Figure 1. A part of graph we will use



Part 1: PageRank

The Graph

We will provide you a dataset for this assignment (see Google Drive). It is derived from the hypertext graph described by Wikipedia pages, in which each page (e.g., the Barack Obama Wikipedia page) is a node and an intra-Wikipedia link is a directed edge in that graph. We have removed all of the links that connect the Wikipedia pages to outside non-Wikipedia sites.

There's nothing special about using Wikipedia; PageRank works on any kind of hyperlink network and is designed to work on the overall Web. However, Wikipedia is a good use case: the pages are easy to understand, the links are not spammy, and the graph is interesting. The dataset has 362 nodes and all of the directed edges between them. It is small enough that you can trace connections by hand when debugging your code.

The Code

You will implement this project in C++.

You will receive an input file: pagerank.net. The dataset is in the 'Pajek NET' format. The dataset is appropriate to represent the graph structure for our purpose. A simple example of the format from mapequation.org is as follows:

Example 'Pajek NET' file

```
*Vertices 6
1 "Node 1" 1.0
2 "Node 2" 1.0
3 "Node 3" 1.0
4 "Node 4" 1.0
5 "Node 5" 1.0
6 "Node 6" 1.0
*Arcs 8
1 2 2.0
2 3 2.0
3 1 2.0
1 4 1.0
4 5 2.0
5 6 2.0
6 4 2.0
4 1 1.0
```

The two lines beginning with `*` tells us the number of nodes and the number of directed edges in the graph, respectively. From the 2nd line to the 7th line, each line is a pair of node id, node label, and node weight. From the 9th line to the end, each line corresponds to a directed edge with source node id, target node id, and edge weight. In both node lines and edge lines, the weights are optional, and in our dataset, the weights are not included.

The first few lines of our `pagerank.net` file are as follows:

First 5 lines from `pagerank.net`

```
*Vertices 362
534366 "Barack_Obama"
307 "Abraham_Lincoln"
569 "Anthropology"
863 "American_Civil_War"
```

You only need the edge connectivity information in order to compute PageRank; the node labels aren't used by the algorithm. However, you will probably want to make sure that the resulting PageRank weights make sense, giving large weights to important topics and small weights to unimportant ones. Using the node id to node label mapping will help you test and debug your results.

PageRank Details

We went over the PageRank details in class. Be sure to keep in mind a few details:

- Every node must have incoming links. However, you do not need to enforce this explicitly. The (1-d) term describes the “teleport” quantity that will add “virtual” outgoing links from every node to every other node. This will give every node “virtual” incoming links
- For Pagerank to work, every node must have at least one outgoing link - we don’t want sink nodes collecting all of the Pagerank scores for the whole graph. In the event that a node has no outgoing links, create “virtual” outgoing links to every other node in the graph. For such nodes you will evenly distribute all of its outgoing PageRank to all nodes in the graph

Make sure to remove all self edges before processing data. You should implement two different stopping criteria: stop after K iterations, and stop after every node changes no more than X, where:

$$X = \frac{|currentRoundValue - previousRoundValue|}{previousRoundValue}$$

Pagerank Output

Your Pagerank program should write the answer to the output file as a pair of id, pagerank pairs. Each line in the output file should start with the integer id of a page, then a comma, then the pagerank value you have computed for that page. You should not have any pagerank scores of 0. If you do, you may need to change the way you are outputting. For example:

Example output file formatting (sample values only)

```
2553,0.1234
736543,0.1231
.....
```

The order of the output should be decreasing pagerank score.

Running Your Code

To build your code (starter file details at the end), run: `make pagerank`

To run the pagerank script:

```
./pagerank config.pagerank
```

Where config.pagerank is an input file containing various configuration to run pagerank scripts with. The format of the config.pagerank file is as follows:

Example config.pagerank file

```
input.net
2
0.85 k 15 out1.txt
0.75 c 0.01 out2.txt
```

Here, the first line is the name of the .NET file you will be using. The second line is the number of configurations that follow. Lines 3 and onwards will each have 4 values separated by whitespace, where the first value is the d-value, the second is either ‘k’ or ‘c’ indicating number of iterations to use or a threshold value to converge to, value 3 is the associated number for k or c, and the last value is the output file to write the Pagerank results in.

In this sample config file, there are 2 configurations, the first one uses a `d` value of 0.85 and will stop after 15 iterations. It will write the results to a file called `out1.txt`. The second configuration uses a different d value, a percent-change convergence criterion with a threshold

of 0.01, where you can calculate the following and converge if the value is less than the threshold:

$$\frac{|currentRoundValue - previousRoundValue|}{previousRoundValue}$$

We do this to amortize the cost of disk access - reading the input files can take a while. As a result, we want you to read the input file once, store it in memory, and then run multiple test cases.

Part 2: HITS

Graph

The graph data provided for HITS is in the same format as in PageRank. However, since HITS constructs a seed set from the input query, we need additional text data to help choose the nodes. We pre-processed the wiki text data and created an inverted index for you to load and use so no need to compute it yourself.

