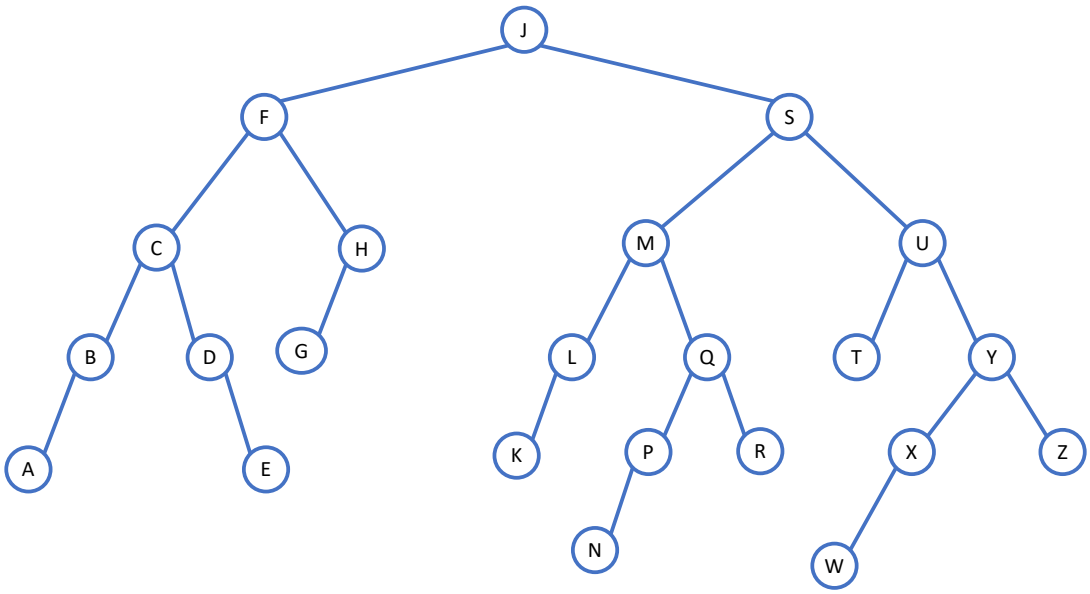
# Final Assessment

- Do not open the exam until you are directed to do so.
- Read **all** the instructions first.
- The exam contains XX multi-part problems You have 90 minutes to earn 100 points.
- The exam contains XX pages, including this one and XX pages of scratch paper.
- The exam is closed book. You may bring two double-sided sheet of A4 paper to the exam. (You may not bring any magnification equipment!) You may NOT use a calculator, your mobile phone, or any other electronic device.
- Enter your solutions on Coursemology. Do not enter any answers as comments.
- Read through the problems before starting. Do not spend too much time on any one problem.
- For the multiple choice questions, no partial credit will be given.
- For the open-ended questions, partial credit *may* be given, so show your work and explain your assumptions. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it.
- Good luck!

| **Problem #** | **Name** | **Possible Points** | **Achieved Points** |
|---|---|---|---|
| 1–13 | MCQ/MCR | 67 | |
| 14-17 | Short Answer | 33 | |
| **Total:** | | 100 | |

