Wikipedia Crawler Documentation

Overview

The code for this project consists of several independent modules that perform different tasks. The modules of the wiki package are responsible for data collection and data cleansing. The data package contains modules that analyze the collected data in various ways.

All the modules contained have been build using Java 1.7. All needed libraries are included in the /lib directory and need to be on the build path. Eclipse Mars was used to execute all of these programs. The WikiCrawler directory contained in the submission is a completed Eclipse workspace: it should be possible to simply import this project into Eclipse and run the project without additional configuration.

This README will explain what each executable does and how to execute each component. It will also explain how to set up the project. However, it is also helpful to have general overview of how these components fit together. Essentially, the modules of the data collection step are used first. The goal of these modules is to crawl Wikipedia and gather all the link data from English Wikipedia articles. After performing data collection, the goal is to have completely populated the 'links' database table, which stores every link as a source URL and a destination URL. Moreover, we must completely populate the 'canon' table, which maps Wikipedia links to article names. This table is necessary because the same Wikipedia article may be referenced by many links, and so these entries allow us to move from saving links at the level of URLs to saving links at the level of articles.

After the raw data has been obtained from Wikipedia, it is sanitized by using the canon table to translate from URLs to article names and also removing duplicates, which are introduced by this mapping. After sanitation, data is saved as a simple text file containing every link. This text file contains all the data needed by the subsequent processing steps, which compute statistics and metrics on this data file.

As a general warning, data collection in this project takes a substantial amount of time—probably about 30 hours at the absolute fastest, and may take longer depending on network conditions and processing speed.

Database Setup

This project has a dependency on MySQL Server, which must be running on localhost. MySQL can be downloaded from the link below.

https://dev.mysql.com/downloads/mysql/

Before any data can be collected, the database must be configured. Under the /src/sql package is a script that creates the database schema used for this project. You can execute this script using mysql using a command like the following on Windows: other operating systems will be similar.

mysql -u"username" -p"password" < wikipedia.sql

The script should execute silently on success. Once this runs, the database schema will exist in MySQL with a database name of “wikipedia”. Next, to be able to connect to the database, you will need to specify your MySQL username and password so that the database module can connect to the database. Under src/config/database.txt, simply change the username and password to match your own.

Data Gathering

The following modules will populate the database with link information from Wikipedia. It is recommended that you run the OrphanFinder first, followed by the crawler, followed by the DuplicateRemover. The OrphanFinder will first populate the database with articles that would be missed by the crawler, and running the DuplicateRemover only makes sense after all data has been obtained, as explained in the introduction.

OrphanFinder

This module finds all the articles that Wikipedia has marked as 'orphaned,' meaning they have no incoming links. These articles will not be found by the crawler, but Wikipedia has a special page that lists all of these articles. This module simply pulls all the articles off of Wikipedia's list and adds them to the 'topics' database table. From there, the main module can crawl these pages just like any others.

This module has a dependency on database, and so MySQL must be configured as described above. Obviously, it also requires Internet access. It takes on the order of around 5 minutes to run, depending on the quality of the internet connection being used. Because it takes relatively little time, this module does not have any checkpoint system. If there is some error during execution—like network errors—the module should just be restarted. There is no need to clear the database or do anything else if the module needs to be restarted.

Crawler

This is the main module for gathering Wikipedia data-- it simply crawls Wikipedia using BFS, recording all links in the database along the way. On startup, this module begins crawling any articles that are present in the 'topics' table but not present in the 'finished' table. This is why running the OrphanFinder first is recommended, because doing so adds those topics to the set that can be crawled.

This module has a dependency on database, and so MySQL must be configured as described above. Again, it also requires Internet access. Note that running this module will take on the order of 40 hours, and this may be notably slower depending on network conditions (which impacts the speed at which we can pull down articles) and also disk write speed (which impacts the speed at which we can write out data to MySQL).

This module writes out results approximately every 2 seconds, and it can be terminated and restarted at any time without losing data that has already been written out. Restarting will only result in crawling the last 2 seconds of articles again, and no other data will be lost. Note that transient network errors can cause the crawler to temporarily pause for a minute or two at a time, but generally it should recover from errors like this on its own.

