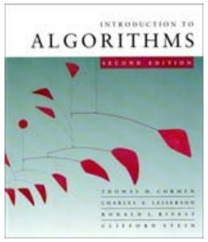
CS60020: Foundations of Algorithm Design and Machine Learning

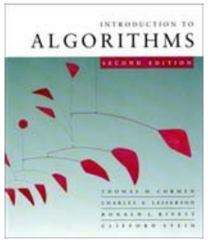
Sourangshu Bhattacharya



Shortest paths

Single-source shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm: $O(E + V \lg V)$
- General
  - Bellman-Ford algorithm: $O(VE)$
- DAG
  - One pass of Bellman-Ford: $O(V + E)$



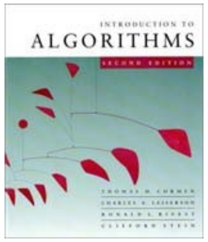
Shortest paths

Single-source shortest paths

- Nonnegative edge weights
  - Dijkstra’s algorithm: $O(E + V \lg V)$
- General
  - Bellman-Ford: $O(VE)$
- DAG
  - One pass of Bellman-Ford: $O(V + E)$

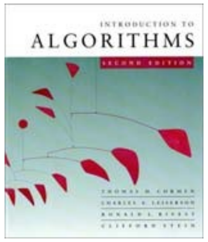
All-pairs shortest paths

- Nonnegative edge weights
  - Dijkstra’s algorithm $|V|$ times: $O(VE + V^2 \lg V)$
- General
  - Three algorithms today.



All-pairs shortest paths

Input: Digraph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, with edge-weight function $w : E \to \mathbb{R}$.
Output: $n \times n$ matrix of shortest-path lengths $\delta(i, j)$ for all $i, j \in V$.



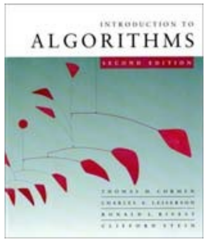
All-pairs shortest paths

Input: Digraph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, with edge-weight function $w : E \to \mathbb{R}$.
Output: $n \times n$ matrix of shortest-path lengths $\delta(i, j)$ for all $i, j \in V$.

IDEA:

- Run Bellman-Ford once from each vertex.
- Time $= O(V^2E)$.
- Dense graph (n^2 edges) $\Rightarrow \Theta(n^4)$ time in the worst case.

Good first try!



Dynamic programming

Consider the $n \times n$ adjacency matrix $A = (a_{ij})$ of the digraph, and define

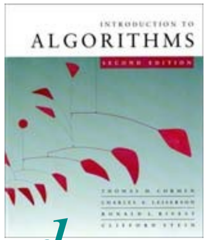
$d_{ij}^{(m)}$ = weight of a shortest path from i to j that uses at most m edges.

Claim: We have

$$d_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j, \\ \infty & \text{if } i \neq j; \end{cases}$$

