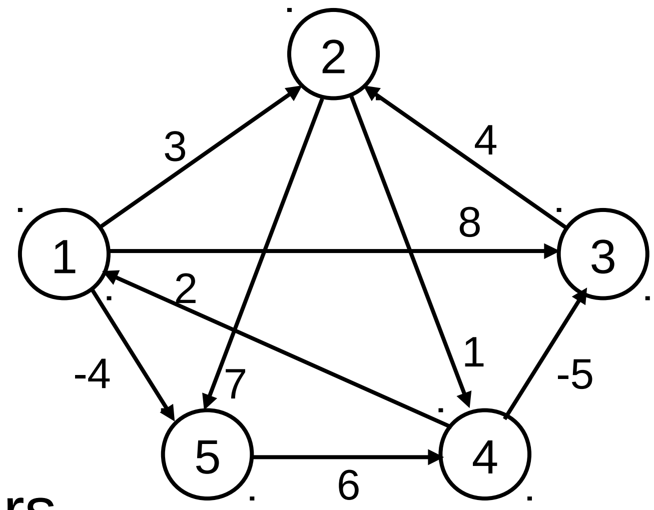
Design & Analysis of Algorithms
CSE 304

All Pairs of Shortest Path

All-Pairs Shortest Paths

- **Given:**
  - Directed graph G = (V, E)
  - Weight function w : E → **R**
- **Compute:**
  - The shortest paths between all pairs of vertices in a graph
  - Representation of the result: an n × n matrix of shortest-path distances δ(u, v)



Dijkstra (G, w, s)

```
INITIALIZE-SINGLE-SOURCE(V, s)  ←  Θ(V)
S ← ∅
Q ← V[G]  ←  O(V) build min-heap
while Q ≠ ∅  ←  Executed O(V) times
    do u ← EXTRACT-MIN(Q)  ←  O(lgV)
        S ← S ∪ {u}
        for each vertex v ∈ Adj[u]
            do RELAX(u, v, w)  ←  O(E) times; O(lgV)
```

Running time: $O(V\lg V + E\lg V) = O(E\lg V)$

BELLMAN-FORD(V, E, w, s)

```
INITIALIZE-SINGLE-SOURCE(V, s)            ← Θ(V)
for i ← 1 to |V| - 1                      ← O(V)
    do for each edge (u, v) ∈ E           ← O(E)      O(VE)
           do RELAX(u, v, w)
for each edge (u, v) ∈ E                  ← O(E)
    do if d[v] > d[u] + w(u, v)
          then return FALSE
return TRUE
```

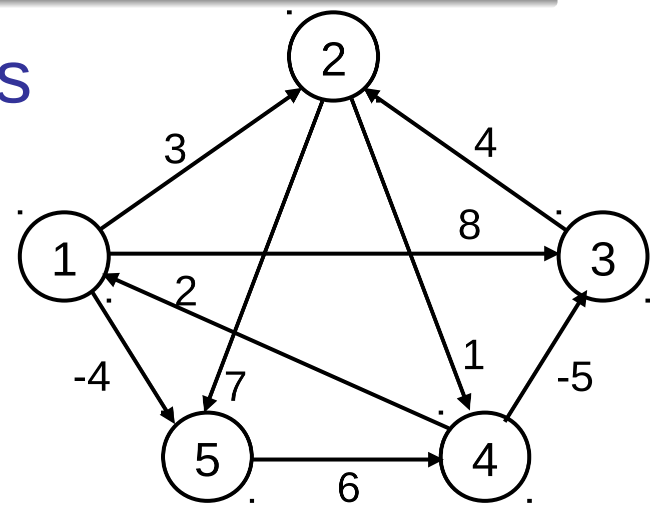
Running time: O(VE)

All-Pairs Shortest Paths - Solutions

- Run **BELLMAN-FORD** once from each vertex:
  - $O(V^2E)$, which is $O(V^4)$ if the graph is dense ($E = \Theta(V^2)$)
- If no negative-weight edges, could run **Dijkstra's** algorithm once from each vertex:
  - $O(VE\lg V)$ with binary heap, $O(V^3\lg V)$ if the graph is dense
- We can solve the problem in $O(V^3)$, with no elaborate data structures

