Com S 311 Programming Assignment 2
500 Points
Due: April 5, 11:59PM
Late Submission Due: April 6, 11:59PM(25% Penalty)

In this programming assignment, you will implement graph traversal algorithms (bfs and dfs) and the algorithm to compute Strongly Connected Components. This assignment has two parts. In the first part, you will use BFS to implement a crawler to crawl and discover the `wikipedia` graph. In the second part, you will implement algorithm to compute strongly connected components (SCC) . We haven't yet covered the SCC algorithm in lectures, will do during the first week after the break.

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 PA, you may work in `teams of 2`. It is your responsibility to find a team member. If you can not find a team member, then you must work on your own. You are allowed to use Java's inbuilt data structures (such as hashMap, hashSet, etc). However, you **not allowed to use any external libraries/packages**.

Your code must strictly *adhere to the specifications.* The names of methods and classes must be exactly as specified. The return types of the methods must be exactly as specified. For each method/constructor the types and order of parameters must be exactly as specified. Otherwise, you will lose a significant portion points (even if your program is correct). All your classes must be in the **default** package. Your grade will depend on *adherence to specifications, correctness of the methods/programs, and efficiency of the methods/programs.* If your programs do not compile, you will receive **zero** credit.

1 BFS and Web Graph

We can the model web as a directed graph. Every web page is a vertex of the graph. We put a directed edge from a page p to a page q, if page p contains a link to page q.

Recall the BFS algorithm from the lecture.

1. Input: Directed Graph $G = (V, E)$, and $root \in V$.
2. Initialize a Queue Q and a list *visited*.
3. Place *root* in Q and *visited*.
4. **while** Q is not empty **Do**
   (a) Let v be the first element of Q.
   (b) For every edge $\langle v, u\rangle \in E$ DO
   - If $u \notin$ *visited* add u to the **end of** Q, and add u to *visited*.

If you output the vertices in *visited*, that will be BFS traversal of the input graph.

We can use BFS algorithm on web graph also. Here, the input to the algorithm is a *seed url* (instead of entire web graph). View that page as the the root for the BFS traversal. Here is the BFS algorithm on Web graph.

1. Input: *seed url*
2. Initialize a Queue Q and a list *visited*.
3. Place *seed url* in Q and *visited*.
4. **while** Q is not empty **Do**
   (a) Let *currentPage* be the first element of Q.
   (b) Send a request to server at *currentPage* and download *currentPage*.
   (c) Extract all links from *currentPage*.
   (d) For every link u that appears in *currentPage*
      - If $u \notin visited$ add u to the **end of** Q, and add u to *visited*.

This will visit all the pages that are reachable from the `seed url`.

1.1 Crawling and Constructing Web Graph

One of the first tasks of a web search engine is to *crawl* the web to collect material from all web pages. While doing this, the the search engine will also construct the *web graph*. The structure of this graph is critical in determining the pages that are relevant to a query.

In this part of the assignment you will write a program to crawl the web and construct web graph. However, web graph is too large, and you do not have computational resources to construct the entire web graph (unless you own a super computer). Thus your program will crawl and construct the web graph over 500 pages. Your program will crawl only *Wiki* pages.

1.2 WikiCrawler

This class will have methods that can be used to crawl Wiki. This class should have following constructor and methods.

`WikiCrawler.` parameters to the constructor are

1. A string *seedUrl*–relative address of the seed url (within Wiki domain).
2. An integer *max* representing Maximum number pages to be crawled
3. A string *fileName* representing name of a file–The graph will be written to this file

`extractLinks(String doc)`. This method gets a string (that represents contents of a .html file) as parameter. This method should return an array list (of Strings) consisting of links from `doc`. Type of this method is `ArrayList<String>`. You may assume that the html page is the source (html) code of a wiki page. This method must

- Extract only `wiki` links. I.e. only links that are of form `/wiki/XXXXX`.
- Only extract links that appear after the first occurrence of the html tag `<p>` (or `<P>`).
- Should not extract any wiki link that contain the characters "#" or ":".

- The order in which the links in the returned array list must be exactly the same order in which they appear in `doc`.

For example, if `doc` looks like the attached file (`sample.txt`), then
Then the returned list must be

```
/wiki/Science,  /wiki/Computation,  /wiki/Procedure_(computer_science),
 /wiki/Algorithm, /wiki/Information, /wiki/CiteSeerX, /wiki/Charles_Babbage
```

`crawl()` This method should construct the web graph over following pages: Consider the **first** `max` many pages that are visited when you do a BFS with `seedUrl`. Your program should construct the web graph **only** over those pages. and writes the graph to the file $fileName$.

