MIDTERM EXAM

- When: May 7th, 3rd - 4th period (now).
- Where: M5 (here).
- 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: $O(nm \log n)$

- If edges are negative:
  - Run Bellman-Ford's algorithm n-times.
  - Running time: $O(n^2m)$

ALL PAIRS SHORTEST PATH

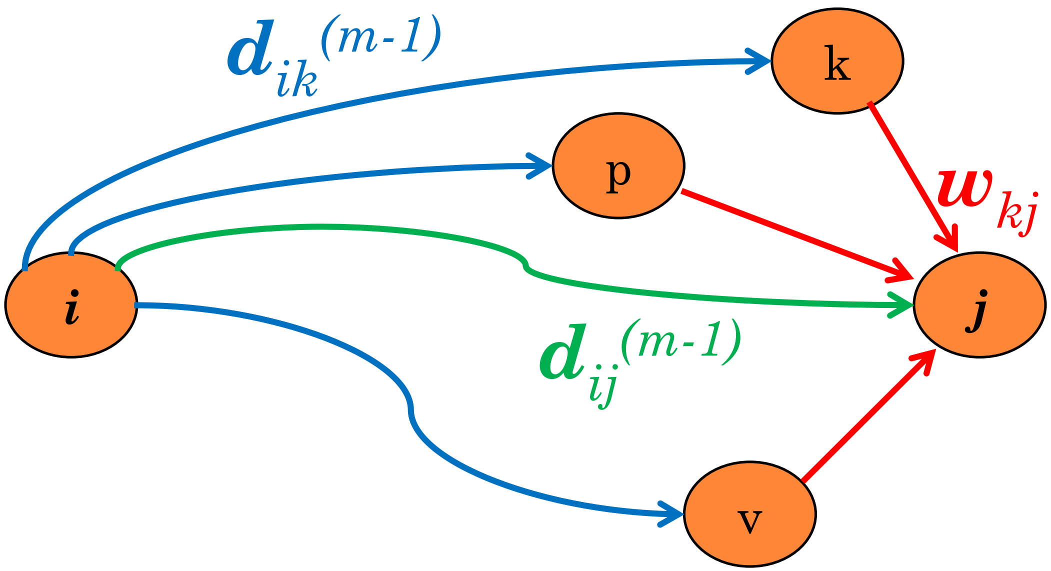
- Adjacency matrix representation

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

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

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



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

MATRIX MULTIPLICATION

- Recursion.
  - 1. $\boldsymbol{d}_{ij}^{(1)} = \boldsymbol{w}_{ij}$
  - 2. $\boldsymbol{d}_{ij}^{(m)} = \min(\boldsymbol{d}_{ij}^{(m-1)}, \min_{1\leq k\leq n} \{\boldsymbol{d}_{ik}^{(m-1)} + \boldsymbol{w}_{kj}\})$

    $= \min_{1\leq k\leq n} \{\boldsymbol{d}_{ik}^{(m-1)} + \boldsymbol{w}_{kj}\}$

    (since $\boldsymbol{w}_{jj}=0$ for all j)
- Equivalent matrix operations.
  - $\boldsymbol{C}=\boldsymbol{A}\cdot\boldsymbol{B}$, $\quad \boldsymbol{c}_{ij}=\sum_{1\leq k\leq n} \boldsymbol{a}_{ik}\cdot\boldsymbol{b}_{kj}$
  - $\boldsymbol{d}_{ij}^{(m)} \rightarrow \boldsymbol{c}_{ij}$, $\boldsymbol{d}_{ik}^{(m-1)} \rightarrow \boldsymbol{a}_{ik}$, $\boldsymbol{w}_{kj} \rightarrow \boldsymbol{b}_{kj}$, $\min \rightarrow \sum$, $+ \rightarrow \cdot$
  - Compute series of matrices

