全源最短路径
All-Pairs Shortest Path

钮鑫涛

Nanjing University

2024 Fall

SSSP and APSP

- Single-Source Shortest Paths (SSSP) Problem:

  ‣ Given a graph $G = (V, E)$ and a weight function w, given a source node s, find a shortest path from s to every $u \in V$.

- All-Pairs Shortest Paths (APSP) Problem:

  ‣ Given a graph $G = (V, E)$ and a weight function w, for every pair $(u, v) \in V \times V$, find a shortest path from u to v.

APSP from multiple SSSP

- Straightforward solution for APSP: For each $v \in V$, execute SSSP algorithm once!

|	SSSP	APSP
BFS **(Unit-weight graphs)**	$O(n + m) = O(n^2)$	$O(n^3)$
Dijkstra **(Positive-weight graphs)**	$O((n + m)\lg n) = O(n^2 \lg n)$ (Using binary heap for priority queue)	$O(n^3 \lg n)$
Bellman-Ford **(Arbitary-weight Directed)**	$O(nm) = O(n^3)$	$O(n^4)$
Topological Sort Variant **(Arbitrary-weight DAG)**	$O(n + m) = O(n^2)$	$O(n^3)$

APSP from multiple SSSP

- Positive-weight Graphs: Repeating Dijkstra gives $O(n^3 \lg n)$.

- Arbitrary-weight Graphs: Repeating Bellman-Ford gives $O(n^4)$.

- Faster algorithms for arbitrary-weight graphs?

  ‣ **Intuition**: modify edge weights without changing shortest path, so that Dijkstra's algorithm can work.

APSP from multiple SSSP

- **Intuition**: modify edge weights without changing shortest path, so that Dijkstra's algorithm can work.

  ‣ Add $\max\{-1 \cdot w(u, v)\}$ to each edge?

- NO! Shortest paths may change!

- Given (u, v), different paths may change by different amount!
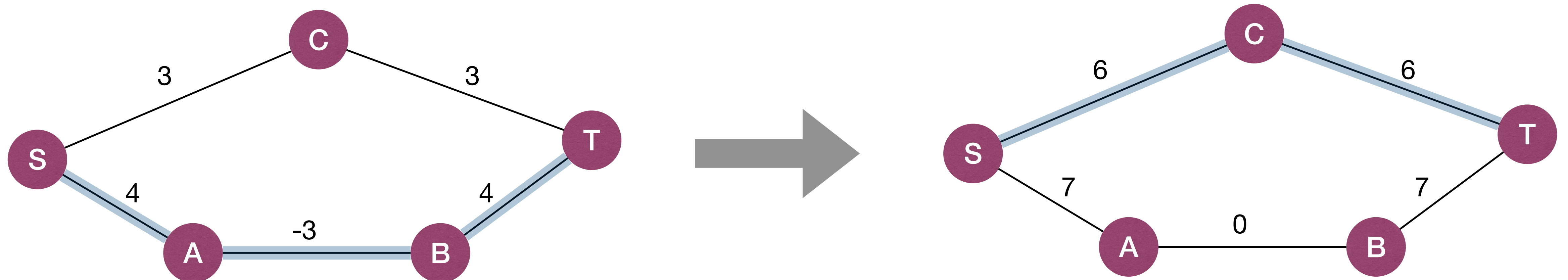
APSP from multiple SSSP

- Faster algorithms for arbitrary-weight graphs?

  ‣ **Intuition**: modify edge weights without changing shortest path, so that Dijkstra's algorithm can work.

  ‣ **Requirement**: $\hat{w}(u \overset{p_1}{\rightsquigarrow} v) > \hat{w}(u \overset{p_2}{\rightsquigarrow} v) \iff w(u \overset{p_1}{\rightsquigarrow} v) > w(u \overset{p_2}{\rightsquigarrow} v)$

    new weight of path

  ‣ **Or alternatively**, for every path from u to v, $\hat{w}$ changes it by the **same** amount:

    - Let the $\hat{w}(u, v) = h(u) + w(u, v) - h(v)$

    new weight of edge

    - Imagine $h(u)$ is **entry gift** and $h(v)$ is **exit tax** for traveling through (u, v).

APSP from multiple SSSP

- $\hat{w}(u \xrightarrow{p_1} v) = \hat{w}(u \to x_1 \to \ldots \to x_k \to v) = \hat{w}(u \to x_1) + \ldots + \hat{w}(x_k \to v)$

$$= \left(h(u) + w(u \to x_1) - h(x_1) \right) + \left(h(x_1) + w(x_1 \to x_2) - h(x_2) \right) + \ldots +$$
$$\left(h(x_{k-1}) + w(x_{k-1} \to x_k) - h(x_k) \right) + \left(h(x_k) + w(x_k \to v) - h(v) \right)$$

$$= h(u) + w(u \to x_1) + \ldots + w(x_k \to v) - h(v)$$

$$= h(u) + w(u \to x_1 \to \ldots \to x_k \to v) - h(v) = h(u) + w(u \xrightarrow{p_1} v) - h(v)$$

APSP from multiple SSSP

- Since we need $\hat{w}(u, v) = h(u) + w(u, v) - h(v) \geq 0$ (for Dijkstra algorithm)

- Just let $h(u) = dist(z, u)$ for some fixed $z \in V$, then
$\hat{w}(u, v) = dist(u) + w(u, v) - dist(v) \geq 0$

