MIDTERM EXAM

- When: June 30, 5th - 6th period (now).
- Where: Lecture Theatre.
- Scope: Lectures 1 to 6.
- What you **CAN** use:
  - Lecture handouts from the course webpage (6 slides x page).
  - Textbooks, dictionary, calculator.
- What you **CANNOT** use:
  - Exercise sheets.
  - Notes, memos, etc.
  - Computer, smart-phone, cell-phone.

ALGORITHMS AND DATA STRUCTURES II

Lecture 6

All Pairs Shortest Paths,

Transitive closure.

Lecturer: K. Markov

markov@u-aizu.ac.jp

OUTLINE

- Applications of all pairs shortest path algorithms.
- Direct methods to solve the problem:
  - Matrix multiplication
  - Floyd's algorithm.
- Transitive closure.
  - Warshall's algorithm.

ALL PAIRS SHORTEST PATH

- Applications
  - Computer networks.
  - Aircraft network (e.g. flying time, fares).
  - Railroad network.
  - Table of distances between all pairs of cities for a road atlas.

ALL PAIRS SHORTEST PATH

- If edges are **non**-negative:
  - Run Dijkstra's algorithm n-times, once for each vertex as the source.
  - Running time: $\boldsymbol{O(nm\log n)}$
- If edges are negative:
  - Run Bellman-Ford's algorithm n-times.
  - Running time: $\boldsymbol{O(n^2m)}$

ALL PAIRS SHORTEST PATH

- Adjacency matrix representation

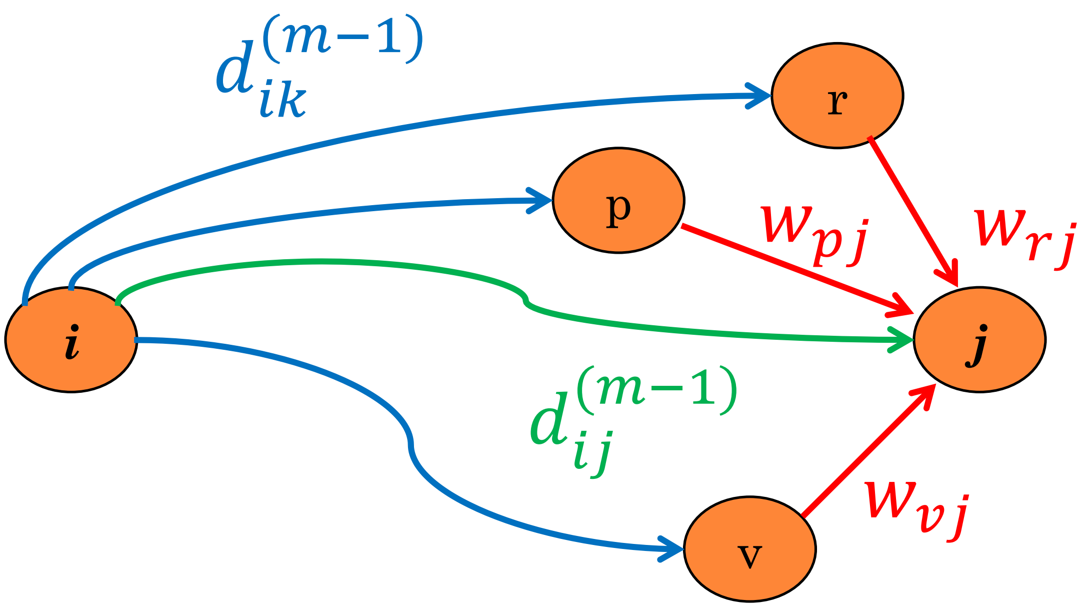
- $\boldsymbol{w}$: $E \rightarrow \mathcal{R}$ as $\boldsymbol{n}\ x\ \boldsymbol{n}$ matrix $\boldsymbol{W}$

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

ALL PAIRS SHORTEST PATH

- Matrix multiplication idea.
  - $d_{ij}^{(m)}$: minimum weight of any path from i to j that contains at most **m** edges.
  - $d_{ij}^{(m)} = \min\left(d_{ij}^{(m-1)}, \min_{1<k<n}\left(d_{ik}^{(m-1)} + w_{kj}\right)\right)$

1 path vs 3 paths



Look at all possible predecessors ***k*** of ***j*** and compare!