Name: _____ Matric. Num.: _____

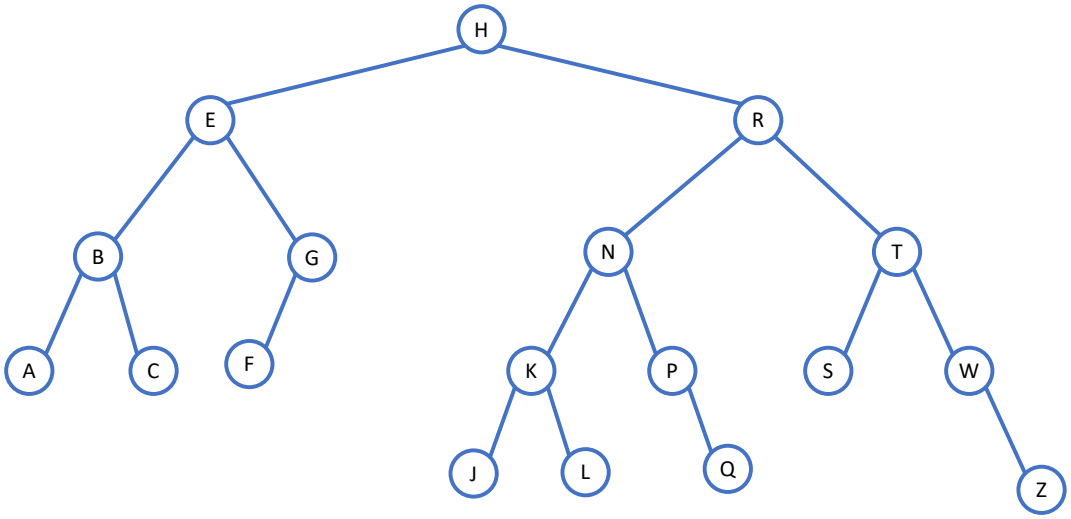
**Problem 1.**




The tree above is an AVL tree, where some node has just been inserted, but no rebalance operations have yet been executed. The keys are letters of the alphabet, where A is smallest and Z is largest. Which node(s) are out of balance?

**Solution:** Node $U$ is out of balance.

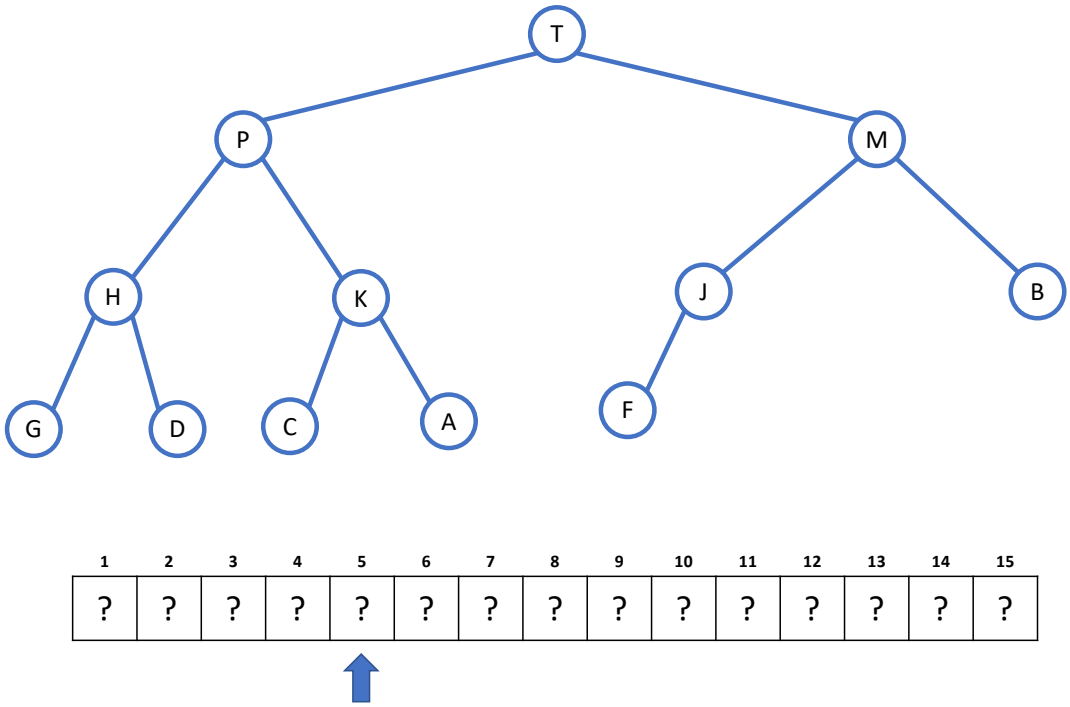
**Problem 2.**




The tree above is an AVL tree. Assume that we insert $M$ into the tree. Which rotation(s) occur? (Identify a rotation with the root of the subtree rotated. For example, a `right-rotate(E)` would move $E$ down and $B$ up.)

