Announcements

Announcements

- Exam grades done (finally)
- Mean ~87%
- Results
  - No significant differences between examiners
  - Some significant differences across problems: #3- sorting array using Reverse- was much higher scoring than #1 and #2
  - We are adjusting scores for people who solved #1 and #2 so that the means for the first three problems were the same

Announcements

- Study groups?

e-mail me by Friday

Dynamic Programming

Introduction to Dynamic Programming

- Dynamic programming is a method of solving a problem in which the solution to a large problem is based on the solutions to smaller subproblems

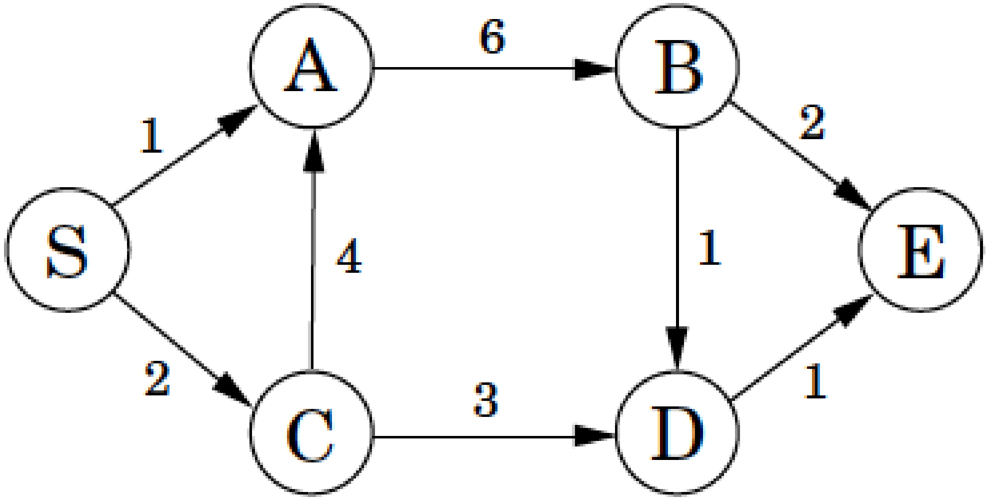
Introduction to Dynamic Programming

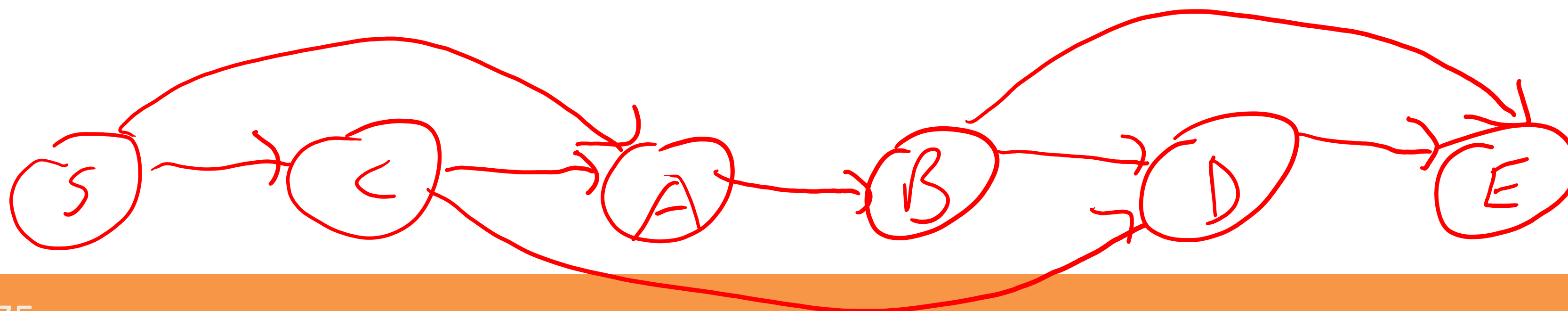
- Dynamic programming is a method of solving a problem in which the solution to a large problem is based on the solutions to smaller subproblems
- So far, sounds like divide-and-conquer!