MATRIX MULTIPLICATION

- Recursion.
  - 1. $d_{ij}^{(1)} = w_{ij}$
  - 2. $d_{ij}^{(m)} = \min\left(d_{ij}^{(m-1)}, \min_{1<k<n}\left(d_{ik}^{(m-1)} + w_{kj}\right)\right)$

    $= \min_{1<k<n}\left(d_{ik}^{(m-1)} + w_{kj}\right)$ (since $w_{jj} = 0,\ \forall j$)
- Equivalent matrix operations.
  - $C = A \cdot B, \quad c_{ij} = \sum_{1\le k\le n} a_{ik} b_{kj}$
  - $d_{ij}^{(m)} \to c_{ij}, d_{ij}^{(m-1)} \to a_{ik},\ w_{kj} \to b_{kj},\ \min \to \sum,\ + \to \cdot$
  - Compute series of matrices

    $D^{(1)}, D^{(2)}, \dots, D^{(n-1)}$ such that $D^{(m)} = D^{(m-1)} W$

MATRIX MULTIPLICATION

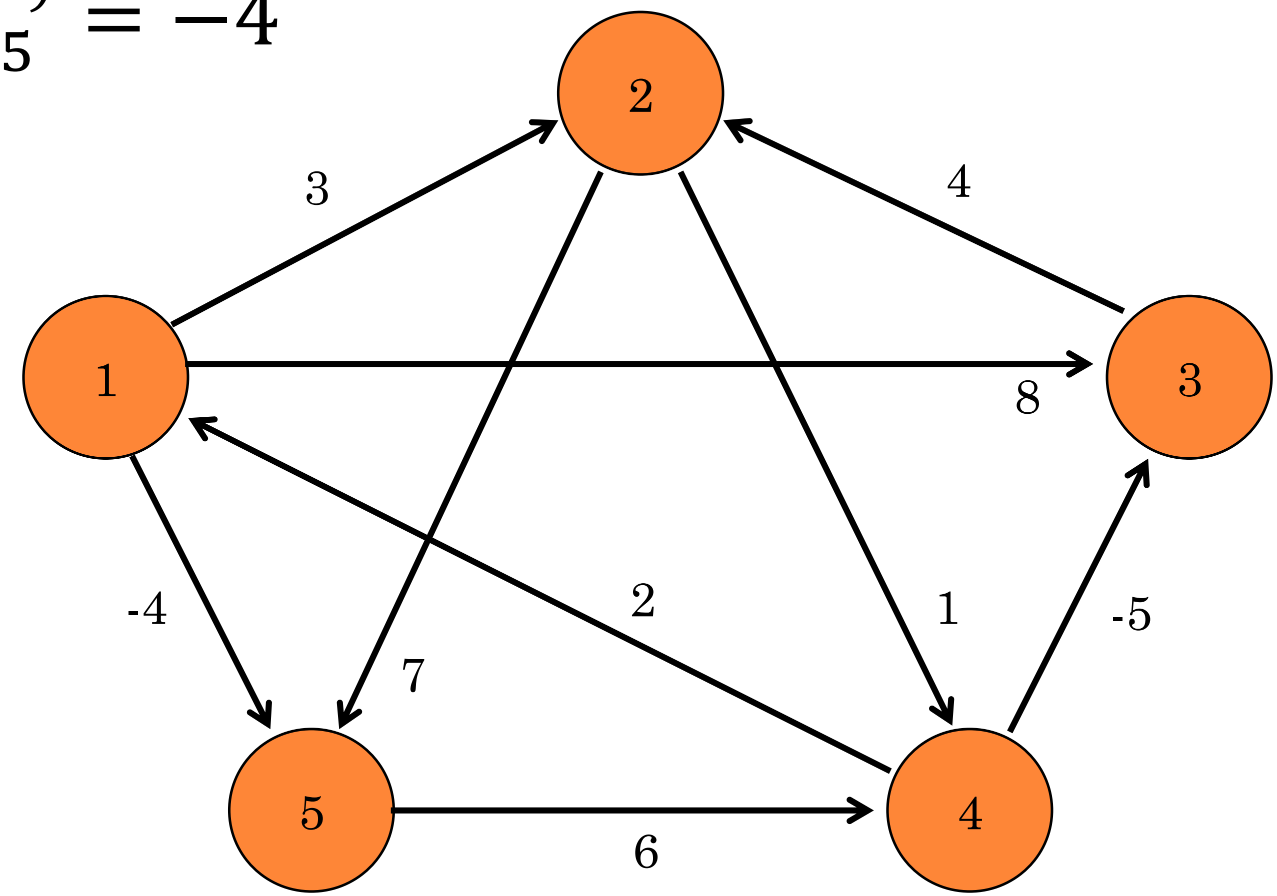
- Algorithm pseudo-code.

```
def EXTEND-SHORTEST-PATHS (D,W)
    // Extends the shortest path computed so far
    // by one more edge.
   n = D.rows
   let D′ = (d′_ij) be an n x n matrix
   for i = 1 to n:
        for j =1 to n:
            d′_ij = ∞
            for k = 1 to n:
                d′_ij = min (d′_ij , d_ik + w_kj)
   return D′
```