All-Pairs Shortest Paths

- Assume the graph (G) is given as adjacency matrix of weights
  - $W = (w_{ij})$, n x n matrix, $|V| = n$
  - Vertices numbered 1 to n



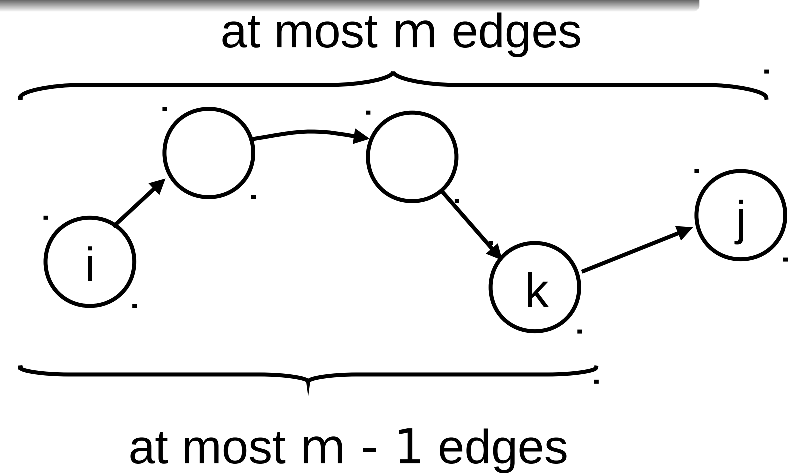
$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ \text{weight of } (i, j) & \text{if } i \neq j,\ (i, j) \in E \\ \infty & \text{if } i \neq j,\ (i, j) \notin E \end{cases}$$

- Output the result in an n x n matrix

  $D = (d_{ij})$, where $d_{ij} = \delta(i, j)$
- Solve the problem using dynamic programming

Optimal Substructure of a Shortest Path

- All subpaths of a shortest path are shortest paths
- Let p: a shortest path p from vertex i to j that contains at most m edges
- If $i = j$
  - $w(p) = 0$ and p has no edges

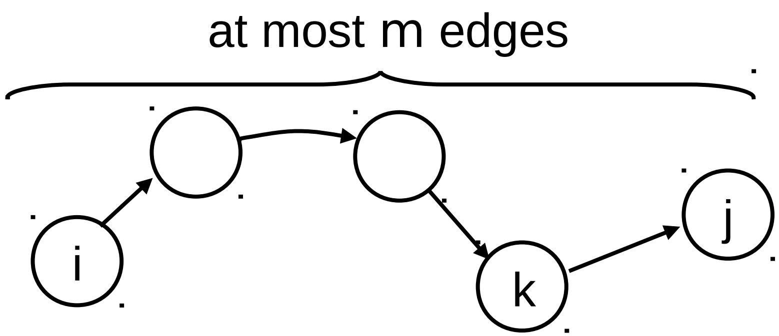


- If $i \neq j$: $p = i \overset{p'}{\leadsto} k \rightarrow j$
  - p’ has at most m-1 edges
  - p’ is a shortest path

$$\delta(i, j) = \delta(i, k) + w_{kj}$$

Recursive Solution

- $l_{ij}^{(m)}$ = weight of shortest path $i \rightsquigarrow j$ that contains at most m edges



- $m = 0$: $l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$

- $m \geq 1$: $l_{ij}^{(m)} = \min \left\{ l_{ij}^{(m-1)}, \min\limits_{1 \leq k \leq n} \{ l_{ik}^{(m-1)} + w_{kj} \} \right\}$

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

  - Shortest path from i to j with at most m – 1 edges
  - Shortest path from i to j containing at most m edges, considering all possible predecessors (k) of j

