W, 09/18/19

Fall'19 CSCE 629

Analysis of Algorithms

Fang Song

Texas A&M U

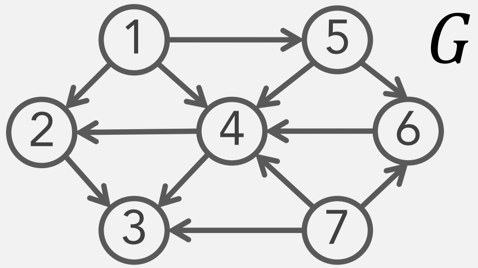
Lecture 8

- Connectivity in directed graph
- Topological sort

Credit: based on slides by A. Smith & K. Wayne

Directed graphs

- $G = (V, E)$
  - Edge $u \to v$ leaves node u and enters node v
  - Adjacency matrix (asymmetric)
  - Adjacency list: track out-going edges (or two for in and out)



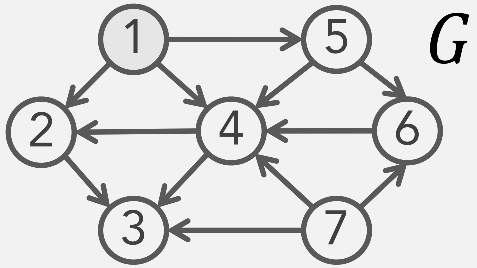
… $Adj_{out}[2] = \{3\}, Adj_{in}[2] = \{1,4\}$

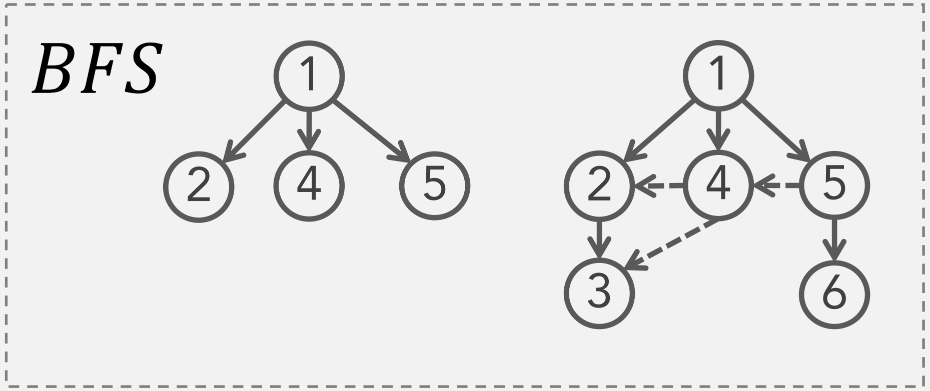
- Some examples

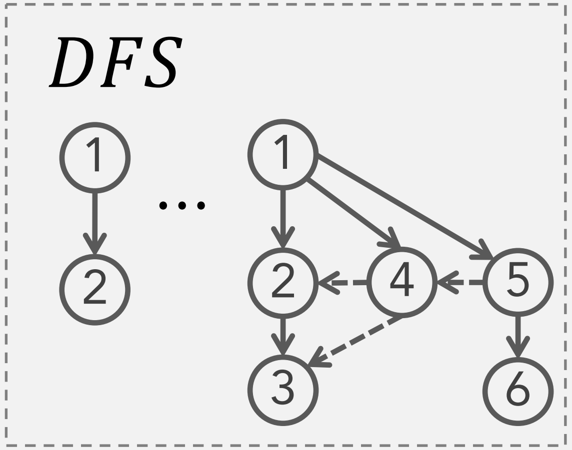
Directed graph	Node	Directed edge
transportation	Street intersection	One-way street
web	webpage	hyperlink
scheduling	task	prereq
cell phone	person	call
citation	article	Citation

Connectivity in directed graphs

- Directed reachability. Find all nodes reachable from a node s.
  - BFS/DFS applies naturally
  - $s \rightsquigarrow t$: there is a path from s to t. Need not be $t \rightsquigarrow s$







- Application: web crawler.
  - Start from web page s. Find all web pages linked from s, either directly or indirectly.

Strong connectivity

- **Def.** u and v are mutually reachable ($u \leftrightsquigarrow v$) if $u \rightsquigarrow v$ & $v \rightsquigarrow u$.
- **Obs.** If $u \leftrightsquigarrow v$ and $v \leftrightsquigarrow w$, then $u \leftrightsquigarrow w$.