- Time complexity: $O(n^3)$

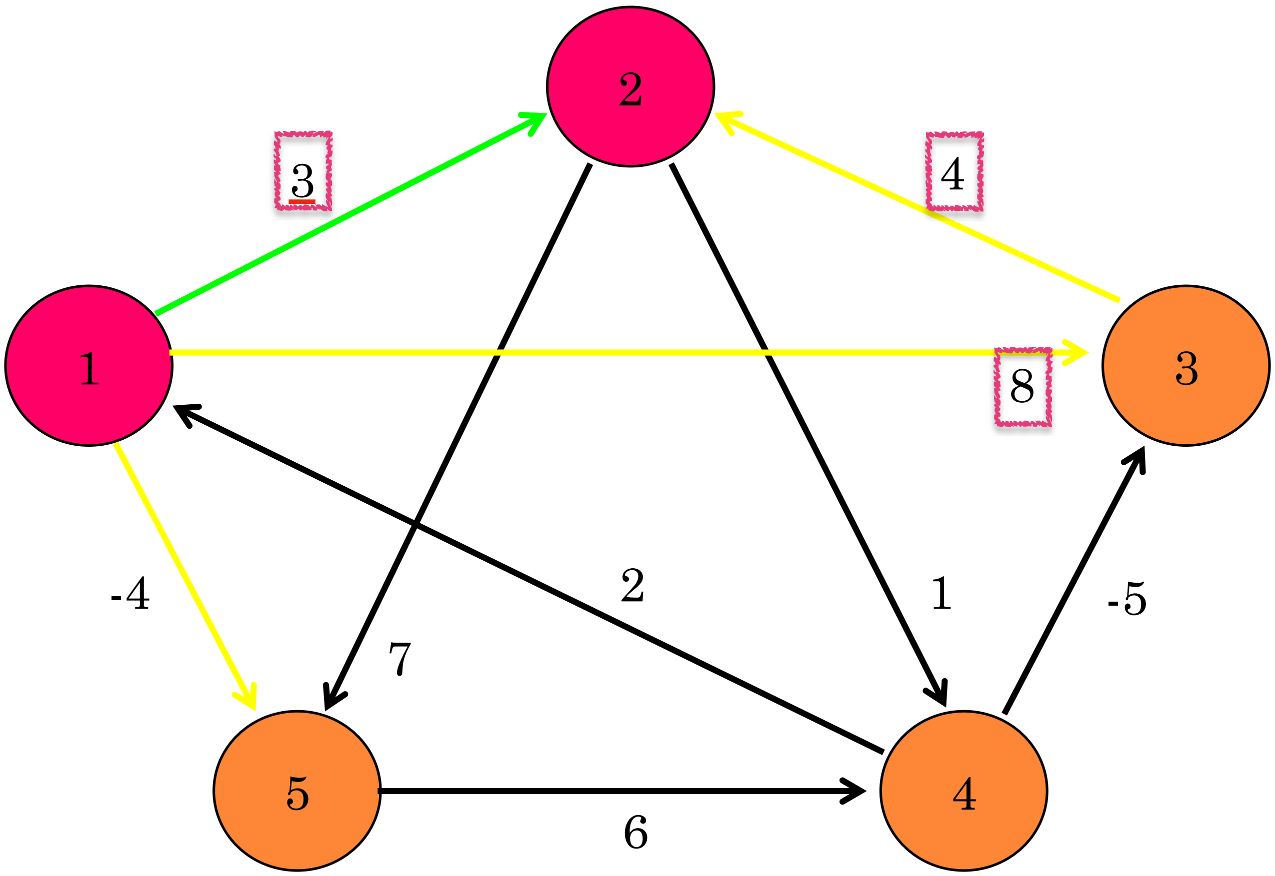
MATRIX MULTIPLICATION

- Example: $d_{12}^{(1)} = 3, d_{13}^{(1)} = 8, d_{14}^{(1)} = \infty,$ $d_{15}^{(1)} = -4$



