Section 2: Solutions

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.

Solution:

1. $\log(\log(n))$
2. $\log(n)$
3. $\log(n^2)$ This function is $\Theta(\log(n))$. They differ only by the constant factor 2.
4. $(\log(n))^2$
5. $\sqrt{n}$
6. $2^{\log(n)}$ Note that this is just n.
7. $2^{\sqrt{n}}$
8. $3^{\sqrt{n}}$
9. $2^{n \log n}$

2. Write it Better: A Proof by Contradiction

Claim: For every simple graph G, if every node of G has degree at least 2, then G has a cycle.

(a) Prove the claim using proof by contradiction.

Solution:

An unclear proof. Suppose, for the sake of contradiction, there is a graph G such that every node of G has degree at least 2, but G has no cycle.
We will construct a simple path in G. Start at some node v_0, of G. Follow v_0 along an edge to find v_1. Now, since v_1 (like every other node) has degree at-least 2, there is another edge attached to v_1. Follow it to a vertex v_2. If v_2 is a vertex we have already visited (i.e., v_0), then we have found a cycle, a contradiction! Otherwise, from v_2, we may repeat the same argument. Continue from v_2 (also degree at-least-2) to a vertex v_3. If v_3 is a repeat (v_0 or v_1) then we have a contradiction! Otherwise continue finding $v_4, v_5, \ldots$. The graph is finite, so we cannot continue this process forever. Eventually we find a repeated vertex, which means we have a cycle, a contradiction! □

(b) Rewrite the proof, using the proof by contradiction with extremality technique.

Solution:

Proof. Suppose, for the sake of contradiction, there is a graph G such that every node of G has degree at least 2, but G has no cycle. Let $P = v_0, v_1, ..., v_k$ be a longest simple path in G.
Since v_0 has degree at-least 2, it must have another neighbor, w other than v_1. Since P is a longest simple path, w must be a repeat of a vertex among $v_1, ..., v_k$ (otherwise $w, v_0, v_1, ...v_k$ would be a longer simple path). Combining the edge (v_0, w) with P from v_0 to w gives a cycle. But G was a cyclic, that's a contradiction! □

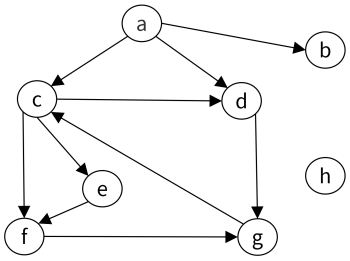
(c) There's another style yellow-flag in the version of this proof from part (a). We're proving an implication and our contradiction was the negation of one of the two things we supposed at the start. That usually means that proof by contrapositive would be clearer. Try writing this proof by contrapositive.

Solution:

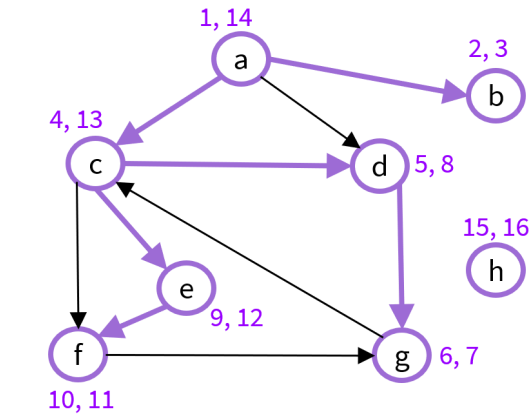
Proof. We argue by contrapositive. That is, we will show for all graphs, G, that if G does not have a cycle, then it has a node of degree at-most 1.
Let G be an arbitrary acyclic graph, and consider a maximal path $v_0, v_1, ..., v_k$. Since the graph is acyclic, v_0 cannot be adjacent to $v_2, ..., v_k$. Since the path is maximal, v_0 cannot be adjacent to any vertex other than the other v_i (as that would lead to a larger path). Thus v_0 is adjacent only to v_1, so it has degree at-most 1 as required. □

3. Mechanical: DFS

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



Solution:



Tree Edges: (a,b), (a,c), (c,d), (d,g), (c,e), (e,f)

Back Edge: (g,c)
Forward Edges: (a,d), (c,f)
Cross Edges: (f,g)

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?

