3-8 Cool? We are APSP Algorithms.

Hengfeng Wei

hfwei@nju.edu.cn

November 19, 2018

Please Help Me Out Here.

Definition (Shortest Path)

$$G = (V, E, w): \text{ weighted digraph}$$

$$\delta(u, v) = \begin{cases} \min \{w(p) : u \overset{p}{\leadsto} v\} & \text{if } u \leadsto v \\ \infty & \text{o.w.} \end{cases}$$

Path *vs.* Simple path

Definition (Shortest Path)

$$G = (V, E, w): \text{ weighted digraph}$$

$$\delta(u, v) = \begin{cases} \min \{w(p) : u \overset{p}{\leadsto} v\} & \text{if } u \leadsto v \\ \infty & \text{o.w.} \end{cases}$$

Path *vs.* Simple path

Shortest-path Problem *vs.* Longest-path Problem

Digraph *vs.* Undirected Graph

Single Source Digraph

Shortest-path Problem *vs.* Longest-path Problem

Single Source Digraph

Shortest-path Problem *vs.* Longest-path Problem

$$\mathsf{SP} \text{ in } G \iff \mathsf{LP} \text{ in } -G$$

Single Source   Digraph

Shortest-path Problem *vs.* Longest-path Problem

$$\textsf{SP} \text{ in } G \iff \textsf{LP} \text{ in } -G$$

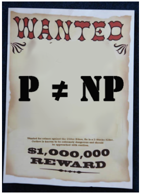
$O(VE)$ (Bellman-Ford) *vs.* NP-hard (I just told you.)

Single Source   Digraph

Shortest-path Problem *vs.* Longest-path Problem

$$\textsf{SP in } G \iff \textsf{LP in } -G$$

$O(VE)$ (Bellman-Ford) *vs.* NP-hard (I just told you.)

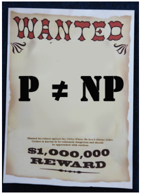


Single Source   Digraph

Shortest-path Problem   *vs.*   Longest-path Problem

$$\textsf{SP in } G \iff \textsf{LP in } -G$$

$O(VE)$ (Bellman-Ford)   *vs.*   NP-hard (I just told you.)



Definition (Shortest Path)

$$G = (V, E, w): \text{ weighted digraph}$$

$$\delta(u, v) = \begin{cases} \min \{ w(p) : \underbrace{u \overset{p}{\leadsto} v}_{\text{Path}} \} & \text{if } u \leadsto v \\ \infty & \text{o.w.} \end{cases}$$

Definition (Shortest Path)

$$G = (V, E, w): \text{ weighted digraph}$$

$$\delta(u,v) = \begin{cases} \min \{ w(p) : \underbrace{u \overset{p}{\leadsto} v}_{\text{Path}} \} & \text{if } u \leadsto v \\ \infty & \text{o.w.} \end{cases}$$

Q : How does Bellman-Ford Handle with Negative-weight Cycles?

Definition (Shortest Path)

$$G = (V, E, w) : \text{ weighted digraph}$$

$$\delta(u, v) = \begin{cases} \min \{ w(p) : \underbrace{u \overset{p}{\leadsto} v}_{\text{Path}} \} & \text{if } u \leadsto v \\ \infty & \text{o.w.} \end{cases}$$

Q : How does Bellman-Ford Handle with Negative-weight Cycles?

A : Report it and Treat Shortest Path as "Undefined".

Definition (Shortest Path)

$$G = (V, E, w): \text{ weighted digraph}$$

$$\delta(u, v) = \begin{cases} \min\left\{w(p) : \underbrace{u \overset{p}{\leadsto} v}_{\text{Path}}\right\} & \text{if } u \leadsto v \\ \infty & \text{o.w.} \end{cases}$$

Q : How does Bellman-Ford Handle with Negative-weight Cycles?

A : Report it and Treat Shortest Path as "Undefined".

$$O(VE)$$

Definition (Shortest Simple Path)

$$G = (V, E, w): \text{ weighted digraph}$$

$$\delta(u, v) = \begin{cases} \min \{ w(p) : \underbrace{u \overset{p}{\leadsto} v}_{\text{Simple Path}} \} & \text{if } u \leadsto v \\ \infty & \text{o.w.} \end{cases}$$

Definition (Shortest Simple Path)

$$G = (V, E, w) : \ \text{weighted digraph}$$

$$\delta(u,v) = \begin{cases} \min\left\{ w(p) : \underbrace{u \overset{p}{\rightsquigarrow} v}_{\text{Simple Path}} \right\} & \text{if } u \rightsquigarrow v \\ \infty & \text{o.w.} \end{cases}$$

Q : How Should an Algorithm for Shortest Simple Path Problem Handle with Negative-weight Cycles?

