

Program Structure and Algorithms

Sid Nath

Lecture 11

Agenda

- Administrative
 - HW5 and PA2 will be announced soon..
- Lecture
- Quiz

Algorithm Design Key Ideas

- For SOME problems, the following property enables an efficient solution:
 - **“Principle of Optimality”**: Any subsolution of an optimal solution is itself an optimal solution
- For such problems, solution strategies can include:
 - **“Tabulate (cache) Subproblem Solutions”**:
 - Avoid recomputation by creating a table of subproblem solutions
 - **“Relaxation / Successive Approximation”**:
 - Make multiple passes, each time solving a less restricted version of the original problem
 - Eventually, solve completely unrestricted = original problem instance

DP is a Promising Approach if...

- **Optimal substructure:** A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to subproblems
 - True for DP and greedy
- **Overlapping subproblems:** A recursive algorithm for the problem would end up solving the same subproblems over and over, rather than always generating new subproblems
- So,
 - Bring in induction mindset. See where ordering is important
 - Ordering from the first index or an interval?

DQ vs. DP

- In both cases, we start by formulating the problem recursively, in terms of subproblems

DQ

- Problem of size n can be decomposed into a few subproblems that are significantly smaller (e.g., $n/2$, $3n/4$, etc.)
- Size of subproblems decreases geometrically
- Use a recursive algorithm

DP

- Problem of size n can be expressed in terms of subproblems that are not much smaller (e.g., $n-1$, $n-2$, etc.)
 - Recursive algorithm takes exponential time
 - But only polynomially many subproblems in total
- Avoid recursion, and solve subproblems one by one, saving answers in a table

DP Problem Examples

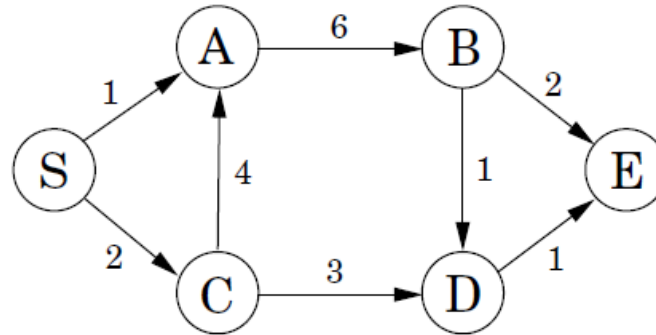
- Rod cutting to maximize revenue
- Longest increasing subsequence
- Minimum edit distance
- Knapsack
- Matrix chain product
- All pairs shortest path (Floyd-Warshall)
- Independent sets in trees
- Vertex cover in trees
- Longest common subsequence
- Transitive closure
- Change making
-

DP: Common Subproblems

Finding the right subproblem takes creativity and experimentation. Some standard choices below

- Template #1: Input is x_1, x_2, \dots, x_n , (or, $x[1:n]$) and a subproblem is x_1, x_2, \dots, x_i (or, $x[1:i]$)
 - Number of subproblems is linear
- Template #2: Input is x_1, x_2, \dots, x_n , and a subproblem is x_i, x_{i+1}, \dots, x_j (or, $x[i:j]$)
 - #subproblems is quadratic $O(n^2)$
- Template #3: Inputs are x_1, x_2, \dots, x_n , and y_1, y_2, \dots, y_m ; a subproblem is x_1, x_2, \dots, x_i and y_1, y_2, \dots, y_j
 - #subproblems is quadratic $O(mn)$