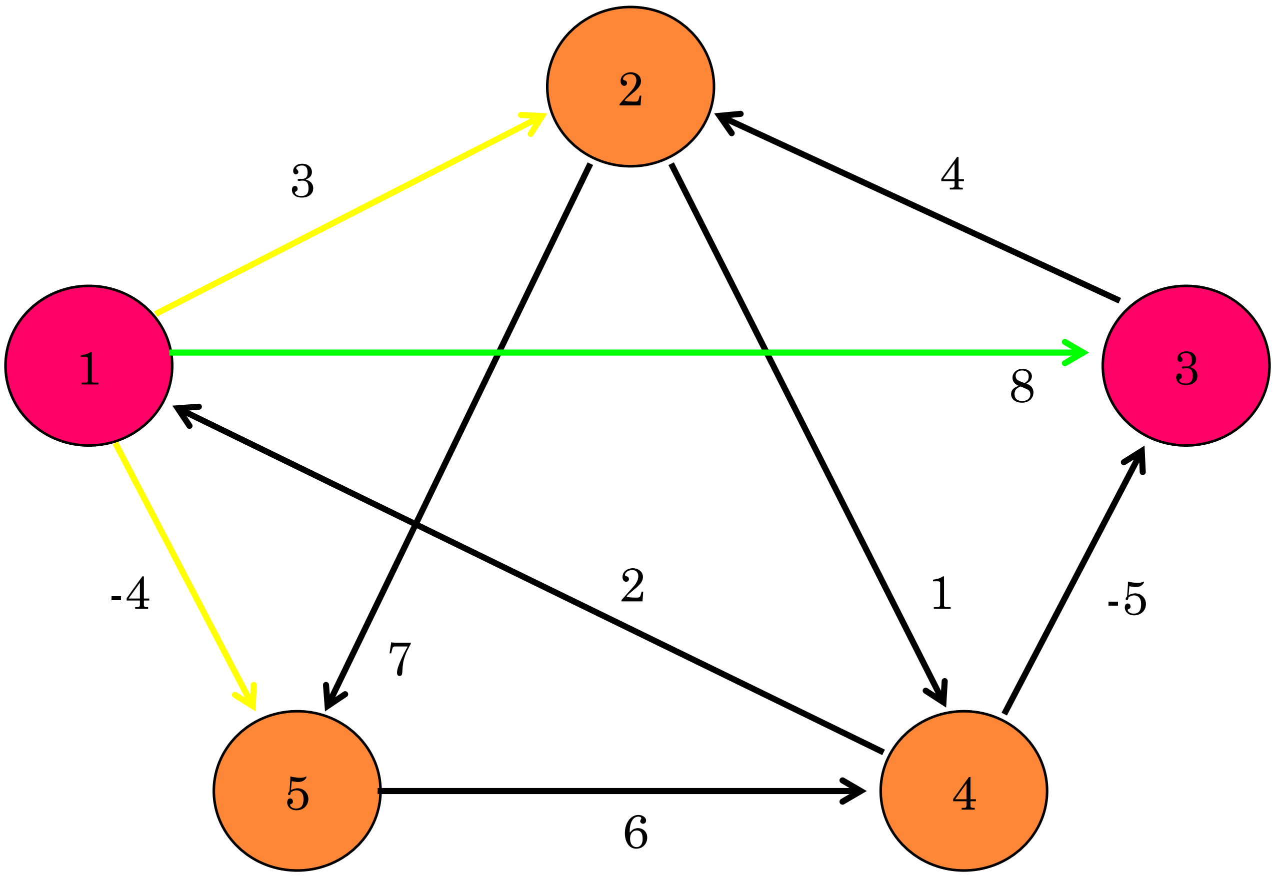
MATRIX MULTIPLICATION

- Example - $d_{12}^{(2)} = \min(3, 8 + 4) = 3$



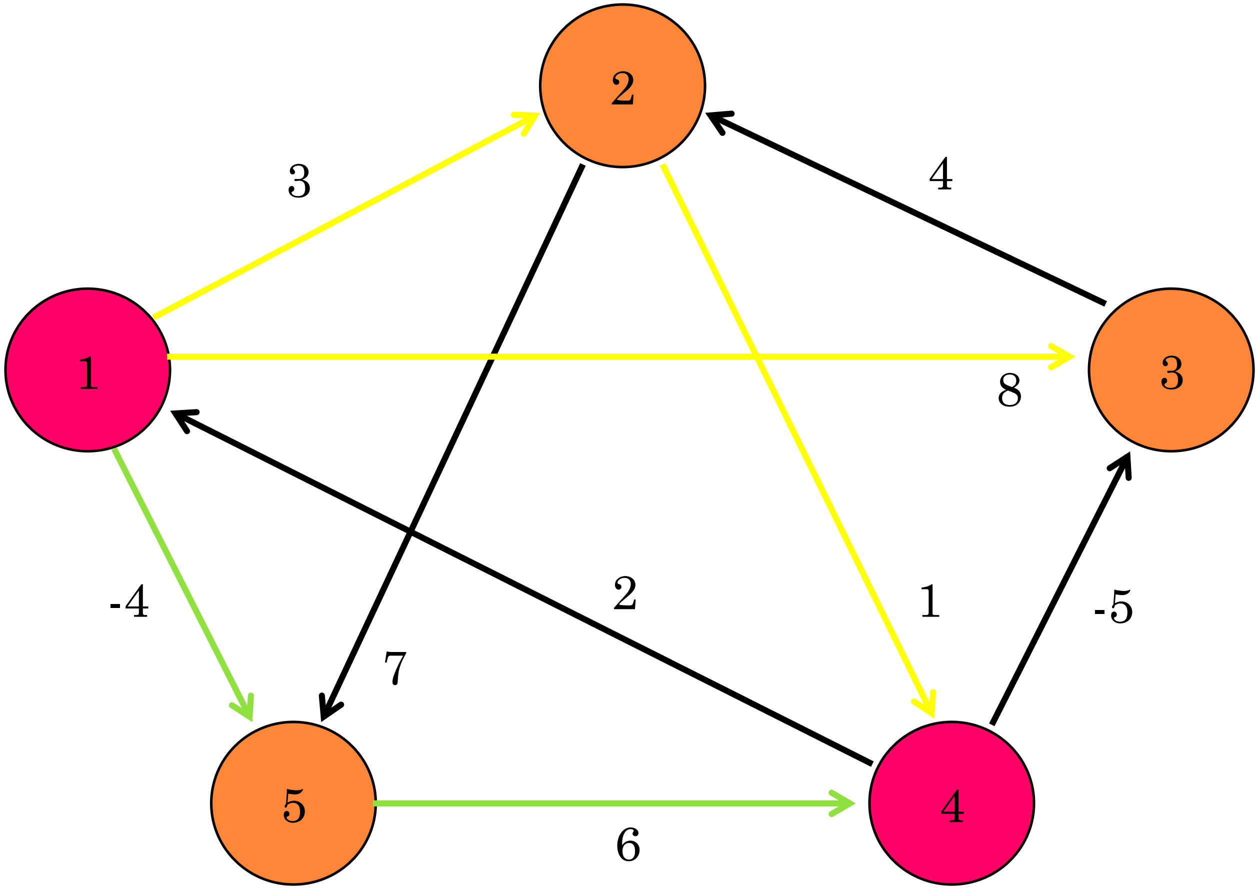
MATRIX MULTIPLICATION

- Example - $d_{13}^{(2)} = \min(8, \underline{\infty}) = 8$



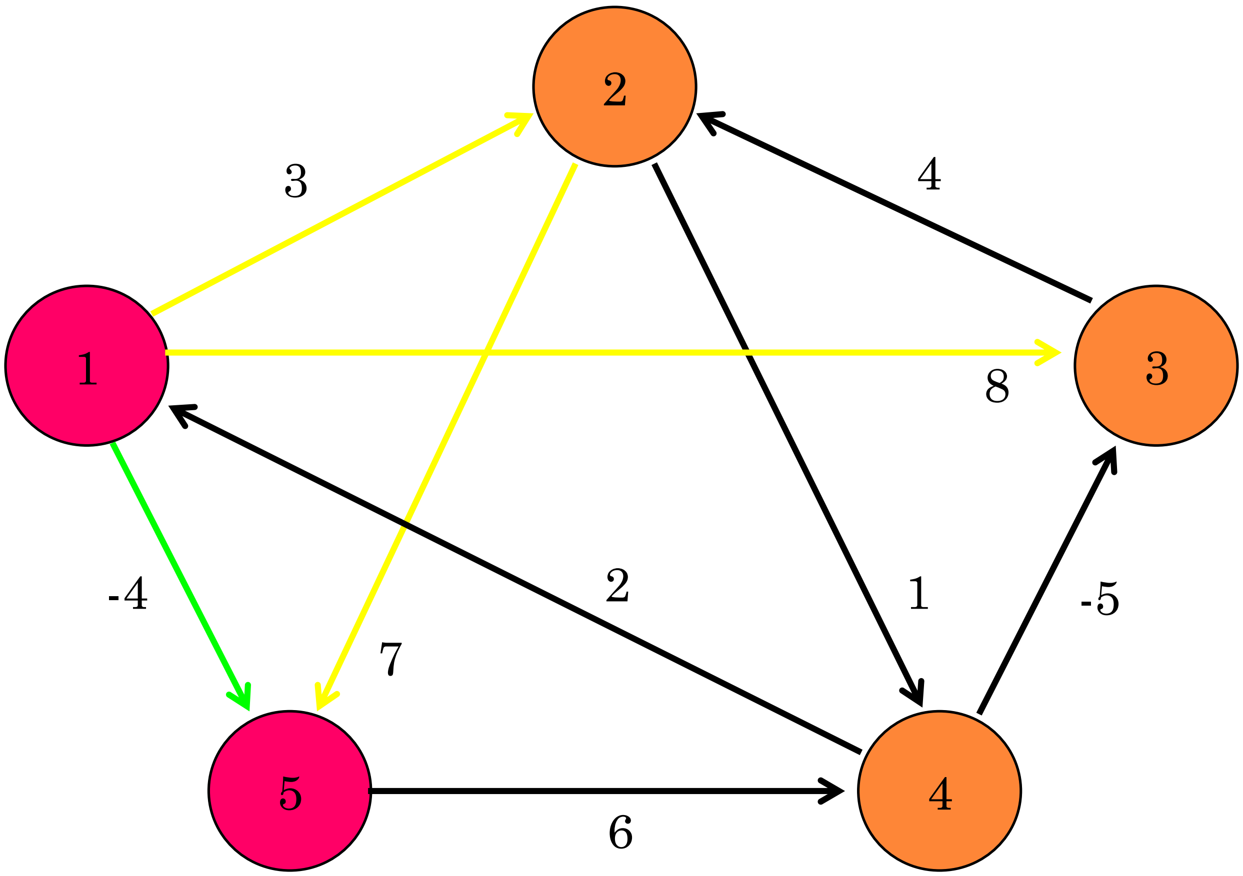
MATRIX MULTIPLICATION

- Example - $d_{14}^{(2)} = \min(\infty, -4 + 6) = 2$



MATRIX MULTIPLICATION

- Example - $d_{15}^{(2)} = \min(-4, 3+7) = -4$



MATRIX MULTIPLICATION

- Example.

Graph adjacency matrix

$$\begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

Find $d_{14}^{(2)}$

MATRIX MULTIPLICATION

- Example.

