Bellman-Ford and All-Pairs Shortest Paths

Lecture 19

April 3, 2025

Part I

SSSP with Negative Weights

Last Time

Want to solve the single-source shortest-path problem: given a vertex s, what is $\text{dist}(s, t)$ for each vertex t?

- Weighted graphs: length is the sum of the weights of the edges.

Last Time

Want to solve the single-source shortest-path problem: given a vertex s, what is $\mathrm{dist}(s, t)$ for each vertex t?

- Weighted graphs: length is the sum of the weights of the edges.

Dijkstra's algorithm makes "guesses" based on shortest path seen so far, and at each step "confirms" the smallest guess.

Last Time

Want to solve the single-source shortest-path problem: given a vertex s, what is $\mathbf{dist}(s, t)$ for each vertex t?

- Weighted graphs: length is the sum of the weights of the edges.

Dijkstra's algorithm makes "guesses" based on shortest path seen so far, and at each step "confirms" the smallest guess.

- Correct *as long as the edge weights are all non-negative*.

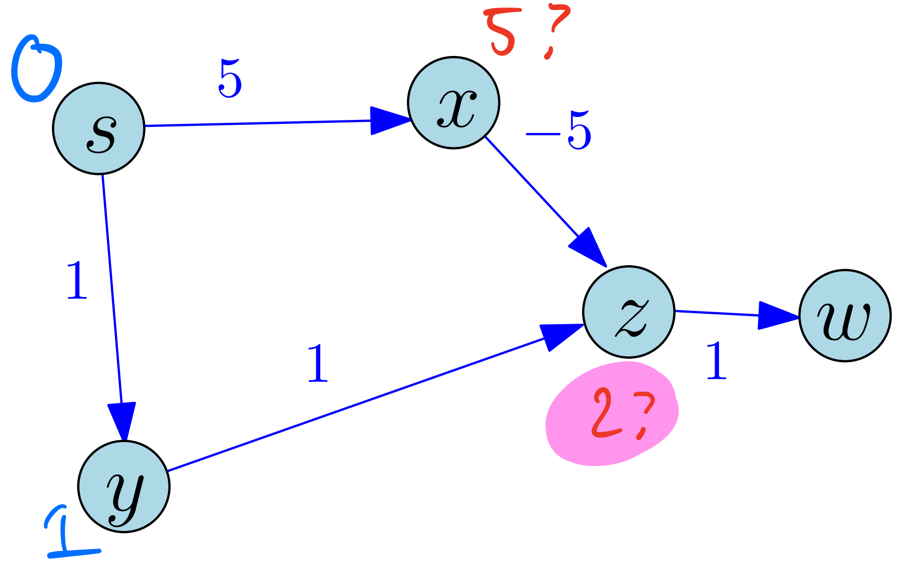
Last Time

Want to solve the single-source shortest-path problem: given a vertex s, what is $\mathbf{dist}(s, t)$ for each vertex t?

- Weighted graphs: length is the sum of the weights of the edges.

Dijkstra's algorithm makes "guesses" based on shortest path seen so far, and at each step "confirms" the smallest guess.

- Correct *as long as the edge weights are all non-negative*.
- May give the wrong answer if there are negative edge weights!



DP: Attempt 1

Why don't we use DP? (It worked for DAGs!)

DP: Attempt 1

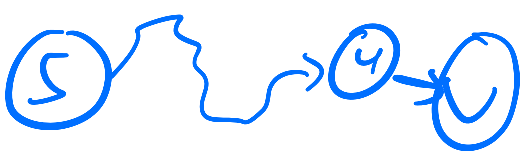
Why don't we use DP? (It worked for DAGs!)

Subproblem definition:

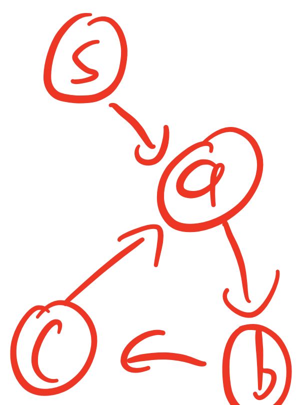
LSP(v) is the length of the shortest path from s to v.

Recurrence:

$$LSP(v) = \begin{cases} \min_{(u,v)\in E} LSP(u) + w(u,v) & \text{if } v \neq s \\ 0 & \text{if } v = s \end{cases}$$



Evaluation Order: 



DP: Attempt 2

How do we avoid circular dependencies? (Hint: add a subproblem parameter.)

