CMSC 451 - Algorithm Design
Lecture 11 - All-pairs Shortest Paths + Floyd-Warshall

Earlier, we covered the single-source shortest path problem for edge-weighted digraphs.

Dijkstra - $O(m + n \log n)$ $n = |V|$ $m = |E|$

Bellman-Ford - $O(n \cdot m)$ [allows neg. edge weights]

Notation:

- The cost of a path is sum of edge weights
- The distance between two vertices u & v is min. cost of any path from u to v.

All-pairs shortest paths:

Given digraph $G = (V, E)$. Each edge $(u,v) \in E$ has weight $w(u,v)$, may be negative (but no neg. cost cycles), compute distance between all pairs of vertices $u, v \in V$.

We'll present Floyd-Warshall algorithm.
DP-based, $O(n^3)$ time.

Notes:

- Discovered independently by Robert Floyd + Stephen Warshall (mid 1960's).
- First discovered (but not credited) by Bernard Roy.
- General DP structure works for many reachability problems:
  - Transitive closure of a binay relation
  - Convert finite state automaton to regular expression.
  - Compute max capacity paths

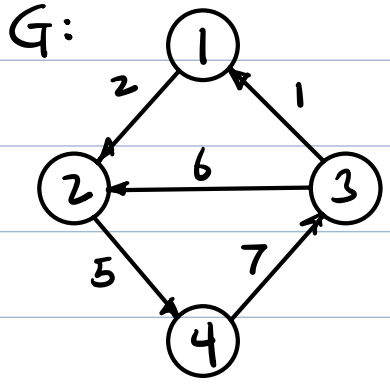
Input/Output Representation:

Input: (Augmented) adjacency matrix:

$V = \{1, 2, \ldots, n\}$

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ w(i,j) & \text{if } i \neq j, (i,j) \in E \\ +\infty & \text{if } i \neq j, (i,j) \notin E \end{cases}$$

G:



w:

	1	2	3	4
1	0	2	∞	∞
2	∞	0	∞	5
3	1	6	0	∞
4	∞	∞	7	0

Output: Distance matrix

d_{ij} = distance from i to j
(∞ if no path)

d:

	1	2	3	4
1	0	2	14	7
2	13	0	12	5
3	1	3	0	8
4	8	10	7	0

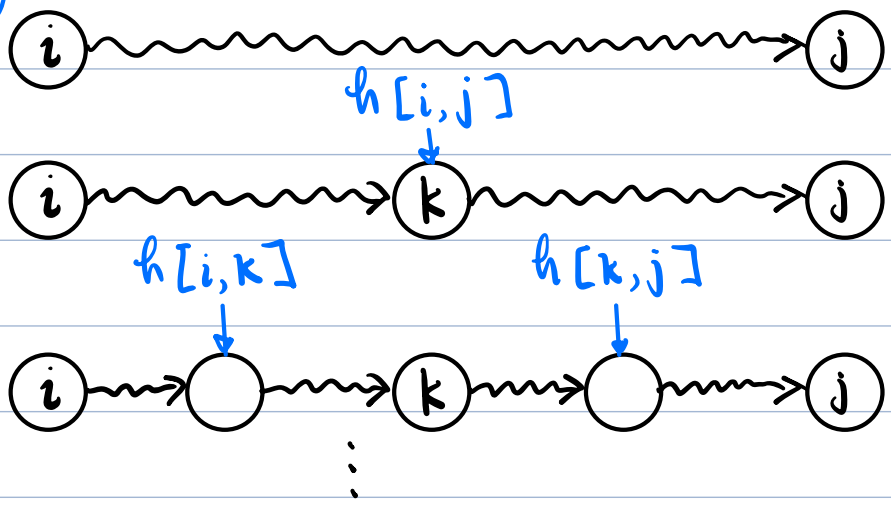
→ Space = $O(n^2)$

Paths? $O(n^2)$ paths ⇒ $O(n^3)$ storage?

Clever trick - Reduces storage to $O(n^2)$
- "Hook" matrix $H = h[1..n, 1..n]$
- For each $i, j \in V$

$$h[i,j] = \begin{cases} \emptyset & \text{if shortest path is direct (edge } (i,j)) \\ k & k = \text{any vertex along the shortest path} \end{cases}$$

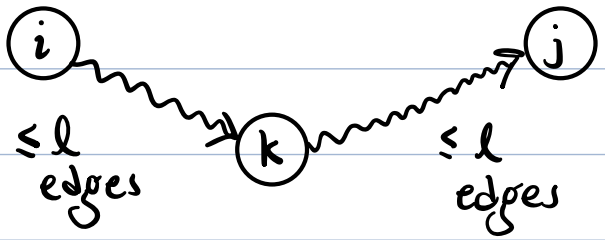
Fill in the path recursively



Floyd-Warshall Algorithm:

- How to reduce shortest paths to smaller subproblems?
- Obvious? (But not best!)
  - Restrict path length
    e.g. $d_{ij}^{(\ell)}$ = shortest path from i to j using $\leq \ell$ edges
  - Build by doubling:

$$d_{ij}^{(2\ell)} = \min_k \left(d_{ik}^{(\ell)} + d_{kj}^{(\ell)} \right)$$



  - This leads to a slower algorithm.

