Graph Modeling, Depth-First Search, and Topological Sorting

Homework #12
By
John Robertson
CS 303 Algorithms and Data Structures
November 16, 2019

0. **Important Note**

Because my driver was designed to work with the .txt files provided on the assignment page, it is now "broken" for those (irrelevant) inputs because the driver originally assumed the first two lines would be removed. Just don't run it on the original files, and only on the new files.

1. **Problem Specification**

This problem requires a graph to be sorted by the amount of parents and children it has using depth-first search.

2. **Program Design**

This program has two files: a file with a driver and a file with the tree class, extended from last week for the purposes of this week's assignment.

The following steps were required to develop this program:

- Make minor edits to change
- Implement dfs() and topological_sort()
- Debug the above and make changes to the internal driver to accommodate them
- Make changes to the driver to accommodate the new files and methods

As an additional note, I changed print_path so that it does nothing if a path doesn't exist to avoid making the output obnoxious.

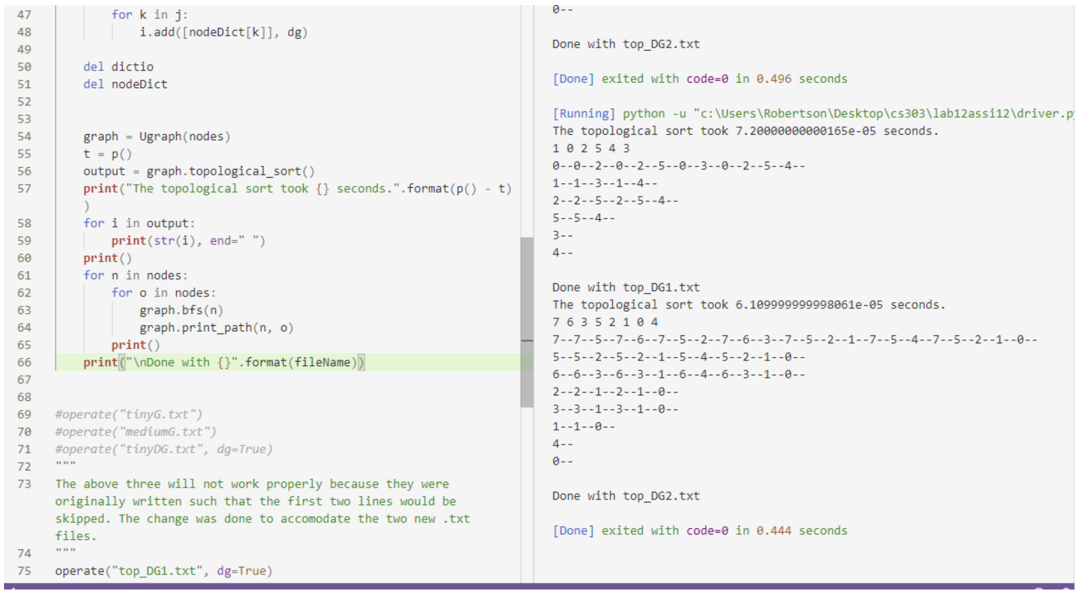
3. **Testing Plan**

In the mini-driver for the file, I modeled a graph that I knew the correct sort for to make sure that my dfs() and topological_sort() worked correctly; it did.

4. **Test Cases**

The driver tests on two files, one with six unique nodes and one with eight. The first file is the file that I hand-introduced; as the output for the first file and my hand-made mini-driver are the same, this confirms the driver to be correct. The second file only builds on the first.

5. **Analysis and Conclusions**



```python
        for k in j:
            i.add([nodeDict[k]], dg)

    del dictio
    del nodeDict


    graph = Ugraph(nodes)
    t = p()
    output = graph.topological_sort()
    print("The topological sort took {} seconds.".format(p() - t)
)
    for i in output:
        print(str(i), end=" ")
    print()
    for n in nodes:
        for o in nodes:
            graph.bfs(n)
            graph.print_path(n, o)
        print()
    print("\nDone with {}".format(fileName))


#operate("tinyG.txt")
#operate("mediumG.txt")
#operate("tinyDG.txt", dg=True)
"""
The above three will not work properly because they were
originally written such that the first two lines would be
skipped. The change was done to accomodate the two new .txt
files.
"""
operate("top_DG1.txt", dg=True)
```

```
0--

Done with top_DG2.txt

[Done] exited with code=0 in 0.496 seconds

[Running] python -u "c:\Users\Robertson\Desktop\cs303\lab12assi12\driver.p
The topological sort took 7.20000000000165e-05 seconds.
1 0 2 5 4 3
0--0--2--0--2--5--0--3--0--2--5--4--
1--1--3--1--4--
2--2--5--2--5--4--
5--5--4--
3--
4--

Done with top_DG1.txt
The topological sort took 6.109999999998061e-05 seconds.
7 6 3 5 2 1 0 4
7--7--5--7--6--7--5--2--7--6--3--7--5--2--1--7--5--4--7--5--2--1--0--
5--5--2--5--2--1--5--4--5--2--1--0--
6--6--3--6--3--1--6--4--6--3--1--0--
2--2--1--2--1--0--
3--3--1--3--1--0--
1--1--0--
4--
0--

Done with top_DG2.txt

[Done] exited with code=0 in 0.444 seconds
```

Why 4 is after 0 is obvious: they both have no children, so they both are potential ends. Why 5 is after 3 is more puzzling to me, given that 5 should be "earlier" in the graph. Nevertheless, it works for top_DG1.txt.

6. **References**

Logan Creel's reference notes