Finally, note that this is a highly multithreaded program. It uses 8 simultaneous threads for crawling Wikipedia and a 9^{th} thread for writing out data. This gives a substantial performance boost, as each crawling thread has a substantial amount of idle time waiting for network responses. However, this means that this program has a high resource footprint, particularly in terms of disk usage.

After running this module, the “links”, “topics”, “canon”, and “finished” tables should be completely populated. The crawler prints out the number of links being written at every step to stdout so that progress can be observed.

DuplicateRemover

This module is responsible for sanitizing the data in the ‘links’ database table and producing the final, canonical version of the data that will be used for all data analysis. It reads all the links from the 'links' database table, and removes duplicates and self-directed links. It also puts all links into canonical form using the ‘canon’ table. It will ultimately output a single text file containing all the sanitized links: this text file will be the input for all data analysis steps.

The algorithm used to do duplication removal is to first split the links from the database into 300 different files. For every link, the destination of the link is hashed modulo 300, yielding the file to place the link into. This technique ensures that all duplicate links are placed into the same file (as they hash to the same value), and all files are small enough to be brought into memory. From there, each file is simply read into memory, duplicates are removed by using Java sets, and the final results are merged back into a single file.

This module takes a single argument, which is a path to a working directory where the final output and temporary files will be saved. Into this directory, a “temp” directory will be written, and the file “links_canon.txt” will be written. This module also depends on the database. It can take several hours due to the slowness of reading from the links MySQL table, but later modules will be much faster because of the increased speed of reading from the simpler text file. Checkpoints

are not used, but they should not be needed, as without network dependencies this module should not encounter transient errors.

Data Analysis

The modules below are what we used to parse our data and do analysis on it. All of the modules below depend on using the data file that was produced by the DuplicateRemover above. Some also still rely on making use of the database.

LinkCounter

This module computes various aggregate statistics about the full body of link data. It computes the total number of links found, number of articles, and distribution of inbound and outbound degrees. It also computes the top 10000 most linked articles and the top 100 articles containing the most outbound links.

This module should take on the order of a few minutes to run. It runs in linear time with respect to the total number of articles. It takes two arguments: the first argument must be the full path to the data file containing all the Wikipedia links. The second argument must be an absolute file path at which to write out the program's output. In other words, the input and output files are the two arguments.

RandomSubsetGenerator

This module generates 5 random subsets of articles from the full set of articles. Each random subset will have approximately 5000 articles. It then finds all the links that exist between the articles within these subsets. The output for a single subset consists of 5 files, one for each subset. Each output file will contain the all the links between articles of a single subset. Links are formatted as a csv file with a semicolon delimiter-- all semicolons in the links themselves are replaced with commas. This is done to make the output files compatible with Gephi.

This module takes on the order of a couple minutes to run. It depends on being able to access the MySQL database canon table. It takes two arguments-- the data file containing every link first, and an output directory for saving all the output files second. The output directory will contain 6 files named links_1.csv – links_6.csv. The first 5 correspond to random subsets, while the 6^{th} contains the subset for the top 5000 articles.

DistanceCalculator

This module performs a breadth-first search starting from any specified article. It records how many articles are X steps away, how many total articles are reachable, and which article is furthest away from the starting point. This data is simply printed to stdout, which is unlike other modules that produce an output file. This is because the output is small in this case, so it is easier

to simply print it to stdout. It takes on the order of 30-60 minutes to run, but this can vary greatly depending on the starting article. The more articles that are reachable in a low number of steps, the faster the algorithm will run. The difference can be very substantial: if there is some starting article for which most articles are 20-30 steps away, the program may take closer to 12 hours to run.

The program takes 3 arguments. The first is the data file containing all the links, and the second is the exact name of the starting article as it appears in the database. The final argument is a path to a temp directory where data can be written out. This module relies on being able to write out temporary data files containing subsets of links that are "useful", meaning they lead to articles that have not yet been visited. Doing this means that later steps of the algorithm run orders of magnitude faster than earlier steps, but it can require many gigabytes of additional disk space.