Example: Shortest Paths in a DAG

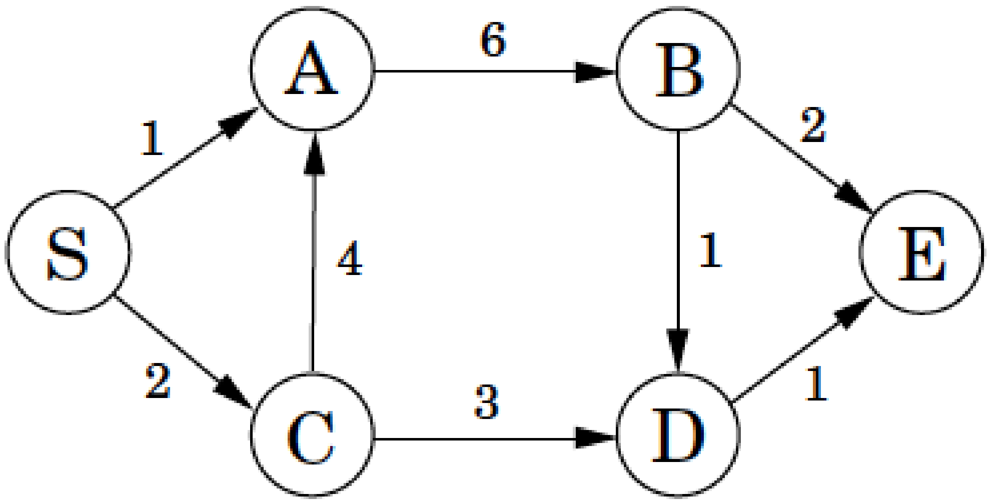
- Linearize this DAG

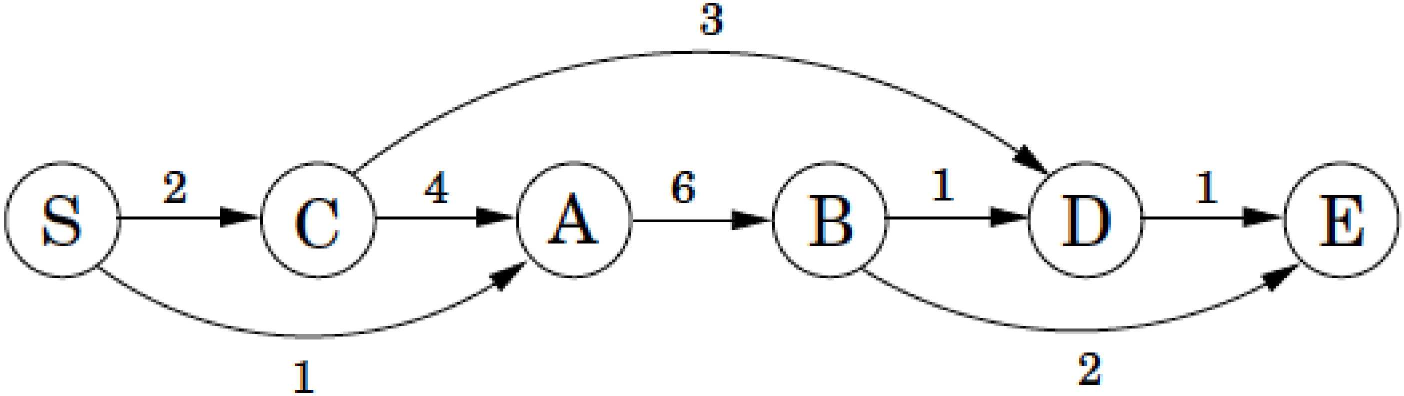
directed acyclic graph





Example: Shortest Paths in a DAG





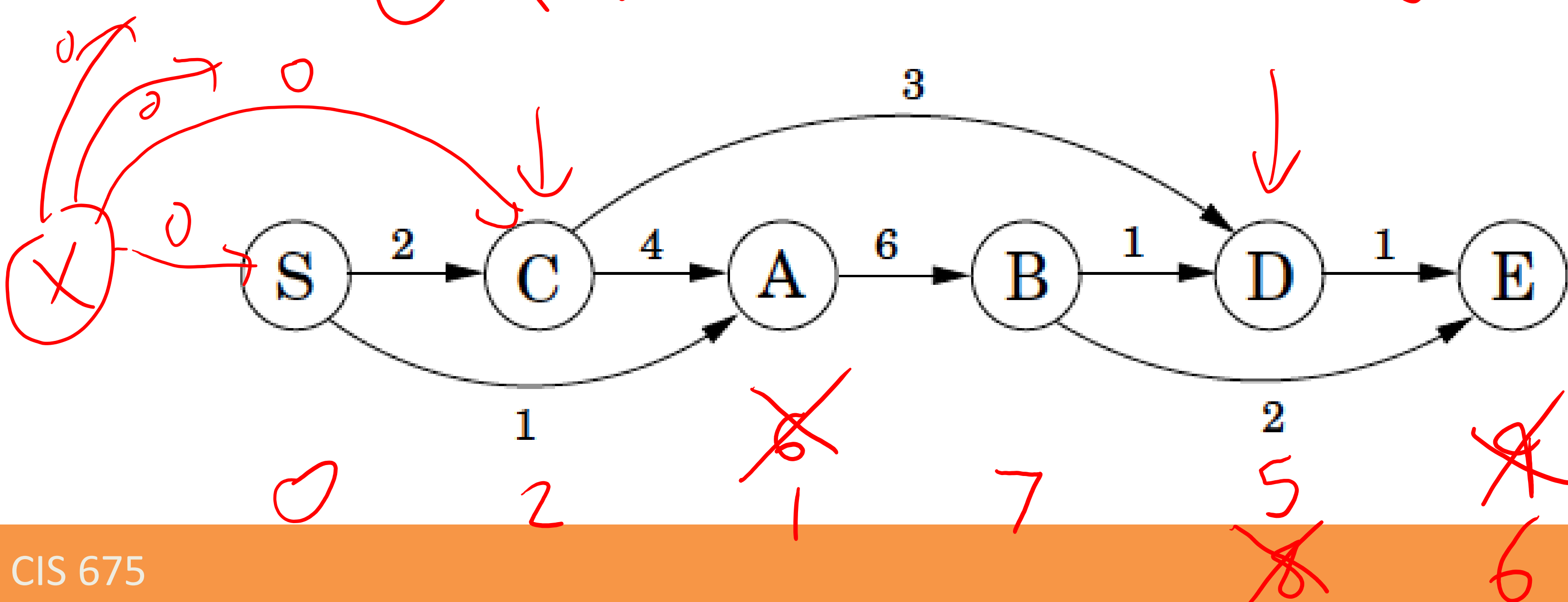
Example: Shortest Paths in a DAG

- Suppose we want to find the shortest path from S to ~~D~~ ~~any~~ every node. How can we do this in one scan of the linearized DAG?

$O(V+E)$

$V+E$

usually $\gg V$



Example: Shortest Paths in a DAG

- Suppose we want to find the shortest path from S to D. How can we do this in one scan of the linearized DAG?

initialize all $\texttt{dist}(\cdot)$ values to ∞

$\texttt{dist}(s) = 0$ after s

for each $v \in \cancel{V \setminus \{s\}}$, in linearized order:

$\quad \texttt{dist}(v) = \min_{(u,v) \in E} \{\texttt{dist}(u) + l(u,v)\}$

Example: Shortest Paths in a DAG

initialize all dist$(\cdot)$ values to ∞
dist$(s) = 0$
for each $v \in V \backslash \{s\}$, in linearized order:
    dist$(v) = \min_{(u,v) \in E}\{$dist$(u) + l(u, v)\}$

find s.p. from specific start node to every other node in a graph with edge weights

same, but for longest paths (replace min → max)