Computing the Shortest Paths

- $m = 1$: $l_{ij}^{(1)} = w_{ij}$ $\quad L^{(1)} = W$
  - The path between i and j is restricted to 1 edge
- Given $W = (w_{ij})$, compute: $L^{(1)}, L^{(2)}, \ldots, L^{(n-1)}$, where

$$L^{(m)} = (l_{ij}^{(m)})$$

- $L^{(n-1)}$ contains the actual shortest-path weights

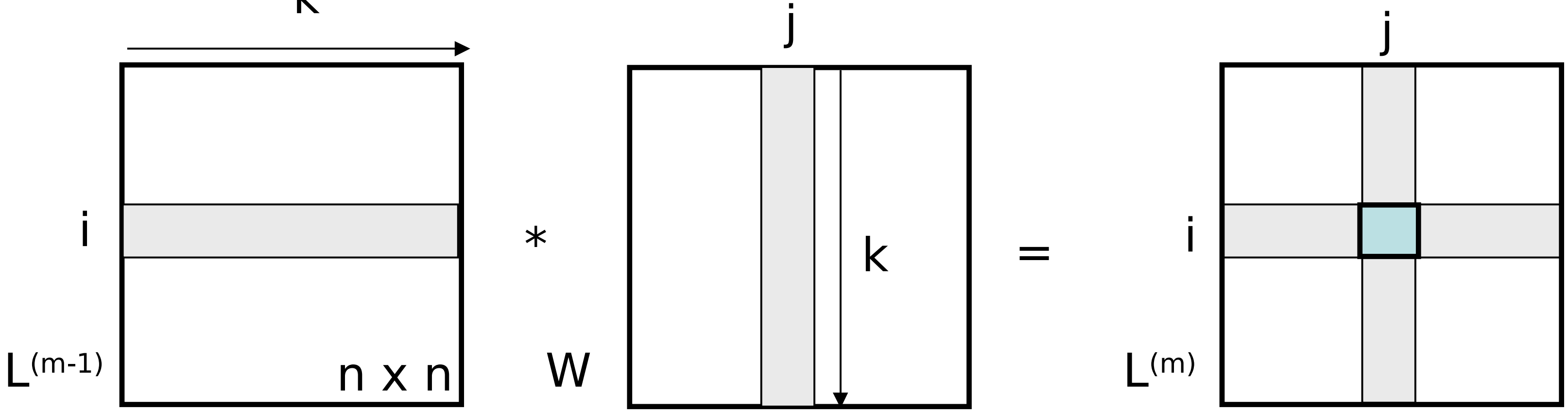
Given $L^{(m-1)}$ and $W \Rightarrow$ compute $L^{(m)}$

  - Extend the shortest paths computed so far by one more edge
- If the graph has no negative cycles: all simple shortest paths contain at most n - 1 edges

$$\delta(i, j) = l_{ij}^{(n-1)} \text{ and } l_{ij}^{(n)} = l_{ij}^{(n+1)} \ldots = l_{ij}^{(n-1)}$$

Extending the Shortest Path

$$l_{ij}^{(m)} = \min_{1 \le k \le n} \{l_{ik}^{(m-1)} + w_{kj}\}$$



Replace: min $\rightarrow$ +
+ $\rightarrow$ •

Computing $L^{(m)}$ looks like matrix multiplication

EXTEND(L, W, n)

```
create L', an n × n matrix
for i ← 1 to n
    do for j ← 1 to n
        do l_ij' ← ∞
            for k ← 1 to n
                do l_ij' ← min(l_ij', l_ik + w_kj)
return L'
```

