Application: DFS

Joy Mukherjee

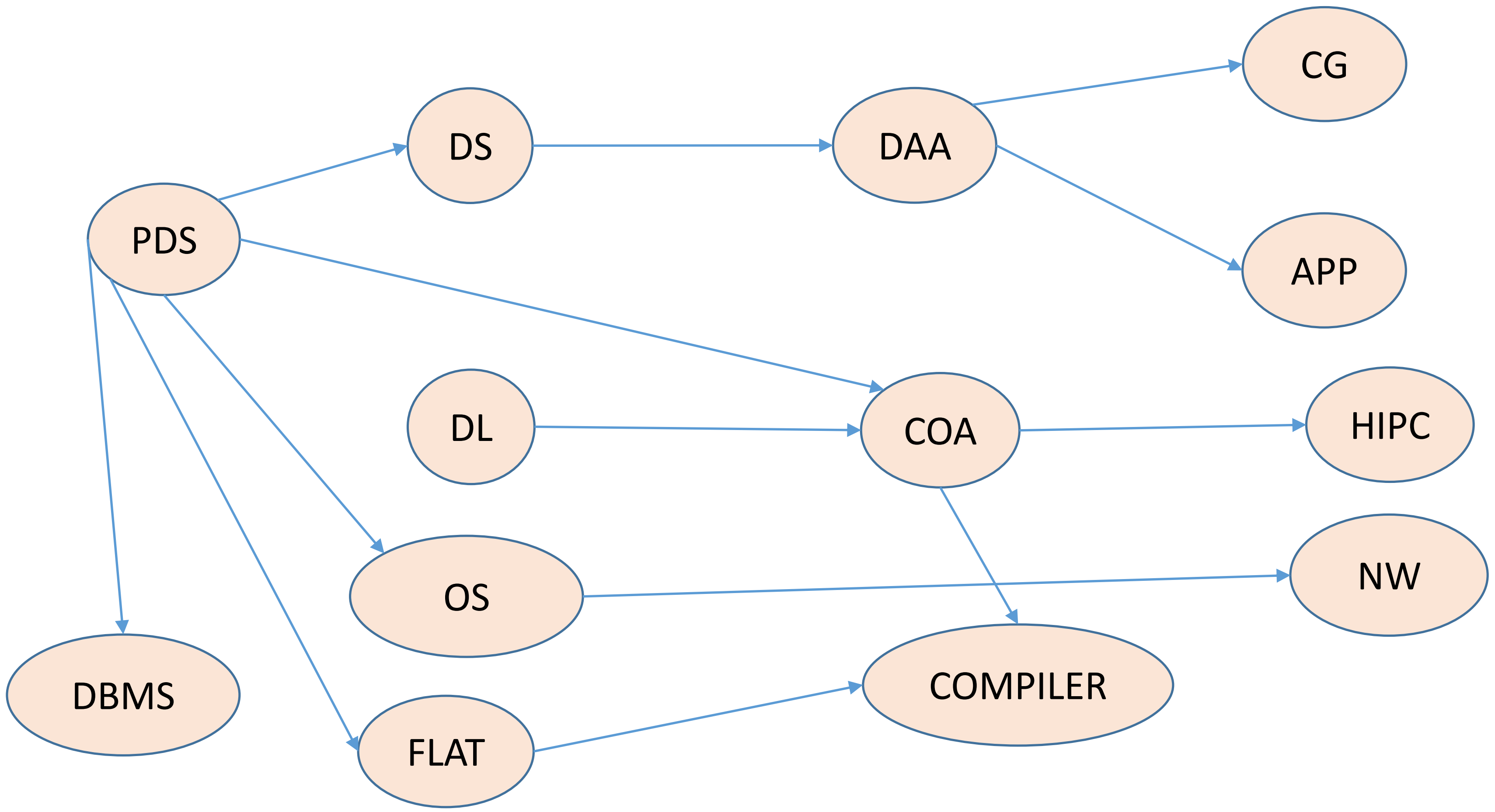
Application: DFS

- **Cycle detection in a graph**
  - **Undirected Graph:** If it is a tree, then no cycle is detected. Check for back edges. If exists, then a cycle is detected.
  - **Directed Graph:** If a back edge is found, then a cycle is detected.
- **Topological Sort on a Directed Acyclic Graph (DAG)**
- **Finding Strongly Connected Components (SCCs) is a directed graph**

Topological Sort on a DAG

- The application areas:
- Prerequisite List in a course curriculum
- PDS -> DS -> DAA
- FLAT -> Compiler
- OS
- DL -> COA
- To design our semester structure in terms of course distribution
- The directed edge indicates the dependency (Asymmetric relation)
- **Definition**: A topological sort in a directed acyclic graph G = (V, E) is a linear ordering of vertices of the graph such that if G contains an edge (u, v), then u appears before v in the ordering.

