DAA HW 5

Pranav Tushar Pradhan N18401944 pp3051@nyu.edu

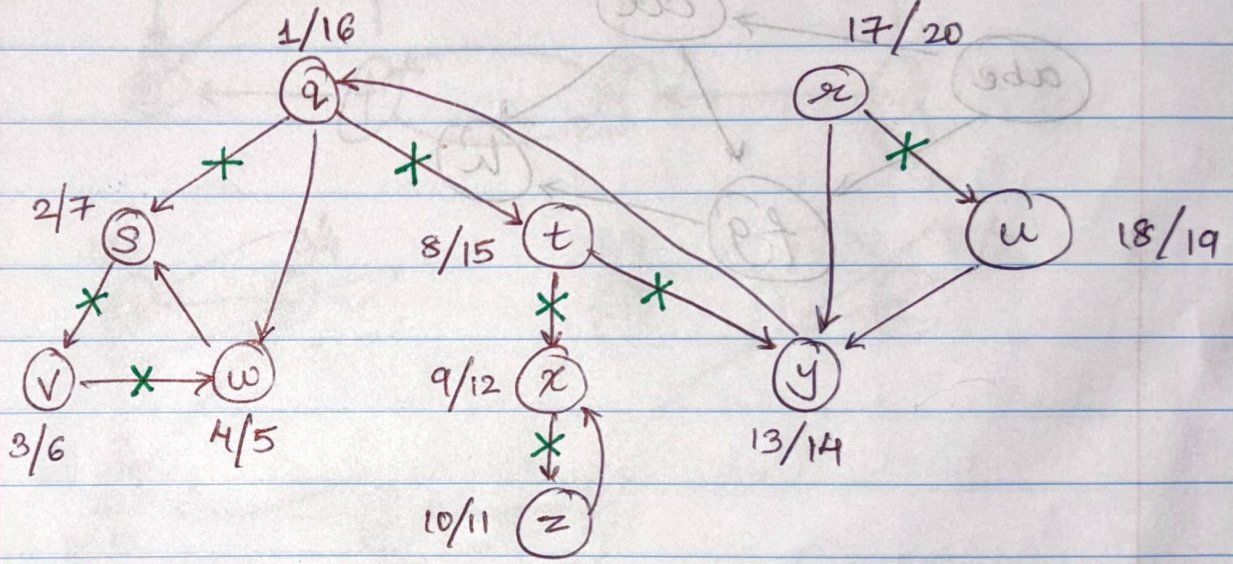
Udit Milind Gavasane N16545381 umg215@nyu.edu

Rohan Mahesh Gore N19332535 rmg9725@nyu.edu

Saavy Singh N16140420 ss19170@nyu.edu

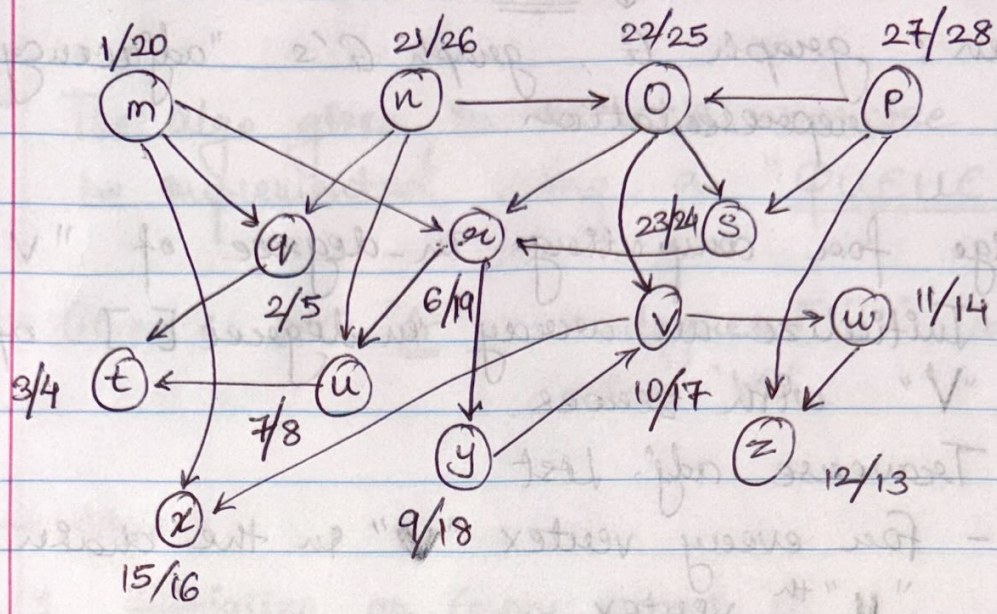
Sashank Badri Narayan N14628839 sb10192@nyu.edu

