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

2/26

OUTLINE

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

3/26

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.

4/26

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: $O(n^2m)$

5/26

ALL PAIRS SHORTEST PATH

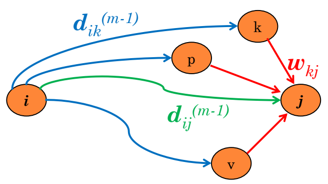
- Adjacency matrix representation
- $w: E \to \Re$ as n x n matrix W

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

6/26

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



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

MATRIX MULTIPLICATION

- Recursion.
  - 1. $d_{ij}^{(1)} = w_{ij}$
  - 2. $d_{ij}^{(m)} = \min(d_{ij}^{(m-1)}, \min_{1\le k\le n}\{d_{ik}^{(m-1)} + w_{kj}\})$
    $= \min_{1\le k\le n}\{d_{ik}^{(m-1)} + w_{kj}\}$
    (since $w_{jj}=0$ for all j)
- Equivalent matrix operations.
  - $C=A\cdot B$, $c_{ij}=\sum_{1\le k\le n} a_{ik}\cdot b_{kj}$
  - $d_{ij}^{(m)} \to c_{ij}$, $d_{ik}^{(m-1)} \to a_{ik}$, $w_{kj} \to b_{kj}$, $\min \to \sum$, $+ \to \cdot$
  - Compute series of matrices
    $D^{(1)}, D^{(2)}, \ldots, D^{(n-1)}$
    such that $D^{(m)} = D^{(m-1)} \cdot W$

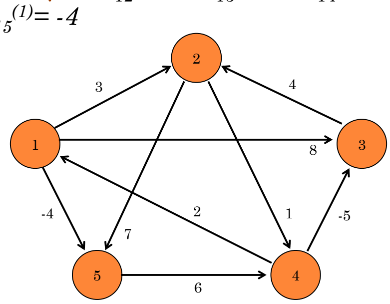
MATRIX MULTIPLICATION

- 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'
```

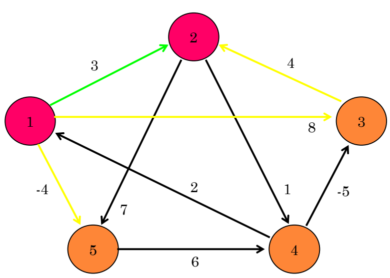

MATRIX MULTIPLICATION

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



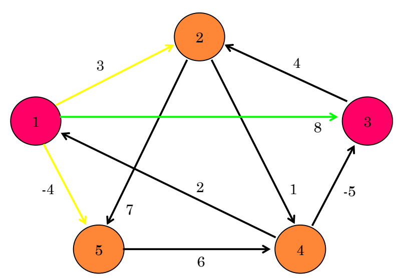
MATRIX MULTIPLICATION

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



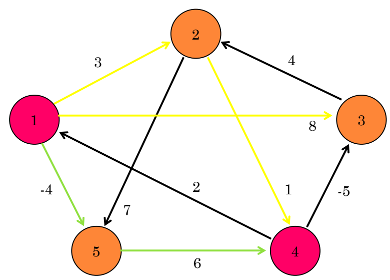
MATRIX MULTIPLICATION

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



MATRIX MULTIPLICATION

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



MATRIX MULTIPLICATION

- Example.

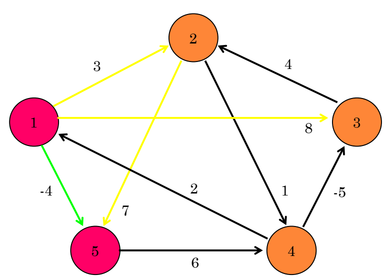
$$d_{14}^{(2)} = (0\ \ 3\ \ 8\ \ \infty\ -4) \bullet \begin{pmatrix} \infty \\ 1 \\ \infty \\ 0 \\ 6 \end{pmatrix}$$

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

$$= 2$$

MATRIX MULTIPLICATION

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



MATRIX MULTIPLICATION

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

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

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

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

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

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

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)
```

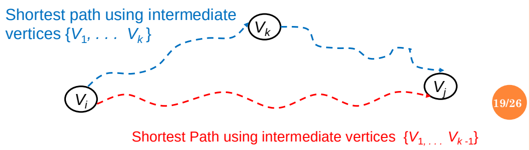
- Time complexity: ***O(n⁴)***

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 $D^{(k-1)}$?

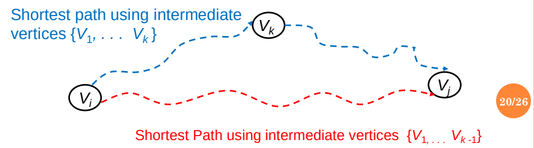
ALL PAIRS SHORTEST PATHS

- **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 $D^{(k)}[i,j] = D^{(k-1)}[i,k] + D^{(k-1)}[k,j]$.



ALL PAIRS SHORTEST PATHS

- **Floyd's algorithm:**
  - Since
    $D^{(k)}[i,j] = D^{(k-1)}[i,j]$ or
    $D^{(k)}[i,j] = D^{(k-1)}[i,k] + D^{(k-1)}[k,j]$.
  - We conclude:
    $D^{(k)}[i,j] = \min\{D^{(k-1)}[i,j],\ D^{(k-1)}[i,k] + D^{(k-1)}[k,j]\}$.



ALL PAIRS SHORTEST PATHS

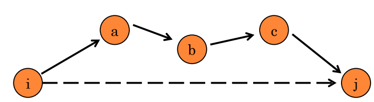
- **Floyd's algorithm – pseudo-code**

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

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'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!