Implementing a Depth First Search and Topological Sort Algorithm
Report #12

By
Eduardo Castro

CS 303 Algorithms and Data Structures

November 19, 2014

1. Problem Specification

The goal of this assignment is to implement a direct and an undirected graph and to implement a depth first search algorithm on these graphs. We also need to apply and implement a Topological Sort into this graph.

2. Program Design

The first part of the assignment was basically to extend a provided Graph class and implement a DepthFirstPath class, reads an input file to populate these graphs and print the path from the vertex 0 to all the others.

The following steps were required to develop this program:

a) Create the Graph class

The following methods were defined within the Graph:

a) Graph()
    a. Empty Constructor
b) Graph(BufferedReader reader)
    a. Build a graph with an input file
c) addEdge(int v, int w)
    a. adds an edge from element v to w. If it is a directed graph it points just from v to w and if it undirected it points from both sides
d) toString()
    a. prints the graph

Then we had to implement a BFS algorithm for an undirected graph. The following methods were defined for the BFS class:

a) DepthFirstPaths(UndirectedGraph G, int s)
    a. Starts the DFS and calls the dfs function
b) dfs(UndirectedGraph G, int s)
    a. Do the search based on source s
c) printPath(UndirectedGraph, int s, int v)
    a. print the path from element s to element v
d) hasPathTo(int v)
    a. returns true if there if a path to v
e) pathTo(int v)
    a. returns the path to v
f) topologicalSort(Graph G)
    a. Call the DFS(G) to compute finishing times v.f for each vertex v and then add each vertex to a LinkedList. The return it is this LinkedList after computing DFS for all the vertexes.
g) printPath(UndirectedGraph G)
    a. print the path from the first element to the others

The Scanner class provided by Java was used to read in the necessary values within the provided driver program. The println method of the System.out object was used to display the inputs and results for the provided driver program.

3. Testing Plan

For the first part of our test plan we need to read an input file as an undirected graph and runs the DFS algorithm to find paths to all the other vertices considering 0 as the source.
For the second part we need to also read an input file as directed graph and run the Topological Sort algorithm on it.

4. Test Cases

For the first part we read the mediumG.txt file as an undirected graph and prints the paths. As the output is very very very big I will print only some random selected elements.

```
0 to 72:  72, 150, 164, 194, 130, 234, 159, 136, 128, 173, 90, 113,
158, 121, 17, 134, 119, 81, 41, 88, 98, 178, 236, 166, 133, 129, 13,
19, 174, 103, 100, 84, 79, 70, 30, 43, 82, 85, 152, 143, 175, 246,
244, 179, 193, 243, 192, 162, 147, 140, 117, 167, 224, 218, 146, 145,
120, 161, 186, 177, 169, 190, 247, 220, 189, 200, 203, 249, 229, 170,
182, 242, 223, 198, 94, 141, 135, 108, 110, 122, 139, 156, 221, 219,
214, 212, 210, 207, 205, 196, 181, 184, 197, 230, 188, 138, 151, 226,
208, 187, 168, 144, 97, 80, 59, 50, 48, 45, 76, 241, 228, 153, 115,
95, 216, 201, 217, 232, 248, 231, 191, 176, 114, 209, 211, 225, 222,
204, 202, 163, 149, 66, 39, 24, 15, 0

0 to 73:  73, 56, 53, 134, 119, 81, 41, 88, 98, 178, 236, 166, 133,
129, 13, 19, 174, 103, 100, 84, 79, 70, 30, 43, 82, 85, 152, 143, 175,
246, 244, 179, 193, 243, 192, 162, 147, 140, 117, 167, 224, 218, 146,
145, 120, 161, 186, 177, 169, 190, 247, 220, 189, 200, 203, 249, 229,
170, 182, 242, 223, 198, 94, 141, 135, 108, 110, 122, 139, 156, 221,
219, 214, 212, 210, 207, 205, 196, 181, 184, 197, 230, 188, 138, 151,
226, 208, 187, 168, 144, 97, 80, 59, 50, 48, 45, 76, 241, 228, 153,
115, 95, 216, 201, 217, 232, 248, 231, 191, 176, 114, 209, 211, 225,
222, 204, 202, 163, 149, 66, 39, 24, 15, 0

0 to 92:  92, 132, 171, 154, 195, 245, 238, 235, 213, 180, 172, 125,
157, 148, 71, 233, 240, 239, 112, 234, 159, 136, 128, 173, 90, 113,
158, 121, 17, 134, 119, 81, 41, 88, 98, 178, 236, 166, 133, 129, 13,
19, 174, 103, 100, 84, 79, 70, 30, 43, 82, 85, 152, 143, 175, 246,
244, 179, 193, 243, 192, 162, 147, 140, 117, 167, 224, 218, 146, 145,
120, 161, 186, 177, 169, 190, 247, 220, 189, 200, 203, 249, 229, 170,
```