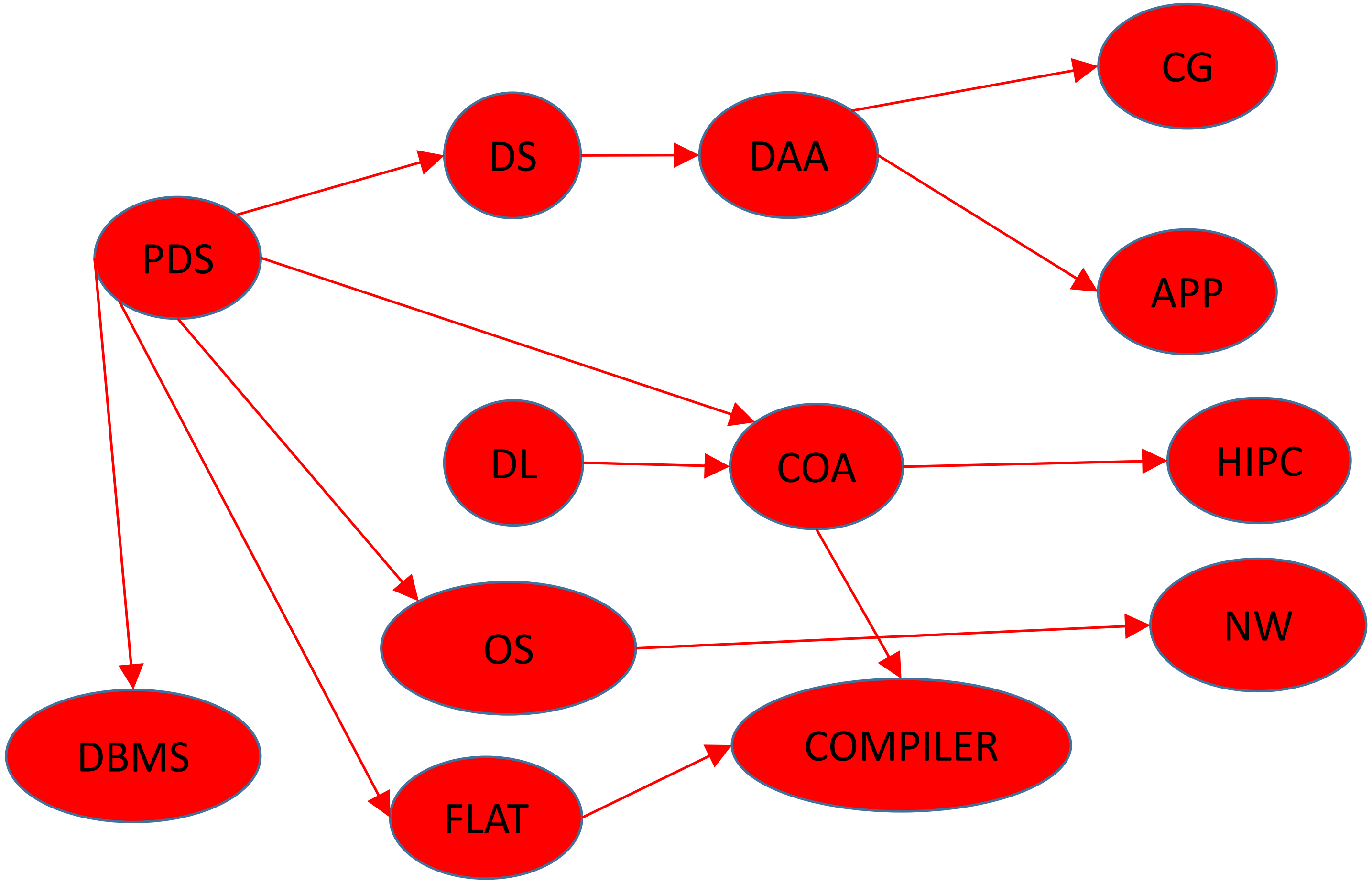
Topological Sort



PDS, DL, DS, DAA, CG, APP, COA, FLAT, COMPILER, HIPC, OS, DBMS, NW (Not a unique ordering)