Solution:

Fill the 5 gallon jug, pour into the 3 gallon jug until it is full (the jugs now contain 2 and 3 gallons respectively). Pour from the larger jug into the large bucket (it now has 2 gallons). Empty the 3 gallon jug, and repeat all these steps to get the desired 4 gallons.
Alternatively, Fill the 3 gallon jug, pour into the 5 gallon jug. Fill the 3 gallon jug again, pour until the 5 gallon jug is full (they now contain 5 gallons and 1 gallon respectively) . Pour the 1 gallon from the 3 gallon jug into the bucket. Refill the 3 gallon jug and pour into the bucket to bring the total to 4 gallons. There may be other solutions.

(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.

Solution:

Intuition
The "state" of the puzzle can be represented as the number of gallons in each of the jugs and the bucket. We encode the rules of the puzzle such that each possible step is an edge.
Algorithm
Let S be the set of all 11-tuples, where for the first 10 entries, the entry is an integer between 0 and c_i, and the final entry is an integer between 0 and t. There are $(C+1)^1 0 \cdot (t+1)$ such states.
We make a graph with a vertex for every element of S. And add an edge from u to v if and only if the states meet one of these conditions.
From a given state $(j_1, j_2, ..., b)$, you can move to another state $(j_1', j_2', .., j_{10}', b')$ if and only if one of the following hold:

- There is only one index, k, where the tuples differ, and $j_k' = 0$. (we emptied a jug or the bucket)
- There is only one index, k, ($k \leq 10$) where the tuples differ, and $j_k = c_k$. (we filled a jug)
- There are two indices, k, ℓ where the tuples differ:
  - $j_k' = 0$ or $j_\ell' = c_\ell$
  - $j_k + j_{ell} = j_k' + j_\ell'$.

Finally, we add a target vertex z, to the graph. Add an edge from every tuple where $b = t$ to z.

We then use [B/D]FS, starting from the all 0's tuple, and searching to see if z is reachable. If it is, we can return true (and predecessor edges will show the steps to take). If t is not reachable, then return false.
Correctness
Suppose our algorithm returns true. Then [B/D]FS found a walk from all 0's to z. By construction of the graph, each edge corresponds to a valid rule we can apply in the original puzzle. Since the only edges going into t are from states where $b = t$, the walk must reach such a state, thus the puzzle can be solved by doing the steps on the edges of that walk.
Conversely, suppose the puzzle is solvable. Then there is a series of steps that can be taken to put t gallons into the bucket. Each legal step has a corresponding edge in the graph to the next state by construction, so there is a path to a valid stopping state (i.e., a tuple where $b = t$), we added an edge from all such vertices to z, so there is a path from all 0's to z. [B/D]FS will discover this path, so we will return true.
Running Time
Let $n = (t+1)\prod(c_i+1)$. Note that this is the number of vertices in our graph. Each vertex has a constant number of edges leaving it (as you are performing one of three operations (dump, pour, fill) among a constant number of jugs. So the graph can be has $\mathcal{O}(n)$ edges and can be constructed in $O(n)$ time. Running [B/D]FS in a graph with $\mathcal{O}(n)$ vertices and edges takes $\mathcal{O}(n)$ time, so the overall running time is $\mathcal{O}(n)$.

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).

Solution:

We'll make a graph as follows: have a vertex for every book and an edge between u and v if they are listed as a "same color" pair. Since "same color" is transitive, if there is a walk from u to v then u and v are the same color. Thus any subset of a connected component can be on its own shelf (and we can't put books from separate connected components on the same shelf).
Our algorithm can run [B/D]FS to find connected components, and then iterate through the vertices to count the number. The graph will have b vertices and p edges, so the running time is $\mathcal{O}(b+p)$.