Definition (Shortest Simple Path)

$$G = (V, E, w): \text{ weighted digraph}$$

$$\delta(u,v) = \begin{cases} \min \{ w(p) : \underbrace{u \overset{p}{\leadsto} v}_{\text{Simple Path}} \} & \text{if } u \leadsto v \\ \infty & \text{o.w.} \end{cases}$$

Q : How Should an Algorithm for Shortest Simple Path Problem Handle with Negative-weight Cycles?

A : Still to Find Shortest Simple Path.

Definition (Shortest Simple Path)

$$G = (V, E, w): \text{ weighted digraph}$$

$$\delta(u,v) = \begin{cases} \min \{ w(p) : \underbrace{u \overset{p}{\rightsquigarrow} v}_{\text{Simple Path}} \} & \text{if } u \rightsquigarrow v \\ \infty & \text{o.w.} \end{cases}$$

Q : How Should an Algorithm for Shortest Simple Path Problem Handle with Negative-weight Cycles?

A : Still to Find Shortest Simple Path.

NP-hard

Shortest Path Problem

Shortest Simple Path Problem

Shortest Path Problem

Longest Path Problem

Shortest Simple Path Problem

Longest Simple Path Problem

Shortest Path Problem $\iff$ Longest Path Problem

Shortest Simple Path Problem $\iff$ Longest Simple Path Problem

Single Source Undirected Graph

Single Source   Undirected Graph

Negative-weight edges allowed   (Why?)

Single Source Undirected Graph

Negative-weight edges allowed (Why?)

Simple path (Why?)

Single Source Undirected Graph

Negative-weight edges allowed (Why?)

Simple path (Why?)

No negative-weight cycles (o.w., NP-hard)

Single Source   Undirected Graph

Negative-weight edges allowed   (Why?)

Simple path   (Why?)

No negative-weight cycles   (o.w., NP-hard)

Single-source $s \rightsquigarrow$ Single-target t

Shortest Path Algorithms

Luis Goddyn, Math 408

Given an edge weighted graph (G, d), $d : E(G) \to \mathbb{Q}$ and two vertices $s, t \in V(G)$, the *Shortest Path Problem* is to find an s,t-path P whose total weight is as small as possible. Here, G may be either directed or undirected. A path in a graph is a sequence $v_0 e_1, v_1, \ldots, v_k$ of vertices and edges such that no vertex or edge appears twice, and e_i joins v_{i-1} to v_i. If G is directed, then e_i should be oriented from v_{i-1} to v_i.

Minimum-weight Perfect Matching

Shortest Path Algorithms

Luis Goddyn, Math 408

Given an edge weighted graph (G, d), $d : E(G) \to \mathbb{Q}$ and two vertices $s, t \in V(G)$, the *Shortest Path Problem* is to find an s, t-path P whose total weight is as small as possible. Here, G may be either directed or undirected. A path in a graph is a sequence $v_0 e_1, v_1, \ldots, v_k$ of vertices and edges such that no vertex or edge appears twice, and e_i joins v_{i-1} to v_i. If G is directed, then e_i should be oriented from v_{i-1} to v_i.

Minimum-weight Perfect Matching

We leave it to the reader to $\cdots$

Shortest Path Algorithms

Luis Goddyn, Math 408

Given an edge weighted graph (G, d), $d : E(G) \to \mathbb{Q}$ and two vertices $s, t \in V(G)$, the *Shortest Path Problem* is to find an s,t-path P whose total weight is as small as possible. Here, G may be either directed or undirected. A path in a graph is a sequence $v_0 e_1, v_1, \ldots, v_k$ of vertices and edges such that no vertex or edge appears twice, and e_i joins v_{i-1} to v_i. If G is directed, then e_i should be oriented from v_{i-1} to v_i.

