Portland State University

W'21 CS 584/684
Algorithm Design & Analysis

Fang Song

Lecture 6

- Topological order cont'd
- Shortest/longest path in DAGs
- Dynamic programming intro

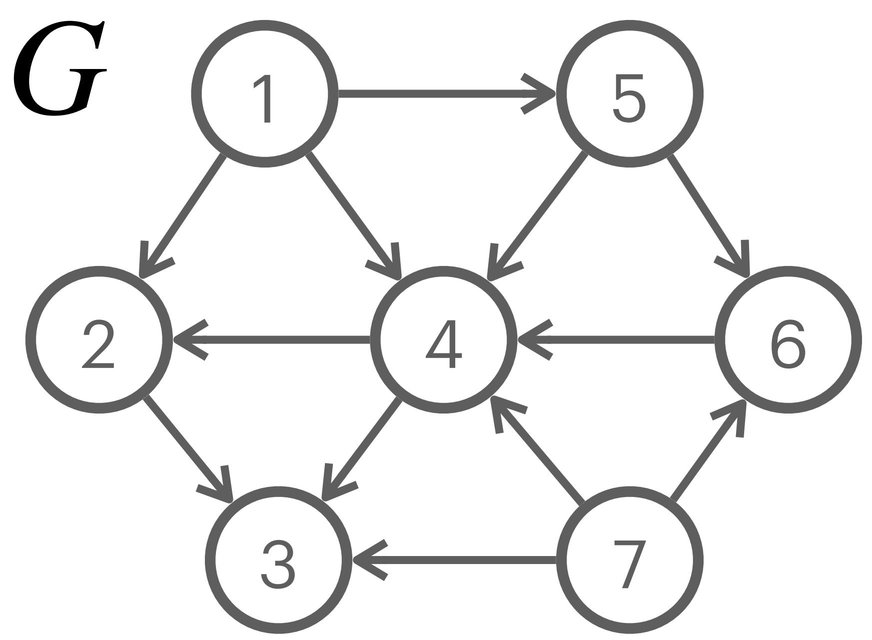
Credit: based on slides by K. Wayne

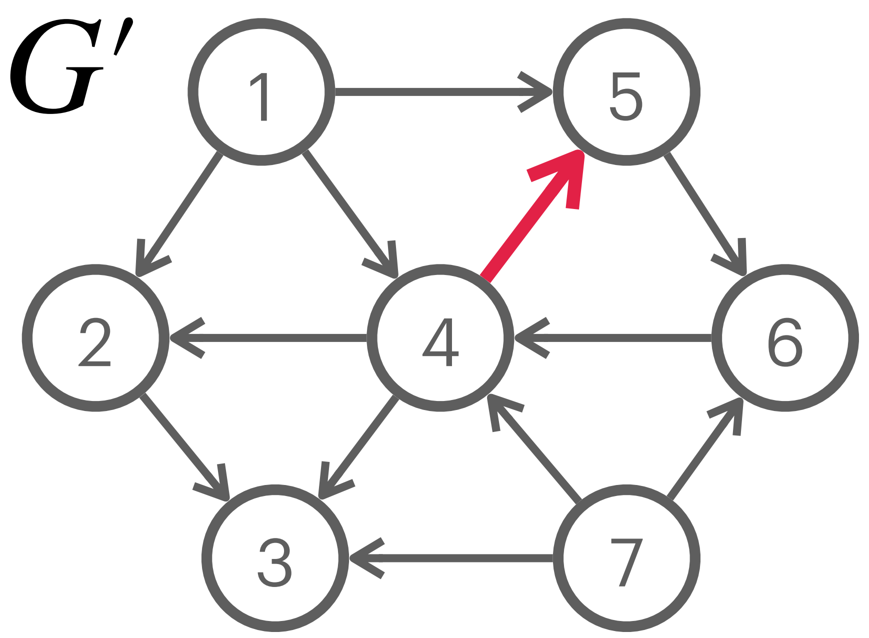
Exercise

- **Let G be a graph with n vertices and m edges. Which of the following statements are TRUE?**
  - **BFS/DFS** always run in time $O(m + n)$.
  - If G is undirected, the connected components of two vertices can be identical.
  - If G is directed, the strong components of two vertices can be neither identical nor disjoint.
  - There is an algorithm to test strong connectivity of directed G in time $o(n(m + n))$ in the worst-case.