**Solution:** To balance the tree, perform two rotations: `right-rotate(R)`, `left-rotate(H)`.

**Problem 3.**

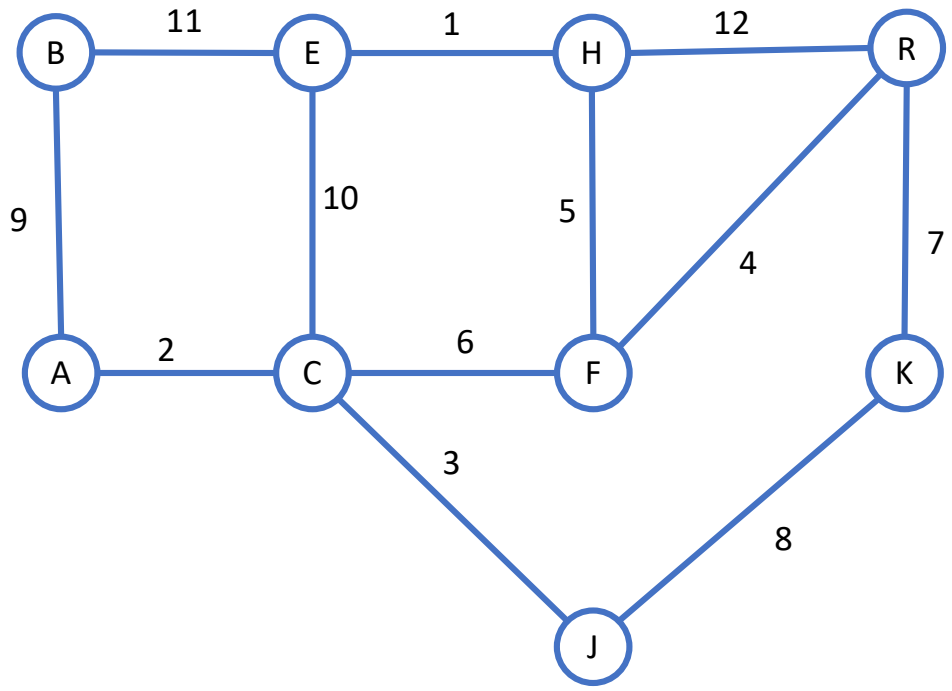



The tree above is max-heap, where the keys are letters of the alphabet. (A is the smallest and Z is the largest, and there is no letter O.) Below is the array in which the heap is being stored (with the keys replaced with question marks). Some of the cells in the array may be empty (which you can depict with a 0).

After an extract-max operation, what value is in the cell indicated by the arrow?

**Solution:** The cell indicated contains an F.

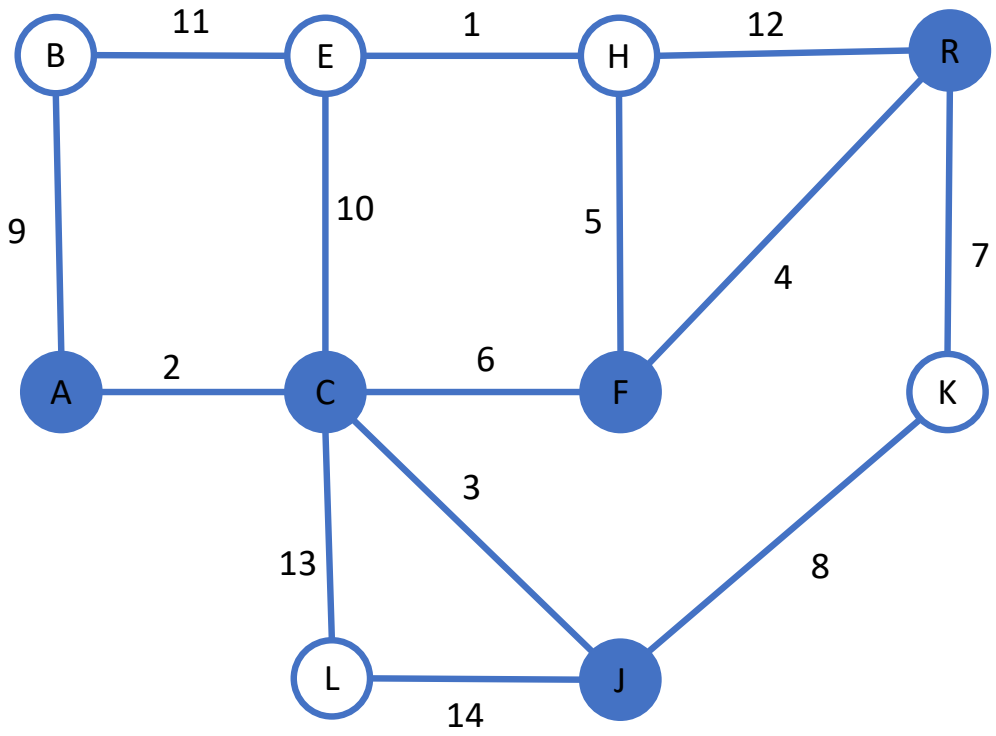
**Problem 4.**




The graph above is a connected, unweighted graph where every edge has a unique weight between 1 and 12. If you run Kruskal's Algorithm on this graph, which is the last edge that the algorithm adds to the MST? (Edges are identified by their weight.)

**Solution:** The edge $(A, B)$ with weight 9.

**Problem 5.**




The graph above is a connected, unweighted graph where every edge has a unique weight between 1 and 12. We have been running Prim's Algorithm on this graph, beginning at node *A*. To this point, we have performed five iterations of the main loop, extracting five items from the priority queue. Those are the nodes filled in on the diagram above (i.e., nodes A, C, F, J, and R). Which is the next node extracted from the priority queue?