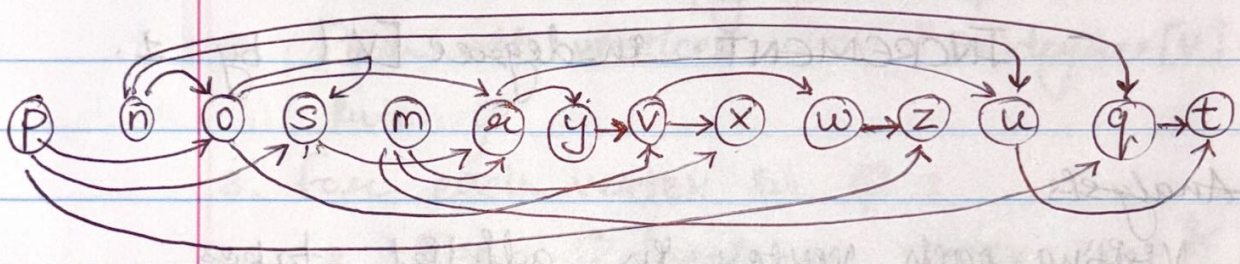
Q.1.
a.



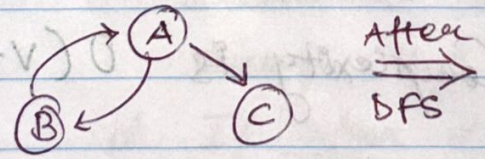
Q.1.

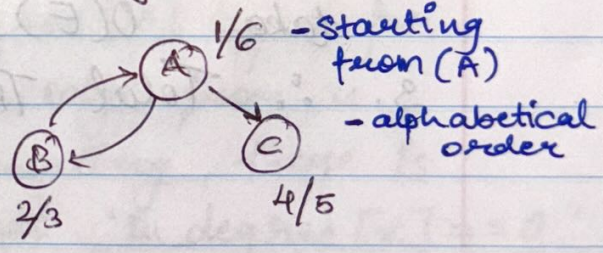
b.





Q.2. Let us consider the below directed graph





Here between nodes B & C ~~there exists~~

① there exists a path i.e $B \rightarrow A \rightarrow C$

② $B.d = 2$, $C.d = 4$ $\therefore$ $B.d < C.d$

BUT

"C" is NOT a descendant of "B".

Hence, solved.

Q.3

a. Given: graph G (DAG), graph G's "adjacency list" representation

— Algo. for computing in-degree of "V" vertices

① Initialize an array in-degree[ ] of size "V" with zeroes.

② Traverse adj. List
- for every vertex "v" in the chain of "u"th vertex.
  - INCREMENT in-degree[v] by 1.

— Analysis

1. Visiting each vertex in adj. list takes O(V) time.
2. As it is a DAG, visiting every edge takes O(E) time.
3. ∴ Total Time Complexity is O(V+E).

Q.3
b.

- The algo. given in textbook exercise can be implemented using a "QUEUE".

- Given : DAG "G", "in-degree[]" array of size "V".

- Algo:

1. Initialize an Empty queue "Q".
2. Push all vertices with "in-degree[v] == 0" into Q.
3. For each vertex in Q:
   a. Pop Q.front as vertex "u" & Add to topological order
   b. Consider all edges in the form of (u, v)
   c. Decrement in-degree[v] by 1 for every (u, v) i.e edge from u.
   d. If after decrementing, there is a vertex such that "in-degree[v] == 0"
      → Then "enqueue" that "v" into "Q".