Minimum-weight Perfect Matching

We leave it to the reader to $\cdots$

It is easy to check that $\cdots$

Shortest Path Algorithms

Luis Goddyn, Math 408

Given an edge weighted graph (G, d), $d : E(G) \to \mathbb{Q}$ and two vertices $s, t \in V(G)$, the *Shortest Path Problem* is to find an s, t-path P whose total weight is as small as possible. Here, G may be either directed or undirected. A path in a graph is a sequence $v_0 e_1, v_1, \ldots, v_k$ of vertices and edges such that no vertex or edge appears twice, and e_i joins v_{i-1} to v_i. If G is directed, then e_i should be oriented from v_{i-1} to v_i.

Minimum-weight Perfect Matching

We leave it to the reader to $\cdots$

It is easy to check that $\cdots$

Why? This will be a homework question.

Shortest Path Algorithms

Luis Goddyn, Math 408

Given an edge weighted graph (G, d), $d : E(G) \to \mathbb{Q}$ and two vertices $s, t \in V(G)$, the *Shortest Path Problem* is to find an s,t-path P whose total weight is as small as possible. Here, G may be either directed or undirected. A path in a graph is a sequence $v_0 e_1, v_1, \ldots, v_k$ of vertices and edges such that no vertex or edge appears twice, and e_i joins v_{i-1} to v_i. If G is directed, then e_i should be oriented from v_{i-1} to v_i.

Minimum-weight Perfect Matching

We leave it to the reader to $\cdots$

It is easy to check that $\cdots$

Why? This will be a homework question.

And Errors.

INTERESTED?

let's talk.

Robert W. Floyd (1936–2001)

For having a clear influence on *methodologies* *for the creation of efficient and reliable software, and for helping to* *found* *the following important subfields of computer science:*

Robert W. Floyd (1936–2001)

For having a clear influence on methodologies for the creation of efficient and reliable software, and for helping to found the following important subfields of computer science:

the theory of parsing, the semantics of programming languages, automatic program verification, automatic program synthesis, and analysis of algorithms

— *Turing Award*, 1978

$$\boxed{d_{ij}^{(k)}:}$$

the weight of a shortest path from i to j

for which all *intermediate* vertices are in $\{1, 2, \cdots, k\}$

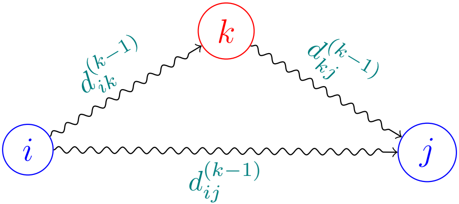
$$\boxed{d_{ij}^{(k)} :}$$

the weight of a shortest path from i to j

for which all *intermediate* vertices are in $\{1, 2, \cdots, k\}$

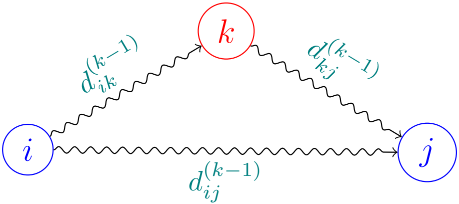
$$\boxed{D^{(n)} \triangleq \left(d_{ij}^{(n)}\right)}$$

$$k \in \mathrm{SP}_{ij}^{(k)}?$$



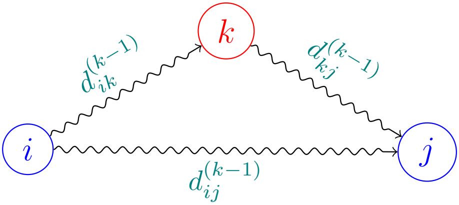
$$d_{ij}^{(k)} = \begin{cases} w_{ij} & k = 0 \\ \min\left\{ d_{ij}^{(k-1)}, \underbrace{d_{ik}^{(k-1)} + d_{kj}^{(k-1)}}_{} \right\} & k \geq 1 \end{cases}$$

$$k \in \mathrm{SP}_{ij}^{(k)}?$$



$$d_{ij}^{(k)} = \begin{cases} w_{ij} & k = 0 \\ \min\left\{ d_{ij}^{(k-1)}, \underbrace{d_{ik}^{(k-1)} + d_{kj}^{(k-1)}}_{\text{why?}} \right\} & k \geq 1 \end{cases}$$

$k \in \mathrm{SP}_{ij}^{(k)}$?



$$d_{ij}^{(k)} = \begin{cases} w_{ij} & k = 0 \\ \min\left\{ d_{ij}^{(k-1)}, \underbrace{d_{ik}^{(k-1)} + d_{kj}^{(k-1)}}_{\text{why?}} \right\} & k \geq 1 \end{cases}$$

$\cdots$, but we assume that there are no negative-weight cycles.

