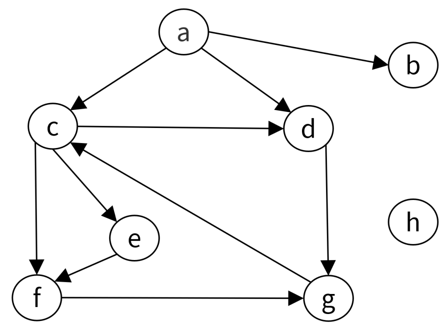
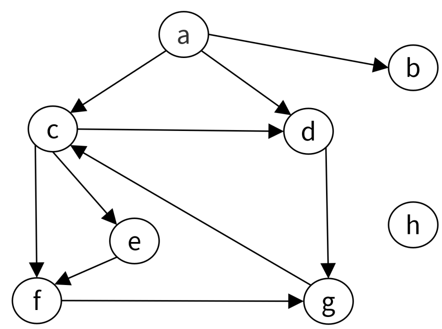
Section 2: Graph traversal and algorithm proofs

1. Warmup: BFS and DFS review

(a) Run breadth first search and record the layer of each vertex. For this problem, start with a, and say that a is in layer 0.



(b) Run depth first search, record the start/end times, and classify the edges (tree/back/forward/cross). When there are multiple choices for the next vertex, pick the alphabetically earliest one.



2. Investigating algorithm proofs

The purpose of this problem is to help you:

- Figure out how to start a proof about algorithms.
- Check the correctness of proofs.

To help with these situations, we will practice completely thinking through an algorithm and proof that you have already seen in lecture. **If you have trouble thinking through your proofs as we do here, it is probably either wrong or incomplete.** Then, we will summarize the general proof techniques.

(a) Answer the questions embedded in the proof below as you read. They are marked with $\triangleright$ and italics.

(b) Discuss with people near you: What is the general structure of a proof that an algorithm is correct? How is the proof related to the pseudocode?

Generic graph traversal

Input: Graph $G = (V, E)$ and starting vertex $s \in V$.
Expected output: Set of all vertices $v \in V$ such that there is a path from s to v.

$R \leftarrow \{s\}$
while there is $\{u, v\} \in E$ with $u \in R$ and $v \notin R$ **do**
    Add v to R.
return R

Claim. The algorithm above is correct.

The proof as written is a near carbon-copy of Monday's lecture. You will add details to it.

Proof.

1. **Claim.** The algorithm terminates.
   a. ▷ *What quantity increases every iteration, but is bounded? Why does it increase? Refer to particular line(s) in the code.*
   b. ▷ *Conclude that there are a bounded number of iterations.*
2. **Claim.** At termination, for all $v \in V$, we have $v \in R$ if and only if there exists a path from s to v.
   a. ($\Longrightarrow$) **Claim.** If $v \in R$, then there exists a path from s to v.
      ▷ *Prove this fact. Refer to particular line(s) in the code.*
   b. ($\Longleftarrow$) **Claim.** If there exists a path from s to v, then $v \in R$.
      i. Suppose for contradiction there exists $v \notin R$, but there is a path P from s to v.
      ii. In this case, we may actually take v to be the first node on P such that $v \notin R$.
         ▷ *Why is this allowed?*
      iii. The predecessor u of v in P satisfies $u \in R$ and $\{u, v\} \in E$.
         ▷ *Why does the predecessor exist? Refer to particular line(s) in the code. Why is $u \in R$ and $\{u, v\} \in E$?*
      iv. This is a contradiction.
         ▷ *What is the contradiction? Refer to particular line(s) in the code.*

   ▷ *Is it possible to directly prove this claim by induction on iterations, as you did for the $\Longrightarrow$ direction, instead of contradiction?*

$\square$

3. Judging books by their covers

You have a large collection of books and want to arrange them by color. You wish to put only books of a single color on any given shelf. Every pair of books is either "same color" or "not same color", and this relation is an equivalence relation (reflexive, symmetric, and transitive).