Review: Directed acyclic graphs (DAG)

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

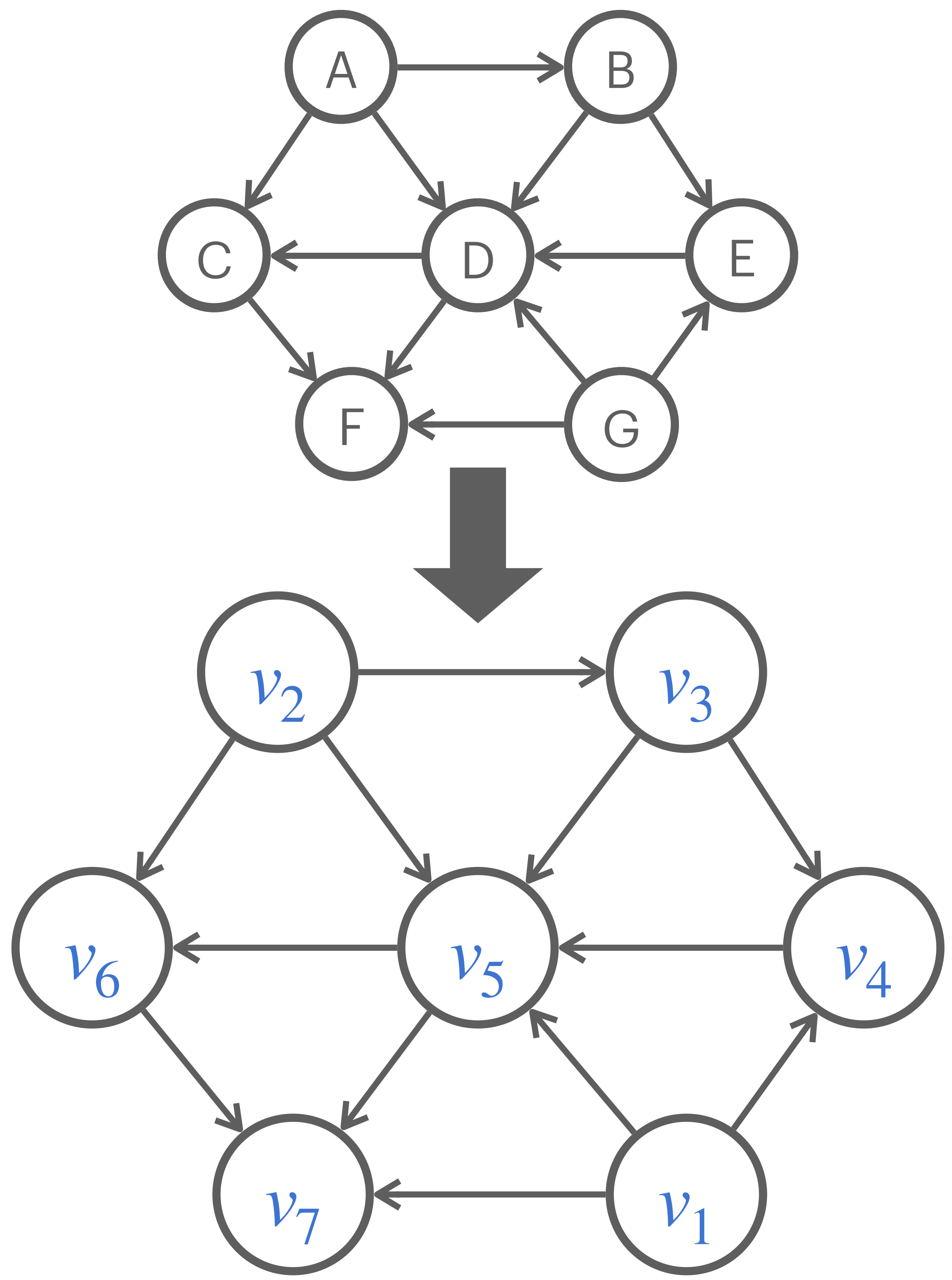




- **Application:** precedence constraints.
  - Course prerequisite: 350 must be taken before 584/684.
  - Compilation: module i must be complied before j.
  - Pipeline of computing jobs: output of job i determines input of job j.