- **Def.** A graph is strongly connected if every pair of nodes is mutually reachable.

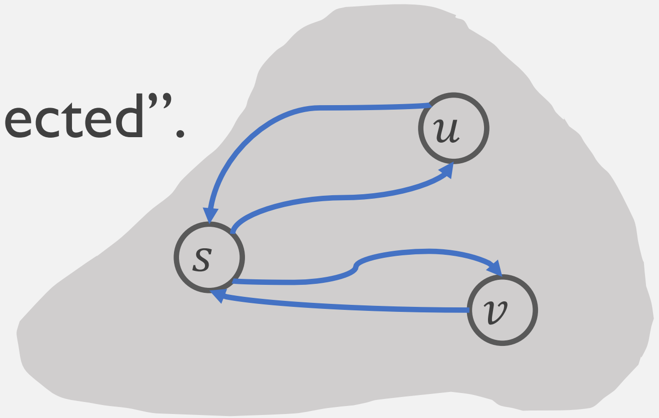
Lemma. Let s be any node. G is strongly connected iff. every node is reachable from s, and s is reachable from every node.

Proof. [Show both "if" and "only if"]

- $\Rightarrow$ (only if) Follows from definition of "strongly connected".
- $\Leftarrow$ (if) for any two nodes u, v:

  $u \rightsquigarrow v$ by following $u \rightsquigarrow s$ then $s \rightsquigarrow v$

  $v \rightsquigarrow u$ by following $v \rightsquigarrow s$ then $s \rightsquigarrow u$



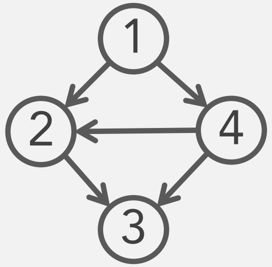
Testing strong connectivity

Theorem. There is an $O(m+n)$ time algorithm that determines if G is strongly connected.

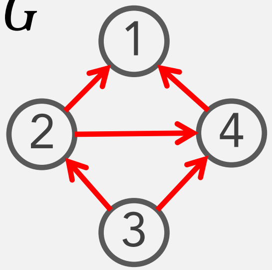
Proof (construction of an algorithm; fill in the analysis on your own)

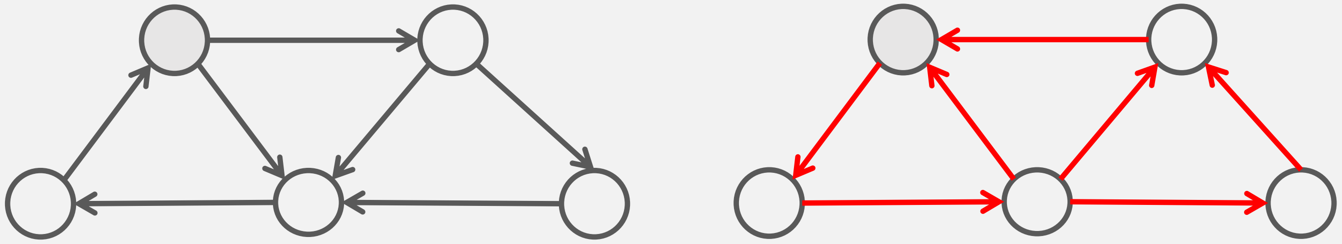
1. Pick any node s
2. Run **BFS** from s on G
3. Run **BFS** from s on G^{rev}
4. Return true if all nodes reached in both **BFS** runs

G



G^{rev}: reverse orientation of every edge in G

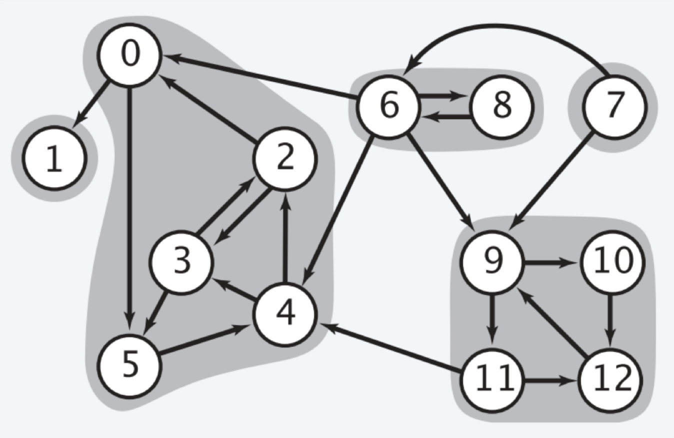




Strongly connected

Strong components

- **Def.** A strong component is a maximal subset of mutually reachable nodes.
- **Obs.** For any two nodes s and t in a directed graph, their strong components are either identical or disjoint.



