Good problems

[DPV] 3.5. The *reverse* of a directed graph $G = (V, E)$ is another directed graph $G^R = (V, E^R)$ on the same vertex set, but with all edges reversed; that is, $E^R = \{(v, u) : (u, v) \in E\}$.

Give a linear-time algorithm for computing the reverse of a graph in adjacency list format.

[DPV] 3.9. For each node u in an undirected graph, let `twodegree`$[u]$ be the sum of the degrees of u's neighbors. Show how to compute the entire array of `twodegree`$[\cdot]$ values in linear time, given a graph in adjacency list format.

[DPV] 3.13. *Undirected vs. directed connectivity.*

(a) Prove that in any connected undirected graph $G = (V, E)$ there is a vertex $v \in V$ whose removal leaves G connected. (*Hint:* Consider the DFS search tree for G.)

(b) Give an example of a strongly connected directed graph $G = (V, E)$ such that, for every $v \in V$, removing v from G leaves a directed graph that is not strongly connected.

(c) In an undirected graph with 2 connected components it is always possible to make the graph connected by adding only one edge. Give an example of a directed graph with two strongly connected components such that no addition of one edge can make the graph strongly connected.

[DPV] 3.16. Suppose a CS curriculum consists of n courses, all of them mandatory. The prerequisite graph G has a node for each course, and an edge from course v to course w if and only if v is a prerequisite for w. Find an algorithm that works directly with this graph representation, and computes the minimum number of semesters necessary to complete the curriculum (assume that a student can take any number of courses in one semester). The running time of your algorithm should be linear.

[DPV] 3.18. You are given a binary tree $T = (V, E)$ (in adjacency list format), along with a designated root node $r \in V$. Recall that u is said to be an *ancestor* of v in the rooted tree, if the path from r to v in T passes through u.

You wish to preprocess the tree so that queries of the form "is u an ancestor of v?" can be answered in constant time. The preprocessing itself should take linear time. How can this be done?

[DPV] 3.24. Give a linear-time algorithm for the following task.

Input: A directed acyclic graph G

Question: Does G contain a directed path that touches every vertex exactly once?

Interesting problems

[DPV] 3.23. Give an efficient algorithm that takes as input a directed acyclic graph $G = (V, E)$, and two vertices $s, t \in V$, and outputs the number of different directed paths from s to t in G.

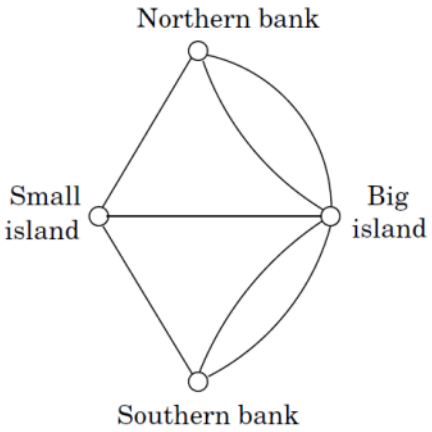
[DPV] 3.23. Give an efficient algorithm that takes as input a directed acyclic graph $G = (V, E)$, and two vertices $s, t \in V$, and outputs the number of different directed paths from s to t in G.

[DPV] 3.27. Two paths in a graph are called *edge-disjoint* if they have no edges in common. Show that in any undirected graph, it is possible to pair up the vertices of odd degree and find paths between each such pair so that all these paths are edge-disjoint.

[DPV] 3.26. An *Eulerian tour* in an undirected graph is a cycle that is allowed to pass through each vertex multiple times, but must use each edge exactly once.

This simple concept was used by Euler in 1736 to solve the famous Konigsberg bridge problem, which launched the field of graph theory. The city of Konigsberg (now called Kaliningrad, in western Russia) is the meeting point of two rivers with a small island in the middle. There are seven bridges across the rivers, and a popular recreational question of the time was to determine whether it is possible to perform a tour in which each bridge is crossed *exactly once*.

