Assignment 12

Module: Graphs

Honor code: *Work on this assignment alone or with one partner. Between different teams, collaboration is at level 1 [verbal collaboration only]. There are lots of resources online, such as animations, visualizations, practice problems, videos, and solutions— which you are encouraged to explore to deepen your understanding. However, you must be careful not to search for the specific problems in the assignment with the intent of getting hints for the solution. Searching for the assignment problems on the internet violates academic honesty for this class.*

---

1. You are given a DAG (directed acyclic graph) G. In this problem you want to compute *longest* paths rather than shortest. The edges in G do not have weights and the length of a path is assumed to be the number of edges on the path.

   (a) Given a vertex u in G, describe how to compute the longest path from u (to any vertex in G). Ideally your algorithm will run in $O(|V| + |E|)$ time.

   (b) Describe how to compute the longest path in G. Ideally your algorithm will run in $O(|V| + |E|)$ time.

   We expect: pseudocode, justification and run time analysis.

   Notes: the problem of determining the *longest path* is known to be NP-complete on arbitrary graphs. But, on the special class of DAGs, it can be solved in linear time, which you'll do in this problem!

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

   We expect: pseudocode, justification, analysis.

   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. But, on the special class of DAGs this problems can be solved in linear time, which you'll do in this problem!

3. The degree requirements for a major can be modeled 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. Let's consider a hypothetical Computer science major with the following assumptions:

   - You must take **all** CS classes listed in the requirements.

- All prerequisites must be obeyed (no instructor permission).
- There is a course, csci101, that must be taken before any other course by everyone (no alternate placement).
- Every course is offered every semester (unlike our department).
- There is no limit to the number of courses you can take in one semester, and, you are guranteed to get into any CS course you register for (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. Aim for $O(|V| + |E|)$ time.

We expect: pseudocode, justification, analysis.