> The shortest path from z to v must be "smaller than" the shortest path from z to u add the edge from u to v.
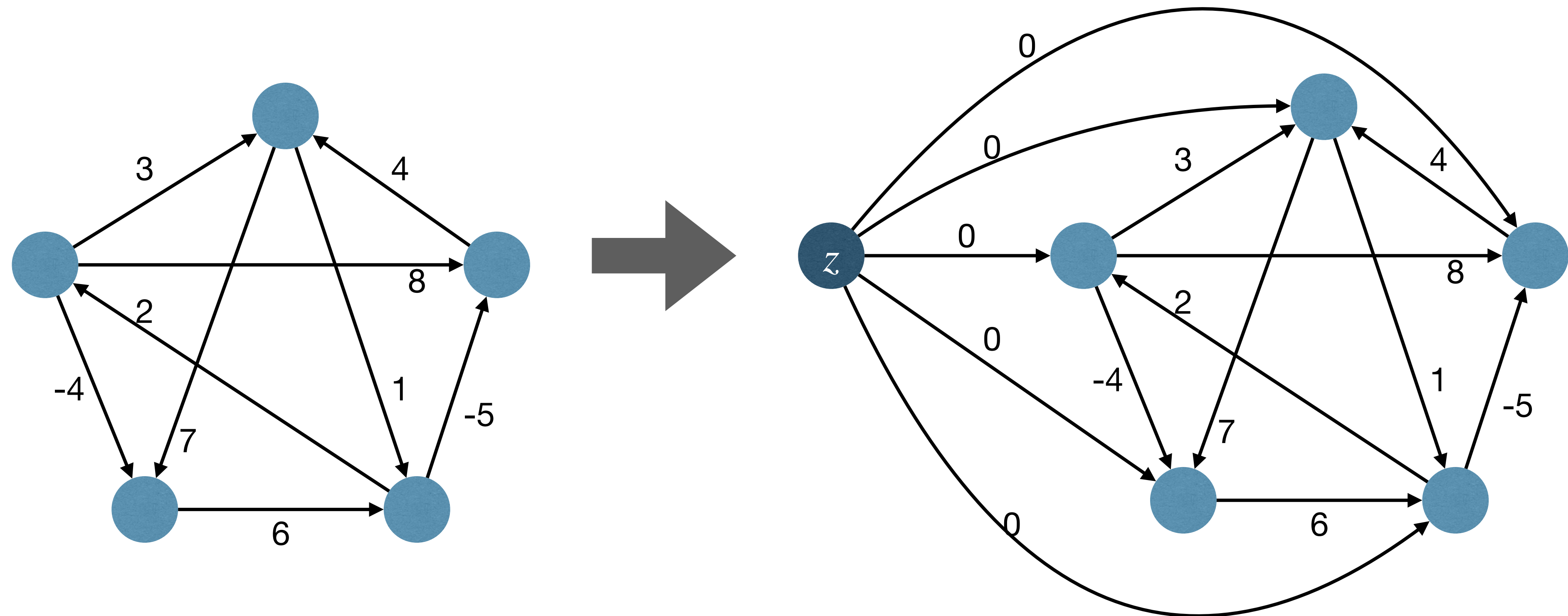
  ‣ But it is possible that we cannot find such z that reaches every node.

APSP from multiple SSSP

- Add node z that goes to every node in G with a weight 0 edge.

  ‣ $H = (V \cup \{z\}, E \cup \{(z, x) \mid x \in V\})$ with $w(z, x) = 0$

APSP from multiple SSSP

- Re-weight edges:
$\hat{w}(u, v) = dist(u) + w(u, v) - dist(v) \geq 0$

  ‣ For node pairs in G, addition of z does not create new shortest path.

  ‣ For node pairs in G, a path is shortest under w iff this path is shortest under $\hat{w}$.

  Proposed by Donald Bruce Johnson

JohnsonAPSP(G,s):