— *Section 25.2 of CLRS*

procedure FLOYD-WARSHALL(W)

  $D^{(0)} = W$

  for $k \leftarrow 1$ **to** n **do**

    $D^{(k)} \triangleq \left(d_{ij}^{(k)}\right) \leftarrow$ a new $n \times n$ matrix

    for $i \leftarrow 1$ **to** n **do**

      for $j \leftarrow 1$ **to** n **do**

        $d_{ij}^{(k)} = \min\left\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right\}$

  return $D^{(n)}$

```
procedure FLOYD-WARSHALL(W)
    D^(0) = W
    for k ← 1 to n do
        D^(k) ≜ (d_ij^(k)) ← a new n × n matrix
        for i ← 1 to n do
            for j ← 1 to n do
                d_ij^(k) = min { d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1) }
    return D^(n)
```

$$\text{Space} : \Theta(n^3)$$

procedure FLOYD-WARSHALL(W)

$D^{(0)} = W$

for $k \leftarrow 1$ **to** n **do**

$D^{(k)} \triangleq \left(d_{ij}^{(k)}\right) \leftarrow$ a new $n \times n$ matrix

for $i \leftarrow 1$ **to** n **do**

for $j \leftarrow 1$ **to** n **do**

$d_{ij}^{(k)} = \min\left\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right\}$

return $D^{(n)}$

$$\text{Space} : \Theta(n^3) \implies \Theta(n^2)$$

FLOYD-WARSHALL Made Simple (Problem 25.2-4)

```
procedure FLOYD-WARSHALL-SIMPLIFIED(W)
    D = W

    for k ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                d_ij = min{d_ij, d_ik + d_kj}

    return D
```

Floyd-Warshall Made Simple (Problem 25.2-4)

```
procedure FLOYD-WARSHALL-SIMPLIFIED(W)
    D = W

    for k ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                d_ij = min{d_ij, d_ik + d_kj}

    return D
```

$d_{ij}^{(k-1)}, d_{ik}^{(k-1)}, d_{kj}^{(k-1)}$ does not change.

Negative-weight Cycle Detection (Problem 25.2-6)

To detect negative-weight cycle (NC) using FLOYD-WARSHALL.

Negative-weight Cycle Detection (Problem 25.2-6)

To detect negative-weight cycle (NC) using FLOYD-WARSHALL.

$$\exists i : d_{ii}^{(n)} < 0$$

Negative-weight Cycle Detection (Problem 25.2-6)

To detect negative-weight cycle (NC) using FLOYD-WARSHALL.

$$\exists i : d_{ii}^{(n)} < 0$$

Proof.

Negative-weight Cycle Detection (Problem 25.2-6)

To detect negative-weight cycle (NC) using FLOYD-WARSHALL.

$$\exists i : d_{ii}^{(n)} < 0$$

Proof.

Negative-weight Cycle Detection (Problem 25.2-6)

To detect negative-weight cycle (NC) using FLOYD-WARSHALL.

$$\exists i : d_{ii}^{(n)} < 0$$

Proof.

Negative-weight Cycle Detection (Problem 25.2-6)

To detect negative-weight cycle (NC) using FLOYD-WARSHALL.

$$\exists i : d_{ii}^{(n)} < 0$$

Proof.

$$\exists i : d_{ii}^{(n)} < 0 \iff \exists \, \mathsf{NC} \subseteq G$$

□

$$\exists i : d_{ii}^{(n)} < 0 \iff \exists \, \mathsf{NC} \subseteq G$$

Proof.

$$\exists i : d_{ii}^{(n)} < 0 \iff \exists\ \mathsf{NC} \subseteq G$$

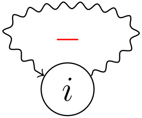
Proof.

“ $\Longrightarrow$ ”

$$\exists i : d_{ii}^{(n)} < 0 \iff \exists\ \mathsf{NC} \subseteq G$$

Proof.

" $\Longrightarrow$ "



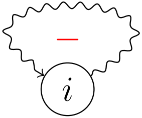
A Negative Cycle

$$\exists i : d_{ii}^{(n)} < 0 \iff \exists\ \mathsf{NC} \subseteq G$$

Proof.