182, 242, 223, 198, 94, 141, 135, 108, 110, 122, 139, 156, 221, 219,
214, 212, 210, 207, 205, 196, 181, 184, 197, 230, 188, 138, 151, 226,
208, 187, 168, 144, 97, 80, 59, 50, 48, 45, 76, 241, 228, 153, 115,
95, 216, 201, 217, 232, 248, 231, 191, 176, 114, 209, 211, 225, 222,
204, 202, 163, 149, 66, 39, 24, 15, 0

0 to 115: 115, 95, 216, 201, 217, 232, 248, 231, 191, 176, 114, 209,
211, 225, 222, 204, 202, 163, 149, 66, 39, 24, 15, 0

0 to 123: 123, 106, 84, 79, 70, 30, 43, 82, 85, 152, 143, 175, 246,
244, 179, 193, 243, 192, 162, 147, 140, 117, 167, 224, 218, 146, 145,
120, 161, 186, 177, 169, 190, 247, 220, 189, 200, 203, 249, 229, 170,
182, 242, 223, 198, 94, 141, 135, 108, 110, 122, 139, 156, 221, 219,
214, 212, 210, 207, 205, 196, 181, 184, 197, 230, 188, 138, 151, 226,
208, 187, 168, 144, 97, 80, 59, 50, 48, 45, 76, 241, 228, 153, 115,
95, 216, 201, 217, 232, 248, 231, 191, 176, 114, 209, 211, 225, 222,
204, 202, 163, 149, 66, 39, 24, 15, 0

0 to 241: 241, 228, 153, 115, 95, 216, 201, 217, 232, 248, 231, 191,
176, 114, 209, 211, 225, 222, 204, 202, 163, 149, 66, 39, 24, 15, 0

0 to 246: 246, 244, 179, 193, 243, 192, 162, 147, 140, 117, 167, 224,
218, 146, 145, 120, 161, 186, 177, 169, 190, 247, 220, 189, 200, 203,
249, 229, 170, 182, 242, 223, 198, 94, 141, 135, 108, 110, 122, 139,
156, 221, 219, 214, 212, 210, 207, 205, 196, 181, 184, 197, 230, 188,
138, 151, 226, 208, 187, 168, 144, 97, 80, 59, 50, 48, 45, 76, 241,
228, 153, 115, 95, 216, 201, 217, 232, 248, 231, 191, 176, 114, 209,
211, 225, 222, 204, 202, 163, 149, 66, 39, 24, 15, 0

0 to 249: 249, 229, 170, 182, 242, 223, 198, 94, 141, 135, 108, 110,
122, 139, 156, 221, 219, 214, 212, 210, 207, 205, 196, 181, 184, 197,
230, 188, 138, 151, 226, 208, 187, 168, 144, 97, 80, 59, 50, 48, 45,
76, 241, 228, 153, 115, 95, 216, 201, 217, 232, 248, 231, 191, 176,
114, 209, 211, 225, 222, 204, 202, 163, 149, 66, 39, 24, 15, 0

For the second part we should read tinyDG.txt file, implement it as a directed graph and run the topological sort algorithm. The output should be a LinkedList.

The output for the tinyDG.txt is:

[9, 4, 12, 12, 11, 10, 6, 6, 9, 9, 4, 8, 0, 4, 3, 2, 5, 2, 0, 3, 5, 1]

4. Conclusion

The Depth First Search algorithm is an interesting way for traversing a whole graph and can be used for a lot of different things, as getting the path from a vertex to another, as made in this lab.

For the topological sorting algorithm it is interesting because the sorting can be made based in a lot of different patterns. The orientation for this lab was a little bit confusing for me, since we just needed to call DFS for computing finishing times for each vertex and then add each vertex to a Linked List.