Create $H := (V+\{z\}, E+\{(z, v) \mid v \in V\})$ with $w(z,v) = 0$

Bellman-FordSSSP(H,z) to obtain $dist_H$

for **each** *edge (u, v)* **in** *H.E*

    $w'(u,v) := dist_H(z,u) + w(u,v) - dist_H(z,v)$

for **each** *node u* **in** *G.V*

    DijkstraSSSP(G,u) with w' to obtain $dist_{G,w'}$

    for **each** *node v* **in** *G.V*

        $dist_G(u,v) := dist_{G,w'}(u,v) + dist_H(z,v) - dist_H(z,u)$
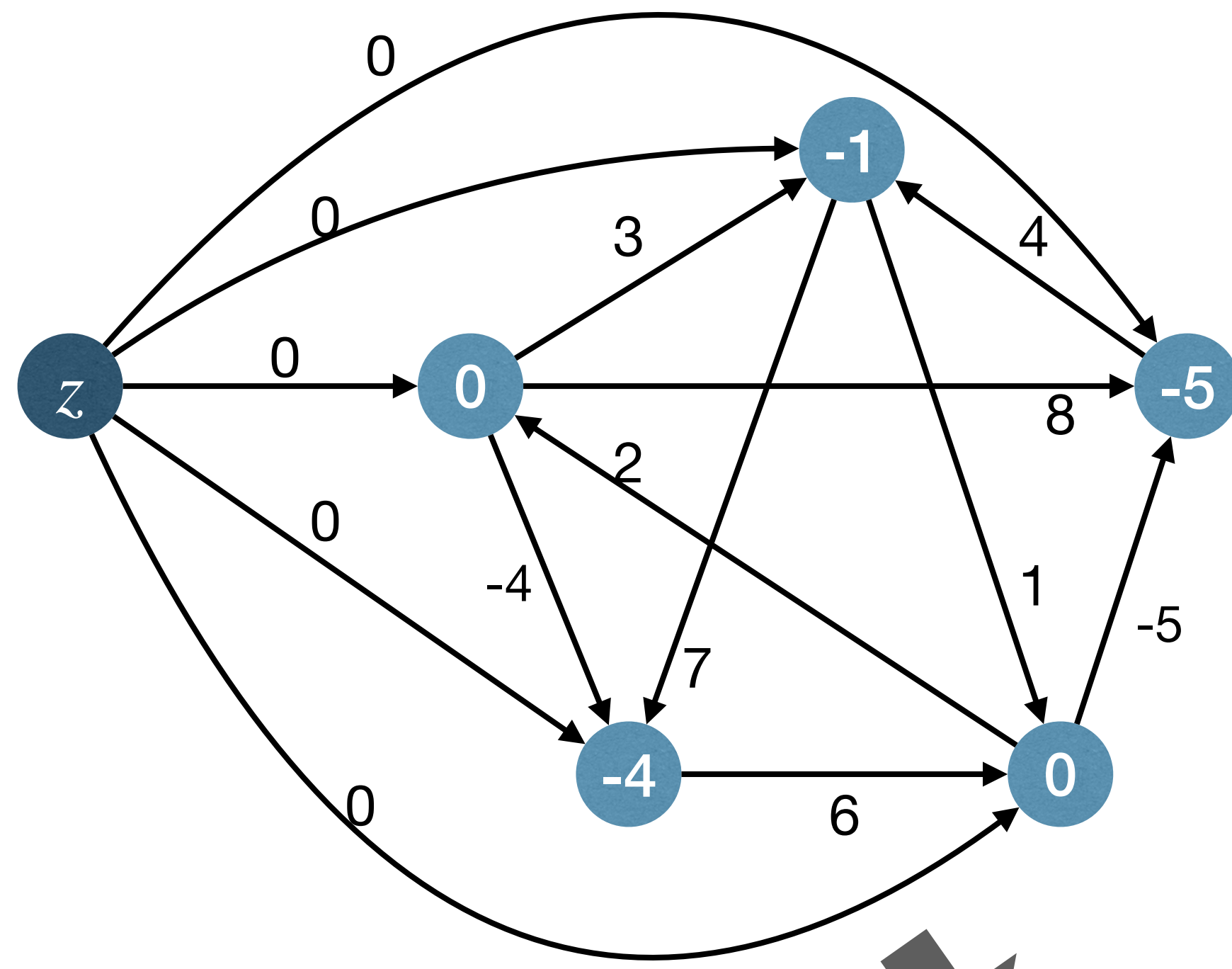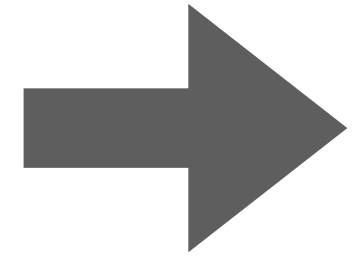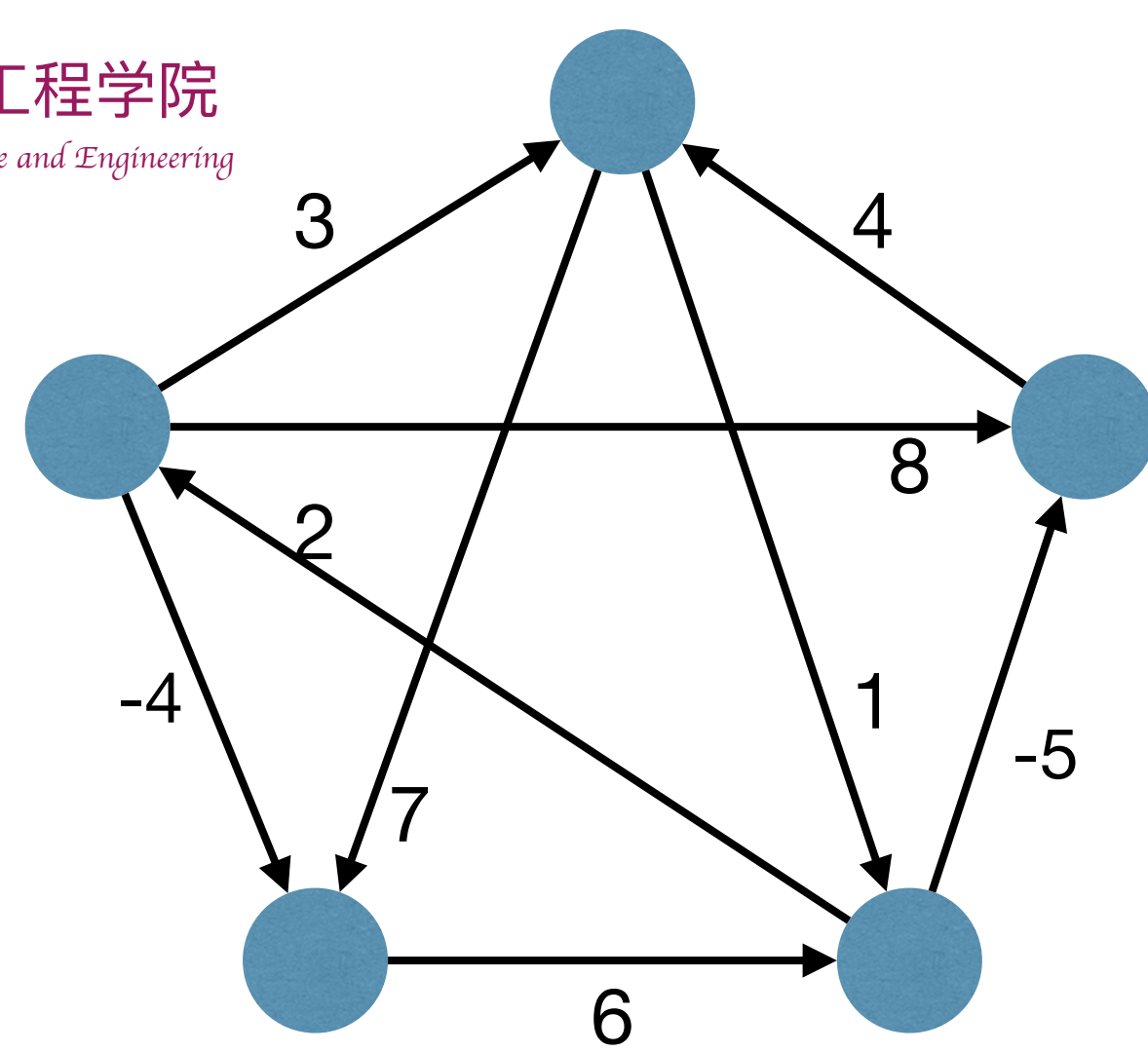
Add a new node z
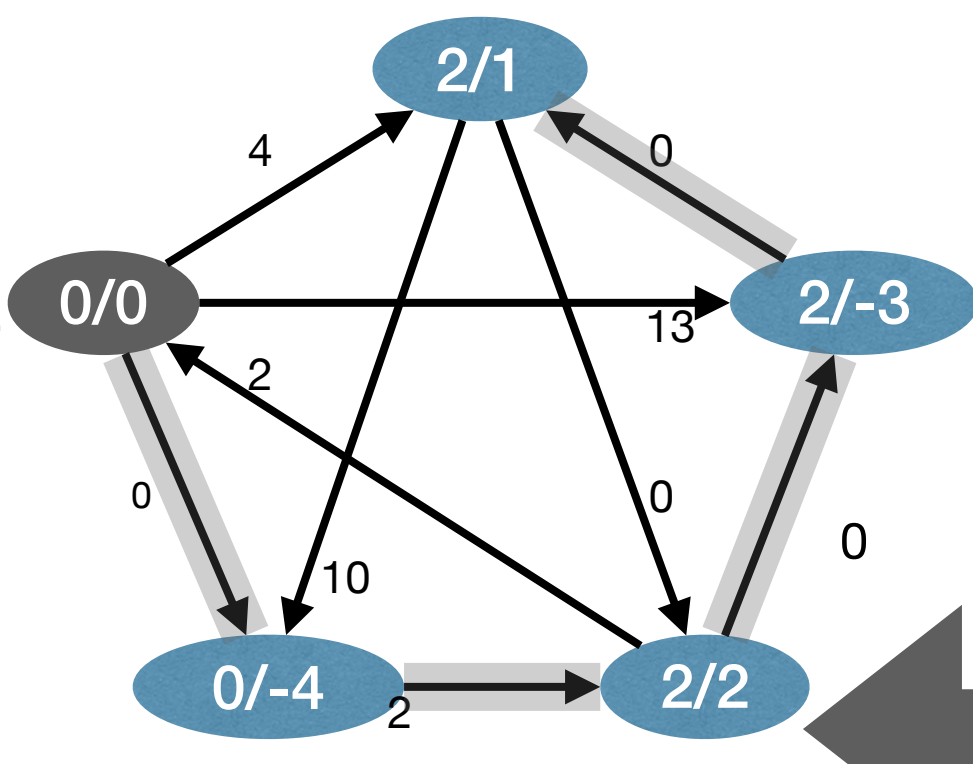Compute each $h(u)$