$$l_{ij}^{(m)} = \min_{1 \le k \le n} \{l_{ik}^{(m-1)} + w_{kj}\}$$

Running time: $\Theta(n^3)$

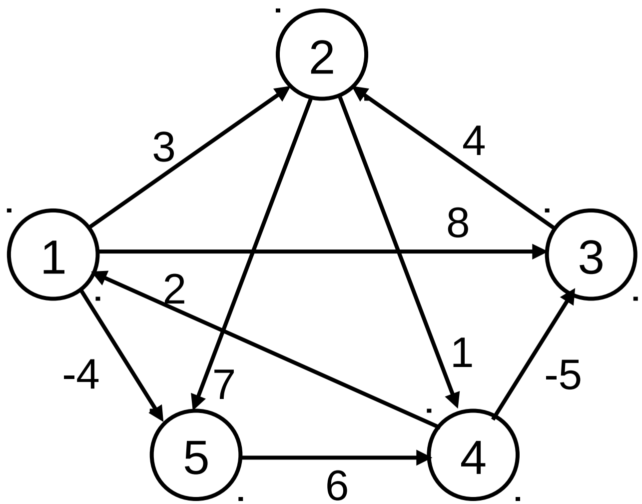
SLOW-ALL-PAIRS-SHORTEST-PATHS(W, n)

1. $L^{(1)} \leftarrow W$
2. **for** $m \leftarrow 2$ **to** $n - 1$
3. **do** $L^{(m)} \leftarrow$ EXTEND $(L^{(m-1)}, W, n)$
4. **return** $L^{(n-1)}$

Running time: $\Theta(n^4)$

Example

$$l_{ij}^{(m)} = \min_{1 \le k \le n} \{ l_{ik}^{(m-1)} + w_{kj} \}$$



$L^{(m-1)} = L^{(1)}$

0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	∞	-5	0	∞
∞	∞	∞	6	0

W

0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	∞	-5	0	∞
∞	∞	∞	6	0

$L^{(m)} = L^{(2)}$

0	3	8	2	-4
3	0	-4	1	7
∞	4	0	5	11
2	-1	-5	0	-2
8	∞	1	6	0

… and so on until $L^{(4)}$

Improving Running Time

- No need to compute all $L^{(m)}$ matrices
- If no negative-weight cycles exist:

  $L^{(m)} = L^{(n-1)}$ for all $m \geq n - 1$
- We can compute $L^{(n-1)}$ by computing the sequence:

$L^{(1)} = W$ $\qquad$ $L^{(2)} = W^2 = W \bullet W$

$L^{(4)} = W^4 = W^2 \bullet W^2$ $\qquad$ $L^{(8)} = W^8 = W^4 \bullet W^4$ …

$$\Rightarrow 2^x = n - 1$$

$$L^{(n-1)} = W^{2^{\lceil \lg(n-1) \rceil}}$$

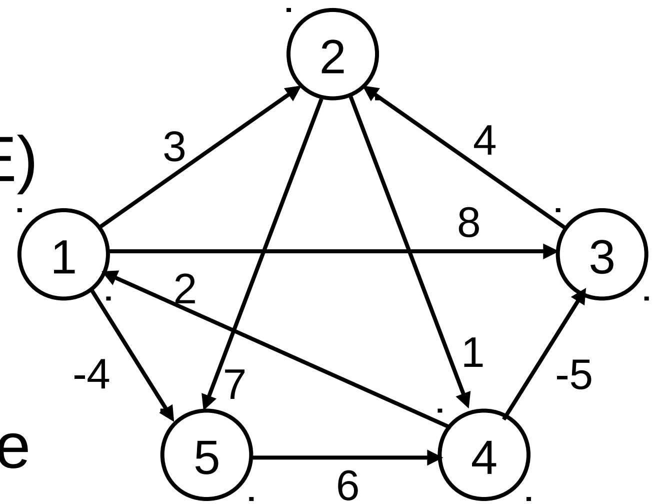
FASTER-APSP(W, n)

1. $L^{(1)} \leftarrow W$
2. $m \leftarrow 1$
3. **while** $m < n - 1$
4. **do** $L^{(2m)} \leftarrow \text{EXTEND}(L^{(m)}, L^{(m)}, n)$
5. $m \leftarrow 2*m$
6. **return** $L^{(m)}$

- OK to overshoot: products don't change after $L^{(n-1)}$
- **Running Time: Θ**$(n^3 \lg n)$

