Problem Set 6

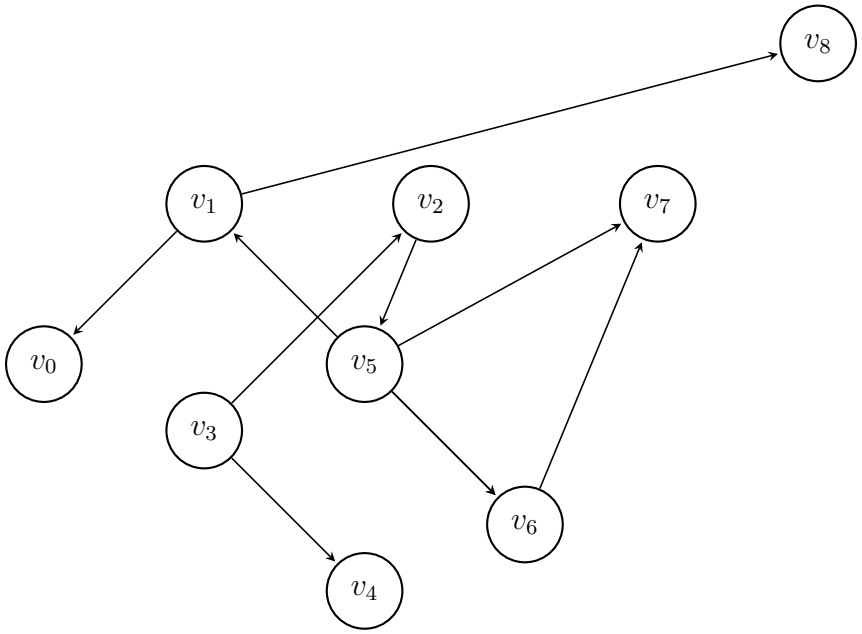
Instructions

All parts are due on April 8, 2021 at 10PM. Aim for concise solutions; convoluted and obtuse descriptions might receive low marks, even when they are correct. Solutions should be submitted on Gradescope, and any code should be submitted for automated checking on `https://alg.mit.edu`.

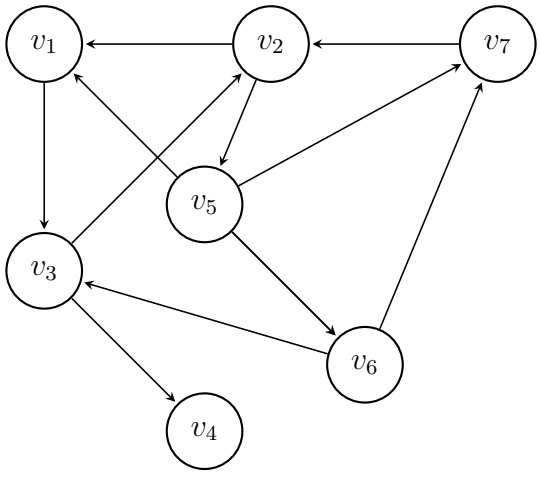
Please take note of the new collaboration policy as described in `https://canvas.mit.edu/courses/7477/discussion_topics/76837`. Unless explicitly stated otherwise, to get full credit you must prove the runtime and correctness of any algorithm you give.

Problem 6-1. [15 points] **Warm-up problems**

(1) [5 points] **Topological Sort** Use DFS to find a topological sort for the following graph. We explore nodes in increasing index order. Starting at the node v_3, for each iteration of DFS, indicate the `parent` and `order` lists maintained, and return the final topological sort that DFS finds.



(2) [10 points] **Edge Classification** Apply DFS to the following graph starting from vertex v_1. Explore nodes in increasing index order. For each node, state its `discover` and `finish` times. Here, we'll use the convention that once every node's neighbors are finished, the node is finished at the same iteration. Classify each edge as either `tree`, `backward`, `forward` or `cross`. Explain your rationale based on the `discover` and `finish` times of the nodes.



Problem 6-2. [20 points] **The Purples** The Purples are planning a fund-raiser. The up-and-coming party has a list of n "luminaries" that they want to invite. Each luminary is bringing a significant other to the party. The problem is that said luminaries, or their SO's, are of the jealous sort. Some luminaries, for reasons unknown, are jealous of some of the others. To make matters worse, some of the SO's are also jealous of some of the other SO's. Luckily, the planning committee knows these "issues" ahead of time and they have a list of the luminary problematic relationships as well as a list of the SO's. We write $s_i \to s_j$ to mean that SO number i is jealous of SO number j and $l_i \to l_j$ to mean luminary i is jealous of luminary j.

Why do the Purples need your help? Well, they are planning an arm-wrestling tournament where all luminaries will each arm-wrestle all the SO's. Each match can be written as (l_i, s_j) for some i, j, to mean that l_i is playing s_j. They ask you to come up with an ordering of the matches so that:

(i) If $l_i \to l_j$ and $s_i \to s_j$ for some i, j, then (l_i, s_i) is scheduled before (l_j, s_j)

(ii) If $l_i \to l_j$, then for every luminary s, (l_i, s) is scheduled before (l_j, s).

(iii) If $s_i \to s_j$, then for every luminary l, (l, s_i) is scheduled before (l, s_j).