**Solution:** The node *H*, because it is connected to the completed nodes by the minimum weight edge, i.e., the edge of weight 5.

**Problem 6.**

This is a problem about hash tables. Assume our hash table is 1-indexed, i.e., the first bucket is 1. There are 13 buckets in our hash table, and the last bucket is number 13. Consider the following hash function that maps elements to a table of size 13:

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 2 | 12 | 3 | 12 | 4 | 10 | 8 | 11 | 9 | 1 | 11 | 4 | 11 | 4 | 4 | 1 | 5 | 11 | 6 | 4 |

Consider the following sequence of items added to the hash table:

| F | R | O | G | S | Q | U | A | C | K |
|---|---|---|---|---|---|---|---|---|---|

If the hash table resolved collisions via chaining, then how many elements are there in the longest linked list?

**Solution:** The items map to:

| F | R | O | G | S | Q | U | A | C | K |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 5 | 4 | 10 | 11 | 1 | 4 | 6 | 12 | 1 |

There are three items in bucket 4.

**Problem 7.**

If the hash table resolves collisions via linear probing (i.e., a form of open addressing), then in what slot is the 'A' placed? (Use 1-based indexing where the first slot of the array is numbered 1.)

**Solution:** The items map to:

| F | R | O | G | S | Q | U | A | C | K |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 5 | 4 | 10 | 11 | 1 | 4 | 6 | 12 | 1 |

F goes in bucket 4 and R goes in bucket 5. O then goes in bucket 6. G, S, and Q each go in their own bucket. U then goes in bucket 7, so A goes in bucket 8. (Buckets 3, 9, and 13 are empty.)

**Problem 8.**

In your copious free time last week, perhaps you invented a new priority queue! Assume your priority queue supports each operation with the following worst-case performance, where $n$ is the number of items current in the priority queue:

- search: $O(1)$
- insert: $O(\log \log n)$
- delete: $O(1)$
- extractMin: $O(\log \log n)$
- decreaseKey: $O(\sqrt{n})$
- isEmpty: $O(1)$

(The priority queue only supports these operations, e.g., it does not support a peek operation or any other operation you might have heard of. And you may not modify Dijkstra's Algorithm to use different operations so that it runs faster.)

You are given a connected, directed, weighted graph $G$ with non-negative weights and a specified source. Graph $G$ has $n$ nodes and $m$ edges. What is the running time of Dijkstra's Algorithm on $G$ using this new priority queue? (There are no other changes made to Dijkstra's Algorithm.)

**Solution:** Dijkstra's Algorithm performs $n$ extractMin operations (i.e., one per node) and $m$ decreaseKey operations (i.e., one per edge). There are assorted other search and insert and isEmpty calls, but they are dominated by the above. (Specifically, there will also be $n$ isEmpty calls to check each time whether the priority queue is empty. During initialization, you have to insert the $n$ nodes into the priority queue. And you may have to do $m$ search operations to find the edges to decrease, depending on the precise interface.)
Thus the running time is $O(n \log \log n + m\sqrt{n})$.

**Problem 9.**

Let $G$ be a connected, undirected graph with positive weighted edges. Assume $G$ has $n$ nodes and $m$ edges. Assume that all the weights of $G$ are distinct. Let $T$ be a minimum spanning tree of $G$. Which of the following statements are always true:

a. If we add one edge $e$ to $G$ and $e$ is heavier than any edge in $T$, then $T$ is still an MST of the new graph.

b. If we add one edge $e = (u, v)$ to $G$ and $e$ is heavier than any edge adjacent to $u$ or $v$, then $T$ is still an MST of the new graph.

c. If we add one edge $e$ to $G$ and $e$ is lighter than any edge in $T$, then the MST of the new graph can be constructed by removing exactly one edge from $T$ and adding $e$ to $T$.

d. If we increase the weight of an edge $e$ in $G$ and $e$ is not in the MST, then $T$ is still an MST of $G$.

e. If $e$ is an edge in $G$ that is not in $T$, then there is always a cycle in $G$ where $e$ is the heaviest edge on the cycle.

f. If $e = (u, v)$ is an edge in $T$, then there is a cycle in $G$ where $e$ is the lightest edge on the cycle.

g. If $e = (u, v)$ is the heaviest edge in $G$ then it is never in the MST.