    $\boldsymbol{D}^{(1)}, \boldsymbol{D}^{(2)}, \ldots, \boldsymbol{D}^{(n-1)}$

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

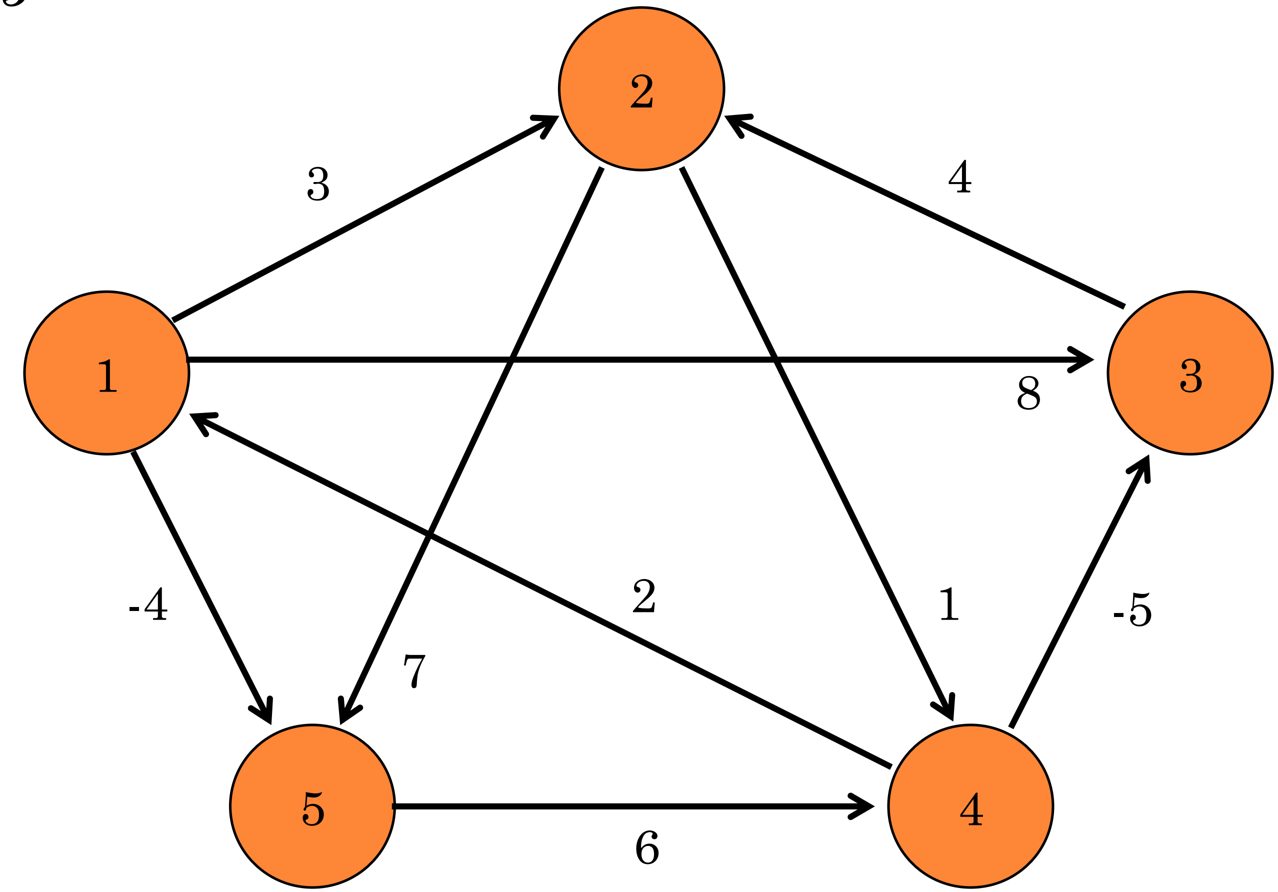
MATRIX MULTIPLICATION

- 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 $\boldsymbol{n}$ x $\boldsymbol{n}$ matrix
  for $i = 1$ **to** n:
    for $j = 1$ **to** n:
      $d'_{ij} = \infty$
      for $k = 1$ **to** n:
        $d'_{ij} = \min(d'_{ij}, d_{ik} + w_{kj})$
  return D'

