# *Algorithm Design and Analysis*

CSE 565

## LECTURE 5

**Graphs**

- Applications of DFS
- Topological sort
- Strongly connected components

**Sofya Raskhodnikova**

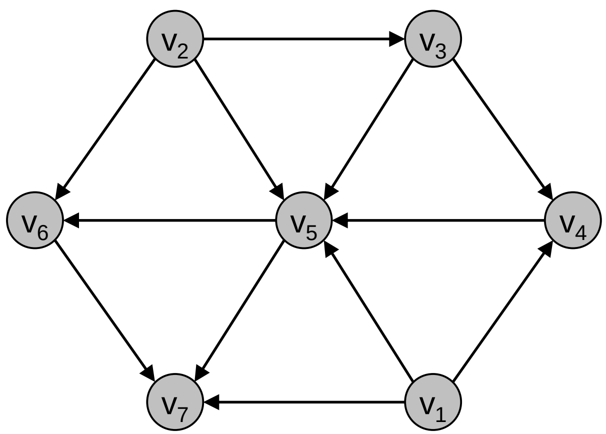
# Review

- If we run DFS on an undirected graph, can there be an edge $(u,v)$
  – where $v$ is an ancestor of $u$? ("back edge")
  – where $v$ is a sibling of $u$? ("cross edge")
- Same questions with a directed graph?
- Same questions with a BFS tree
  – directed?
  – undirected?

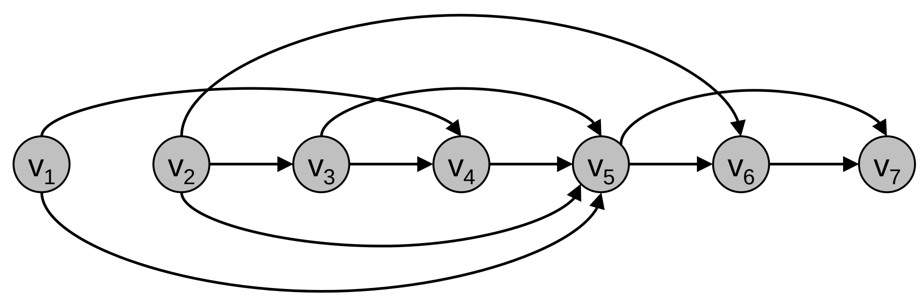
# Application 1 of DFS: Topological Sort

# Directed Acyclic Graphs

Def. A topological order of a directed graph $G = (V, E)$ is an ordering of its nodes as $v_1, v_2, \ldots, v_n$ so that for every edge $(v_i, v_j)$ we have $i < j$.




a DAG




a topological ordering

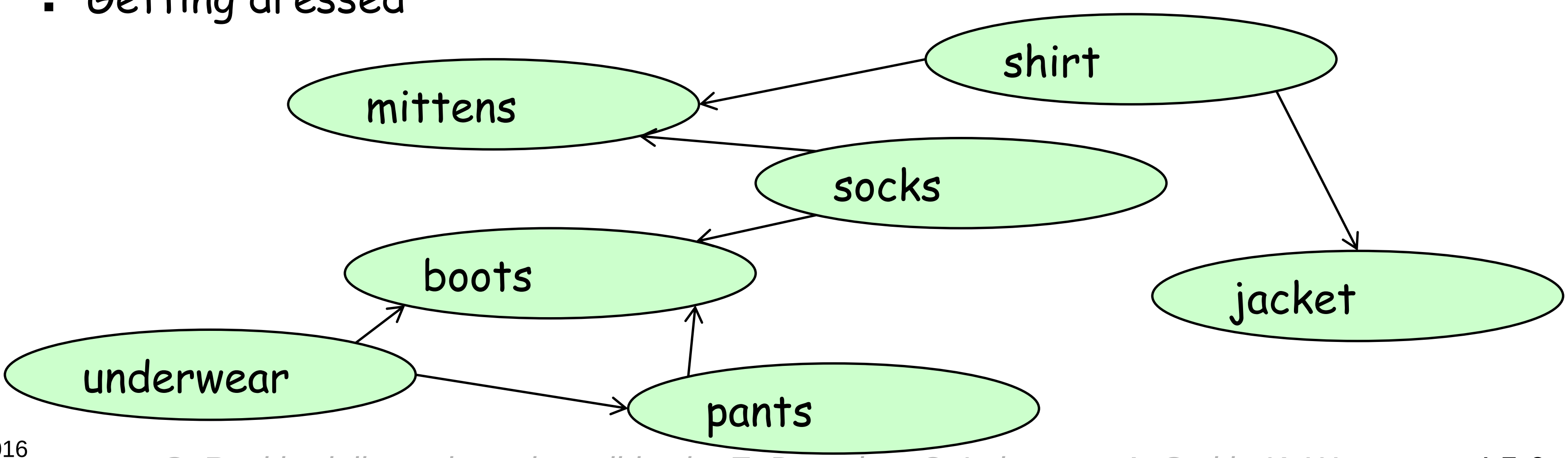
# Precedence Constraints

Def. An DAG is a directed graph that contains no directed cycles.