- Analysis

1. Each vertex enqueued or dequeued once ⇒ O(V)
2. Each edge considered once while decrementing ∴ O(E).

∴ Final Time Complexity is $O(V+E)$

Q.3.

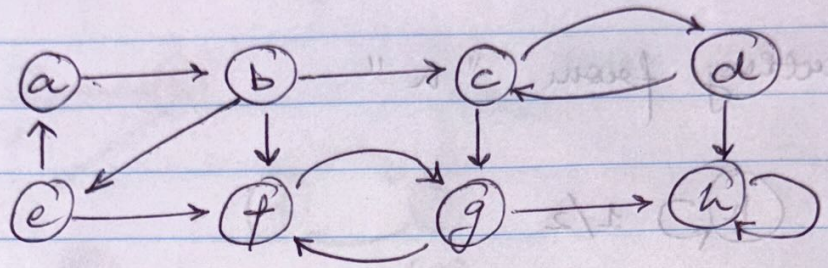
c - If G has cycles :

1. It is possible that at the start we might not have ANY vertex with indegree zero → ∴ algo. wont go ahead.
   - NOTE : Only happens when "TOO MANY CYCLES".

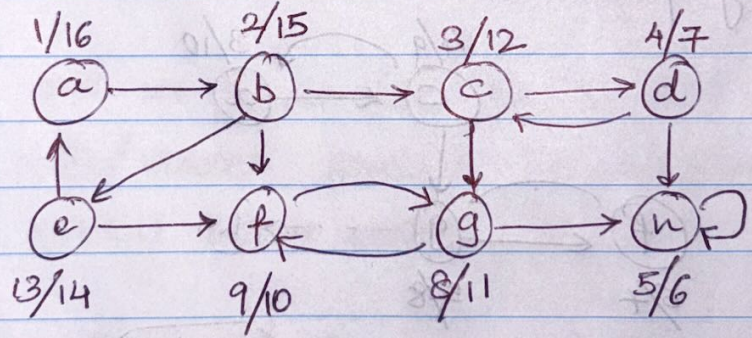
2. The queue "Q" will become empty before all vertices have been processed. which will end the algo before giving the complete topological order.
   - NOTE : This happens even if there is a "SINGLE CYCLE".
   - If this happens we can state that the "Given Graph "G" is NOT a DAG". and hence a VALID Topological sort would not be produced.

Q.4.

- Let us consider the following graph as "G"



- Step 1: DFS Traversal
  - starting from (a)
  - alphabetically sorted

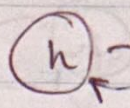


∴ Order of finished timing is (u.f)

h	d	f	g	c	e	b	a
6	7	10	11	12	14	15	16

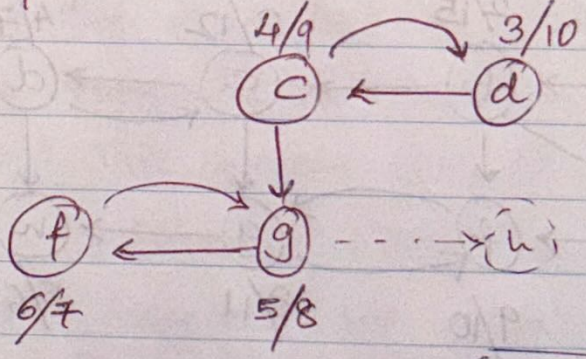
- Step 2 : Executing "Prof. Smith's" algo.
i.e 2nd DFS on original graph G & follow increasing finish time order.

i. Starting from "h"



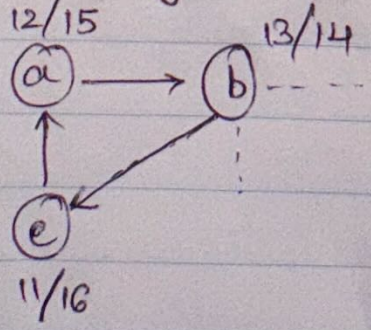
- DFS terminated ∴ (h) is a single node SCC.

ii. Starting from "d"



∴ DFS terminated ∴ (cdfg) is an SCC.

