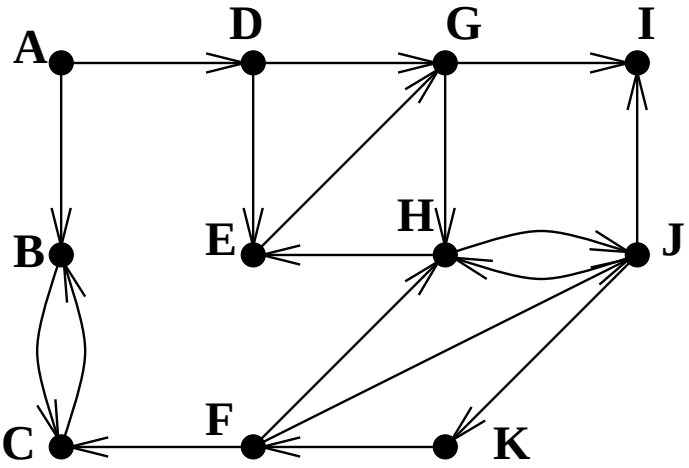
- **Answer to Version 1 of Question 1. (13 points)**
  - What are the strongly connected components of the directed graph shown below?



The strongly connected components are: $\{A\}, \{B, C\}, \{D\}, \{E, F, G, H, J, K\}$, *and* $\{I\}$.

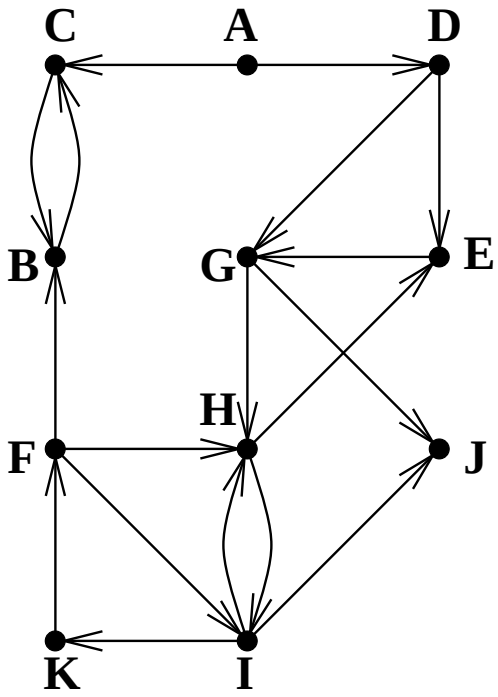
  - Which is the strongly connected component that will be discovered first by our algorithm (first do DFS on the reverse graph, then on the graph)? Which one will be discovered last? *As always, when DFS has a choice, it visits the alphabetically smallest vertex first.*

Here are the correct $(pre, post)$ *numbers of the vertices when DFS is run on the reverse graph:* $A = (1, 2), B = (3, 20), C = (4, 19), D = (10, 11), E = (12, 13), F = (5, 18), G = (9, 14), H = (8, 15), I = (21, 22), J = (7, 16), K = (6, 17)$. *Then, when DFS is run on the original graph ordering the vertices by decreasing posts shown above, the strongly connected component discovered first is* $\{I\}$ *and the strongly connected component discovered last is* $\{A\}$.

  - Label the vertices with their postorder numbers computed by DFS, i.e. the order in which they are popped from the stack.

The correct postorder numbers appear as the second coordinate in the following list (the first coordinate is the preorder number): $A = (1, 22), B = (2, 5), C = (3, 4), D = (6, 21), E = (7, 20), F = (14, 15), G = (8, 19), H = (9, 18), I = (11, 12), J = (10, 17), K = (13, 16)$. *Credit was also given for the order in which the vertices are popped from the stack, which is:* $(C, B, I, F, K, J, H, G, E, D, A)$.

- **Answer to Version 2 of Question 1. (13 points)**
  - What are the strongly connected components of the directed graph shown below?



    The strongly connected components are: $\{A\}, \{B, C\}, \{D\}, \{E, F, G, H, I, K\}$, *and* $\{J\}$

  - Which is the strongly connected component that will be discovered first by our algorithm (first do DFS on the reverse graph, then on the graph)? Which one will be discovered last? *As always, when DFS has a choice, it visits the alphabetically smallest vertex first.*