Typical "meaning": Precedence constraints. Edge $(v_i, v_j)$ means task $v_i$ must occur before $v_j$.

Applications.

- Course prerequisite graph: course $v_i$ must be taken before $v_j$.
- Compilation: module $v_i$ must be compiled before $v_j$. Pipeline of computing jobs: output of job $v_i$ needed to determine input of job $v_j$.
- Getting dressed



# Recall from book

- Every DAG has a topological order

- If G graph has a topological order, then G is a DAG.

# Review

- Suppose your run DFS on a DAG $\boldsymbol{G=(V,E)}$
- True or false?
  - Sorting by **discovery** time gives a topological order
  - Sorting by **finish** time gives a topological order

Proof of correctness:

**Lemma:** If $G$ is a DAG and $(u,v)$ is an edge, then $u.f > v.f$ .

Proof on board.

# Generalizations

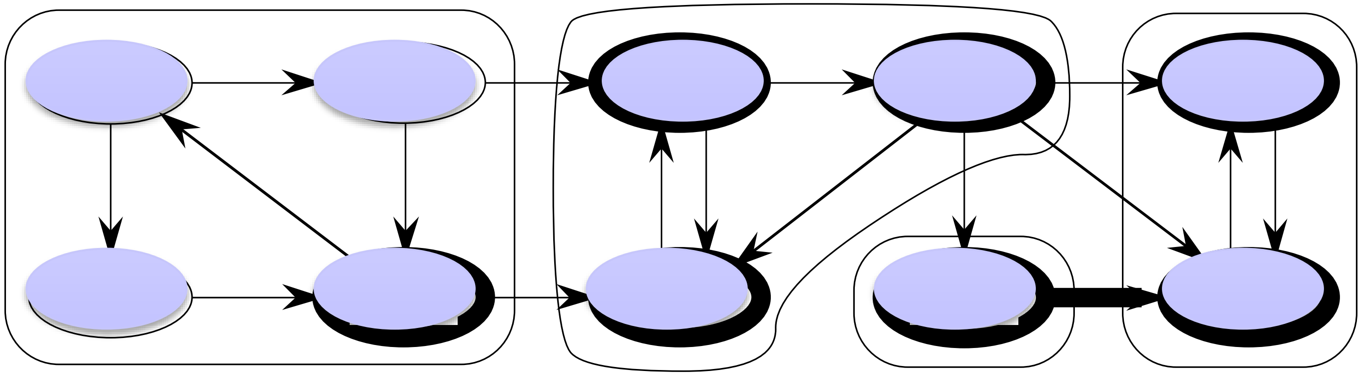
- Which of the following is always true in an arbitrary graph?
  - If $u \rightsquigarrow v$ and $v \rightsquigarrow u$ then $u.f > v.f$
  - If $u \rightsquigarrow v$ and ***not***($v \rightsquigarrow u$) then $u.f > v.f$
  - If $u.f > v.f$ then $u \rightsquigarrow v$

- **Key Lemma:** In any graph $G$, if $u \not\rightsquigarrow v$ but
  $u$ is not reachable from $v$,
  then $u.f > v.f$.
- Proof: Same as for DAGs.

# Application 2 of DFS: Strongly Connected Components

# Strongly Connected Components

- Undirected graphs:
  - $u,v$ are **connected** if there is a path between them.
- Directed graphs:
  - $u,v$ are **strongly connected** if there are paths $u \rightsquigarrow v$ and $v \rightsquigarrow u$
- SCC($u$): set of vertices strongly connected to $u$
- ***Observation:*** *Two SCC's either disjoint or equal.*



# How do we find all SCC's?

- First idea:
  - Pick a vertex $u$
  - Run DFS (or BFS) from u to find all vertices reachable from u
  - How do we find vertices that can reach u?
- Look at **reverse** graph $G^{rev}$
  - Same vertices: V
  - All edges are reversed: $(u, v)$ becomes $(v, u)$
- Run DFS or BFS in $G^{rev}$ to find all vertices that can reach $u$