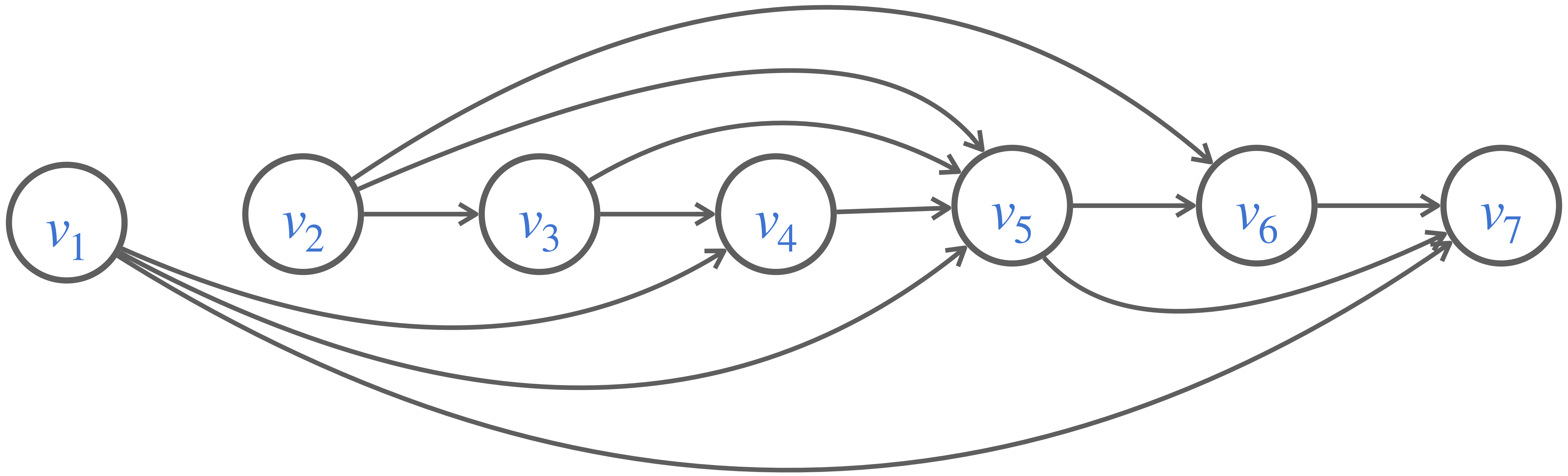
Topological order

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



A topological order

All edges go from left to right



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

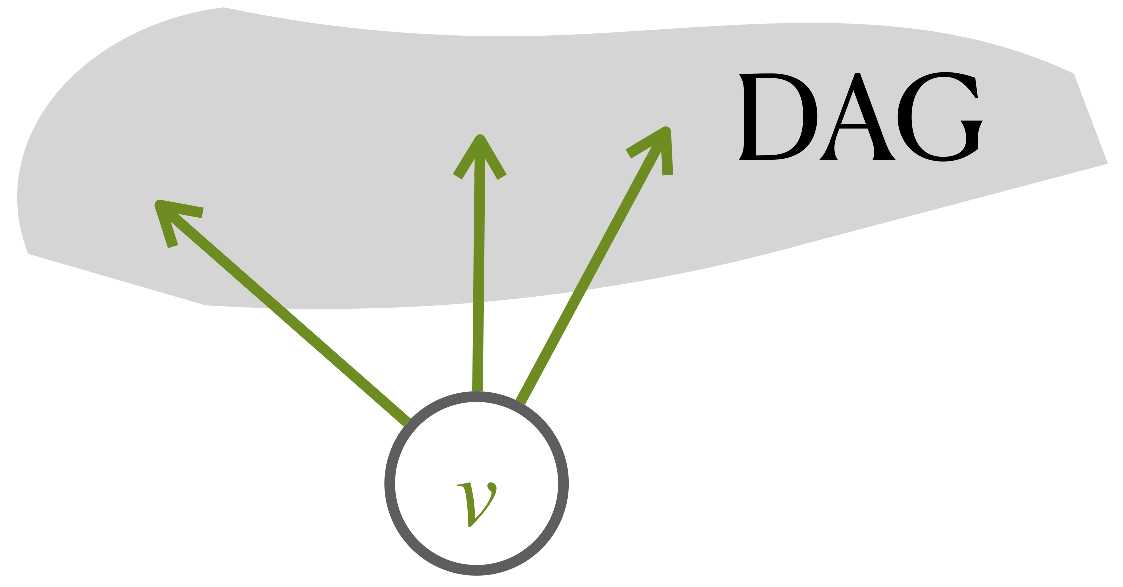
2. Does every DAG have a topological order?

