Homework # 12

due Monday, April 30, 10:00 PM

In this assignment, we will use the classes from last assignment and an HTML reader to implement a web crawler to perform breadth-first search over the web. Please read Chapter 14 in the textbook.

1 Concerning the URLObject ADT

The URLObject class from last week’s assignment is updated with the ability to list the nested URLs within the object. It is “iterable” with an iterator method that yields the nested URLs in the URL object. This part is written for you.

The ability to find URLs within the object at a particular URL implicitly defines the world-wide web as a graph that can be traversed on demand. Thus a major part of the entire internet is now accessible to a Java program.

2 Concerning Functional Objects

In this class, we have seen the Key and Comparator interfaces. Values of these types don’t have any interesting fields, but instead implement a method of interest. We call them “functional objects.” For example, a comparator provides a compare method that compares two values to indicate which comes first.

Functional objects are used to communicate over an abstraction barrier. For example, a comparator can be passed to a data structure or algorithm to determine what ordering to use for elements.

Another kind of functional object is one that is used to provide another data pathway between components. In this assignment, we will use a Consumer to give a way for the implementation of method to “return” values to the caller before it is ready to return. As with many functional interfaces, Consumer takes a type parameter indicating what is the type of values to be expected.

3 Concerning Logging

In several parts of a program, one may encounter a problem that doesn’t merit stopping the program, but which may be important nonetheless. So you will need to catch the exception and do something. Rather than just printing a message which could get lost and anyway doesn’t permit automated handling, in such situations one should log the problem. For this assignment, you will occasionally see problems being logged (don’t be alarmed by the output—it doesn’t mean your program has crashed!). You also are expected to use logging to indicate problems of interest if they occur.

We will use Java’s built-in logging Library. Use the log function on a logger:

```java
logger.log(Level.kind of warning, message(), exception-object));
```

If you are logging an exception being thrown, include it as the third (optional) parameter, and use the WARNING level.
4 Concerning Breadth-First Web Crawling

The web is very large. When we traverse part of it, we’d like to explore a small compact area. For that reason, this assignment has us create a breadth-first web-crawler, which first explores everything reachable in one “hop” from the starting URL, then two “hops,” etc. Since the web is practically infinite, we additional have a limit on the number of web-sites we will traverse. However a website may be a cul-de-sac without links to the outside web, and so a breadth-first search could terminate without reaching the limit.

In order to implement breadth-first search, you will need to use a queue to ensure that you visit URLs in the correct order, and a resource table to make sure you don’t load the same URL multiple times. For the queue, instead of implementing our own queue, we will be using the Queue interface and the LinkedList implementation. Make sure that you use these with the correct type parameters. (No “raw” types, please!)

The breadth-first web crawler checks that two URLs may be the same, but does not concern itself with whether the contents of the resources are the same.

The crawler is started at an initial URL, and then has a method to examine more URLs from where it last left off. The client passes in a “consumer” that will be informed as each URL object is successfully read. (It will not be called with a URL that is malformed or which cannot be loaded.) The client of the crawler can decide what it wants to do with the URL Object. In this assignment, the client will be checking for duplicate contents, but this doesn’t concern the web crawler.

According to web standards, a URL is a particular kind of URI. Normal URIs have a number of parts, but we want to ignore the “fragment” which indicates a point within the resource. Obviously two URLs which are the same except with different fragments do not represent unique resources! Thus we will clip off any fragment before loading the resource. The code to do this operation is simple, but provided for you.

5 Finding Duplicate Resources

The top-level program is one that uses the web crawler to find duplicate resources: resources with the same CRC-32 hash. You should find URL objects that have the same hash and group them together and then print out all groups with size greater than one.

You should use library classes to find duplicates. It is up to you to find a good design and to use resources well. We also want you to use the built-in List#sort method using a lambda comparator that puts larger groups first. Our solution uses Map and List as types for variables and creates HashMap and ArrayList instances.

5.1 Sample Output

Running our solution on Prof. Boyland’s web page

http://www.cs.uwm.edu/faculty/boyland/index.html

the following output is generated:

8 (potential) duplicates found, for a waste of 8.0%
Duplicate resources at:
  http://www.ucdavis.edu
  http://www.cs.berkeley.edu
  http://www.nju.edu.cn/
Duplicate resources at:
Duplicate resources at:
http://www.cs.uwm.edu/faculty/boyland/
Duplicate resources at:
http://uwm.edu/engineering/our-people/community/girls-who-code-club/

The first few duplicate resources are apparently because the given websites all reply with 302: “Moved temporarily” which is supposed to cause the agent to try a new URL. Our URL reader however simply returns an empty stream (no contents) in each case, which obviously has the same content hash.

The next group shows that the URL shows up twice: once with an extra (probably erroneous) ampersand.

The third group shows that our system doesn’t distinguish the implicit extension of a directory URL with the default contents. (Some implementations of Java may realize these are the same URL, in which case, this group would not print and there would be only seven duplicates found.)

The last group shows possible actual duplication: the “Girls Who Code” club page might be duplicated in two places. (Unless the query in the second URL actually loads the contents from the first URL.)

6 What you need to do

You need to finish the implementation of the BreadthFirstWebCrawler and DuplicateResourceFinder classes.

7 Files

Your Homework #12 repository includes the following files:

resources/ Some sample web pages for testing.
src/Main.java Fa çade for the duplicate resource finder.
src/TestBFWC.java Test suite for the web crawler.
src/edu/uwm/cs351/BreadthFirstWebCrawler.java The skeleton file of the web crawler.
src/edu/uwm/cs351/DuplicateResourceFinder.java The skeleton file of the duplicate resource finder.
src/edu/uwm/cs351/URLObject Updated class for URL objects.