Input: A list of books, and a list of pairs that are the same color
Expected output: The best upper bound on the number of shelves you will need

For example, if there are books u, v, w, and x, and you are given "u and v have the same color" and "v and w have the same color", the best upper bound is 2. (While u, v, and w are all the same color, we do not know for certain that x is a different color—we are just not given any information about x. The best upper bound assumes the worst case, so 2 colors total.)

Give an algorithm, prove its correctness, and describe the running time in terms of (whichever subset is appropriate): b (the number of books), p (the number of pairs listed), and s (the number of shelves required, i.e. your final answer).

4. Graphs in hiding: water jugs

This is a classic puzzle that you might have heard of before.

You have a 5-gallon jug and 3-gallon jug, which start out empty. Your goal is to have 4 gallons of water in the 5-gallon jug and 0 gallons of water in the 3-gallon jug. Unfortunately, you are only allowed the following operations:

- Completely fill any of your jugs.
- Pour one of your jugs into another, until the first jug is empty or the second is full.
- Empty out all the water in a jug.

(a) Describe a method to reach the goal. (No need to use any general algorithm yet, just solve the puzzle however you like.)

(b) Solve the following problem with a graph algorithm:

Input: Jug sizes a and b, with target amounts x and y, respectively.
Expected output: The minimum number of steps to reach the target amount, or "unreachable".

Warning: Graph algorithms are not the most efficient way to solve this problem. It turns out that this problem can be solved more efficiently (in logarithmic time) with the Euclidean algorithm and some fancier mathematical arguments. But today's goal is to practice graph algorithms.

The following problems will not be covered in section, but may be useful to think about.
We recommend trying them by yourself first. Solutions will be posted in the evening.

5. More algorithms proof practice

Consider the following algorithm for the following problem.

Input: A graph $G = (V, E)$ that is guaranteed to have out-degree at least 1 for every vertex.
Expected output: A directed cycle.

1: Let v_1 be an arbitrary vertex in V.
2: $i \leftarrow 1$
3: `cycleFound` $\leftarrow$ `false`
4: **while** not `cycleFound` **do**
5:     Let v_{i+1} be an arbitrary out-neighbor of v_i.
6:     **if** there exists $k \in \{1, \ldots, i\}$ such that $v_k = v_{i+1}$ **then**
7:         Remember this k.
8:         `cycleFound` $\leftarrow$ `true`
9:     **else**
10:         $i \leftarrow i + 1$
11: $C \leftarrow v_k, v_{k+1}, \ldots, v_{i+1}$
12: **return** C

(a) Follow the techniques and structure discussed earlier to prove that every line is valid (if necessary) and that the algorithm terminates.

(b) Read the following attempted "solution" for proving that the output is a directed cycle, assuming termination. Is it correct and sufficiently complete?

Proof. The algorithm constructs longer and longer paths in G. By an induction argument, line 5 ensures that $P = v_1, v_2, \ldots, v_i$ is always a path. In each iteration, there are two cases. If there exists $k \in \{1, \ldots, i\}$ such that $v_k = v_{i+1}$, then we would set `cycleFound` to `true` and exit the loop. In this case, because P is a path and $v_k = v_{i+1}$, the sequence $C = v_k, v_{k+1}, \ldots, v_{i+1}$ is a subpath of P (hence also a path) with endpoints that are the same, i.e. a cycle. On the other hand, if there is no $k \in \{1, \ldots, i\}$ such that $v_k = v_{i+1}$, then we incremented the loop counter and kept going. But we cannot go forever because the algorithm terminates as noted above, so eventually we end up in the first case, where we output correctly. $\square$

This proof is not very concise. It wastes a lot of writing to unnecessarily repeat the algorithm. To improve it, rewrite this proof using the technique discussed before, starting with what you want to prove, then unwrapping definitions. That is, start your proof with the line:

We will show that C is a cycle.

6. Big-$\mathcal{O}$ review

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.