CS103 - Pagerank

1 Introduction

You will write a program to rank webpages in an artificial webgraph. Your program will implement Pagerank algorithm [1] used by Google to order search results. Pagerank is not the only algorithm used currently by google to order search results, but it is the first used by the company[2].

This assignment requires you to create C++ classes to model objects in a webgraph such as webpages. You will also use file I/O and stringstreams to read and write text files containing web information. Further, you will understand how to implement directed graphs in C++ that represent webpages and their outbound links. Pages are represented by vertices, while links between pages are directed edges connecting pages.

In this document, the word **link** is used interchangeably with **hyperlink**.

2 Background

The world wide web (WWW) is a network of web pages that are connected through hyperlinks. WWW is described by a webgraph: vertices of the graph correspond to webpages, the directed edges between vertices represent hyperlinks.

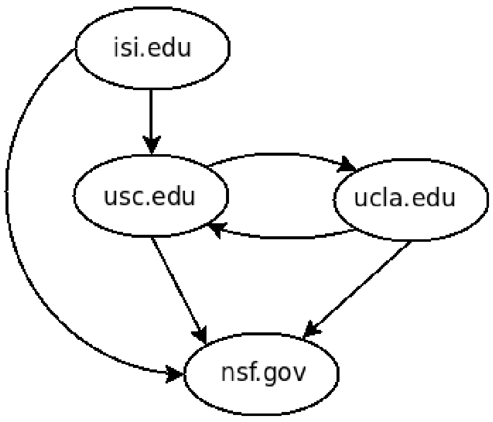


Figure 1: Example of a webgraph

Figure 1 shows an artificial webgraph. The graph shows that there is a link from **isi.edu** to **usc.edu**. It also shows that **usc.edu** and **ucla.edu** both have links to **nsf.gov**.

You will create a **Page** class that represent webpages. A webpage object should contain information about the page name and the links from the page.

Links can be represented by arrays, vectors or lists. Each webpage should store the page ids of the pages it is referencing – have an outbound links to them.

3 Pagerank

Pagerank [1] assigns numerical weights to pages in a webgraph. The purpose of the weights is to measure the importance of a page among the rest. The algorithm attempts to model random surfer's behavior who clicks on links at random. The pagerank of a page represents the probability that a random surfer end up visiting that page.

Pagerank is defined as:

$$PR(A) = \sum_{v \in B_A} \frac{PR(v)}{L(v)}$$

where $PR(A)$ is the Pagerank of page **A**, $PR(v)$ is the Pagerank of pages **v** which link to page **A**, $L(v)$ is the number of outbound links on page **v**, and B_A is the set of pages that link to **A**

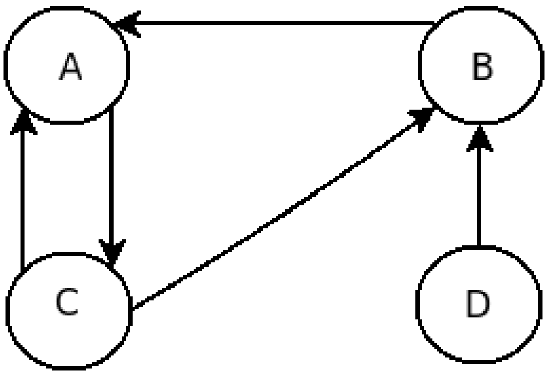


Figure 2: Pagerank example

Consider the graph in Figure 2, where A links to C, B links A, C links to A and B, and D has no incoming links. We get the following equations for the Pagerank calculation:

Since we are dealing with probabilities we have $PR(A) + PR(B) + PR(C) + PR(D) = 1$. We need three more equations to calculate pageranks, we pick:

$$PR(A) = PR(B) + \frac{PR(C)}{2}$$

$$PR(B) = \frac{PR(C)}{2} + PR(D)$$

$$PR(C) = PR(A)$$

Solving the above equations we get

$$PR(A) = \frac{2}{5} = 0.4 \quad PR(B) = \frac{1}{5} = 0.2 \quad PR(C) = \frac{2}{5} = 0.4 \quad PR(D) = 0.0 \tag{1}$$

4 Prelab

The purpose of the **prelab** is to give you intuition into the Pagerank. Before getting into the details, you need to get familiar with the project files and skeleton code.

