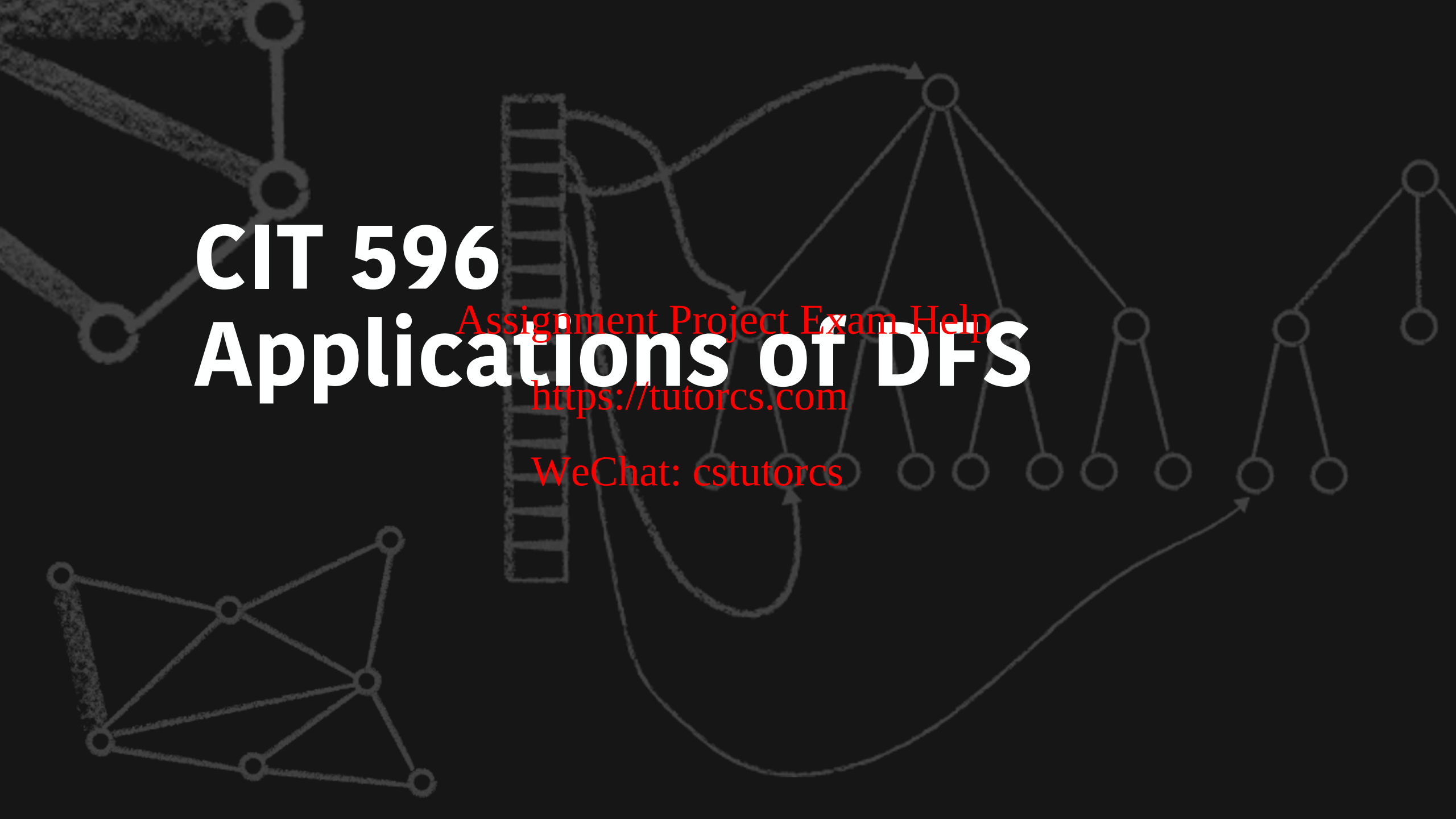


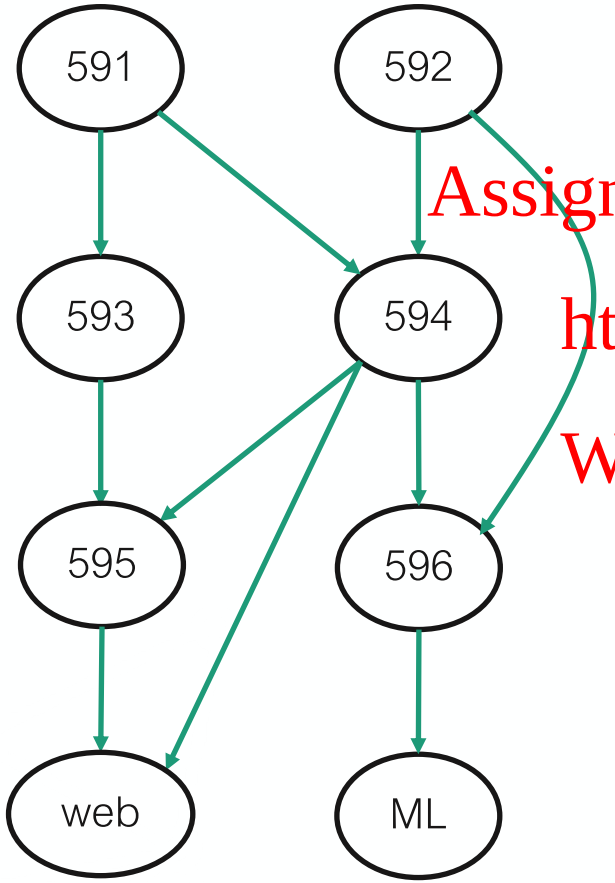
CIT 596
Applications of DFS

SOURCES AND SINKS IN DAGs

- DAG: Directed Acyclic Graph
- Call a node v a source if it has no incoming edges and a sink if it has no outgoing edges.
- Fact: DAGs always have sources and sinks.
- Proof:
  - Suppose not. Then start from some vertex v and walk along its edges.
  - Either the walk terminates, or we repeat a vertex.
  - But repeating a vertex would imply a cycle, which we assume not to be the case (DAG).
  - Thus we terminate at a sink, so a sink exists!
  - Showing a source exists can be done by walking backward on edges.

DFS APPLICATION 1

