

Com S 311 Programming Assignment 2

500 Points

Due: April 20, 11:59PM

Late Submission Due: April 21, 11:59PM(25% Penalty)

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 BFS algorithm to compute certain graph properties.

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 1 or 2 or 3**. It is your responsibility to find a team member. If you can not find a team member, then you must work on your own. **Only one submission per team please.** 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.

Please note that the description of a PA is not a linear narrative. It may take multiple readings before things start to click.

1 BFS and Web Graph

We can 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 $u \notin \textit{visited}$ 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**.

2 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 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 do a *focussed crawling* of 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 100 pages. Your program will crawl only *Wiki* pages.

3 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. An array list of keywords called *topics*. The keywords describe a particular topic.
4. A string *fileName* representing name of a file—The graph will be written to this file

crawl() This method should construct the web graph over following pages: If **seedUrl** does not contain all words from *topics*, then the graph constructed is empty graph(0 vertices and 0 edges). Suppose that **seedUrl** contains all words from *topics*. Consider the **first max** many pages (including **seedUrl** page), *that contain every word from the list topics*, and are visited when you do a BFS with **seedUrl** as root. Your program should construct the web graph **only** over those pages, and write the graph to the file *fileName*.

For example, **WikiCrawler** can be used in a program as follows. Say *topics* has strings *Iowa State*, *Cyclones*.

```
WikiCrawler w = new WikiCrawler("/wiki/Iowa_State_University", 100, topics, "WikiISU.txt");
w.crawl();
```

This program will start crawling with `/wiki/Iowa_State_University` as the root page. It will collect the **first 100 pages that contain both the words “Iowa State” and ”Cyclones” 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 `WikiISU.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 which will be 100. Below is sample contents of the file

```
100
/wiki/Iowa_State_University /wiki/Ames,_Iowa
/wiki/Iowa_State_University /wiki/Iowa
/wiki/Iowa_State_University /wiki/Iowa_State_Cyclones
/wiki/Iowa_State_University /wiki/Big_12_Conference
/wiki/Iowa_State_University /wiki/Cy_the_Cardinal
/wiki/Iowa_State_University /wiki/Campus_of_Iowa_State_University
/wiki/Iowa_State_University /wiki/Jack_Trice_Stadium
/wiki/Iowa_State_University /wiki/Iowa_State_Center
/wiki/Iowa_State_University /wiki/Midwestern_United_States
/wiki/Iowa_State_University /wiki/ISUtv
.....
.....
.....
```

The first line tells that there is a link from page `/wiki/Iowa_State_University` to the page `/wiki/Ames,_Iowa`.

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

3.1 Guidelines, Clarifications and Suggestions for Part 1

1. Here is an example. Suppose that there following pages. $A, B, C, D, E, F, G, H, I, J$. Page A has links to B, C and D appearing in that order. Page C has links to E, F, B and D . Page D has links to G, H and A . Page B has links to I and J . Page E has two links to page A . None of the other pages have any links. Suppose that page B does not contain the strings `cyclones`, `hilton`, but all other pages have both of these words. Now if you perform BFS (crawl) with A as root, `cyclones`, `hilton` as topics, and 5 as *max*: Starting at A , you will first visit B, C and D . However, B does not the words from topics. So Page B will not be considered in future while doing BFS. The resulting output graph will have the vertices A, C, D, E, F with edges from A to C , A to D , C to E , C to F , C to D , E to A , and D to A .
2. The seed url is specified as *relative address*; for example `/wiki/Iowa_State_University` not as `https://en.wikipedia.org/wiki/Iowa_State_University`.
3. Extract only links from “actual text component” of wiki pages. A typical wiki page will have a panel on the left hand side that contains some navigational links, that is not “actual text

component”. For this PA, you may assume the following: 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**.