Q2: Dose every DAG have a topological order?

Lemma 2. 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 Lemma 1 [by induction on number of nodes]
  - Base case: true if $n = 1$.
  - Given a DAG on $n > 1$ nodes, find a node v with no entering edges [Lemma 2].
    - $G - \{v\}$ is a DAG, since deleting v cannot create cycles.
  - Induction hypothesis, $G - \{v\}$ (with $n - 1$ nodes) has a topological order.
  - 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
// $V[1,\ldots,n]$ 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

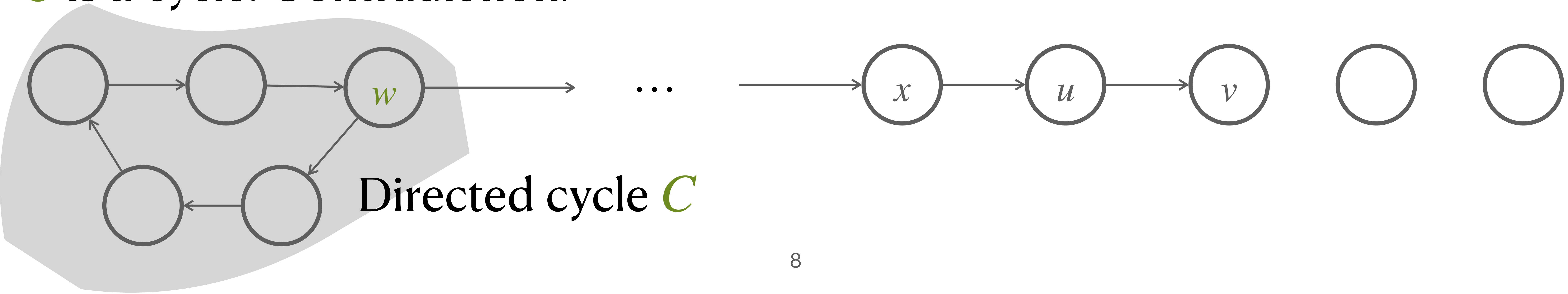
$O(1)$, run once per edge

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

Completing the proof

Lemma 2. A DAG G has a node with no entering edges.

