Finding Shortest Paths between All Pairs of Points

stevenhalim@gmail.com
Outline

• Review: The **Single-Source** Shortest Paths Problem
• Introducing: The **All-Pairs** Shortest Paths Problem
  – With three motivating examples
• Floyd Warshall’s **Dynamic Programming** algorithm
  – The short code first 😊 (discussed in class)
  – The DP formulation (FYI, not discussed in class)
• Cool Floyd Warshall’s variants
The SSSP problem is about...

1. Finding the shortest path between a pair of vertices in the graph (source to destination)
2. Finding the shortest paths between any pair of vertices
3. Finding the shortest paths between one vertex to the other vertices in the graph
The four lines wonder

ALL-PAIRS SHORTEST PATHS
Motivating Problem 1

Diameter of a Graph

The diameter of a graph is defined as the greatest shortest path distance between any pair of vertices.

- For example, the diameter of this graph is 2:
  - The paths with length equal to diameter are:
    - 1-0-3 (or the reverse path)
    - 1-2-3 (or the reverse path)
    - 1-0-4 (or the reverse path)
    - 2-0-4 (or the reverse path)
    - 2-3-4 (or the reverse path)
What is the diameter of this graph?

1. 8, path = ____
2. 10, path = ____
3. 12, path = ____
4. I do not know 😞...
Motivating Problem 2
Analyzing the average number of clicks to browse the WWW

In year 2000, only 19 clicks are necessary to move from any page on the WWW to any other page :O

- That is, if the pages on the web are viewed as vertices in a graph, then the average path length between arbitrary pairs of vertices in the graph is 19
  - For example, the average path length between arbitrary pair of vertices in this graph below is:
    - 0→1 = 1; 0→2 = 1
    - 1→0 = 2; 1→2 = 1
    - 2→0 = 1; 2→1 = 2
    - Average = (1+1+2+1+1+2) / 6 = 8 / 6 = 1.333
What is the avg path length of this graph?

1. \( \frac{22}{10} = 2.200 \)
2. \( \frac{22}{12} = 1.833 \)
3. \( \frac{23}{12} = 1.917 \)
4. I do not know 😞 ...
Motivating Problem 3
Finding the best meeting point

Given a weighted graph that models a city and the travelling time between various places in that city, find the best meeting point for two persons who are currently in two different vertices (lots of such queries)

- For example, the best meeting point between two persons currently in $A = 0$ and $B = 3$ is at vertex 2
  - B just need 1 unit of time to walk from $3 \rightarrow 2$ and then wait for A
  - A needs 12 units of time to walk from $0 \rightarrow 2$
  - After 12 units of time, they meet 😊
What is the best meeting point for C and D?

1. Vertex 0, ____ units of time
2. Vertex 1, ____ units of time
3. Vertex 2, ____ units of time
4. Vertex 3, ____ units of time
5. Vertex 4, ____ units of time
6. I do not know 😞 ...
All-Pairs Shortest Paths (APSP)

Simple problem definition:

*Find the shortest paths between any pair of vertices in the given directed weighted graph*
APSP Solutions with SSSP Algorithms

Several solutions from what we have known earlier:

• On unweighted graph
  – Call BFS $V$ times, once from each vertex
    • Time complexity: $O(V \times (V+E)) = O(V^3)$ if $E = O(V^2)$

• On weighted graph, for simplicity, non (-ve) weighted graph
  – Call Bellman Ford’s $V$ times, once from each vertex
    • Time complexity: $O(V \times VE) = O(V^4)$ if $E = O(V^2)$
  – Call Dijkstra’s $V$ times, once from each vertex
    • Time complexity: $O(V \times (V+E) \times \log V) = O(V^3 \log V)$ if $E = O(V^2)$
APSP Solution: Floyd Warshall’s

Floyd Warshall’s uses an Adjacency Matrix: $D[|V|][|V|]$  
- Originally $D[i][j]$ contains the weight of $\text{edge}(i, j) \rightarrow O(1)$  
- After Floyd Warshall’s stop, it contains the weight of $\text{path}(i, j)$  
- It is usually a nice algorithm for the pre-processing part 😊

```java
for (int k = 0; k < V; k++) // remember, k first
    for (int i = 0; i < V; i++) // before i
        for (int j = 0; j < V; j++) // then j
            D[i][j] = Math.min(D[i][j], D[i][k]+D[k][j]);
```

It runs in $O(V^3)$ since we have three nested loops!

