The first page of your homework submission must be a cover sheet answering the following questions. Do not leave it until the last minute; it's fine to fill out the cover sheet before you have completely finished the assignment. Assignments submitted without a cover sheet, or with a cover sheet obviously dashed off without much thought at the last minute, will not be graded.

- How many hours would you estimate that you spent on this assignment?

- Explain (in one or two sentences) one thing you learned through doing this assignment.

- What is one thing you think you need to review or study more? What do you plan to do about it?

Question 1. Given a simple graph G with vertices numbered from 1 to n, an *adjacency matrix* for G is an $n \times n$ matrix A where $A[i,j] = 1$ if i and j are adjacent, and $A[i,j] = 0$ otherwise. In other words, the entry at row i and column j tells us whether there is an edge between i and j, with 1 for yes and 0 for no.

(a) Make a drawing of the graph represented by the adjacency matrix

$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}.$$

(b) Given a vertex u and the adjacency matrix A, how long does it take to list all the neighbors of u: $\Theta(n)$, or $\Theta(\deg(u))$? Why?

$\deg(u)$ denotes the *degree* of u.

(c) A *directed* graph is a graph where each edge has a particular *direction*, that is, an edge is an *ordered pair* of vertices rather than just a set of two vertices. Typically such graphs are drawn with a little arrow on each edge showing which direction is goes.

Explain how to generalize adjacency matrices to represent directed graphs.

(d) A *weighted* graph is a graph where each edge has an associated number, or *weight*, representing something like cost, time, or distance.

Explain how to generalize adjacency matrices to represent weighted graphs.

(e) (Advanced, optional) Let A be the adjacency matrix for a simple (undirected, unweighted) graph G. What does the matrix A^2 represent?

Question 2. Given a simple graph G, an *adjacency list* representation for G consists of a map (*aka* dictionary) associating each vertex with the *list* (or set) of all its adjacent neighbors. In other words, using Java-like notation, `Map<Vertex, List<Vertex> >`.

(a) Make a drawing of the graph represented by the map

$$\{1 \mapsto [3,4], 2 \mapsto [3], 3 \mapsto [1,2,4], 4 \mapsto [1,3]\}.$$

(b) Given a vertex u and an adjacency list representation of a graph G, how long does it take to list all the neighbors of u: $\Theta(n)$, or $\Theta(\deg(u))$? Why?

(c) Explain how to generalize adjacency lists to represent directed graphs.

(d) Explain how to generalize adjacency lists to represent weighted graphs.

Question 3. Given a graph T which is guaranteed to be a tree, and a particular starting vertex s, design an $O(V + E)$ algorithm to find the distance of every vertex from s. In other words, your algorithm should output some kind of dictionary/map or array associating every vertex v to a natural number $\ell(v)$ which is the length of a path from s to v. Note that the length of a path from s to itself is 0, the length of a path from s to any of its neighbors is 1, and so on.

You may assume that T is represented in whichever way is easiest for you (adjacency matrix or adjacency list).

Explain how your algorithm works, and analyze its asymptotic running time. Psuedocode is fine, though you may also choose to write real code if you wish.

Question 4. Given a graph G with n vertices labelled 0 through $n-1$, design an $O(V + E)$ algorithm to find and label all the connected components of G. That is, your algorithm should output a dictionary or array associating each vertex v with a component label $c(v)$ such that two vertices u and v have the same component label (that is, $c(u) = c(v)$) if and only if u and v are connected. For example, you might give all the vertices in the first connected component the label 1, all the vertices in the second connected component the label 2, and so on.

You may assume that G is represented in whichever way is easiest for you (adjacency matrix or adjacency list).

Explain how your algorithm works, and analyze its asymptotic running time. Psuedocode is fine, though you may also choose to write real code if you wish.

Question 5. Prove: for all $n \geq 1$, if G is a simple connected graph with n vertices and $n-1$ edges, then G has no cycles. (This is the third part of the proof from class, characterizing trees as having any two out of three properties.)

5(a)

5(b)