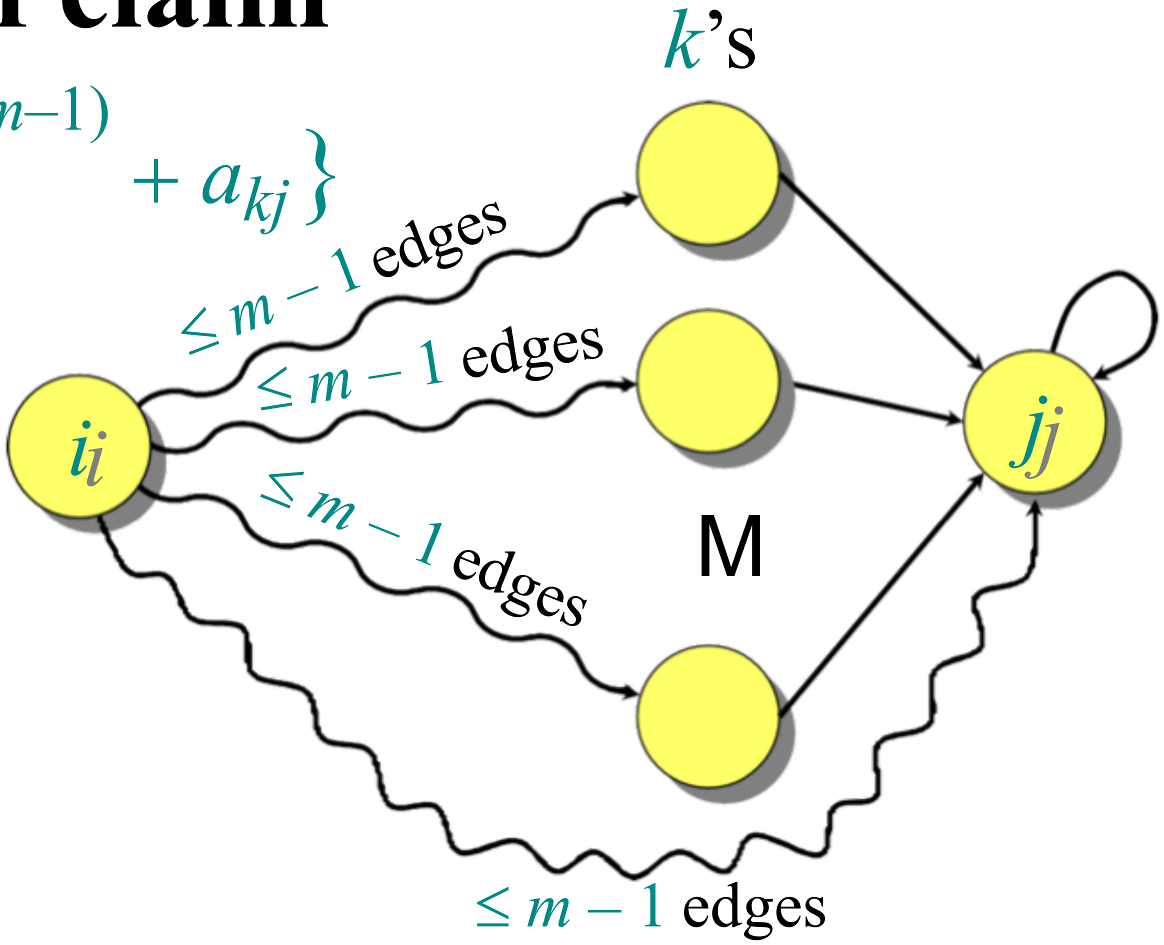
and for $m = 1, 2, \ldots, n - 1$,

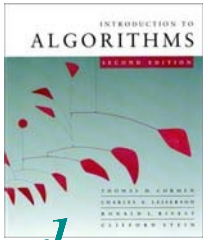
$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \} .$$