Example: Shortest Paths in a DAG

→ length of longest path to each node

initialize all dist$(\cdot)$ values to ∞

dist$(s) = 0$

for each $v \in V \setminus \{s\}$, in linearized order:

$$\text{dist}(v) = \max\{0, \cancel{\min}^{\max}_{(u,v)\in E}\{\text{dist}(u) + l(u,v)\}\}$$

↑ ~~To~~ node v is the optimal start node

Find l.p. in graph overall (graph still has edge weights)

Example: Shortest Paths in a DAG

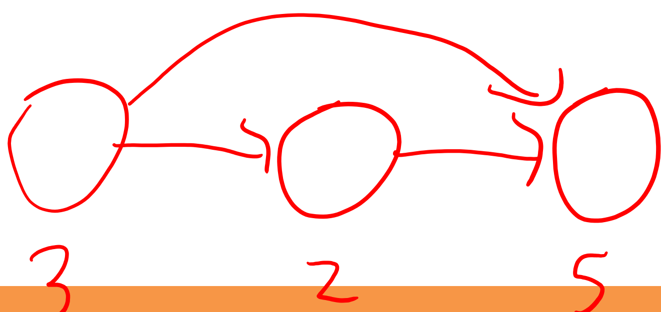
initialize all $\texttt{dist}(\cdot)$ values to ∞

$\texttt{dist}(s) = \cancel{0}$ $w(s)$

for each $v \in V \setminus \{s\}$, in linearized order:

$\qquad \texttt{dist}(v) = \min_{(u,v)\in E}\{\texttt{dist}(u) + \cancel{l(u,v)}\}$ $w(v)$

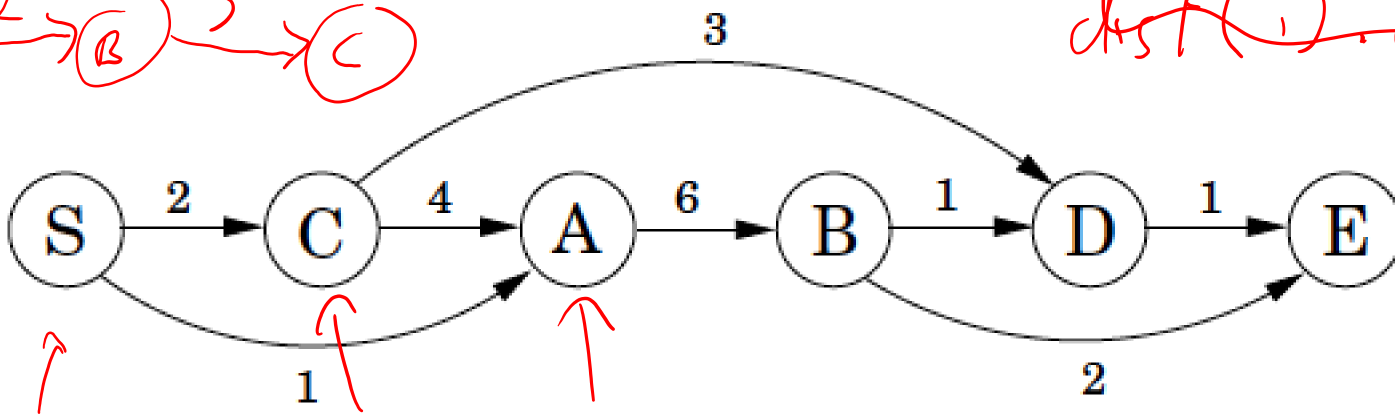
node weights instead of edge weights?



Example: Shortest Paths in a DAG

$A \xrightarrow{-2} B \xrightarrow{5} C$

dist(i): the




```
initialize all dist(·) values to ~~∞~~ −∞
dist(s) = 0
for each v ∈ V\{s}, in linearized order:
    dist(v) = max{0, {max ~~min~~_(u,v)∈E {dist(u) + l(u,v)}}}
```

In-Class Exercise

[1, 2, 1, 2]

- Suppose we want to find the longest path from S to D. How would we modify the pseudocode?