TOPOLOGICAL SORT ON DAGs



Prerequisite structure for MCIT courses

Topological sort: a valid ordering of courses

Example: 591, 592, 593, 594, 595, 596, ML, web

But also: 592, 591, 594, 596, 593, ML, 595, web

Definition: for every edge (u, v), u must precede v

DFS FOR TOPOLOGICAL SORT

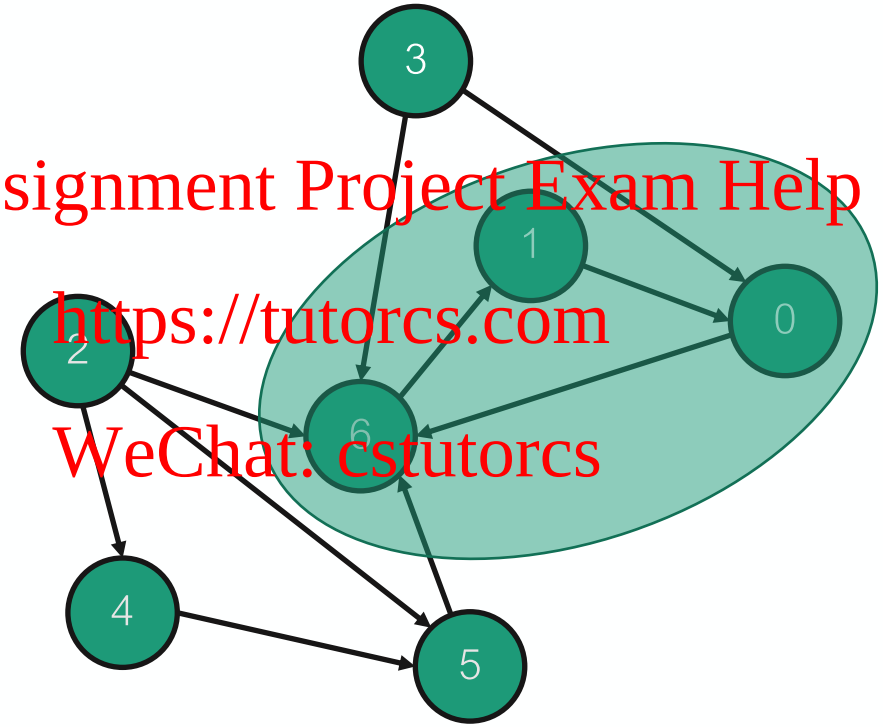
- One topological sorting algorithm:
  - Run DFS.
  - Output the vertices in decreasing order of finish time.
  - Why does this work? If (u, v) is an edge, then
    - $s(u) < s(v) \Rightarrow s(u) < s(v) < f(v) < f(u)$
    - $s(u) > s(v) \Rightarrow s(v) < f(v) < s(u) < f(u)$
  - In either case, u finishes after v, so u precedes v in the output order