DL, PDS, DS, DAA, CG, APP, COA, FLAT, COMPILER, HIPC, OS, DBMS, NW (Not a unique ordering)

Topological Sort: Algorithm




```
while(G has a vertex v with indegree 0) {
    Delete v and its associated edges
    Print v
}
```

PDS, DBMS, FLAT, DS, DAA, APP, CG, OS, DL, NW, COA, COMPILER, HIPC

Algorithm: Topological Sort

- DFS(G)

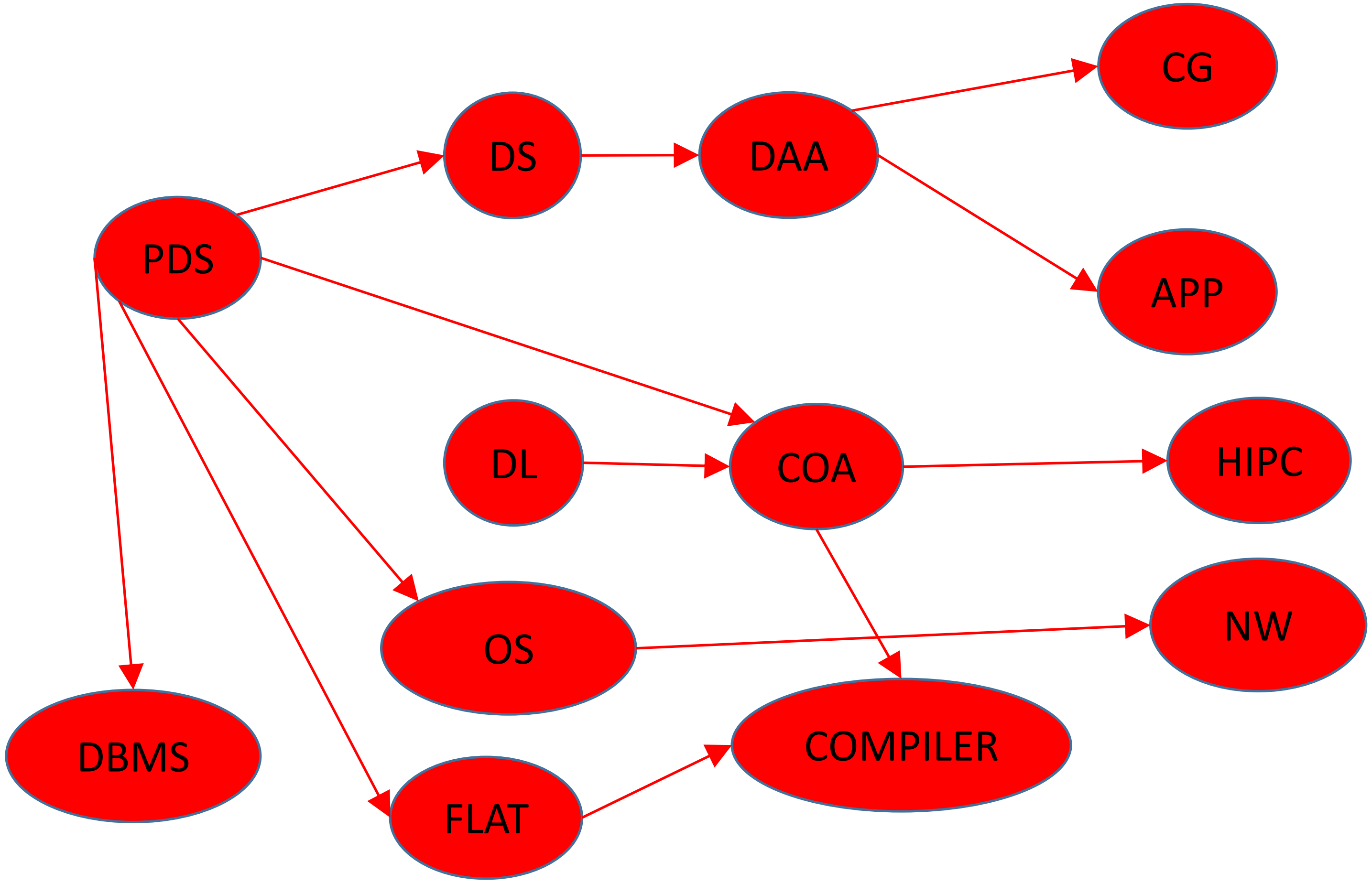
1. For all vertex x in V
   parent[x] = NULL
   visited[x] = 0
2. time = 0
3. For each vertex u in V
   If(visited[u] == 0)
   DFS_VISIT(G, u)
4. While(Stack is not empty)
   a. u = Stack.Pop()
   b. Print u