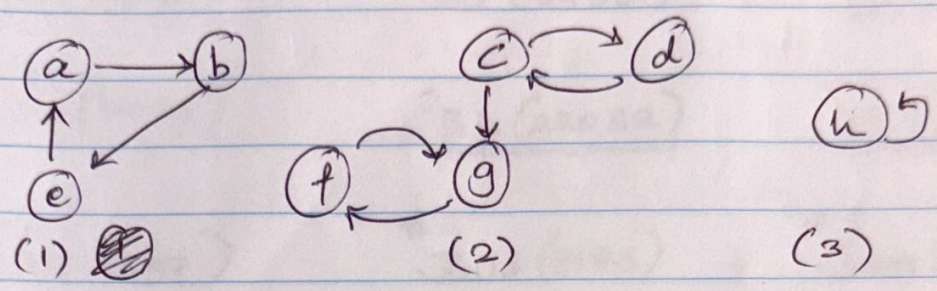
iii. Starting from "e"



∴ DFS terminated
∴ (abe) is a SCC.

= Final Results — strongly connected components

∴ we have got 3 components



BUT in SCC (2) there is a problem. as a path CANNOT be TRACED from "f to d"

and we know that in an SCC each node/vertex should be reachable to every other vertex, in SCC, via a path.

∴ The 2nd SCC is INVALID and hence Professor Smith's Algorithm is INCORRECT.

Hence Demonstrated.

Q5 undirected graph $G = (V, E)$

(a) Case 1: G has no cycle.

We will use a traversal based approach.
Since the graph has no cycles, it consists of trees.

Step 1 Select a root for the tree.
(arbitrary)

Step 2 Do a Depth first search. Maintain an array of visited vertices to avoid revisiting them. This will ensure indegree ≤ 1 for each node.

Step 3 Assign directions such that

current vertex ⟶ adjacent unvisited vertex
(parent) (child).

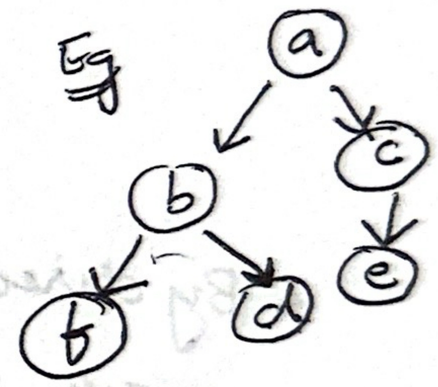
Step 4 Marks the visited vertex in the array.

Time complexity:

- Each vertex is visited only once. $(O(V))$
- Each edge is visited only once while assigning direction. $(O(E))$

$\hookrightarrow O(V+E)$

Eg



(b) Case 2: G has at most one cycle.

In this case we need to detect the cycle and handle directions to maintain the indegree constraint.

Step 1 Mark all vertices unvisited in the "visited" array.

Step 2 Do a DFS to detect the cycle.

⇒ For each neighbour v of current node 'u';

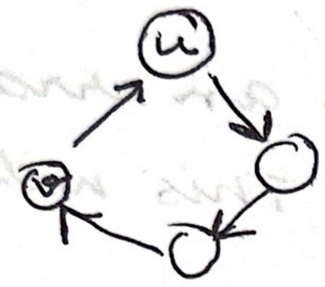
- if v is unvisited:
  parent [v] = u
  continue DFS from v
- if v is visited and v ≠ parent [u].
  cycle is detected
  record this back edge

Step3. Assign the direction of the cycle.
(single)

Use parent map to backtrack from u to v to form cycle.

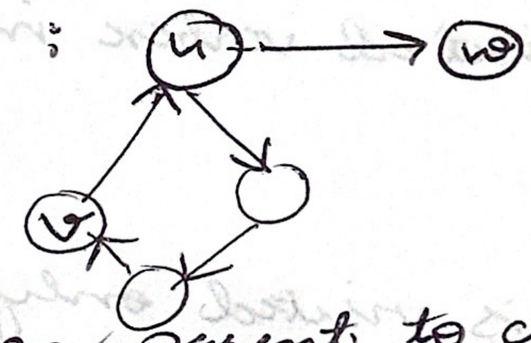
⇒ [v, parent[v], parent[parent(v)], ... u]

For each edge in the cycle, give a single direction
(say clockwise). (from u to v).