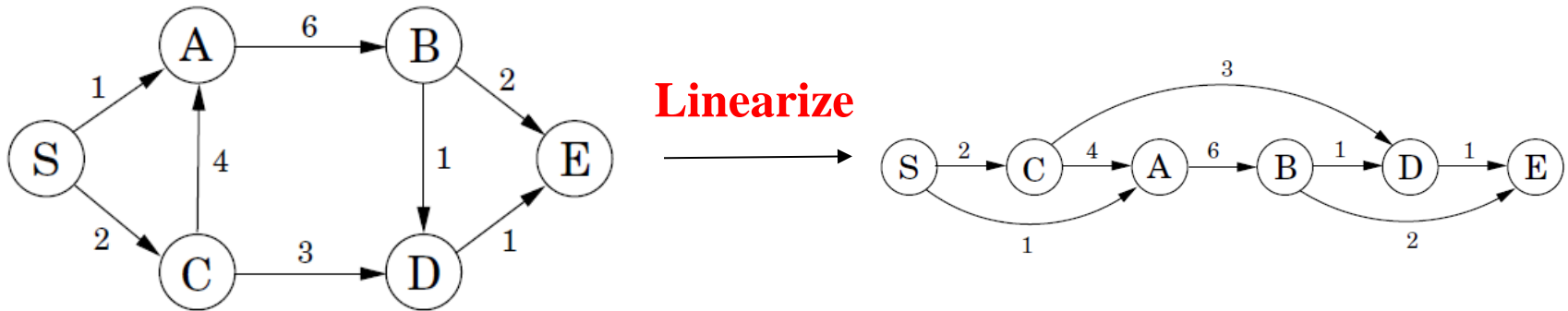
P1: Shortest Paths in DAGs



Given DAG $G = (V; E)$, what is the shortest path from S to any node in G ?

E.g., D

P1: Shortest Paths in DAGs



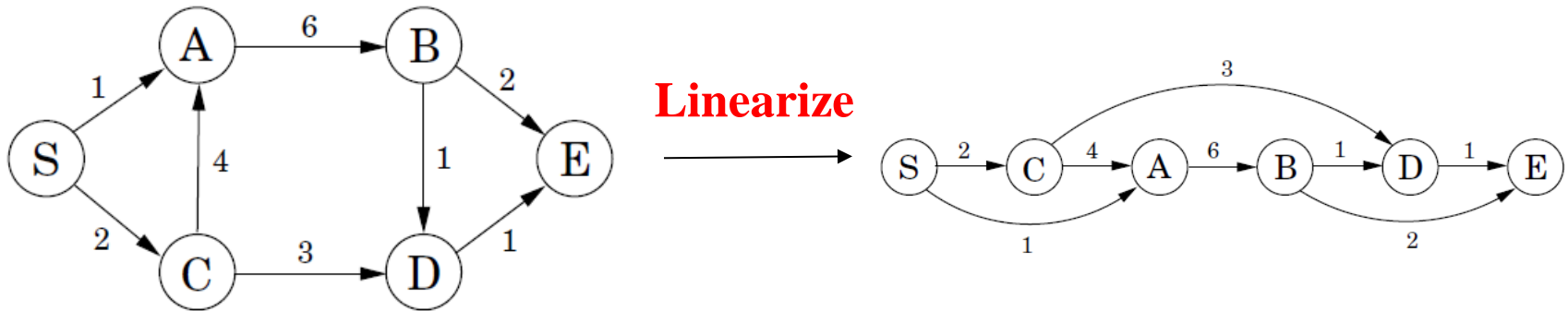
Given DAG $G = (V; E)$, what is the shortest path from S to any node?

E.g., D

Intuition: What if we knew the shortest paths from S to the parents of D?

$$\text{dist}(D) = \min\{\text{dist}(B) + 1, \text{dist}(C) + 3\}$$

P1: Shortest Paths in DAGs



Intuition: What if we knew the shortest paths from S to the parents of D? $dist(D) = \min\{dist(B) + 1, dist(C) + 3\}$

Does it work for all nodes?

$$dist(S) = 0$$

$$dist(C) = dist(S) + 2 = 2$$

$$dist(A) = \min\{dist(C) + 4, dist(S) + 1\} = \min(2 + 4, 0 + 1) = 1 \text{ // 2 decisions}$$

