COSC2320: Data Structures and Algorithms
HW7: Graphs and the World Wide Web

1 Introduction

In this homework you will create a C++ program to calculate page ranks in a small model of the world wide web, using graphs. A graph is defined as $G = \{V, E\}$ where V is a set of vertices or nodes and E is a set of edges o links between the nodes.

In essence a webpage is a node in huge graph. All hyperlinks in the page are the edges in the graph, connecting it to other pages (other nodes). Evidently these edges have a direction, since ”clicking” on a link takes you away from the page you were and unless there is a link back (assuming there is no ”back” button in your browser) you couldn’t return to where you were originally. This is called a ”directed graph”.

The degree of a vertex is the number of edges that are incident on the vertex. In a directed graph we can divide this number into two parts: the *indegree* is the number of edges that are incident on the vertex and the *outdegree* is the number of edges that is ”pointing outward” from the vertex.

In this homework you will calculate the degree of each page of the test case we will provide. You should distinguish the indegree and outdegree. The files will consist of random keywords and links to other files. Those links will be clearly marked as in HTML language:

```
<a href = "//test.html"> link </a>
```

we will strictly adhere to this convention. Furthermore since the files will have the “.html” extension you will be able to visualize them in your browsers. The output will consist of three parts: A list of the top five referenced pages (pages that have the highest indegree); a list of all broken links (links to files that don’t exist) and finally a list of all sinks or black holes (files with indegree ≥ 0 but outdegree $= 0$.

2 Examples

INPUTS

```
Kuratowski1.html
-----------------------------------------
<a href = "Kuratowski2.html"> Kuratowski2 </a>
<a href = "Kuratowski3.html"> Kuratowski3 </a>
<a href = "Kuratowski4.html"> Kuratowski4 </a>
<a href = "Kuratowski5.html"> Kuratowski5 </a>
<EOF>

Kuratowski2.html
-----------------------------------------
<a href = "Kuratowski1.html"> Kuratowski1 </a>
```

```
<a href = "Kuratowski3.html"> Kuratowski3 </a>
<a href = "Kuratowski4.html"> Kuratowski4 </a>
<a href = "Kuratowski5.html"> Kuratowski5 </a>
<EOF>

Kuratowski3.html
----------------------------------------------
<a href = "Kuratowski1.html"> Kuratowski1 </a>
<a href = "Kuratowski2.html"> Kuratowski2 </a>
<a href = "Kuratowski4.html"> Kuratowski4 </a>
<a href = "Kuratowski5.html"> Kuratowski5 </a>
<EOF>

Kuratowski4.html
----------------------------------------------
<a href = "Kuratowski1.html"> Kuratowski1 </a>
<a href = "Kuratowski2.html"> Kuratowski2 </a>
<a href = "Kuratowski3.html"> Kuratowski3 </a>
<a href = "Kuratowski5.html"> Kuratowski5 </a>
<EOF>

Kuratowski5.html
----------------------------------------------
<a href = "Kuratowski1.html"> Kuratowski1 </a>
<a href = "Kuratowski2.html"> Kuratowski2 </a>
<a href = "Kuratowski3.html"> Kuratowski3 </a>
<a href = "Kuratowski4.html"> Kuratowski4 </a>
<EOF>

filenames1.txt
----------------------------------------------
Kuratowski1.html
Kuratowski2.html
Kuratowski3.html
Kuratowski4.html
Kuratowski5.html

page1.html
----------------------------------------------
<a href = "page2.html"> page2 </a>
<a href = "page3.html"> page3 </a>
<a href = "page4.html"> page4 </a>
<a href = "page5.html"> page5 </a>
<a href = "page6.html"> page6 </a>
<EOF>
```

```
page2.html
----------------------------------------------
<a href = "page1.html"> page1 </a>
<EOF>

page3.html
----------------------------------------------
<a href = "page1.html"> page1 </a>
<a href = "page2.html"> page2 </a>
<EOF>

page4.html
----------------------------------------------
<a href = "page1.html"> page1 </a>
<a href = "page2.html"> page2 </a>
<a href = "page3.html"> page3 </a>
<EOF>

page5.html
----------------------------------------------
<EOF>

filenames2.txt
----------------------------------------------
page1.html
page2.html
page3.html
page4.html
page5.html
<EOF>

tree1.html
----------------------------------------------
<a href = "tree2.html"> tree2 </a>
<a href = "tree3.html"> tree3 </a>
<EOF>

tree2.html
----------------------------------------------
<a href = "tree4.html"> tree4 </a>
<a href = "tree5.html"> tree5 </a>
<EOF>

tree3.html
----------------------------------------------
<a href = "tree6.html"> tree4 </a>
<EOF>
```

```
tree4.html
----------------------------------------------
<EOF>

tree5.html
----------------------------------------------
<EOF>

tree6.html
----------------------------------------------
<EOF>

filenames3.txt
----------------------------------------------
tree1.html
tree2.html
tree3.html
tree4.html
tree5.html
tree6.html
<EOF>

script.txt
----------------------------------------------
load(W1,‘filenames1.txt’)
load(W2,‘filenames2.txt’)
load(W3,‘filenames3.txt’)
rank(W1,‘out1.txt’)
rank(W2,‘out2.txt’)
rank(W3,‘out3.txt’)
<EOF>
```