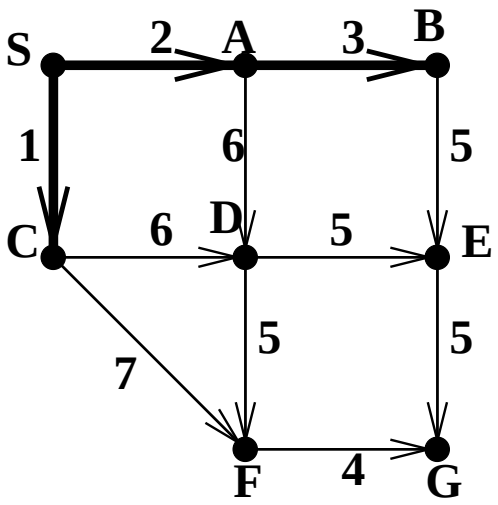
    Here are the correct $(pre, post)$ *numbers of the vertices when DFS is run on the reverse graph:* $A = (1, 2), B = (3, 20), C = (4, 5), D = (11, 12), E = (13, 14), F = (6, 19), G = (10, 15), H = (9, 16), I = (8, 17), J = (21, 22), K = (7, 18)$. *Then, when DFS is run on the original graph ordering the vertices by decreasing posts shown above, the strongly connected component discovered first is* $\{J\}$ *and the strongly connected component discovered last is* $\{A\}$.

  - Label the vertices with their postorder numbers computed by DFS, i.e. the order in which they are popped from the stack.

    The correct postorder numbers appear as the second coordinate in the following list (the first coordinate is the preorder number): $A = (1, 22), B = (3, 4), C = (2, 5), D = (6, 21), E = (7, 20), F = (14, 15), G = (8, 19), H = (9, 18), I = (10, 17), J = (11, 12), K = (13, 16)$. *Credit was also given for the order in which the vertices are popped from the stack, which is:* $(B, C, J, F, K, I, H, G, E, D, A)$.

- **Answer to Version 1 of Question 2. (13 points)** We are running one of these three algorithms on the graph below, where the algorithm has already "processed" the bold-face edges. (Ignore the directions on the edges for Prim's and Kruskal's algorithms.)

  - Prim's for the minimum spanning tree, starting from s.
  - Kruskal's for the minimum spanning tree.
  - Dijkstra's for shortest paths from s.

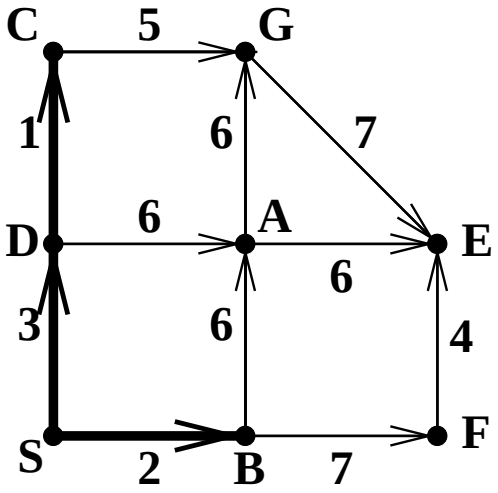


Which edge would be added next in Prim's algorithm? Which edge would be added next in Kruskal's algorithm? Which vertex would be marked next in Dijkstra's algorithm, i.e. deleted from the top of the heap? Which final edge would Dijkstra's algorithm choose as part of the shortest path to this vertex (i.e. which edge connects to this vertex as part of the shortest path from s)?

Edge (B, E) would be added next in Prim's algorithm. Edge (F, G) would be added next in Kruskal's algorithm. Vertex D would be marked next in Dijkstra's algorithm, and the final edge as part of the shortest path to D would be edge (C, D).

- **Answer to Version 2 of Question 2. (13 points)** We are running one of these three algorithms on the graph below, where the algorithm has already "processed" the bold-face edges. (Ignore the directions on the edges for Prim's and Kruskal's algorithms.)

  - Prim's for the minimum spanning tree, starting from s.
  - Kruskal's for the minimum spanning tree.
  - Dijkstra's for shortest paths from s.



Which edge would be added next in Prim's algorithm? Which edge would be added next in Kruskal's algorithm? Which vertex would be marked next in Dijkstra's algorithm, i.e. deleted from the top of the heap? Which final edge would Dijkstra's algorithm choose as part of the shortest path to this vertex (i.e. which edge connects to this vertex as part of the shortest path from s)?

Edge (C, G) would be added next in Prim's algorithm. Edge (F, E) would be added next in Kruskal's algorithm. Vertex A would be marked next in Dijkstra's algorithm, and the final edge as part of the shortest path to A would be edge (B, A).

- **Answer to Question 3. 20 points.** Give an efficient algorithm to find the *maximum spanning tree* in an undirected connected graph, i.e. a spanning tree the sum of whose edge weights is as *large* as possible. Your solution should consist of a modification of an algorithm you already know. Analyze the complexity of your algorithm, and justify its correctness (if you use an algorithm presented in class, you do not need to rederive its complexity analysis or correctness proof).