- PS: Apparently, if we only given a short amount of time, we can only solve the APSP problem for small graphs, as none of the APSP solution in this and last slides runs better than $O(V^3)$
Preprocessing + (Lots of) Queries

This is another problem solving technique

- Preprocess the data once (can be a costly operation)
- But then future queries can be (much) faster by working on the processed data

Example with the APSP problem:

- Once we have pre-processed the APSP information with $O(V^3)$ Floyd Warshall’s algorithm...
  - Future queries that asks “what is the shortest path weight between vertex i and j” can now be answered in $O(1)$...
This section is optional for this semester... :O
Please read it on your own...
You may revisit this DP explanation in future algorithm module

Jump to Floyd Warshall’s variants

EXPLANATION OF FLOYD WARSHALL’S ALGORITHM
Floyd Warshall’s – Basic Idea (1)

- Assume that the vertices are labeled as $[0 .. V - 1]$.
- Now let $sp(i, j, k)$ denotes the shortest path between vertex $i$ and vertex $j$ with the restriction that the vertices on the shortest path (excluding $i$ and $j$) can only consist of vertices from $[0 .. k]$
  - How Robert Floyd and Stephen Warshall managed to arrive at this formulation is beyond this lecture...
- Initially $k = -1$ (or to say, we only use direct edges only)
  - Then, iteratively add $k$ by one until $k = V - 1$
Suppose we want to know the shortest path between vertex 3 and 4, using any intermediate vertices from \( k = [0 .. 4] \), i.e.

\[
sp(3, 4, 4) = ?
\]
Floyd Warshall’s – Basic Idea (3)

We will monitor these two values

\[ k = -1 \]

The current content of Adjacency Matrix D at \( k = -1 \)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>( \infty )</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>( \infty )</td>
<td>0</td>
<td>( \infty )</td>
<td>4</td>
<td>( \infty )</td>
</tr>
<tr>
<td>2</td>
<td>( \infty )</td>
<td>1</td>
<td>0</td>
<td>( \infty )</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>( \infty )</td>
<td>3</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>0</td>
</tr>
</tbody>
</table>

Direct Edges Only

\[ i = 3, j = 4, k = -1 \]

sp(3, 2, -1) = 3  sp(2, 4, -1) = 1  sp(3, 4, -1) = 5
The current content of Adjacency Matrix $D$ at $k = 0$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>$\infty$</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>$\infty$</td>
<td>0</td>
<td>$\infty$</td>
<td>4</td>
<td>$\infty$</td>
</tr>
<tr>
<td>2</td>
<td>$\infty$</td>
<td>1</td>
<td>0</td>
<td>$\infty$</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>0</td>
</tr>
</tbody>
</table>

Floyd Warshall’s – Basic Idea (4)

- $i = 3$, $j = 4$, $k = 0$
- Vertex 0 allowed
- $sp(3, 2, 0) = 2$
- $sp(2, 4, 0) = 1$
- $sp(3, 4, 0) = 4$
Floyd Warshall’s – Basic Idea (4)

The current content of Adjacency Matrix D at \( k = 1 \)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>(\infty)</td>
<td>0</td>
<td>(\infty)</td>
<td>4</td>
<td>(\infty)</td>
</tr>
<tr>
<td>2</td>
<td>(\infty)</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>0</td>
</tr>
</tbody>
</table>

\( i = 3, j = 4, k = 1 \)

Vertex 0-1 allowed

\( \text{sp}(3, 2, 1) = 2 \quad \text{sp}(2, 4, 1) = 1 \quad \text{sp}(3, 4, 1) = 4 \)
Floyd Warshall’s – Basic Idea (5)

The current content of Adjacency Matrix D at \( k = 2 \)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>∞</td>
<td>0</td>
<td>∞</td>
<td>4</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>∞</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
</tr>
</tbody>
</table>

\( i = 3, j = 4, k = 2 \)
Vertex 0-2 allowed

sp(3, 2, 2) = 2  sp(2, 4, 2) = 1  sp(3, 4, 2) = 3
Floyd Warshall’s – Basic Idea (6)