re-weighting each edge
By $\hat{w}(u,v) = h(u) + w(u,v) - h(v)$
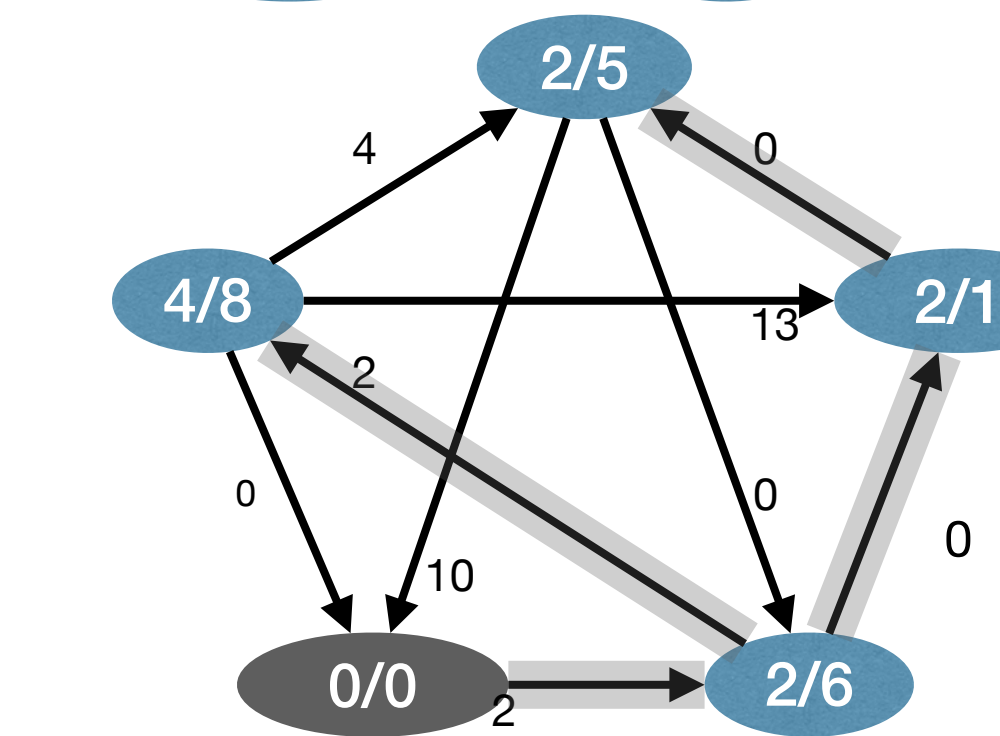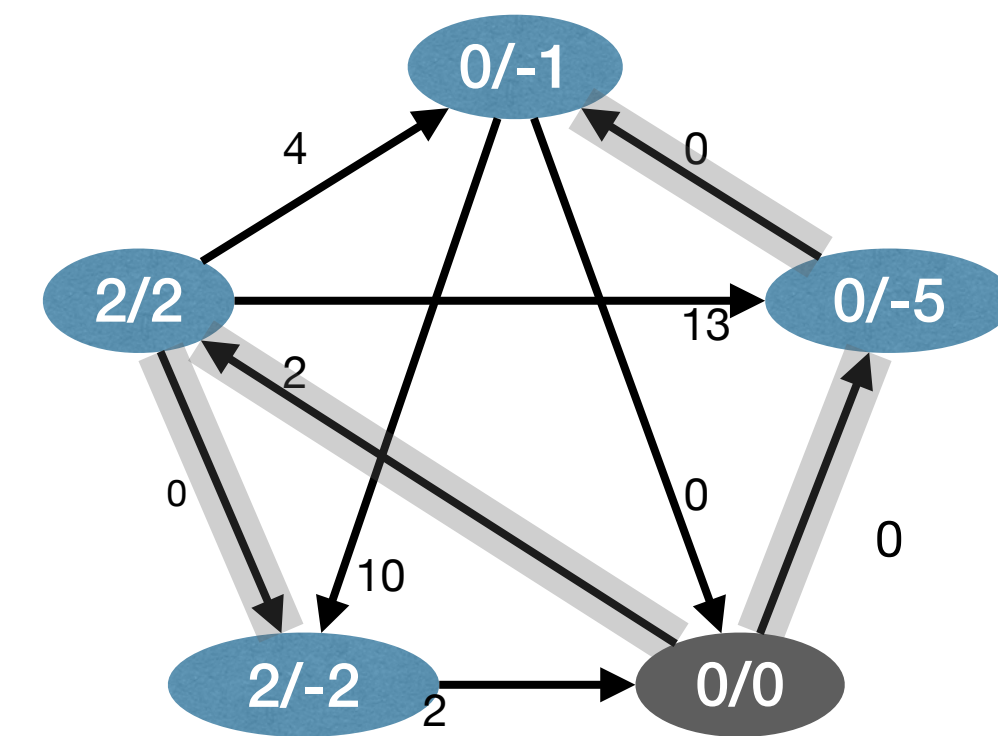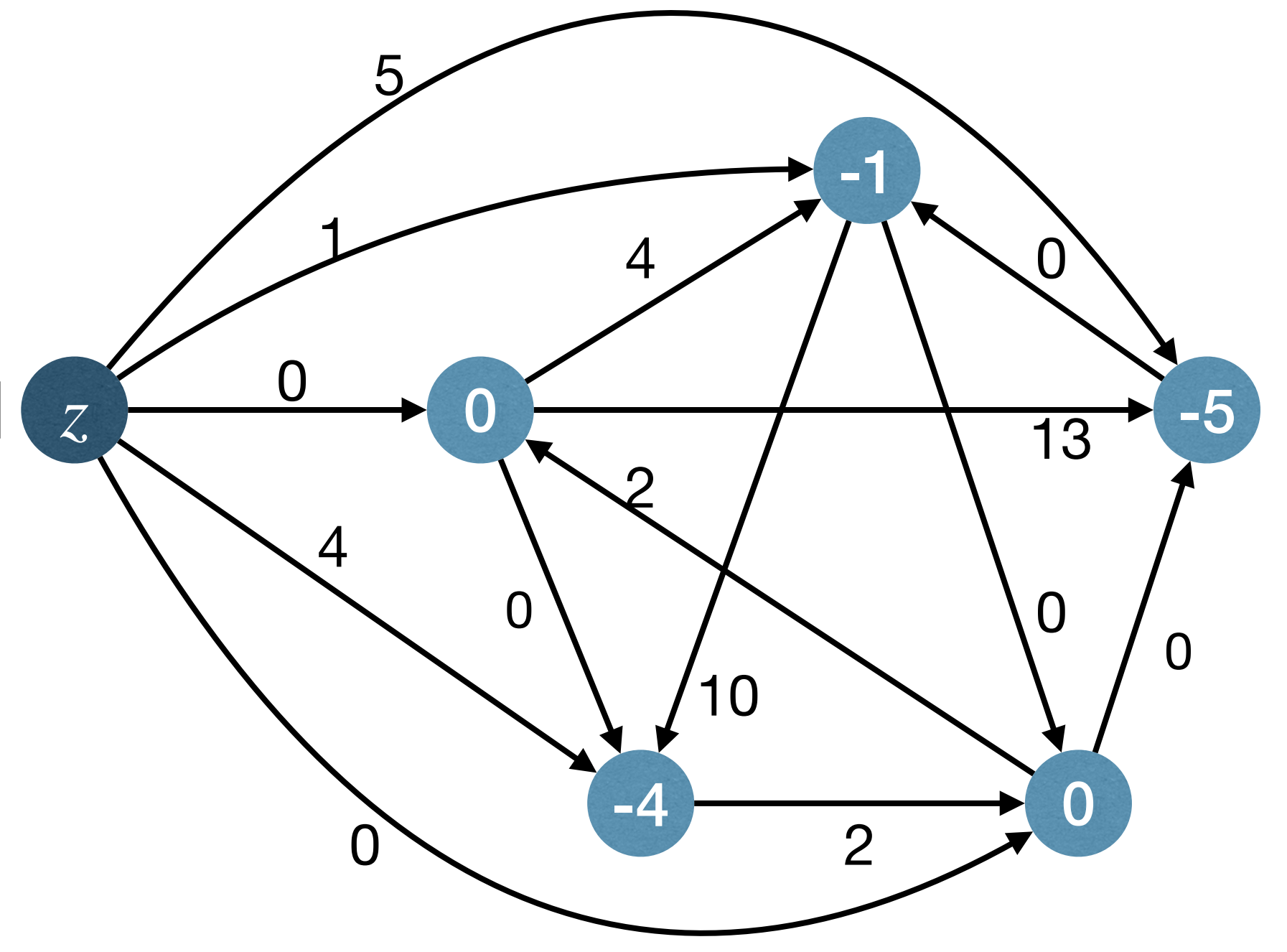