Algorithm Design & Analysis (CSE222)

Lecture-14

Recap

- DFS Revisit
  - Back Edge, Forward Edge and Cross Edge
  - Start / Finish Time
  - v is descented of u if start(u) < start(v) < finish(v) < finish(u)
  - u and v are not related if finish(u) < start(v) or finish(v) < start(u)

- DFS Application
  - Cycle identification
  - Longest heavy path using topological sorting.

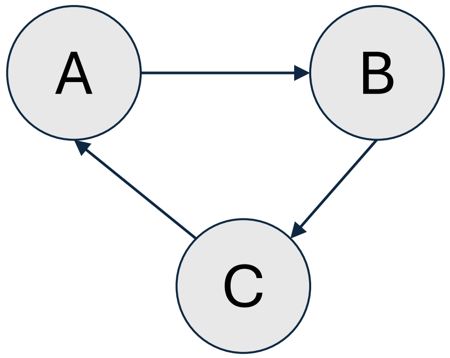
Outline

- Strongly Connected Components

Strongly Connected

Two vertices u and v are **strongly connected** if u can reach v as well as v can reach u.

A directed graph is **strongly connected** if and only if every pair of vertices is strongly connected.



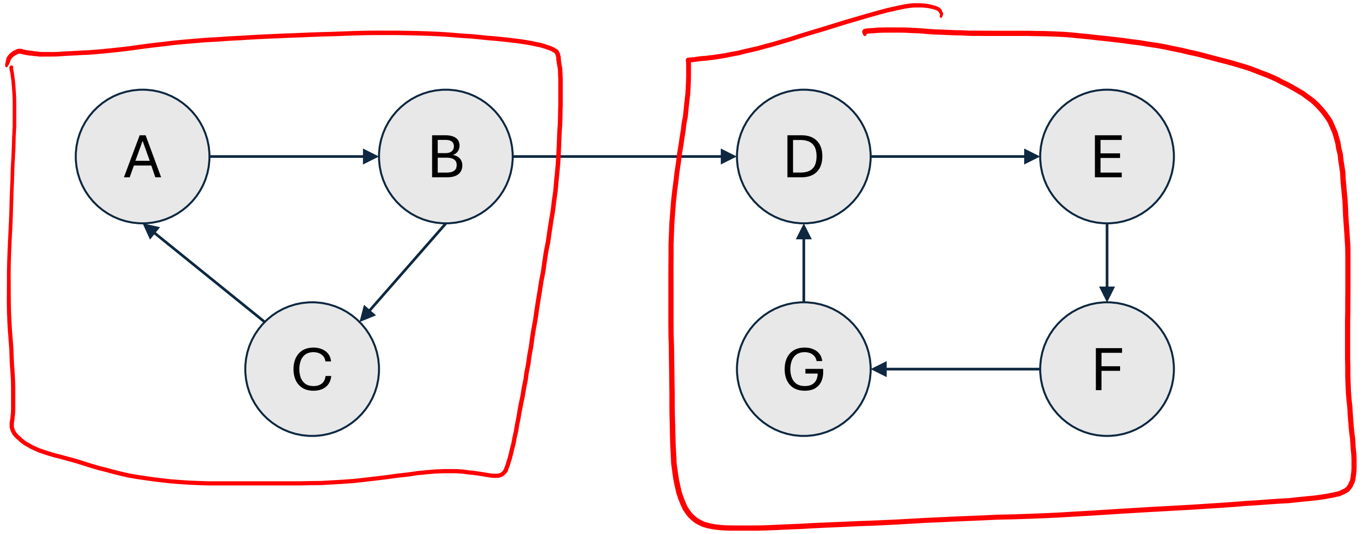
Strongly Connected

Two vertices u and v are **strongly connected** if u can reach v as well as v can reach u.

A directed graph is **strongly connected** if and only if every pair of vertices is strongly connected.