Forth Column

First Row

$$d_{14}^{(2)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \end{pmatrix} \cdot \begin{pmatrix} \infty \\ 1 \\ \infty \\ 0 \\ 6 \end{pmatrix}$$

$$= \min(\infty, 4, \infty, \infty, 2)$$

$$= 2$$

MATRIX MULTIPLICATION

- True matrix multiplication - $\boldsymbol{C} = \boldsymbol{A} \cdot \boldsymbol{B}$

$\Rightarrow$ $$\boldsymbol{c}_{ij} = \sum_{k=1}^{n} \boldsymbol{a}_{ik} \cdot \boldsymbol{b}_{kj}$$

- Compare $\boldsymbol{D}^{(m)} = \boldsymbol{D}^{(m-1)} \cdot \boldsymbol{W}$

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

- Compute sequence of $\boldsymbol{n-1}$ matrices:

$\boldsymbol{D}^{(1)} = \boldsymbol{D}^{(0)} \cdot \boldsymbol{W} = \boldsymbol{W}$, $\boldsymbol{D}^{(2)} = \boldsymbol{D}^{(1)} \cdot \boldsymbol{W} = \boldsymbol{W}^2$,

$\boldsymbol{D}^{(3)} = \boldsymbol{D}^{(2)} \cdot \boldsymbol{W} = \boldsymbol{W}^3$, ..., $\boldsymbol{D}^{(n-1)} = \boldsymbol{D}^{(n-2)} \cdot \boldsymbol{W} = \boldsymbol{W}^{n-1}$

ALL PAIRS SHORTEST PATHS

- Algorithm pseudo-code:

```
def ALL-PAIRS-SHORTEST-PATHS (W)
    // Given the weight matrix W, returns APSP matrix D^(n-1)
    n = W.rows
    D^(1) = W
    for m = 2 to n – 1:
        D^(m) = EXTEND-SHORTEST-PATHS (D^(m-1),W)
    return D^(n-1)
```

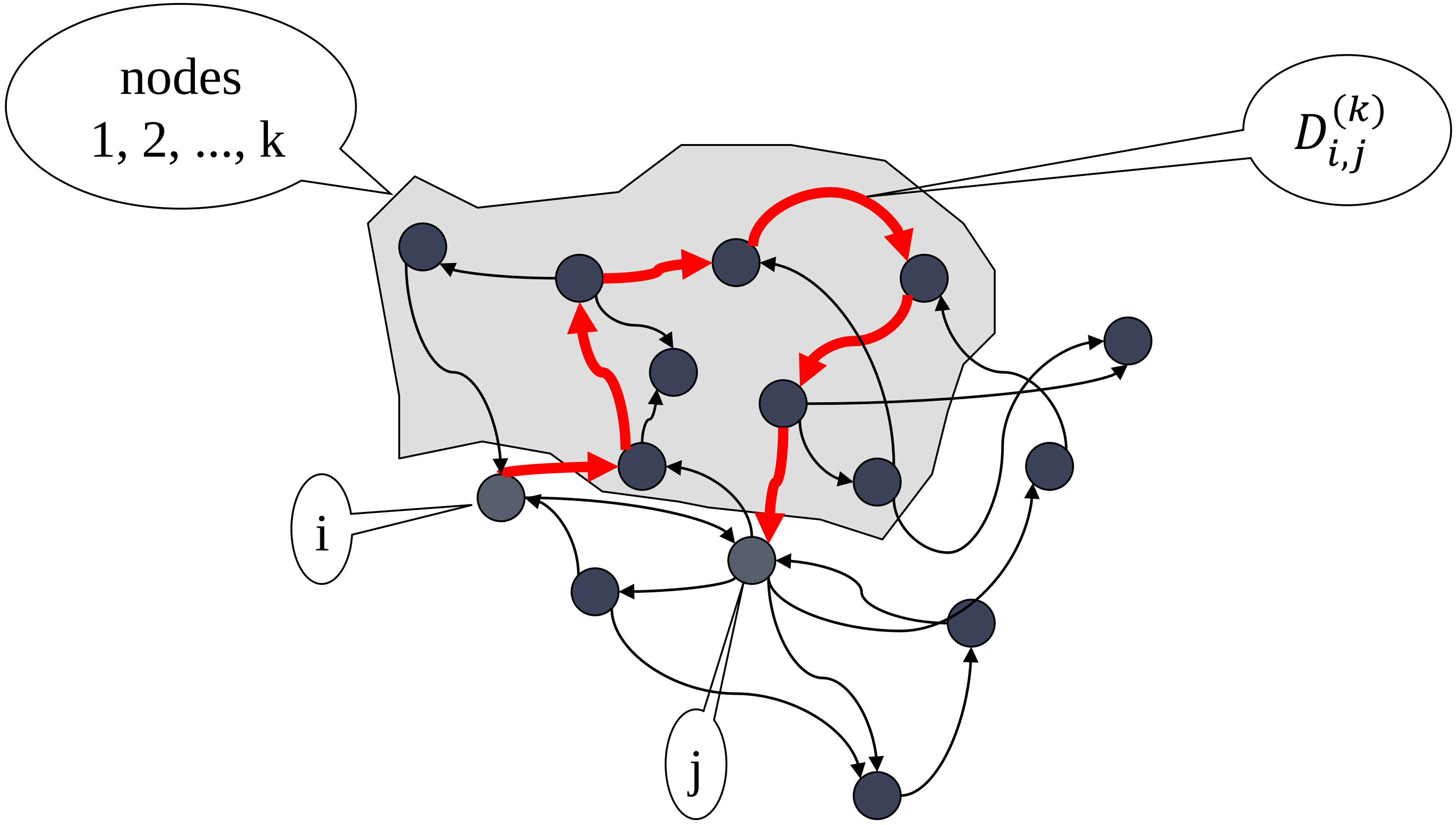
- Time complexity: $\mathbf{O}(\boldsymbol{n}^4)$