4. When determining whether a page contains all words from *topics*, your algorithm must check if the “actual text component” contains all words from *topics*.
5. Your program should form the graph of pages from the domain `https://en.wikipedia.org` only.
6. Your program should extract only links of the form `/wiki/XXXX`. For example, it should ignore any links of the form `https://en.wikipedia.org/wiki/XXX`
7. 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.
8. 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).
9. Your program must work even if *topics* list is empty.
10. 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, and empty list as topics). The graph is attached along with the assignment (named `wikiCC.txt`). Crawl Date: Tuesday, April 3, 2:00PM.
11. 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 25 requests. **If you do not adhere to this policy you will receive ZERO credit**. Please use `Thread.sleep()` for waiting.
12. **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. No exceptions.** 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));
```

When we grade your PA, we will change the value of `BASE_URL` and run the program. So it is imperative that you use this static, final variable.

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 the packages of the form `java.*`.
14. If most of you work in the last hour and send requests to wiki pages, it is quite possible that wiki may slow down, or deny service to any request from the domain `iastate.edu`. This can not be an excuse for late submissions or extending deadline. You are advised to start working on crawler as soon as possible.

4 Influential Nodes in a Network

How does an opinion or information spread in a social network such as Twitter or Facebook? When an individual expresses a opinion (say on Twitter), how does that opinion propagated over the network? When do we say one person *more influential* than other person in a social network? Who are the most influential people in a social network? Can you identify most influential people in a network? These problems are of importance in social network analysis. In this PA, you will learn a way to formalize these questions and write a program to identify influential people in a network.

Let $G = (V, E)$ be a *directed, unweighted* graph. Given two vertices x and y , $dist(x, y)$ denotes the length of the shortest path between x and y . Distance of a vertex to itself is 0. I.e., $dist(x, x) = 0$. Let $S \subseteq V$ a set of vertices and y be a vertex. Distance from S to y is

$$dist(S, y) = \min\{dist(x, y) \mid x \in S\}$$

A simplistic way to model influence/opinion propagation as follows: Given a node x , it will influence $1/2$ the nodes at distance 1 from x , one-fourth of the nodes at distance 2 from x , one-eighth of the nodes at distance 3 from x , and so on. For formally, given a vertex x , the *influence of* x , denoted by $Inf(x)$, is defined as

$$Inf(x) = \sum_{i=0}^n \frac{1}{2^i} \times |\{y \mid dist(x, y) = i\}|$$

Here $|A|$ denotes elements in set A . Note that

$$\{y \mid dist(x, y) = i\}$$

is a set—not a multi-set. I.e, there are no duplicates.

Similarly given $S \subset V$, *influence of* S , denoted by $Inf(S)$, is

$$Inf(S) = \sum_{i=0}^n \frac{1}{2^i} \times |\{y \mid dist(S, y) = i\}|$$

We are interested in the following computational problem. Given a network/graph G and an integer k , find a most-influential set $S \subseteq V$ of size atmost k . I.e, which set of k people/nodes are the most influential? In general, this problem is very hard to solve and in NP-Complete (you will learn about NP-Completeness latter in the course). Thus we settle for some heuristic algorithms. Here we discuss three algorithms and you should implement these algorithms (along with a few other algorithms).

Degree Greedy. Output nodes with top k outdegrees.

Modular Greedy. For each vertex x in G compute $Inf(x)$. Now, S is the set of nodes with top k values of $Inf(x)$.

SubModular Greedy. This algorithm works as follows:

1. Input $G = (V, E)$, k .
2. $S = \emptyset$.
3. For $i = 1$ to k DO
 - (a) Find a node $v \notin S$ such that for every $u \neq v$, $Inf(S \cup \{v\}) \geq Inf(S \cup \{u\})$.
 - (b) $S = S \cup \{v\}$.

It can be proved that this algorithm has an approximation ratio of $(1 - 1/e)$ (Proof is beyond the scope of this course).

4.1 Your Task

Design a class named `NetworkInfluence`. This class will have the following constructor.

`NetworkInfluence(String graphData)` `graphData` holds 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 have following methods.

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

`shortestPath(String u, String v)` Returns a BFS path from u to v . This method returns an array list of strings that represents a shortest path from u to v . Note that this method must return an array list of `Strings`. 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>`

`distance(String u, String v)`. Returns $dist(u, v)$. Type is `int`.

`distance(ArrayList<String> s, String v)`. Here s a subset of vertices. This method returns $dist(s, v)$. Type is `int`.

`influence(String u)`. Returns $Inf(x)$. Return type is `float`.

`influence(ArrayList<String> s)`. Here s holds a set of vertices of G . This method returns $Inf(s)$ whose type is `float`.

`mostInfluentialDegree(int k)`. Returns a set of k nodes obtained by using **Degree Greedy** algorithm. Return type must be `ArrayList<String>`.

`mostInfluentialModular(int k)`. Returns a set of k nodes obtained by using **Modular Greedy** algorithm. Return type must be `ArrayList<String>`.

`mostInfluentialSubModular(int k)`. Returns a set of k nodes obtained by using **SubModular Greedy** algorithm. Return type must be `ArrayList<String>`.

Any other methods that you write must be **private**. Only the above listed methods must be **public**. You may write additional classes.

5 Report

Using your crawler program, crawl the wiki pages with `/wiki/Computer.Science` as `seedUrl`, 100 as `max`, and empty list as `topics`. Write the constructed graph to a file named `WikiCS.txt`. Using **NetworkInfluence**, compute the top 10 most influential nodes using each of the three algorithms the graph `WikiCS.txt`. 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`
- Top 10 most influential nodes computed by `mostInfluentialDegree` and the influence of those nodes.
- Top 10 most influential nodes computed by `mostInfluentialModular` and the influence of those nodes.
- Top 10 most influential nodes computed by `mostInfluentialSubModular` and the influence of those nodes.
- Pseudo code for the constructor and the public methods from the class **NetworkInfluence**
- Analyze and report the asymptotic run time for each of the public methods and the constructor from the class **NetworkInfluence**. 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).

6 What to Submit

- WikiCrawler.java
- NetworkInfluence.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. Name of the folder must be **YouNetID-PA2**. 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**.