420-LCW-05 Programming Techniques and Applications - Lab 7

April 6, 2022

This lab assignment is just an exercise - it will graded only pass/fail, but please do submit your modified file(s) at the end of the lab session. I do NOT expect you to finish all of the exercises, but do think about them and see if you can imagine how a solution might work. I guarantee that some elements of these labs will appear on future examinations!

1 Introduction

In this lab you will work with directed graphs.

Recall that a *graph* is a structure that consists of a set of *vertices* connected by *edges*. In this course I generally represent a graph by associating a list of adjacent vertices with each individual vertex.

A *tree* is a specific kind of graph. A tree is fully connected, and contains no cycles. This implies that a tree has no "extra" edges, and that there is only one path between any two vertices v and w. Trees therefore always have exactly one fewer edges than they have vertices.

Remember that when we do breadth-first search on a graph, we visit each vertex exactly once, starting at a particular vertex. One way to think of this is that the BFS algorithm finds a "shortest path tree" (SPT) leading from the starting vertex to all other reachable vertices using subset of the edges of the original graph. The shortest-path tree will be rooted at the start vertex.

Files provided in this lab:

- `graph.py` - The implementation of the directed and undirected graphs.
- `bfs.py` - The implementation of the BFS algorithm.
- `tinyG.txt`, `tinyDG.txt`, and `mediumDG.txt` - Test files for the provided graph code.
- `j5.pdf` - the original problem description.
- `J5/*` - these are the test files for Exercise 1. You will need to use these files to check your solution.

Exercise 1

This was the final problem in the "Junior" section of the Canadian Computing Competition for 2018. In this lab, you will use what we've learned in class to solve it.

Read the separate PDF file that defines the problem (`j5.pdf`) for all of the details. The problem is best solved by applying breadth-first search, so I have provided you with an implementation of BFS in the file `bfs.py` as well as the basic `digraph` class in `graph.py`.

Your tasks are as follows:

- Create a file `lab7ex1.py`
- Write the code to read the input files as specified in the problem, creating a `digraph` object. You can base this on the `fromfile()` method in the `graph.py` file, but do not modify the provided files.
  - Prompt the user to enter a file name.
  - Use that file name to open the associated file.
  - Read the first line to get the number of vertices (pages) in the graph (book).
  - Read the rest of the lines to construct the edges of the graph.
  - Note that the numbering used in the files assumes that page numbering starts at one, whereas the graph model assumes that vertices are numbered starting at zero. You can easily correct for this by subtracting one from every page number in the file, or by creating a "fake" vertex 0 that is not used in the rest of your calculations.
- Search the resulting digraph using BFS.
- Calculate and print the two things the problem requests:
  - What is the shortest path from the first page to any one of the ending pages?

- Are all of the pages reachable from the first page?

You may use the provided code for your work, but feel free to create your own implementations if you prefer.

The `J5` subdirectory contains all of the testing files used for this problem in the CCC. The input files all end with the extension `.in`, and the corresponding output file ends with `.out`. Your program should produce the correct output for each of the 27 input files. You do not *have* to test with all 27 files, but you want to be sure your code works with a good sampling of them.

Exercise 2

Create a copy of your work for the first part, and call it `lab7ex2.py`.

In addition to finding the information above, compute the following additional information:

1. Print the first page with the highest *outdegree*, and the outdegree of that page.
2. Print the first page with the highest *indegree*, and the indegree of that page.
3. Find the length of the *longest* among the shortest paths from the start to end page, and print the number of pages read in that case.

For example, for the file `J5/j5.1-2.in`, your code would probably print:

```
Y
4
8 5
15 6
4
```

Meaning all pages are reachable, the shortest path through the book is 4 pages long, page 8 has the maximum outdegree of 5, and page 15 has the maximum indegree of 6. Finally, there are 4 pages in the longest of the shortest paths.

For the file `J5/j5.4-5.in`, your code should print:

```
N
3
46 5
1785 6
13
```

Not all pages are reachable, the shortest path through the book is 3 pages long, page 46 has an outdegree of 5, and page 1785 has an indegree of 6. Finally, the longest of the shortest paths through the book has length 13.

Exercise 3

This one is more-or-less optional. The text files `tinyDG.txt` and `mediumDG.txt` contian directed graphs in a format readable by the `fromfile()` method. Create a file named `lab7ex3.py` and implement the cycle counting method described in the lecture slides. Determine how many directed cycles there are in each of these graphs.