We provided you a skeleton code. You can check the code by going to Vocareum, select the PA, launch the terminal and type:

```
$ls -1
gen_web.py                      #Python script to generate random graphs
graph_20_1_random.png           #Picture visualizing a graph with 20 pages.
graph_20_1_random.txt           #Description of a graph with 20 pages - visualized in graph_20_1_random.png
graph_30_0.5_random.png
graph_30_0.5_random.txt
graph_rep.png                   #Picture visualizing a graph shown in Figure 2
graph_rep.txt                   #Description of a graph shown in Figure 2
Makefile                        #Makefile to compile your code.
page.h                          #Models a page in the webgraph.
page_rank.cpp                   #main() function that reads graph description from a file,calcuate
                                #pagerank and write results into a file.
readme.txt                      #readme file, where you answer prelab questions.
web.h                           #Web class contains all Pages and implement pagerank algorithm
```

Generate a random graph by using the gen_web.py script. Make sure you are in the assignment directory and run:

```
./gen_web.py -N 4 -s 1 -o mygraph.txt -g
```

This generates a random webgraph with 4 pages similar to the one in Figure 3. The graph description is at mygraph.txt. A picture of the graph is at mygraph.png.

Calculate pageranks of the pages in the graph you generated. Let $PR(id = k)$ be the page rank of the page with id=k. Formulate four equations starting with $PR(id = 0) + PR(id = 1) + PR(id = 2) + PR(id = 3) = 1$ and solve them by any mean you want. Pageranks for pages in the graph shown in Figure 3 is: $PR(id = 0) = \frac{2}{5}$, $PR(id = 1) = \frac{1}{5}$, $PR(id = 2) = \frac{1}{5}$ and $PR(id = 3) = \frac{1}{5}$.

Note: In your readme.txt, include the equations and solution for the pageranks of the graph you generated. Make sure to show your work.

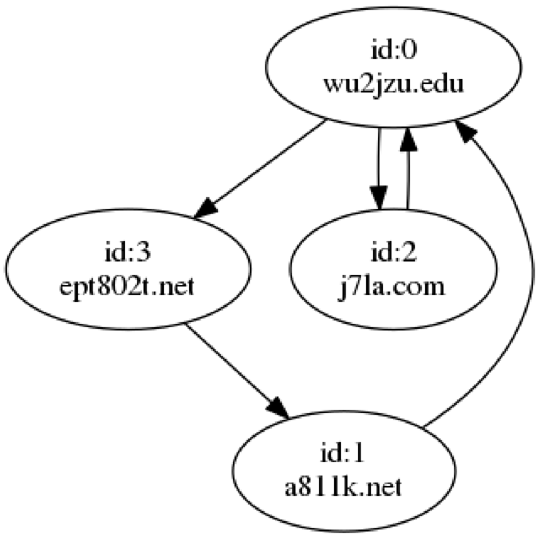


Figure 3: A Random Webgraph

5 Randomwalk and Pagerank

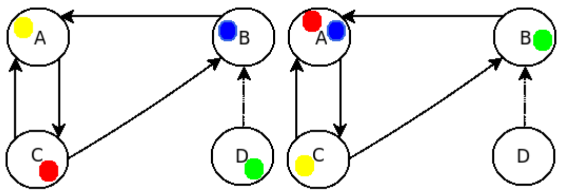
Calculating Pagerank analytically for large webgraph - millions of pages - is computationally challenging due to the large number of variables, and equations. It requires solving millions of equations.

Since PageRank is trying to measure the behavior of a random web surfers traversing the web, we can simulate those surfers. You will use a randomwalk to model the behavior of a random web surfer. In randomwalk, a walker (modeling a web surfers) starts at a random page and keeps moving from one page to another by selecting one of the outgoing edges randomly. In order to simulate, we need some way model the web graph and the links. You will need to create classes to represent pages and their connections. Since the pagerank of a page represents the probability that a random surfer eventually visits that page, you are going to simulate a large number of surfers who randomly click on links. After some time, the population at each page is an estimate of the page rank.

The algorithm is as follows.

```
Let S be the number of simulation iterations.
Let N is the number of walkers.
Divide the walkers equally across pages.
for i = 1 : S do
  Each walker chooses a link randomly from its current page
  and walk over the edge to a new page
done

foreach page p do
  pagerank(p) = the proportion of walkers in page p.
done
```



(a) time $t = 0$ (b) time $t = 1$

Figure 4: Random surfers behavior

Suppose at time $t = 0$ the four walkers, red,green, blue and yellow, are distributed uniformly among the pages in the graph shown in Figure 4. Walker green has only one link to click, which is B, while Red has two options A or B. Yellow and blue both have one link to clicks. Red chooses one randomly let assume it is A. At time $t = 1$, page A has two walkers , while B and A have one each. After simulating randomwalks for some time, you can calculate the pagerank. The proportion of the walkers on a page is an estimate of the pagerank.

6 File format

Your program is required to read and writing from a textfile containing information about the webgraph. The format of the file is as below.