Subproblem definition:

$LSP(v, \ell)$ is the length of the shortest path from s to v that uses at most ℓ edges

Recurrence:

$$LSP(v,\ell) = \begin{cases} \min\left(\min_{(u,v)\in E} LSP(u,\ell-1) + w(u,v),\ LSP(v,\ell-1) \right) & \text{if } \ell > 0 \\ 0 & \text{if } \ell = 0,\ v = s \\ \infty & \text{if } \ell = 0,\ v \neq s \end{cases}$$

Evaluation Order:

ℓ from 1 up to V - 1

v in arbitrary order

DP Pseudocode

Let $\mathrm{LSP}(v, \ell)$ be the length of the shortest path from s to v that uses at most ℓ edges.

SSSP-DP(G, s):
    Initialize LSP as a $V \times V$ matrix
    Set $\mathrm{LSP}[s, 0] = 0$ and $\mathrm{LSP}[v, 0] = \infty$ for all $v \neq s$
    for ℓ from 1 to $V - 1$:
        for all vertices v:
            $\mathrm{LSP}[v, \ell] = \mathrm{LSP}[v, \ell - 1]$
            for all edges (u, v):
                $\mathrm{LSP}[v, \ell] = \min(\mathrm{LSP}[v, \ell], \mathrm{LSP}[u, \ell - 1] + w(u, v))$
    return $\mathrm{LSP}[u, V - 1]$ for all $u \in V$

DP Pseudocode

Let LSP(v, ℓ) be the length of the shortest path from s to v that uses at most ℓ edges.

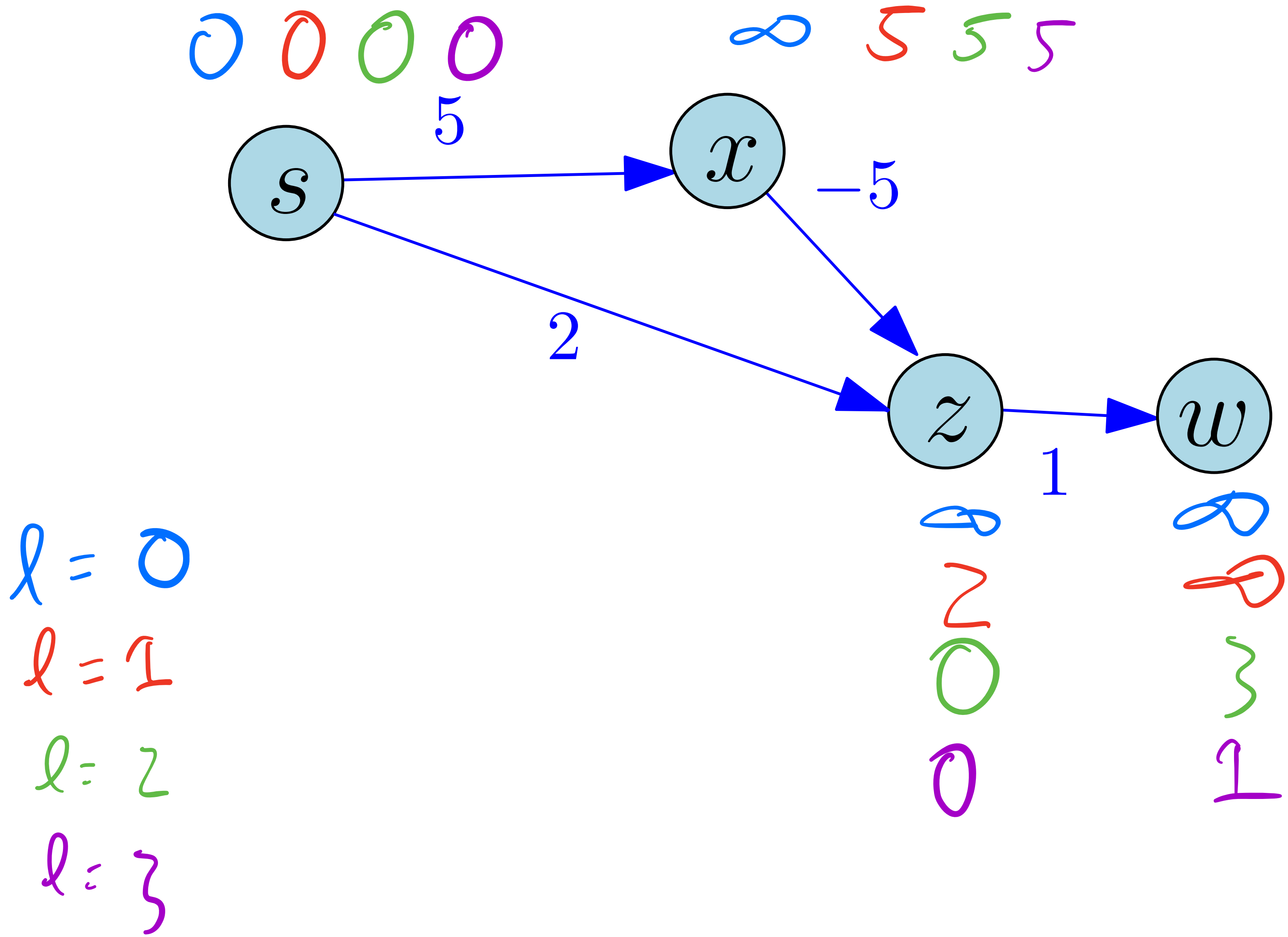
SSSP-DP(G, s):
    Initialize LSP as a $V \times V$ matrix
    Set LSP$[s, 0] = 0$ and LSP$[v, 0] = \infty$ for all $v \neq s$
    for ℓ from 1 to $V - 1$:
        for all vertices v:
            LSP$[v, \ell]$ = LSP$[v, \ell - 1]$
            for all edges (u, v):
                LSP$[v, \ell]$ = $\min($LSP$[v, \ell]$, LSP$[u, \ell - 1] + w(u, v))$
    return LSP$[u, V - 1]$ for all $u \in V$

