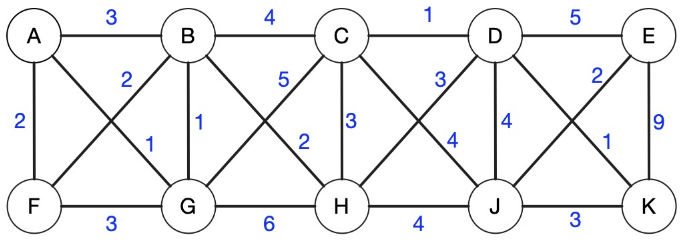
Assignment 5
CS-GY 6033 Spring 2025
Due Date: April 28th 11:55pm on Gradescope

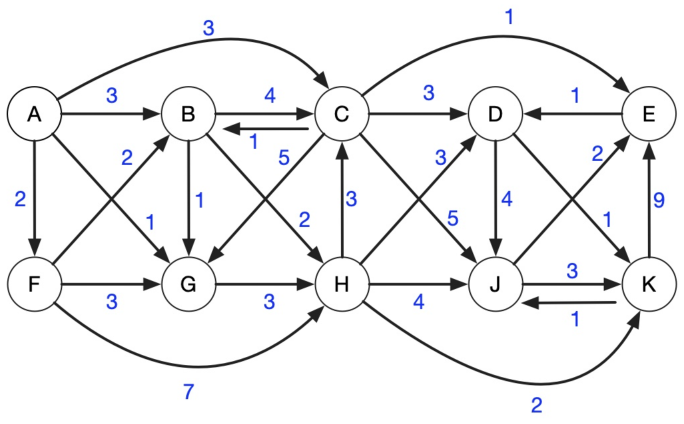
Question 1: Graph Traversals, Warm up

.

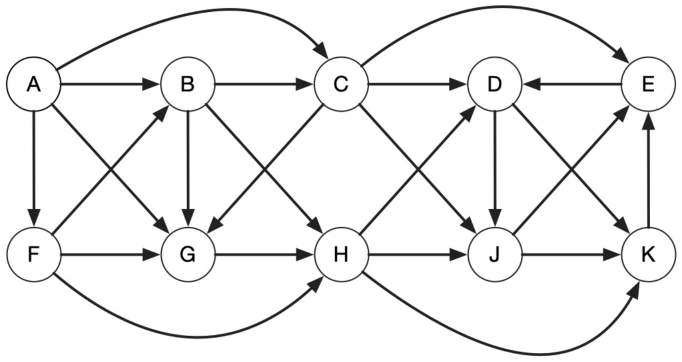
(a) 5 points Execute Prim's algorithm on the weighted graph below, starting at vertex A. You must show the state of the priority queue, and the parent references for each node, during the execution.



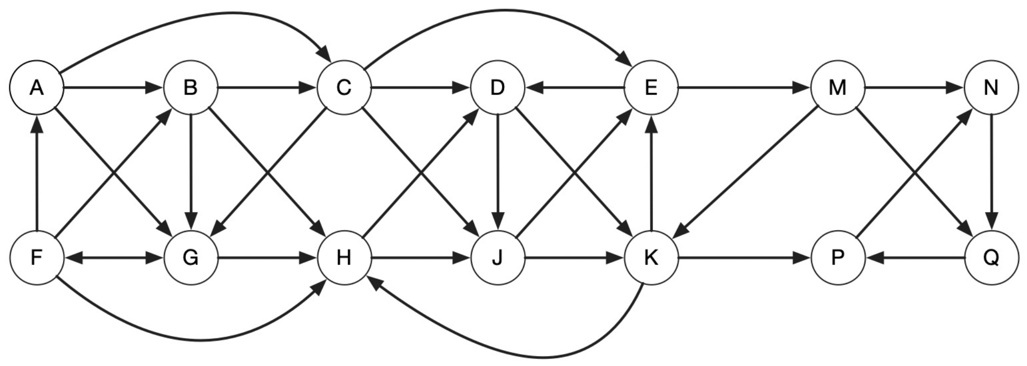
(b) 8 points Execute Dijkstra's algorithm on the directed graph shown below, starting at vertex A. You must show the state of the priority queue, and the parent references for each node, during the execution.