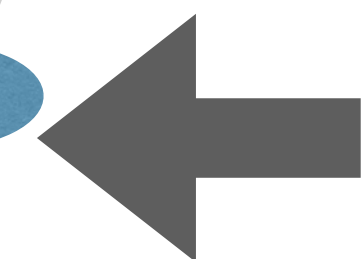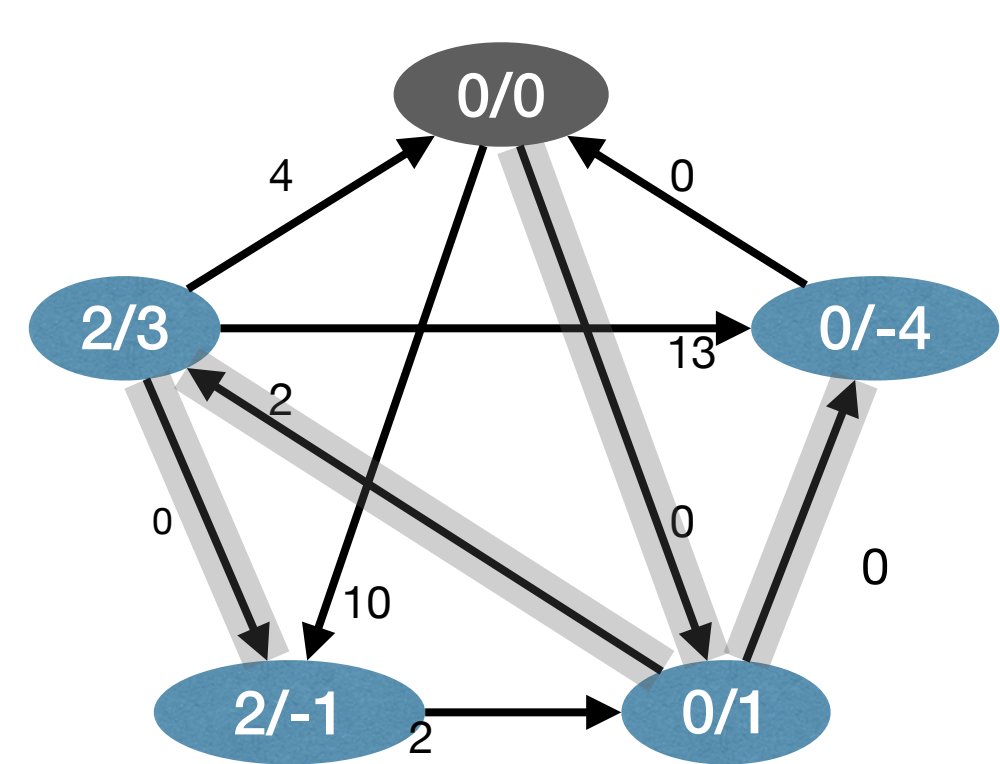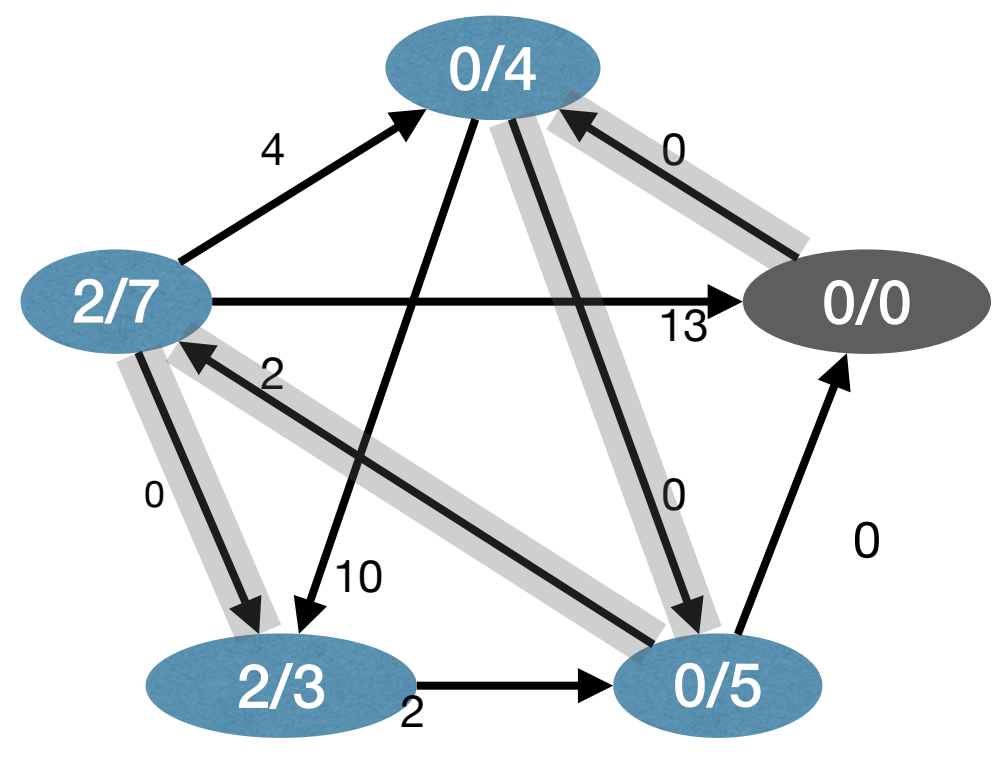
$\text{dist}_{G,w'}(u,v)$ / $\text{dist}_{G,w}(u,v)$
in each node