Efficiency?

time to solve LSP(v, ℓ) = $O(\deg v)$

total: $\sum_{\ell} \sum_{v} O(\deg v) = \sum_{\ell} O(E) = O(VE)$

DP Example



Simplification

Observation: we don't *really* care that the path we find for LSP(u, ℓ) has at most ℓ edges...

Simplification

Observation: we don't *really* care that the path we find for LSP(u, ℓ) has at most ℓ edges...

May as well always use our current best guess at the distance!

Simplification

Observation: we don't *really* care that the path we find for LSP(u, ℓ) has at most ℓ edges...

May as well always use our current best guess at the distance!

```
BellmanFord(G, s):
    Set s.dist = 0 and v.dist = ∞ for all v ≠ s
    for ℓ from 1 to V − 1:
        for all edges (u, v):
            v.dist = min(v.dist, u.dist + w(u, v))
    return u.dist for all u ∈ V
```

Simplification

Observation: we don't *really* care that the path we find for LSP(u, ℓ) has at most ℓ edges...

May as well always use our current best guess at the distance!

```
BellmanFord(G, s):
    Set s.dist = 0 and v.dist = ∞ for all v ≠ s
    for ℓ from 1 to V − 1:
        for all edges (u, v):
            v.dist = min(v.dist, u.dist + w(u, v))
    return u.dist for all u ∈ V
```

Correctness?

Simplification

Observation: we don't *really* care that the path we find for LSP(u, ℓ) has at most ℓ edges...

May as well always use our current best guess at the distance!

