Dennis Kuzminer
CSCI-UA 310-001 PS3a

1.

1) Find in-degree of all nodes

A, B, C, D, E, F, G, H

[0, 0, 2, 1, 1, 2, 1, 1]

We will also keep track of the nodes with in-degree of 0 with a queue, and we will decrease the in-degree of all affected nodes. Begin:

Queue: A, B → Move A to top sort → top sort = A, in-degree of C = 1

Queue: B → Move B to top sort → Top sort = A, B, in-degree of C = 0

Queue: C → Move C to top sort → Top sort = A, B, C, in-degree of D, E = 0

Queue: D, E → Move D to top sort → A, B, C, D, in-degree of F = 1

E → E → A, B, C, D, E, indegree of F = 0

F → F → A, B, C, D, E, F, in-degree of G, H = 0

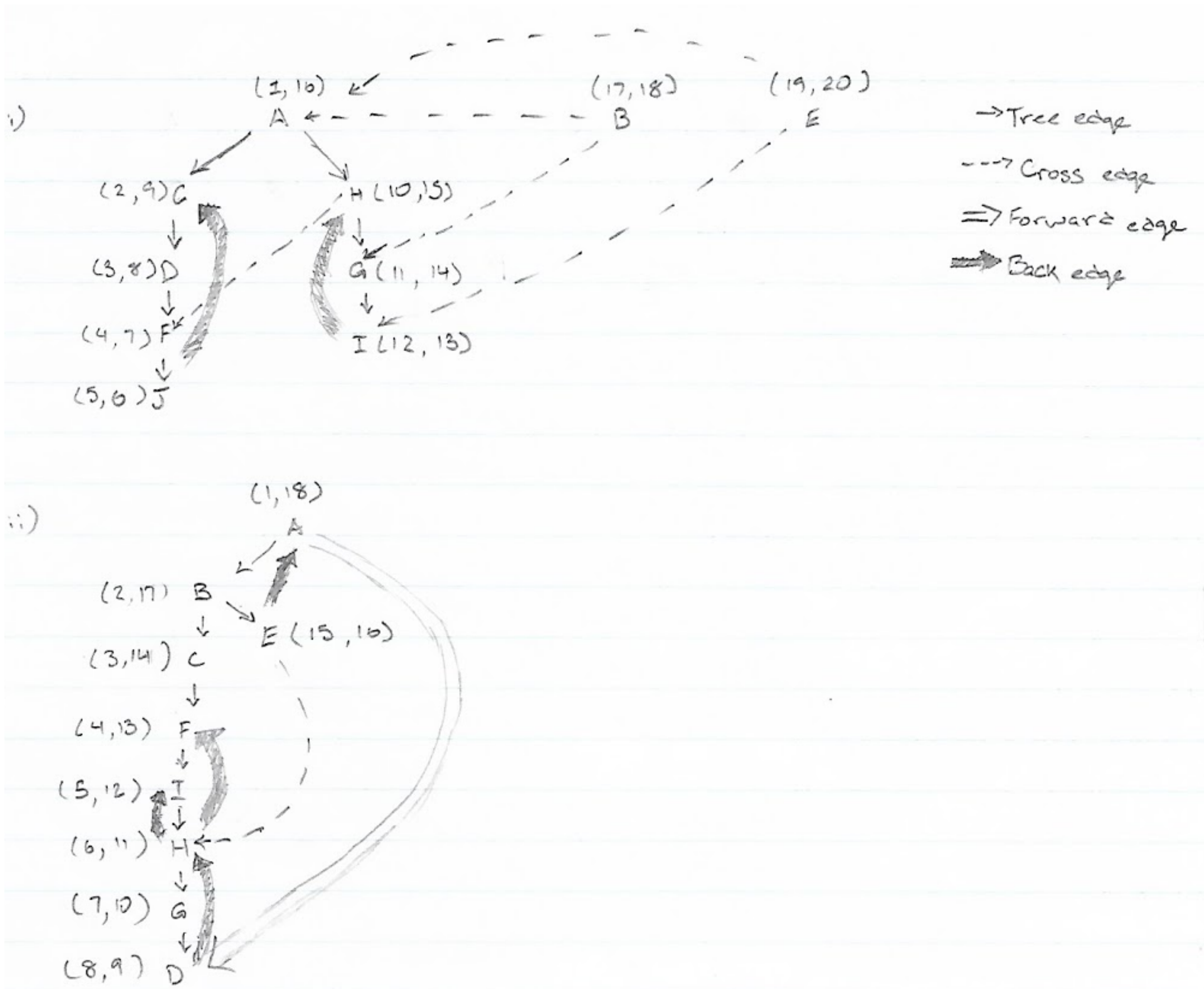
G, H → G → A, B, C, D, E, F, G

H → H → A, B, C, D, E, F, G, H

Resulting top sort

Dennis Kuzminer
CSCI-UA 310-001 PS3a

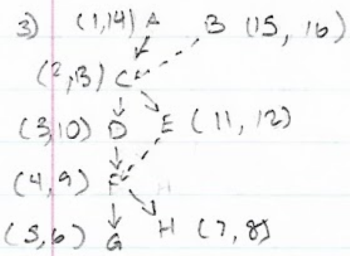
2.



Dennis Kuzminer
CSCI-UA 310-001 PS3a

3.

3)



→ Decreasing Finishing times a.k.a. last painted black to first painted black

↳ B, A, C, E, D, F, H, G

B A C E D F H G

Dennis Kuzminer
CSCI-UA 310-001 PS3a

4.