**Solution:** The following statements are true:

a. If we add one edge $e$ to $G$ and $e$ is heavier than any edge in $T$, then $T$ is still an MST of the new graph.

c. If we add one edge $e$ to $G$ and $e$ is lighter than any edge in $T$, then the MST of the new graph can be constructed by removing exactly one edge from $T$ and adding $e$ to $T$.

d. If we increase the weight of an edge $e$ in $G$ and $e$ is not in the MST, then $T$ is still an MST of $G$.

e. If $e$ is an edge in $G$ that is not in $T$, then there is always a cycle in $G$ where $e$ is the heaviest edge on the cycle.

The following statements are false:

b. If we add one edge $e = (u, v)$ to $G$ and $e$ is heavier than any edge adjacent to $u$ or $v$, then $T$ is still an MST of the new graph.

f. If $e = (u, v)$ is an edge in $T$, then there is a cycle in $G$ where $e$ is the lightest edge on the cycle.

g. If $e = (u, v)$ is the heaviest edge in $G$ then it is never in the MST.

**Problem 10.** Let $G$ be a connected, undirected graph where every edge has weight 0 or 1. There is a designated source $s$, and a shortest path tree from $s$ consists of all the shortest paths from $s$ to every other node in the graph.

Boaty McBoatface claims the following:

A. If $T$ is a minimum spanning tree of $G$, then $T$ is also a shortest path tree for source $s$ in graph $G$.

His friend Ferry McFerryface disagrees. He claims:

B. If $T$ is a shortest path tree for source $s$ in graph $G$, then $T$ is also a minimum spanning tree for $G$.

Dory McDoryface is not completely sure, but thinks she can show that:

C. There exists a tree $T$ that is both a shortest path tree for source $s$ in graph $G$ and also a minimum spanning tree of $G$.

Select *all* of the statements that are true (or none if they are all false):

- Statement A
- Statement B
- Statement C

**Solution:** Statement [C] is correct, but neither A nor B is correct.

Think about the graph $G_0$ where all the weight 1 edges are deleted. Let $k$ be the number of connected components in $G_0$. The cost of every minimum spanning tree of $G$ is exactly $k-1$. (Think about the way Kruskal's algorithm works!)

There is a shortest path tree with at most $k-1$ edges of weight 1, and that will be an MST. Assume that shortest path tree $T$ has at least $k$ edges of weight 1. Then by the pigeon-hole principle, there must be some connected component in $G_0$ that has two incoming edges of weight 1 in the shortest path tree. One of those two edges can always be deleted and replaced with weight zero edges within the connected component. This process can be continued until we have a shortest path tree with only $k-1$ edges of weight 1.

So we know that there is such a tree. But neither A nor B is true. Consider the graph that is a line on $n$ nodes, where the source is the first node on the line, and every node has an additional edge to the source. All edges have weight 1. Then the line itself is an MST, but is not a shortest path tree.

For the other direction, consider a graph with a source $s$, two nodes $A$ and $B$ connected to the source with edges of with 1, node $C$ connected to $A$ with an edge of weight 1, node $D$ connected to $B$ with an edge of weight 1, and $C$ and $D$ connected with an edge of weight 0. The MST must include the edge of weight 0, but the shortest path tree can just consist of all the edges of weight 1. (Nodes $C$ and $D$ are both at distance 2 from the source.)

**Problem 11.**

Assume you have a max-heap $H$ which is stored in an array $A$. It contains $n > 100$ unique items. Assume 1-based indexing, meaning the first slot of the array is numbered 1. All the statements below refer to the array when there are no operations in progress. Which of the following statements is true:

A. The item in array slot $\sqrt{n}$ may be the smallest item in the heap.

B. The item in array slot $3n/4$ may be the smallest item in the heap.

C. The item in array slot $3n/4$ may be the largest item in the heap.

D. At the end of an operation, array slot $n - 5$ **may** be empty.

*Announcement during exam: assume at most one operation has taken place. This means that the heap has at least* $n - 1$ *and at most* $n + 1$ *items in it.*

**Solution:** Statement 2 is true. Statement 1 is false, because the smallest item in a max-heap has to be at a leaf. Statement 3 is false because the largest item is at the root. Statement 4 is false because the first $n$ slots of the array must be fall.

**Problem 12.** How much memory is required to store a connected, directed, unweighted graph with $n$ nodes and $m$ edges as an adjacency list? Assume that we can store an array index and a vertex identifier in $O(1)$ space. Choose the tightest possible bound.

**Solution:** It takes $O(m) = O(n+m)$ space, since the array takes $O(n)$ space and the linked lists take $O(m)$ space. We know that $m > n$ because the graph is connected.