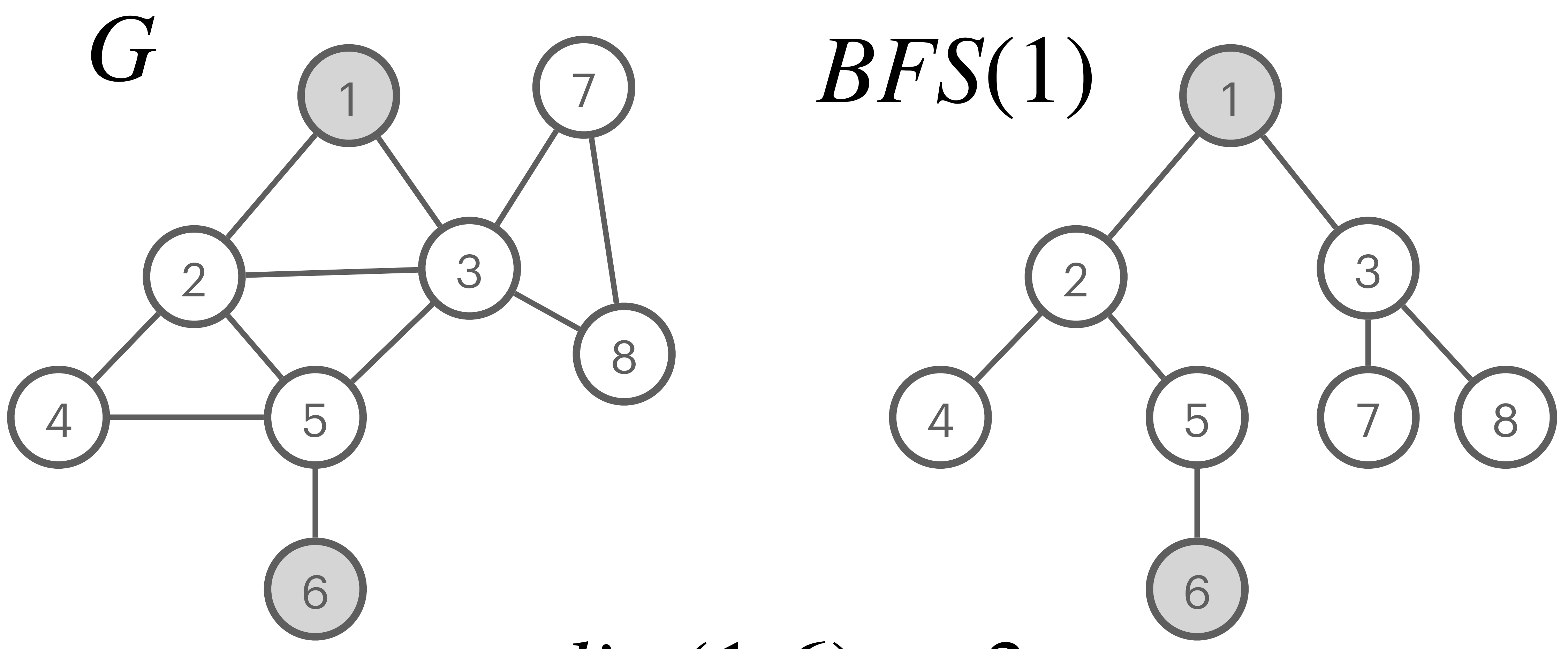
- **Proof** [by contradiction]
  - Suppose G is a DAG, and every node has at least one entering edge.
  - Pick any node v, and follow edges backwards from v.
  - Continue till we visit a node, say w, twice. ($v \leftarrow u \leftarrow x\ldots \leftarrow w\ldots \leftarrow w$)
  - Let C be the sequence of nodes between successive visits to w.
  - C is a cycle. Contradiction!