ALL PAIRS SHORTEST PATHS

- Floyd's algorithm:
  - Let $\boldsymbol{D}^{(k)}[i, j] = weigth$ of a shortest path from $\boldsymbol{v}_i$ to $\boldsymbol{v}_j$ using only vertices from $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_k\}$ as intermediate vertices in the path.
  - Obviously: $\boldsymbol{D}^{(0)} = \boldsymbol{W}$, we need $\boldsymbol{D}^{(n)}$
  - How to compute $\boldsymbol{D}^{(k)}$ from $\boldsymbol{D}^{(k-1)}$?

ALL PAIRS SHORTEST PATHS

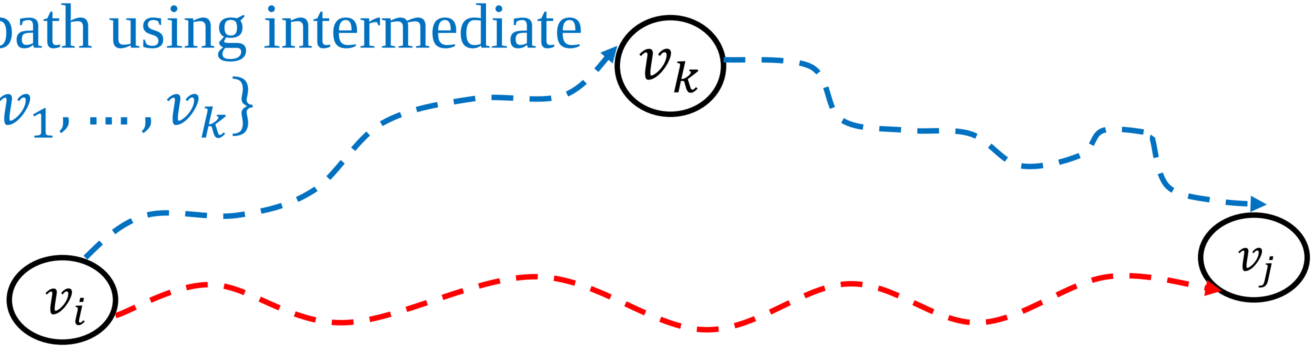
- Example of $D_{i,j}^{(k)}$



ALL PAIRS SHORTEST PATHS

- Floyd's algorithm:
  - Case 1: The shortest path from $\boldsymbol{v}_i$ to $\boldsymbol{v}_j$ does not use $\boldsymbol{v}_k$. Then $\boldsymbol{D}^{(k)}[i,j] = \boldsymbol{D}^{(k-1)}[i,j]$.
  - Case 2: The shortest path from $\boldsymbol{v}_i$ to $\boldsymbol{v}_j$ does use $\boldsymbol{v}_k$. Then $\boldsymbol{D}^{(k)}[i,j] = \boldsymbol{D}^{(k-1)}[i,k] + \boldsymbol{D}^{(k-1)}[k,j]$.

Shortest path using intermediate vertices $\{v_1, \dots, v_k\}$



Shortest Path using intermediate vertices $\{v_1, \dots, v_{k-1}\}$

ALL PAIRS SHORTEST PATHS

- Floyd's algorithm:
  - Since

$$\boldsymbol{D}^{(k)}[i,j] = \boldsymbol{D}^{(k-1)}[i,j] \qquad or$$
$$\boldsymbol{D}^{(k)}[i,j] = \boldsymbol{D}^{(k-1)}[i,k] + \boldsymbol{D}^{(k-1)}[k,j].$$

  - We conclude:

$$\underline{\boldsymbol{D}^{(k)}[i,j] = \min(\boldsymbol{D}^{(k-1)}[i,j],}$$
$$\underline{\boldsymbol{D}^{(k-1)}[i,k] + \boldsymbol{D}^{(k-1)}[k,j]).}$$

ALL PAIRS SHORTEST PATHS

- Floyd's algorithm – pseudo-code

```
def FLOYD (W)
    // Given weight matrix W, returns APSP matrix D^(n)
    n = W.rows
    D^(0) = W
    for k = 1 to n:
        for i = 1 to n:
            for j = 1 to n:
                d_ij^(k) = min (d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
    return D^(n)
```

- Time complexity: $O(n^3)$

better than ALL-PAIRS-SHORTEST-PATHS

want to know the path

EXTRACTING THE SHORTEST PATH

- The predecessor pointers $pred(i, j)$ can be used to extract the final path. The idea is as follows:
- Whenever we discover that the shortest path from i to j passes through an intermediate vertex k, we set $pred(i, j) = k$.
- If the shortest path does not pass through any intermediate vertex, then $pred(i, j) = nil$.
- To find the shortest path from i to j, we consult $pred(i, j)$.
- If it is nil, then the shortest path is just the edge (i, j).
- Otherwise, we recursively compute the shortest path
- from i to $pred(i, j)$ and the shortest path from $pred(i, j)$ to j.