The Floyd-Warshall Algorithm

- **Given:**
  - Directed, weighted graph G = (V, E)
  - Negative-weight edges may be present
  - No negative-weight cycles could be present in the graph
- **Compute:**
  - The shortest paths between all pairs of vertices in a graph



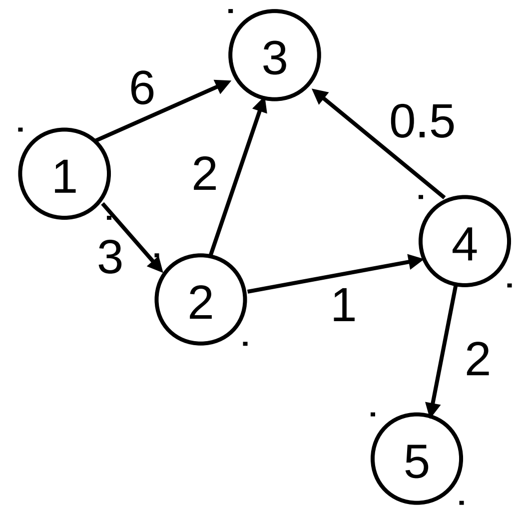
The Structure of a Shortest Path

- Vertices in G are given by

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

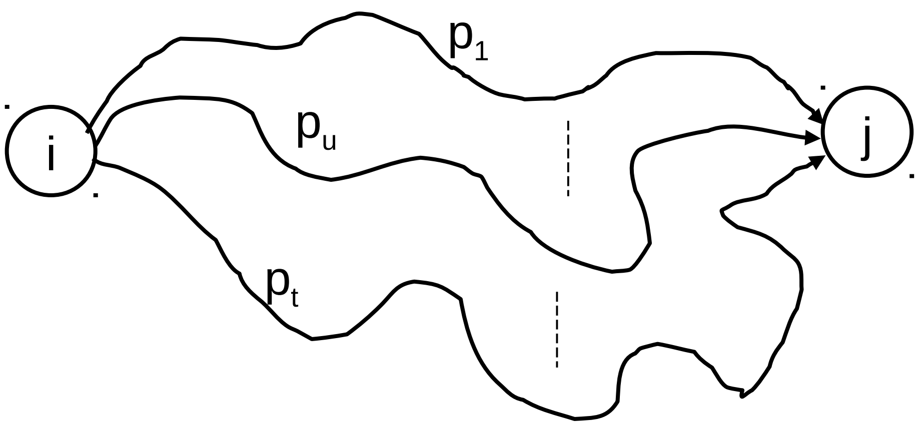
- Consider a path $p = \langle v_1, v_2, \ldots, v_l \rangle$
  - An **intermediate** vertex of p is any vertex in the set $\{v_2, v_3, \ldots, v_{l-1}\}$
  - E.g.: $p = \langle 1, 2, 4, 5 \rangle$: $\{2, 4\}$

    $p = \langle 2, 4, 5 \rangle$: $\{4\}$



The Structure of a Shortest Path

- For any pair of vertices i, j ∈ V, consider all paths from i to j whose intermediate vertices are all drawn from a subset {1, 2, ..., k}
  - Find p, a minimum-weight path from these paths

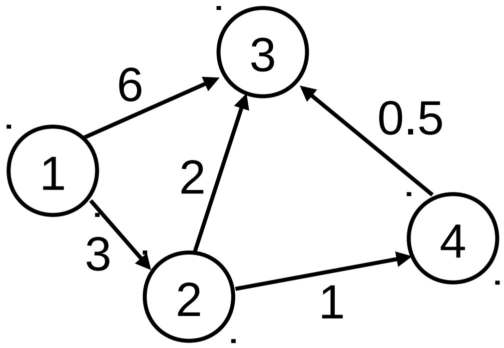


No vertex on these paths has index > k

Example

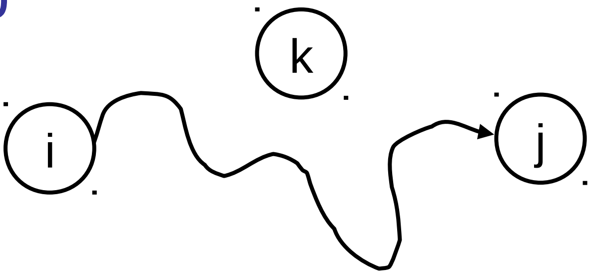
$d_{ij}^{(k)}$ = the weight of a shortest path from vertex i to vertex j with all intermediary vertices drawn from {1, 2, ..., k}