The current content of Adjacency Matrix D at $k = 3$

<table>
<thead>
<tr>
<th>k = 3</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>0</td>
</tr>
</tbody>
</table>

i = 3, j = 4, k = 3
Vertex 0-3 allowed

$sp(3, 2, 2) = 2$  $sp(2, 4, 2) = 1$  $sp(3, 4, 2) = 3$
Floyd Warshall’s – Basic Idea (7)

The current content of Adjacency Matrix D at $k = 4$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>0</td>
</tr>
</tbody>
</table>

$i = 3, j = 4, k = 4$

Vertex 0-4 allowed

$sp(3, 2, 2) = 2$  $sp(2, 4, 2) = 1$  $sp(3, 4, 2) = 3$
Floyd Warshall’s – DP (1)
Recursive Solution / Optimal Sub-structure

\( D_{i,j}^{-1} \): Edge weight of the original graph

\( D_{i,j}^k \): Shortest distance from \( i \) to \( j \) involving \([0 .. k]\) only as intermediate vertices

\[
D_{i,j}^k = \begin{cases} 
  w_{i,j} & \text{for } k = -1 \\
  \min(D_{i,j}^{k-1}, D_{i,k}^{k-1} + D_{k,j}^{k-1}) & \text{for } k \geq 0
\end{cases}
\]

Not using vertex \( k \)  \hspace{1cm} Using vertex \( k \)
Floyd Warshall’s – DP (2)

Overlapping Sub problems

- Avoiding re-computation: To fill out an entry in the table \( k \), we make use of entries in table \( k - 1 \), row by row, left to right
  - The topological order is obtained via 3 nested loops: \( k \rightarrow i \rightarrow j \)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>(\infty)</td>
<td>0</td>
<td>(\infty)</td>
<td>4</td>
<td>(\infty)</td>
</tr>
<tr>
<td>2</td>
<td>(\infty)</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>(\infty)</td>
<td>0</td>
<td>(\infty)</td>
<td>4</td>
<td>(\infty)</td>
</tr>
<tr>
<td>2</td>
<td>(\infty)</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>0</td>
</tr>
</tbody>
</table>
Floyd Warshall’s – DP (3)
The Near Final Code

// the "memory unfriendly" version, O(V^3) space complexity
int[][][] D3 = new int[V+1][V][V]; // 3D matrix
for (k = 0; k <= V; k++) // initialization phase
    for (i = 0; i < V; i++) {
        Arrays.fill(D3[k][i], 1000000000); // cannot use Collections.nCopies
        D3[k][i][i] = 0;
    }

for (i = 0; i < E; i++) { // direct edges
    u = sc.nextInt(); v = sc.nextInt(); w = sc.nextInt();
    D3[0][u][v] = w; // directed weighted edge
}

// main loop, O(V^3): this three nested loops are the "topological order"
for (k = 0; k < V; k++) // be careful, put k first
    for (i = 0; i < V; i++) // before i
        for (j = 0; j < V; j++) // and then j
            D3[k+1][i][j] = Math.min(D3[k][i][j], // note, I shift index k by +1
                                      D3[k][i][k]+D3[k][k][j]);
Floyd Warshall’s – DP (4)

The Final Code, drop dimension ‘k’

```java
int[][] D = new int[V][V]; // 2D adjacency matrix
for (i = 0; i < V; i++) { // initialization phase
    Arrays.fill(D[i], 1000000000); // cannot use nCopies
    D[i][i] = 0;
}

for (i = 0; i < E; i++) { // direct edges
    u = sc.nextInt(); v = sc.nextInt(); w = sc.nextInt();
    D[u][v] = w; // directed weighted edge
}

// main loop, O(V^3): the "topological order"
for (k = 0; k < V; k++) // be careful, put k first
    for (i = 0; i < V; i++) // before i
        for (j = 0; j < V; j++) // and then j
            D[i][j] = Math.min(D[i][j], D[i][k]+D[k][j]);
```
VARIANTS OF FLOYD WARSHALL’S
Variant 1 – Print the Actual SP (1)

We have learned to use array/Vector p (predecessor/parent) to record the BFS/DFS/SP Spanning Tree

• But now, we are dealing with all-pairs of paths :O

Solution: Use predecessor matrix p

• let p be a 2D predecessor matrix, where p[i][j] is the last vertex before j on a shortest path from i to j, i.e. i -> ... -> p[i][j] -> j