" $\Longleftarrow$ "

" $\Longrightarrow$ "



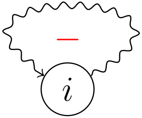
A Negative Cycle

$$\exists i : d_{ii}^{(n)} < 0 \iff \exists\ \mathsf{NC} \subseteq G$$

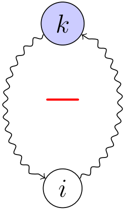
Proof.

“ $\Longleftarrow$ ”

“ $\Longrightarrow$ ”



A Negative Cycle



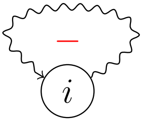
A Simple Negative Cycle

$k : \max \#$

$$\exists i : d_{ii}^{(n)} < 0 \iff \exists\ \mathsf{NC} \subseteq G$$

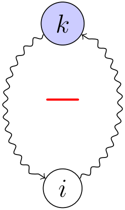
Proof.

“ $\Longrightarrow$ ”



A Negative Cycle

“ $\Longleftarrow$ ”



A Simple Negative Cycle

$k : \max \#$

$d_{ii}^{(k)} < 0$

□

$l_{ij}^{(m)}$: the length of a shortest path from i to j consisting of $\leq m$ edges

$l_{ij}^{(m)}$: the length of a shortest path from i to j consisting of $\leq m$ edges

$$l_{ij}^{(m)} = \min_{1 \leq k \leq n} \{ l_{ik}^{(m-1)} + w_{kj} \}, \quad m \geq 2$$

$l_{ij}^{(m)}$: the length of a shortest path from i to j consisting of $\leq m$ edges

$$l_{ij}^{(m)} = \min_{1 \leq k \leq n} \{l_{ik}^{(m-1)} + w_{kj}\}, \quad m \geq 2$$

$$l_{ij}^{(1)} = w_{ij}$$

$l_{ij}^{(m)}$: the length of a shortest path from i to j consisting of $\leq m$ edges

$$l_{ij}^{(m)} = \min_{1 \leq k \leq n} \{ l_{ik}^{(m-1)} + w_{kj} \}, \quad m \geq 2$$

$$l_{ij}^{(1)} = w_{ij}$$

$$L^{(n-1)} \triangleq \left(l_{ij}^{(n-1)} \right)$$

$l_{ij}^{(m)}$: the length of a shortest path from i to j consisting of $\leq m$ edges

$$l_{ij}^{(m)} = \min_{1 \leq k \leq n} \{ l_{ik}^{(m-1)} + w_{kj} \}, \quad m \geq 2$$

$$l_{ij}^{(1)} = w_{ij}$$

$$L^{(n-1)} \triangleq \left(l_{ij}^{(n-1)} \right)$$

$$L^{(n-1)} = W^{n-1} \triangleq \underbrace{((W \cdot W) \cdot \ldots) \cdot W}_{n-1}$$

Associative EXTEND-SHORTEST-PATHS

Show that matrix multiplication defined by EXTEND-SHORTEST-PATHS is associative.

Associative Extend-Shortest-Paths

Show that matrix multiplication defined by Extend-Shortest-Paths is associative.

$$(W^a W^b) W^c = W^a (W^b W^c)$$

Associative EXTEND-SHORTEST-PATHS

Show that matrix multiplication defined by EXTEND-SHORTEST-PATHS is associative.

$$(W^a W^b) W^c = W^a (W^b W^c)$$

$$(i, j)$$

$$\min_{1 \le k \le n} \Big((W^a W^b)_{ik} + W^c_{kj} \Big) \qquad \min_{1 \le k \le n} \Big(W^a_{ik} + (W^b W^c)_{kj} \Big)$$

$$= \min_{1 \le k \le n} \Big(\min_{1 \le q \le n} (W^a_{iq} + W^b_{qk}) + W^c_{kj} \Big) \quad = \min_{1 \le k \le n} \Big(W^a_{ik} + \min_{1 \le q \le n} (W^b_{kq} + W^c_{qj}) \Big)$$

Associative EXTEND-SHORTEST-PATHS

Show that matrix multiplication defined by EXTEND-SHORTEST-PATHS is associative.

$$(W^a W^b) W^c = W^a (W^b W^c)$$

$$(i, j)$$

$$\min_{1 \le k \le n} \left((W^a W^b)_{ik} + W^c_{kj} \right) \qquad \min_{1 \le k \le n} \left(W^a_{ik} + (W^b W^c)_{kj} \right)$$

$$= \min_{1 \le k \le n} \left(\min_{1 \le q \le n} (W^a_{iq} + W^b_{qk}) + W^c_{kj} \right) \quad = \min_{1 \le k \le n} \left(W^a_{ik} + \min_{1 \le q \le n} (W^b_{kq} + W^c_{qj}) \right)$$

Q : Why do we care about this?