STEP 1:

4) This graph is actually G^T of G presentee in problem 2 ::
Therefore, DFS for G^T is:::

STEP 2:

(1,18) A
(2,17) B
(15,16) E
(3,14) C
(4,13) F
(5,12) I
(6,11) H
(7,10) G
(8,9) D

Top sort: Decreasing finishing time
A, B, E, C, F, I, H, G, D

STEP 3:

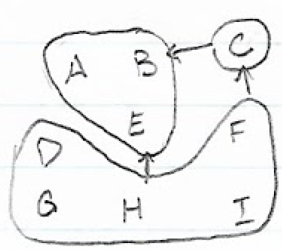
(1,6) A
(2,5) E
(3,4) B

(7,8) C

(9,18) F
(10,17) H
(11,12) I
D (13,16)
G (14,15)

$\hookrightarrow$ SCCs = (A, E, B), (C), (F, H, I, D, G)

G^{SCC}:



Dennis Kuzminer
CSCI-UA 310-001 PS3a

5. `
    a. For each C_i (i in [1, k])
        For each v in C_i
            Map[v]=i
    Run time of O(n)
    b. For each C_i (i in [1, k])
        For each v in C_i up until the number of edges in C_i or m
            Find some node x that is connected to some node v by an edge
            If Map[v] != Map[x]
                Add v → x into L
    Run time on O(m)
    c. Extra: Remove all dependencies that satify the condition
    u → v is a duplicate to x → y if v is a different node from y and Map[u] == Map[x] and Map[v] == Map[y]

Dennis Kuzminer
CSCI-UA 310-001 PS3a

6.

a. If r is a root, then the in-degree must be zero. This is because we have 2 presumptions here: 1) the root must be able to reach every V in the graph 2) G is acyclic.
Counterexample: Suppose there is an edge that connects v to r. This implies that there is a path from v to r. Similarly, we already know that r must have a path to v as the condition states that the root must be able to reach every V in the graph. Hence, we are creating a path from r to v and one from v to r. This is a cycle and would break our second condition that G is acyclic, meaning we cannot have a root with any incoming edges, only outgoing.

b. If r can reach any vertex in the graph, and r has an in-degree of 0, then no other point can have an in-degree of 0. This is because if some v has an in-degree of 0, then r will not be able to reach it (as there are no incoming edges). If r cannot reach v, then r and v cannot be roots as they have no connection, thus breaking the second condition.

c. If there is more than one node with an in-degree of 0, this will imply that r is not a root and that the graph is somehow disconnected. If the graph is connected and r has an in-degree of 0, this implies that r can reach every other node by some path, meaning that r must be a root, as stated by the first condition.

Dennis Kuzminer
CSCI-UA 310-001 PS3a

7.

a. It is important to note that a path in G is called k-alternating if it changes color ***at least*** k times (not necessarily exactly k).
   i. Run a topological sorting algorithm such that the runtime is O(|V| + |E|) (Kahn)
   ii. Run DFS starting on the first node with in-degree = 0
   During DFS: If a node in the path changes the color from the starting color, increment a counter variable associated with that particular tree branch. (Cross edges will already be accounted for)
   iii. Suppose i is the number of tree branches
   If i > 1
   For 0 to i-1
   m[v] = max(counter at branch i, counter at branch i+1)
   Return m[v] >= k //boolean
b. An arbitrary graph implies that there could a cycle(s) in the graph.
   i. Therefore, run the SCC algorithm
   ii. If a particular SCC contains 2 or more nodes with different colors, then return true. This is because we know that there a cycle in each SCC, and we can achieve at least k by just remaining in the loop. Once k is achieved, we continue to our path.
   iii. Else if all of the nodes in the SCC have the same color, check for a k-alternating path the same way we did in part a.

Dennis Kuzminer
CSCI-UA 310-001 PS3a

8.

a. First run reverse topological sort so that you can begin with the largest value
b. Run the following algorithm similar to the one presented in class with the coin problem
   P[v] is the max number off stones what can be on a path starting at v
   N[u] is the number of stones on a particular spot
   for i in reverse [0 . . n)
   u ← TopSort[i]
   m ← 0
   for each v ∈ Sccessor(u) do
   if P[v] > m then m ← min(P[v], c(e)), where e is the edge from v to its successor
   P[u] ← N[u] + m

c. However, c(e) must be considered. If c(e) is less than P[v] at a particular edge, then we must drop stones until we can cross the edge.

Dennis Kuzminer
CSCI-UA 310-001 PS3a

9.

a. Run a Depth-First Search algorithm on some node that need not be a root with runtime O(|V| + |E|). If there is more than one tree in the DFS forest, this must mean that there is a particular intersection that we cannot reach from some point on a different tree using only one-way streets.

b. Run a DFS with runtime O(|V| + |E|) starting on the node with the smallest in-degree (if possible)
If all nodes are connected by tree edges, then the root of the tree is privileged
If there is a back edge from a node to the root, then all of the nodes including and above that node will also be privileged

c. First, compute G^T
Then, call DFS(G^T), and order the nodes 1, . . . , n in order of decreasing finishing time (as in DFSTopSort)
Lastly, call DFS(G) — but in the top-level loop, process in the order 1, . . . , n
The result will be all the safe spaces or Strongly Connected Components
This will run in time O(|V| + |E|).