Proof of claim

$$d_{ij}^{(m)} = \min_k\{d_{ik}^{(m-1)} + a_{kj}\}$$



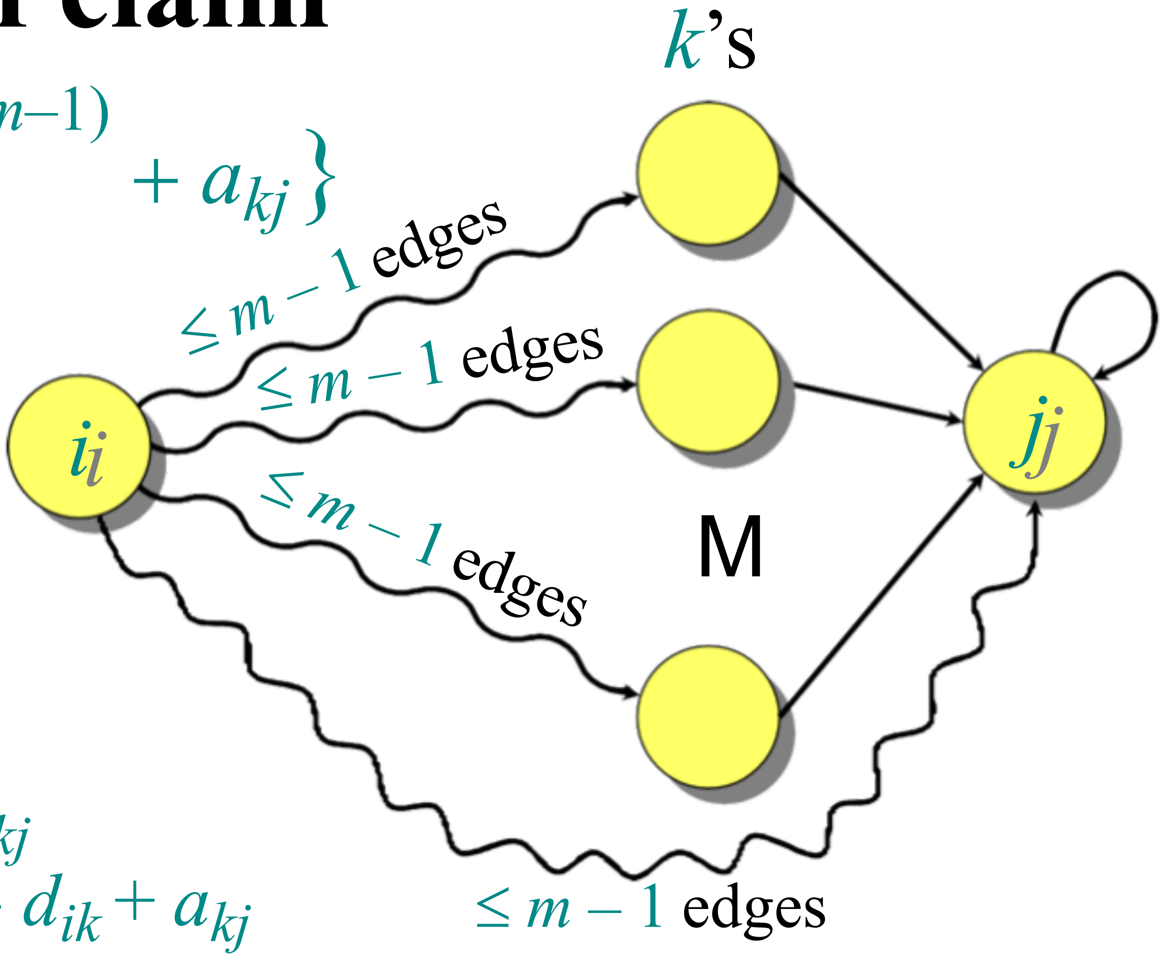


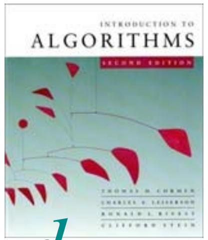
Proof of claim

$$d_{ij}^{(m)} = \min_k\{d_{ik}^{(m-1)} + a_{kj}\}$$

Relaxation!

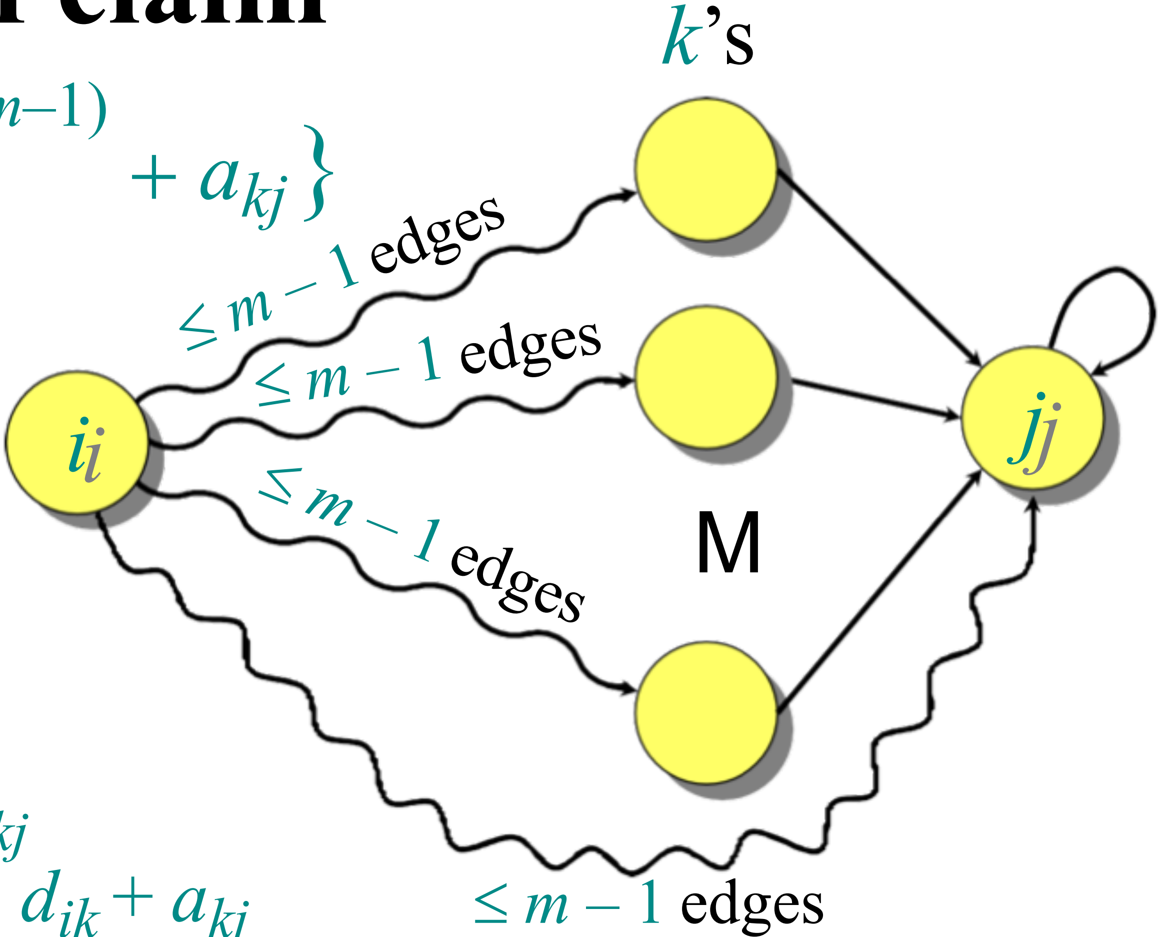
```
for $k \leftarrow 1$ to $n$
    do if $d_{ij} > d_{ik} + a_{kj}$
        then $d_{ij} \leftarrow d_{ik} + a_{kj}$
```





