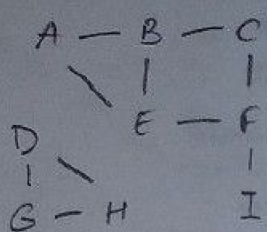
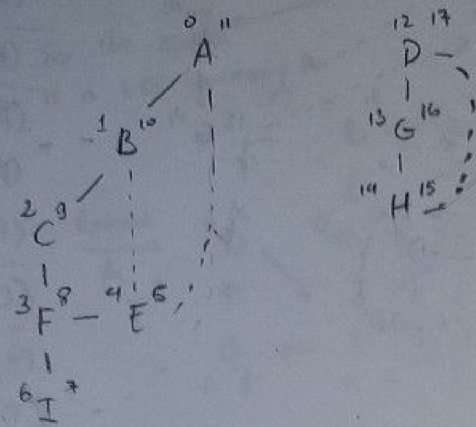
CS 216 - HW5

Quan Nguyen

3.1

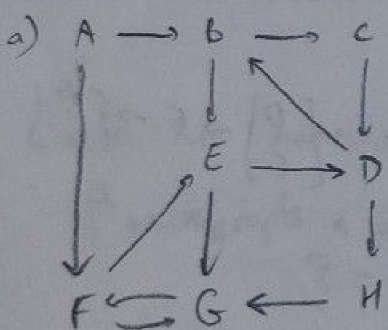


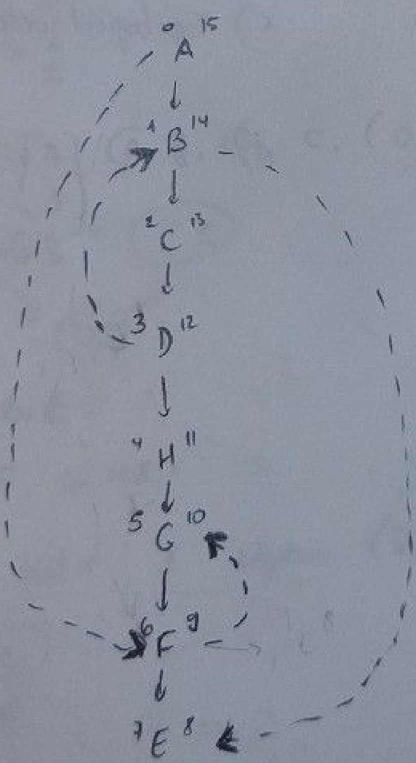


Tree edge	Back Edge
(A, B)	(E, B),
(B, C)	(E, A),
(C, F)	
(F, E)	
(F, I)	
(D, G)	
(D, H)	
(G, H)	

3.2

a)

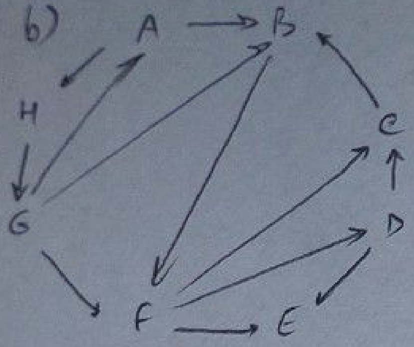


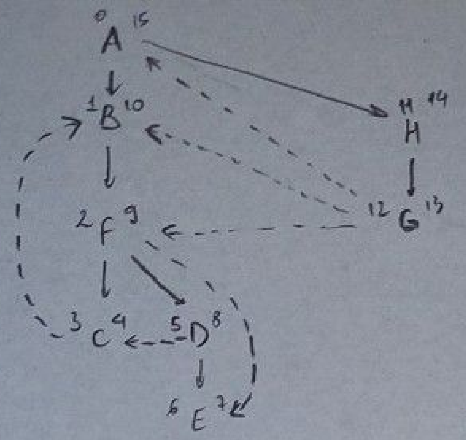


All edges are tree edges

3.2

b)