Euler formulated the relevant information as a graph with four nodes (denoting land masses) and seven edges (denoting bridges), as shown here.



Notice an unusual feature of this problem: multiple edges between certain pairs of nodes.

(a) Show that an undirected graph has an Eulerian tour if and only if all its vertices have even degree. Conclude that there is no Eulerian tour of the Konigsberg bridges.

(b) An *Eulerian path* is a path which uses each edge exactly once. Can you give a similar if-and-only-if characterization of which undirected graphs have Eulerian paths?

(c) Can you give an analog of part (a) for *directed* graphs?

[CLRS]

22-4 Reachability

Let $G = (V, E)$ be a directed graph in which each vertex $u \in V$ is labeled with a unique integer $L(u)$ from the set $\{1, 2, \ldots, |V|\}$. For each vertex $u \in V$, let $R(u) = \{v \in V : u \rightsquigarrow v\}$ be the set of vertices that are reachable from u. Define $\min(u)$ to be the vertex in $R(u)$ whose label is minimum, i.e., $\min(u)$ is the vertex v such that $L(v) = \min\{L(w) : w \in R(u)\}$. Give an $O(V + E)$-time algorithm that computes $\min(u)$ for all vertices $u \in V$.

Challenging problems (optional)

[CLRS]

22-3 Euler tour

An ***Euler tour*** of a strongly connected, directed graph $G = (V, E)$ is a cycle that traverses each edge of G exactly once, although it may visit a vertex more than once.

a. Show that G has an Euler tour if and only if in-degree(v) = out-degree(v) for each vertex $v \in V$.

b. Describe an $O(E)$-time algorithm to find an Euler tour of G if one exists. (*Hint:* Merge edge-disjoint cycles.)

[Remark:] There is a very simple program to find an Euler tour. Find it!

[DPV] 3.28. In the 2SAT problem, you are given a set of *clauses*, where each clause is the disjunction (OR) of two literals (a literal is a Boolean variable or the negation of a Boolean variable). You are looking for a way to assign a value `true` or `false` to each of the variables so that *all* clauses are satisfied – that is, there is at least one true literal in each clause. For example, here's an instance of 2SAT:

$$(x_1 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee \overline{x}_3) \wedge (x_1 \vee x_2) \wedge (\overline{x}_3 \vee x_4) \wedge (\overline{x}_1 \vee x_4).$$

This instance has a satisfying assignment: set x_1, x_2, x_3, and x_4 to `true`, `false`, `false`, and `true`, respectively.

(a) Are there other satisfying truth assignments of this 2SAT formula? If so, find them all.

(b) Give an instance of 2SAT with four variables, and with no satisfying assignment.

The purpose of this problem is to lead you to a way of solving 2SAT efficiently by reducing it to the problem of finding the strongly connected components of a directed graph. Given an instance I of 2SAT with n variables and m clauses, construct a directed graph $G_I = (V, E)$ as follows.

- G_I has $2n$ nodes, one for each variable and its negation.
- G_I has $2m$ edges: for each clause $(\alpha \vee \beta)$ of I (where α, β are literals), G_I has an edge from from the negation of α to β, and one from the negation of β to α.

Note that the clause $(\alpha \vee \beta)$ is equivalent to either of the implications $\overline{\alpha} \Rightarrow \beta$ or $\overline{\beta} \Rightarrow \alpha$. In this sense, G_I records all implications in I.

(c) Carry out this construction for the instance of 2SAT given above, and for the instance you constructed in (b).

(d) Show that if G_I has a strongly connected component containing both x and $\overline{x}$ for some variable x, then I has no satisfying assignment.

(e) Now show the converse of (d): namely, that if none of G_I's strongly connected components contain both a literal and its negation, then the instance I must be satisfiable. (*Hint:* Assign values to the variables as follows: repeatedly pick a sink strongly connected component of G_I. Assign value `true` to all literals in the sink, assign `false` to their negations, and delete all of these. Show that this ends up discovering a satisfying assignment.)

(f) Conclude that there is a linear-time algorithm for solving 2SAT.