Note that luminaries are only jealous of other luminaries and SO's are only jealous of other SO's. You can assume that all matches are played in a sequential order (so that no matches are played at the same time).

(1) [15 points] Help the Purples design an algorithm to create a suitable match schedule, or let them know that no such schedule is possible. Make sure to analyze its performance and correctness.

Hint: Try to show that a suitable schedule exists if and only if there is no cycle of jealousies.

(2) [5 points] Say we change the requirements for (i) to the following:

(i*) If $l_i \to l_j$ *or* $s_i \to s_j$ for some i, j, then (l_i, s_i) is scheduled before (l_j, s_j)

and we additionally drop requirements (ii) and (iii). Show that the hint from the previous part does not hold. That is, either give an example where there is a cycle among either the luminaries or the SOs (or both), and yet one can find a suitable schedule that obeys the new requirement (i*) or give an example where there is no cycle but there does not exist a suitable schedule.

Problem 6-3. [30 points] **Do you hear the people sing?** Arius and Menjolras are living in Paris and they've decided there is "One day more" until revolution. Ahead of time, they get a map of the city, which consists of streets connecting plazas throughout the city (a single street connects a single pair of plazas and a plaza may have an arbitrary number of streets leaving it).

Thematic music (optional)

Arius and Menjolras have just enough people and furniture to block a single street. Their goal is to be as disruptive as possible. They want to make sure that there exist at least two plazas, x and y such that people cannot go from x to y without passing through their barricades.

We can view this problem as a graph with nodes that correspond to plazas and edges between two nodes iff there is a street between the corresponding plazas. What they are looking for is a *bridge*, or an edge that can be removed from a graph to increase the number of connected components. In this problem, we will walk you through how to find bridges in a graph (as promised in Lecture 1).

We will work off of the articulation points algorithm given in the recitation notes. Let $v.in$ be the discovery time of vertex v. For a given run of DFS, we will calculate $v.low$ for each vertex v in the graph. We define $v.low$ such that $v.low := \min\{v.in, \min_x\{x.in\}\}$ where the inner minimum is taken over x such that wx is a back edge and w is a descendant of v. We know from lecture and recitation that these values can be calculated in $O(|E|)$ time. You can assume throughout this question that the graph is *undirected and connected*.

(1) [5 points] Prove that if uv is a bridge, then all paths from u to v must go through edge uv.

(2) [5 points] Prove that if uv is a bridge, then uv must be a tree edge in the DFS tree.

(3) [5 points] From the previous claim, we know that if uv is a bridge, it must be in the DFS tree. Thus, we can assume that v is a child of u in this tree. Prove that $u.in < v.low$.

Actually, as it turns out, the condition on edge uv in Part (3) is not only necessary for uv to be a bridge, but also sufficient. So, uv is a bridge *iff* uv is a tree edge (where v is u's child) and $u.in < v.low$. As a challenge, You may want to prove this on your own (we will provide a proof when we release solutions).

(4) [5 points] Give a linear time algorithm to find all bridges in the graph. Don't forget to analyze runtime and correctness. (You may use the above claims whether you proved them or not.)

(5) [10 points] Arius and Menjolras learn that Jinspector Avert plans to lead his troops from a particular plaza s to another plaza t, and they want to place their barricade such that they cannot make this journey without going through the barricade. In linear time (in the number of streets/plazas), help them find a list of all streets e such that blocking e (and only e) will achieve this. Remember to analyze runtime and correctness. *Hint:* For your correctness argument, you may find useful the fact proved in Part (1).

Problem 6-4. [35 points] **pip install** You want to run a python file you found online, but it uses a lot of bizarre imported libraries, some of which depend on other libraries. You'll need a correct order in which to pip install the libraries so that a library is only installed if every library it depends on is already installed. You have the list of L libraries that you want to install (labeled by the numbers 0 through $L-1$) and the adjacency list, `Dependencies`, showing which libraries each library is dependent upon. There are D dependencies.

(1) [8 points] Describe an $O(L+D)$ time algorithm to return a correct order in which to install the libraries, so they can be installed one at a time, or return `None` if no such order exists.

Example: With dependency list `[[1,2], [], [1], [1]]`, one possible order is `[1, 2, 0, 3]`.

(2) [12 points] You now have access to a multicore machine that can install an unlimited amount of libraries at the same time. That is to say, in one install session you can install multiple libraries in parallel, but each library must still already have all of its dependencies installed. Describe an $O(L+D)$ time algorithm to determine an order to install libraries that uses the least number of install sessions. You should return a list of sets indicating which libraries to install in each session.

Example: With dependency list `[[1,2], [], [1], [1]]`, the ordering is `[{1}, {2, 3}, {0}]`.

(3) [15 points] Code the function `determine_install_order` described in the previous part (but now returning number of sessions rather than an ordering of libraries) using the outline provided on `alg.mit.edu`. The portion of the outline given below shows all functionality that you must implement (denoted by YOUR CODE HERE).

```
def determine_install_order(dependencies):
    """
    Args:
        dependencies (list): adjacency list showing which libraries
                                          each library is dependent on

    Output:
      int: The minimum number of sessions needed to install all libraries
            or None if there is no valid install order
    """

    ###################
    # YOUR CODE HERE #
    ###################

    return None
```