1:	Number of webpages in the file
2:	pageid_0
3:	<TAB> Page URL
4:	<TAB> Page Rank
5:	<TAB> ids of hyperlinked webpages(outgoing links separated by spaces).
...	...
n-3:	pageid_k
n-2:	<TAB> Page URL
n-1:	<TAB> Page Rank
n:	<TAB> ids of hyperlinked webpages(outgoing links separated by spaces).

Below is an example of a file describing the graph shown in Figure 2.

```
4
0
        A.com
        0.0
        2
1
        B.com
        0.0
        0
2
        C.com
        0.0
        0 1
3
        D.com
        0.0
        1
```

7 Procedure

1. Implement a class to model a web page and its outgoing links following the specification in Figure 5.

   The class Page is already defined in page.h. You need to implement the class in page.cpp. You may also need to declare new methods in page.h and implement them in page.cpp. The class should contain:

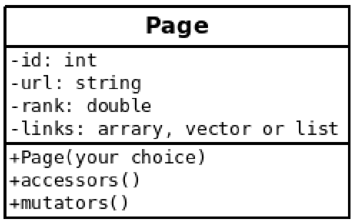


Figure 5: Page class

   (a) An integer to represent the pageid.

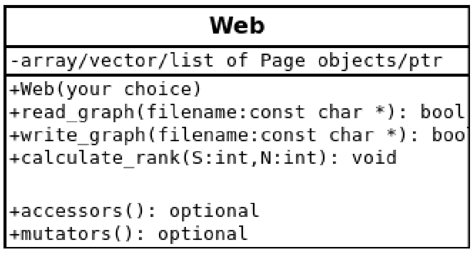
   (b) A string to represent the page URL.

   (c) An array or vector to model the outbound links of the page.

   (d) Methods to access and modify internal data.

2. Implement a class to model the webgraph following the specification in Figure 6. The class should be able to read/write a graph description, and calculate pagerank.

   The class Web defined in web.h models the webgraph. You need to implement web.cpp and you can declare additional methods and data members in web.h



```
Web
-array/vector/list of Page objects/ptr
+Web(your choice)
+read_graph(filename:const char *): bool
+write_graph(filename:const char *): boo
+calculate_rank(S:int,N:int): void

+accessors(): optional
+mutators(): optional
```

Figure 6: Web class

(a) A list of the pages in the webgraph.

(b) A method read_graph() to read a graph from the file.

(c) A method write_graph() to write a webgraph to a file.

(d) A method to calculate the pagerank.

3. The main program is already implemented in page_rank.cpp. The main() creates an object of type Web and instructs it to read from a file, then calculate pagerank and finally, write the graph into a file.

   Through command line arguments main() takes a graph description file, output file, number of random surfers and number of simulation iterations.

   To run the main you can type the following from the command line arguments:

   ```
   $./pagerank <graphinput> <graphoutput> <Number of walkers> <Number of simulations>
   ```

4. Run your program with the example provided with the project files.

   (a) To calculate the page rank of a graph in shown in Figure 2:

   ```
   $./pagerank graph_rep.txt graph_out.txt 1000 16000
   ```

   This reads the webgraph information from graph_rep.txt – graph depicted in figure2, performs randomwalk to calculate pagerank with 1000 walkers for 16000 simulation steps, and write the webgraph with the pagerank in graph_out.txt. The output graph should look like, which is pretty close to what we calculated in Equation 1

   ```
   4
   0
             A
              0.397
             2
   1
             B
              0.199
             0
   2
             C
              0.404
             0  1
   3
             D
             0
             1
   ```

   (b) Calculate the pagerank of graph_20_1_random.txt by running

   ```
   ./pagerank graph_20_1_random.txt graph_20_1_random_out.txt 10000 16000
   ```

   Page rank of **hrs5lwt8.mil** should be close to 0.3595, **2005b.gov** 0.3672, and **s3b.com** is 0.2733.

   (c) Pagerank of graph_30_0.5_random.txt

   ```
   ./pagerank graph_30_0.5_random.txt graph_30_0.5_random_out.txt 10000 16000
   ```

   Pagerank of **vnm1um.edu** is 0.3631, **1cr.mil** is 0.311 and **2s.net** is 0.3259.

8 Submission

In addition to completed web.h, web.cpp, page.h, page.cpp and page_rank.cpp files, you need to include a readme.txt with the solution to the pagerank equations for the random graph you generated. Be sure to include **mygraph.txt** as well.

References

[1] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web.,” Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[2] Wikipedia, “Pagerank — wikipedia, the free encyclopedia,” 2017. [Online; accessed 28-October-2017].