Proof of claim

$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}$

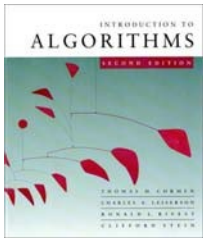


Relaxation!

```
for $k \leftarrow 1$ to $n$
    do if $d_{ij} > d_{ik} + a_{kj}$
        then $d_{ij} \leftarrow d_{ik} + a_{kj}$
```

Note: No negative-weight cycles implies

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

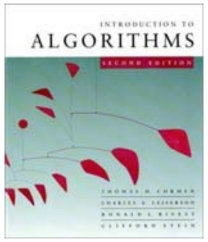


Matrix multiplication

Compute $C = A \cdot B$, where C, A, and B are $n \times n$ matrices:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \,.$$

Time $= \Theta(n^3)$ using the standard algorithm.



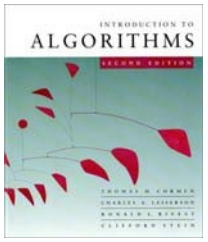
Matrix multiplication

Compute $C = A \cdot B$, where C, A, and B are $n \times n$ matrices:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \, .$$

Time $= \Theta(n^3)$ using the standard algorithm.

What if we map “$+$” $\to$ “min” and “$\cdot$” $\to$ “$+$”?



Matrix multiplication

Compute $C = A \cdot B$, where C, A, and B are $n \times n$ matrices:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} .$$

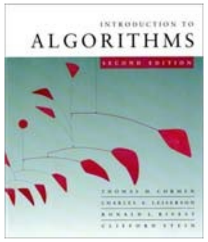
Time $= \Theta(n^3)$ using the standard algorithm.

What if we map "$+$" $\to$ "min" and "$\cdot$" $\to$ "$+$"?

$$c_{ij} = \min_k \{a_{ik} + b_{kj}\} .$$

Thus, $D^{(m)} = D^{(m-1)}$ "$\times$" A.

Identity matrix $= I = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix} = D^0 = (d_{ij}^{(0)})$.



Matrix multiplication (continued)

The (min, +) multiplication is ***associative***, and with the real numbers, it forms an algebraic structure called a ***closed semiring***.

Consequently, we can compute

$$
\begin{aligned}
D^{(1)} &= D^{(0)} \cdot A = A^1 \\
D^{(2)} &= D^{(1)} \cdot A = A^2 \\
&\vdots \\
D^{(n-1)} &= D^{(n-2)} \cdot A = A^{n-1},
\end{aligned}
$$

yielding $D^{(n-1)} = (\delta(i, j))$.

Time $= \Theta(n \cdot n^3) = \Theta(n^4)$. No better than $n \times$ B-F.

Improved matrix multiplication algorithm

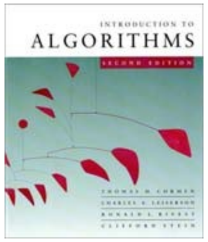
Repeated squaring: $A^{2k} = A^k \times A^k$.

Compute $\underbrace{A^2, A^4, \ldots, A^{2^{\lceil \lg(n-1) \rceil}}}_{O(\lg n)\text{ squarings}}$.

Note: $A^{n-1} = A^n = A^{n+1} = \mathrm{L}$.

Time $= \Theta(n^3 \lg n)$.