  We reduce this problem to finding a minimum spanning tree. Given a graph G let e_m be the edge with maximum weight w_m, i.e. $w_m = w(e_m) \geq w(e)$ for any edge e. We now define a graph G', which is just G with modified edge weights: The weight function w' corresponding to G' is $w'(e) := w_m - w(e)$. We claim a minimum spanning tree T' of this graph G' is a maximum spanning tree of G.
  To see this, consider $w'(T')$. Since this is a minimum spanning tree of G', $w'(T') \leq w'(T'')$ for all other spanning trees of G'. But since each spanning tree has $n-1$ edges, this implies $(n-1)w_m - w(T') \leq (n-1)w_m - w(T'')$, i.e. $w(T') \geq w(T'')$ for all spanning trees T'' of G, which proves our claim.

- **Answer to Question 4. 20 points.** You are given a connected undirected graph.
  - Show that it is possible to direct the edges so that every vertex has in-degree (the number of edges pointing to it) at least 1, if and only if the graph contains a cycle.

    ($\Rightarrow$) *Assume the graph contains no cycle. Then the graph has at most $n-1$ edges (and in this case, exactly $n-1$ since it is connected). Directing $n-1$ edges cannot point to n vertices.*

    Or: Assume each of n vertices has in-degree at least 1. Since each edge is only directed in one way, there are n edges. We know a graph with n edges must have a cycle (a tree can have at most $n-1$ edges).

    Or: Assume the graph contains no cycle. Then for every way of directing edges, the graph is a DAG. A DAG has a vertex with in-degree 0 (a source), so no matter which way we direct the edges some vertex has in-degree 0.

    Or: Assume each vertex has in-degree at least 1. Then from any vertex we can pick a parent. So pick some vertex, pick a parent, and repeat this process (of picking a parent). This process does not stop by our assumption, but there are a finite number of vertices, so there must be a cycle.

    ($\Leftarrow$) *The algorithm below directs the edges in the required way when there is a cycle.*
  - Give an efficient algorithm for directing the edges in the manner just described. The algorithm should return FALSE if it is impossible to do so. Analyze the complexity of your algorithm, and justify its correctness (if you use an algorithm presented in class, you do not need to rederive its complexity analysis or correctness proof).

    Algorithm: *Run DFS and find a back edge, call it (u,s). If no backedge exists, return FALSE. Run DFS again starting at s. Direct edges from parent to child.*

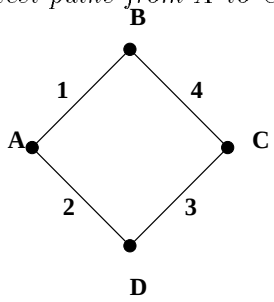
    Proof: *Since the graph is connected, it is clear that running DFS and pointing parent to child will give all the nodes except the root an in-degree of one (every node except the root has a parent). Since we chose the root to be in a cycle, there will be a backedge to it, so it will also have in-degree 1. If there was no backedge, there is no cycle, so return false.*

    Running Time: *2 DFS calls $= O(n+e)$.*

- **Answer to Question 5. 9 points.** There were two versions of this question, with the same true/false questions in different orders. We only give the answers once.

  True or false? No explanation required, except for partial credit. Each correct answer is worth 1 point, but 1 point will be *subtracted* for each wrong answer, so answer only if you are reasonably certain.

  1. $\log(\log^* n) = O(\log^*(\log n))$.
     True, since $\log^*(\log n) = \log^*(n) - 1$.
  2. The sum of the degrees of the vertices of a tree with n vertices is $2n - 2$.
     True, since the sum of the degrees is twice the number of edges, which is twice $n - 1$.
  3. A graph $G(V, E)$ with $|E| = |V| - 1$ is a tree.
     False, since G *might be disconnected and have cycles.*
  4. Let $G(V, E)$ be an undirected graph with k connected components. Then $|E| \geq |V| - k$.
     True, since if s_i *nodes form the* i*-th connected component, there must be at least* $s_i - 1$ *edges to connect them, and* $|E| \geq \sum_{i=1}^{k} s_i - 1 = \sum_{i=1}^{k} s_i - \sum_{i=1}^{k} 1 = |V| - k$.
  5. The sum of the in-degrees (the number of edges pointing to a vertex) of all vertices is equal to the sum of the out-degrees (the number of edges pointing away from a vertex) of all vertices in any directed graph.
     True, since each edge (u, v) *contributes one to the out-degree of* u *and to the in-degree of* v.
  6. The shortest path between two vertices is unique if all edge weights are distinct.
     False. Consider the shortest paths from A to C below.

  7. The path returned by Bellman-Ford at iteration $n - 2$ can be the shortest path.
     True. Bellman-Ford can be correct after even just one iteration: consider a "chain", just with edges $(i, i + 1)$ *for* $i = 1$ *to* $n - 1$.
  8. An undirected graph in which there is a unique path between every pair of vertices is a tree.
     True.
  9. You filled in your name and your TA's name on the first page of this exam.
     True, if you remembered your TA's name and filled it in, and False if you only filled in your section number or meeting time.