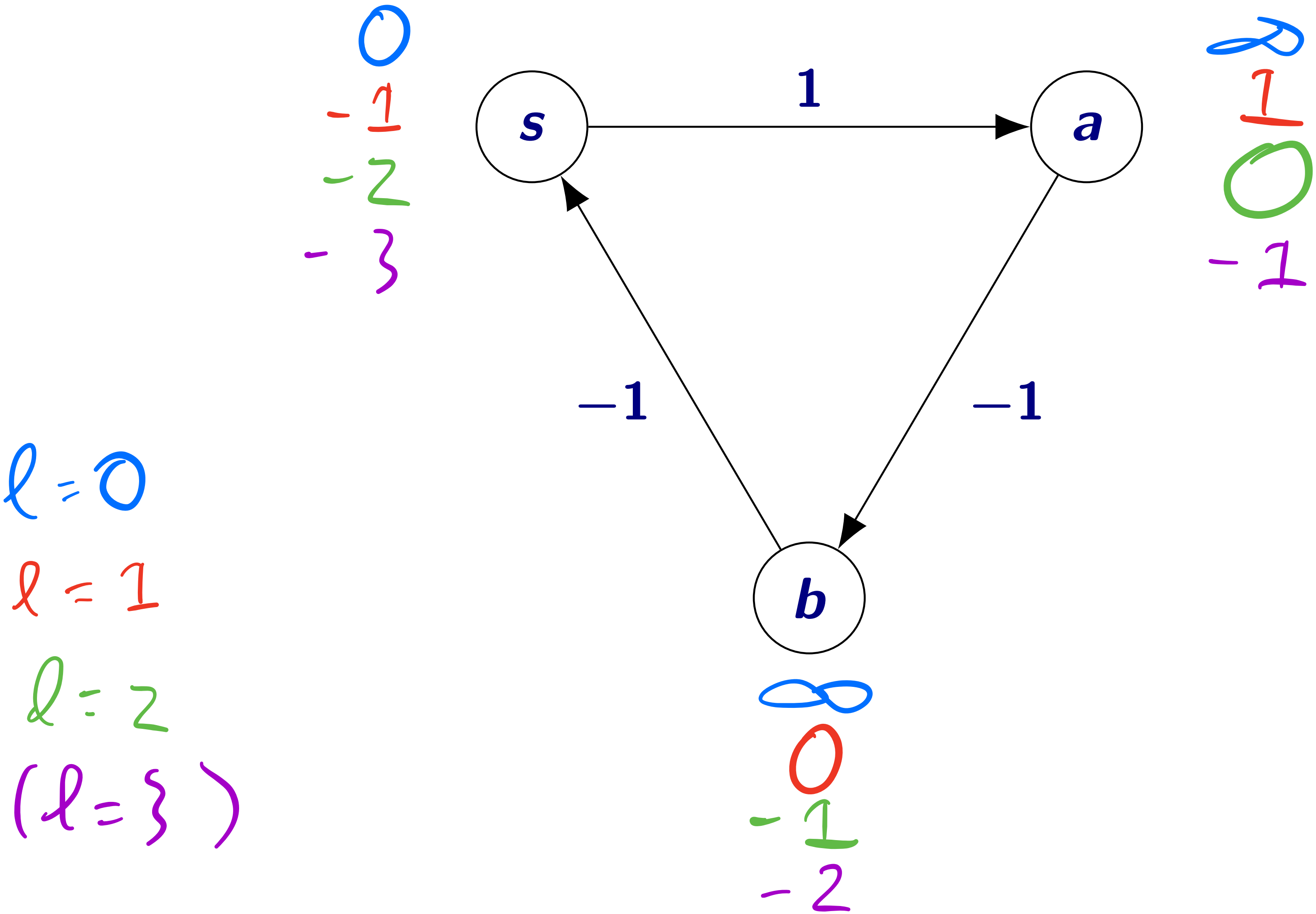
```
BellmanFord(G, s):
    Set s.dist = 0 and v.dist = ∞ for all v ≠ s
    for ℓ from 1 to V − 1:
        for all edges (u, v):
            v.dist = min(v.dist, u.dist + w(u, v))
    return u.dist for all u ∈ V
```

Correctness?

Proof by induction: after iteration i, $v.\text{dist} \leq \text{LSP}(v, i)$ for all v.

Just One Catch

What happens to Bellman-Ford on the following graph?



Negative Cycles

If our graph has a cycle with negative *total* length, our argument that the shortest path must use at most $V - 1$ edges breaks down.

Negative Cycles

If our graph has a cycle with negative *total* length, our argument that the shortest path must use at most $V - 1$ edges breaks down.

This is no accident—if there's a negative cycle, the concept of a shortest path doesn't make sense, since we can always shorten our path by taking the cycle more times!*

Negative Cycles

If our graph has a cycle with negative *total* length, our argument that the shortest path must use at most $V - 1$ edges breaks down.

This is no accident—if there's a negative cycle, the concept of a shortest path doesn't make sense, since we can always shorten our path by taking the cycle more times!*

(Technically, it's actually the concept of a shortest *walk* that breaks down. But note that all our algorithms implicitly are computing the shortest walk and using the fact that the shortest walk *must* be a path as long as there are no negative cycles. In fact, if we really want to find the shortest *path* with no restrictions on the weights at all, this turns out to be one of the canonically hard problems we study in the last part of the course!)

Detecting Negative Cycles

How can we (algorithmically) tell if a graph has a negative cycle?

