Problem 3 (60 points) Anchestors Set. Given a directed graph $G = (V, E)$, find a minimum-size set of vertices $S \subset V$, such that any vertex in V can be reached from some vertex in S : for each $v \in V$, there exists $u \in S$ with path $u \to v$. *HINT: try to solve the problem for a DAG fist.*

Solution for DAG :

① Perform a topological sort on the graph.

② Perform DFS in order of the topological sort for unvisited vertices. Every new DFS's start vertex will be added to the set.

Algorithm :

```
function   TopologicalSort(G) :
    for  u  in G.V :
         u.d = 0, u.f = ∞
         u.color = white
         u.parent = None
    time = 0  ⇐ Global  ──→  list = [ ]  { linked list of nodes/vertices }
    for  u  in G.V :
         if   u.color == white :
                DFS-Recursive(G, u)
    return list

function  DFS-Recursive(Graph G, vertex u) :
     u.color = grey
     time ++
     u.d = time
     for v  in  G.adj[u] :
            if  v.color == white :
                  v.parent = u
                  DFS-Reursive(G, v)
     u.color = black
     time++
     u.f  = time
     list.addFront(u)
```

to find the topological sort, we use DFS & insert vertices into a linked list when they are changed to black. This is done in O(V+E) in the algorithm mentioned in the previous page.

Now, we perform another DFS on the topologically sorted list & add the source vertices to the set.

~~function findMinimalSet(G):~~
~~curr list = TopologicalSort(G)~~
~~while curr != null:~~
~~if curr.~~

```
function findMinimalSet(G):
    head = TopologicalSort(G)
    temp = head
    while temp != null:          } Reset colors to white
        temp.color = white
        ~~temp.d = 0~~
        ~~temp.f = 0~~
    Set S = {}
    temp = head
    while temp != null:
        if temp.color = white:
            S.add(temp)
            DFS-Recursive(temp)      // Same DFS-Recursive as before,
                                     except we do not
        temp = temp.next;            add nodes to a list
    return S
```

Time Complexity Analysis: O(V+E) for topological sort
O(V+E) for DFS
∴ overall = O(V+E)

6

Auxiliary space: linked list: O(V)

Result is <u>correct</u> because these vertices have no incoming edges.

Problem3: Solution for all Directed Graphs:

Solution: • Create a DAG of Strongly connected components.

• Remember one node from each strongly connected component.

• Now since we have a DAG, we can use the previously stated solution to find the vertices in S, but the vertices we add to S are not the Strongly Connected Components but the nodes we remembered for each strongly connected component.

Finding Strongly Connected Components:

function SCC (G):

Call DFS(G) to compute finish times of all vertices.

Compute G^T {in $O(V+E)$}

Call DFS (G^T), but inside the main loop of DFS, consider vertices in decreasing order of finish times

* For each component found in the DFS of G^T

~~Add~~ Add a new vertex to ~~the~~ graph of SCC.

* Check for edges in the DFS run & ~~connect them to~~ add edges to SCC graph if any.

* Remember one vertex for each SCC in the SCC graph.

- Now, the SCC graph is A DAG.
- Use the previously described algorithm to find the Ancestor Set S on this DAG.

Run time Analysis :

SCC part :

computing finish times : $O(V+E)$

Computing G^T : $O(V+E)$

Finding SCCs : $O(V+E)$ {DFS on G^T}

↳ SCC graph created simultaneously.

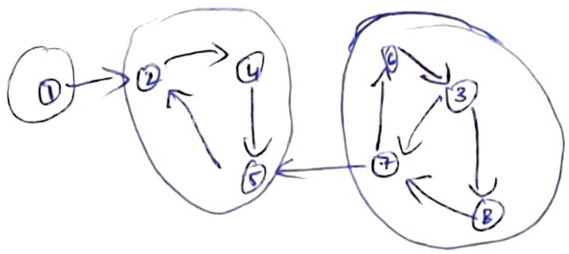
Finding Ancestor Set from DAG :

Topological sort : $O(V+E)$

DFS : $O(V+E)$

Overall time Complexity : $O(V+E)$

Depiction :



SCC graph

Ancestor Set : {1, 3}