



Design and analysis of algorithms

Lecture 27

Edyta Szymańska

edyta@cc.gatech.edu

All-pairs shortest paths

INPUT: graph $G = (V, E)$, directed,
 $w : E \rightarrow \mathfrak{R}, |V| = n, |E| = m$

All-pairs shortest paths

INPUT: graph $G = (V, E)$, directed,

$w : E \rightarrow \mathfrak{R}, |V| = n, |E| = m$

OUTPUT: a $n \times n$ matrix of shortest-path distances $\delta(u, v)$

All-pairs shortest paths

INPUT: graph $G = (V, E)$, directed,

$w : E \rightarrow \mathfrak{R}$, $|V| = n$, $|E| = m$

OUTPUT: a $n \times n$ matrix of shortest-path distances $\delta(u, v)$

Methods:

All-pairs shortest paths

INPUT: graph $G = (V, E)$, directed,

$w : E \rightarrow \mathfrak{R}$, $|V| = n$, $|E| = m$

OUTPUT: a $n \times n$ matrix of shortest-path distances $\delta(u, v)$

Methods:

- ⑥ Bellman-Ford algorithm - run once for every $v \in V$,
time: $O(n^2m) = O(n^4)$

All-pairs shortest paths

INPUT: graph $G = (V, E)$, directed,

$w : E \rightarrow \mathfrak{R}$, $|V| = n$, $|E| = m$

OUTPUT: a $n \times n$ matrix of shortest-path distances $\delta(u, v)$

Methods:

- ⑥ Bellman-Ford algorithm - run once for every $v \in V$,
time: $O(n^2m) = O(n^4)$
- ⑥ Better: use clever dynamic programming

All-pairs shortest paths

INPUT: graph $G = (V, E)$, directed,

$w : E \rightarrow \mathfrak{R}$, $|V| = n$, $|E| = m$

OUTPUT: a $n \times n$ matrix of shortest-path distances $\delta(u, v)$

Methods:

- ⑥ Bellman-Ford algorithm - run once for every $v \in V$,
time: $O(n^2m) = O(n^4)$
- ⑥ Better: use clever dynamic programming
Assume for now that there are no negative cycles.

All-pairs shortest paths

Graph representation: adjacency matrix $W = (w_{ij})$ of edge weights and $w_{ii} = 0$, for all i .

All-pairs shortest paths

Graph representation: adjacency matrix $W = (w_{ij})$ of edge weights and $w_{ii} = 0$, for all i .

Dynamic programming:

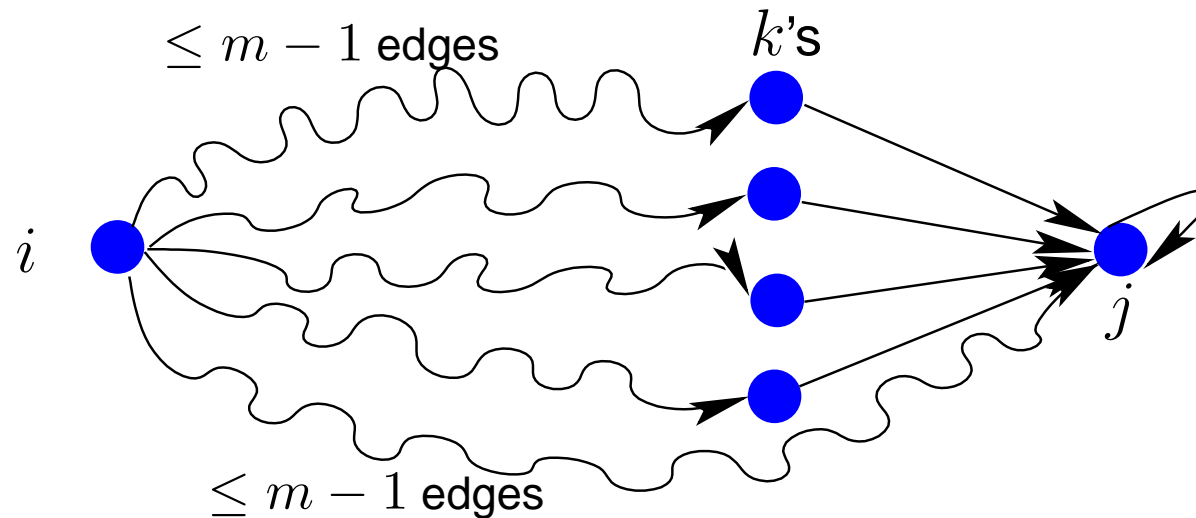
What are the subproblems? - all subpaths of a shortest path are shortest paths

All-pairs shortest paths

Let $d_{ij}^{(m)}$ be the weight of a shortest path from i to j that uses at most m edges

All-pairs shortest paths

Let $d_{ij}^{(m)}$ be the weight of a shortest path from i to j that uses at most m edges



All-pairs shortest paths



All-pairs shortest paths

Recurrence:

$$\begin{cases} d_{ij}^{(0)} &= \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases} \\ d_{ij}^{(m)} &= \min_{1 \leq k \leq n} \{d_{ik}^{(m-1)} + w_{kj}\} \end{cases}$$

All-pairs shortest paths

Answer:

$$\delta(i, j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \dots$$

$d_{ij} \rightarrow \delta(i, j)$ as in Bellman-Ford

if no negative cycles, then every shortest path has at most $n - 1$ edges.

