CS 5329 – Programming Project – Group 1

Submitted by:

1. Madhurika Gelli
2. Sayali Patil
3. Sanya Kapoor

2. Explanation for the “Web Crawler” project

* 1. HOW TO RUN

**Files Included**

* GraphDiameter.java
* Node.java
* Result.java
* GraphDiameter.jar

**Output Generated**

* Output.txt

**How to Run**

Copy the above files to the location and check if java is available in that location.

>> cd <FOLDER\_WITH\_FILES>

>> java -version // check

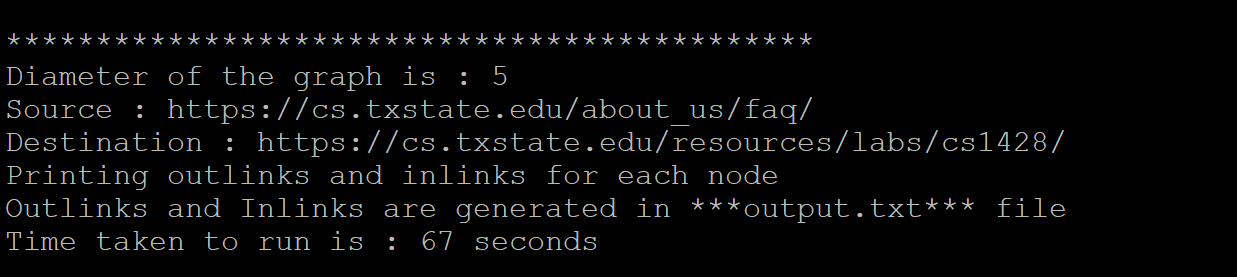
Run the code

>> java – jar GraphDiameter.java

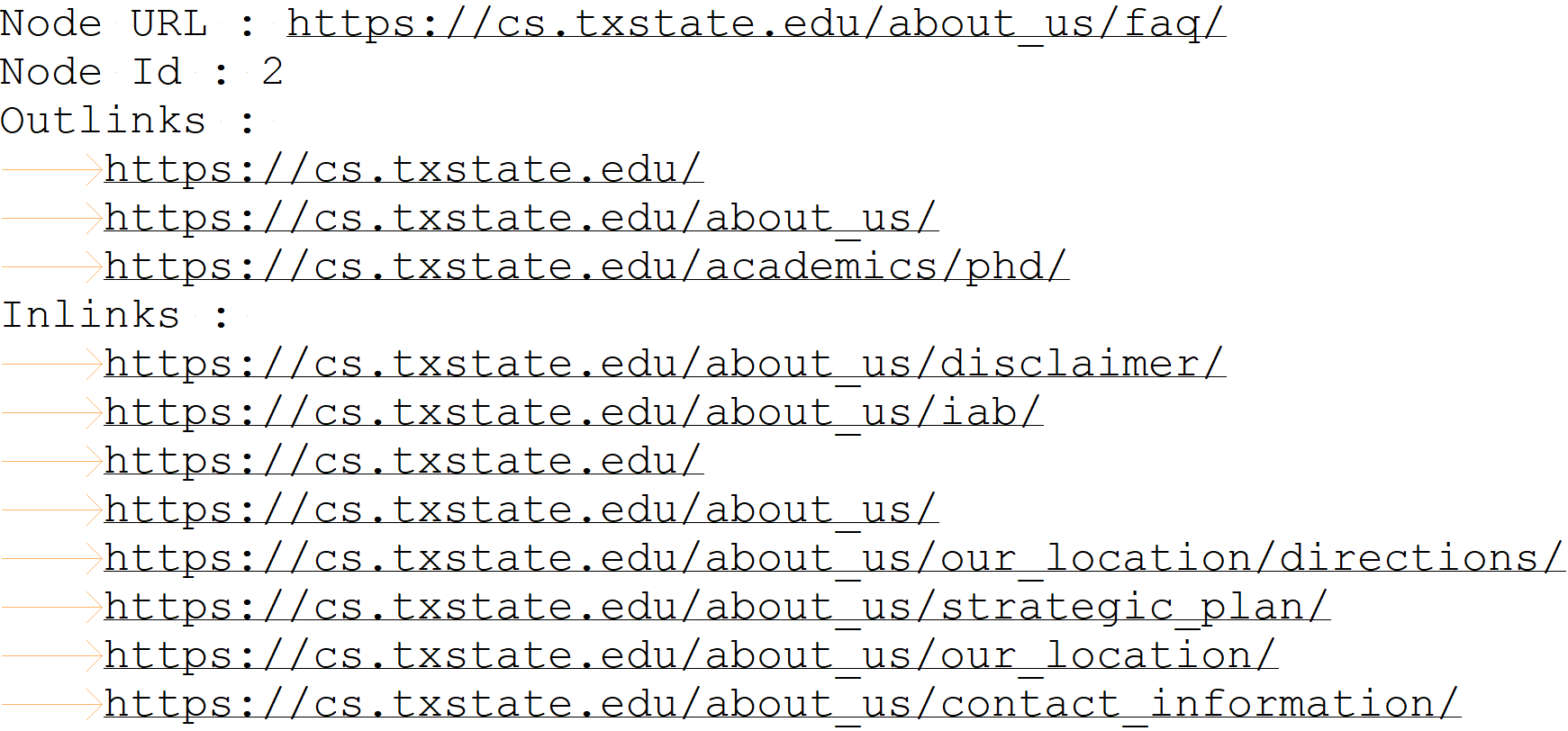


Output

* The diameter of the graph the source and destination are displayed on the screen.
* The inlinks and outlinks for each node are written into a file “output.txt” in the same location as running the file.