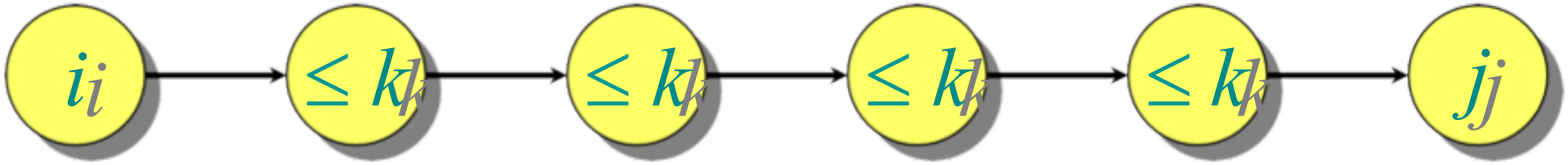
To detect negative-weight cycles, check the diagonal for negative values in $O(n)$ additional time.



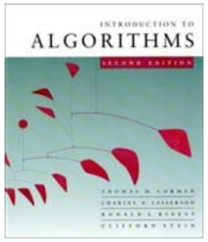
Floyd-Warshall algorithm

Also dynamic programming, but faster!

Define $c_{ij}^{(k)} =$ weight of a shortest path from i to j with intermediate vertices belonging to the set $\{1, 2, \ldots, k\}$.

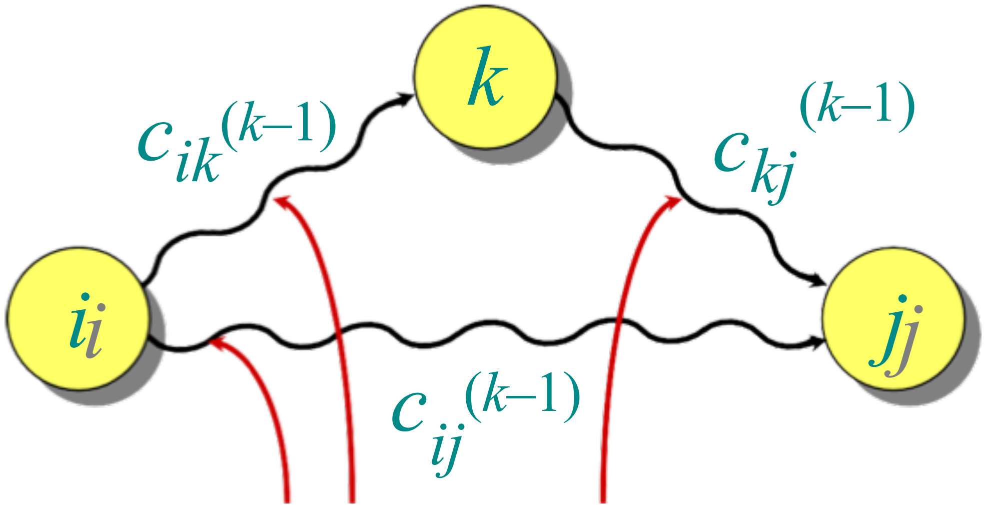


Thus, $\delta(i, j) = c_{ij}^{(n)}$. Also, $c_{ij}^{(0)} = a_{ij}$.

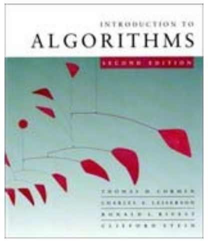


Floyd-Warshall recurrence

$$c_{ij}^{(k)} = \min_k \{c_{ij}^{(k-1)}, c_{ik}^{(k-1)} + c_{kj}^{(k-1)}\}$$



intermediate vertices in $\{1, 2, \ldots, k\}$

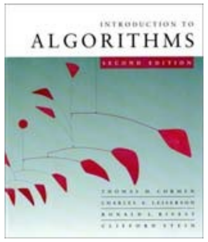


Pseudocode for Floyd-Warshall

```
for $k \leftarrow 1$ to $n$
    do for $i \leftarrow 1$ to $n$
        do for $j \leftarrow 1$ to $n$
            do if $c_{ij} > c_{ik} + c_{kj}$          }
                then $c_{ij} \leftarrow c_{ik} + c_{kj}$  } relaxation
```

Notes:

- Okay to omit superscripts, since extra relaxations can't hurt.
- Runs in $\Theta(n^3)$ time.
- Simple to code.
- Efficient in practice.



Transitive closure of a directed graph

Compute $t_{ij} = \begin{cases} 1 & \text{if there exists a path from } i \text{ to } j, \\ 0 & \text{otherwise.} \end{cases}$

IDEA: Use Floyd-Warshall, but with $(\vee, \wedge)$ instead of $(\min, +)$:

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

Time $= \Theta(n^3)$.