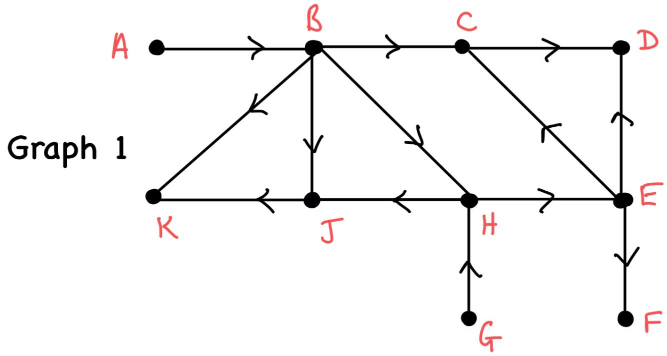
Practice Problem Set 12

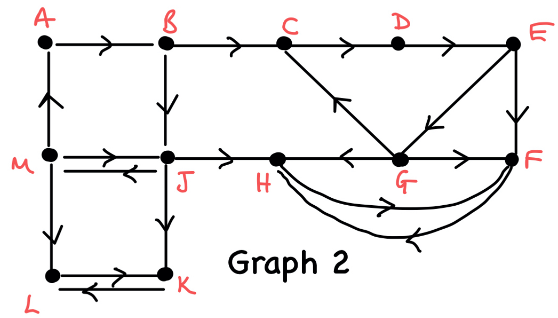
.

Problem 1

Run the topological sort algorithm on Graph 1 below, starting with vertex H. Run the topological sort algorithm again starting from vertex J, showing that you get a different topological sort. Process the neighbours in alphabetical order.

Run the strongly connected component algorithm on Graph 2, processing the vertices in alphabetical order. Show the resulting strongly connected components.





Problem 2

Answer the following with justification:

- Is it possible that two different BFS trees exist for the same graph G?
- Can a DFS tree on an undirected graph have any cross edges or forward edges?

Problem 3

Given an undirected connected graph G, write an algorithm that returns *true* if graph G contains a cycle, and false otherwise. Provide the pseudo-code for your algorithm.

Problem 4

The pseudo-code for DFS-visit(u) sets references to the *parent* of each node, but no where do we keep track of the *children* of a node during DFS. Update the pseudo-code for DFS-visit(u) so that it correctly assigns the children of node v in a list called $v.children$. You may assume that you can add to a list object using the method .add(x). Next, write the pseudo-code for an algorithm called PrintTree(u) that traverses the completed DFS tree rooted at u and prints out the keys in *pre-order*.

Problem 5

Rewrite the pseudocode for BFS so that it uses an adjacency *matrix* instead of an adjacency list to represent the edges of G. What is the runtime of this version? Explain the advantage of using the adjacency list over the adjacency matrix.

Problem 6

Update DFS from class so that it determines if the vertices of G can be colored in black and white such that no two vertices of the same color are adjacent. Assume G is a connected undirected graph. Your algorithm must return *true* if the coloring is possible, and *false* otherwise. Justify the runtime of $O(V + E)$.

Problem 7

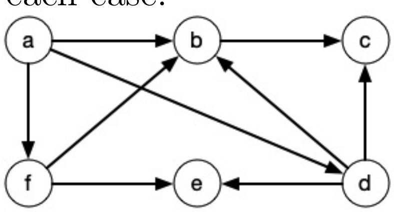
Repeat the above problem, using BFS instead of DFS. Justify the runtime of $O(V+E)$.

Problem 8

In a directed graph, the number of edges coming *out* of v is the out-degree and the number of edges coming *into* v is the in-degree. Explain why a DAG must have at least one vertex that has in-degree 0. Explain why a DAG must have at least one vertex that has out-degree 0. Use this fact to describe a **recursive** algorithm that outputs a topological ordering for a DAG G. Do not use DFS in your solution! Explain the runtime of your algorithm.

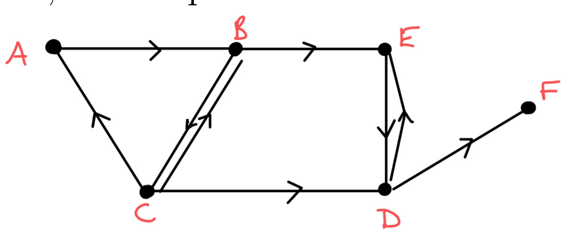
Problem 9

For the DAG below, determine the total number of topological sorts. Draw the sort in each case:



Problem 10

Let G be a directed, unweighted graph. Design an algorithm that prints out the vertices in each of the strongly connected components of G. For example, for the graph below, the output is shown on the left:




```
Component 1:  A, B, C
Component 2:  D, E
Component 3:  F
```

Problem 11

You are given as input a directed graph G and a source vertex s and a target vertex t. Suppose that each vertex of the graph has an attribute $v.color$ which is set to either black or white. Provide the pseudo-code for the following algorithms:

- Update BFS so that it returns *true* if there is a path from s to t and *false* otherwise.
- Update DFS so that it returns *true* if there is a path from s to t and *false* otherwise.

- Update BFS so that it returns *true* if there is path of alternately colored vertices from s to t, and $false$ otherwise.
- Update DFS so that it returns *true* if there is path of alternately colored vertices from s to t, and $false$ otherwise.

Problem 12

Given a directed graph G, write the pseudo-code for an algorithm that returns TRUE of G contains a directed cycle, and false otherwise. Explain the runtime of your algorithm.

Problem 13

Suppose G is an undirected graph with *no cycles*. Update the DFS-visit(u) algorithm from class so that it returns the length of the longest path in the DFS tree starting from node u.

Problem 14

While on vacation, your tour operator shows you a map of the n different tourist attractions that you can visit, starting from your hotel. The map includes a set of shuttle lines that connect pairs of tourist attractions. Not all pairs of tourist attractions are connected by a shuttle, and not all shuttles go in both directions. Suppose that you wish to venture out, but you're feeling lazy. You would like to find a trip that visits a set of tourist attractions, where your route **doesn't pass through the same attraction twice**, and comes back to your hotel. Of all the possibilities, you want the **shortest** route possible! For example, if you could go from $H \to A \to B \to H$, you visited 2 sites. Whereas if you go from $H \to A \to B \to C \to H$, you visited 3 sites, which is longer. Your job is to design an algorithm that outputs the **smallest** number of sites you can visit and successful get back to the hotel without visiting a site more than once. Call your algorithm LazyTourist(G), provide the pseudo-code for your algorithm, and justify the runtime of $O(V + E)$.