* Open output.txt with any editor. Sample output for a node. (A complete output.txt is attached)



* 1. Step by Step Process
     1. Crawling CS domain

**CRAWL FUNCTION**

We are crawling the CS domain starting from the homepage (cs.txstate.edu) using the [Jsoup crawler](https://jsoup.org/). As we crawl the CS domain home page, we look for all the links on the page with “a[href]” to determine the outlinks. So, we create a directed edge between source and outlink. Here is the sample code:



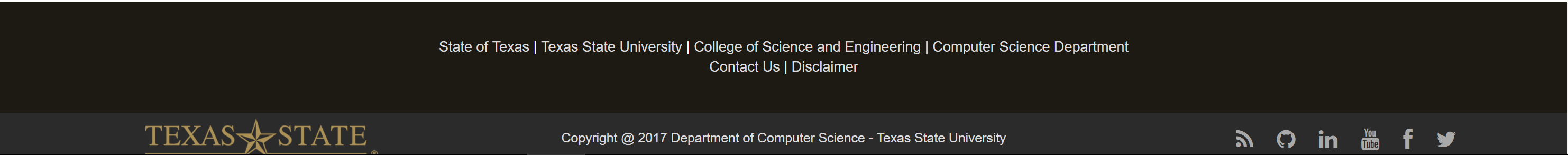
**IGNORE HEADERS AND FOOTERS**

During finding outlinks on a page, we ignore the header and footer section on the page. Because, these links exist in every page. We may get an incorrect diameter considering these pages (Note: we got diameter as 2 considering header and footer on every page).

Header:



Footer:

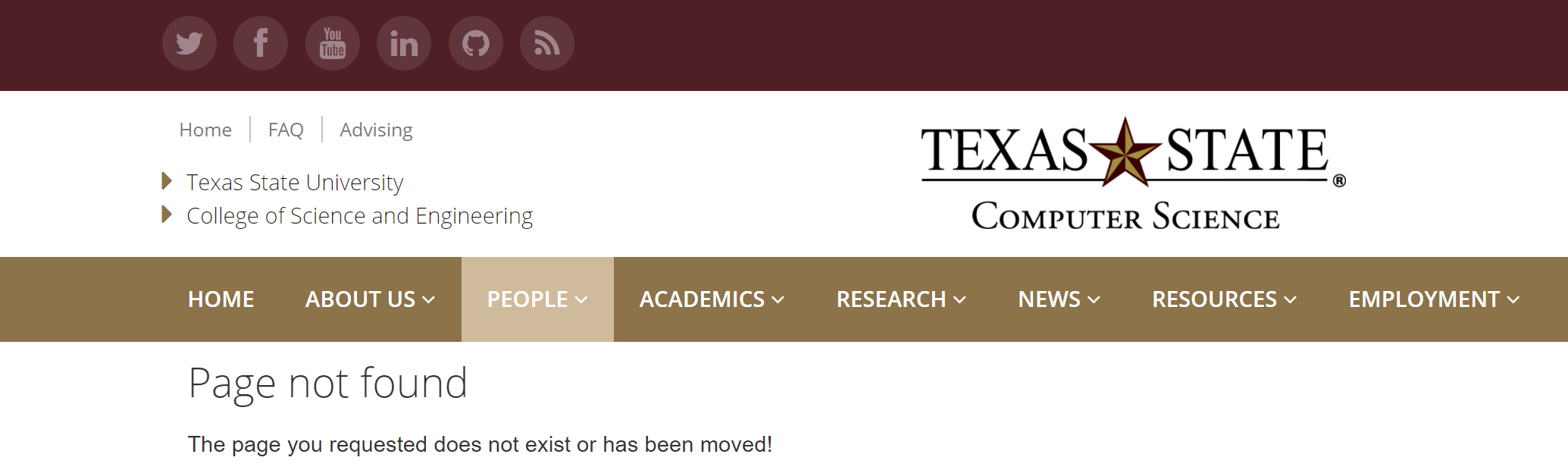


Sample Code:



**IGNORE BAD PAGES & SET TIMEOUT**

We ignore pages which give 404 error, “Page Not Found” error.



We set a timeout of 8 seconds for JSOUP crawler to connect to page before it can crawl for links.

Sample code: 

**BREADTH FIRST SEARCH**

We use Breadth First Search technique to crawl all the individual links on the page. For BFS, we use a Queue data structure to store the URL’s to crawl and HashSet – to store already crawled URL’s, so as to not crawl them again. Here is the sample code and code highlighted in yellow:



**VALIDATE & FORMAT URL’S**

During the process of finding the links on a page, we also format URL’s by removing and trailing “/”or “#” character and remove query parameters “?s\_id=abc” and “#caraousel-1” etc. We also ignore page which end with “.doc”, “.cpp”, “.docx” etc. as they are not considered valid URL’s. Sample code:



* + 1. GRAPH & NODE STRUCTURE

We store graph as adjacency list for each Node. The “generateAdjacencyMatrix()” function below will return an adjacency matrix(graph). Each graph node consists of URL, Id, and Set of incoming links and outgoing links. Sample Code below:



* + 1. CALCULATE DIAMETER OF GRAPH

**Definition**

The graph generated is directed with non-negative weights. Diameter of a graph is defined as “The length of the "longest shortest path" (i.e., the longest [graph geodesic](http://mathworld.wolfram.com/GraphGeodesic.html)) between any two [graph vertices](http://mathworld.wolfram.com/GraphVertex.html) (u,v) of a [graph](http://mathworld.wolfram.com/Graph.html), where d(u,v) is a [graph distance](http://mathworld.wolfram.com/GraphDistance.html). In other words, a graph's diameter is the largest number of vertices which must be traversed in order to travel from one vertex to another when paths which backtrack, detour, or loop are excluded from consideration. ”

**Possibilities**

To calculate the diameter of a graph we have several approaches:

1. [Single-source shortest path](https://en.wikipedia.org/wiki/Shortest_path_problem#Single-source_shortest_paths) - Calculate diameter from each vertex and return the maximum value among all vertices to determine diameter of the graph. This can be done either through
   1. Topological Sorting
   2. Dijkstra’s algorithm
   3. Bellman-Ford algorithm
2. [All-pair shortest paths](https://en.wikipedia.org/wiki/Shortest_path_problem#All-pairs_shortest_paths) - The all-pairs shortest path problem finds the shortest paths between every pair of vertices in the graph. The all-pairs shortest paths problem for unweighted directed graphs was introduced by [Shimbel (1953)](https://en.wikipedia.org/wiki/Shortest_path_problem" \l "CITEREFShimbel1953), who observed that it could be solved by a linear number of matrix multiplications that takes a total time of *O*(*V*4).



**Implementation**

We implemented “Johnson - Djikstra” algorithm to calculate the diameter of the graph. We have also verified using single vertex shortest past using Topological sorting and Dijkstra’s algorithms