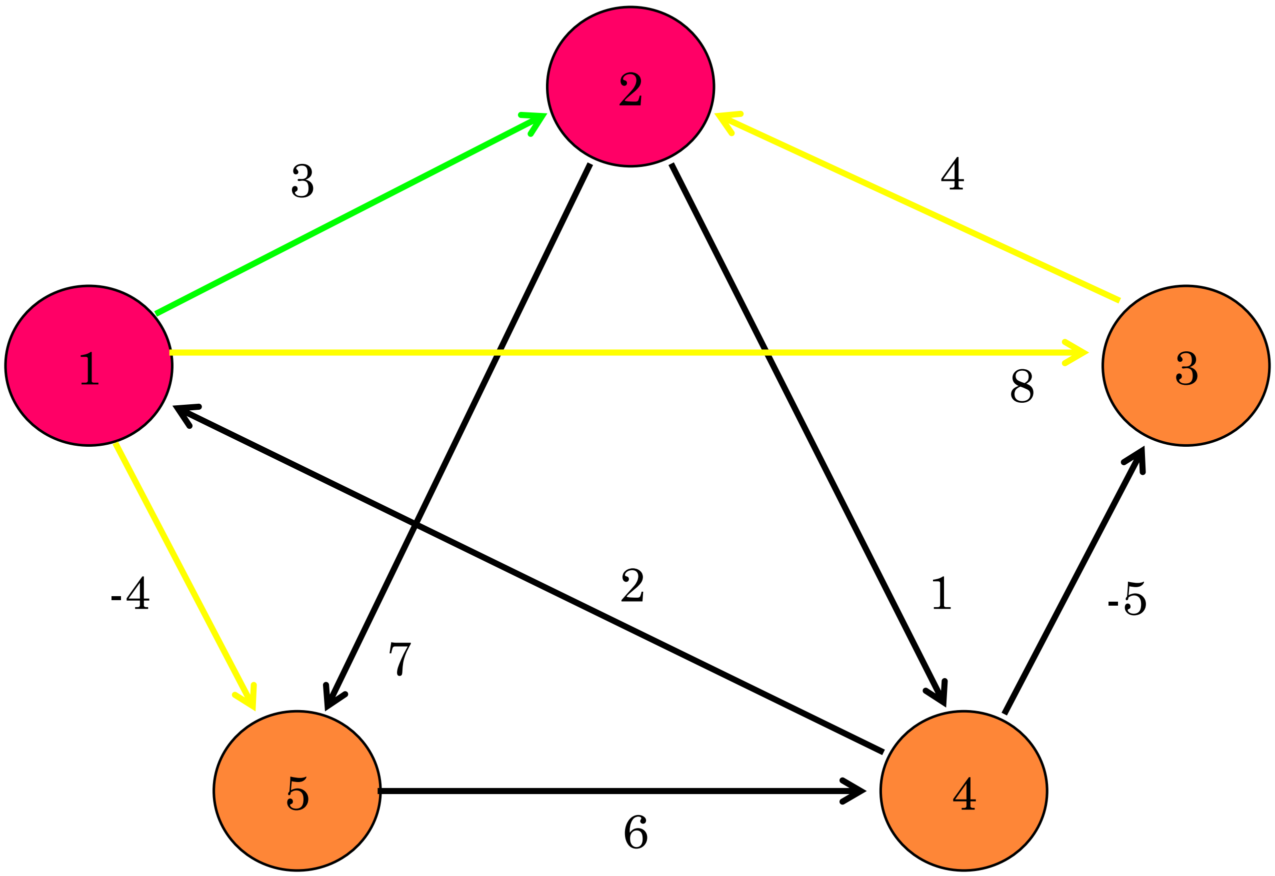
MATRIX MULTIPLICATION

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



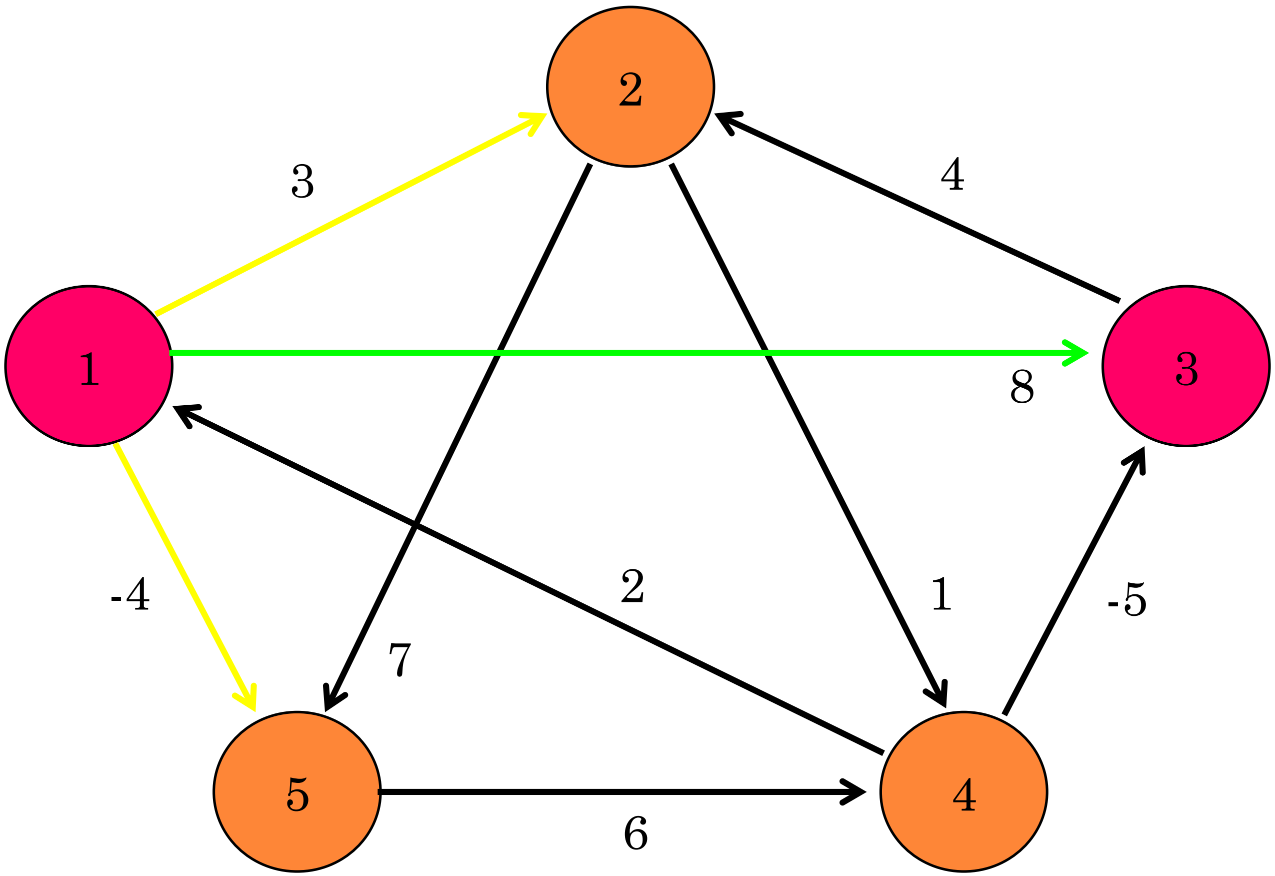
