3.1

A — B — C
D — E — F
G — H — I

(graph with DFS numbering around B, C, F, E, I, A and D, G, H)

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) A → B → C
E → D
F ⇆ G ← H

(directed graph with DFS numbering)

All edges are tree edges

3.2

b)
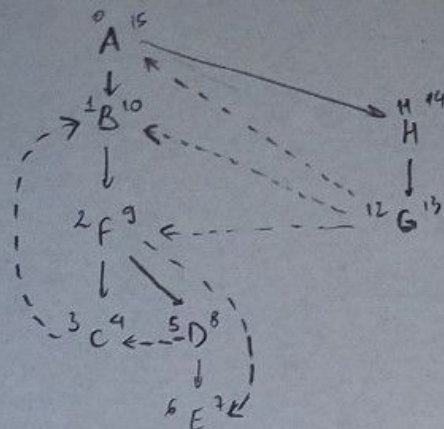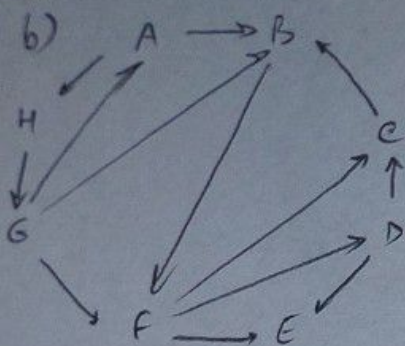
(A, H) : tree
(A, B) : tree
___
(B, F) : tree
___
(F, C) : tree
(F, D) : tree
(F, E) forward
___
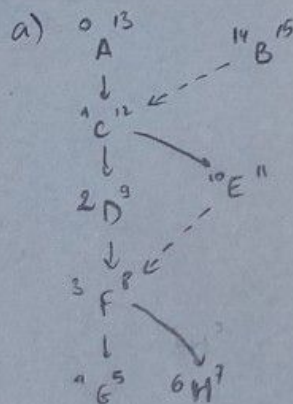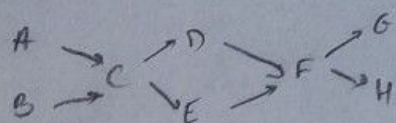(C, B) : back
___
(D, C) : cross
(D, E) : tree
___
(H, G) : tree
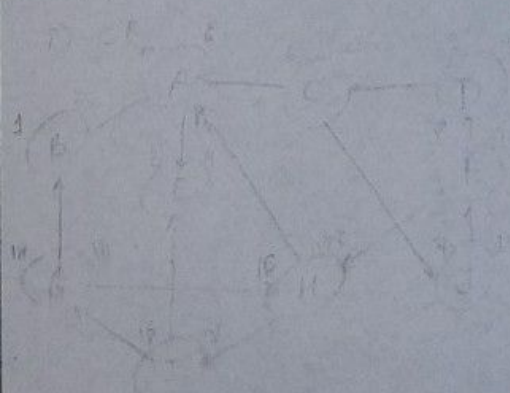___
(G, A) : back
(G, B) : cross
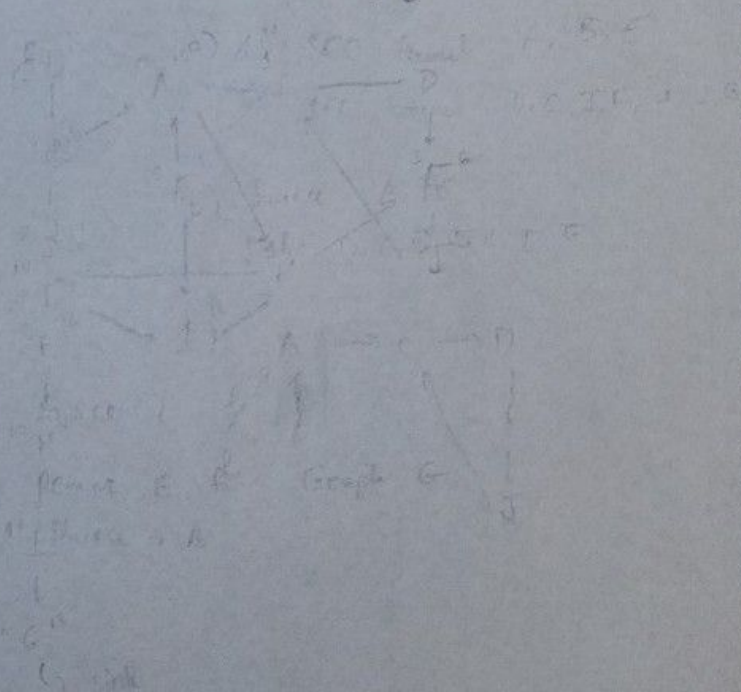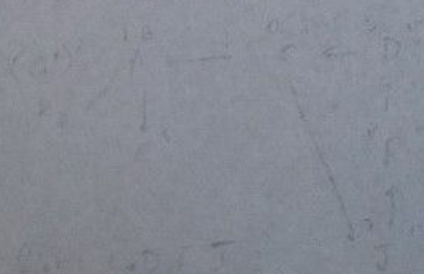(G, F) cross