- Floyd-Warshall insight:
  - Don't restrict length, restrict which vertices you can go through.

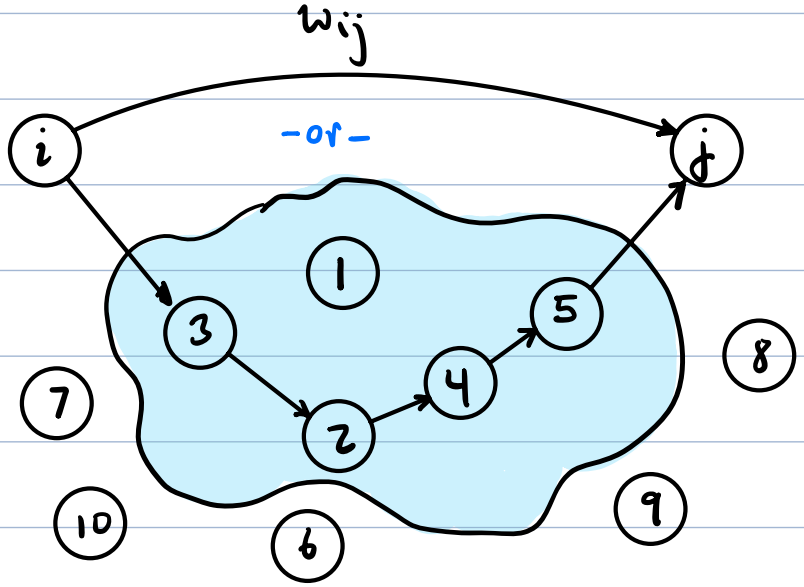
  - Given a path $\langle v_1, v_2, \ldots, v_{\ell-1}, v_\ell \rangle$
    $v_2 \ldots v_{\ell-1}$ are the intermediate vertices.

- for $1 \le i,j \le n$ & $0 \le k \le n$, define:

$d_{ij}^{(k)}$ = cost of shortest path from i to j with intermediate vertices from $\{1,\dots,k\}$.

Eg. $k=5$

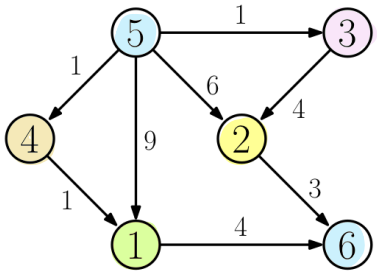
$d_{ij}^{(5)}$:



Approach - $d_{ij}^{(0)} = w_{ij}$ (no intermediates $\Rightarrow$ edge)

for $k = 1$ to n

compute $d_{ij}^{(k)}$ for all $1 \le i,j \le n$.



$d_{5,6}^{(0)} = \infty$ (no path)

$d_{5,6}^{(1)} = 13 \ \langle 5,1,6 \rangle$

$d_{5,6}^{(2)} = 9 \ \ \langle 5,2,6 \rangle$

$d_{5,6}^{(3)} = 8 \ \ \langle 5,3,2,6 \rangle$

$d_{5,6}^{(4)} = 6 \ \ \langle 5,4,1,6 \rangle$

$d_{5,6}^{(5)} = d_{5,6}^{(6)} = 6$ (no change)

DP Formulation:

Basis: $d_{ij}^{(0)} = w_{ij}$ (no intermediates ⇒ edge weight)

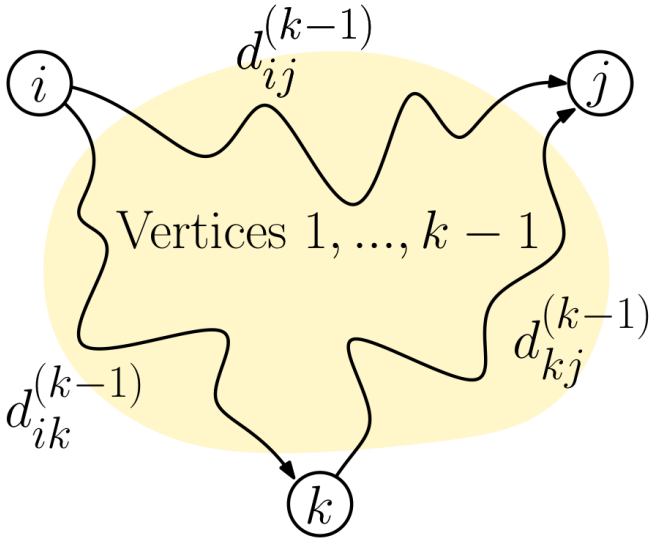
$k \geq 1$: Cases:

- Shortest path does not pass through k:
  - ⇒ only intermediates are $\{1, \ldots, k-1\}$
  - ⇒ $d_{ij}^{(k)} \leftarrow d_{ij}^{(k-1)}$

- Shortest path passes through k:
  - Since no neg. cost cycles, visits k once
  - Path goes $i \rightsquigarrow k$ then $k \rightsquigarrow j$
  - These paths use intermediates $\{1, \ldots, k-1\}$
  - ⇒ $d_{ij}^{(k)} \leftarrow d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

Principle of optimality holds

k may be equal to i or j. Check that this is still correct



DP Credo - Try all options + take best

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

Bottom-Up Implementation

- Build matrix $d[1..n, 1..n]$
- Matrix of "hooks" to save decisions

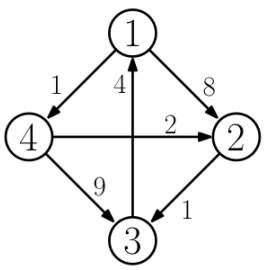
```
floyd-warshall (w[1..n, 1..n])
    for (1 ≤ i,j ≤ n)                    // init d^(0)
        d[i,j] ← w[i,j]
        h[i,j] ← ∅
    for (k = 1 to n)                     // compute d^(k)
        for (1 ≤ i,j ≤ n)
            newCost ← d[i,k] + d[k,j]
            if (newCost < d[i,j])        // better to go
                d[i,j] ← newCost         // through k
                h[i,j] ← k               // update hook
    return d + h matrices                // return dists + hooks
```

Running time: $O(n^3)$

Huh?

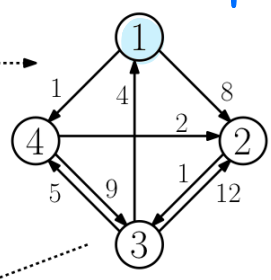
- DP formulation had three parameters: i, j, k but algorithm does not use k.
- We only use k + k-1, so we really only need two copies of d_{ij} matrix
  - current + previous
- In fact, we can show that only one matrix suffices (see pdf notes for details)

Example:



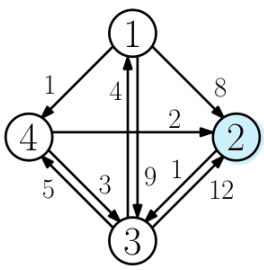
$$d^{(0)} = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix}$$

(w_{ij})

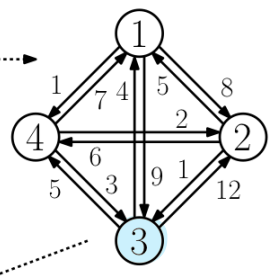


$$d^{(1)} = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix}$$

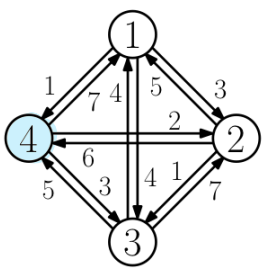
(updates, hooks)



$$d^{(2)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{bmatrix}$$



$$d^{(3)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$$



$$d^{(4)} = \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$$

$$d = \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \qquad H = \begin{bmatrix} \emptyset & 4 & 4 & \emptyset \\ 3 & \emptyset & \emptyset & 3 \\ \emptyset & 4 & \emptyset & 1 \\ 3 & \emptyset & 2 & \emptyset \end{bmatrix}$$

Extracting the Shortest Path:

- As mentioned above, we use $h[i,j]$ to recursively extract path
- If $d[i,j] = \infty \Rightarrow$ no path
  otherwise get-path(i,j) yields edges on shortest path.

```
get-path(i, j)      // path from i to j
    if (h[i,j] = ∅)     // direct path
        return (i,j)    // path is single edge
    else
        mid ← h[i,j]    // i ⇝ mid ⇝ j
        return get-path(i, mid) ⊕ get-path(mid, j)
                          ↑ concatenate
```

Summary:

- Floyd-Warshall all-pairs shortest paths
- $O(n^3)$ time / $O(n^2)$ space
- General structure can be applied to many similar path-related problems.