S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne

# Overall algorithm

- Maintain function Comp:V→{0,…,n}
  - An array, or a field for each vertex
  - Initialize to 0 for all *v*
- $i = 1$
- For each vertex *v*
  - if *v.scc*=0
    - BFS(*G*,*v*)
    - BFS($G^{rev}$,*v*)
    - For all vertices reachable from v in both *G* and $G^{rev}$
      - *v.scc*=*i*
    - $i = i + 1$

Time $O\big(n(m + n)\big)$ in the worst case

# Fast SCC

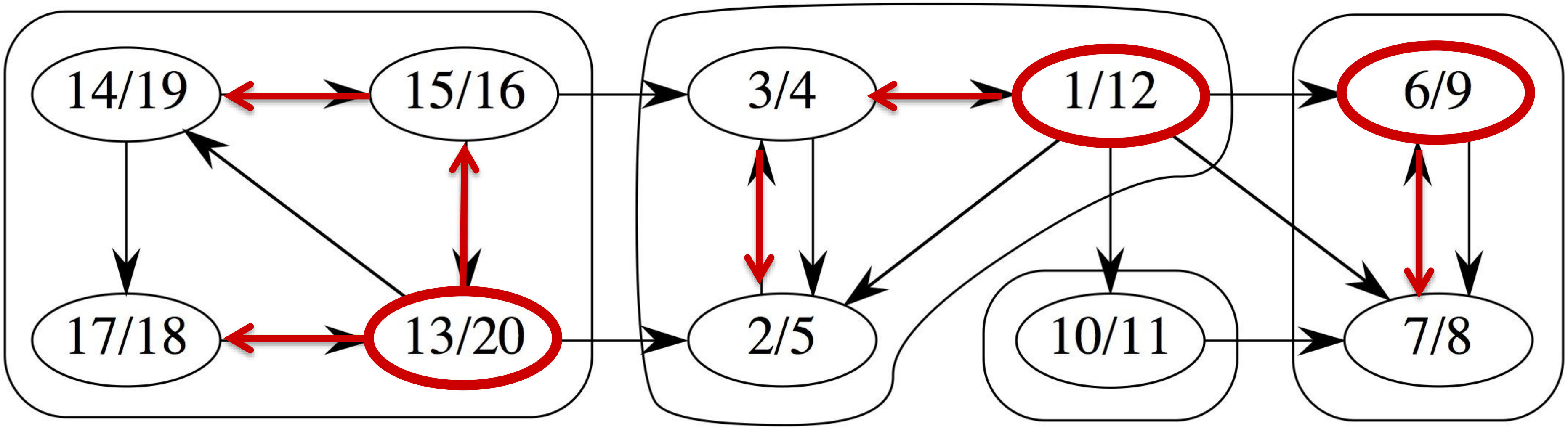
Algorithm $\text{SCC}_{\text{fast}}(G)$

- Call DFS($G$) to get finishing times *u.f* for all *u*
- Compute $G^{rev}$
- Call DFS($G^{rev}$), with one modification:
  - in main loop, consider vertices in decreasing order of *u.f*
- Output vertices of each tree in DFS forest as separate SCC

Could we use BFS…
- For the first pass (on G)?
- For the second pass (on $G^{rev}$)?

- Running time?
- Correctness?

# Example




- Numbers: discover/finish times of first DFS
- Red arrows: Forest of DFS($G^{rev}$)
- Red ovals: roots of second DFS forest

# Proof of Correctness

- Fix graph $G$ on $n$ vertices
- For each SCC $C$ in G, define
  - $f(C)$ = latest finish time (from first DFS) in $C$
- Order the SCC's $C_1$, $C_2$, … in decreasing order of $f(C)$

**Theorem:** The algorithm outputs each of the $C_i$ correctly.

- Proof by induction on $i$
- $i = 1$: Second DFS will start at a vertex $x$ in $C_1$
  - **There are no edges in $G^{rev}$ leaving $C_1$ (by key lemma)**
  - So DFS-Visit($x$) will visit exactly the vertices of $C_1$
- For $i > 1$:
  - Suppose $C_1, C_2, \ldots C_{i-1}$ are correctly output. Then
    - $i$th DFS call starts from within $C_i$.
    - All vertices of $C_i$ will be reached.
    - **Edges in $G^{rev}$ only leave $C_i$ towards $C_j$ with $j < i$.**
  - So $C_i$ is output correctly. QED.

# Exercise

- Consider the SCC graph $G_{SCC}$ of G:
  - vertices are SCC's of G
  - edge (C,C') means G has an edge (u,v) with u in C and v in C'
- Prove that $G_{SCC}$ is a DAG.

# Exercise

Consider the following modification to the algorithm for SCC:

- Use G instead of $G^{rev}$ in 2nd DFS, but scan vertices in order of increasing finish times from the 1st DFS.

Is this algorithm correct?