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 x 2   x   2

                = 8

3.4 i)

Graph G
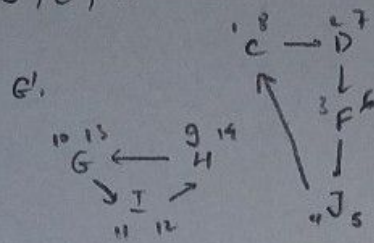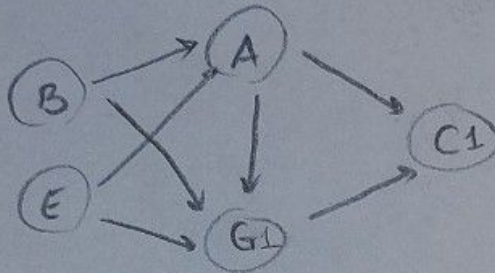
SCC: B, E, A

G'.
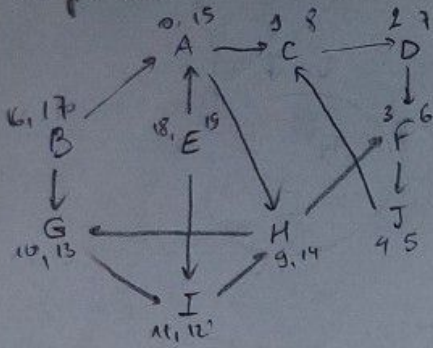
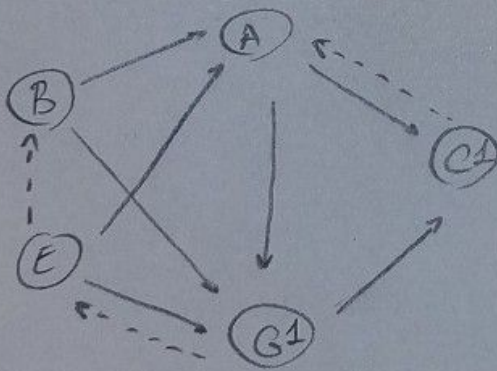Remove
B, E, A

⇒ 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) } → A, G1, E1, B will
  (E, B)   }   become a SCC

3.4. ii)