A **strong connected component (SSC or SC)** of a graph G is the maximal strongly connected subgraph of G.

Strong Component



$S_1 = \{A, B, C\} \subseteq V$

$A \to B,\ B \to A;\ B \to C,\ C \to B;\ A \to C,\ C \to A$

$S_2 = \{A, B, C, D\}$ $\quad A \to D,\ D \not\to A$

$\quad B \to D \quad D \not\to B$

Strongly Connected

Two vertices u and v are **strongly connected** if u can reach v as well as v can reach u.

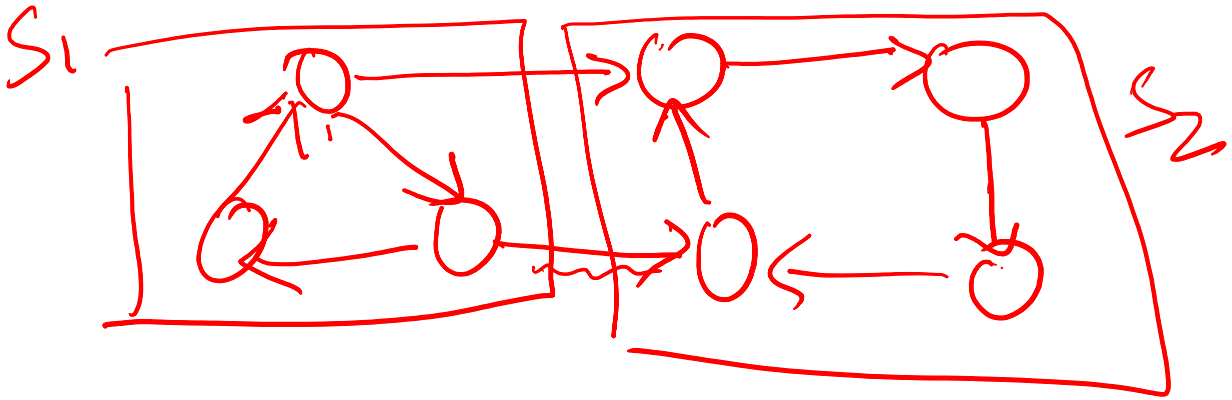
A directed graph is **strongly connected** if and only if every pair of vertices is strongly connected.

A **strong connected component (SSC or SC)** of a graph G is the maximal strongly connected subgraph of G.

A directed graph is strongly connected if and only if G has exactly one strong component.

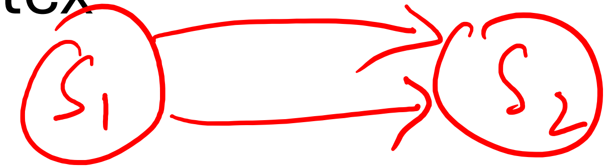
What is the strong component in a DAG?

Strong Component

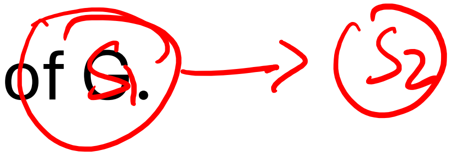


Let G be a directed graph.

- Collapse every strong component to a single vertex
- Collapse parallel edges between these vertex.



The resultant graph is called **meta-graph** or **condensation** of G.



Prove that condensation of G is a DAG.

Strong Component

Design an algorithm to compute all the strong components from a vertex v.

- Run DFS on G and compute reach(v)
- Run DFS on G and compute $\text{reach}^{-1}(v)$.
- Compute reach(v) ⋂ $\text{reach}^{-1}(v)$.

What is the running time to check if the complete graph G is strongly connected?

Design an algorithm for computing all strong component in G.

Strong Component

Claim: Fixing a DFS traversal of any directed graph G. Each strong component C of G has exactly one node that does not have a parent in C.