⇒ 

] → indegree = 1

Step4. For other vertices outside the cycle, do DFS
starting from cycle vertices.

⇒ For each edge (u, v) s.t 'v' is unvisited
assign direction: 

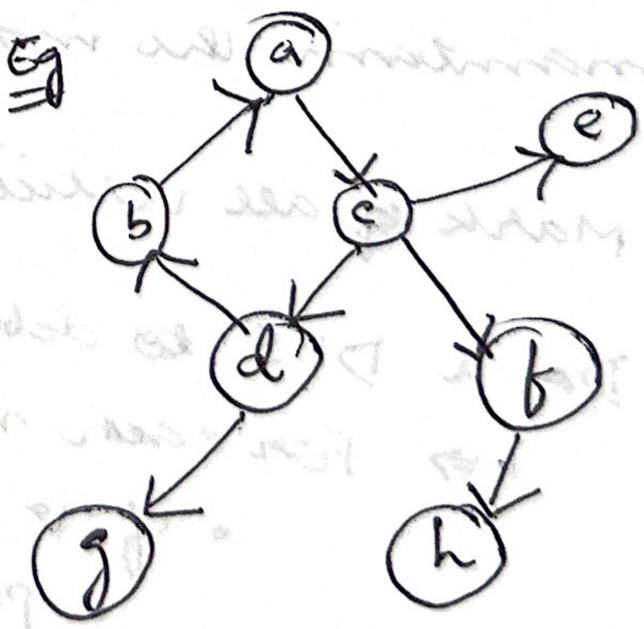
- By directing edges from parent to child (outside the cycle)
each vertex receives one incoming edge. (indegree = 1)

Time complexity:

Cycle detection and reconstruction : O(V+E)

DFS traversal : O(V+E)

↳ O(V+E)

Eg ⇒ 

12:57 AM Mon Nov 25

DAA assignment 5

6. To solve the problem of computing Max key (u) for all vertices $u \in V$ in $O(V+E)$ time, we will approach it in 2 cases : directed acyclic graphs (DAG's) and general direct graphs.

(a) Special case: Directed Acyclic Graph (DAG)

Algorithm:

1. Topological Sort :

• Since G is a DAG, perform a topological sort on the graph to get a linear ordering of vertices $v_1, v_2 \ldots v_n$ such that for every edge (U,V), U appears before V in the ordering. This can be done in

$O(V+E)$ time using a depth - first search (DFS).

2. Compute Max key (u):

- Initialize max key (u) = key (u) for all $u \in V$
- Process the vertices in reverse topological order (from the last vertex to the first)
- For each vertex u, consider all its outgoing edges (u, v). Update Max key (u) as:

  Max key (u) = max (Max key (u), Max key (v)).

Analysis:

1. Topological sort:

- A single DFS traversal computes the topological order in

- A single DFS traversal computes the topological order in $O(V+E)$ time.

2. Propagation of Max key:

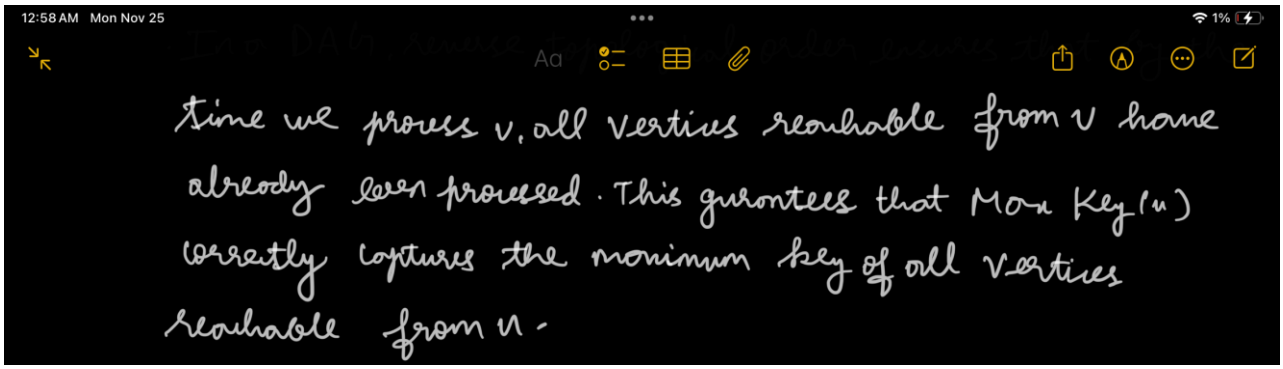
- Each edge (v,v) is considered once, leading to an update of max key (u). This takes $O(E)$ time in total