MATRIX MULTIPLICATION

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



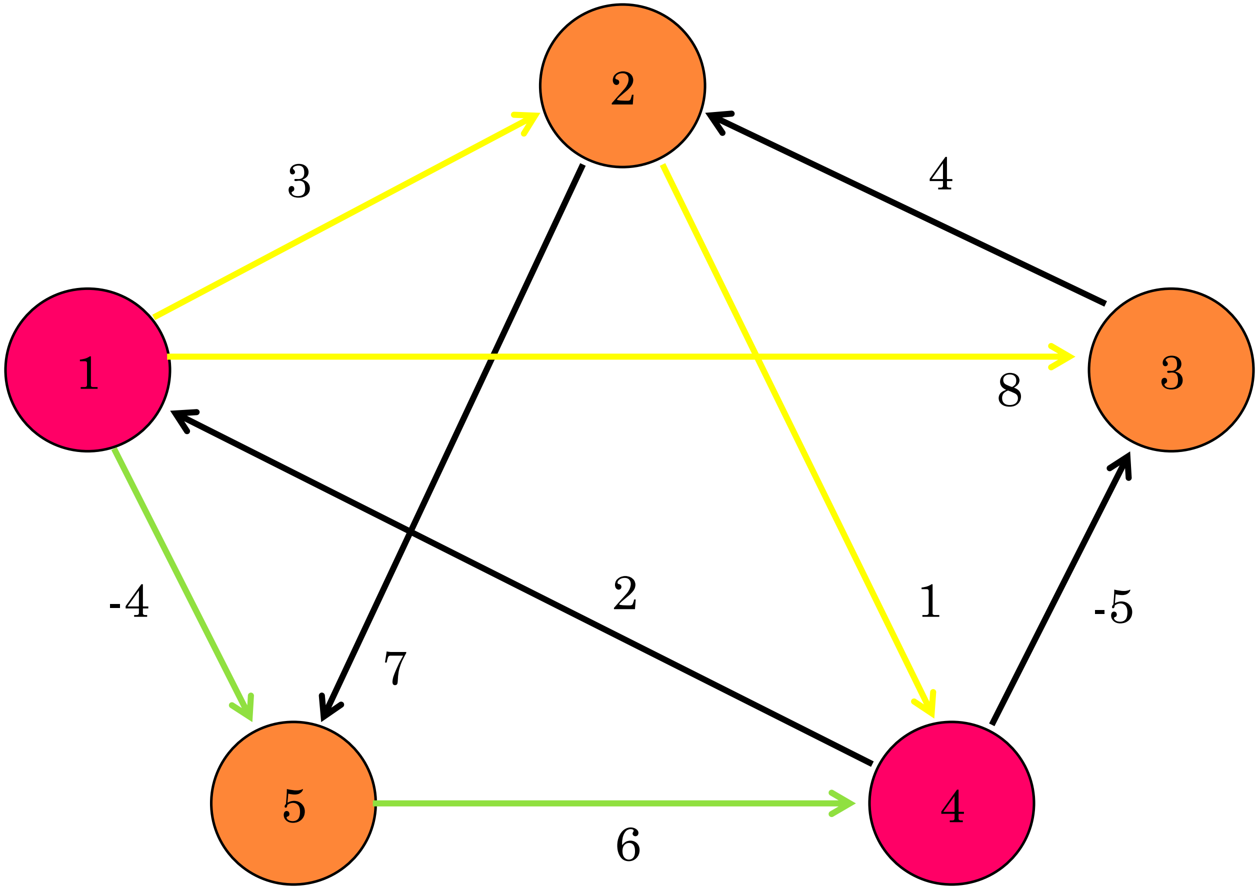
MATRIX MULTIPLICATION

- Example - $\boldsymbol{d}_{13}^{(2)} = min(8, \infty) = 8$



MATRIX MULTIPLICATION

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



MATRIX MULTIPLICATION

- Example.