- DFS_VISIT(G, u)

1. visited[u] = 1
2. disc[u] = time = time+1
3. For each v in Adj[u]
   a. If(visited[v] == 0)
      i. parent[v] = u
      ii. DFS_VISIT(G, v)
4. finish[u] = time = time + 1
5. Stack. Push(u)

Time Complexity = $O(|V| + |E|)$

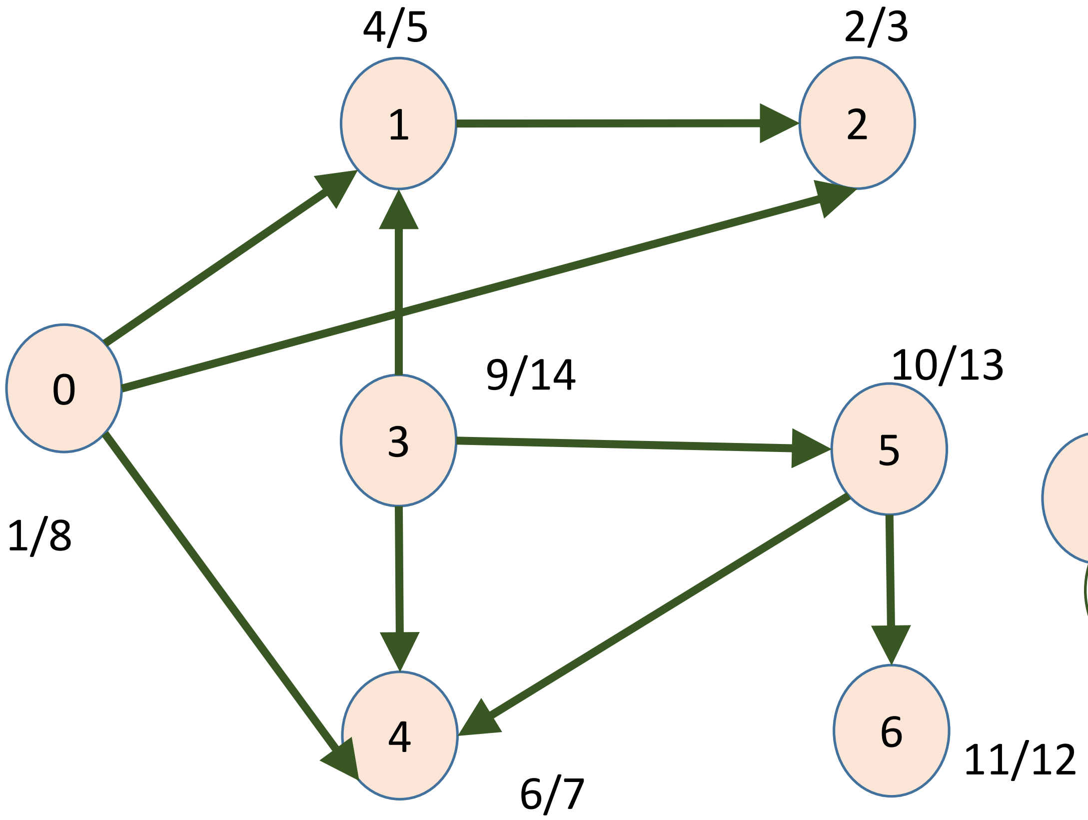
Topological Sort: Algorithm

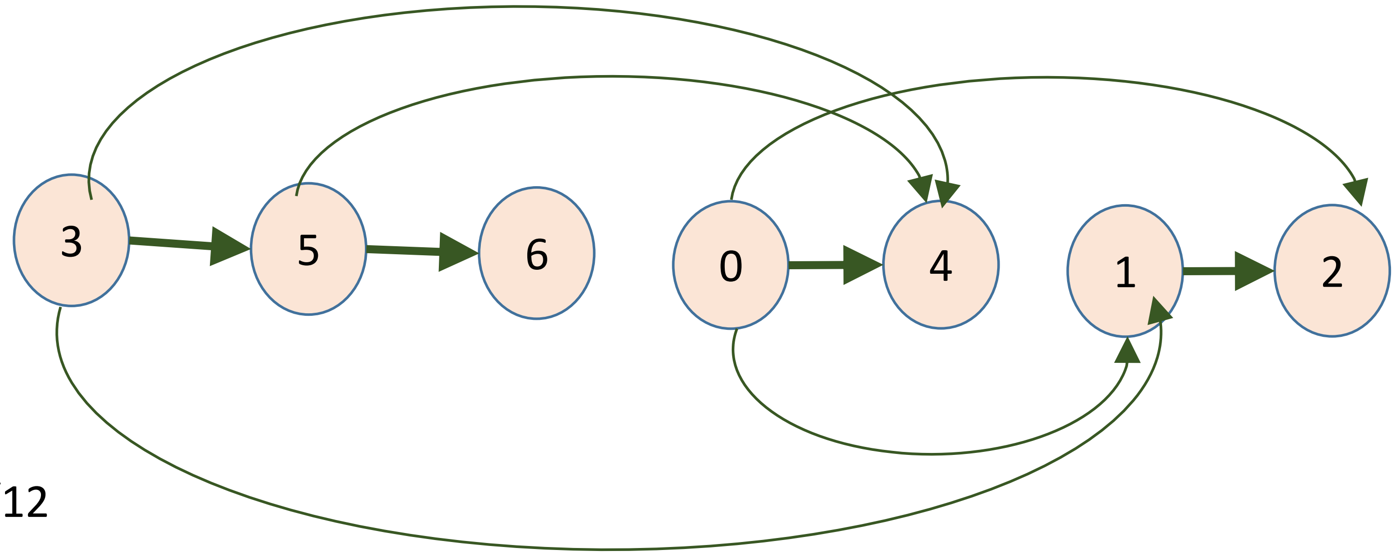



```
while(G has a vertex v with indegree 0) {
    Delete v and its associated edges
    Print v
}
```