**Problem 13.** How much memory is required to store a connected, directed, unweighted graph with $n$ nodes and $m$ edges as an adjacency matrix? Assume that we can store an array index and a vertex identifier in $O(1)$ space. Choose the tightest possible bound.

**Solution:** It takes $O(n^2)$ space to store the adjacency matrix.

**Problem 14.**

Mr. Webster has a very big dictionary $D$ containing all the legal words in the English language. The dictionary is saved in a text file, one word per line.

He decides to store his entire very big dictionary in a Bloom Filter with two hash functions: for each word in the dictionary $D$, he performs an insert operation to place it in the Bloom Filter.

He then uses that Bloom Filter to implement a spell checker that looks up each word in your document in the Bloom Filter, and puts a little red squiggle if the word is not found. The dictionary contains $n$ words, and the Bloom Filter contains $m$ cells, and Webster chooses $m/n = 3$. Assume that the hash functions satisfy the Simple Uniform Hashing Assumption, i.e., each word is equally likely to hash to each cell. Which of the following statements are true (select all that apply):

A. Some words have red squiggles that are spelled correctly.

B. Some words are spelled incorrectly but have no little red squiggles.

C. The Bloom Filter (even though it is implemented well) uses more space than the original dictionary (because $m > n$).

D. The Bloom Filter (because it is implemented well) uses less space than the original dictionary.

E. If Mr. Webster loses his original dictionary, then he can reconstruct it efficiently from the Bloom Filter.

**Solution:** The correct statements are (B) and (D). Because of collisions, a misspelled word may collide with a real word and hence not receive a red squiggle. All correctly spelled words will show up in the table and hence not be squiggled. The Bloom Filter uses less space than the original dictionary because it only requires $3n$ bits, and the average English word has more than 3 bits. (And the originally dictionary cannot be constructed efficiently from the Bloom Filter because it contains so much less information: you cannot map back from a cell in the filter with a 1 to all the possible words that might map to that call.)

## Short Answer

### Problem 15. Relax Efficiently

The function $relax(u)$ is used in Bellman-Ford and Dijkstra to relax all the outgoing edges of $u$, i.e., if $est(z)$ is some distance estimate for $z$, and if $w(x, y)$ is the weight of an edge $(x, y)$, then the $relax(v)$ function checks, for every neighbor $u$ of $v$, whether $est(u) > est(v)+w(v, u)$, and if so, then it sets $est(u) = est(v) + w(v, u)$.

Your tutor claims that if $G$ is a weighted, connected, directed graph with no negative-weight cycles, then there exists a sequence of nodes $u_1, u_2, \ldots, u_n$ that includes each node exactly once; and if we begin with the source with estimate 0, every other node with estimate $\infty$, and we relax nodes exactly in the specified order, then when all the relaxations are done, all the distance estimates are correct.

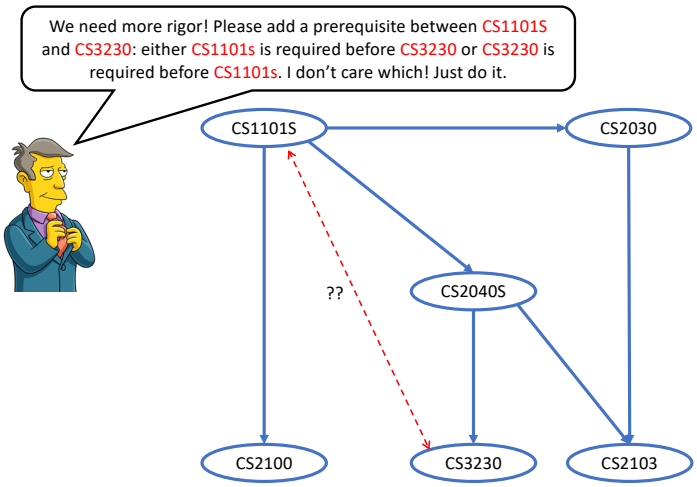
For example:


In this case, if you relax the nodes in order: $s, E, H$, then every node will have the correct distance estimate from the source at the end.

Is your tutor correct? If so, explain (in one short sentence) how you might efficiently construct such a sequence of nodes to relax. If not, explain why not (and/or give a counterexample). (You may use any algorithm we have seen in this class as a blackbox. Please do not spend time re-explaining how an existing algorithm works.)

**Solution:** Your tutor is correct. To find such a sequence, construct a shortest path tree (e.g., using Bellman-Ford) and construct the sequence by taking the nodes in BFS order.

