F, 09/20/19

Fall'19 CSCE 629

Analysis of Algorithms

Fang Song
Texas A&M U

Lecture 9

- Topological sort cont'd
- Dynamic programming

Credit: based on slides by K. Wayne

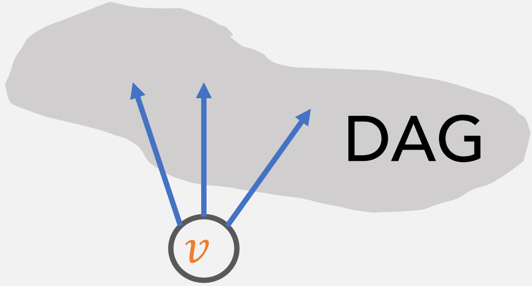
1. Does every DAG have a topological order?

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 inductive 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

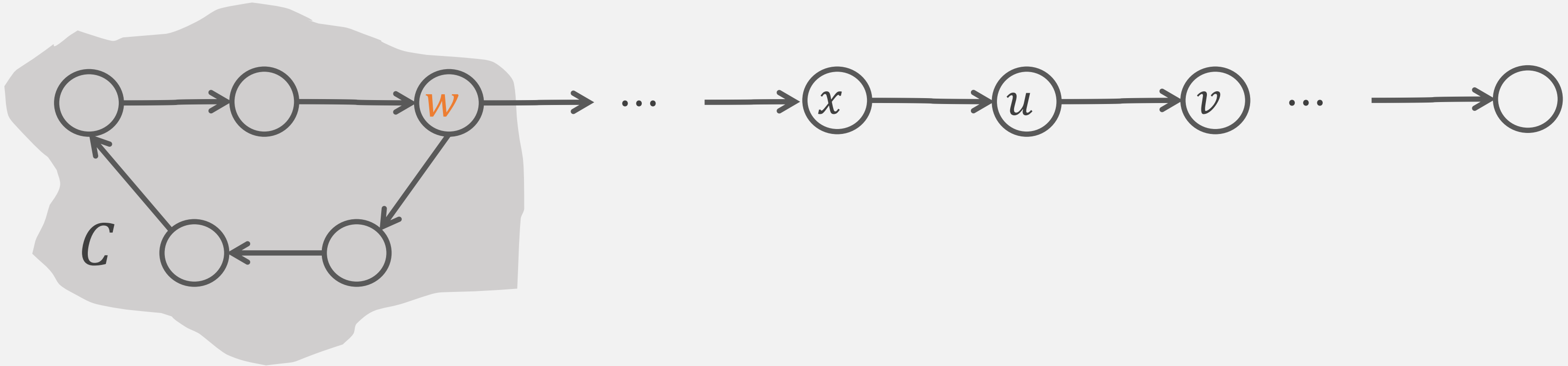
Completing the argument

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

Proof. [by contradiction]

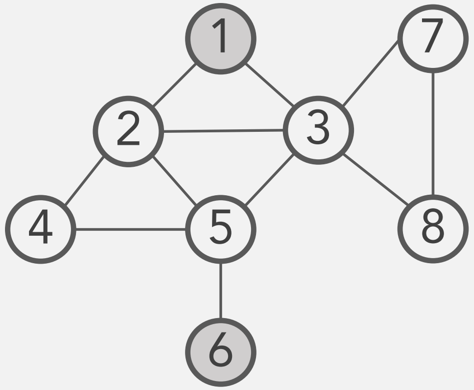
- Suppose G is a DAG, and every node has at least one entering edge.
- Pick any node v, and follow edges backward from v. Repeat till we visit a node, say w twice. ($v \leftarrow u \leftarrow x \cdots \leftarrow w \cdots \leftarrow w$)
- Let C be the sequence of nodes between successive visits to w.

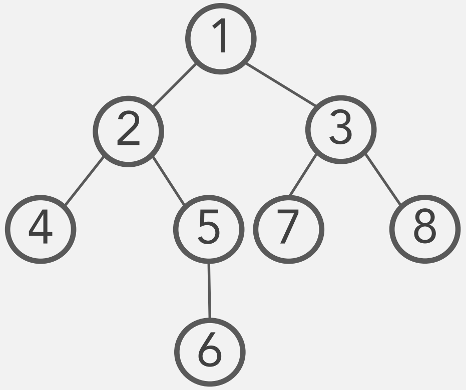
→ C is a cycle! Contradiction!