DL, PDS, DBMS, FLAT, OS, NW, COA, HIPC, COMPILER, DS, DAA, APP, CG

Topological Sort on a DAG

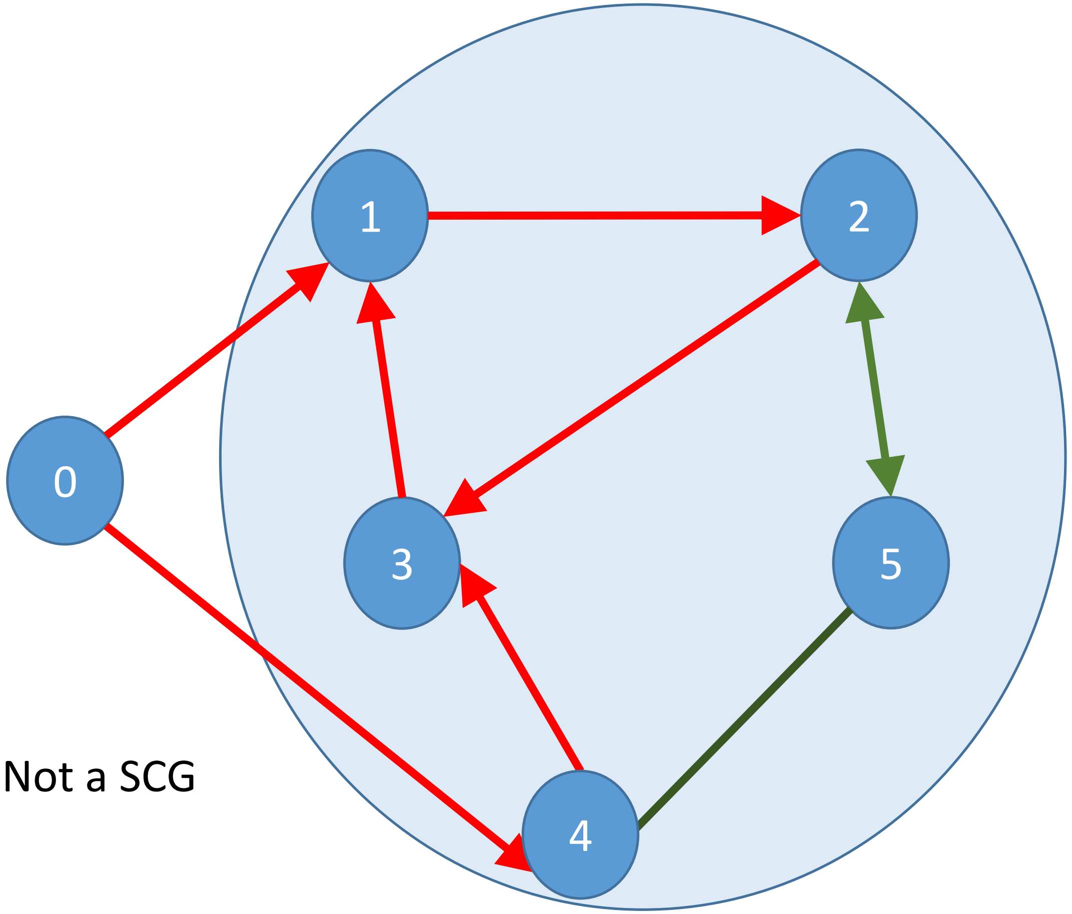




TS: 3, 5, 6, 0, 4, 1, 2

Finding Strongly Connected Components

- A directed graph is called a strongly connected graph if there is a path between every pair of vertices.
- Strongly Connected Components: In a directed graph, a SCC is defined as a maximal strongly connected subgraph.