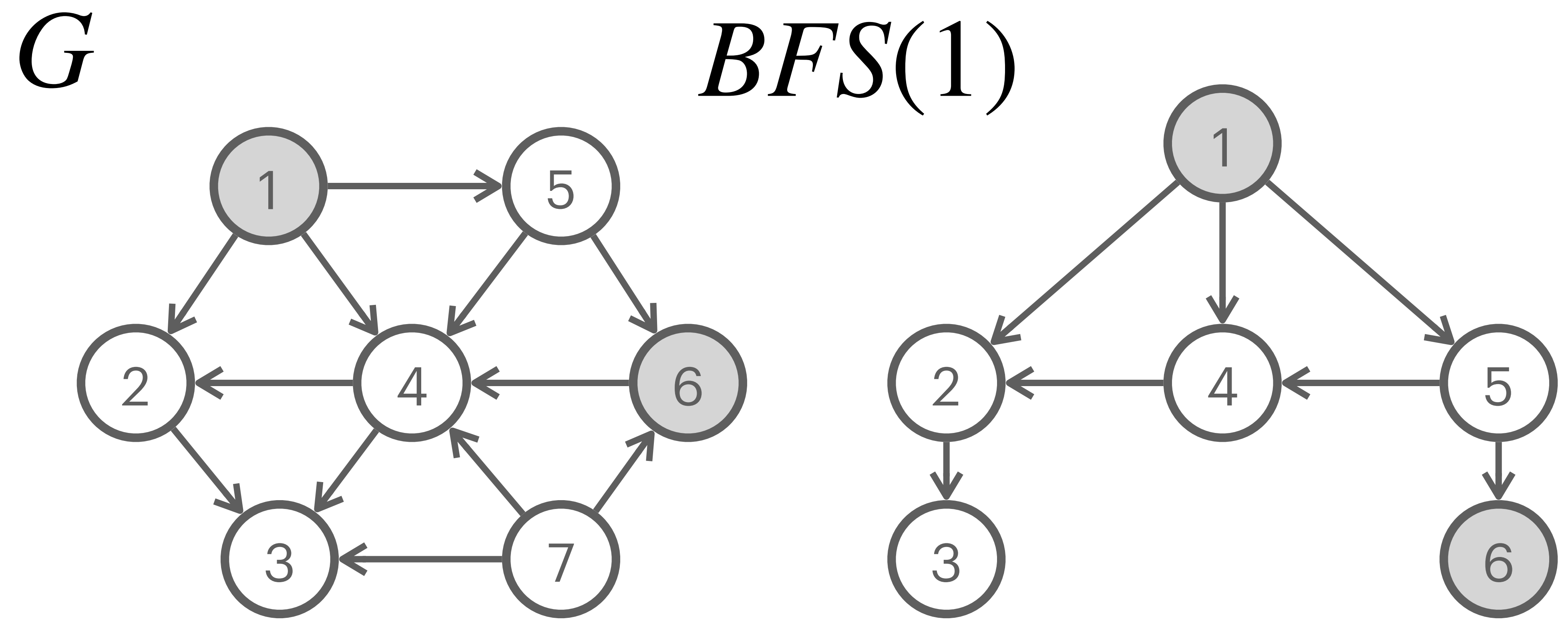
Shortest path in a graph

Input: Graph G, nodes s and t .

Output: $dist(s, t)$.



$dist(1,6) = 3$



$dist(1 \rightarrow 6) = 2$

Shortest path in a weighted graph

- **Weighted graphs**
  - Every edge has a length ℓ_e.
  - Length of a path $\ell(P) = \sum_{e \in P} \ell_e$.
  - Distance $dist(s,t) = \min_{P: u \rightsquigarrow v} \ell(P)$.

- **Length function:** $\ell : E \to \mathbb{Z}$
  - $\ell(u,v) = \infty$ if not an edge
  - Model time, distance, cost ...
  - Can be negative: fund transfer, heat in chemistry reaction ...

- $\forall e \in E, \ell(e) = 1$: **BFS solves it.**
- **How to solve weighted case?**

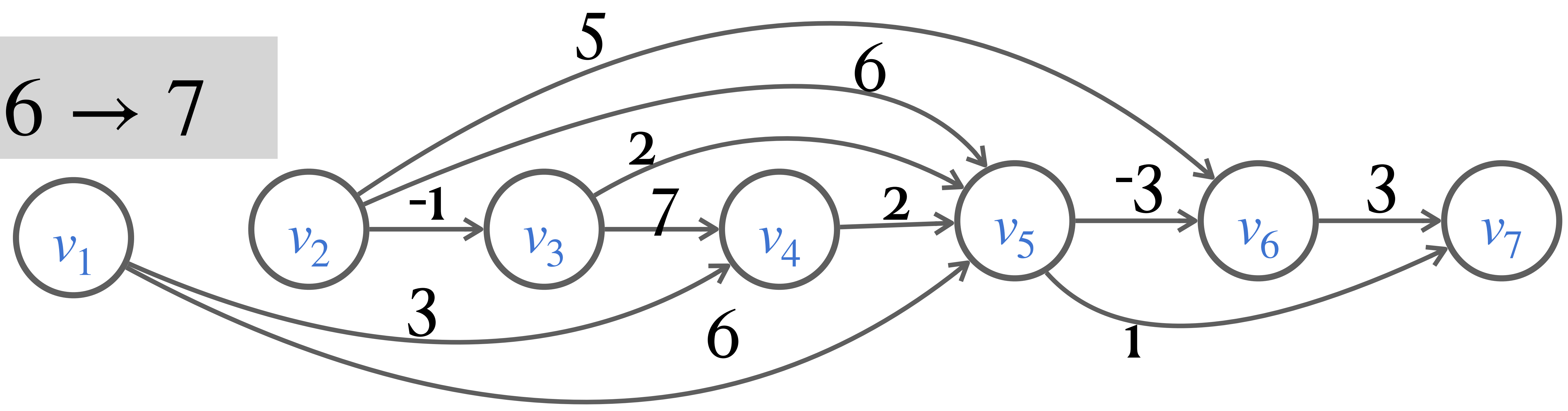
Shortest path in DAGs

Input: DAG G, length function ℓ, nodes s and t .