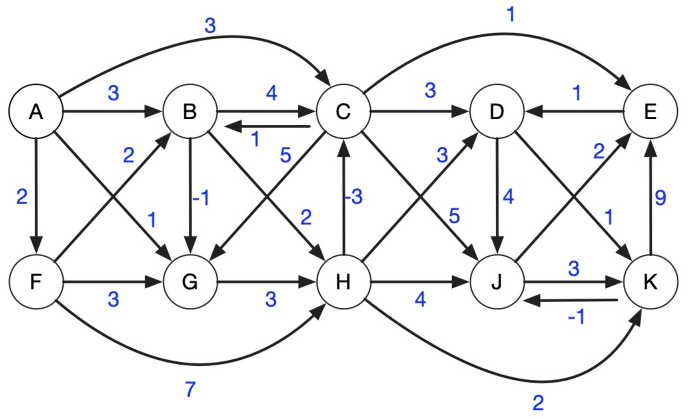
(c) 5 points Execute the algorithm from class for finding a topological sort on the directed acyclic graph shown below. Start DFS on vertex A. You must indicate the start/finish times for each vertex and draw the final top sort with edges shown.



(d) 8 points Execute the algorithm from class for finding the strongly connected components in the directed graph shown below. Start DFS from vertex A. You must show the output of DFS on G, and the output of DFS on G-transpose, and the final strongly-connected components.



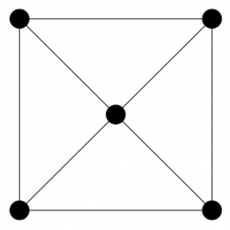
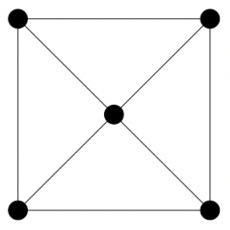
(e) 8 points Execute Bellman-Ford's algorithm on the following directed, weighted graph, using source vertex A. You must process the edges in lexicographic order (AB, AC, etc). You must show the values of $v.d$ and how they change as each edge is added. You must also show the current edges of the tree as the algorithm executes.



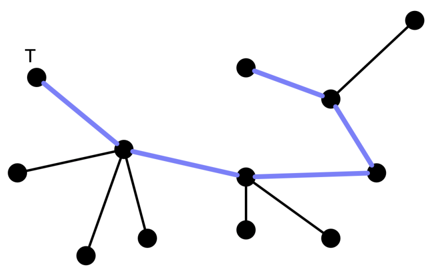
Question 2: Graph traversals: warm up.

.

(a) 4 points Consider the graph below. Your job is to place the following weights 1, 1, 2, 2, 3, 3, 4, 4 on the graph in **two different ways:** in the first way, ensure that only ONE MST exists, and in the second way, place the weights so that more than one MST exists. Justify each of your cases.

(b) 6 points Let G be a connected tree, which as you recall, is a graph without cycles. Note that it is not necessarily a binary tree. Let T be the root of the tree. Update DFS-visit from class so that it returns the length of the **longest simple path** in G from vertex T. Call your procedure LongPath(T). In the example below, the procedure returns 5.



Do not call any external procedures or use any external variables other than those defined in G. Justify that your algorithm has a runtime of $O(V+E)$.

(c) 4 points Let G be a directed, weighted graph with **no cycles**. A student tries to update the pseudo-code to Dijkstra's algorithm so that if find the **longest path** from vertex S to all other vertices in G. The update uses a MAX priority queue instead of a MIN priority queue, and extracts the Max instead of the Min.

```
For all vertices $v$ in $G$, set $v.d$ =-INF
Set $s.distance = 0$ and $s.visited = TRUE$
Add $s$ to the Max Priority Queue.
 while $Q \neq empty$
     u = Extract-max(Q)
     for each $v$ in $Adj[u]$
         if $v.d < u.d + w(u,v)$
             Increase-Key$(Q, v, u.d + w(u,v))$
             v.parent = u
```

Does this procedure correctly set the maximum path length from vertex S to all other vertices in G? Justify your answer.