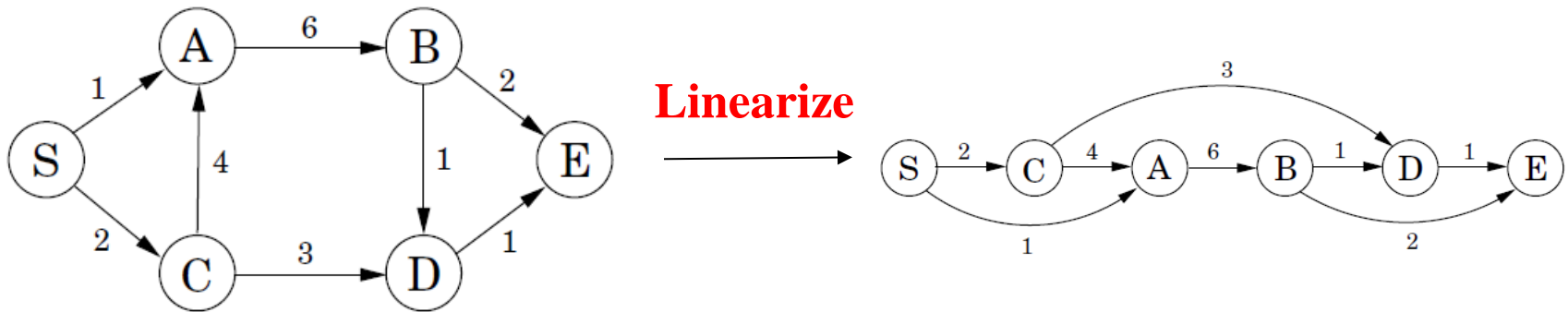
$$dist(B) = dist(A) + 6 = 1 + 6 = 7$$

$$dist(D) = \min\{dist(B) + 1, dist(C) + 3\} = \min(7 + 1, 2 + 3) = 5 \text{ // 2 decisions}$$

$$dist(E) = \min\{dist(B) + 2, dist(D) + 1\} = \min(7 + 2, 5 + 1) = 6 \text{ // 2 decisions}$$

Without linearization/topo sort, this idea will NOT work!!!

Shortest Paths in DAGs

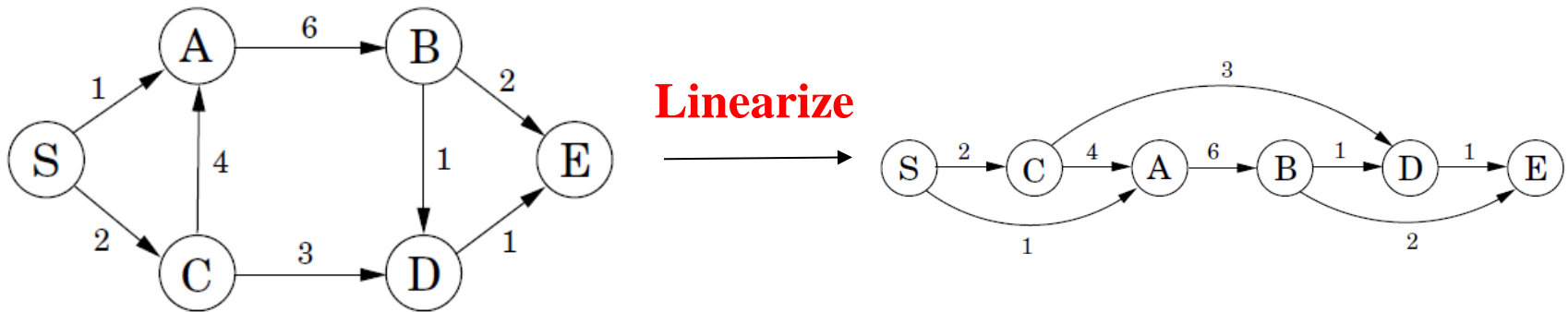


Given DAG $G = (V; E)$, what is the shortest path from S to any node?

For any DP, we need to

- Identify and articulate subproblems
- Articulate the decisions
- Describe the recursion
- Analyze running time: #subproblems x time to solve each subproblem

P1: Shortest Paths in DAGs



Given DAG $G = (V; E)$, what is the shortest path from S to any node?