Output: $d(t) := dist(s, t)$.

- **Example. What is $dist(1 \to 7)$?**

$$dist(1 \to 7) = 5, 1 \to 4 \to 5 \to 6 \to 7$$



$$d(7) = \min\{d(6) + 3, d(5) + 1\}$$

$$d(6) = \min\{d(5) - 3, d(2) + 5\}$$

$$d(5) = \min\{d(4) + 2, d(3) + 2, d(2) + 6, d(1) + 6\}$$

$$d(4) = \min\{d(3) + 7, d(1) + 3\}$$

$$d(2) = \infty$$

$$d(3) = d(2) - 1$$

$$d(1) = 0$$

Shortest path in DAGs: algorithm

- **Key observations**
  - Reduce to subproblems $d(6), d(5), \ldots$
  - Subproblems overlap: e.g., $d(6)$, $d(5)$ both involve $d(2)$.
  - An ordering of subproblems (DAG: edges go left to right)

Distance(G, s):
// Initialize all $d(\cdot) = \infty$
1. $d(s) = 0$
2. For $v \in V - \{s\}$ in topological order
   $d(v) = \min_{u \to v}\{d(u) + \ell(u, v)\}$

$d(7) = \min\{d(6) + 3, d(5) + 1\}$

$d(6) = \min\{d(5) - 3, d(2) + 5\}$

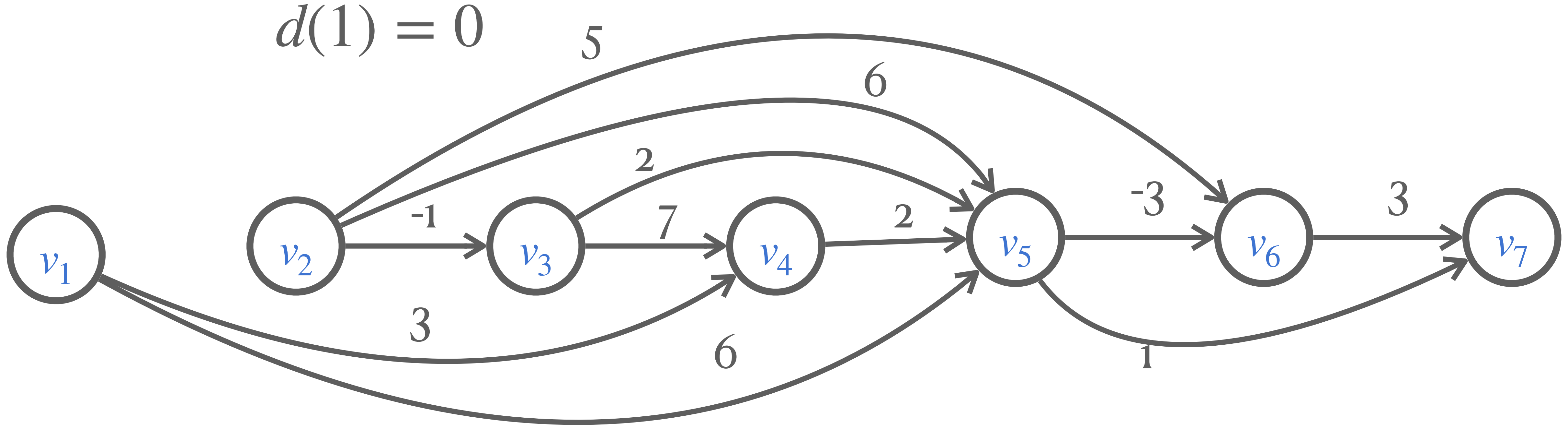
$d(5) = \min\{d(4) + 2, d(3) + 2, d(2) + 6, d(1) + 6\}$

$d(4) = \min\{d(3) + 7, d(1) + 3\}$

$d(3) = d(2) - 1$

$d(2) = \infty$

$d(1) = 0$



Algorithm design arsenal

- **Dynamic programming**
  - Break up a problem into a series of overlapping subproblems.
  - Combine solutions to smaller subproblems to form a solution to large problem.