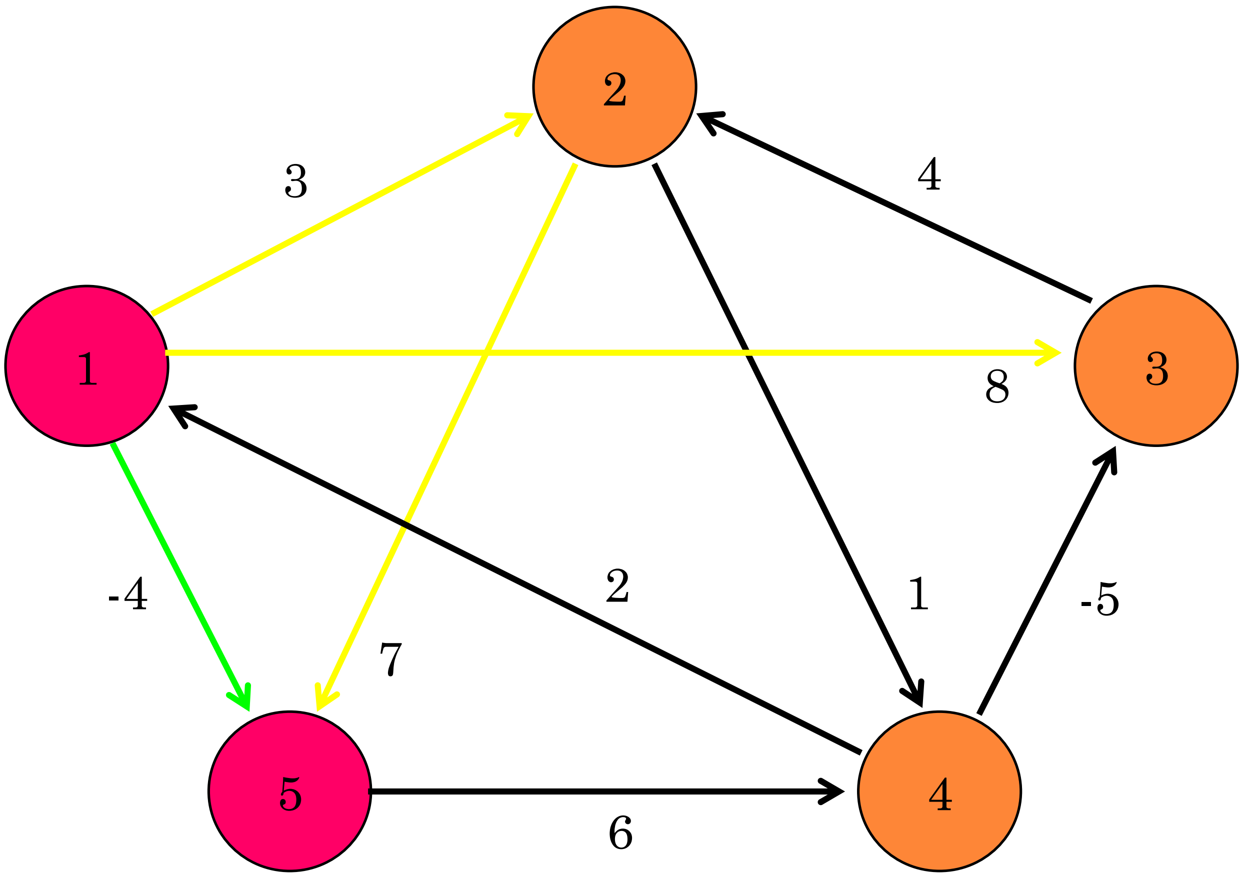
$$d_{14}^{(2)} = (0 \quad 3 \quad 8 \quad \infty \quad -4) \bullet \begin{pmatrix} \infty \\ 1 \\ \infty \\ 0 \\ 6 \end{pmatrix}$$