• Initially, p[i][j] = i for all pairs of i and j

• If D[i][k] + D[k][j] < D[i][j], then D[i][j] = D[i][k] + D[k][j] and p[i][j] = p[k][j] ← this will be the last vertex before j in the shortest path
Variant 1 – Print the Actual SP (2)

The two matrices, $D$ and $p$

- The shortest path from $3 \rightarrow 4$
  - $3 \rightarrow 0 \rightarrow 2 \rightarrow 4$

$$
\begin{array}{cccccc}
D & 0 & 1 & 2 & 3 & 4 \\
0 & 0 & 2 & 1 & 6 & 2 \\
1 & 5 & 0 & 6 & 4 & 7 \\
2 & 6 & 1 & 0 & 5 & 1 \\
3 & 1 & 3 & 2 & 0 & 3 \\
4 & \infty & \infty & \infty & \infty & 0 \\
\end{array}
$$

$$
\begin{array}{cccccc}
p & 0 & 1 & 2 & 3 & 4 \\
0 & 0 & 0 & 0 & 1 & 2 \\
1 & 3 & 1 & 0 & 1 & 2 \\
2 & 3 & 2 & 2 & 1 & 2 \\
3 & 3 & 0 & 0 & 3 & 2 \\
4 & 4 & 4 & 4 & 4 & 4 \\
\end{array}
$$
Variant 2 – Transitive Closure (1)

Stephen Warshall actually invented this algorithm for solving the transitive closure problem

- Given a graph, determine if vertex \( i \) is connected to vertex \( j \) either directly (via an edge) or indirectly (via a path)

Solution: Modify the matrix \( D \) to contain only 0/1

- In the main loop of Warshall’s algorithm:

```c
// Initially: \( D[i][i] = 0 \)
// \( D[i][j] = 1 \) if edge\((i, j)\) exist; 0 otherwise
// the three nested loops as per normal
D[i][j] = D[i][j] | (D[i][k] & D[k][j]); // faster
```
Variant 2 – Transitive Closure (2)

The matrix $D$, before and after

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The minimax problem is a problem of finding the minimum of maximum edge weight along all possible paths from vertex \( i \) to vertex \( j \) (maximin is the reverse)

- For a single path from \( i \) to \( j \), we pick the maximum edge weight along this path
- Then, for all possible paths from \( i \) to \( j \), we pick the one with the minimum max-edge-weight

Solution: Again, a modification of Floyd Warshall’s

```java
// Initially: D[i][i] = 0
// D[i][j] = weight of edge(i, j) exist; INF otherwise
// the three nested loops as per normal
D[i][j] = Math.min(D[i][j], Math.max(D[i][k], D[k][j]));
```
The minimax path from 1 to 4 is 4, via edge (1, 3)
• 1→3→0→2→4

<table>
<thead>
<tr>
<th>D,init</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>∞</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>∞</td>
<td>0</td>
<td>∞</td>
<td>4</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td>∞</td>
<td>1</td>
<td>0</td>
<td>∞</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>∞</td>
<td>3</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D,final</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
</tr>
</tbody>
</table>
Variant 4 – Detecting Any/-ve Cycle

1. Set the main diagonal of $D$ to $\infty$
2. Run Floyd Warshall’s
3. Recheck the main diagonal

![Graph with nodes 0, 1, 2, 3, 4 and edges with weights]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\infty$</td>
<td>2</td>
<td>1</td>
<td>$\infty$</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>4</td>
<td>$\infty$</td>
</tr>
<tr>
<td>2</td>
<td>$\infty$</td>
<td>1</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>$\infty$</td>
<td>3</td>
<td>$\infty$</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>1</td>
<td>7</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>
Java Implementations

See FloydWarshallDemo.java for more details

• These four variants are listed inside that demo code
Summary

In this lecture, we have seen:

• Introduction to the APSP problem (3 motivating examples)
• Introduction to the Floyd Warshall’s DP algorithm
  – But many just view this algorithm as “another graph algorithm”
• Introduction to 4 variants of Floyd Warshall’s
• Simple Java implementations of Floyd Warshall’s variants
Extra Stuffs

Steven estimates that this lecture will end quite fast.

So he will use the remaining time to discuss questions from past exam papers.