- $d_{13}^{(0)} =$ 6
- $d_{13}^{(1)} =$ 6
- $d_{13}^{(2)} =$ 5
- $d_{13}^{(3)} =$ 5
- $d_{13}^{(4)} =$ 4.5

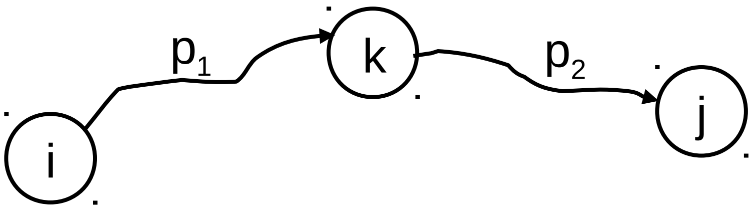


The Structure of a Shortest Path

- k is not an intermediate vertex of path p
  - Shortest path from i to j with intermediate vertices from {1, 2, …, k} is a shortest path from i to j with intermediate vertices from {1, 2, …, k - 1}



- k is an intermediate vertex of path p
  - p_1 is a shortest path from i to k
  - p_2 is a shortest path from k to j
  - k is not intermediary vertex of p_1, p_2
  - p_1 and p_2 are shortest paths from i to k with vertices from {1, 2, …, k - 1}



A Recursive Solution (cont.)

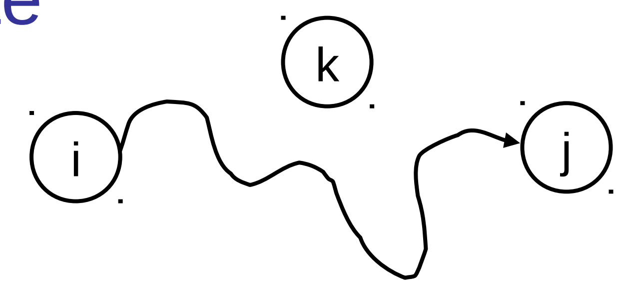
$d_{ij}^{(k)}$ = the weight of a shortest path from vertex i to vertex j with all intermediary vertices drawn from $\{1, 2, \ldots, k\}$

- $k = 0$
- $d_{ij}^{(k)} = w_{ij}$

A Recursive Solution (cont.)

$d_{ij}^{(k)}$ = the weight of a shortest path from vertex i to vertex j with all intermediary vertices drawn from {1, 2, …, k}

- $k \geq 1$
- **Case 1:** k is not an intermediate vertex of path p

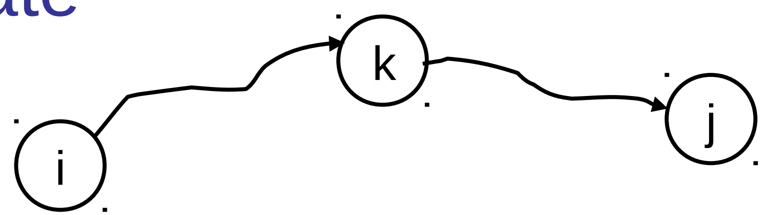


- $d_{ij}^{(k)} = \quad d_{ij}^{(k-1)}$

A Recursive Solution (cont.)