(d) 6 points Suppose graph G is a directed, unweighted graph that is KNOWN to have exactly **three** strongly connected components. Your job is to determine how to add **exactly three edges** so that there is only ONE strongly connected component.

Your job: Design an algorithm that solves the problem and carefully specify each step. You may write things like "run DFS(G)" or "run DFS-visit(v)", but you can't simply state "find the SCCs", since we did not see a procedure with this name in class. The output of your procedure must be the PAIRS OF VERTICES between which you add the edges. Your procedure must clearly find the vertices involved in the new edges. Justify the overall runtime of $O(V+E)$.

Question 3: Applied Graph traversals

(a) 6 points Let G be a weighted, directed graph where each vertex represents a location, and each edge represents a directed edge between two locations. Suppose you are at location M and your friend is at location F. The goal is to somehow **get pizza at location** P and **beer at location** B and then both end up at location W **for a wedding**. The problem is, there are several ways to get there as fast as possible! You could pick up the pizza *and* beer and drive to the wedding and your friend drives directly to the party. Another option would be that you pick up the beer, and your friend picks up the pizza, and then you both go to the party. The goal is to minimise the time it takes for the last person to arrive at the party. For example, if it takes you 10 minutes to get beer and pizza and get to the party, but it takes 15 minutes for your friend to get to the party, the "overall" completion time was 15 minutes.

Design an algorithm that determines the fastest possible option for you both to arrive at the party, including the time to complete the two tasks. Carefully describe any algorithms used from class, their input and outputs, and runtime, and any attributes you indeed to use in the vertex objects. You may assume for simplicity that if one person makes both pickups, the pizza is picked up last, so that it doesn't get cold! Justify the runtime of $O(E \log V)$.

(b) 8 points Let G be an **undirected, unweighted** graph with vertex set V and edge set E. Suppose that each edge e has additional attribute $e.color$, which is either *red* or *green*. A particular vertex is labelled N for the north pole, and a particular vertex is labelled S for the south pole.The goal is to find a path of ALTERNATING EDGE COLORS that starts at N and finishes at S. Suppose for simplicity that the path **must start with a green edge**. Your job is the write the pseudo-code for an algorithm that solves this problem. Your solution should be based on DFS-visit from class, and may include an extra parameter for edge color. Do not include any extra data structures.

Hint: you will need TWO different v.visited attributes which keeps track of whether or not the veretx was visited by a green edge or by a red edge.

(c) 8 points Let G be an undirected, connected graph, where each vertex has an attribute for color: $x.color$ which is either black or white. Furthermore, the graph may contain cycles, however the cycles are all *verex-disjoint*, which means that no two cycles share a vertex.

Update the pseudo-code for DFS-visit so that it returns TRUE if there is a cycle of G that consists of **all-white vertices**. Call your algorithm FindWhiteCycle(u) which takes as input any node u of the connected graph G. **Ensure that you do not use any new external variables.**. Instead use recursion to RETURN the correct result.

(d) 10 points You're going on a canoe trip through the canal system built by the settlers of the 1800s! Your map consists of n campgrounds, where campground A is where you will start and campground B is where you must finish. Most campgrounds are connected by rivers, lakes and man-made canals, making it easy to traverse by water from one campground to another. The map clearly labels the distance by water between campgrounds. There are some campgrounds that unfortunately are not connected by water: you must *portage (ie: carry)* your canoe between these locations. Portaging is very tiring for those who haven't done it! So you decide to limit yourself to at most ONE portage during the journey.

Your Job:. Determine the minimum total distance you must travel by *water* in order to travel from A to B, with the restriction that you will carry out at most one portage. You do not need to consider the length of the portage. Assume your input is in the graph G with vertex set equal to the campgrounds and weighted edges equal to the connections between campgrounds. If the edge represents a portage, assume that is stored in the attribute $e.portage = true$. You may assume $|V| = n$ and $|E| = 2n$.

Provide the pseudo-code for your solution, and justify that it runs in time $O(n \log n)$.