ALL PAIRS SHORTEST PATHS

- Path extraction algorithm

```
def Path(i,j):

    if pred[i, j] == NIL:
        return (i, j)

    else:
        Path (i, pred[i,j])
        Path (pred[i, j], j)
```

ALL PAIRS SHORTEST PATHS

- Path extraction example

There is no node bw 2 and 5
=> 2 and 5 are adjancy

Find the shortest path from vertex 2 to vertex 3.

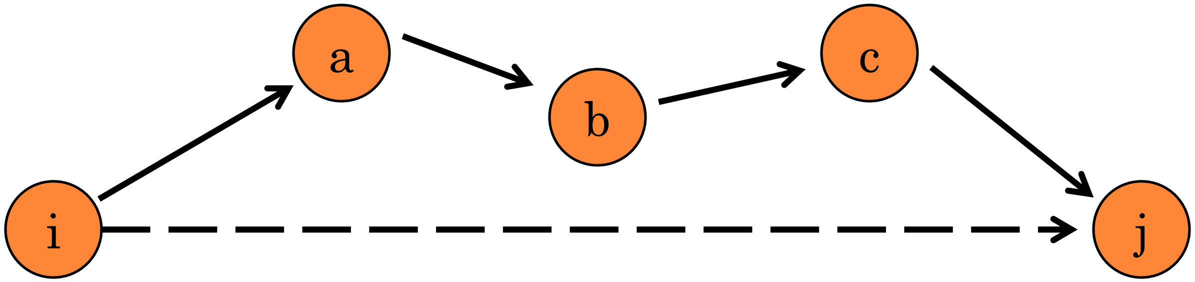
2..3	Path(2, 3)	$pred[2,3] = 4$	
2..4..3	Path(2, 4)	$pred[2,4] = 5$	
2..5..4..3	Path(2, 5)	$pred[2,5] = nil$	$Output(2,5)$
25..4..3	Path(5, 4)	$pred[5,4] = nil$	$Output(5,4)$
254..3	Path(4, 3)	$pred[4,3] = 6$	
254..6..3	Path(4, 6)	$pred[4,6] = nil$	$Output(4,6)$
2546..3	Path(6, 3)	$pred[6,3] = nil$	$Output(6,3)$
25463			

TRANSITIVE CLOSURE

- Given a directed graph $G = (V, E)$ find whether there is a path from $\boldsymbol{v}_i$ to $\boldsymbol{v}_j$ for all vertex pairs $\boldsymbol{v}_i, \boldsymbol{v}_j \in V$.

- **Transitive closure** of graph G is the graph $G^* = (V, E^*)$ where

$E^* = \{(i, j)\text{: there is a path from } \boldsymbol{v}_i \text{ to } \boldsymbol{v}_j \text{ in } G\}$



TRANSITIVE CLOSURE

- Solution 1
  - Set $w_{ij} = 1$ and run the Floyd's algorithm.
  - Time complexity: $\boldsymbol{O(n^3)}$

- Solution 2 (Warshall's algorithm)
  - Define $t_{ij}^{(k)}$ such that

$$\begin{cases} t_{ij}^{(0)} = 0, & if\ i \neq j \text{ and } (i,j) \notin E, \\ \underline{t_{ij}^{(0)} = 1,} & \underline{if\ i = j\ \boxed{\text{or}}\ (i,j) \in E} \end{cases}$$

  - and for $k \geq 1$

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

TRANSITIVE CLOSURE

- Warshall's algorithm – pseudo-code

```
def WARSHALL (G ):
    n = | V [G ] |
    for i = 1 to n:
        for j = 1 to n:
            if i = j  or (i,j ) ∈ E [G]:
                t_ij^(0) = 1
            else:
                t_ij^(0) = 0
    for k = 1 to n:
        for i = 1 to n:
            for j = 1 to n:
                t_ij^(k) = t_ij^(k-1) OR (t_ik^(k-1) AND t_kj^(k-1))
    return T^(n)
```

TRANSITIVE CLOSURE

- Warshall's algorithm
  - Same as Floyd's algorithm if we substitute "+" and "min" operations by "AND" and "OR" operations.
  - Time complexity: $\boldsymbol{O(n^3)}$

ALGORITHMS COMPARISON

Algorithm	Time complexity
$n \times$ Dijkstra's non-negative	$O(nm \log n)$
$n \times$ Bellman-Ford's negative	$O(n^2 m)$
Matrix Multiplication	$O(n^4)$
Floyd's	$O(n^3)$
Warshall's (transitive closure)	$O(n^3)$

That's All for Today!