ANOTHER TOPOLOGICAL SORT

- Output and delete vertices with indegree 0, along with edges incident on them.
- Update degrees for remaining vertices and repeat until there are no vertices left.

- Explore efficient implementation and correctness of this algorithm on your own.

STRONGLY CONNECTED COMPONENTS

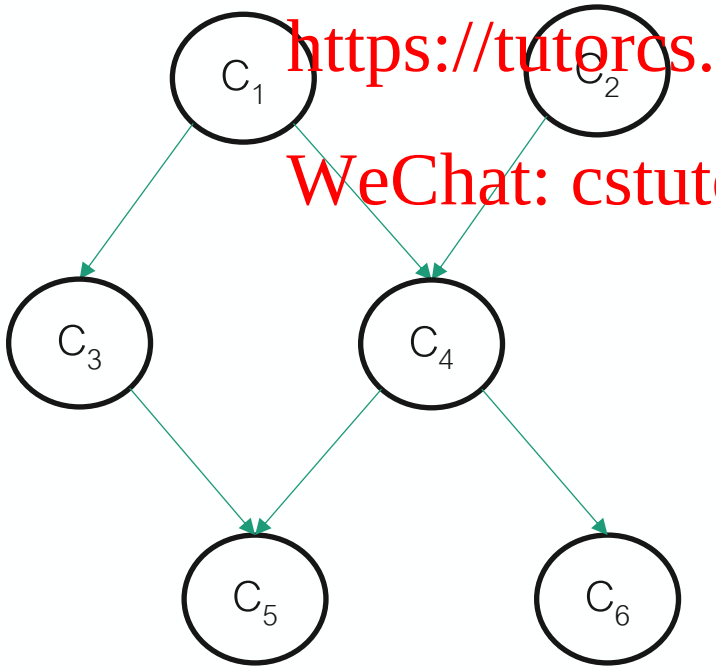


Vertices 0, 1, and 6 are in one strongly-connected component.

Every other vertex is in a component by itself.

INTUITION FOR FINDING SCCs

- Recall $\mathrm{SCC}(G)$ for a directed graph G
  - Has components of G as vertices
  - Edge (C_1, C_2) if some vertex $u \in C_1$ has an edge to some vertex $v \in C_2$
- $\mathrm{SCC}(G)$ is a DAG
- Example:



C_1, C_2 : Source components

C_5, C_6 : Sink components

INTUITION CONTINUED

- If DFS started at vertex v, it will visit all vertices in the connected component of v and in descendant components.

- Vertices with latest finish times are always in source components

- If DFS is started at a vertex v in a sink component, it will exactly visit the vertices in the strongly connected component of v

INTUITION CONTINUED

- Given graph G:
  - Run DEPTH-FIRST-SEARCH on G
  - Record finish times for all vertices in f
  - Create G^T: the graph obtained by reversing all edges of G
  - Run DEPTH-FIRST-SEARCH on G^T
    - But pick unseen vertices in decreasing order of finish time $f(x)$
    - Output vertices visited during each DFS call as one SCC

CORRECTNESS

- The latest finish times are in source components of G, which are sink components of G^T.
- The first vertex chosen in the second DEPTH-FIRST-SEARCH is in sink component of G^T
- So the first DFS during this DEPTH-FIRST-SEARCH will identify this component.
- Recursively, one can argue that the next starting vertex in the second DEPTH-FIRST-SEARCH must be in a new sink component.
- Correctness follows.

RUNNING TIME

- Just two DEPTH-FIRST-SEARCH calls
- $O(n + m)$