$d_{ij}^{(k)}$ = the weight of a shortest path from vertex i to vertex j with all intermediary vertices drawn from {1, 2, ..., k}

- $k \geq 1$
- **Case 2:** k is an intermediate vertex of path p

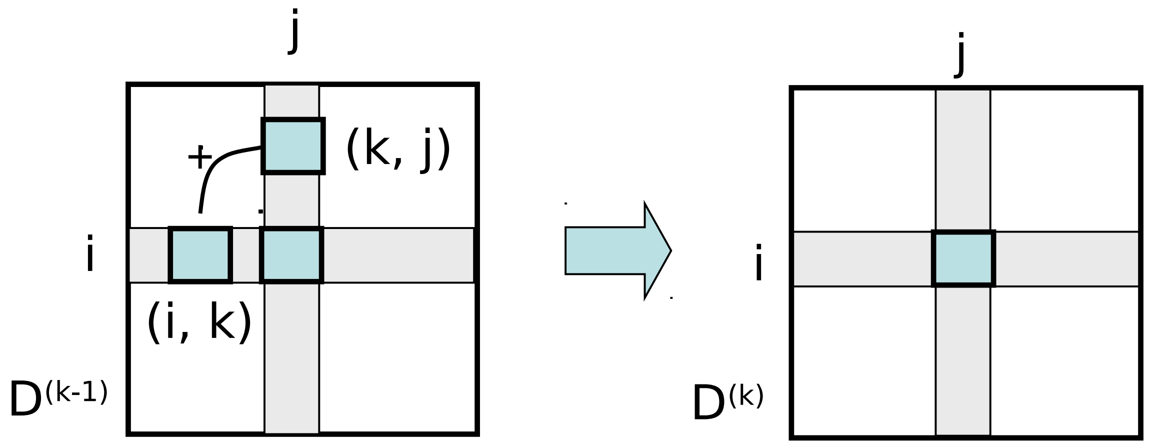


- $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

Computing the Shortest Path Weights

- $d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} & \text{if } k \geq 1 \end{cases}$
- The final solution: $D^{(n)} = (d_{ij}^{(n)})$:

  $d_{ij}^{(n)} = \delta(i, j) \ \forall\, i, j \in V$



The Floyd-Warshall algorithm

```
Floyd-Warshall(W[1..n][1..n])
D ← W     // D(0)
for k ← 1 to n do // compute D(k)
   for i ←1 to n do
      for j ←1 to n do
        if D[i][k] + D[k][j] < D[i][j] then
           D[i][j] ← D[i][k] + D[k][j]
return D
```

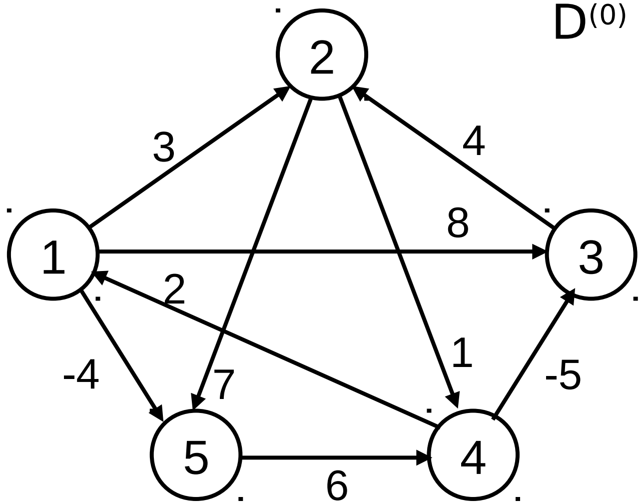
Running Time: $O(n^3)$

Computing predecessor matrix

- *How do we compute the predecessor matrix?*
  - Initialization: $p^{(0)}(i,j) = \begin{cases} nil & \text{if } i = j \text{ or } w_{ij} = \infty \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty \end{cases}$
  - Updating: $p^{(k)}(i,j) = \boldsymbol{p^{(k-1)}(i,j)}$ $if(d^{(k-1)}(i,j) <= d^{(k-1)}(i,k)+(d^{(k-1)}(k,j)$
  - $\boldsymbol{p^{(k-1)}(k,j)}$ $if(d^{(k-1)}(i,j) > d^{(k-1)}(i,k)+(d^{(k-1)}(k,j)$

```
Floyd-Warshall(W[1..n][1..n])
…
for k ← 1 to n do // compute D(k)
    for i ←1 to n do
        for i ←1 to n do
            if D[i][k] + D[k][j] < D[i][j] then
                D[i][j] ← D[i][k] + D[k][j]
                P[i][j] ← P[k][j]
return D
```