All-pairs shortest paths

Answer:

$$\delta(i, j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \dots$$

$d_{ij} \rightarrow \delta(i, j)$ as in Bellman-Ford

if no negative cycles, then every shortest path has at most $n - 1$ edges.

Time: $n - 1$ passes (extensions of shortest paths), each time computing n^2 entries (d_{ij} 's) in $\Theta(n)$.

Total: $\Theta(n^4)$. No improvement over Bellman-Ford!

Faster dynamic programming solution

Floyd-Warshall algorithm

Faster dynamic programming solution

Floyd-Warshall algorithm

Subproblems:

subpaths of shortest paths but characterize differently!

Faster dynamic programming solution

Floyd-Warshall algorithm

Subproblems:

subpaths of shortest paths but characterize differently!

Let $c_{ij}^{(m)}$ be the weight of a shortest path from i to j with intermediate vertices in $\{1, 2, \dots, m\}$, now $m \leq n$.

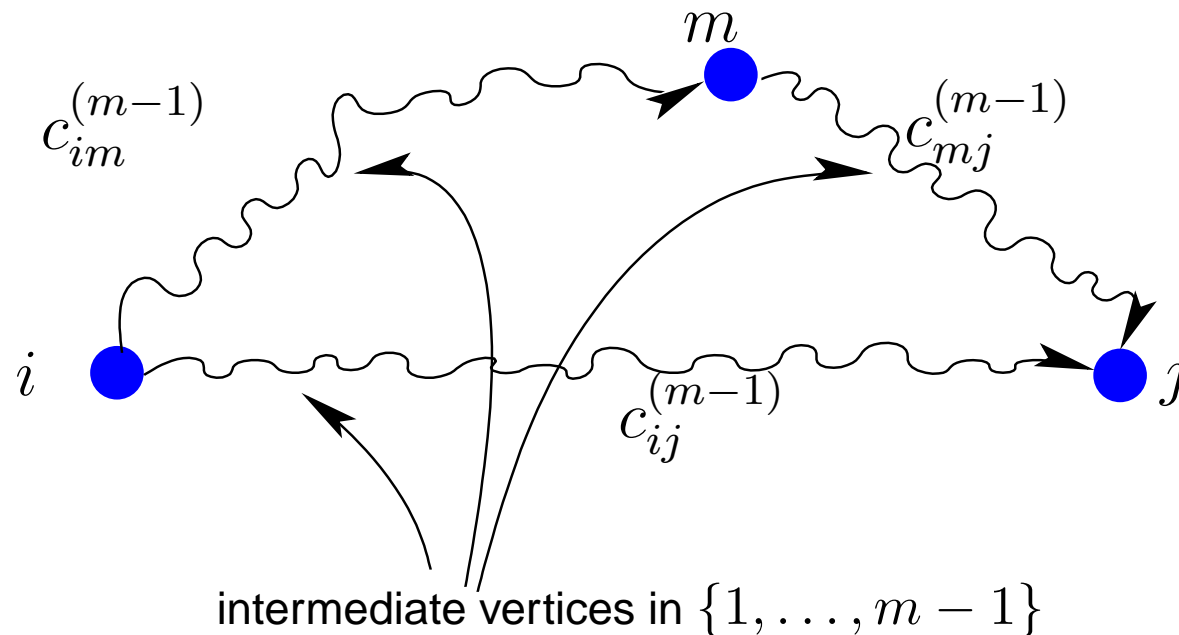
Faster dynamic programming solution

Floyd-Warshall algorithm

Subproblems:

subpaths of shortest paths but characterize differently!

Let $c_{ij}^{(m)}$ be the weight of a shortest path from i to j with intermediate vertices in $\{1, 2, \dots, m\}$, now $m \leq n$.



Floyd-Warshall algorithm

$$\begin{cases} c_{ij}^{(0)} \\ c_{ij}^{(m)} \end{cases} = \begin{cases} w_{ij} \\ \min \left\{ \underbrace{c_{ij}^{(m-1)}}_{\text{doesn't}}, \underbrace{c_{im}^{(m-1)} + c_{mj}^{(m-1)}}_{\text{contains } m} \right\} \end{cases}$$

Floyd-Warshall algorithm

$$\begin{cases} c_{ij}^{(0)} & = w_{ij} \\ c_{ij}^{(m)} & = \min \left\{ \underbrace{c_{ij}^{(m-1)}}_{\text{doesn't}}, \underbrace{c_{im}^{(m-1)} + c_{mj}^{(m-1)}}_{\text{contains } m} \right\} \end{cases}$$

Answer: matrix $C^{(n)} = (c_{ij}^{(n)})$.