Theorem. There is an $O(m+n)$ time algorithm that finds all strong components.

SIAM J. COMPUT.
Vol. 1, No. 2, June 1972

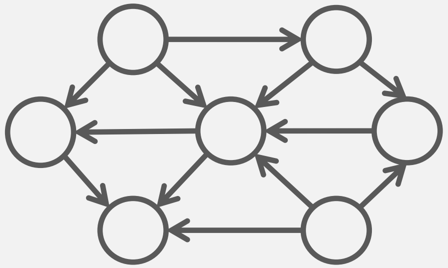
DEPTH-FIRST SEARCH AND LINEAR GRAPH ALGORITHMS*

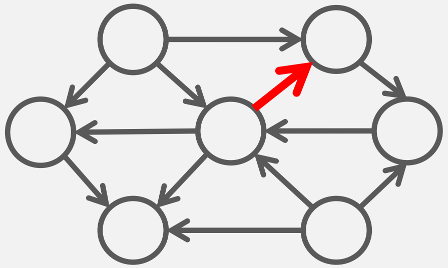
ROBERT TARJAN†

Abstract. The value of depth-first search or "backtracking" as a technique for solving problems is illustrated by two examples. An improved version of an algorithm for finding the strongly connected components of a directed graph and an algorithm for finding the biconnected components of an un-direct graph are presented. The space and time requirements of both algorithms are bounded by

Directed acyclic graphs (DAG)

- Def. A DAG is a directed graph that contains no directed cycles.

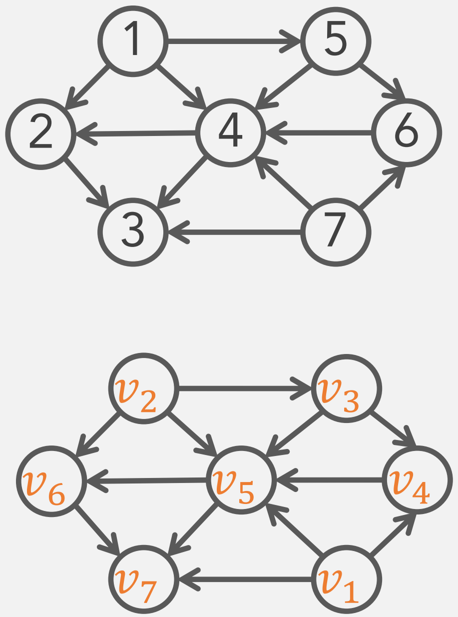


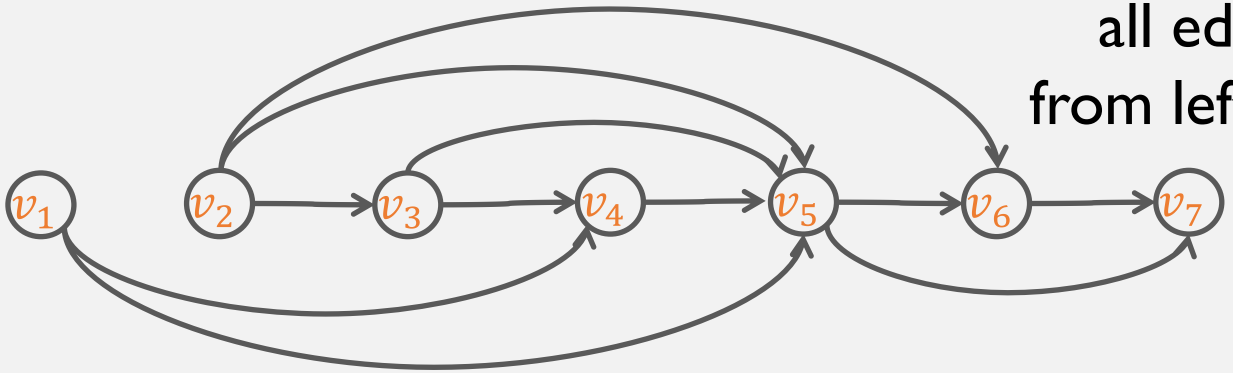


- Application: precedence constrains.
  - Course prerequisite: course i must be taken before j
  - Compilation: module i must be compiled before j
  - Pipelein of computing jobs: output of job i determins input of job j

