Homework 2: BFS, DFS, and Graphs

Due Date: This assignment is due at 11:59 PM Friday January 29 (Seattle time, i.e. GMT-8).
You will submit the written problems as a PDF to gradescope. Please put each numbered problem on its own page of the pdf (this will make selecting pages easier when you submit).

Collaboration: Please read the full collaboration policy. If you work with others (and you should!), you must still write up your solution independently and name all of your collaborators somewhere on your assignment.

Houmework Guidelines Please read the full homework style guide, in particular: the phrase "describe an algorithm" includes:

- Describing the algorithm itself
- Justifying its correctness
- Finding a big-O description of its running time

Unless otherwise noted, you are allowed to use algorithms described in class (or prerequisite courses) as though you had library implementations of them. For example, you can say "run the BFS-based 2-coloring algorithm from class on the graph G" or "run the bipartite checking algorithm from lecture 5 on G."

Since this is a course about efficient algorithms, we do deduct for algorithms that are slower than necessary, but (unless otherwise noted) we do not care about constant factors. In general, an algorithm that is correct but (mildly) slower than optimal will get more points than an algorithm that is fundamentally incorrect, but fast.

1. Comparisons [12 points]

You have designed a machine learning system to classify dog breeds – given a picture of a dog, the system prints out what breed it thinks the dog is.

You'd like to build a new dataset to test your system on; you've scraped the internet for TONS of dog pictures. A few of the pictures came "pre-labeled" with the dog breed, but many of them weren't labeled with the dog breed. And they won't make for a good test set without knowing what the right answer is supposed to be. In this problem we'll figure out how to quickly extend a few accurate labels to the resut of the pictures we've scraped.

You can use an online survey system (like Mechanical Turk) to get humans to identify the pictures for you, but the average person isn't an expert on dog breeds, and might not identify the breed of a dog in a picture accurately. What humans can do consistently is tell whether two dogs are the same breed or not. So you set up your survey to show people a few dog pictures at the same time. The user will then identify (exactly) two of the pictures they were shown to say they believe those dogs are the same breed. You'll have each survey taker repeat this process for many sets of pictures (each time identifying exactly two dogs of the same breed).

You get the responses from your first survey participant. But you know that surveys like this aren't reliable – some people just click randomly (or are really bad at identifying dogs). You'd like to make sure a single person's responses are **consistent** with your starting data. For example, that from one person's data you can't find them saying picture A is the same breed as picture B, B is the same breed as C, and both A and C are pre-labeled dog pictures that you know are different breeds.

More formally, a person's responses are consistent if there is no sequence of pictures $p_1, \ldots, p_k$ such that p_1 and p_k are pre-labeled with different breeds, and for every i, the user marked p_i and p_{i+1} as the same breed.

Describe an algorithm to check if someone's responses are consistent with the starting data. When analyzing the running time, assume you have p unlabeled pictures, ℓ labeled-pictures, and that each person has identified k connections (i.e. identified k pairs as being the same breed).

(a) Describe a graph you can use in this problem: [4 points]

(i) What are your vertices?

(ii) What are your edges?

(iii) Briefly justify that given your starting data you could build your graph, and state how much time it would take (in terms of p, ℓ, k). (Our justification is 1 sentence; depending on what you choose for your graph, you may require more justification)

(b) Describe an algorithm to solve this problem. You do not have to give a formal proof of correctness, but should briefly justify why it works (2-3 sentences). [8 points]

2. The Same or Different from the last one [16 points]

You have many pictures of alpacas and llamas, but none have been labeled with which species they are. Your friend, an alpaca expert, has repeatedly told you that if you look at the ears you can tell whether two animals are of the same species. But you absolutely cannot remember what she said well enough to look at just one photo and tell whether it's an alpaca or llama.

What you can do, is given two photos say "those are the same species" or "those are different species." You do a bunch of such comparisons and write them all down. You then pick one of the photos and say "this animal is species A." Your goal is to perfectly place every animal into the buckets "species A" and "species B."

Given your data, you need to respond with one of three options:

- Inconsistent: you have analyzed the pictures in such a way that you can't possibly classify them all into species A and B correctly (for example you said photo A and B were the same species, photo B and C were different species, but A and C were the same species)
- Underspecified: your answers aren't inconsistent, but there is at least one picture that (from the data you've collected so far) could validly be either species A or species B.
- Exact answer: your answers aren't inconsistent, and every photo can be only one of species A or B.

