CS 594/690, Graph Algorithms, Applications and Implementations
Spring 2017, Homework 2

1. Consider the following algorithm, *BruteForceEdgeConnectivity*, which takes as input a graph *G* of order *n*, and outputs the edge connectivity of *G*. Assume the function *isConnected* takes *T*(*n*) time and returns true or false, based on whether *G* is connected.

```
BruteForceEdgeConnectivity(G)
   If not isConnected($G$)
      Return 0
   For $k$ = 1 to $n$-1
      For each set $S$ of $k$ edges in $G$
         If not isConnected($G \setminus S$)
            Return $k$
   Return 0
```

a. On which graphs, if any, does *BruteForceEdgeConnectivity* reach its final return statement?

b. On what type(s) of graphs does the worst-case behavior occur?

c. What is the time complexity of *BruteForceEdgeConnectivity* as a function of $T(n)$ and $\lambda(G)$, where $\lambda(G)$ denotes the (maximum) edge connectivity of *G*?

2. Write a program that inputs a simple, unweighted, undirected graph from a file and outputs vertices in the order visited by a DFS. The program should take two command-line arguments: the name of the graph file and the index of the vertex on which to start the DFS. Whenever your DFS has a choice between more than one child vertices, always choose the lower-indexed vertex first.

A user should see something very similar to the following when invoking your program.

```
>./DFS graph2.txt 3
3 4 0 7
>
```

3. Write a program with the same specifications as problem 2, but using BFS.

4. Write a program that inputs a simple, unweighted, undirected graph from a file and outputs to standard output the connected components of the graph, one per line. Use either a DFS or a BFS as a subroutine. When selecting an unvisited vertex to start each DFS/BFS, always choose the lowest-indexed unvisited vertex. A user should see something very similar to the following when invoking your program.

```
>./ConnectedComponents graph2.txt
0 4 3 7
1 5 8
2 6
>
```

5. Compare the performance of your DFS and BFS programs on the 15 graphs at http://web.eecs.utk.edu/~cphillip/cs594_spring2017/HW2_graphs/. Provide the timings of your DFS and BFS on each graph. You may find it helpful to use the Linux **time** command, as in the following example.

```
> time ./ConnectedComponents graph2.txt
```

On how many graphs does DFS run faster than BFS, and vice versa? What conclusions, if any, can you draw about the relative performance of DFS and BFS?

The following apply as usual.

- You may choose any programming language you wish, as long as your program compiles and runs when invoked from the Linux command line on the EECS Linux machines, using only software currently installed. I will test your code on one of the Hydra Lab machines.
- Do not use any library routines specifically designed for graphs (e.g. Boost).
- Include an example how to compile and/or invoke your program in your README.txt file.
- Put your answers to non-programming questions in a plain text file named README.txt.

Submit your program by emailing all files necessary to compile and run your code to cphill25@utk.edu prior to the beginning of class next Wednesday, January 25. Include your README.txt file. If you have any questions, please do not hesitate to email me or drop by during office hours.