ENEE641-ENME808x
Software Project Specification

Due: December 12, 2016

Objective:

Given a directed graph G = (V,E), apply the topological sort algorithm using DFS, as described in the section 22.4 of the textbook, to find out one topological order, or report that it contains a cycle. The topological order is not necessarily unique. If G contains an edge (u,v), then vertex u should appear before v in the ordering. You need to implement the algorithm in a C program, and analyze its time requirement. Your program should have both the best time complexity and the best constant factor (assuming that $|V| = o(|E|)$).

Input Format:

In each of the input graphs, vertices are numbered from 1 to $|V|$. Each row includes a vertex, followed by zero or more other vertices.
For example:
1 4
2
3 1 2
4 2

This input should be interpreted as follows. The directed edges of G are (1,4), (3,1), (3,2) and (4,2).

Output format:

A:

The output file of your program should include the following:

vertice-number1 vertice-number2 vertice-number3 ..... vertice-number10
vertice-number11 vertice-number12 vertice-number13 ..... vertice-number 20
.
.
..... vertice-number(V-1) vertice-numer(V)

The output format can be understood in the following manner. Appended, the lines will contain the vertices' number in topological order. In the example above the output will be:

3 1 4 2
In case the input graph has a cycle, output a single line:
This graph has a cycle!

B:

Only for the input graph marked as number 1 ("in1.txt"), do the following.

The output file of your program should include the following:

Vertex 1 : total # operations (C commands) charged to Vertex 1 in unary.
Vertex 2 : same for vertex 2.
...
Vertex n : same for vertex n.
Edge from Vertex number x to Vertex number y: total # operations (C commands) charged to this edge in unary.
...
Edge from Vertex number z to Vertex number w: same for this edge.
Namely, for each of the n vertices and each edge, print out in unary the total number of operations that have been charged to that item. The printed out numbers should match as closely as possible with your asymptotic analysis.

Total number of operations charged to all vertices is : ...
Total number of operations charged to edges is : ...
Total number of operations is : ...

C:

For input graphs number 2,3,...,7 only ("in2.txt" to "in7.txt")

Total number of operations charged to vertices is : ...
Total number of operations charged to edges is : ...
Total number of operations is : ...

Notes:

There is a total of 7 input graphs for this project. In your report, you only need to print out results for "in1.txt" (15 vertices) in format A&B. For the rest of the input graphs, print out the results in format A&C.
Input files are uploaded on the course website, `http://www.umiacs.umd.edu/~vishkin/TEACHING/enee641f16`.

Output:

The following is expected from you:

1. Your code (with documentation) along with a separate overview.
2. Asymptotic analysis of the runtime of your code up to a constant factor, counting all the C commands in your code.
3. Experimental results for the inputs provided. Report the topological order as well as the actual runtime of the program, counting all C command executed. (See Output format for details. The extra code introduced just for counting of executed commands should be properly marked, and not included in the counting.)

Deliverables:

Submit the following to the TA by the due date (amir.majlesi.kupaei@gmail.com).

- Source code compatible with glue (allowing us to run your code). Name your source code "UID.c", e.g. "123456789.c". After compiling your code, it should be executable in the following format.
  "./UID input.txt output.txt"
  "input.txt" could be any of the 7 input files provided, and "output.txt" is the output of your program.
- Report (at most 4 page report) illustrating points [1,2,3] under Output above.
- The output files for the provided inputs. Name the output files "out1.txt" to "out7.txt".

Grading Policy:

The grading will account for issues such as: correctness of code & explanation, complexity of code (use as few C commands as possible; constant factors matter!), correctness of asymptotic analysis, and correctness of counting.