Shortest path in a graph

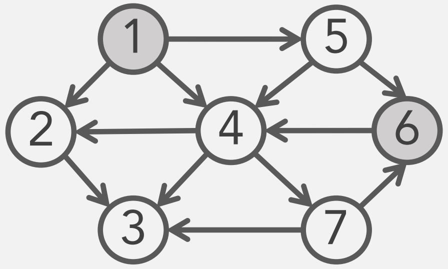
- Input: graph G, nodes s and t.
- Output: $dist(s,t)$

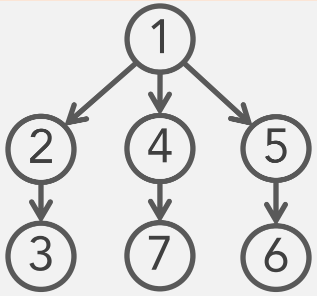




$dist(1,6) = 3$

BFS solves it!





$dist(1 \rightarrow 6) = 2$

Weighted graphs

- Every edge has a length l_e
- Length of a path $l(P) = \sum_{e \in P} l_e$
- Distance $dist(u,v) = \min_{P: u \rightsquigarrow v} l(P)$

$\forall e \in E, l_e = 1$

Length function $l: E \rightarrow \mathbb{Z}$

- $l(u,v) = \infty$ if not an edge
- Model time, distance, cost …
- Can be negative, e.g., fund transfer, heat in chemistry reaction …

How to solve weighted case?

Shortest path in DAGs

- Input: DAG G, length l, nodes s and t
- Output: $d(t) \coloneqq dist(s,t)$

// Initialize all $d(\cdot) = \infty$

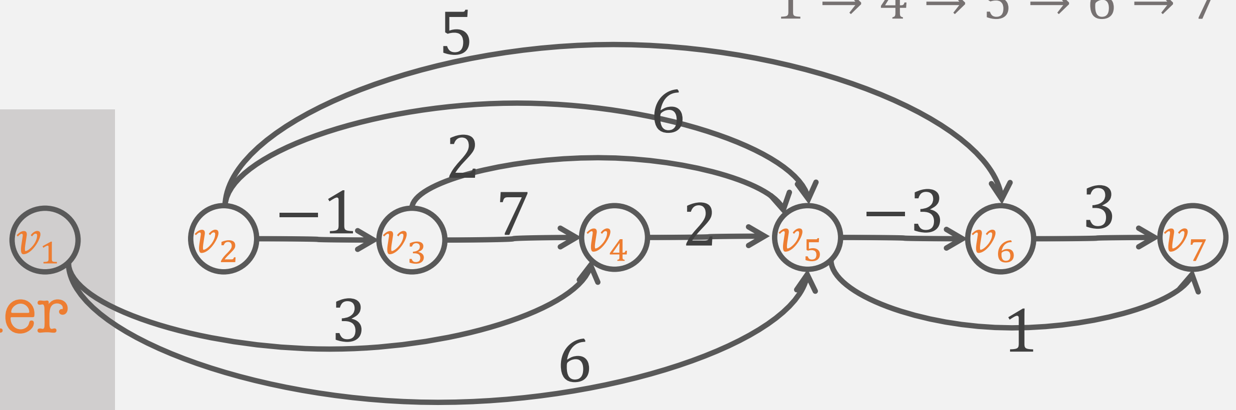
1. $d(s) = 0$
2. For $v \in V \backslash \{v\}$ in topological order

$$d(v) = \min_{(u,v)\in E} \{d(u) + l(u,v)\}$$

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

$dist(1 \to 7) = 5$

$1 \to 4 \to 5 \to 6 \to 7$



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

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

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

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

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

$d(2) = \infty$

$d(1) = 0$

Algorithm design arsenal

- Dynamic Programming. Break up a problem into a series of overlapping subproblems; combine solutions to smaller subproblems to form solution to large subproblem.

  An implicit DAG: nodes=subproblems, edges = dependencies

- Divide-&-Conquer. Break up a problem into independent (typically significantly smaller) subproblems; combine solutions to subproblems to form solution to original problem.

Longest increasing subsequences

- Input: a sequence of numbers $a_1, \dots, a_n$
- Output: a longest increasing subsequence $a_{i_1}, \dots, a_{i_k}$
  - $a_{i_1} < a_{i_2} < \cdots < a_{i_k}$ $(1 \le i_1, \dots, i_k \le n)$