S - E

S - B　　S - D

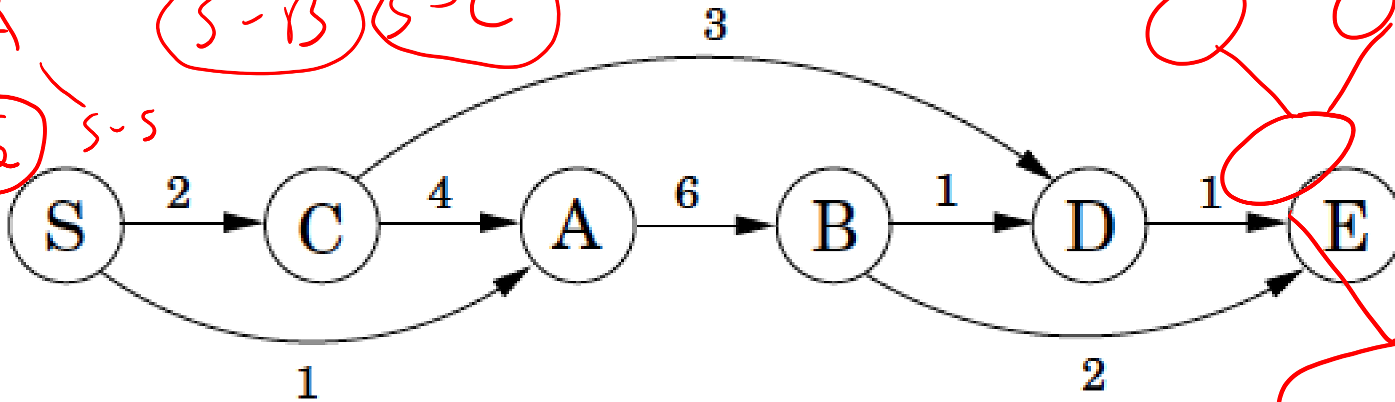
S - A　　S - B　　S - C

S - C　　S - S

S - S

$F(n): F(n-1) + F(n-2)$

$F(n-2) + F(n-3)$



Recursion?

- Would it make sense to implement this using recursion/D&C? What is the difference between this and recursion/D&C?

D-C: should not have repeated subproblems

top-down

initialize all dist$(\cdot)$ values to ∞
dist$(s) = 0$
for each $v \in V \backslash \{s\}$, in linearized order:
  dist$(v) = \min_{(u,v)\in E}\{$dist$(u) + l(u,v)\}$

DP: redundancy is ok (store values)

bottom-up approach

Shortest Paths in a DAG

- The problem of finding shortest paths in a DAG is a perfect analogy for dynamic programming!
  - Find solutions to subproblems (s.p.s to early nodes)
  - Use subproblems to find solutions to larger problems

Finding the Longest Increasing Subsequence

- Given a sequence of numbers, we want to find the longest increasing subsequence