## Problem 16. Adding Prerequisites




The curruculum of the School of Computing consists of a set of modules with prerequisites: you can only take module $x$ if you have already taken modules $a$, $b$, and $c$. We can represent this prerequisite structure as a connected, directed acyclic graph: let $G = (V, E)$ be a DAG where $V$ is the set of $n$ modules and $E$ is the set of $m$ edges representing prerequisites. (Assume $n > 10$, and that the graph is given as an adjacency list.)

After some thought, we have decided that the curriculum provides too much flexibility. We decide to add $k$ more prerequisites, in order to provide more guidance. The Vice Dean for Confusion comes up with a set $K$ of $k$ pairs $(u, v)$ with the idea that there should be a prerequisite either $(u, v)$ or $(v, u)$, but he does not care which way the prerequisite goes. It is important that after adding the edges in $K$, there are still no cycles in the new graph.

Your goal is to find an efficient algorithm that orients the edges in $K$ so that they can be added to $G$ while preserving acyclicity. The algorithm should take the DAG $G$ and the set $K$ and output an orientation for each edge in $K$.

Is it always possible to find a legal orientation for the edges in $K$? If so, explain why; your algorithm should always output an orientation. If not, explain why not; your algorithm should output FAIL if it is impossible.

Either way, describe your algorithm in **two** short sentences. (You may use any algorithm from class as a blackbox. Please do not spend time re-explaining how an existing algorithm works.) In one additional sentence, explain why your algorithm works and why it can or cannot always find a legal orientation.

**Solution:** Find a topological order of the graph $G$. To orient edge $(u, v)$, put $u$ before $v$ if $u$ precedes $v$ in the topological order, and add $(v, u)$ otherwise. This always works since the topological order remains correct so there can be no cycles.

**Problem 17. Graph Transformation**

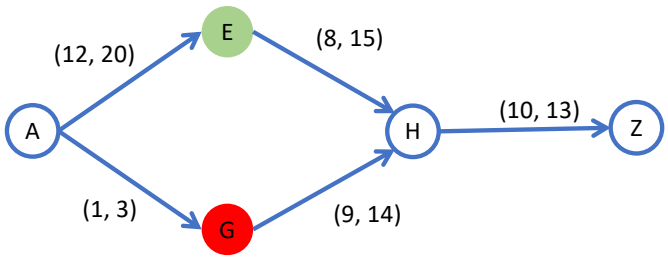
In our favorite SuperVirusFighter game, there are three types of locations: (i) Neutral, (ii) Risky (e.g., crowded places), and (iii) Safe (e.g., a handwashing station). To limit movement, there is a cost for moving from any one location to another. And the cost depends on whether you were more recently at a risky or a safe locaiton.

We will model this as a graph, where each node is a location (of one of the three types) and each edge represents a route connecting two locations. Each edge has two different costs: a safe cost and a risky cost. If you were more recently at a risky location than a safe location, then you pay the risky cost. If you were more recently at a safe location than a risky location, then you pay the safe cost. For example, if A is risky, B is safe, and C is neutral, then:

- the route $B \to A \to C \to D$ means you pay the safe cost when leaving $B$ and the risky cost when leaving $A$ and $C$;
- the route $A \to B \to C \to D$ means you pay the risky cost when leaving $A$ and the safe cost when leaving $B$ and $C$.

You start at a location $A$ which is safe, and you are trying to get to $Z$ which is also safe. Your goal is to find the cheapest cost path. You are given the map as a connected, directed graph $G = (V, E)$ where each node $V$ is a location that is labelled as safe or risky, and for each edge you are given two cost functions $c_1(e)$ and $c_2(e)$ indicating the safe and risky costs, respectively.

For example:




Here, node $E$ is safe and node $G$ is risky and the others are neutral. An edge with label $(x, y)$ has safe cost $x$ and risky cost $y$. There are two possible paths in this graph from $A$ to $Z$: (i) via the safe node $E$ at cost 30, or (ii) via the risky node $G$ at cost 28. Thus in this case, the cheapeast path is $A \to G \to H \to Z$.

Explain how to **transform** an arbitrary input graph $G$ into a new shortest path problem so that we can solve the problem using a blackbox shortest path algorithm. (You are not allowed to modify the shortest path algorithm; you can only build a new weighted, directed graph to run an existing shortest path algorithm on. Specify what each node and edge in the new graph represents. (Please do *not* just explain the example, but explain how to transform *any* input graph.)

