420-LCW-05 Programming Techniques and Applications - Assignment 3

March 12, 2019

Here's a reminder of the general requirements for all of the course assignments:

1. **Identification section** Do this for *every* Python file in every assignment in this course. This section must be either in a comment, with a '#' preceding each line, or enclosed within triple quotes (''). The grader and I need this section for the *accurate processing of your assignment*. Assignments missing this may lose up to 5% of the total mark.

   Example:

   ```
   """
   Justin Trudeau, 1234567
   Sunday, February 31
   R. Vincent, instructor
   Assignment 2
   """
   ```

   Obviously substitute your name, Marianopolis ID, and the correct date for the appropriate fields!

2. Always include additional comments with your code. These need not explain every individual line of your program, but consider using comments for the following situations:

   - A brief explanation of a particular variable's purpose, included on the first line where the variable is defined, e.g.:

     ```
     hi = 100 # Define the upper limit of the range.
     ```

   - A note mentioning any website or person you may have consulted with to help with the assignment.
   - A comment describing any constant value that appears in your code.

   In addition, each `def` statement, whether for a global function or class method should include at least a brief docstring. *You should also provide docstrings for any new classes you create*.

3. Your submission for assignment will typically include multiple Python files, which have the extension `.py`. Before submission, these files must be combined into a single ZIP archive file (extension `.zip`). If you do not know how to create a ZIP file, I will demonstrate this in lab.

4. Be sure to respect other instructions specified in the assignment. Part of each assignment is to correctly follow the instructions as closely as possible.

1 Introduction

In this assignment you will work with directed graphs.

Recall that a *graph* is a structure that consists of a set of *vertices* connected by *edges*. We 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. It is also *rooted* at a single vertex. This implies that a tree has no "extra" edges, and that there is only one path between any two vertices v and w. Trees may be directed or undirected, depending on the application. If a tree is directed, it may be organized either with all edges leading *away* from the root, in which case it is an *out-tree*, or with all edges leading *to* the root, in which case it is an *in-tree*.

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) over the vertices and a subset of the edges of the original graph. The shortest-path tree will be rooted at the start vertex.

Files provided in this assignment:

- `graph.py` - The implementation of the directed and undirected graphs.
- `bfs.py` - The implementation of the BFS algorithm, as a class.
- `tinyG.txt`, `tinyDG.txt`, and `mediumDG.txt` - Test files for the provided graph code. You can ignore these.
- `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 assignment, 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 `a3ex1.py`
- Write the code to read the input files as specified in the problem, creating a `digraph` object. You should prompt the user to enter a file name.
- 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?

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 `a3ex2.py`.

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

1. Print the number of paths of length equal to the shortest path.
2. Find the length of the *longest* path from the start to end page, and print the number of pages read in that case.

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

```
N
3
2
13
```

Exercise 3 (Optional)

Create a copy of your work for the first part, and call it `a3ex3.py`.

In addition to the information required for exercises 1 and 2, figure out the number of *connected components* in the graph implied in the problems.

You may find that the Python `set` type will be useful for implementing this.

What to hand in

A zip file containing:

- Your `a4ex1.py` file.
- Your `a4ex2.py` file.