Floyd-Warshall algorithm

$$\begin{cases} c_{ij}^{(0)} & = w_{ij} \\ c_{ij}^{(m)} & = \min \left\{ \underbrace{c_{ij}^{(m-1)}}_{\text{doesn't}}, \underbrace{c_{im}^{(m-1)} + c_{mj}^{(m-1)}}_{\text{contains } m} \right\} \end{cases}$$

Answer: matrix $C^{(n)} = (c_{ij}^{(n)})$.

FLOYD_WARSHALL(G, w)

for $m := 1$ to n do

 for $i := 1$ to n do

 for $j := 1$ to n do

 if $c_{ij} > c_{im} + c_{mj}$

 then $c_{ij} := c_{im} + c_{mj}$

Floyd-Warshall algorithm

$$\begin{cases} c_{ij}^{(0)} & = w_{ij} \\ c_{ij}^{(m)} & = \min \left\{ \underbrace{c_{ij}^{(m-1)}}_{\text{doesn't}}, \underbrace{c_{im}^{(m-1)} + c_{mj}^{(m-1)}}_{\text{contains } m} \right\} \end{cases}$$

Answer: matrix $C^{(n)} = (c_{ij}^{(n)})$.

FLOYD_WARSHALL(G, w)

for $m := 1$ to n do

 for $i := 1$ to n do

 for $j := 1$ to n do

 if $c_{ij} > c_{im} + c_{mj}$

 then $c_{ij} := c_{im} + c_{mj}$

Time: $\Theta(n^3)$

Floyd-Warshall algorithm

$$\begin{cases} c_{ij}^{(0)} & = w_{ij} \\ c_{ij}^{(m)} & = \min \left\{ \underbrace{c_{ij}^{(m-1)}}_{\text{doesn't}}, \underbrace{c_{im}^{(m-1)} + c_{mj}^{(m-1)}}_{\text{contains } m} \right\} \end{cases}$$

Answer: matrix $C^{(n)} = (c_{ij}^{(n)})$.

FLOYD_WARSHALL(G, w)

for $m := 1$ to n do

 for $i := 1$ to n do

 for $j := 1$ to n do

 if $c_{ij} > c_{im} + c_{mj}$

 then $c_{ij} := c_{im} + c_{mj}$

Time: $\Theta(n^3)$

Best to date is $O(n^2 \lg n + nm)$.

Transitive closure

Transitive closure $G^* = (V, E')$ of $G = (V, E)$ is a graph with edges $(i, j) \in E'$ if and only if there is a path $i \rightsquigarrow j$ in G

Transitive closure

Transitive closure $G^* = (V, E')$ of $G = (V, E)$ is a graph with edges $(i, j) \in E'$ if and only if there is a path $i \rightsquigarrow j$ in G

Solution:

Transitive closure

Transitive closure $G^* = (V, E')$ of $G = (V, E)$ is a graph with edges $(i, j) \in E'$ if and only if there is a path $i \rightsquigarrow j$ in G

Solution:

- ⑥ adjacency matrix (elements in $\{0, 1\}$, no weights needed)
- ⑥ FLOYD_WARSHALL algorithm, replacing
 $\min \rightarrow \text{BooleanOR}$
 $+$ $\rightarrow \text{BooleanAND}$

$$t_{ij}^{(k)} := t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$$

- ⑥ Time: $O(n^3)$

Transitive closure

Transitive closure $G^* = (V, E')$ of $G = (V, E)$ is a graph with edges $(i, j) \in E'$ if and only if there is a path $i \rightsquigarrow j$ in G

Solution:

- ⑥ adjacency matrix (elements in $\{0, 1\}$, no weights needed)
- ⑥ FLOYD_WARSHALL algorithm, replacing
 $\min \rightarrow \text{BooleanOR}$
 $+$ $\rightarrow \text{BooleanAND}$

$$t_{ij}^{(k)} := t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$$

- ⑥ Time: $O(n^3)$

Transitive closure

Transitive closure $G^* = (V, E')$ of $G = (V, E)$ is a graph with edges $(i, j) \in E'$ if and only if there is a path $i \rightsquigarrow j$ in G

Solution:

- ⑥ adjacency matrix (elements in $\{0, 1\}$, no weights needed)
- ⑥ FLOYD_WARSHALL algorithm, replacing
 $\min \rightarrow \text{BooleanOR}$
 $+$ $\rightarrow \text{BooleanAND}$

$$t_{ij}^{(k)} := t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$$

- ⑥ Time: $O(n^3)$

Dynamic programming - summary

- ⑥ brute force algorithm is exponential
- ⑥ optimal substructure property (recursive solution)
- ⑥ overlapping subproblems (memoize)
- ⑥ constructing the optimal solution

Dynamic programming - summary

- ⑥ brute force algorithm is exponential
- ⑥ optimal substructure property (recursive solution)
- ⑥ overlapping subproblems (memoize)
- ⑥ constructing the optimal solution