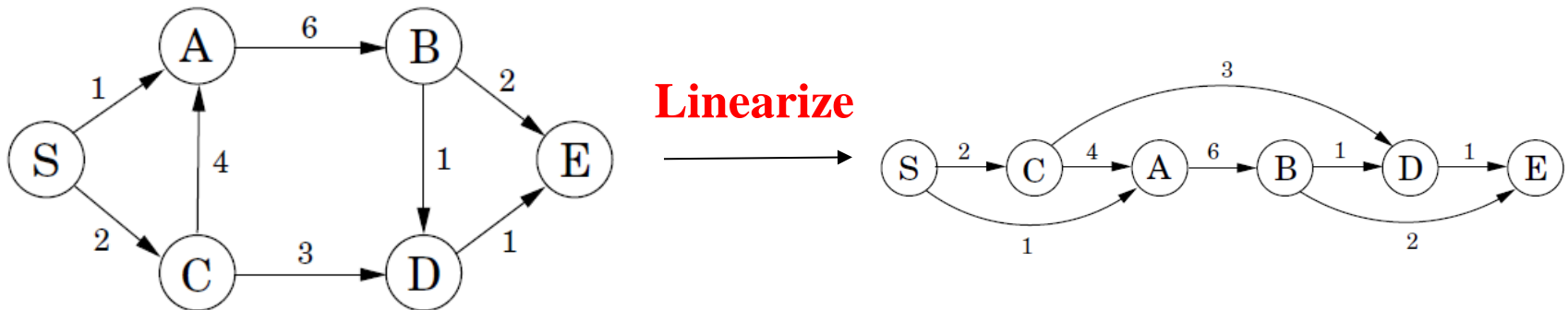
Subproblem: Let $\text{dist}(v)$ denote the **shortest path** to any node v in G **from** **S**

- When we linearize, we get an order of vertices x , if $\text{src} = 1$, $v = i$ are indices of the source and v
- We can say subproblems are to look at the min distance $x[1:i]$, i.e., starting from the source and ending at vertex index i

Decisions: Consider $\text{dist}(u)$ to all parents of v and pick the min of $\text{dist}(u) + l(u,v)$

Recursion: $\text{dist}(v) = \min\{\text{dist}(u) + l(u,v) \mid \text{for all } (u,v) \text{ in linearized } G\}$

P1: Shortest Paths in DAGs



Given DAG $G = (V; E)$, what is the shortest path from S to any node?

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

How many subproblems? $O(|V| + |E|)$

Running time per subproblem? $O(1)$ (one min operation)

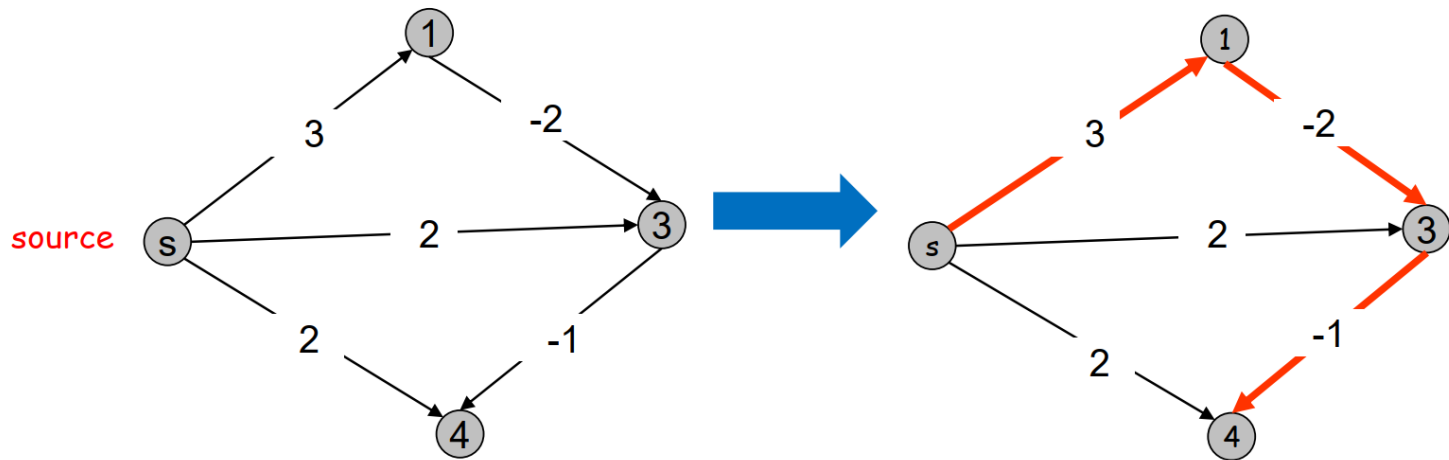
Total Running time: $O(|V| + |E|)$

P2: Shortest Paths w/ Negative Edge Weights

(aka, Bellman-Ford)

