CS5800 Final Programming Project

This assignment must be submitted by 09/12/2016.

Feedback will be provided by 15/1/2016.

This assignment contributes up to 105% to your coursework mark. Your coursework represents 31.5% of your final mark.

Learning outcomes assessed

This assignment covers the material studied in the weeks 1–7 of the course. The emphasis is on functions, Java container classes, input/output, working with strings, and text processing. The notes are available on the course Moodle page. The learning outcomes being assessed are

- Functions
- Java container classes
- Input/output
- String datatype
- Text processing
- Using arrays to represent and manipulate graphs

Instructions

Submission will be via Turnitin using the link provided on the course Moodle page. Your submission must consist of the following **four** parts:

- **Part 1**: `UrlExtractor.java` (Q1 and Q2)
- **Part 2**: `ArticleIndexer.java` (Q3)
- **Part 3**: `PageRank.java` (Q4)
- **Part 3**: `README` (Q4.7): must be a **text** file.

The files you submit cannot be overwritten by anyone else, and they cannot be read by any other student. You can, however, overwrite any of the four parts of your submission as often as you like, by resubmitting, though only the last version submitted will be kept. Submission after the deadline will be accepted but it will automatically be recorded as being late and is subject to College Regulations on late submissions. Please note that all your submissions will be graded anonymously.

Marking criteria

The maximum percentage of the assignment mark awarded for each question is given in brackets next to each question. Whenever a question consists of multiple sub-questions, the percentage breakdown is indicated next to the corresponding sub-question. You can score up to 105% for this assignment.

NOTE: All the work you submit should be solely your own work. Turnitin (http://turnitin.com/) will be used to **automatically** check your submissions for originality against both: other students' submissions and a wide variety of online sources. It will alert the course staff of all the submissions attaining higher than usual similarity scores. All cases of suspected plagiarism will be passed on for further consideration by the departmental disciplinary committee. The students involved in the confirmed plagiarism cases will receive the grade of **0%** for this assignment, and might be subjected to other disciplinary measures.

Preliminaries

In this assignment, you will implement various building blocks and programs to prepare raw data for processing by the PageRank algorithms (Questions 1–3), which constitutes the core of the Google web search technology. You will then apply PageRank (Question 4) to determine page rank scores for a collection of Guardian articles dealing with the Ebola epidemics in Africa.

It is recommended that you familiarize yourself with basics of PageRank in Chapter 1.6 of "Introduction to Programming in Java: An Interdisciplinary Approach" by Sedgewick and Wayne (the course's main textbook) prior to attempting Question 4 of the assignment. A scanned copy of the chapter is available for download from the course Moodle page.

The Resources section below includes pointers to datasets and code libraries used in the assignment's questions.

Resources

All items below are available from the course Moodle page.

- `three-page.txt`: a file consisting of three HTML pages on three separate lines. It can be used to test and debug programs and methods in Questions 1 and 2.
- `article2content.txt`: a file consisting of 10 Guardian articles along with their HTML content to test and debug programs in Questions 3 and 4.
- `scrape.txt`: a file consisting a full set of articles along with their HTML content to use as input to PageRank algorithm in Question 4.
- `MyURL.java`: Java source of the `parseURL(url)` method implementation. This method checks whether the URL provided as its argument complies with the standard syntax, and if so, converts it to the form `scheme://server/path` (where `scheme` is either `http` or `https`, `server` is the name of a web server, such as, `www.google.com`, and `path` is the path to the page on the server), and returns the result; and if not, returns `null`. This method must be used in all programs as described in the specific questions.

Questions

Answer the following questions:

Question 1 (15% in total)

You are given a dataset consisting of web pages. The dataset is organised so that each line holds the *entire* content of a *single* web page in HTML format. The hyperlinks within each page are embedded using a pair of `<a href="URL">` and `</a>` tags. Note that the HTML tag names are **not** case-sensitive. That is, arbitrary combinations of upper and lower case letters are permitted in the tag names (e.g., both `<a HREf...>` and `<A href...>` are valid tags).

1. Create a public class `UrlExtractor` and store it in the `UrlExtractor.java` file.

2. Add method

   ```
   public static void scanAndPrint(Scanner in) {}
   ```

   to `UrlExtractor` that takes a `Scanner` object `in` associated with the given dataset (via either file name or standard input) as argument, scans sequentially line-by-line through the dataset, and prints the *URL strings* appearing between the quotes for each start tag `<a href>` being encountered. (In particular, if the same URL appears in multiple `<a href>` tags, it will be output multiple times – once for each `<a href>` in which it was found.)

   Prior to printing, each extracted URL must be passed to the provided `MyURL.parseURL` method (see the Resources section above). Your implementation must print the return value of `MyURL.parseURL` if it is not `null`, and skip it otherwise. (7%)

3. Your implementation must be able to correctly identify hyperlink tags **regardless** of the case (upper or lower) used for individual letters in the tag name. (3%)

4. Add a `main` method to `UrlExtractor` to test your implementation of `scanAndPrint` on two test cases: (1) the dataset is provided via standard input, and (2) the dataset is provided in a file. (5%)

For example, if the four hyperlinks below are encountered while scanning sequentially through the collection (and no other hyperlinks are found)

```
<a href="http://www.who.int/mediacentre/factsheets/fs180/en">
<a hrEf="www.infowars.com/swine-flu-hysteria-spreads-faster-than-actual-virus"
<A HRef="http://www.who.int/mediacentre/factsheets/fs180/en">
<a HREF="mailto:globaldevpros@theguardian.com">
```

then the output must be as follows:

```
http://www.who.int/mediacentre/factsheets/fs180/en
http://www.infowars.com/swine-flu-hysteria-spreads-faster-than-actual-virus
http://www.who.int/mediacentre/factsheets/fs180/en
mailto:globaldevpros@theguardian.com
```

Question 2 (20% in total)

1. Add method

   ```
   private static boolean matchServer(String url, String str)
   ```

   to `UrlExtractor` that takes two String parameters: `url` in the form `scheme://server/path`, and `str`. The method returns `true` if

   (a) both `url` and `str` are not `null`, **and**

   (b) the `server` component of `url` contains `str` as a substring. The comparison must be **not** case-sensitive.

