Lecture 8: Dynamic Programming II
Shortest Paths

Overview

- Shortest paths finding using DP
- Dynamic programming principles overview

Dynamic Programming Principles

DP ~ "smart brute force"

DP ~ recursion + memoization

DP ~ shortest path in some DAG

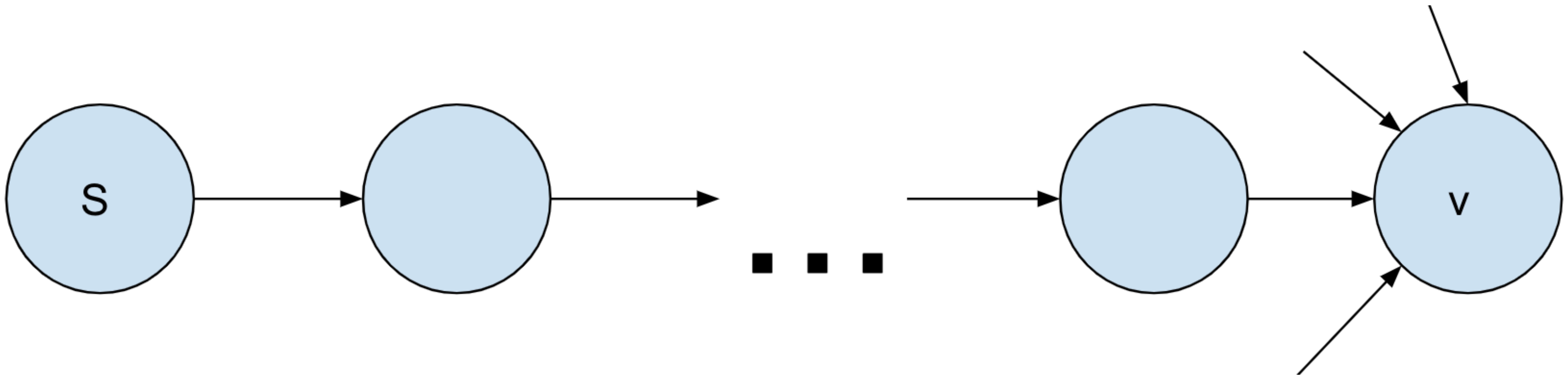
- Time = (# of subproblems) · (time per subproblem)

Shortest Paths

Recursive formulation:

$\delta(s, v) = \min\{ \delta(s, u) + w(u, v) \mid (u, v) \in E\}$

$\delta(s, s) = 0$



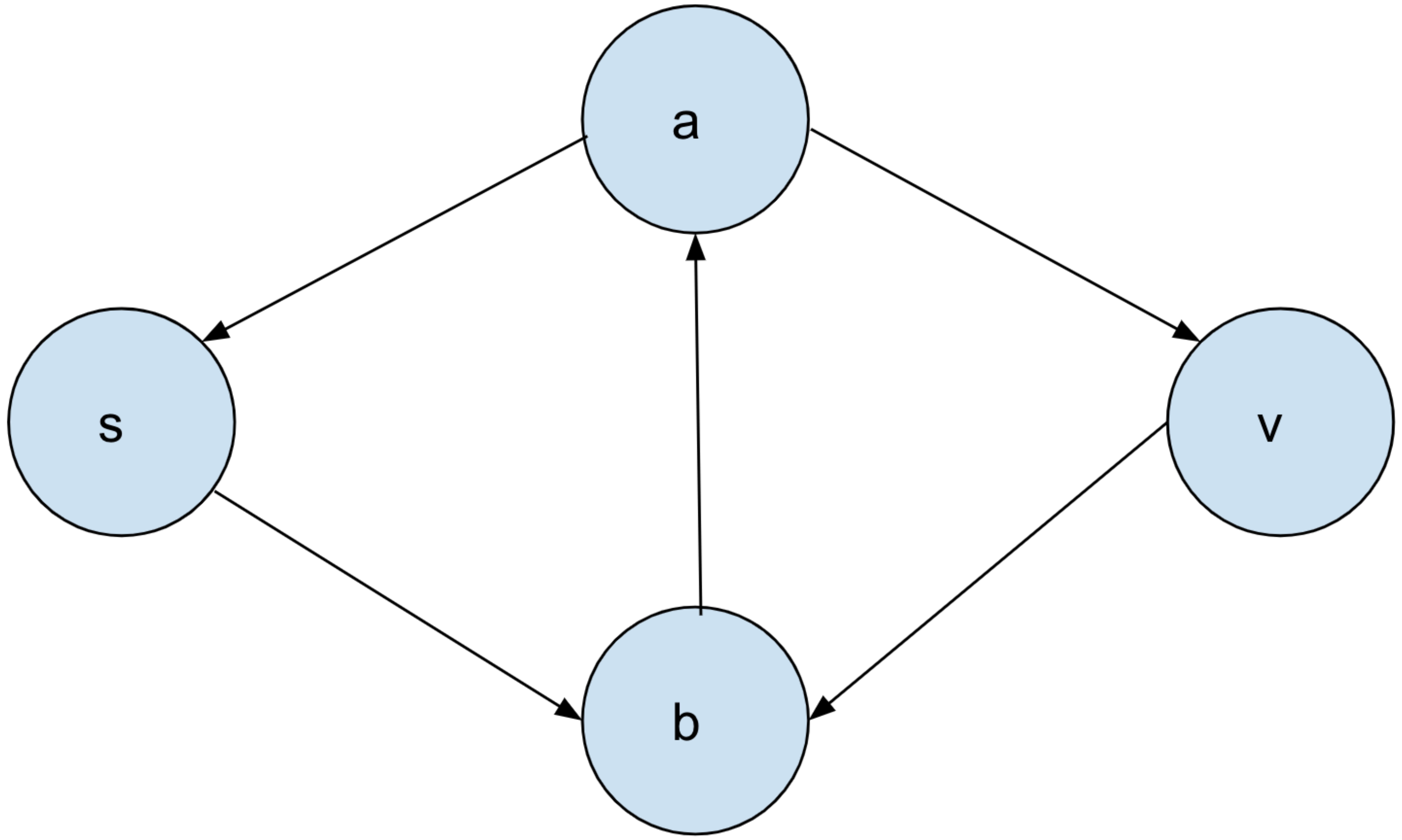
Let's use memoized approach to save time!