Associative EXTEND-SHORTEST-PATHS

Show that matrix multiplication defined by EXTEND-SHORTEST-PATHS is associative.

$$(W^a W^b) W^c = W^a (W^b W^c)$$

$$(i, j)$$

$$\min_{1 \le k \le n} \Big((W^a W^b)_{ik} + W^c_{kj} \Big) \qquad \min_{1 \le k \le n} \Big(W^a_{ik} + (W^b W^c)_{kj} \Big)$$

$$= \min_{1 \le k \le n} \Big(\min_{1 \le q \le n} (W^a_{iq} + W^b_{qk}) + W^c_{kj} \Big) \quad = \min_{1 \le k \le n} \Big(W^a_{ik} + \min_{1 \le q \le n} (W^b_{kq} + W^c_{qj}) \Big)$$

Q : Why do we care about this?

A : Repeated squaring.

Associative EXTEND-SHORTEST-PATHS

Show that matrix multiplication defined by EXTEND-SHORTEST-PATHS is associative.

$$(W^a W^b)W^c = W^a(W^b W^c) \quad vs. \quad (WW)W = W(WW)$$

$$(i, j)$$

$$\min_{1 \le k \le n} \Big((W^a W^b)_{ik} + W^c_{kj} \Big) \qquad \min_{1 \le k \le n} \Big(W^a_{ik} + (W^b W^c)_{kj} \Big)$$

$$= \min_{1 \le k \le n} \Big(\min_{1 \le q \le n} (W^a_{iq} + W^b_{qk}) + W^c_{kj} \Big) \quad = \min_{1 \le k \le n} \Big(W^a_{ik} + \min_{1 \le q \le n} (W^b_{kq} + W^c_{qj}) \Big)$$

Q : Why do we care about this?

A : Repeated squaring.

SSSP from s

SSSP from s

$l_v^{(m)}$: the length of a shortest path $s \rightsquigarrow v$ consisting of $\leq m$ edges

SSSP from s

$l_v^{(m)}$: the length of a shortest path $s \leadsto v$ consisting of $\le m$ edges

$$l_v^{(m)} = \min_{1 \le u \le n} \left\{ l_u^{(m-1)} + w_{uv} \right\}, \quad m \ge 2$$

SSSP from s

$l_v^{(m)}$: the length of a shortest path $s \rightsquigarrow v$ consisting of $\leq m$ edges

$$l_v^{(m)} = \min_{1 \leq u \leq n} \left\{ l_u^{(m-1)} + w_{uv} \right\}, \quad m \geq 2$$

$$l_v^{(1)} = w_{sv}$$

SSSP from s

$l_v^{(m)}$: the length of a shortest path $s \rightsquigarrow v$ consisting of $\leq m$ edges

$$l_v^{(m)} = \min_{1 \leq u \leq n} \left\{ l_u^{(m-1)} + w_{uv} \right\}, \quad m \geq 2$$

$$l_v^{(1)} = w_{sv}$$

$$L^{(n-1)} \triangleq \left(l_v^{(n-1)} \right)$$

SSSP as a Product of Matrices and a Vector (Problem 25.1-5)

To express SSSP as a product of matrices and a vector.

SSSP as a Product of Matrices and a Vector (Problem 25.1-5)

To express SSSP as a product of matrices and a vector.

$$
\begin{aligned}
L^{(1)} &= \langle w_{sv} \rangle_{v \in V} \\
L^{(2)} &= L^{(1)} \cdot W \\
\cdots &= \cdots \\
L^{(n-1)} &= \left(\left(\left(L^{(1)} \cdot W \right) \cdot W \right) \cdot \ldots \cdot W \right)
\end{aligned}
$$

SSSP as a Product of Matrices and a Vector (Problem 25.1-5)

What is the relationship between it and the Bellman-Ford algorithm?