Proof:

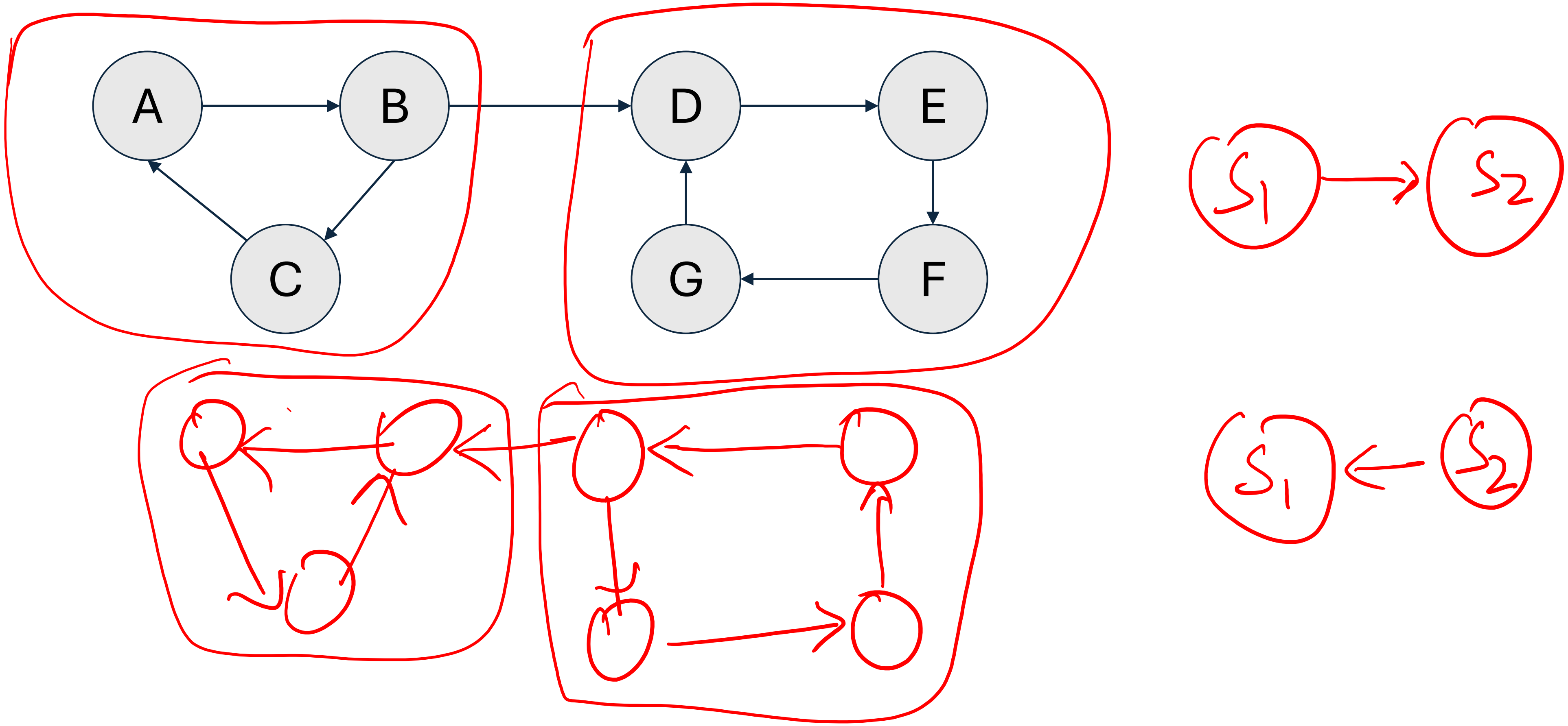
- Let C be an arbitrary sc of G. Consider a path $v \in C$ to $w \in C$.
- Every vertex in this path can reach w and it can also reach C.
- Similarly every vertex in this path cab reached from v and can also be reached from any vertex in C.
- So every vertex in this path is also in C.
- Let $v \in C$ is the earliest vertex, i.e., start(v) is lowest among all vertices $\in C$.
- So, at the call DFS(v) all vertex in C were not visited, so any $w \in C$ is a descendant of v. So except v, parent of every vertex in path v to w is in C.

