Applications of DFS:

Topological Sort (TS):
Given a digraph $G$,
Find ordering of vertices s.t.
$(u,v) \in E \Rightarrow u$ comes before $v$.

$$
	ext{DFS}(G) \{
1. \text{ unmark all vertices} \\
2. \text{ for } u \in V \\
    \text{ if } u \text{ is unmarked} \\
    \text{ DFS}(G, u)
\}
$$

$acbed \ aecbd$

Q: Does TS always exist? NO!

If $G$ a cycle then no TS.

So let's assume that $G$ is acyclic (DAG).

Aside: A DAG always has a source node (in-degree=0)
and a sink node (out-degree=0).
First Algorithm:

1. Find a source vertex \( u \).
2. Output \( u \).
3. Remove \( u \) and its edges. Repeat.

How to find a source vertex?

- Run DFSAll(6)
- Pick \( u \) = vertex \( u \) by largest finish time.

Proof of Correctness (Sketch):

DFSAll(6)

- **DFS(6, u)**
  - Mark \( u \)
  - \( \text{Finish}(u) = \text{first} \) output \( u \).

Proof and correctness (Sketch):

- A vertex \( u \) is a descendant of a vertex \( v \).
  - \( (v, u) \) is a back edge.
  - \( (v, u) \) is not possible.
**Observation**

Assume $G$ is a DAG

1. $(u, v) \in E$
2. $\text{finish}(u) > \text{finish}(v)$

Q: How to remove $u$ & repeat?

Do nothing! Just pick the next largest finish time.

**Final Algorithm:**

1. Run DFSAll($G$) & compute finish times.
2. Output vertices in the decreasing order of finish time.

Running time $O(n + m)$

Corollary: If a topological sort $\Leftrightarrow G$ is acyclic (DAG).

**Strongly Connected Component (SCC)**

- $a \rightarrow b$
- $c \leftarrow b$
- $d \rightarrow c$
- $e \leftarrow d$
- $e \leftarrow e$

back edge $\Rightarrow$ cycle inside.
Given a digraph $G$, we say $u, v \in V$ are s.c. if $\exists \; u \rightarrow v \land v \rightarrow u$.

s.c. Relation: symmetric, reflexive, transitive

is an equivalence relation

Decomposes $V$ into disjoint components/sets.

Find $\text{SCC}$ of digraph $G$:

partition $V$ into components s.t.

$u, v$ are in the same component $\iff$

$u \sim v$ & $v \sim u \quad (u, v \text{ are s.c.})$

**Example:**

[Diagram of a directed graph with labeled nodes and arrows indicating connectivity.]
\[ \{a, b, c, d, e, f, g, h, i, k\} \]

within a s.c.c we can go from anywhere to anywhere

App'n - control flow in program, ...

Simplify a digraph into a DAG

Naive Approach:
- Test readability of every pair of vertices.
  \[ t & \text{ find s.c.c.} \]
  for each, just run a DFS/BFS,
  \[ O(n) \times O(m+n) = O(n(m+n)) \]
  - Find a cycle, shrink it, repeat.

History:
- Ruzdov '68 \[ O(n^2) \]
- Munro '71 \[ O(m+n \log n) \]
- Tarjan '72 \[ O(m+n) \] < complex.
- Kosaraju '78 \[ O(m+n) \] simple.
- Shiloach '81

First idea: ...
First idea:
1. Find a vertex \( u \) in the source component of the metagraph.
2. Find \( u \)'s component.
3. Remove and repeat.

Q: How to find a vertex in the source component?
1. Run DFSAll(G).
2. Pick a vertex in the largest finish order.

Prob sketch:

\[ \text{Not possible} \]