APSP from multiple SSSP

- Johnson's algorithm combines Dijkstra and Bellman-Ford, resulting a runtime of $O(n^3 \lg n)$, for arateary weight graphs.

<u>JohnsonAPSP(G,s):</u>

Create H := (V+{z}, E+{(z, v) | v∈V}) with w(z,v) = 0

Bellman-FordSSSP(H,z) to obtain dist$_H$

for each *edge (u, v)* **in** *H.E*

   w'(u,v) := dist$_H$(z,u) + w(u,v) - dist$_H$(z,v)

for each *node u* **in** *G.V*

   DijkstraSSSP(G,u) with w' to obtain dist$_{G,w'}$

   for each *node v* **in** *G.V*

      dist$_G$(u,v) := dist$_{G,w'}$(u,v) + dist$_H$(z,v) - dist$_H$(z,u)

Floyd-Warshall Algorithm

APSP via Recursion

- $dist(u, v) = \begin{cases} 0 & if\ u = v \\ \min_{(x,v)\in E}\{dist(u, x) + w(x, v)\} & otherwise \end{cases}$

- This recurrence is correct, but it does not lead to a recursive algorithm directly!

  ‣ Cycle in the graph can make the recursion never ends!
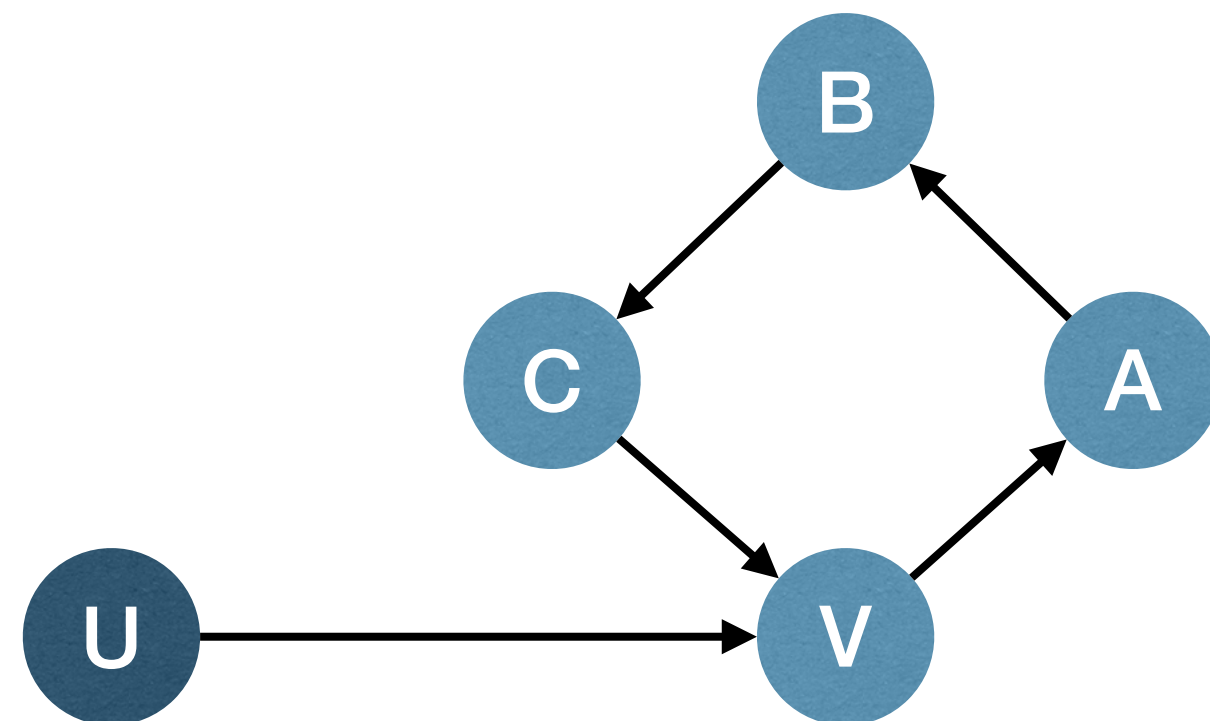
APSP via Recursion

- Introduce an additional parameter in the recurrence:

  ▸ $dist(u, v, l)$: shortest path from u to v that uses at most l edges.

