Homework 2: BFS, DFS, and Graphs

Due Date: The written parts of this assignment will be due at 11:59 PM on Friday October 21. As always, that means that the TAs will grade **every** written problem you submit to gradescope before that deadline. You will also be able to resubmit two written problems with every assignment, starting with this homework. The coding problems may be submitted at any time through the last day of classes (Friday Dec. 9), but we strongly recommend you finish any you intend to do by the due date for the written assignment so you have time to do next week's problems next week.
Collaboration: You are allowed (and encouraged!) to discuss these problems at a high level with others. But, please read the full collaboration policy to ensure you are keeping your discussions at the right level, and remember to cite any collaborators.

Problems to Submit: We will count your 1 best mechanical question and your 3 best long-form questions.
We strongly recommend you skim all the problems in a section before attempting them. While we try to make problems in a given section of approximately equal difficulty, you may find some to be easier than others.

Directions 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." We also have a list of data structures and algorithms you can use from 373 here.

Since this is a course about efficient algorithms, algorithms that are slower than necessary are unlikely to get "E" scores, 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 an equal or higher score to an algorithm that is fundamentally incorrect, but fast.

Mechanical Problems

Both of the mechanical questions on this homework are about cut edges.

Definition: Cut Edge

> 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.

The mechanical questions in this assignment are designed to hint at an algorithm that discovers all cut edges in a graph.

First some background: when we run DFS in an undirected graph, we classify an edge based on how it was first discovered. So if we discover u first, then follow an edge to v, we think of (u, v) as going from u to v (even though the edge itself is undirected). When we process v, we don't go back along u, so we don't try to change its classification.

1. A Short Proof

Prove that in an undirected graph, if an edge is a cut edge, then it must be a tree edge in DFS.

Our proof is about 4 sentences. Hint: it may help you to notice that there are no cross or forward edges in DFS on an undirected graph.

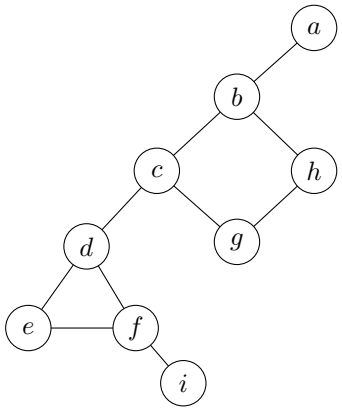
2. Run the Algorithm Yourself

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)

(a) 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.



(b) An edge is a cut edge if:

(i) It is a tree edge, discovered going from u to (new discovery) v, and

(ii) `v.low` is greater than `u.start`

List the cut edges in the graph above. We recommend you use this problem as a chance to check your numbers in the previous part!

Long-form Problems

3. Modify DFS

(a) Write pseudocode for a modification of DFS that takes as input a directed graph G and returns `true` if G is strongly connected (i.e., if you can get from every vertex to every other, and back the other way), and `false` otherwise.

For this problem, we want to see you actually make the modifications (e.g., you may not use another graph algorithm as a black box for this problem), but you may add any fields you like to the vertices and edges.

You can use code like `DFSWrapper` (from Lecture 7, slide 11) to control how DFS is called, but you must use only one run of DFS (or `DFSWrapper`)

Hint: You might use something like the `low` numbers from problem 2.

(b) Explain how your algorithm works – for any fields or variables you added, briefly describe what they do, and add a sentence or two to describe overall how the field(s)/variable(s) work together to get you the correct output. You do **not** need a full proof.

4. Comparisons

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 rest 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.

Sample Input I: Pre-labeled images: A is a Golden Retriever, B is a Golden Retriever, C is a Labrador Retriever, D,E,F,G are unlabeled
User input: A and D are the same breed, C and F are the same breed, E and G are the same breed, D and F are the same breed.

Correct Output: Inconsistent

Sample Input II: Pre-labeled images: A is a Golden Retriever, B is a Golden Retriever, C is a Labrador Retriever, D,E,F,G are unlabeled
User input: A and B are the same breed, A and E are the same breed, F and G are the same breed, D and F are the same breed.

Correct Output: Consistent

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:

  (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).

5. The Same or Different from the last one

You are working for an avid outdoor explorer and have been asked to design a tool to help them tell the difference between blueberries and nightshade berries for when they are backpacking! Nightshade berries looks similar to blueberries, but are toxic, so it's important you get every picture correct.

When designing the tool, you absolutely cannot remember how to look at just one photo and tell whether it's a blueberry or a nightshade berry. But you *can* tell if two berries are the same species as each other or not.

For example, given two photos you can say that "those are the same berry species" or "those are different berry species". You are not able to look at a photo and say "that's a blueberry." As a result, you do a bunch of comparisons and write them all down; finally, you pick one image and say, "this is an A berry". You goal for this tool is to perfectly identify each berry image as part of either berry group (Berry A or Berry 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 berry A and B correctly (for example you said photo 1 and 2 were the same berries, photo 2 and 3 were different berries, but 1 and 3 were the same berries).
- 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 berry A or berry B. Intuitively, this means there is an image that is not connected (even indirectly) to the image you started with.
- Exact answer: your answers aren't inconsistent, and every photo can be only one of berry 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.

Sample Input I: There are 4 images.
Image 1 and 2 are different berries
Image 3 and 4 are the same berries
Image 1 is "berry A"

Correct Output: Underspecified

Sample Input II: There are 4 images.
Image 1 and 2 are different berries
Image 1 and 4 are the same berries
Image 3 and 4 are different berries
Image 1 and 3 are different berries
Image 1 is "berry A"

Correct Output: Exact Answer

Sample Input III: There are 4 images.
Image 1 and 2 are different berries
Image 1 and 4 are the same berries
Image 3 and 4 are different berries
Image 1 and 3 are the same berries
Image 1 is "berry A"

Correct Output: Inconsistent

(a) Give pseudocode to run for this problem. You need to construct at least one graph (but you might want to make more than one, or alter the first one you make), you can assume creating a vertex or an edge takes $O(1)$ time each.

(b) Give the running time of your algorithm. 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."

This problem has multiple valid solutions – just because you and another student are coming at the problem differently, don't assume that only one of you can have a right idea!

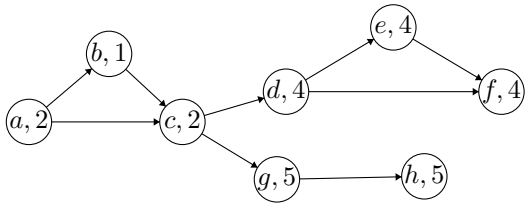
6. Points!

You are given a **directed** graph G, and a start vertex u. Every vertex in the graph contains a non-negative 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 figure out how many points you could collect in the graph.

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 the number of points in this walk. **Hint:** Don't find the walk itself! It takes longer to find the walk than to find its value.

(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 step should also take $\mathcal{O}(V + E)$ time)



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

Resubmissions

You may resubmit up to two problems this week. The due date for resubmissions is Monday October 24th. You will both: (1) submit a pdf to gradescope **in the resubmission box** (not the box where you submitted the problem originally), and then fill out the google form to tell us which one you resubmitted (we will post the form on the assignments page on the course website).