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



$D^{(0)} = W$

	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0

$D^{(1)}$

	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

$D^{(2)}$

	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

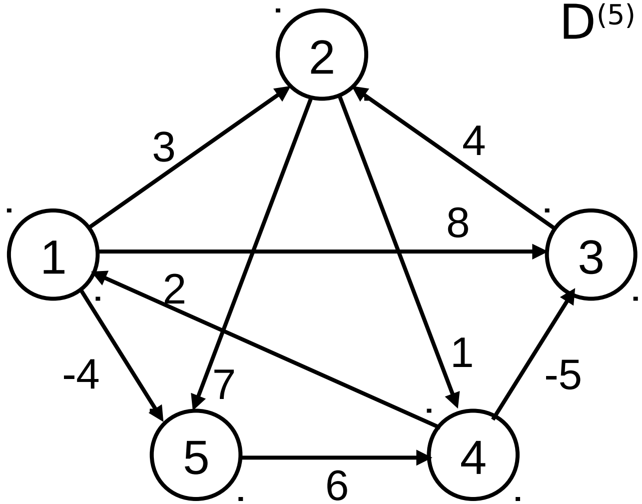
$D^{(3)}$

0	3	8	4	-4
∞	0	∞	1	7
∞	4	0	5	11
2	-1	-5	0	-2
∞	∞	∞	6	0

$D^{(4)}$

0	3	-1	4	-4	
3	0	-4	1	-1	
7	4	0	5	3	------
2	-1	-5	0	-2	
8	5	1	6	0	

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



$D^{(5)}$

	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

$P^{(5)}$

	1	2	3	4	5
1	-	3	4	5	1
2	4	-	4	2	1
3	4	3	-	2	1
4	4	3	4	-	1
5	4	3	4	5	-

Source: 5, Destination: 1

Shortest path: 8

Path: 5 …1 : 5…4…1: 5->4…1: 5->4->1

Source: 1, Destination: 3

Shortest path: -3

Path: 1 …3 : 1…4…3: 1…5…4…3: 1->5->4->3

PrintPath for Warshall’s Algorithm

```
PrintPath(s, t)
{
  if(P[s][t]==nil) {print(“No path”);return;}
  else if (P[s][t]==s){
      print(s);
  }
  else{
      print_path(s,P[s][t]);
      print_path(P[s][t], t);
  }
}
Print (t) at the end of the PrintPath(s,t)
```

Question

- Why should we use D[i, j] instead of $D^{(k)}[i, j]$?
- Exercise:
  - 25.2-4: Memory $O(n^2)$
  - 25.2-6: Negative weight cycle
  - Find the shortest positive cycle

Transitive closure of the graph

- Input:
  - Un-weighted graph G: $W[i][j] = 1$, if $(i,j) \in E$, $W[i][j] = 0$ otherwise.
- Output:
  - $T[i][j] = 1$, if there is a path from i to j in G, $T[i][j] = 0$ otherwise.
- Algorithm:
  - Just run Floyd-Warshall with weights 1, and make $T[i][j] = 1$, whenever $D[i][j] < \infty$.
  - More efficient: use only Boolean operators

Transitive closure algorithm

```
Transitive-Closure(W[1..n][1..n])
T ← W     // T^(0)
for k ← 1 to n do // compute T^(k)
   for i ←1 to n do
      for i ←1 to n do
         T[i][j] ← T[i][j] ∨ (T[i][k] ∧ T[k][j])
return T
```

Readings

- Chapters 25