- A SCC of a directed graph G=(V, E) is a maximal subset of vertices U ⊆ V such that for each pair of vertices x and y in U, there is a path from x to y and y to x.
- If a directed graph is not Strongly Connected, then it can be decomposed into SCCs.



Not a SCG

Finding Strongly Connected Components

- If a directed graph is strongly connected, then there is only one SCC, which is the graph itself.
- Otherwise, the graph can be decomposed into SCCs.

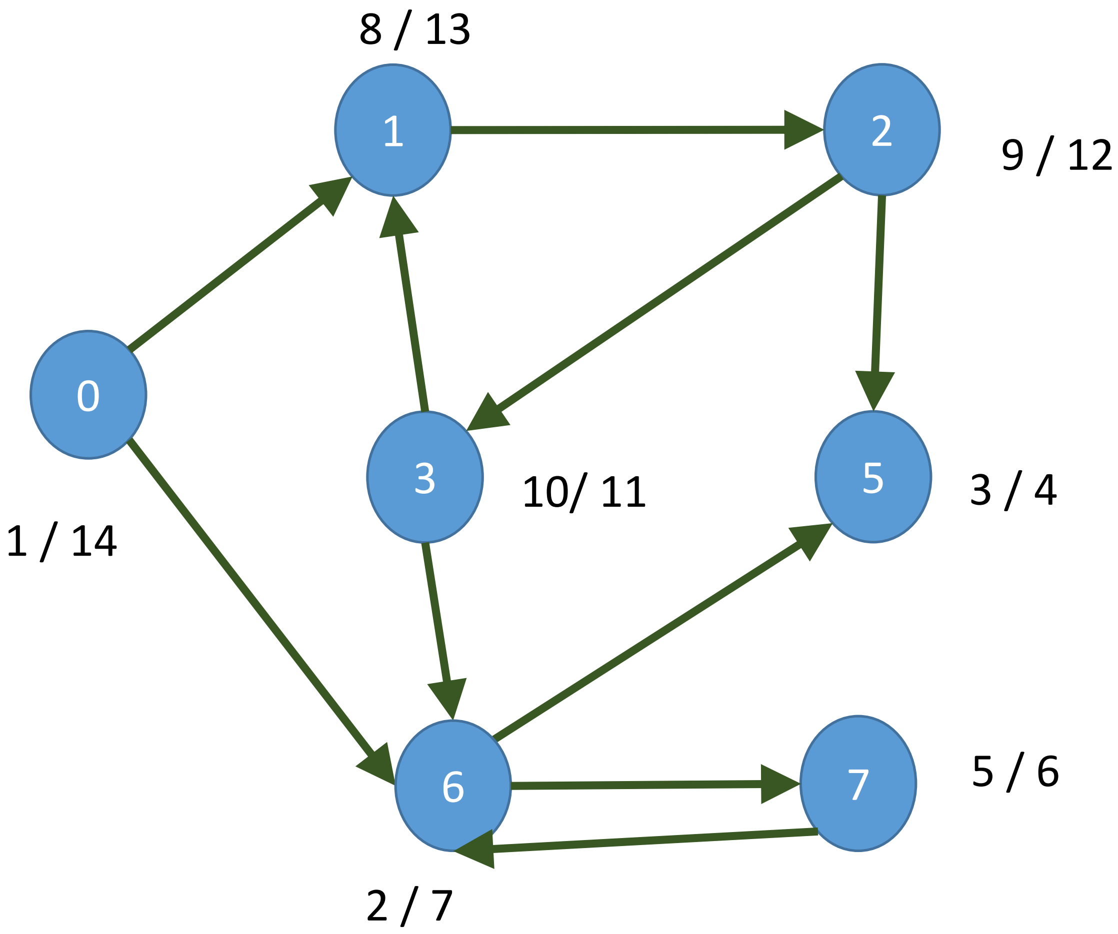
Kosaraju's Algorithm:

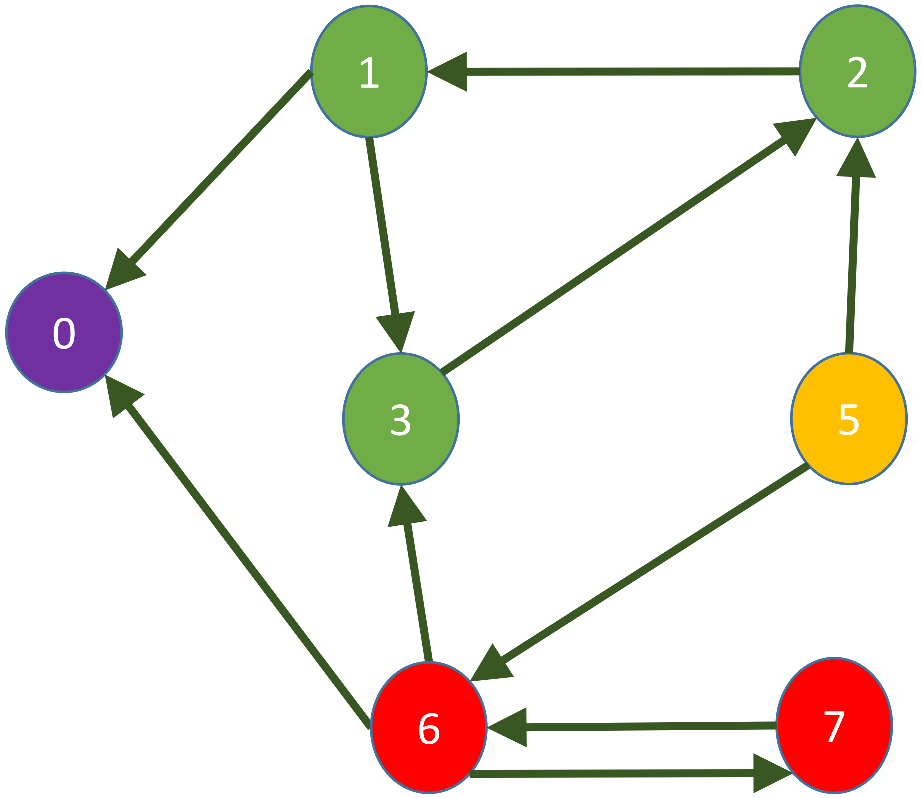
- Input: The directed graph G = (V, E)

1. DFS(G), and compute the finish times of the vertices.
2. Compute $G^T = (V, E^T)$.
3. DFS(G^T) where the vertices are explored in order of decreasing finish times.