  An implicit DAG: nodes = subproblems, edges = dependencies

- **Divide-&-Conquer**
  - Break up a problem into a series of independent subproblems, typically of much smaller size.
  - Combine solutions to smaller subproblems to form a solution to large problem.

Longest increasing subsequences

Input: a sequence of numbers $a_1, \ldots, a_n$.

Output: a **longest increasing** subsequence $a_{i_1}, \ldots, a_{i_k}$.

- $a_{i_1} < a_{i_2} < \ldots < a_{i_k} \quad (1 \leq i_1, \ldots, i_k \leq n)$

5 2 8 6 3 6 9 7

Brute-force algorithm

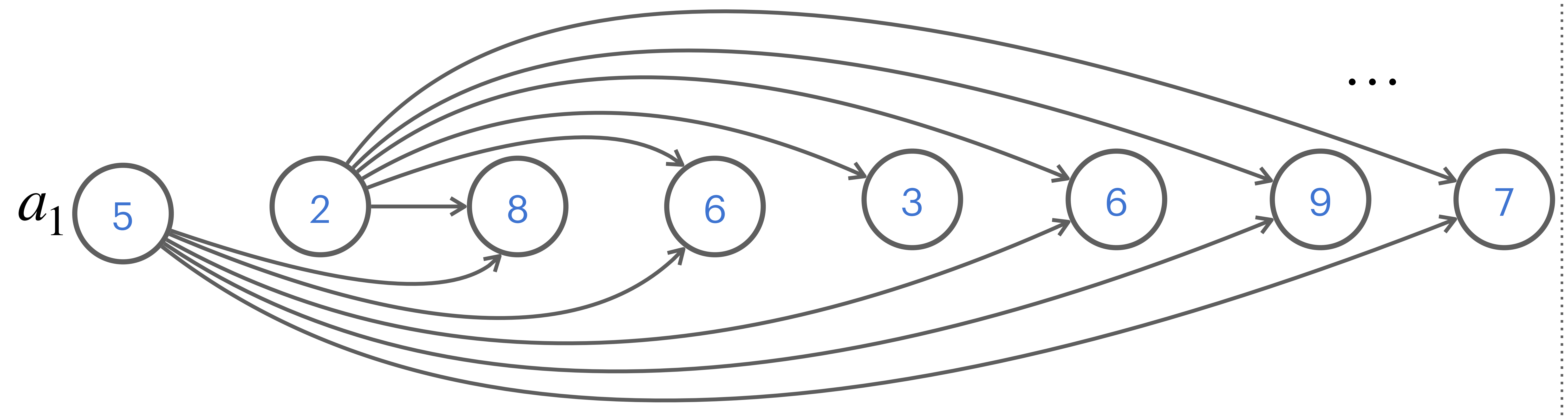
- For each $1 \leq k \leq n$, check if exists an increasing subsequence of length k.
- Running time: $\Omega(2^n)$

Dynamic programming approach

Input: a sequence of numbers $a_1, \ldots, a_n$.

Output: a longest increasing subsequence $a_{i_1}, \ldots, a_{i_k}$.

- Form a DAG G: if $a_i \leq a_j$, add an edge $i \to j$.



Increasing subsequence ⇔ path in G

Amounts to finding a longest path in the DAG

Longest increasing subsequence / longest path

Input: a sequence of numbers $a_1, \ldots, a_n$.

Output: a longest increasing subsequence $a_{i_1}, \ldots, a_{i_k}$.

LSeq(a):
// Initialize all $L(j) = 1$; length of longest path ending at j.
1. For $j = 1, 2, \ldots, n$
   $L(j) = \max_{i \to j} \{1 + L(i)\}$
2. Return $\max_j L(j)$

- **Running time:** $O(n + m) = O(n^2)$.
  - What is the worst case scenario?
- **Can you output the subsequence?**

Recap on DP

- There is an ordering on the subproblems.
- A relation showing how to solve a subproblem given answers to smaller subproblems (= those appear earlier in the ordering).

Scratch