CS 6301.502. Implementation of advanced data structures and algorithms

Fall 2017

Short Project 3: Depth-first search (DFS)

Fri, Sep 8, 2017

Version 1.0: Initial description (Fri, Sep 8).

Due: 11:59 PM, Sun, Sep 17.

Solve as many problems as you wish. Maximum score: 50

DFS and its applications will be discussed in class on Fri, Sep 8.

1. [30 points]

Topological ordering of a DAG.

Implement two algorithms for ordering the nodes of a DAG topologically.

Both algoritms should return null if the given graph is not a DAG.

/\*\* Algorithm 1. Remove vertices with no incoming edges, one at a

\* time, along with their incident edges, and add them to a list.

\*/

List<Graph.Vertex> toplogicalOrder1(Graph g) { ... }

/\*\* Algorithm 2. Run DFS on g and add nodes to the front of the output list,

\* in the order in which they finish. Try to write code without using global variables.

\*/

List<Graph.Vertex> toplogicalOrder2(Graph g) { ... }

2. [30 points]

Strongly connected components of a directed graph. Implement the

algorithm for finding strongly connected components of a directed

graph (see page 617 of Cormen et al, Introduction to algorithms,

3rd ed.). Run DFS on G and create a list of nodes in decreasing

finish time order. Find G^T, the graph obtained by reversing all

edges of G. Note that the Graph class has a field revAdj that is

useful for this purpose. Run DFS on G^T, but using the order of

the list output by the first DFS. Each DSF tree in the second DFS

is a strongly connected component.

int stronglyConnectedComponents(Graph g) { ... }

Each node is marked with a component number, and the function returns

the number of strongly connected components of G.

3. [30 points]

Is a given directed graph Eulerian?

A directed graph G is called Eulerian if it is strongly connected

and the in-degree of every vertex is equal to its out-degree. It

is known that such graphs have a tour (cycle that may not be

simple) that goes through every edge of the graph exactly once.

Write a function that tests whether a given graph is Eulerian.

Your algorithm need not find an Euler tour of the graph.

boolean testEulerian(Graph g) { ... }

4. [20 points]

Is a given directed graph a DAG (directed, acyclic graph)?

Solve the problem by running DFS on the given graph, and checking

if there are any back edges.

boolean isDAG(Graph g) { ... }

5. [50 points]

For a connected, undirected graph G=(V,E), an edge e in E is

called a bridge if the removal of e from G breaks the graph into 2

components. A vertex u in V is called a cut vertex if the removal

of u, along with its incident edges from G breaks it into 2 or more

components. The problem of finding bridges and cut vertices of a

given graph will be discussed in class (see also Problem 22-2 in

Cormen et al's Introduction to Algorithms, 3rd ed).

/\*\* Find bridges and cut vertices of an undirected graph g. Assume that g is connected.

\* The list of bridges of g is returned by the function. Cut vertices are marked

\* by setting to true a boolean field "cut" defined for each vertex.

\*/

List<Graph.Edge> findBridgeCut(Graph g) { ... }