The Code

As with PageRank, you will implement this project in C++. In addition to the .NET file, your HITS code will also read an inverted index, and will have the following format:

Example inverted index data format

```
Acadians 5042916
Acadie 21184
Acadien 21184
Acadien 731832
Acadiensis 21184
...
```

Each line is a “tuple” where the first column is a word, and the second column is the page id which contains the word. The same word can appear many times in the file. Your code should read both input files (net file and inverted index) and use them to compute hub and authority values.

HITS Details

Follow the general algorithm for HITS covered in the class. A few things to keep in mind are as follows:

- Make sure to remove all self edges before processing data.
- Queries and words in the inverted index should be processed by converting all words to lowercase. You should also be able to handle queries with multiple words using boolean AND.
- Select the seed set by Boolean retrieval. Find top-h (h will be a config value given to you) results from the given query (documentIDs of results in increasing order). The query will also be given to you in the config. To find the relevant results, use Boolean AND - all terms must be present in the document (you don't need to use TF-IDF scoring for this project). This set of result pages will become the seed set.
- Grow the seed set into base set by including pages that are pointed to, or that point to, a page in the seed set. Now calculate hub and authority values iteratively. After every iteration of the algorithm, normalize hub and authority scores so that the sum of the squares of the hub scores is equal to 1 and the sum of the squares of the authority scores is equal to 1.
- You may find that some queries have items in the seed set that are much more popular than others. As a result, the contents of the base set can be quite lopsided.
  e.g., if `yahoo.com/index.html` is in the seed set, then it will bring a huge number of pages into the Base Set, far more than any other page in the seed set. To address this problem, we will limit the number of base set pages "brought in" by a single page. So we'll revise the above algorithm slightly:
  - Find all pages that contain the user's query (i.e. seed set)
  - For each page in the seed set, collect all of the pages that are connected to it and sort them .
  - Of these pages, take the first 50 pages ordered by ID (lowest -> highest) <u>that are not already part of the seed set</u> and add these new pages to the base set. If you try to add a page and see that it is already in the Base Set, you skip it but still count it against the 50.
  - Of course, you will also add everything in seed set to the base set.
- You should implement two different stopping criteria: stop after K iterations, and stop after all Hub and Authority scores for every node changes no more than X. The values X and K will be given in the config file.
- Initialize hub and authority scores of pages in the base set to 1.
- Only compute hub and authority scores for items in the Base Set.
- Remember, you will use the hub and auth values from the LAST round in order to calculate the values in the current iteration (so you will need to keep track of old and new values).

HITS Output

Your program should print the answer as a pair of id, hub, authority pairs. You may have values that are 0 (think about what this case is) but make sure that very small values are not outputted as 0. Each line in the output file should start with the integer id of a page, followed by hub and authority values you have computed for that page (delimited by comma). For example:

Example output file format (sample values only)

```
27,0.1995,0.0201
334,0.1338,0.2502
....
```

The order of the output should be decreasing Hub scores.

Running Your Code

To build your code, run: `make hits`

To run the hits script:

```
./hits config.hits
```

Where config.hits is an input file containing various configuration to run HITS scripts with. The format of the config.hits file is as follows:

Example config.hits file

```
input.net
inverted_index.txt
2
100 k 15 out1.txt pizza
15 c 0.01 out2.txt apple orange
```

Here, the first line is the .NET file you will be using. The second line is the name of the inverted index file to read. The third line is the number of configurations that follow. Lines 4 and onwards will be 5 values (mostly) separated by whitespace, where the first value is the h-value, the second is either ‘k’ or ‘c’ indicating number of iterations to use or a threshold value to converge to, value 3 is the associated number with k or c, and the 4th value is the output file to store the Pagerank results in. After the first 4 values, the rest of the line is the query.

In this sample file, there are 2 configurations, the first one uses a `h` value of 100 and will stop after 15 iterations. It will write the results to a file called `out1.txt`. The second configuration use a different h value, a percent-change convergence criterion with a threshold of 0.01, where you can calculate the following and converge if the value is less than the threshold:

$$\frac{|currentRoundValue - previousRoundValue|}{previousRoundValue}$$

Part 3: Starter Files and Submitting

You will receive the following starter files (in the Drive folder):

- `Makefile`
- `pagerank_code/pagerank.cpp`
- `pagerank_code/pagerank.net`
- `hits_code/hits.cpp`
- `hits_code/hits.net`
- `hits_code/inverted_index.txt`

You will only use the above two CPP files for this project. Do not add any more .h files or .cpp files. When submitting, only submit pagerank.cpp and hits.cpp; all your code should be contained within these two cpp files. Your code must compile normally with the given makefile.

We will post a link to the autograder when it is ready for this project.

In the `README.md` at the root of your repository please provide the following details:

- Group Name (if you have one)
- List the contribution for each team member:
  `User Name (uniqname): "agreed upon" contributions`
- Any need-to-know comments about your site design or implementation.

Other Notes

- We will run it on graphs that may be different from the Wikipedia test graphs, and with different parameter settings than the ones in the sample.
- You will receive full points for a test case if the average distance for each node does not exceed 3% (|ours - yours| / ours).
- You will receive 0 points for the test case if it exceeds 3%.
- Use the given makefiles for compiling & only editing the two .cpp files.
- Make sure it works on CAEN!