Section 2: Graph Search

1. Big-$\mathcal{O}$-No

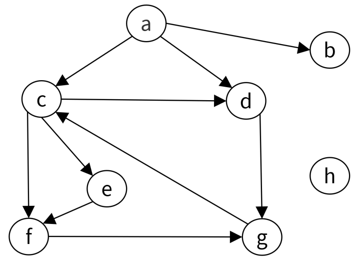
Put these functions in increasing order. That is, if f comes before g in the list, it must be the case that $f(n)$ is $\mathcal{O}(g(n))$. Additionally, if there are any pairs such that $f(n)$ is $\Theta(g(n))$, mark those pairs.

- $2^{\log(n)}$
- $2^{n \log n}$
- $\log(\log(n))$
- $2^{\sqrt{n}}$
- $3^{\sqrt{n}}$
- $\log(n)$
- $\log(n^2)$
- $\sqrt{n}$
- $(\log(n))^2$

Hint: A useful trick in these problems is to know that since $\log(\cdot)$ is an increasing function, if $f(n)$ is $O(g(n))$, then $\log(f(n))$ is $O(\log(g(n)))$. But be careful! Since $\log(\cdot)$ makes functions much smaller it can obscure differences between functions. For example, even though n^3 is less than n^4, $\log(n^3)$ and $\log(n^4)$ are big-Θ of each other.

2. Mechanical: DFS

Run Depth-First-Search on the graph below to classify the edges. Mark the start and end times for each vertex.



3. Another DFS application

In an *undirected* graph, call an edge $e = (u, v)$ a **cut edge** if u and v cannot reach each other in $G \setminus e$.

Intuitively, a cut edge is one which, when "cut" (i.e., removed) causes the graph to have more components than it did before. Cut edges are also called *bridges*.

We'll adapt DFS to find all the cut edges in a graph.

3.1. Setting the Foundation

Usually the first step in designing an algorithm is making absolutely sure you know what you're trying to find! That is, to make sure you really understand the problem. And secondly, to try to get some intuition for what the thing you're looking for "usually" looks like. The best way to do both of these steps is to look at a few examples. We'll do that before we get to designing the algorithm.

(a) We're in an undirected graph now, so edge classification will be different from lecture. Look back at the edge classification for DFS for directed graphs. Convince yourself that it's not possible to have a cross edge in a DFS on an undirected graph. (1-3 sentences of intuition, not a proof).

(b) Convince yourself it is not possible to have a forward edge in a DFS on an undirected graph. (2-3 sentences of intuition, not a proof).

(c) NOW, let's try to recognize cut edges. Prove that a cut edge must be a tree edge.

(d) Draw a graph, and a corresponding DFS, which has at least one tree edge that is not a cut edge, and at least one tree edge that is a cut edge.

3.2. Building The Bridges

Now, we're ready to start designing an algorithm.

(a) Prove that an edge is a cut edge if and only if it is not part of a simple cycle.

(b) Define `low`(u) to be the smallest `pre` number that can be reached by: starting at u, following any number of tree edges, and at most one back edge. Show that if (u, v) is a tree edge, discovered going from u to v, and (u, v) is part of a cycle, then `low`$(v) \leq$ `u.pre`. (Notice we're comparing v's low to u's pre!)

(c) Show that if (u, v) is a tree edge, discovered going from u to v, and (u, v) is not part of a cycle then `u.pre` $<$ `low`(v).

(d) Now, modify DFS to detect every cut edge of an undirected graph.

4. Graph Modeling

In this problem we're going to solve a classic riddle.

(a) First, you should solve the classic riddle yourself to get a feel for the problem.

> You are on the beach with a jug that holds exactly 5 gallons, a jug that holds exactly 3 gallons, and a large bucket. Your goal is to put **exactly** 4 gallons of water into the bucket. Unfortunately, the jugs are not graduated (e.g., you can't just fill the larger jug $4/5$ full). What you can do are the following operations.
>
> - Completely fill any of your jugs.
> - Pour from one of your containers into another until the first container is empty or the second is full.
> - Pour out all the remaining water in a container.
>
> How do you get 4 gallons of water into the bucket?

(b) Now, let's write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, ..., c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into the bucket.

5. Judging Books by Their Covers

You have a large collection of books, and just got a new bookshelf. For aesthetic reasons, you're going to arrange your books by the color of their covers (not by author or subject). You wish to put only books of a single color on any given shelf. You have a list of pairs of books which you know to be the same color. This list might be only partial (it's possible that $u, v,$ and w are all the same color, but your list might only have "u and v are the same color. w and v are the same color.", for example). You should assume that the "same color" relation is transitive.

Given your list, your job is to give an upper-bound on the number of shelves you need so that no shelf has more than one color of book. Describe an algorithm to give the best bound you can on the number of shelves needed.

You do not need a full proof of correctness, but you should describe the running time in terms of (whichever subset is appropriate): b, the number of books; p the number of pairs listed; s the number of shelves required (i.e., your final answer).