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

$$= 2$$

MATRIX MULTIPLICATION

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



MATRIX MULTIPLICATION

- True matrix multiplication - $C=A\cdot B$

$\Rightarrow$ $$c_{ij} = \Sigma_{k=1}^{n} \; a_{ik} \cdot b_{kj}$$

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

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

- Compute sequence of ***n-1*** matrices:

$$D^{(1)} = D^{(0)} \cdot W = W, \qquad D^{(2)} = D^{(1)} \cdot W = W^2,$$
$$D^{(3)} = D^{(2)} \cdot W = W^3, \quad \ldots., \quad D^{(n-1)} = D^{(n-2)} \cdot W = W^{n-1}$$

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: $O(n^4)$

ALL PAIRS SHORTEST PATHS

- Floyd's algorithm:

  - Let $D^{(k)}[i,j]=weight$ of a shortest path from v_i to v_j using only vertices from $\{v_1,v_2,\ldots,v_k\}$ as intermediate vertices in the path.

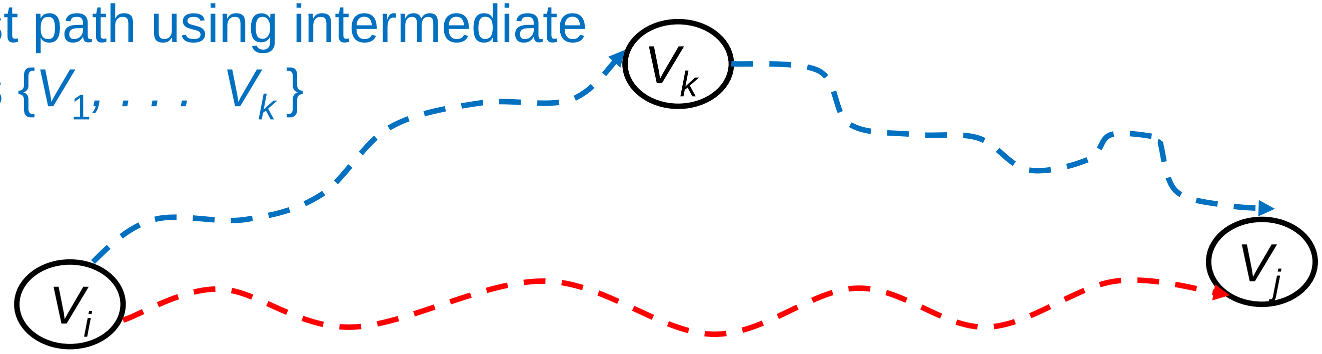
  - Obviously: $D^{(0)}=W$, we need $D^{(n)}$

  - How to compute $D^{(k)}$ from $D^{(k\text{-}1)}$?

ALL PAIRS SHORTEST PATHS

- Floyd's algorithm:
  - Case 1: A shortest path from v_i to v_j does not use v_k . Then $D^{(k)}[i,j] = D^{(k-1)}[i,j]$.
  - Case 2: A shortest path from v_i to v_j does use v_k . Then $D^{(k)}[i,j] = D^{(k-1)}[i,k] + D^{(k-1)}[k,j]$.