Time Complexity = $O(3*(|V| + |E|)) = O(|V| + |E|)$

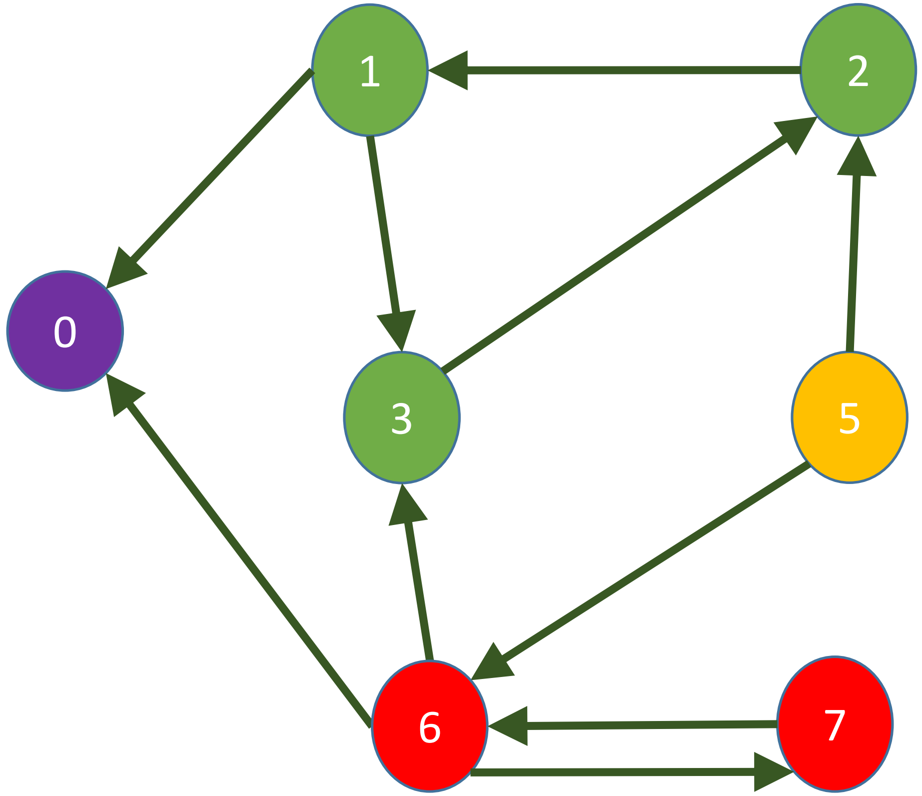
Finding Strongly Connected Components



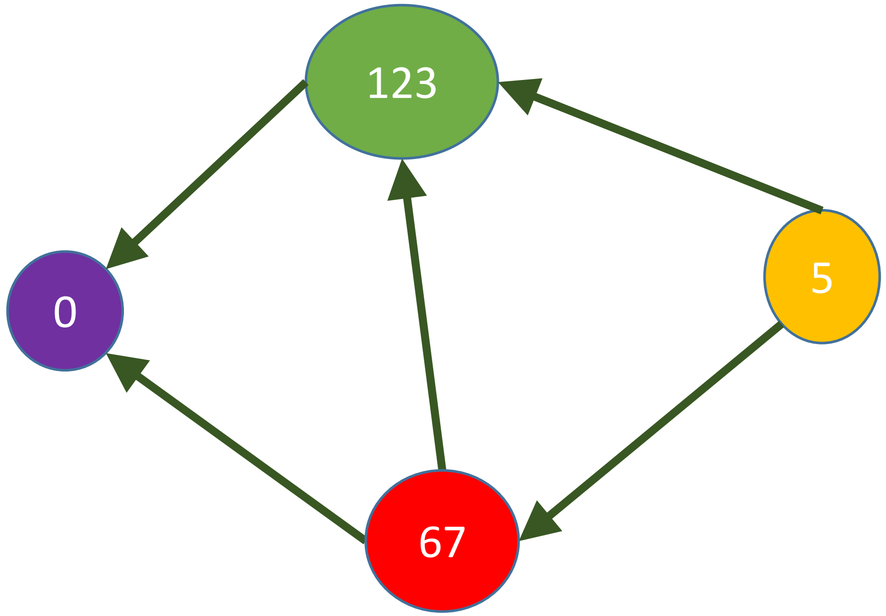


SCC = {(0), (1, 2, 3), (6, 7), (5)}

Finding Strongly Connected Components



SCC = {(0), (1, 2, 3), (6, 7), (5)}



Component Graph: DAG