Cyclic example

Recursive formulation:

$\delta(s, v) = \min\{\ \delta(s, u) + w(u, v) \mid (u, v) \in E\}$

$\delta(s, s) = 0$



$$
\begin{tikzcd}
 & \delta(s, v) \arrow[d] & \\
 & \delta(s, a) \arrow[d] & \\
 & \delta(s, b) \arrow[dl] \arrow[dr] & \\
\delta(s, s) & & \delta(s, v)
\end{tikzcd}
$$

Time analysis

Recurrence:

$\delta(s, v) = \min\{ \delta(s, u) + w(u, v) \mid (u, v) \in E\}$

time = (# of subproblems) · (time/subproblem)

- # subproblems = |V|
- time per subproblem = indeg(v)

time = $\sum_{(v \text{ in } V)}(\text{indeg}(v) + 1) = O(E + V)$

Works only for acyclic graphs in O(V + E), else goes into infinite loop

We need subproblems to be acyclic

More subproblems remove cyclic dependence!

- $\delta_k(s, v)$ = shortest s → v path using ≤ k edges

Recurrence:

- $\delta_k(s, v) = \min\{ \delta_{k-1}(s, u) + w(u, v) \mid (u, v) \in E \}$
- $\delta_0(s, v) = \infty$ | for s != v (base case)
- $\delta_k(s, s) = 0$ | for any k (base case, if no negative cycles)

Goal:

$\delta(s, v) = \delta_{|V|-1}(s, v)$ (if no negative cycles)

Subproblem memoization table

i =	S	A	B	C	D	E
0	0	∞	∞	∞	∞	∞
1	0	6	∞	∞	∞	∞
2	0	6	9	∞	∞	∞
3	0	6	9	13	∞	14
4	0	6	9	13	16	14
5	0	6	9	13	16	14
6	0	6	9	13	16	14

- $\delta_k(s, v)$ = shortest $s \to v$ path using $\leq k$ edges

Time analysis

Recurrence:

$\delta_k(s, v) = \min\{ \delta_{k-1}(s, u) + w(u, v) \mid (u, v) \in E \}$

time = (# of subproblems) · (time/subproblem)

- # subproblems = |V| * |V|
- time per subproblem = indeg(v)

time = V * ($\Sigma_{(v \text{ in } V)}$(indeg(v))) = O(V * E)

Dynamic Programming Principles Summary

DP ~ "smart brute force"

DP ~ recursion + memoization + <u>guessing</u>

DP ~ shortest path in some DAG

- Time = (# of subproblems) · (time per subproblem)

5 Easy Steps to Dynamic Programming:

(1) **define subproblems**

(2) **guess (part of solution)**

(3) **relate subproblems solutions**

(4) **recurse or memoize or build DP table bottom up**

(5) **solve original problem**

5 Easy Steps to Dynamic Programming:

(1) **define subproblems** | *count # subproblems*

(2) **guess (part of solution)** | *count # choices*

(3) **relate subproblems solutions** | *compute (time/subproblem)*

(4) **recurse or memoize or build DP table bottom up** | *time = (time/subproblem) * #subproblems*

(5) **solve original problem** | a subproblem or a combination of subproblems

Examples:

	Fibonacci	Shortest Paths
1) subproblems	**F_k for $1 \le k \le n$**	**$\delta_k(s, v)$ for $v \in V$, $0 \le k < \|V\|$**
# subproblems	n	V^2
2) guess	**nothing to guess**	**edge into V (if any)**
# choices	1	indegree(v) + 1
3) recurrence	**$F_k = F_{k-1} + F_{k-2}$**	**$\delta_k(s, v)$=** $\min\{\delta_{k-1}(s, u)+w(u, v)\}$
time / subproblem	O(1) to compute	O(1 + indegree(v))
4) topological order	**for k = 1, …, n**	**for k=0, 1, …, |V| - 1**
total time	**O(n)**	**O(VE)**
5) original subproblem	**F_n**	**$\delta_{\|V\|-1}(s, v)$ for $v \in V$**
extra time	O(1)	O(V)

Questions