Shortest path using intermediate vertices $\{V_1, \ldots\ V_k\}$



Shortest Path using intermediate vertices $\{V_{1, \ldots}\ V_{k-1}\}$

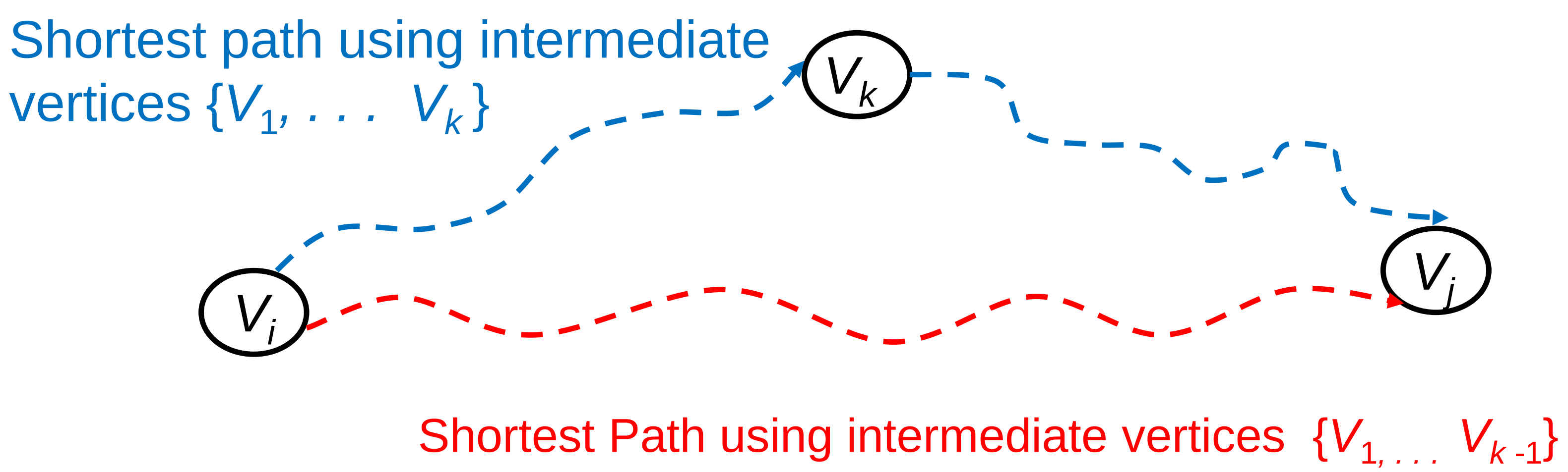
ALL PAIRS SHORTEST PATHS

- Floyd's algorithm:
  - Since

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

    $\boldsymbol{D}^{(k)}[i,j] = \boldsymbol{D}^{(k\text{-}1)}[i,k] + \boldsymbol{D}^{(k\text{-}1)}[k,j].$
  - We conclude:

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



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)
```

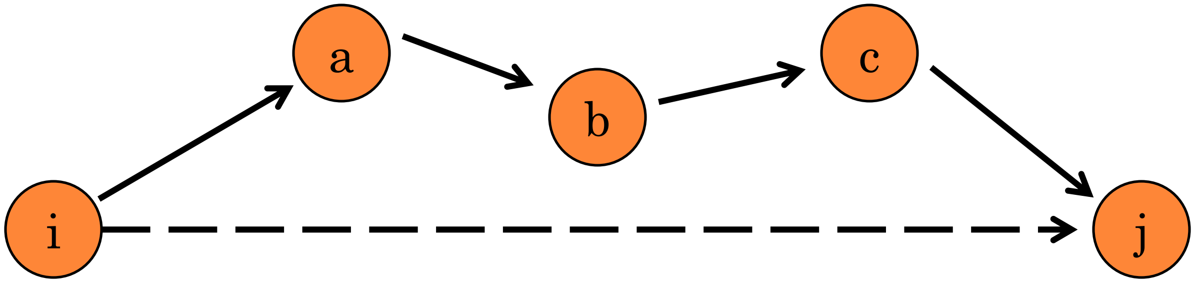
- Time complexity: $O(n^3)$

TRANSITIVE CLOSURE

- Given a directed graph $G=(V,E)$ find whether there is a path from v_i to v_j for all vertex pairs $v_i, v_j \in V$.

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

  $E^* = \{(i,j)$: there is a path from v_i to v_j in $G\}$



TRANSITIVE CLOSURE

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

- Solution 2 (Warshall's algorithm)
  - Define $t^{(k)}_{ij}$ such that

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

  - and for $k \geq 1$

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

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: ***$O(n^3)$***

ALGORITHM COMPARISON

Algorithm	Time complexity
Dijkstra�代s	O(nm log n)
Bellman-Ford‵s	$O(n^2 m)$
Matrix Multiplication	$O(n^4)$
Floyd‵s	$O(n^3)$
Warshall‵s (transitive closure)	$O(n^3)$

That's All for Today!