Detecting Negative Cycles

How can we (algorithmically) tell if a graph has a negative cycle?

Observation: a negative cycle will cause `BellmanFord` to continue trying to update distances even after $V - 1$ iterations.

Detecting Negative Cycles

How can we (algorithmically) tell if a graph has a negative cycle?

Observation: a negative cycle will cause `BellmanFord` to continue trying to update distances even after $V - 1$ iterations.

```
BellmanFord(G, s):
    Set s.dist = 0 and v.dist = ∞ for all v ≠ s
    for ℓ from 1 to V − 1:
        for all edges (u, v):
            v.dist = min(v.dist, u.dist + w(u, v))
    for all edges (u, v):
        if v.dist > u.dist + w(u, v):  return ''Negative cycle!''
    return u.dist for all u ∈ V
```

Detecting Negative Cycles

How can we (algorithmically) tell if a graph has a negative cycle?

Observation: a negative cycle will cause `BellmanFord` to continue trying to update distances even after $V - 1$ iterations.

```
BellmanFord(G, s):
    Set s.dist = 0 and v.dist = ∞ for all v ≠ s
    for ℓ from 1 to V − 1:
        for all edges (u, v):
            v.dist = min(v.dist, u.dist + w(u, v))
    for all edges (u, v):
        if v.dist > u.dist + w(u, v):  return ``Negative cycle!''
    return u.dist for all u ∈ V
```

Exercise: modify the pseudocode to output a negative cycle.

Negative Weights Takeaways

If G has no negative cycles, we can use `BellmanFord` to solve the SSSP problem in $O(VE)$ time.

- Can also *check* if G has a negative cycle!

If G may have negative cycles, the SSSP problem is (depending on your definitions) either (1) not well-defined, or (2) one of the canonical hard problems we'll consider in the last part of the class.

Part II

All-Pairs Shortest Paths

Recall APSP

Previously, we fixed a single source s and asked for $\textbf{dist}(s, t)$ for all targets t.

Recall APSP

Previously, we fixed a single source s and asked for $\text{dist}(s, t)$ for all targets t.

More general question: output $\text{dist}(s, t)$ for *all pairs* of s and t.

(Similar to SSSP, we will need to assume no negative cycles.)

Recall APSP

Previously, we fixed a single source s and asked for $\mathrm{dist}(s, t)$ for all targets t.

More general question: output $\mathrm{dist}(s, t)$ for *all pairs* of s and t.

(Similar to SSSP, we will need to assume no negative cycles.)

Naïve approach: run SSSP from each vertex.

- Non-negative weights (Dijkstra): $O(V(V + E) \log V)$ time
- Negative weights (Bellman-Ford): $O(V^2 E)$ time

Recall APSP

Previously, we fixed a single source s and asked for $\text{dist}(s, t)$ for all targets t.

More general question: output $\text{dist}(s, t)$ for *all pairs* of s and t.

(Similar to SSSP, we will need to assume no negative cycles.)

Naïve approach: run SSSP from each vertex.

- Non-negative weights (Dijkstra): $O(V(V + E)\log V)$ time
- Negative weights (Bellman-Ford): $O(V^2E)$ time

Can we do better?

When In Doubt, Try Using DP

How would we write a DP algorithm for this?

Subproblem definition:

LSP(u, v, ℓ) is length of shortest path from u to v with $\leq \ell$ edges

Recurrence:

$$LSP(u,v,\ell) = \begin{cases} \min\left(\min_{(x,v)\in E} LSP(u,x,\ell-1) + w(x,v),\ LSP(u,v,\ell-1) \right) & \text{if } \ell > 0 \\ 0 & \text{if } \ell = 0,\ u = v \\ \infty & \text{if } \ell = 0,\ u \neq v \end{cases}$$

Evaluation Order:

ℓ from 1 to V − 1

u v arbitrary order

DP Pseudocode

Let $\text{LSP}(u, v, \ell)$ be the length of the shortest path from u to v that uses at most ℓ edges.

