Dynamic Programming

Shortest path cost from A to F?
A to G?
Dynamic Programming

Shortest Path Cost from A to F? A to G?
Dynamic Programming

\[
\text{Cost}[F] = \min \{ \min \{ \text{Cost}[C] + 2, \text{Cost}[D] + 3 \} \},
\]

\[
\text{Cost}[C] = \min \{ \min \{ \text{Cost}[A] + 4, \text{Cost}[B] + 2 \} \},
\]

\[
\text{Cost}[D] = \min \{ \min \{ \text{Cost}[B] + 3 \} \},
\]

\[
\text{Cost}[B] = \min \{ \min \{ \text{Cost}[A] + 1 \} \},
\]

\[
\text{Cost}[A] = 0
\]
def shortest_path(G, s):
    for u in V:
        dist[u] = oo
    dist[s] = 0

    for v in V - {s}:
        # linearized order.
        dist[v] = min((u,v) in E) ? dist[u] + l(u,v)

    return dist
Dynamic Programming

Longest Increasing Subsequence

5 2 8 6 3 6 9 7
Dynamic Programming

Longest Increasing Subsequence

- Can you see "nodes" of a "DAG"?
- Nodes are subproblems that look like smaller versions of the original.
- What are the edges?
- How can we process the nodes in linearized order?
- What is the initialization?
def longest_increasing_subsequence(nums):
    L[0] = 0
    for j in range(1, len(nums)):
        L[j] = 1 + max([L[i] for i, j in E | L[i]]
    return max([L[j]]