   Otherwise, the method returns `false`.

   For example:

   ```
   matchServer("http://www.google.com", "google.com")=>true
   matchServer("http://www.theguardian.com", "GUARDIAN")=>true
   matchServer("http://www.Guardian.co.uk/global-development",
                                              "guardian")=>true
   matchServer(null, "guardian.co.uk") => false
   matchServer("http://theguardian.com", "theguardian.com")=>true
   ```

   (10%)

2. Copy the code in `scanAndPrint` from Question 1 into new method

   ```
   public static void scanAndPrint(Scanner in, String str)
   ```

   and add it to the `UrlExtractor` class. Use `matchServer` to modify `scanAndPrint(Scanner in, String str)` so that the URL returned by `MyURL.parseURL` is output only if its server component contains `str` as a substring. (5%)

3. Add code to the `main` method of `UrlExtractor` to test your implementations of `matchServer` on the 5 examples above, and **2** examples of your own. (3%)

4. Add code to the `main` method of `UrlExtractor` to test your implementations of the two-parameter version of `scanAndPrint` on two test cases: (1) the dataset is provided via standard input, and (2) the dataset is provided in a file. (2%)

Question 3 (20%)

You are given a dataset consisting of articles published in Guardian. The dataset is organised so that each line consists of two attributes: (1) the article's URL in the form `scheme://server/path`, and (2) the article's text formatted as an HTML web page. The two fields are separated by a **single** white space. The articles include links to other web pages embedded into the `<a href>` tags as explained in Question 1.

Use the code and/or methods developed in Questions 1 and 2, to write program `ArticleIndexer`, which assigns each **unique** URL whose server component has the string "guardian" as a subsequence (in a non-case-sensitive fashion) a unique sequence number starting from 0. The program must index the URLs of the articles themselves (i.e., those appearing in the first attribute of each line) as well as those embedded in their texts. The program must print to **standard output** a list of URL and sequence number pairs separated by a **single** white space with each pair appearing on a **separate** line as in the following example:

```
http://www.guardian.co.uk/global-development 0
http://www.theguardian.com/global-development/girl-summit 1
...
```

Your program must use `HashMap` to ensure duplicate appearances of the same URL are not assigned different sequence numbers.

Question 4 (50% in total)

Write program `PageRank` that takes the URL index of the Guardian dataset produced in Question 3 from standard input and proceeds as follows:

1. Read in the index and store it in memory so that both the sequence number for a given URL, and the URL for a given sequence number can be efficiently retrieved. (*Hint*: Use two HashMap objects.) (5%)

2. Create two integer arrays: a two-dimensional array to hold an adjacency matrix M for the URL graph, and a one-dimensional array to hold the out-degree of each node in the URL graph. Traverse the dataset ignoring non-Guardian URLs, and use the in-memory index to populate both arrays. Each element $M_{i,j}$ of the adjacency matrix must be equal to the number of times URL j is referenced from URL i. That is, the URL graph is a *directed multi-graph.* (10%)

3. Create a two-dimensional array of doubles to hold the transition probability matrix A manipulated by the Page Rank algorithm. Use the two arrays constructed above to populate A so that each $A_{i,j} = 0.1p + 0.9q$ where p is the probability to select an arbitrary Guardian article in the collection uniformly at random, and q is the probability to select URL j when browsing URL i. (10%)

4. Create a one-dimensional array of doubles to hold a row vector R of page ranks for the articles in the collection. Initialise R with 1.0 in one of the entries (e.g., the first one), and 0 in all other entries. In a loop, multiply R by A storing the results of each multiplication in R. Continue until the entries R_i^k and R_i^{k+1} obtained at two consecutive iterations k and $k+1$ are equal with **relative** error of less than 0.1%. (10%)

5. Sort the resulting array of page ranks using one of the methods in the Java `Arrays` library. Output to standard output the 10% most popular and 10% least popular articles according to their page ranks giving URL and page rank of each article as follows:

```
Most popular 10%:
URL1 page-rank-of-URL1
URL2 page-rank-of-URL2
...
Least popular 10%:
URL1 page-rank-of-URL1
```

```
URL2 page-rank-of-URL2
...
```
(5%)

6. Your program must be reasonably modular. That is, use functions (i.e., public or private static methods) and/or classes to separate tasks within a program whenever it makes sense rather than writing a monolithic code. (5%)

7. **Briefly** discuss scalability of our PageRank implementation. Can it be used to compute the ranks of all pages in the world-wide web? If not, how can it be made more scalable? Note that your answer to this question must be submitted in a **text** file, called `README`. The answers submitted in any other file types (such as MS Word or PDF) will **not** be assessed. (5%)