APSP-DP(G):
    Initialize LSP as a $V \times V \times V$ matrix
    Set $\text{LSP}[u, u, 0] = 0$ for all u
    Set $\text{LSP}[u, v, 0] = \infty$ for all $u \neq v$
    for ℓ from 1 to $V - 1$:
        for all vertices u:
            for all vertices v:
                $\text{LSP}[u, v, \ell] = \text{LSP}[u, v, \ell - 1]$
                for all edges (x, v):
                    $\text{LSP}[u, v, \ell] = \min \begin{pmatrix} \text{LSP}[u, v, \ell], \\ \text{LSP}[u, x, \ell - 1] + w(x, v) \end{pmatrix}$
    return $\text{LSP}[u, v, V - 1]$ for all $u, v \in V$

DP Pseudocode

Let $\text{LSP}(u, v, \ell)$ be the length of the shortest path from u to v that uses at most ℓ edges.

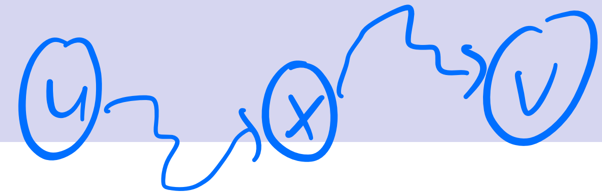
```
APSP-DP(G):
    Initialize LSP as a V × V × V matrix
    Set LSP[u, u, 0] = 0 for all u
    Set LSP[u, v, 0] = ∞ for all u ≠ v
    for ℓ from 1 to V − 1:
        for all vertices u:
            for all vertices v:
                LSP[u, v, ℓ] = LSP[u, v, ℓ − 1]
                for all edges (x, v):
                    LSP[u, v, ℓ] = min( LSP[u, v, ℓ],
                                        LSP[u, x, ℓ − 1] + w(x, v) )
    return LSP[u, v, V − 1] for all u, v ∈ V
```

Efficiency?

time for $LSP(u, v, \ell)$: $O(\deg v)$

total: $\sum_{\ell} \sum_{u} \sum_{v} O(\deg(v)) = \sum_{\ell} \sum_{u} O(E) = O(V^2 E)$!!

Improving Our Approach



Can we come up with a more efficient recurrence?

Subproblem definition: Let LSP(u, v, ℓ) be the length of the shortest path from u to v that uses at most ℓ edges.

Recurrence:

$$LSP(u,v,\ell) = \begin{cases} \min\left(\min_x \left(LSP(u,x,\tfrac{\ell}{2}) + LSP(x,v,\tfrac{\ell}{2}) \right),\ LSP(u,v,\tfrac{\ell}{2}) \right) & \text{if } \ell > 1 \\ w(u,v)^{*} & \text{if } \ell = 1 \end{cases}$$

* define $w(u,u) = 0$

$w(u,v) = \infty$ if $(u,v) \notin E$

Evaluation Order:

ℓ from $1, 2, 4, \ldots, 2^{\lceil \log_2 V \rceil}$

u, v arbitrary

Fischer-Meyer Pseudocode

Let $\text{LSP}(u, v, \ell)$ be the length of the shortest path from u to v that uses at most 2^ℓ edges.

```
FischerMeyer(G):
    Initialize LSP as a V × V × (⌈log₂(V)⌉ + 1) matrix
    Set LSP[u, v, 0] = w(u, v) for all u, v ∈ V
    # Let w(u, u) := 0, and for all (u, v) ∉ E let w(u, v) := ∞
    for ℓ from 1 to ⌈log₂(V)⌉:
        for all vertices u:
            for all vertices v:
                LSP[u, v, ℓ] = min_x (LSP[u, x, ℓ − 1] + LSP[x, v, ℓ − 1])
    return LSP[u, v, ⌈log₂(V)⌉] for all u, v ∈ V
```

Fischer-Meyer Pseudocode

Let $\text{LSP}(u, v, \ell)$ be the length of the shortest path from u to v that uses at most 2^ℓ edges.

```
FischerMeyer(G):
    Initialize LSP as a V × V × (⌈log_2(V)⌉ + 1) matrix
    Set LSP[u, v, 0] = w(u, v) for all u, v ∈ V
    # Let w(u, u) := 0, and for all (u, v) ∉ E let w(u, v) := ∞
    for ℓ from 1 to ⌈log_2(V)⌉:
        for all vertices u:
            for all vertices v:
                LSP[u, v, ℓ] = min_x (LSP[u, x, ℓ − 1] + LSP[x, v, ℓ − 1])
    return LSP[u, v, ⌈log_2(V)⌉] for all u, v ∈ V
```

Efficiency?

time per subproblem: $O(V)$

# of subproblems: $O(V^2 \log V)$

$\rightarrow O(V^3 \log V)$

A Clever Subproblem

Subproblem definition:

$LSP(u, v, r)$ is len short path from u to v that only uses intermediate vertices from $\{1, 2, \ldots, r\}$

Recurrence:

$LSP(u, v, r) = \min(\rightsquigarrow, \rightsquigarrow)$

Evaluation Order:

Clever DP Pseudocode

Let $\mathrm{LSP}(u, v, r)$ be the length of the shortest path from u to v *that only uses intermediate vertices from* $\{1, 2, \ldots r\}$.

```
APSP-CleverDP(G):
    Initialize LSP as a V × V × (V + 1) matrix
    Set LSP[u, v, 0] = w(u, v) for all u, v ∈ V
    # Let w(u, u) := 0, and for all (u, v) ∉ E let w(u, v) := ∞
    for r from 1 to V:
        for all vertices u:
            for all vertices v:
                LSP[u, v, r] = min( LSP[u, v, r − 1],
                                    LSP[u, r, r − 1] + LSP[r, v, r − 1] )
    return LSP[u, v, V] for all u, v ∈ V
```

Clever DP Pseudocode

Let LSP(u, v, r) be the length of the shortest path from u to v *that only uses intermediate vertices from* $\{1, 2, \ldots r\}$.

```
APSP-CleverDP(G):
    Initialize LSP as a V × V × (V + 1) matrix
    Set LSP[u, v, 0] = w(u, v) for all u, v ∈ V
    # Let w(u, u) := 0, and for all (u, v) ∉ E let w(u, v) := ∞
    for r from 1 to V:
        for all vertices u:
            for all vertices v:
                LSP[u, v, r] = min ( LSP[u, v, r − 1],
                                     LSP[u, r, r − 1] + LSP[r, v, r − 1] )
    return LSP[u, v, V] for all u, v ∈ V
```

Efficiency?

Simplification

Similar to Bellman-Ford, we don't *really* care that the path we find for LSP$(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{r})$ doesn't use vertices larger than $\boldsymbol{r}$—we may as well use whatever best path we've found so far!

Simplification

Similar to Bellman-Ford, we don't *really* care that the path we find for LSP(u, v, r) doesn't use vertices larger than r—we may as well use whatever best path we've found so far!

```
FloydWarshall(G):
    Set dist[u, v] = w(u, v) for all u, v ∈ V
    # Let w(u, u) := 0, and for all (u, v) ∉ E let w(u, v) := ∞
    for all vertices r:
        for all vertices u:
            for all vertices v:
                dist[u, v] = min( dist[u, v], dist[u, r] + dist[r, v] )
    return dist[u, v] for all u, v ∈ V
```

Simplification

Similar to Bellman-Ford, we don't *really* care that the path we find for LSP(u, v, r) doesn't use vertices larger than r—we may as well use whatever best path we've found so far!

```
FloydWarshall(G):
    Set dist[u, v] = w(u, v) for all u, v ∈ V
    # Let w(u, u) := 0, and for all (u, v) ∉ E let w(u, v) := ∞
    for all vertices r:
        for all vertices u:
            for all vertices v:
                dist[u, v] = min( dist[u, v], dist[u, r] + dist[r, v] )
    return dist[u, v] for all u, v ∈ V
```

Correctness?

Simplification

Similar to Bellman-Ford, we don't *really* care that the path we find for $\text{LSP}(u, v, r)$ doesn't use vertices larger than r—we may as well use whatever best path we've found so far!

```
FloydWarshall(G):
    Set dist[u,v] = w(u,v) for all u,v ∈ V
    # Let w(u,u) := 0, and for all (u,v) ∉ E let w(u,v) := ∞
    for all vertices r:
        for all vertices u:
            for all vertices v:
                dist[u,v] = min( dist[u,v], dist[u,r] + dist[r,v] )
    return dist[u,v] for all u,v ∈ V
```

Correctness? Proof by induction: after iteration i of outermost loop, $\texttt{dist}[u, v] \leq \text{LSP}(u, v, i)$ for all $u, v \in V$.

Problems and Algorithms From Today

- Shortest paths from s with no negative cycles
  - Bellman-Ford, $O(VE)$
  - Can also check for negative cycles

- All-pairs shortest paths with no negative cycles
  - Negative weights: Floyd-Warshall, $O(V^3)$
  - Non-negative weights: Djikstra from each vertex, $O(V(V+E)\log V)$ (Floyd-Warshall is more efficient if $E \approx V^2$)