Due Dates: One numbered question of your choice is due by **11:59PM Sunday, 2 October** and will be graded on completion only (1 point). Your full answers to all questions are due by **11:59PM Wednesday, 5 October**. Each solution will be graded for correctness and clarity (4 points per question).

At the top of your write-up for each problem, estimate the amount of time you spent on the problem, list your collaborators, and briefly describe the nature of your collaboration.

1. **Getting to know some graph definitions**

   For the following, give graph examples with the specified properties. Include a brief explanation of why your example satisfies the definitions of those properties. Notationally, please mimic the textbook by calling your graphs $G = (V, E)$, and letting $n = |V|$ and $m = |E|$. Also, when I say just *graph*, I mean *undirected graph*. If I want you to produce a directed graph, I'll say *directed graph* or *digraph*.

   Please show your examples as images rather than lists of edges. Look at latex-miscellany.pdf and latex-miscellany.tex for some help drawing graphs and including images in LaTeX documents. I'm happy to have your graphs appear in either form–that is, the graph-drawing LaTeX module is wonderful, but if it's easier for you, go ahead and draw a graph by hand and include a photo of it in your PDF file.

   (a) Show a graph that is not a tree, but for which $m = n - 1$.

   (b) Show a directed graph that is weakly connected but not strongly connected.

   (c) Show a graph that has a Hamiltonian cycle but not an Eulerian cycle

   (d) Show a a connected weighted graph and two nodes u and v for which there is no lowest-weight path from u to v. (Note that having non-unique lowest-weight paths–i.e. two or more paths from u to v that have the identical, lowest weight is not what I'm after here. I'm looking for a situation where there are zero paths that can qualify as having the lowest weight. Which is weirder than "non-unique lowest-weight path".)

   (e) How different can the BFS and DFS trees of a given graph be? Here's one approach to this question. Let the diameter of a graph be the longest number of edges in the shortest path between two nodes in the graph. (This is often called the "length of the longest shortest path" in the graph.) For this last item, show: a graph G, a node s, a BFS tree T_B, and a DFS tree T_D such that the diameter of T_D is at least 2 times as large as the diameter of T_B. Then, show how you would extend your example to make the factor 3 or 10 or 1000 instead of 2.

2. **6th cousins 3 times removed? What's a 6th cousin? And removed from what?**

   The English terminology describing genealogical relationships is a bit of a mess. Of course, there are a ton of specialized words expressing family relationships (*mother*, *brother*, *niece*, etc.). But there's also a slightly odd designation system for cousins. I suspect something like this is also true in other languages, but I know for sure that the terminology in English is wacky.

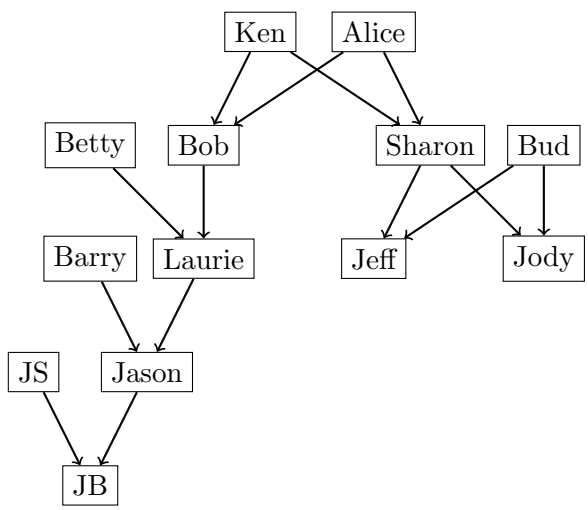
   Let's take an example. Ken and Alice got married in 1930 and had a son Bob in 1932 and a daughter Sharon in 1935. Sharon had two children, my sister Jody and me, Jeff. My uncle

Bob had a daughter Laurie. Laurie had a son Jason. And Jason has a 1-year-old who we will call JB (for *Jason's Baby*, since I figure we should let her family decide when to put her name on the internet). I tell you all this so I can draw this conclusion: ***JB and I are first cousins twice removed***.

This lineal kinship terminology (or, if you prefer, the *removed cousins* system) has two integer parameters–what degree of cousin are you (D) and how far removed (R)?

For our purposes, define a *family tree* be a directed graph $G = (V, E)$ where each node in V represents an individual human being and each edge $(u, v) \in E$ represents a biological parent (u) to child (v) relationship. Because any family tree data will be necessarily incomplete, we will assume that each node has either two, one, or zero in-edges. (We're ignoring the many complications present in real-life family trees, including adoption, surrogacy, mitochondrial DNA, divorce, people lying, absence of historical information, etc.). Also, note that despite its name, large enough family trees are generally not trees in the graph theoretical sense. They are, however DAGs (except in the occasional time-travel story).

Here, for example, is a family tree that includes the pieces I mentioned above (plus a few spouses to round things out–Bud, Betty, Barry, and Jason's spouse JS).



Given two members $x, y \in V$, you compute the degree $D_{x,y}$ and removal $R_{x,y}$ of their cousin relationship by first finding the most recent common ancestor z of x and y. Let L_x be the number of edges in the path from z to x. Define L_y similarly. Then the degree $D_{x,y}$ of the relationship between x and y is $D_{x,y} = min(L_x, L_y) - 1$, and the removal $R_{x,y} = max(L_x, L_y) - min(L_x, L_y)$.

For example, let's look back at the little fragment of my family tree described above. (You should draw it!) If you let $x = Jeff$ and $y = JB$, you get $z = Alice\ (or\ Ken)$, $L_x = 2$, $L_y = 4$, $D = min(L_x, L_y) - 1 = 1$ and $R = L_y - L_x = 2$. That is, JB and I are Dth cousins R times removed, that is, first cousins twice removed.

This naming system extends nicely to far-flung relationships. It's a little weird, on the other hand, for closer relatives (e.g., my uncle is my 0th cousin 1 time removed, and I am my own -1st cousin 0 times removed), but it still basically works. (If you're home at Thanksgiving,

try referring to your sister as your 0th cousin 0 times removed and let me know how that goes!)

Try playing with this system in your own family for a little while to get used to how D and R are connected to the graph structure.

OK, it's now time for an algorithm. For this exercise, I want you to describe an efficient algorithm to compute $D_{x,y}$ and $R_{x,y}$ given a family tree $G = (V, E)$ and two nodes $x, y \in V$.

You may assume that G is preloaded using adjacency lists (thus requiring $O(m+n)$ bytes of memory where $n = |V|$ and $m = |E|$). You may preprocess G so long as you don't use more than $O(m+n)$ additional bytes of memory or more than $O(m+n)$ preprocessing time. (In particular, you are *not* allowed to sneakily precompute $D_{x,y}$ and $R_{x,y} \forall x, y \in V$ and then just store the answers in a big $n \times n$ lookup table. Not that you could fit that table in memory if n got big enough anyway, but still.)

As usual, please organize your writeup like so:

- Preamble, only if there's any intro material you want to include.
- Description of the algorithm
- Proof/explanation of the correctness of your algorithm
- Analysis of running time
- In this case, also demonstrate that any additional memory you're using takes $O(m+n)$ space