- $dist(u, v) = \begin{cases} 0 & if\ l = 0\ and\ u = v \\ \infty & if\ l = 0\ and\ u \neq v \\ \min \begin{cases} dist(u, v, l-1) \\ \min\limits_{(x,v) \in E} \{dist(u, x, l-1) + w(x, v)\} \end{cases} & otherwise \end{cases}$

APSP via Recursion

- $dist(u, v) = \begin{cases} 0 & \text{if } l = 0 \text{ and } u = v \\ \infty & \text{if } l = 0 \text{ and } u \neq v \\ \min \left\{ \begin{array}{l} dist(u, v, l - 1) \\ \min\limits_{(x,v) \in E} \{ dist(u, x, l - 1) + w(x, v) \} \end{array} \right\} & otherwise \end{cases}$

- Evaluate this recurrence easily in a "**bottom-up**" fashion!

  ‣ $dist(\cdot, \cdot, 0)$ are easy to compute, given input graph.

  ‣ $dist(\cdot, \cdot, 1)$ are easy to compute, if $dist(\cdot, \cdot, 0)$ are known.

  ‣ $dist(\cdot, \cdot, l + 1)$ are easy to compute, if $dist(\cdot, \cdot, l)$ are known.

  ‣ $dist(\cdot, \cdot, n - 1)$ are what we want!

Don't always need a recursive algorithm to evaluate recurrence, often an iterative alternative exists.

APSP via Recursion

RecursiveAPSP(G):

for **each** *pair* (*u*,*v*) **in** *V*V*

     if *u* = *v* then *dist*[*u*,*v*,0] := 0

     else *dist*[*u*,*v*,0] := *INF*

for *l* := 1 to *n* -1

     for **each** *node u*

         for **each** *node v*

             dist[*u*,*v*,*l*] := *dist*[*u*,*v*, *l* - 1]

             for **each** *edge* (*x*,*v*) going to *v*

                 if *dist*[*u*,*v*,*l*] > *dist*[*u*,*x*,*l* - 1] + *w*(*x*,*v*)

                    dist[*u*,*v*,*l*] := *dist*[*u*,*x*,*l* - 1] + *w*(*x*,*v*)

Time complexity:
$O(n^4)$

Can we do better?

APSP via Recursion

- $dist(u, v) = \begin{cases} 0 & \text{if } l = 0 \text{ and } u = v \\ \infty & \text{if } l = 0 \text{ and } u \neq v \\ \min \begin{cases} dist(u, v, l-1) \\ \min_{(x,v) \in E} \{dist(u, x, l-1) + w(x, v)\} \end{cases} & \text{otherwise} \end{cases}$

- This recursion is like "1 and $l - 1$ split" in divide-and-conquer. How about "$l/2$ and $l/2$ split"?

- $dist(u, v, l) = \begin{cases} w(u, v) & \text{if } l = 1 \text{ and } (u, v) \in E \\ \infty & \text{if } l = 1 \text{ and } (u, v) \notin E \\ \min_{x \in V}\{dist(u, x, l/2) + dist(x, v, l/2)\} & \text{otherwise} \end{cases}$

- Start with $dist(\cdot, \cdot, 1)$, then double l each time, until $2^{\lceil \lg n \rceil}$.

APSP via Recursion

FasterRecursiveAPSP(G):

for **each** *pair* (*u*,*v*) **in** *V*V*

    if (*u*, *v*) **in** *E* then *dist*[*u*,*v*,1] := *w*(*u*, *v*)

    else *dist*[*u*,*v*,1] := *INF*

for *i* := 1 **to** $\lceil \lg n \rceil$

    for **each** *node u*

        for **each** *node v*

            $dist[u,v, 2^i] := INF$

            for **each** *node x*

                if $dist[u,v,2^i] > dist[u,x,2^{i-1}] + dist[x,v,2^{i-1}]$

                    $dist[u,v,2^i] := dist[u,x,2^{i-1}] + dist[x,v, 2^{i-1}]$

> Time complexity:
> $O(n^3 \lg n)$

Can this approach be better?

APSP via Recursion

- **Strategy:** recuse on the *set of node* the shortest paths use.(Previous algorithms recuse on number of edges the shortest paths use.)

- Number the vertices arbitrarily: $x_1, x_2, \ldots, x_n$; Define $V_r = \{x_1, x_2, \ldots, x_r\}$ to be the set of vertices numbered at most r.

- Define $dist(u, v, r)$ be length of shortest path from u to v, s.t. only nodes in V_r can be *intermediate* nodes in paths. Let $\pi(u, v, r)$ be such a shortest path.

$$dist(u, v) = dist(u, v, 6) = w(u \to x_5 \to x_6 \to v)$$

$$dist(u, v, 4) = w(u \to x_1 \to x_2 \to x_3 \to v)$$
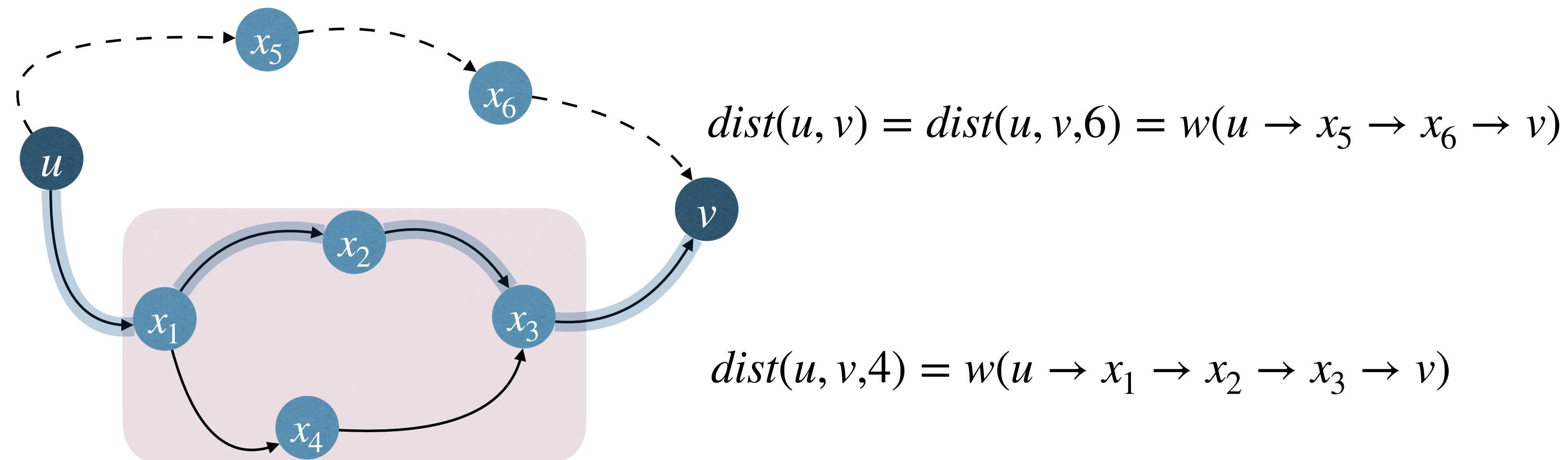
APSP via Recursion

- **Observation:** either $\pi(u, v, r)$ goes through x_r or not.