Algorithm Strong Component

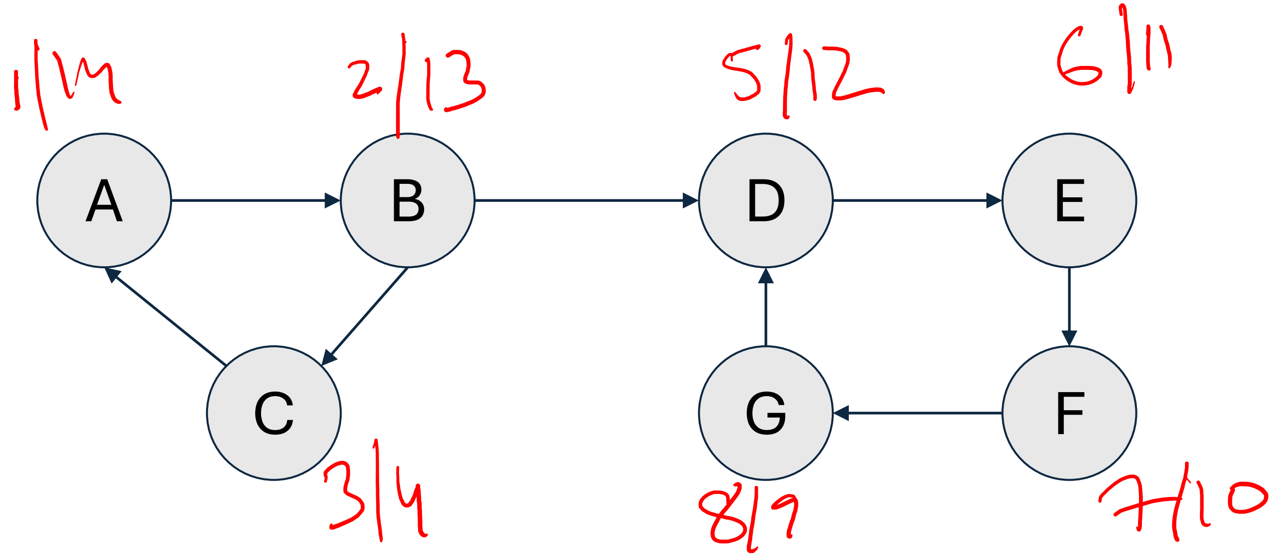
```
StrongComponents(G):
  count ← 0
  while G is non-empty
      C ← ∅
      count ← count + 1
      v ← any vertex in a sink component of G    ⟨⟨Magic!⟩⟩
      for all vertices w in reach(v)
          w.label ← count
          add w to C
      remove C and its incoming edges from G
```

How to find sink component?

Strong Component



Strong Component



SC(G) is equal to SC(reverse(G))

C G F E D B A

Strong Component

- Find source component of G.
- Compute SC(reverse(G)). (why?)