(A, H) : tree
(A, B) : tree

(B, F) : tree

(F, C) : tree
(F, D) : tree
(F, E) forward

(C, B) : back

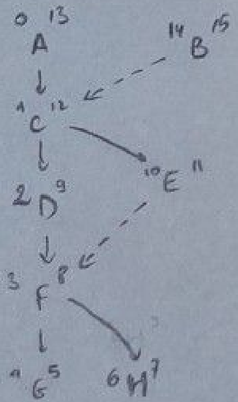
(D, C) : cross
(D, E) : tree

(H, G) : tree

(G, A) : back
(G, B) : cross
(G, F) cross

3.3

a)



b) Sources : A, B

Sinks : G, H

c) Topological order : A, B, C, D, E, F, G, H

d) 8 : $\begin{bmatrix} A \\ B \end{bmatrix} \to C \to \begin{bmatrix} D \\ E \end{bmatrix} \to F \to \begin{bmatrix} G \\ H \end{bmatrix}$

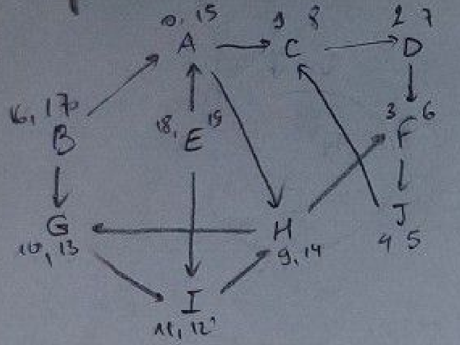
2 choices × 2 × 2

= 8

3.4 i)

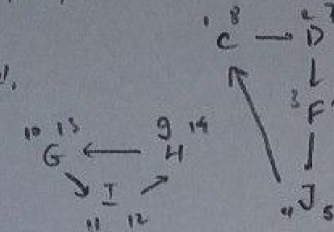
SCC: B, E, A

Graph G



Remove
B, E, A

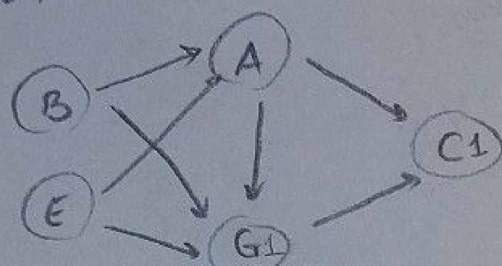
G'



⇒ SCC: (C, D, F, J)
and (G, H, I)

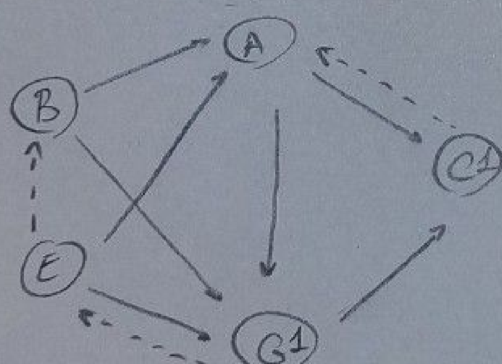
Let's call
SCC group C1 and G1

c) Metagraph:



b) ⇒ Source: B, E
Sink: (C, D, F, J)

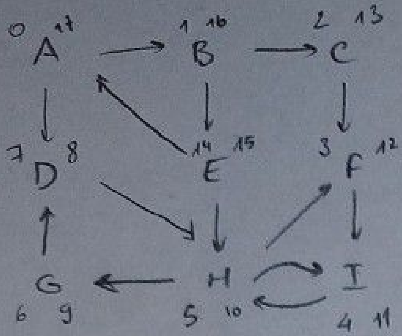
d)



I expect we need 3 more edges.
(C1, A) → A, G1, C become a SCC
then (G1, E)
(E, B) } → A, G1, E1, B will become a SCC