inc. order of value

inc. order of index

5 2 8 6 3 6 9 7 5

Finding the Longest Increasing Subsequence

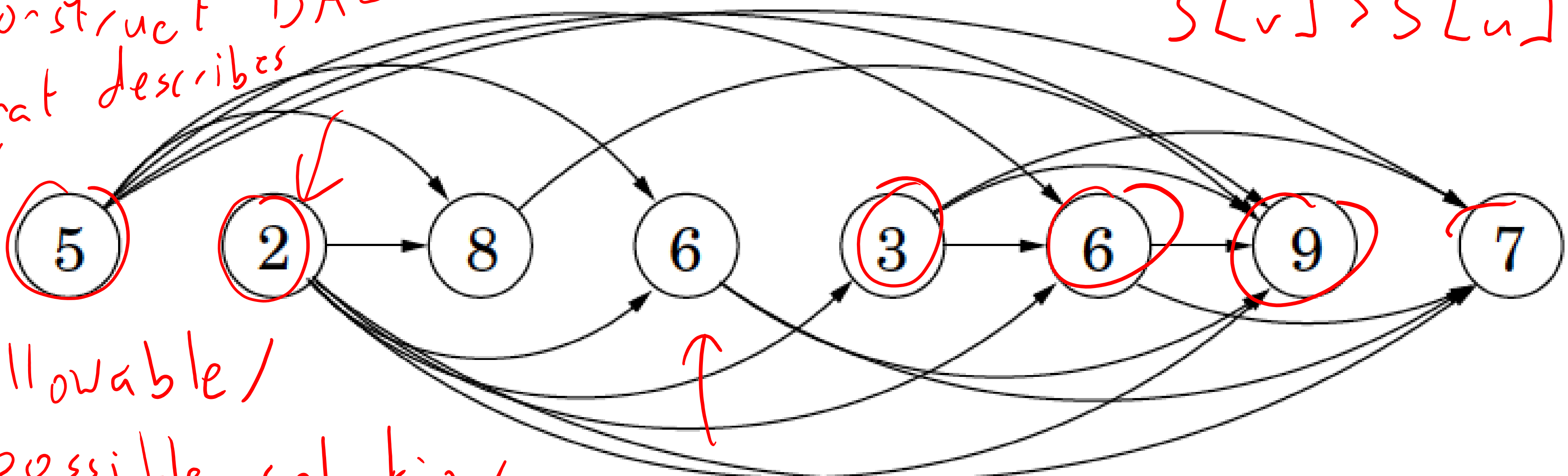
draw a node for every index in S

Step 1: Create a DAG

objective: construct DAG that describes allowable/possible solutions

$u \rightarrow v$: if v is after u in the sequence and $S[v] > S[u]$

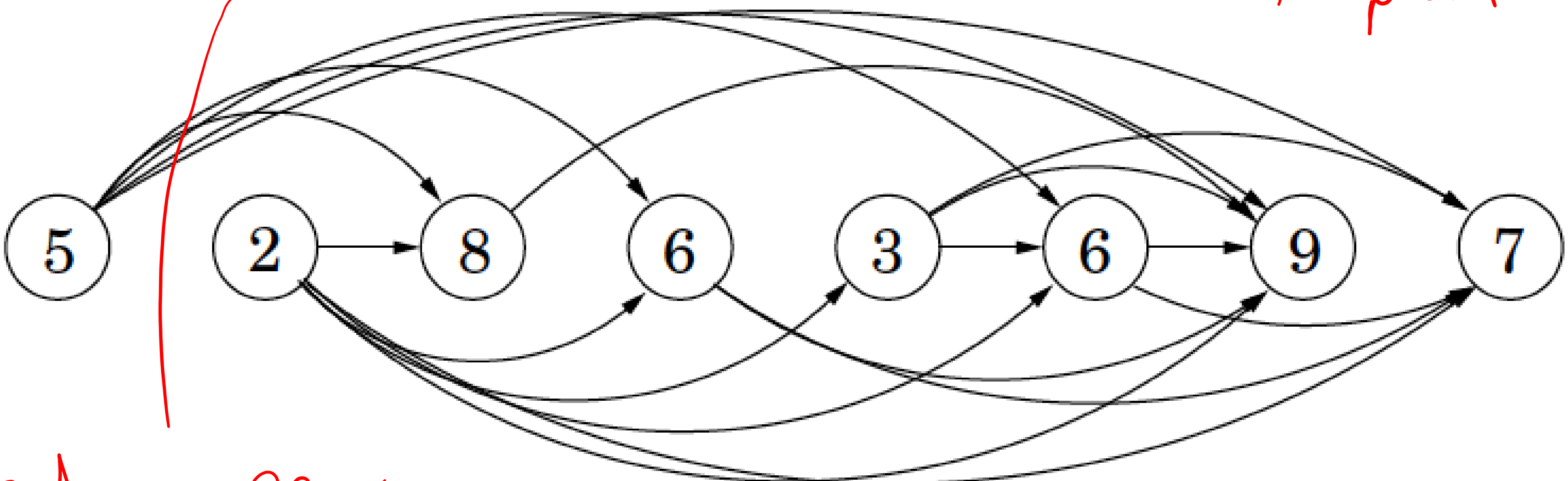
$v > u$



Finding the Longest Increasing Subsequence → find the longest path

Step 2: describe problem as a DAG problem

find longest path, return the nodes in path



edge no spec.
start point

Finding the Longest Increasing Subsequence

update equation

($L(j)$: length of the longest ISS ending at index i