Given $G = (V, E)$, a source vertex $s \in V$ and real-valued edge weights for each edge in E as $c_{u,v}$, find the shortest path from s to all other vertices in G

Why does Dijkstra fail?



Shortest Paths w/ Negative Edge Weights

We cannot have negative cycles in G

Let $d(v)$ be the length of the shortest $s - v$ path

$$d(v) = \begin{cases} 0 & \text{if } v = s \\ \min_{u:(u,v) \in E} d(u) + c_{u,v} & \end{cases}$$

How do we compute?

Shortest Paths w/ Negative Edge Weights

Subproblem: Let $d(v, i)$ be the length of the shortest $s - v$ path with at most i edges

Intuition: Solve all intermediate routes from s to v (obey triangle inequality)

Decisions:

Case 1: i^{th} edge is not used, then $d(v, i) = d(v, i - 1)$

Case 2: i^{th} edge is used

Let $s, v_1, v_2, \dots, v_{i-1}$ be the $d(v, i)$ path with i edges

Then, $s, v_1, v_2, \dots, v_{i-1}$ MUST be the shortest $s - v_{i-1}$ path with at most $(i - 1)$ edges

That is, $d(v, i) = d(v_{i-1}, i - 1) + c_{v_{i-1}, v}$

Shortest Paths w/ Negative Edge Weights

Let $d(v, i)$ be the length of the shortest $s - v$ path with at most i edges

Recursion:

$$d(v, i) = \begin{cases} 0 & \text{if } v = s \\ \infty & \text{if } v \neq s, i = 0 \\ \min(d(v, i - 1), \min_{u:(u,v) \in E} d(u, i - 1) + c_{u,v}) & \text{otherwise} \end{cases}$$

This is Bellman-Ford algorithm

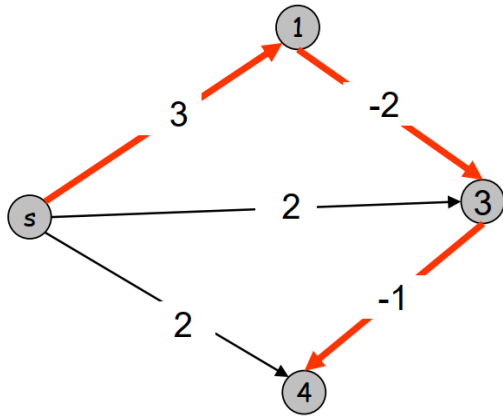
Bellman Ford

```
for v=1 to n
    if v ≠ s then
        M[v,0]=∞
    M[s,0]=0

for i=1 to n-1
    for v=1 to n
        M[v,i]= M[v,i-1]
        for every edge (u,v)
            M[v,i]=min(M[v,i] , M[u,i-1]+c[u,v])
```

Running time: $O(|V||E|)$

Working of Bellman Ford



P3: All Pairs Shortest Path

Given directed graph $G = (V, E)$ and real-valued edge weights, create $n \times n$ matrix of shortest distances $\delta(u, v)$ between all pairs of vertices; $|V| = n$

Adjacency matrix representation of G

- $n \times n$ matrix $W = (w_{ij})$ of edge weights (can be -ve)
- $w_{ii} = 0 \ \forall i$, i.e., SP to self has no edges, as long as there are no negative cycles