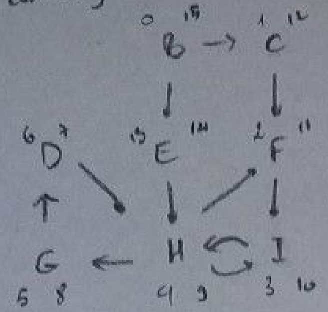
3.4. ii)

Graph G:

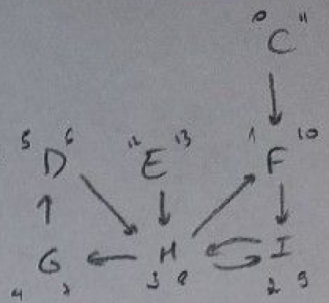


So the source is a SCC containing A

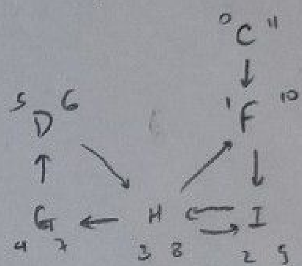
Remove A



Remove B



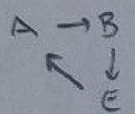
Remove E



Remove C

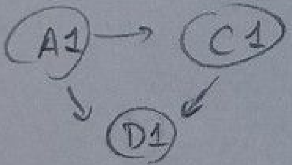
D, F, G, H, I is a SCC ⇐

Among A, B, C, E, we have A, B, E is another SCC:



a) So we have 3 SCCs: (A, B, E), C, (D, F, G, H, I) (call SCC A1, C1, D1)

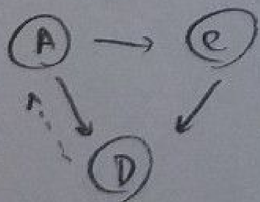
c) Metagraph:



b) So source: (A, B, E)

sink: (D, F, G, H, I)

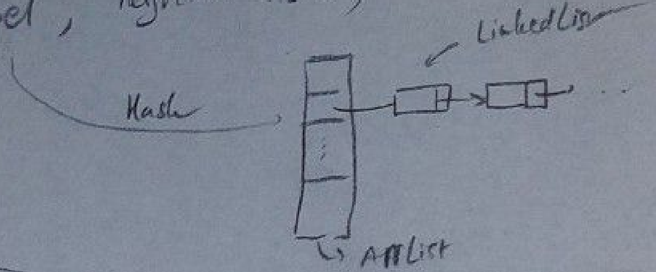
d) We need to add 1 edge (D1, A1)



3.5

Data structure:

(label , neighbor lists)



keyList = oldGraph.keys()

// Add key into newGraph

for each key in keyList: $\longrightarrow O(|V|)$

    newGraph.addVertex(key)

// Add edge

For each key in keyList: $\longrightarrow O(|V|)$

    LinkedList neighList = oldGraph.ArrList.get(key)

    For each neigh in neighList: $\longrightarrow O(|E|)$

        newGraph.addEdge(neigh, key)

$$\Sigma = O(2|V| + |E|) = O(|V| + |E|)$$

3.10

```
Explore (u)
    Stack.push(u)

    while ( stack not Empty):
        curr = stack.peek()
        if (curr not visited):
            previsit (curr)
            visit (curr) = true
            if (curr 's neighborList EMPTY):   // reach the end of graph
                post visit (curr)
                Stack.pop()
            else:
                stack.push (curr's neighbors)   // add all neighbors of currNode
        else:                 // after visit all neigbors, curr is visited means
            post visit (curr)     that we explored all subtree starting with u,
            Stack.pop()           so remove u from stack
```

3.11. Given edge $e = (u, v)$

```
boolean method (u, v))
    visit(u) = true

    for each neighbor in (u's neighbors).
        if (neighbor == v)
            return True;          // stop if find v in u's neighbors, which
                                  //   means edge e : u -> v exists
        else if (neighbor not visit)
            method (neighbor, v)

    return false
```