$L(0) = 1$

$\textbf{for } j = 1, 2, \ldots, n\textbf{:}$

$\quad L(j) = 1 + \max\{L(i) : \cancel{(i,j) \in E}\}$

$\textbf{return } \max_j L(j)$

$i < j$ and $S[i] < S[j]$

Solving Problems with Dynamic Programming

- Key property:

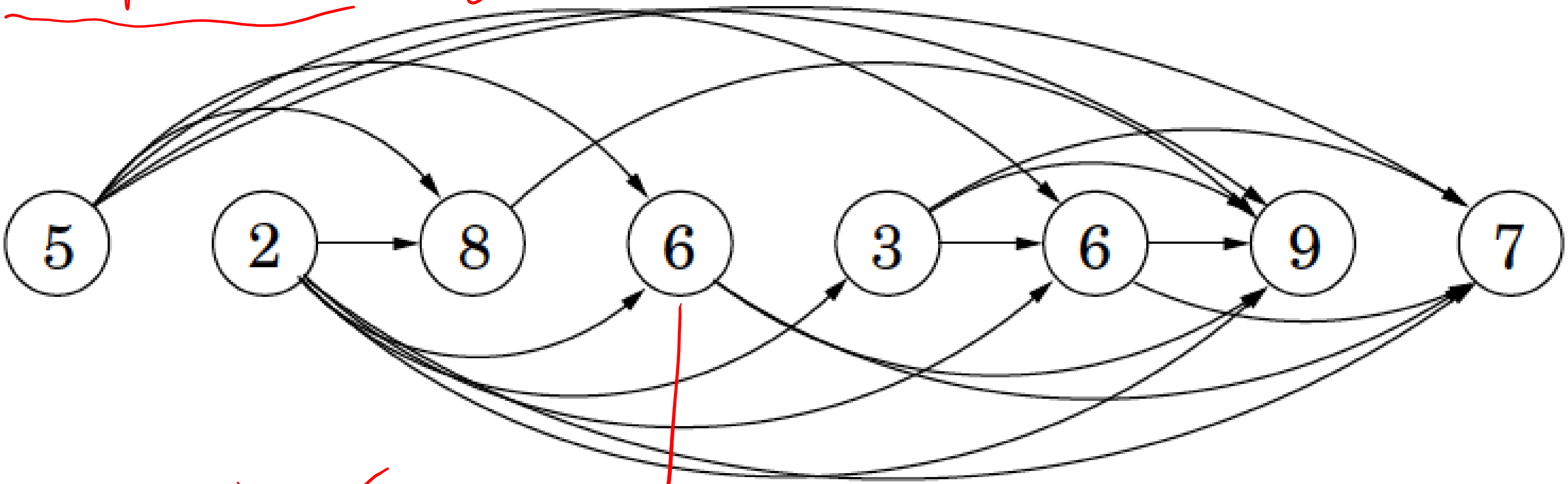
linearized

  - There is an ordering on the subproblems (DAG!)
  - There is a relation between the subproblems that allow you to solve later subproblems using earlier subproblems

Finding the Longest Increasing Subsequence

in a DAG, nodes = subproblems
edges = relationship b/w sps.

subproblems = questions



sub-problem (what is the length of the LISS ending at this node?)