RESULTS

```
out1.txt
----------------------------------------------
Kuratowski1.html 4
Kuratowski2.html 4
Kuratowski3.html 4
Kuratowski4.html 4
Kuratowski5.html 4

No Broken Links

No Sinks
<EOF>
```

```
out2.txt
---------------------------------------------
page1.html 3
page2.html 3
page3.html 2
page4.html 1
page5.html 1

page6.html Broken Link

page5.html Sink
<EOF>

out3.txt
---------------------------------------------
tree2.html 1
tree3.html 1
tree4.html 1
tree5.html 1
tree6.html 1

No Broken Links

tree4.html Sink
tree5.html Sink
tree6.html Sink
<EOF>
```

3 Program and output specification

The main program should be called ”pagerank”. Syntax:

```
pagerank script=script.txt
```

where `script.txt` is a script file similar to the ones used in previous homeworks. There will be two new instructions: `load(W,‘list.txt’)` that loads a set of files into memory. This list of files is identified by `W`. The second instruction is `rank(W, ‘out.txt’)` that calculates the quantities required by the homework and outputs the results to the file `out.txt`.

4 Requirements

- Your program should count the number of edges that are incident on each node and classify them as indegree and outdegree.
- Your program should output a list of the top 5 most referenced pages, a list of all broken links and a list of all sinks.
- Your output files should look **exactly** like those shown in the examples. We will make no allowances for differences in format. For the top five list, you should output one filename per line, followed by a space and the indegree. If two or more pages have the same indegree, you should list them in alphabetical order. After the top five list you should leave a blank line and list all broken links in alphabetical order. There should be one filename per line, followed by a space and the keywords “Broken Link”. Finally you should leave one more space and the list of all sinks, also in alphabetical order, with the keyword “Sink”.
- If there are no broken links you should output “No Broken Links” to the file. Similarly if there are no sinks you should output “No Sinks”.
- Even if it was not explicitly shown in the examples, the files contents will not be limited to just links. There will be other text and html code, as in a real webpage.
- The test cases can contain hundreds of webpages with many links connecting each other, therefore you should design your data structures accordingly.
- Links will be marked in HTML language: `<a href = "file.html"> link </a>`. You don’t need to keep track of any malformed links.
- The program should not halt when encountering errors. It should just send a message to the log file and continue with the next line. The only error that is unrecoverable is a missing input file or a missing output file.

  Extra credit

- For extra credit you can calculate any (or all) of the following:
- **(30 points)** Compute G^*, the transitive closure. Show the output in binary matrix form.
- **(20 points)** Shortest paths between given vertices using Dijkstra’s algorithm.
- **(10 points)** Topologically sort the graph.

- Extra credit work should be demonstrated in person to the T.A.s by appointment. Appointment times will be announced the day the homework is due.
- Do not use the STL library.
- Your program should write error messages to a log file (and optionally to the screen). Your program should not crash, halt unexpectedly or produce unhandled exceptions. Consider empty input, zeroes and inconsistent information. Each exception will be -10.
- Test cases: Your program will be tested with 10 test scripts, going from easy to difficult. You can assume 80% of test cases will be clean, valid input files. If your program fails an easy test script 10-20 points will be deducted. A medium difficulty test case is 10 points off. Difficult cases with specific input issues or complex algorithmic aspects are worth 5 points.
- A program not submitted by the deadline is zero points. A non-working program is worth 10 points. A program that does some computations correctly, but fails several test cases (especially the easy ones) is worth 50 points. Only programs that work correctly with most input files that produce correct results will get a score of 80 or higher. In general, correctness is more important than speed.