Describe an algorithm to return which of those three cases the input matches, and if you're in the exact answer case to store the species labels for each photo.

For analyzing the running time, assume that you have p photos, s pairs identified as "the same" and d pairs identified as "different." Give a big-$\mathcal{O}$ bound in terms of p, s, d.

Note that your input in this problem is different from the last problem. In the last problem, you only got information of "these two are the same" (there was no way to identify two pictures as different). Here, you will have some pairs identified as "the same as each other" and others identified as "different from each other."

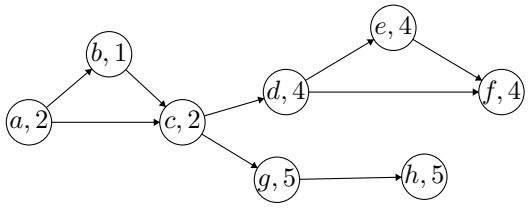
3. Points! [15 points]

You are given a **directed** graph G, and a start vertex u. Every vertex in the graph contains a number of `points` you can collect on the first time you visit the vertex (if you visit the vertex again you get no points the second time, but there's no penalty). You would like to plan a walk through the graph that collects the most points.

For this question, you do not have to prove your algorithms correct for any part. Instead, you should explain why you think it works in 2-3 sentences per part.

(a) Suppose G is strongly connected. Describe a walk that collects the maximum possible points (you don't need the shortest walk, just a walk). Describe an algorithm to find this walk. [4 points]

(b) Suppose G is a DAG. Describe an algorithm to find the walk that collects the most points. We recommend you consider the example below, where the best walk is: a, b, c, d, e, f. (Hint: this is conceptually easier if you do a topological sort first!) [5 points]



(c) Now, stitch together the last two parts and describe an algorithm to handle any directed graph. [6 points]

4. Find a Cut Edge [17 points]

In an undirected graph, a *cut edge* (also called a "bridge") is an edge such that if the edge is removed, the number of connected components will increase. In this problem we'll build up to using DFS to find a cut edge in an undirected graph. For simplicity, we will assume that the input graph G is connected.

(a) Convince yourself that there are no cross edges or forward edges when DFS is run on an undirected graph. You do not have to write anything for this part. [0 points]

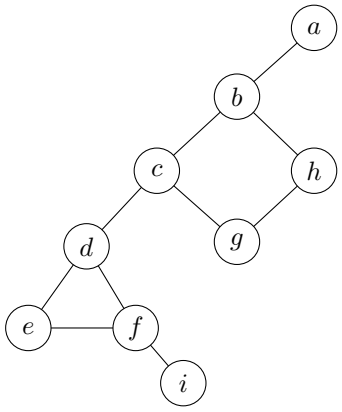
(b) Prove that a cut edge in a graph must be a tree edge. [3 points]

(c) To help us find out whether an edge is a cut edge, it will help to know how "high up" in the tree you could get from u by going down tree edges and taking a back edge up. Define the quantity "`low`" on each vertex:

$$\texttt{u.low} = \min\left\{\texttt{u.start}, \min_{v:(u,v) \text{ is a back edge}}\{\texttt{v.start}\}, \min_{d:d \text{ is a descendant of } u}\{\texttt{d.low}\}\right\}$$

In words, `u.low` is the minimum of: `u.start`, the `start` values of vertices that are (directly) connected to u by a back edge and the `low` values of the descendants of u in the DFS tree (i.e. any vertex which will be seen for the first time while u is on the stack)
Describe how to modify the DFS code from class to calculate `low` for each vertex as part of the DFS (you'll need to return an `int` variable with the recursive calls, instaed of having `void` as the return type). [5 points]

(d) Run DFS starting from a in this graph. Break ties by putting the alphabetically first one on the stack first. Record `start`, `end`, `low` for each vertex. [5 points]



(e) Based on the example and the facts given, describe how to use `start`, `end`, and `low` numbers to check if an edge (u, v) is a cut edge from the modified DFS calculation (you do not have to prove this answer correct, but make sure you've accounted for every requirement). [4 points]
Hint: an edge is a cut edge if and only if it's not part of at least one cycle. Think about how `low` cand tell you about whether an edge is part of a cycle.