Graph G :
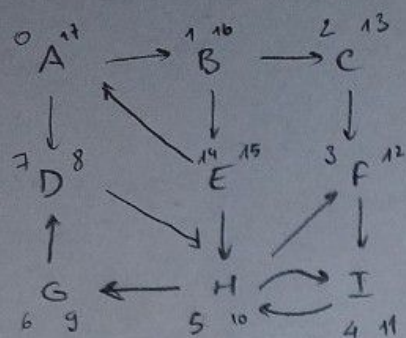
So the source
is a SCC containing A
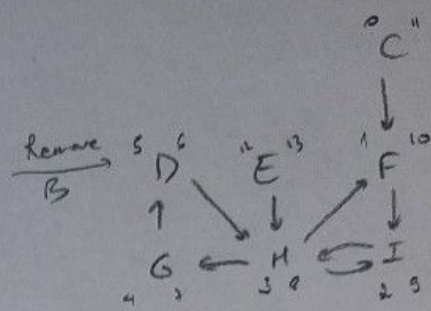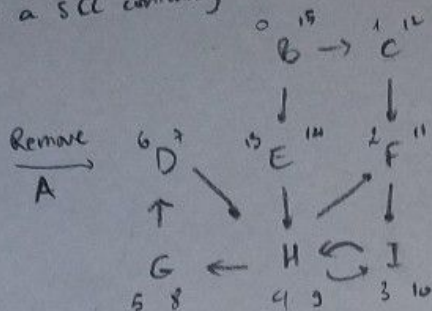
D, F, G,H,I
is a SCC ⇐

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

A → B
↖ ↓
  E

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

c) Metagraph: (A1) → (C1)
                  ↘   ↗
                   (D1)
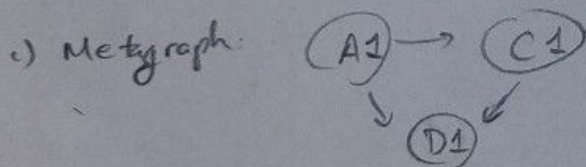
b) So source : (A, B, E)
   sink : (D, F, G, H, I)

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

(A) → (C)
 ↖ ↘   ↙
    (D)

3.5

Data structure:
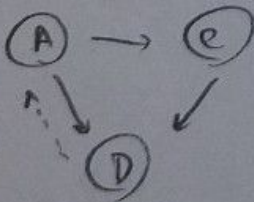
(label , neighbor lists)
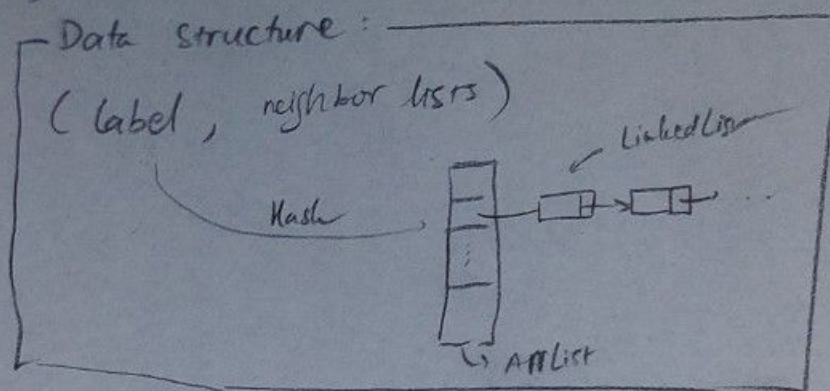
keyList = oldGraph.keys()

// Add key into newGraph
for each key in keyList: $\longrightarrow$ $O(|V|)$
   newGraph.addVertex(key)
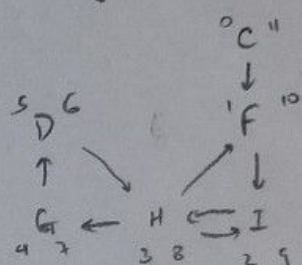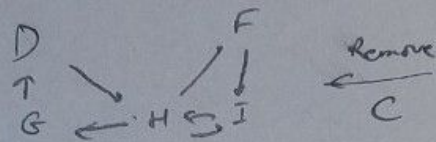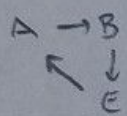

// Add edge

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
        else if (neighbor not visit)     means edge e : u -> v exists
            method (neighbor, v)


    return false
```