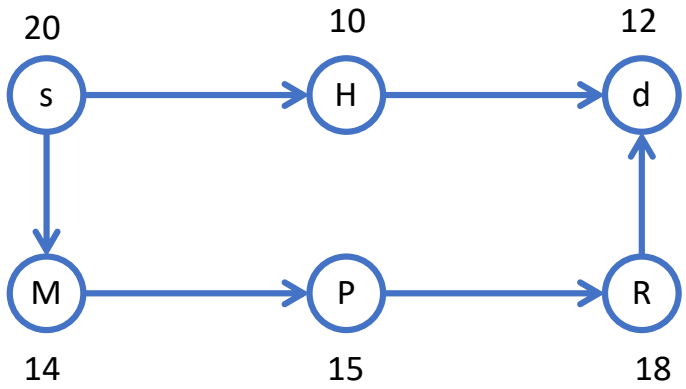
**Solution:** Create two copies of the original graph: a safe copy and a risky copy. For every $u \in V$, create $u_1$ (the safe version of $u$) and $u_2$ (the risky version of $u$).

- For every neutral node $u$, for every edge $(u, v)$ in the original graph, add two edges to the new graph: $(u_1, v_1)$ with weight $c_1(u, v)$, and $(u_2, v_2)$ with cost $c_2(u, v)$. (A neutral node does not change your state, whether safe or risky.)
- For every risky node $u$, for every edge $(u, v)$ in the original graph, add two edges to the new graph: $(u_1, v_2)$ with weight $c_1(u, v)$, and $(u_2, v_2)$ with cost $c_2(u, v)$. (After a risky node, you always end up in the risky graph.)
- For every safe node $u$, for every edge $(u, v)$ in the original graph, add two edges to the new graph: $(u_1, v_1)$ with weight $c_1(u, v)$, and $(u_2, v_1)$ with cost $c_2(u, v)$. (After a safe node, you always end up in the safe graph.)

**Problem 18. Slow Servers**

We can model the internet as a (directed) graph where each node is a server and each (directed) edge is an (unweighted) link. Each server $u$ has a speed, designated $speed(u)$, that is assigned to it. When you send a message long a path in the network, the limiting factor is the speed of the server. For a given path in the graph, we call the slowest server on that path the *bottleneck*, and its speed is the *bottleneck speed*.

For example:




Here, the path from $s \to H \to d$ has bottleneck speed 10, i.e., the slowest server on the path. On the other hand, the path from $s \to M \to P \to R \to d$ has bottleneck speed 12.

We want to find a path from the source $s$ to the destination $d$ that maximizes the bottleneck speed of the path. Give below the appropriate relax function $relax(u, v)$ so that Dijkstra's Algorithm works correctly to find this maximum bottleneck path from $s$ to $d$.

Your job is only to change the relax function. The other key changes to Dijkstra's Algorithm are: (i) for every node $u$ the estimate at $u$ is intialized to 0, except for the source which is initialized to $speed(s)$, and (ii) it uses a max-priority queue, instead of a min-priority queue. In terms of notation, use $est[u]$ to refer to the Dijkstra estimate at each node, and use $speed(u)$ to refer to the speed of node $u$.)

Give the revised relax function here:

**Solution:**

```
relax(u,v)
   if est[v] < min(est[u], speed(u), speed(v)) then
       est[v] = min(est[u], speed(u), speed(v))
```

*Other unnecessary explanatory notes:* Notice that the estimate at a node is always non-decreasing, and the estimate at a node $u$ never gets larger than the maximum bottleneck speed of a path from $s$ to $u$. More specifically, you can show (by induction) that the estimate at a node is always equal to the bottleneck speed of *some* path from the source $s$.

To see that it works correctly, we can reproduce the argument for Dijkstra's Algorithm. Throughout, we maintain three sets of nodes: $F$, that have been removed from the priority queue, $B$ that have had an adjacent edge relaxed at least once (and have an estimate $> 0$), and $U$ that have never had a neighboring edge relaxed (and have estimate 0). We will assume inductively that every node in $F$ has its estimate equal to its correct maximum bottleneck speed.

Consider the case where you take some node $u \in B$ out of the priority queue, move it to $F$. We want to show that it has its estimate correct. Assume not. Then there must be some path from $s$ with a higher bottleneck speed. Let $w$ be the last node on that path in $B$. (There must be at least one since the source $s$ is in $F$.) Since we chose to remove $u$ from the priority queue instead of $w$, that means that $est[w] \leq est[u]$. Moreover, as you extend the path from $w$ to $u$, we know that the bottleneck speed is non-increasing. Thus the bottleneck speed of that path cannot be better than $est[u]$.