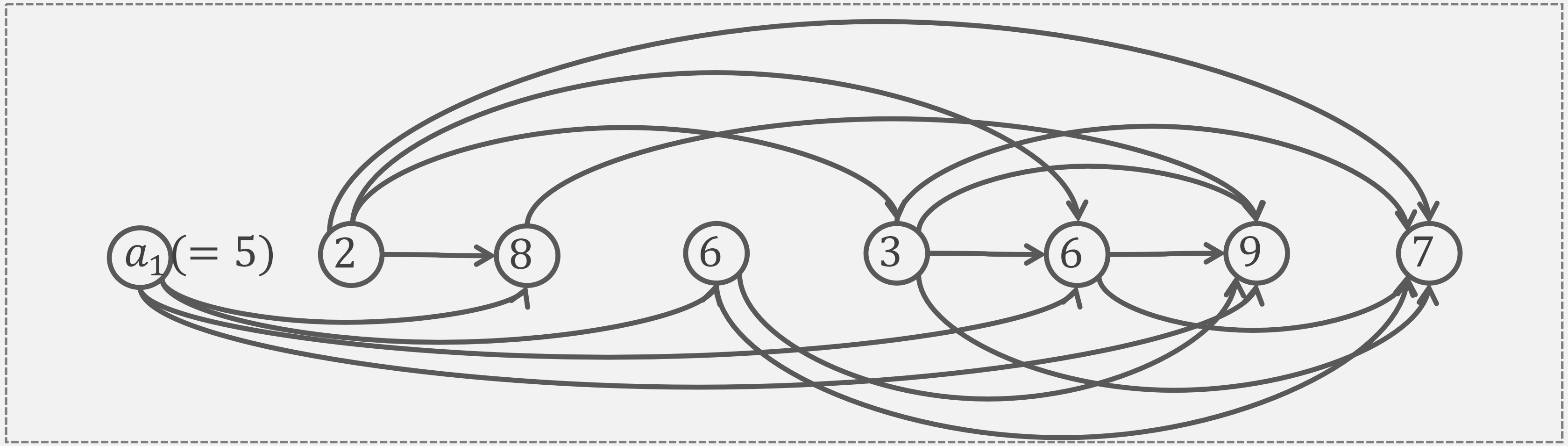
$$5 \quad 2 \quad 8 \quad 6 \quad 3 \quad 6 \quad 9 \quad 7$$

- Brute-force algorithm
  - For each $1 \le k \le n$, check if exists an increasing subsequence of length k
  - $\Omega(2^n)$ …

DP for longest increasing subsequences

- Input: a sequence of numbers $a_1, \dots, a_n$
- Output: a longest increasing subsequence $a_{i_1}, \dots, a_{i_k}$

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



Increasing subsequence $\Leftrightarrow$ path in G

➔Reduced to finding a **longest** path in the DAG!

Longest increasing subsequences/longest path

- Input: a sequence of numbers $a_1, \dots, a_n$
- Output: a longest increasing subsequence $a_{i_1}, \dots, a_{i_k}$

// Initialize all $L(j) = 1$; length of longest path ending at j

1. For $j = 1,2, \dots, n$

   $L(j) \;= 1 + \max\{L(i) : (i, j) \in E\}$
2. Return $\max_j L(j)$

Recap on DP

There is an ordering on the subproblems, and a relation showing how to solve a subproblem given answers to "smaller" subproblems (i.e., those appear earlier in the ordering)

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

Dynamic Programming history

- Richard Bellman
  - DP [1953]
  - B-Ford alg. for general shortest path (stay tuned!),
  - Curse of dimensionality…

THE THEORY OF DYNAMIC PROGRAMMING

RICHARD BELLMAN

1. **Introduction.** Before turning to a discussion of some representative problems which will permit us to exhibit various mathematical features of the theory, let us present a brief survey of the fundamental concepts, hopes, and aspirations of dynamic programming.

To begin with, the theory was created to treat the mathematical problems arising from the study of various multi-stage decision processes, which may roughly be described in the following way: We have a physical system whose state at any time *t* is determined by a set of quantities which we call state parameters, or state variables.

- Etymology
  - Dynamic programming = planning over time
  - Secretary of Defense was hostile to mathematical research
  - Bellman sought an impressive name to avoid confrontation

"it's impossible to use dynamic in a pejorative sense"
"something not even a Congressman could object to"

Reference: Bellman, R. E. Eye of the Hurricane, An Autobiography.

Dynamic Programming applications

Indispensable technique for optimization problems

Many solutions, each has a value

Goal: a solution w. optimal (min or max) value

- Areas
  - Computer science: theory, graphics, AI, compilers, systems, …
  - Bioinformatics
  - Operations research, information theory, control theory
- Some famous DP algorithms
  - Avidan–Shamir for seam carving
  - Unix diff for comparing two files
  - Viterbi for hidden Markov models
  - Knuth–Plass for word wrapping text in TeX.
  - Cocke–Kasami–Younger for parsing context-free grammars