- **Latter case:** $\pi(u, v, r) = \pi(u, v, r - 1)$

- **Former case:** $\pi(u, v, r) = \pi(u, x_r, r) + \pi(x_r, v, r) = \pi(u, x_r, r - 1) + \pi(x_r, v, r - 1)$

$$\pi(u, v, r) = \pi(u, v, r - 1)$$
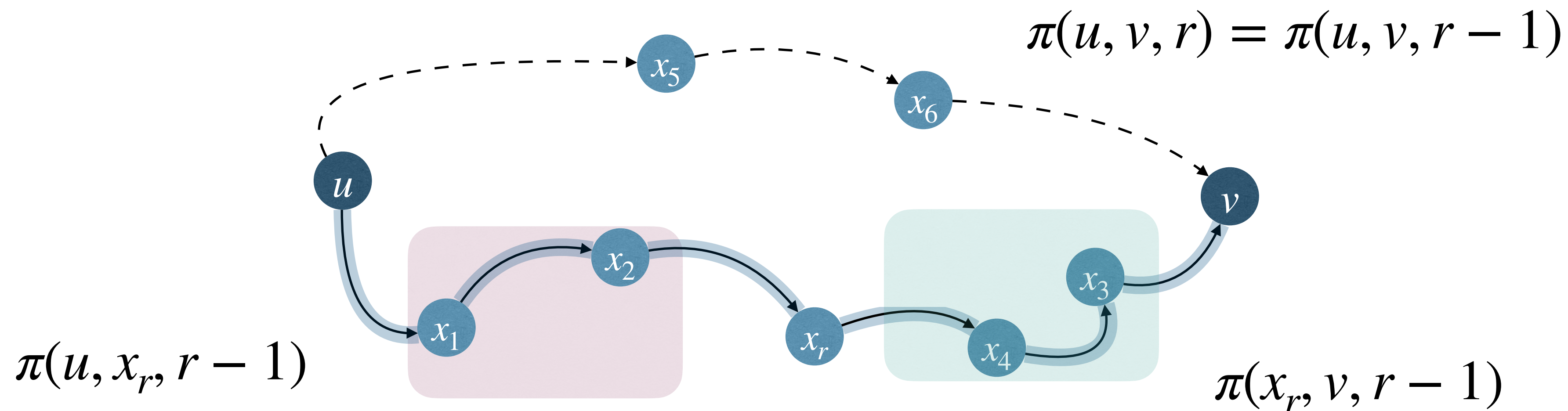
$\pi(u, x_r, r - 1)$

$\pi(x_r, v, r - 1)$

The Floyd-Warshall Algorithm

- $dist(u, v, r) = \begin{cases} w(u, v) & if \ r = 0 \ and \ (u,v) \in E \\ \infty & if \ r = 0 \ and \ (u,v) \notin E \\ \min \begin{cases} dist(u, v, r - 1) \\ dist(u, x_r, r - 1) + dist(x_r, v, r - 1) \end{cases} & otherwise \end{cases}$

Bernard Roy   **Robert W. Floyd**   **Stephen Warshall**

FloydWarshallAPSP(G):
for **each** *pair* (u,v) **in** *V*V*
    if (u, v) **in** *E* **then** *dist*[u,v, 0] := *w*(u, v)
    else *dist*[u,v,0] := *INF*
for *r* := 1 **to** *n*
    for each *node u*
        for each *node v*
            dist[u,v,r] := *dist*[u,v,r - 1]
            if *dist*[u,v,r] > *dist*[u,x_r, r - 1] + *dist*[x_r,v, r - 1]
                dist[u,v,r] := *dist*[u,x_r, r - 1] + *dist*[x_r,v, r - 1]

Time complexity:
$O(n^3)$

Transitive Closure of a directed graph

- Given directed graph $G = (V, E)$ with vertex set $V = \{1,2,...,n\}$, define the **transitive closure** of G as the graph $G^* = (V, E^*)$, where

  ‣ $E^* = \{(i, j) : \text{there is a path from vertex } i \text{ to } j \text{ in } G\}$.

- Just assign weight 1 to each edge, and run Floyd-Warshall. Then if there is a path between u and v, $dist(u, v) < n$, otherwise $dist(u, v) = \infty$

- Or alternatively (and more efficiently), use $\vee$ (logical Or) and $\wedge$ (logical And) for the arithmetic operations $\min$ and $+$, and Define $t_{u,v}^{(k)}$ to indicate if there is a path from u to v with all intermediate vertices in $\{1,2,...,k\}$:

  ‣ $t_{uv}^{(0)} = \begin{cases} 0, & \text{if } u \neq v \text{ and } (u, v) \notin E \\ 1, & \text{if } u = v \text{ or } (u, v) \in E \end{cases}$

  ‣ For $k \geq 1$, $t_{u,v}^{(k)} = t_{u,v}^{(k-1)} \vee \left(t_{u,x_k}^{(k-1)} \wedge t_{x_k,v}^{(k-1)} \right)$

Application of APSP: Compute Transitive Closure

<u>FloydWarshallAPSP(G):</u>

for **each** *pair* (u,v) **in** $V*V$
    if (u, v) **in** E **then** $dist[u,v, 0] := w(u, v)$
    else $dist[u,v,0] := INF$
for $r := 1$ **to** n
    for **each** *node u*
        for **each** *node v*
            $dist[u,v,r] := dist[u,v,r - 1]$
            if $dist[u,v,r] > dist[u,x_r, r - 1] + dist[x_r,v, r - 1]$
                $dist[u,v,r] := dist[u,x_r, r - 1] + dist[x_r,v, r - 1]$

<u>FloydWarshallTransitiveClosure(G):</u>

for **each** *pair* (u,v) **in** $V*V$
    if (u, v) **in** E **then** $t[u,v, 0] :=$ TRUE
    else $t[u,v,0] :=$ FALSE
for $r := 1$ **to** n
    for **each** *node u*
        for **each** *node v*
            $t[u,v,r] := t[u,v,r - 1]$
            if $t[u,x_r, r - 1]$ **AND** $t[x_r,v, r - 1]$
                $t[u,v,r] :=$ TRUE

Further reading

- [CLRS] Ch.25

- [Erickson] Ch.9