Algorithms Lab 9

(Graphs basics)

csci 2200, Laura Toma. Bowdoin College

This week's topics

- connectivity, cycles, strong connectivity, paths, reachability
- BFS, DFS, topological sorting

Review the topics discussed in class this week. This is a great time to understand all the details and ask questions.

In lab: (COLLABORATION LEVEL 0: EVERYTHING ALLOWED)

1. (repeat from class) Suppose you are given an undirected graph given as an adjacency list, and you want to figure out if it is a *tree* (a tree is an undirected graph that's connected and has no cycles). Describe an algorithm to solve this problem, and analyze its running time.

2. (repeat from class) In a directed graph, two vertices u and v are said to be in the same *strongly connected component (SCC)* if u can reach v and v can reach u.

   (a) Describe a linear time algorithm for computing the *strong component* containing a given vertex v.

   (b) On the basis of that algorithm, describe a simple quadratic time algorithm for computing the strong components of a directed graph G.

3. (from: Sedgewick Wayne 4.2.27) Explain why the following algorithm does not necessarily produce a topological order: Run BFS, and label the vertices by increasing distance to their respective sources. Note: To prove that a certain algorithm does not work, it's sufficient to show a counter-example.

4. All-pair connectivity: Given a graph, support queries of the form: are u, v connected? (a) with no-preprocessing, how fast can you answer a query?; (b) Pre-process the graph into an approptiate data structure in order to answer connectivity queries in $O(1)$ time.

5. All-pair reachability: Given a graph, support queries of the form: given u, v, is v reachable from u? (a) no pre-precessing; (b) with pre-precessing, in $O(1)$ time per query;

6. A graph is called *bipartite* if it is possible that its vertices can be assigned one of two colors, such that no edge connects vertices of the same color. Given an undirected graph $G = (V, E)$, come up with an algorithm to determine if G is bipartite and analyze its running time.

Homework problems (COLLABORATION[1] LEVEL:1)

1. Suppose the degree requirements for a computer science major are organized as a DAG (directed acyclic graph), where vertices are required courses and an edge (x, y) means course x must be completed prior to beginning course y. Make the following assumptions:

   - All prerequisites must be obeyed.
   - There is a course, CPS1, that must be taken before any other course.
   - Every course is offered every semester (unlike our department).
   - There is no limit to the number of courses you can take in one semester (again, unlike our department).

   Describe an efficient algorithm to compute the minimum number of semesters required to complete the degree and analyze its running time.

2. (CLRS 22.4-2) Give a linear-time algorithm that takes as input a directed acyclic graph $G = (V, E)$ and two vertices s and t, and returns the number of simple paths from s to t in G. For example, the DAG in Fig. 22.8 CLRS contains exactly four simple paths from p to v: pov, $poryv$, $posryv$ abd $psryv$. Your algorithms needs to only count the simple paths, not list then.

   Hint: dynamic programming

3. Assume you are given a DAG (directed acyclic graph) G, and you want to compute *longest* paths rather than shortest. The edges do not have weights, the length of a path is the number of edges on the path.

   (a) Given a vertex u in G, describe how to compute the longest path from u.

[1] Collaboration level 1: verbal collaboration without solution sharing. You are allowed and encouraged to discuss ideas with other class members, but the communication should be verbal and additionally it can include diagrams on board. Noone is allowed to take notes during the discussion (being able to recreate the solution later from memory is proof that you actually understood it). Communication cannot include sharing pseudocode for the problem. Check complete guidelines at: https://turing.bowdoin.edu/dept/collab.php

(b) Describe how to compute the longest path in G.

Hint: dynamic programming

4. Given a DAG, design a linear time algorithm to determine whether there is a directed path that visits each vertex exactly one.

   Note:

Notes: In an undirected graph G: A path that visits each vertex exactly once is called a *Hamiltonian path*. A cycle that visits each vertex once is called a *Hamiltonian cycle*, and a graph that has a Hamiltonian cycle is called a *Hamiltonian graph*. The problem of determining whether an arbitrary graph has a Hamiltonian path/cycle is known to be NP-complete. On the special class of DAGs, it can be solved in linear time. Also, the problem of determining the *longest path* in an arbitrary graph is known to be NP-complete. On the special class of DAGs, it can be solved in linear time.