Brute-Force Approach

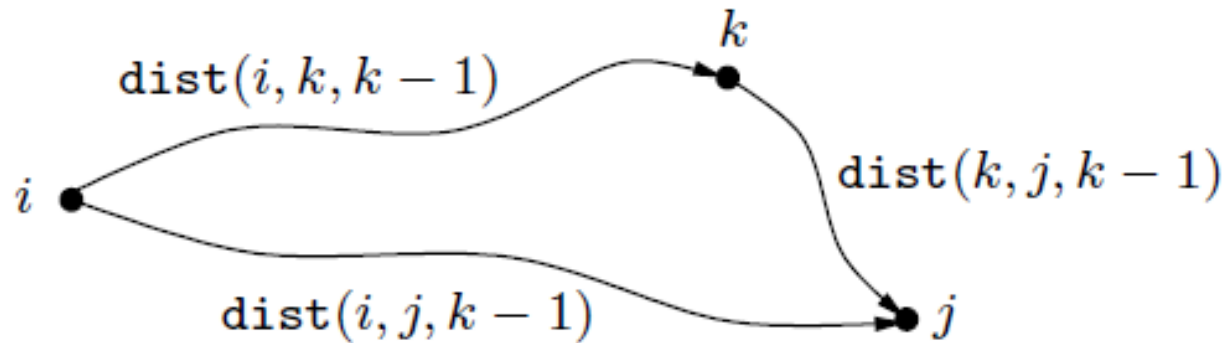
- Run Bellman-Ford once from each vertex (since edge weights may be negative)
- $O(|V||E|) \times O(|V|) = O(|V|^4)$ for dense graphs

DP: Structure of an optimal solution

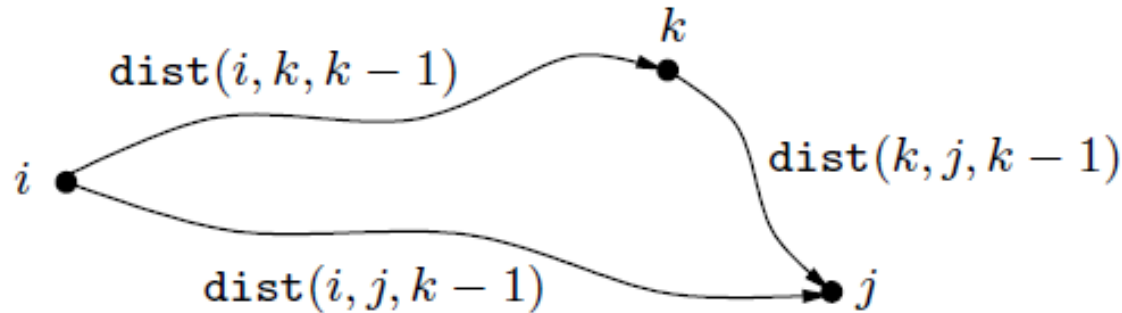
“Relaxation” idea: Similar to Bellman Ford

Shortest dist between vertices i and j , $\delta(i, j)$ with $\leq m$ edges (relax $0 \leq m \leq n - 1$)

- Equivalent to saying shortest path between i and j (or, SP $\delta(i, j)$) uses some of $\{1, 2, \dots, k\}$ intermediate nodes
- That is, $i \rightarrow w_1 \rightarrow w_2 \rightarrow \dots \rightarrow w_k \rightarrow j$



DP: Structure of an optimal solution



Subproblems: Let $\text{dist}(i, j, k)$ be the cost of the shortest path $i \rightarrow j$ with intermediate vertices in the set $\{1, 2, \dots, k\}$

Intuition: In BF, s was given. Here, the source can be any vertex, so need a variable for it.

Decisions

- Either w_k is an intermediate vertex in SP $\delta(i, j) \rightarrow \delta(i, j) = \delta(i, w_k) + \delta(w_k, v)$
- Or, w_k is not an intermediate vertex, so consider $\delta(i, j)$ only involving $\{w_1 \rightarrow w_2 \rightarrow \dots \rightarrow w_{k-1}\}$

DP: Structure of an optimal solution

Decisions

- Either w_k is an intermediate vertex in SP $\delta(i, j) \Rightarrow \delta(i, j) = \delta(i, w_k) + \delta(w_k, v)$
- Or, w_k is not an intermediate vertex, so consider $\delta(i, j)$ only involving $\{w_1 \rightarrow w_2 \rightarrow \dots \rightarrow w_{k-1}\}$

Case 1: If $i = j$, $d_{ij}^0 = 0$ (base case #1)

Case 2: If $i \neq j$, $dist(i, j, k)$ is the $\delta(i, j)$ in which only nodes $\{1, 2, \dots, k\}$ can be used as intermediates.