SSSP as a Product of Matrices and a Vector (Problem 25.1-5)

What is the relationship between it and the Bellman-Ford algorithm?

$$l_v^{(m)} = \min_{1 \le u \le n} \left\{ l_u^{(i-1)} + w_{uv} \right\}, \quad m \ge 2$$

SSSP as a Product of Matrices and a Vector (Problem 25.1-5)

What is the relationship between it and the Bellman-Ford algorithm?

$$l_v^{(m)} = \min_{1 \le u \le n} \left\{ l_u^{(i-1)} + w_{uv} \right\}, \quad m \ge 2$$

$$d_v^{(i)} = \min_{u \to v} \left\{ d_u^{(i-1)} + w_{uv} \right\}, \quad i \ge 1$$

SSSP as a Product of Matrices and a Vector (Problem 25.1-5)

What is the relationship between it and the Bellman-Ford algorithm?

$$l_v^{(m)} = \min_{1 \le u \le n} \left\{ l_u^{(i-1)} + w_{uv} \right\}, \quad m \ge 2$$

$$d_v^{(i)} = \min_{u \to v} \left\{ d_u^{(i-1)} + w_{uv} \right\}, \quad i \ge 1$$

$$d_v^{(0)} = \begin{cases} 0 & v = s \\ \infty & v \ne s \end{cases}$$

procedure Bellman-Ford-DP(G, w, s)

  $d[0, s] \leftarrow 0$

  for $(v \neq s) \in V$ **do**

    $d[0, v] \leftarrow \infty$

  for $i \leftarrow 1$ **to** $|V| - 1$ **do**

    for $v \in V$ **do**

      $d[i, v] = \infty$

      for $(u, v) \in E$ **do**

        if $d[i, v] > d[i-1, u] + w(u, v)$ **then** $\triangleright$

          $d[i, v] = d[i-1, u] + w(u, v)$

procedure BELLMAN-FORD-DP(G, w, s)

    $d[0, s] \leftarrow 0$

    for $(v \neq s) \in V$ **do**

        $d[0, v] \leftarrow \infty$

    for $i \leftarrow 1$ **to** $|V| - 1$ **do**

        for $v \in V$ **do**

            $d[i, v] = \infty$

            for $(u, v) \in E$ **do**

                if $d[i, v] > d[i-1, u] + w(u, v)$ **then** $\triangleright$ Simplify?

                    $d[i, v] = d[i-1, u] + w(u, v)$

```
procedure Bellman-Ford-DP-Simplified(G, w, s)
    d[s] ← 0
    for (v ≠ s) ∈ V do
        d[v] ← ∞

    for i ← 1 to |V| − 1 do
        for v ∈ V do
            for (u, v) ∈ E do
                if d[v] > d[u] + w(u, v) then                ▷
                    d[v] = d[u] + w(u, v)
```

```
procedure BELLMAN-FORD-DP-SIMPLIFIED(G, w, s)
    d[s] ← 0
    for (v ≠ s) ∈ V do
        d[v] ← ∞

    for i ← 1 to |V| − 1 do
        for v ∈ V do
            for (u, v) ∈ E do
                if d[v] > d[u] + w(u, v) then                ▷ Relax!
                    d[v] = d[u] + w(u, v)
```

```
procedure BELLMAN-FORD-DP-SIMPLIFIED(G, w, s)
    d[s] ← 0
    for (v ≠ s) ∈ V do
        d[v] ← ∞

    for i ← 1 to |V| − 1 do
        for v ∈ V do
            for (u, v) ∈ E do
                if d[v] > d[u] + w(u, v) then          ▷ Relax!
                    d[v] = d[u] + w(u, v)
```

```
procedure BELLMAN-FORD-WITHOUT-NE(G, w, s)
    INIT-SINGLE-SOURCE(G, s)

    for i ← 1 to |V| − 1 do
        for (u, v) ∈ E do
            RELAX(u, v, w)
```

Bellman-Ford: $L \cdot W$

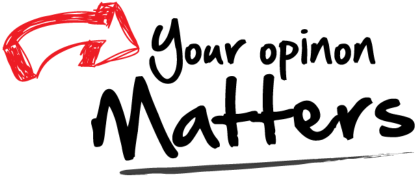
SLOW-ALL-PAIRS-SHORTEST-PATHS: $W \cdot W$

Bellman-Ford: $L \cdot W$

SLOW-ALL-PAIRS-SHORTEST-PATHS: $W \cdot W$

SLOW-ALL-PAIRS-SHORTEST-PATHS is n instances of Bellman-Ford, one for each source.


THANK YOU



Office 302

Mailbox: H016

hfwei@nju.edu.cn