Com S 435X/535X Programming Assignment 3, Part I
300 Points
Due: April 8, 11:59PM
Late Submission Due: April 9, 11:59PM(25% Penalty)

In Part I of this programming assignment, you will write a web crawlers that crawls pages from `wikipedia` and will compute page ranks for the crawled pages.

Part II of PA3 will be on information retrieval and will posted later and most likely will be due on April 15.

Note that the description of a programming assignment is not a linear narrative and may require multiple readings before things start to click. You are encouraged to consult instructor/Teaching Assistant for any questions/clarifications regarding the assignment.

For this assignment, you may work in groups of two.

1 WikiCrawler

This class will have methods that can be used to crawl Wiki. Instead of crawling entire wikipedia, you will do a *focussed crawling*–Only crawl pages that are about a particular topic. This class should build a web graph of the crawled pages.

This class should have following constructor and methods.

`WikiCrawler.` parameters to the constructor are

1. A string *seedUrl*–relative address of the seed url
2. Array of Strings *keywords* –contains key words that describe a topic
3. An integer *max* representing Maximum number sites to be crawled
4. A string $fileName$ representing name of a file–The graph will be written to this file

`crawl()` Method to collect *max* many pages. Every collected page must have all of the words from *keywords*. This method should construct the web graph of all collected pages and write the graph to the file $fileName$.

For example, `WikiCrawler` can be used in a program as follows

```
String[] topics = {"tennis", "grand slam"};
WikiCrawler w = new WikiCrawler("/wiki/Tennis", topics, 100, "WikiTennisGraph.txt");
w.crawl();
```

This program will start crawling with `/wiki/Tennis` as the root page. Collects 100 wiki pages that contain the both the words `tennis` and `grand slam` and stores the graph over these 100 pages in a file named `WikiTennisGraph.txt`. This file will list all edges of the graph. Each line of this file should have one directed edge, except the first line. The first line of the graph should indicate number of vertices. Below is sample contents of the file

```
100
/wiki/tennis /wiki/Tennis_ball
/wiki/tennis /wiki/Tennis_court
```

```
/wiki/tennis /wiki/Wheelchair_tennis
/wiki/tennis /wiki/England
/wiki/tennis /wiki/Real_tennis

... ...
.. ...
... ...
```

The first line tells that there is a link from page `/wiki/Tennis` to the page `/wiki/Tennis_ball`.

1.1 Clarifications and Suggestions

1. The seed url is specified as *relative address*; for example `/wiki/Tennis` not as `http://en.wikipedia.org/wiki/Tennis`.

2. Extract only links from "actual text component". A typical wiki page will have a panel on the left hand side that contains some navigational links. Do not extract any of such links. Wiki pages have a nice structure that enables us to do this easily. The "actual text content" of the page starts immediately after the first occurrence of the html tag `<p>`.

3. Take a look at wiki tennis page. This page has a link to page `/wiki/Croquet` (in the second paragraph). The page `/wiki/Croquet` does not contain both words `tennis` and `grand slam`. Thus this page not about our topics and thus should not be one of the 100 pages.

4. Well, the page `/wiki/Tennis` has a link to `/wiki/Croquet`. How can the program know that the page `/wiki/Croquet` is not about about our topic? The only way is to download the page `/wiki/Croquet` and check if that page contains both the words `tennis` and `grand slam`. Thus even though the page `/wiki/Croquet` is not among the 100 pages that we wish to collect, your crawler must still send a request and download that page.

5. How to check if a page has both the key words `tennis` and `grand slam`? One way is to download the actual `html source code` and check for key words. However, there is a (somewhat) quicker way. For every wiki page, there is a `wiki raw text page`. And the size of the `raw text page` is much smaller than the size of the `html source page`. Thus it takes less time to search for `tennis` and `grand slam` in the raw text page. The raw text page for /wiki/Croquet is at

   `http://en.wikipedia.org/w/index.php?title=Croquet&action=raw`

   For every wiki page `/wiki/xxxx`, its raw text page is at

   `http://en.wikipedia.org/w/index.php?title=xxxx&action=raw`

   Your program can use the raw text pages.

6. Your search for keywords in a page must be case-insensitive.

7. Your program should only form the graph of pages from the domain `http://en.wikipedia.org`

8. Your program should not explore any wiki link that contain the characters "#" or ":". Links that contain these characters are either links to images or links to sections of other pages.

9. Finally, the graph you constructed should not have self loops nor it should have multiple edges.

10. If you wish you take a look at the graph that the crawler that I wrote constructed (with key words tennis and grand slam). This graph has 2707 edges and my crawler took nearly 90 seconds. My crawler program sent a request to wiki 325 times. The graph is attached along with the PA description.

11. You must follow politeness policies. Download `robots.txt` file and do not crawl any site that is `disallowed`. Your program should not continuously send requests to wiki. The program I wrote waited for 5 seconds after every 100 requests.

12. Your program must be robust to any network errors. If a web page can not be accessed for any reason, it will generate an exception, your program should catch the exception and proceed.

13. Your program should not use any external packages to parse html pages, to extract links from html pages and to extract text from html pages. You can only use inbuilt packages of Java that are of the form `java.*`. I used `java.net`

1.2 Focussed Crawlers

Write a class named `WikiTennisCrawler` containing a main method. The seed url is `/wiki/Tennis`. This program should crawl 1000 pages that contain the keys words `tennis` and `grand slam`. Store the graph constructed in a file named `WikiTennisGraph.txt`

Write a another class named `MyWikiCrawler` with a main method. Pick your favorite topic and key words that describe the topic. Form a graph by collecting at least 1000 pages. Store the graph constructed in a file named `MyWikiGraph.txt`.

2 PageRank

This class will have methods to compute page rank of nodes/pages of a web graph. This class should have following methods and constructors.

The constructor `PageRank` will following parameters

1. Name of a file that contains the edges of the graph. You may assume that the first line of this graph lists the number of vertices, and every line (except first) lists one edge. You may assume that each vertex is a wiki url represented as string.

2. ϵ; approximation parameter for pagerank.

A method named `pageRankOf` its gets name of vertex of the graph as parameter and returns its page rank.

A method named `outDegreeOf` its gets name of vertex of the graph as parameter and returns its out degree.

A method named `inDegreeOf` its gets name of vertex of the graph as parameter and returns its in degree.

A method named `numEdges` that returns number of edges of the graph.

A method named `topKPageRank` that gets an integer k as parameter and returns an array (of strings) of pages with top k page ranks.

A method named `topKInDegree` that gets an integer k as parameter and returns an array (of strings) of pages with top k in degree.

A method named `topKOutDegree` that gets an integer k as parameter and returns an array (of strings) of pages with top k out degree.

2.1 Crawling and Page Rank

Write a program named `WikiTennisRanker`. This program will have a main method that will do the following:

1. Construct a wiki tennis graph as in Subsection 1.2. Compute the page ranks with $\epsilon = 0.1$ as approximation factor. Output pages with highest page rank, highest in-degree and highest out-degree. Compute the following sets: Top 100 pages as per page rank, top 100 pages as per in-degree and top 100 pages as per out degree. For each pair of the sets, compute Jaccard Similarity. Repeat the same with $\epsilon = 0.05$ as approximation factor.

Write another program named `MyWikiRanker` that does exactly same as above with the graph `MyWikiGraph.txt` constructed in Subsection 1.2

3 Report

This section will be added later

4 What to Submit

- WikiCrawler.java
- WikiTennisCrawler.java
- MyWikiCrawler.java
- PageRank.java
- WikiTennisranker.java
- MyWikiRanker.java
- report.pdf

Only one submission per group please.