- Each vertex is processed once in reverse topological order, contributing $O(V)$.

Thus the total time complexity is $O(V+E)$

Correctness:

- In a DAG, reverse topological order ensures that by the


12:58 AM Mon Nov 25
1%
Aa
time we process v, all vertices reachable from v have
already been processed. This guarantees that Min Key(u)
correctly captures the minimum key of all vertices
reachable from u.

6b.

— In this part, as we have a general directed graph, the vertices can be a part of a cycle.

— So to tackle this problem we would prefer using "SCCs" (Strongly Connected Components) to deal with cycles.

— Algorithm

1. Find SCCs in the graph.

2. Apply "Bottom up" Approach for finding max-key values

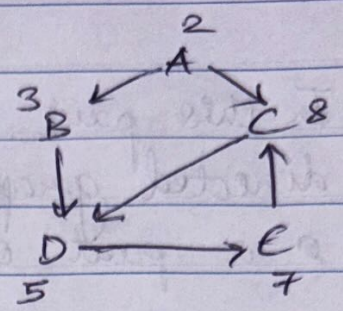
   i. First find max-key values for all nodes in a single SCC.

   ii. Then consider max. key within an SCC and PROPAGATE it throughout the graph i.e to other SCCs (if its higher value & reachable.

   iii. After propagating max values of SCCs to each other → assign the final max-key values that the resp. SCC has to every node in that given SCC.

→

— Example graph

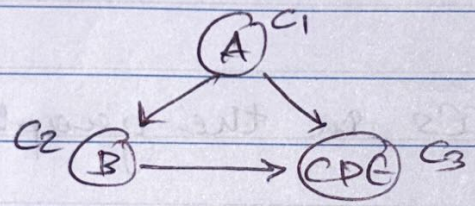


Step 1 : Find SCCs

- SCCs will be

(A) (B) (CDE)

∴



Step 2 :

using "max" instead of "max-key" as ALL NODES in an SCC will have SAME value for max-key as all nodes are reachable from every other node in the SCC.

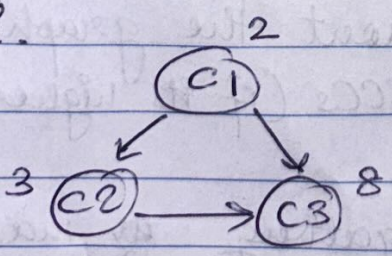
i. — for each SCC

— C_1 : $max\{A\} = 2$

$C_2 = max\{B\} = 3$

$C_3 = max\{CDE\} = 8$

ii.



iii. for C_1 — ∴ traveling examining this collapsed graph for max value.

∴ for

$c_1 = \max\{c_1, c_2, c_3\} = \{2, 3, 8\} = 8$

$c_2 = \max\{c_2, c_3\} = \{3, 8\} = 8$

$c_3 = \max\{c_3\} = \{8\} = 8$.

iii. Assigning "SCC_max" to all nodes in respective SCC.

∴ final answer is.

max_key(C1) = max_key(C2) = max_key(C3) = 8.

∴ max_key(A) = max_key(B) = max_key(C) = max_key(D) = max_key(E) = 8.

— Analysis.

1. Finding SCCs → $O(V+E)$
2. i. Finding max for every SCC → $O(V+E)$.
   ii. Propagate values via every edge → $O(V+E)$ (since propagation via DFS)
   iii. Assigning final max_key values → $O(V)$

∴ Total Time Complexity is $O(V+E)$.