Strong Component

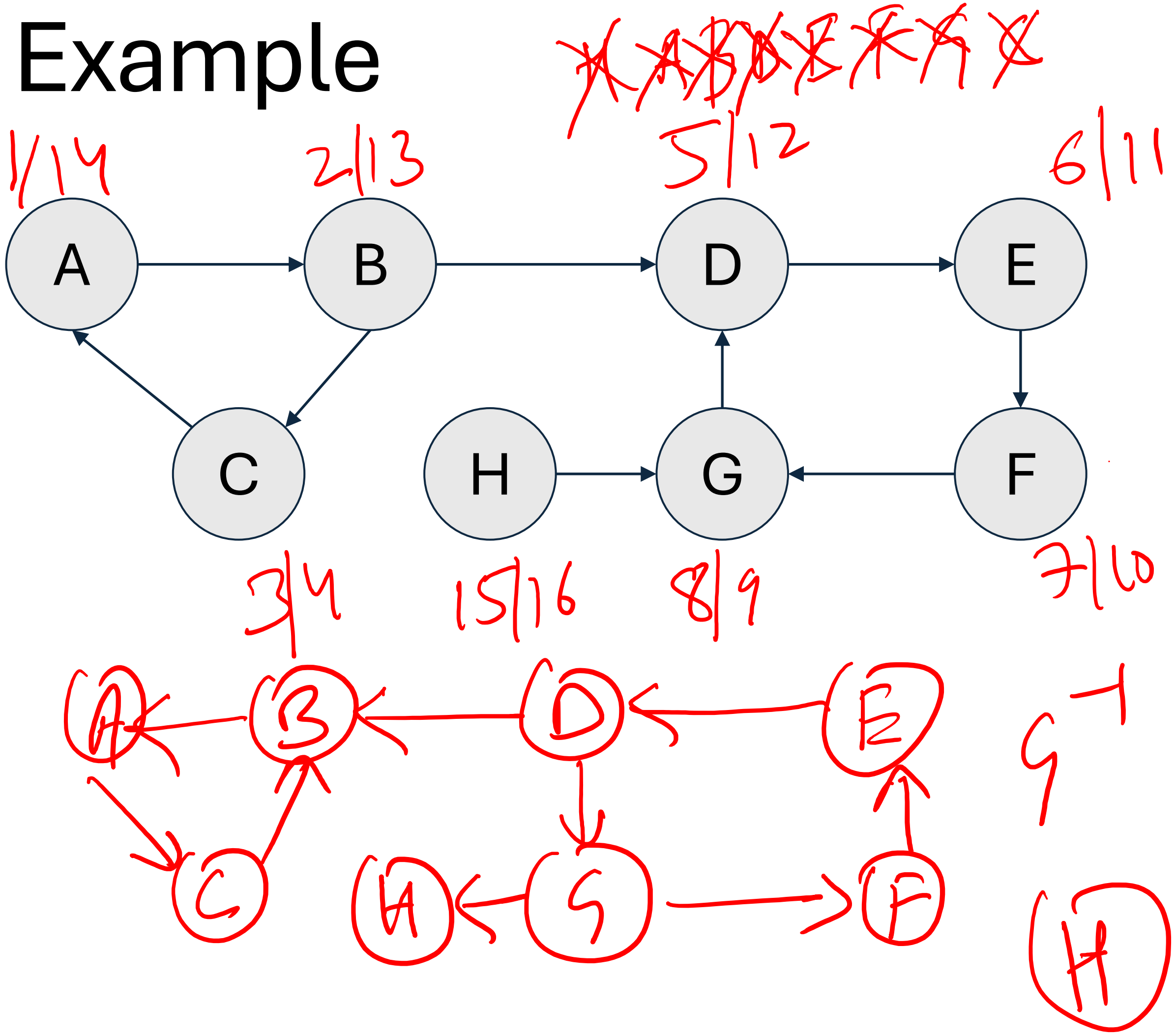
Claim: The last vertex in any post ordering of directed a graph lies in a source component of G.

Algorithm

Input: Graph G
Output: Compute strong components.

1. Compute postordering(G) and store it in a stack.
2. Compute G^{-1}.
3. Pop vertex from stack and save nodes reachable from that vertex in G^{-1}.
4. Remove the all the reachable vertices from the stack and repeat 3.

Example



Input: Graph G
Output: Compute strong components.

1. Compute postordering(G) and store it in a stack. $O(V+E)$
2. Compute G^{-1}.
3. Pop vertex from stack and save nodes reachable from that vertex in G^{-1}. $O(E)$
4. Remove the all the reachable vertices from the stack and repeat 3. $O(V)$

H

A, C, B

D, G, F, E

Reference

Slides

Jeff Erickson Chp-6.5 & 6.6