Topological order

- Def. A topological order of a directed graph is an ordering of its nodes as $v_1, \ldots, v_n$, so that for every edge $v_i \to v_j$ we have $i < j$.





all edges go from left to right

A topological order

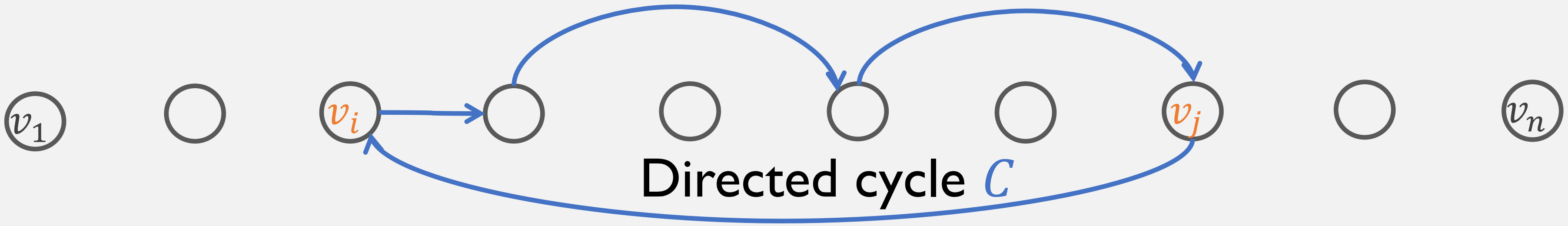
1. Does every DAG have a topological order?
2. If G has a topological order, is G necessarily a DAG?

2. If G has a topological order, is G necessarily a DAG?

Lemma2. If G has a topological order, then G is a DAG.

Proof. [by contradiction]

- Suppose G has a topological order $v_1, \dots, v_n$; and G also has a directed cycle C.
- Let v_i be the lowest-indexed node in C, v_j be the node right before v_i (in C); thus $v_j \rightarrow v_i$ is an edge.
- By our choice $i < j$.
- But $v_1, \dots, v_n$ a topological order; if $v_j \rightarrow v_i$ an edge, then $j < i$. Contradiction!



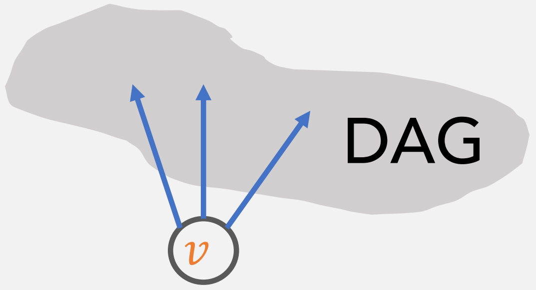
1. Does every DAG have a topological order?

Lemma1. A DAG G has a node with no entering edges.

Corollary. If G is a DAG, then G has a topological order.

Proof (of corollary) given Lemma1. [by induction]

- Base case: true if $n = 1$.
- Given DAG on $n > 1$ nodes, find a node v with no entering edges [Lemma1]
- $G - \{v\}$ is a DAG, since deleting v cannot create cycles.
- By indictive hypothesis, $G - \{v\}$ has a topological ordering.
- Place v first; then append nodes of $G - \{v\}$ in topological order. [valid because v has no entering edges]



Topological sorting algorithm

TopSort(G)
// $Count(w)$ = remaining number of incoming edges
// S = set of remaining nodes with no incoming edges
// V[1,...,n] topological order
1. Initialize S and $Count(\cdot)$ for all nodes
2. For $v \in$ S
   Append v to V
   For all w with $v \to w$ // delete v from G
      $Count(w) --$
      If $Count(w) == 0$, add w to S

$O(n+m)$ a single scan of adjacency list

$O(1)$ per edge

Theorem. TopSort computs a topological order in $O(n+m)$ time

Completing the argument

Lemma1. A DAG G has a node with no entering edges.

Proof. [by contradiction]

- Suppose G is a DAG, and every node has at least one entering edge.
- Pick any node v, and follow edges backward from v. Repeat till we visit a node, say w twice. ($v \leftarrow u \leftarrow x \cdots \leftarrow w \cdots \leftarrow w$)
- Let C be the sequence of nodes between successive visits to w.

→ C is a cycle! Contradiction!