Case 3: $dist(i, j, 0)$ is the length of direct edge between i and j , i.e., $dist(i, j, 0) = w_{ij}$ (base case #2)

DP: A Recursive Solution

Let $\text{dist}(i, j, k)$ be the cost of the shortest path $i \rightarrow j$ with intermediate vertices in the set $\{1, 2, \dots, k\}$

Try for all k and pick the min cost one.

Recursion:

$$\begin{aligned} & \text{dist}(i, j, k) \\ &= \min \begin{cases} w_{ij} & \text{if } k = 0 \\ \text{dist}(i, j, k-1), & \text{if } k \text{ is not an intermediate node} \\ \text{dist}(i, k, k-1) + \text{dist}(k, j, k-1), & \text{otherwise} \end{cases} \end{aligned}$$

This is **Floyd-Warshall** algorithm!

DP: Pseudocode (Bottom-up)

```
for  $i = 1$  to  $n$ :  
    for  $j = 1$  to  $n$ :  
         $\text{dist}(i, j, 0) = \infty$   
for all  $(i, j) \in E$ :  
     $\text{dist}(i, j, 0) = \ell(i, j)$   
for  $k = 1$  to  $n$ :  
    for  $i = 1$  to  $n$ :  
        for  $j = 1$  to  $n$ :  
             $\text{dist}(i, j, k) = \min\{\text{dist}(i, k, k - 1) + \text{dist}(k, j, k - 1), \text{dist}(i, j, k - 1)\}$ 
```

Runtime: $\Theta(n^3)$ due to
 n^2 subproblems, each takes $\Theta(n)$ to solve

DP: Pseudocode (Top-down)

FW-RECUR($V, i, j, k, w, memo$)

```
1  if  $memo[i, j, k]$ 
2      return  $memo[i, j, k]$ 
3  if  $k = 0$ 
4      return  $w(v_i, v_j)$ 
5   $memo[i, j, k] \leftarrow$ 
    $\min [FW-RECUR(V, i, j, k - 1, w), FW-RECUR(V, i, k, k - 1, w) + FW-RECUR(V, k, j, k - 1, w)]$ 
6  return  $memo[i, j, k]$ 
```

FLOYD-WARSHALL(V, w)

```
1   $memo \leftarrow$  Empty memo
2  for  $v_i \in V$ 
3      for  $v_j \in V$ 
4           $\delta(v_i, v_j) \leftarrow FW-RECUR(V, i, j, |V| - 1, w, memo)$ 
5  return  $\delta$ 
```

Similar: Transitive Closure of Graph

Transitive closure of $G = (V, E)$ is a graph $G^* = (V, E^*)$

$(i, j) \in E^*$ iff $\exists \text{ path } i \rightarrow j \text{ in } G$

Input: Adjacency matrix elements with elements in $\{0, 1\}$

Run Floyd-Warshall with

- “min” \rightarrow “OR”
- “+” \rightarrow “AND”

Runtime: $\Theta(n^3)$

Applications

- Dependencies in SW modules
- Reachability in social / transportation networks, node importance
- Speedup database queries by clustering related data
- Feature learning in ML

Summary of DP Problems

Prob #	Definition	Opt Type	Template	Running Time (only DP part)
1	Shortest paths in DAGs	Min	#1	$O(V + E) \times O(1)$
2	Bellman Ford	Min	#1	$O(V) \times O(E)$
3	Floyd Warshall	Min	#2	$O(n^2) \times O(n)$
4	Transitive Closure	Min	#2	$O(n^2) \times O(n)$

Lecture 11 summary

- DP
- P1: Shortest path in a DAG
- P2: SSSP in a graph (Bellman Ford)
- P3: APSP
- P4: Transitive closure
- Please practice writing code given the pseudocodes