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

```
WikiCrawler w = new WikiCrawler("/wiki/Computer_Science",  100, "WikiCS.txt");
w.crawl();
```

This program will start crawling with `/wiki/Computer_Science` as the root page. It will collect the **first 100 pages that are visited by a BFS algorithm**. Determines the graph (links among the those 100 pages) and writes the graph to a (text) file named `WikiCS.txt`. This file will contain 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/Computer_science /wiki/Science
/wiki/Computer_science /wiki/Computation
/wiki/Computer_science /wiki/Procedure_(computer_science)
/wiki/Computer_science /wiki/Algorithm
/wiki/Computer_science /wiki/Information
/wiki/Computer_science /wiki/Computer_scientist
.....
.....
.....
```

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

All other methods of the class must be `private`.

1.2.1 Guideline, Clarifications and Suggestions for WikiCrawler

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

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>`. So, your program should extract links (pages) that appear after the first occurrence of `<p>`. Your program must extract the links in the order they appear in the page, and place them in Q in that order only.

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

4. 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.

5. The graph constructed **should not have self loops nor it should have multiple edges.** (even though a page might refer to itself or refer to another page multiple times).

6. If you wish you may take a look at the graph our crawler constructed (With `/wiki/Complexity_theory` as root and 20 as maximum number of pages) This graph has 115 edges. The graph is attached along with the assignment (named `wikiCC.txt`). (Crawl Date: Fri, March 10, 2017, 5PM).

7. Crawlers place considerable load on the servers (from which they are requesting a page). If your program continuously sends request to a server, you may be denied access. Thus your program must adhere to the following politeness policy: Wait for at least 3 seconds after every 100 requests. **If you do not adhere to this policy you will receive ZERO credit.**

8. **Your class `WikiCrawler` must declare a `static, final` global variable named `BASE_URL` with value `https://en.wikipedia.org` and use in conjunction with links of the form `/wiki/XXXX` when sending a request fetch a page. Otherwise, you will receive ZERO credit.** For example, your code to fetch page at `https://en.wikipedia.org/wiki/Physics` could be

```
URL url = new URL(BASE_URL+"/wiki/Physics");
InputStream is = url.openStream();
BufferedReader br = new BufferedReader(new InputStreamReader(is));
```

9. 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.

2 Graph Processor

You will design and implement a class named `GraphProcessor` that will read a graph stored in a file, implements Strongly Connected Components (SCC) algorithm discussed in lectures, build appropriate data structures to answer following queries efficiently:

- Out degree of a vertex
- Whether two vertices are in the same SCC
- Number of SCC's of the graph
- Size of the largest component
- Given a vertex v, find all vertices that belong to the same SCC as v

- Find shortest (BFS) path from a vertex v to u.

Design a class named `GraphProcessor`. This class will have following constructor.

`GraphProcessor(String graphData)` `graphData` hold the absolute path of a file that stores a directed graph. This file will be of the following format: First line indicates number of vertices. Each subsequent line lists a directed edge of the graph. The vertices of this graph are represented as strings. For example contents of the file may look like

```
4
Ames  Minneapolis
Minneapolis Chicago
Chicago Ames
Ames Omaha
Omaha Chicago
```

This class should create **efficient data structures** so that the following `public` methods of the class run **efficiently**.

`outDegree(String v)` Returns the out degree of v.

`sameComponent(String u, String v)` Returns `true` if u and v belong to the same SCC; otherwise returns `false`.

`componentVertices(String v)` Return all the vertices that belong to the same Strongly Connected Component of v (including v). This method must return an array list of `String`s, thus the return type is `ArrayList<String>`

`largestComponent()` Returns the size of the largest component.

`numComponents()` Returns the number of strongly connect components.

`bfsPath(String u, string v)` Returns the BFS path from u to v. This method returns an array list of strings that represents the BFS path from u to v. Note that this method must return an array list of `String`s. First vertex in the path must be u and the last vertex must be v. If there is no path from u to v, then this method returns an `empty list`. The return type is `ArrayList<String>`

Any other methods that you write must be `private`. For example, the methods that you write to do any pre-processing, computing SCC, performing BFS/DFS, and creating necessary data structures must be `private`. Only the above listed methods must be `public`. You may write additional classes.

3 Report

Using your crawler program, crawl the wiki pages with `/wiki/Computer_Science` as seedUrl. This program should crawl 500 pages and write the constructed graph to a file named `WikiCS.txt`. Using `GraphProcessor`, compute the following statistics for the graph `WikiCS.txt`

- Vertex with highest out degree
- Number of components of the graph
- Size of the largest component.

Write a report that describes the following:

- Data structures used for Q and *visited*. Your rationale behind the choice of data structures.
- Number of edges and vertices in the graph `WikiCS.txt`
- Vertex with largest out degree in the graph `WikiCS.txt`
- Number of strongly connected components in `WikiCS.txt`
- Size of the largest component in `WikiCS.txt`
- The data structures that you built/used in `GraphProcessor`
- Analyze and report the asymptotic run time for each of the public methods from the class `GraphProcessor`. Note that the run times depend on the choice of your data structures. Your grade partly depends on your the asymptotic run time of these methods (which in turn depends on choice of data structures).

4 What to Submit

- WikiCrawler.java
- GraphProcessor.java
- report.pdf (must be in pdf format)
- Any Additional classes that you created.
- ReadMe.txt (list the name(s) of team member(s)).

You must include any additional helper classed that you designed. Please include only source .java files, do not include any .class files. Please remember to use default package for all the java classes. Place all the files that need to be submitted in a folder (without any sub-folders) and `zip` that folder. Submit the `.zip` file. Only `zip` files please. Please include all team members names as a JavaDoc comment in each of the Java files. Only **one** submission per team please.