

# All-Pairs Shortest Paths

Madhavan Mukund

<https://www.cmi.ac.in/~madhavan>

Programming, Data Structures and Algorithms using Python

Week 5

# Shortest paths in weighted graphs

Two types of shortest path problems of interest

## Single source shortest paths

- Find shortest paths from a fixed vertex to every other vertex
- Transport finished product from factory (single source) to all retail outlets
- Courier company delivers items from distribution centre (single source) to addressees

## All pairs shortest paths

- Find shortest paths between every pair of vertices  $i$  and  $j$
- Optimal airline, railway, road routes between cities

# Shortest paths in weighted graphs

Two types of shortest path problems of interest

## Single source shortest paths

- Find shortest paths from a fixed vertex to every other vertex
- Transport finished product from factory (single source) to all retail outlets
- Courier company delivers items from distribution centre (single source) to addressees
- Dijkstra's algorithm (non-negative weights), Bellman-Ford algorithm (allows negative weights)

## All pairs shortest paths

- Find shortest paths between every pair of vertices  $i$  and  $j$
- Optimal airline, railway, road routes between cities

# Shortest paths in weighted graphs

Two types of shortest path problems of interest

## Single source shortest paths

- Find shortest paths from a fixed vertex to every other vertex
- Transport finished product from factory (single source) to all retail outlets
- Courier company delivers items from distribution centre (single source) to addressees
- Dijkstra's algorithm (non-negative weights), Bellman-Ford algorithm (allows negative weights)

## All pairs shortest paths

- Find shortest paths between every pair of vertices  $i$  and  $j$
- Optimal airline, railway, road routes between cities
- Run Dijkstra or Bellman-Ford from each vertex

# Shortest paths in weighted graphs

Two types of shortest path problems of interest

## Single source shortest paths

- Find shortest paths from a fixed vertex to every other vertex
- Transport finished product from factory (single source) to all retail outlets
- Courier company delivers items from distribution centre (single source) to addressees
- Dijkstra's algorithm (non-negative weights), Bellman-Ford algorithm (allows negative weights)

## All pairs shortest paths

- Find shortest paths between every pair of vertices  $i$  and  $j$
- Optimal airline, railway, road routes between cities
- Run Dijkstra or Bellman-Ford from each vertex
- Is there is another way?

# Transitive closure

- Adjacency matrix  $A$  represents paths of length 1
- Matrix multiplication,  $A^2 = A \times A$ 
  - $A^2[i, j] = 1$  if there is a path of length 2 from  $i$  to  $j$
  - For some  $k$ ,  $A[i, k] = A[k, j] = 1$
- In general,  $A^{\ell+1} = A^\ell \times A$ ,
  - $A^{\ell+1}[i, j] = 1$  if there is a path of length  $\ell+1$  from  $i$  to  $j$
  - For some  $k$ ,  $A^\ell[i, k] = 1$ ,  $A[k, j] = 1$
- $A^+ = A + A^2 + \dots + A^{n-1}$

# Transitive closure

- Adjacency matrix  $A$  represents paths of length 1
- Matrix multiplication,  $A^2 = A \times A$ 
  - $A^2[i, j] = 1$  if there is a path of length 2 from  $i$  to  $j$
  - For some  $k$ ,  $A[i, k] = A[k, j] = 1$
- In general,  $A^{\ell+1} = A^\ell \times A$ ,
  - $A^{\ell+1}[i, j] = 1$  if there is a path of length  $\ell+1$  from  $i$  to  $j$
  - For some  $k$ ,  $A^\ell[i, k] = 1$ ,  $A[k, j] = 1$
- $A^+ = A + A^2 + \dots + A^{n-1}$

An alternative approach

# Transitive closure

- Adjacency matrix  $A$  represents paths of length 1
- Matrix multiplication,  $A^2 = A \times A$ 
  - $A^2[i, j] = 1$  if there is a path of length 2 from  $i$  to  $j$
  - For some  $k$ ,  $A[i, k] = A[k, j] = 1$
- In general,  $A^{\ell+1} = A^\ell \times A$ ,
  - $A^{\ell+1}[i, j] = 1$  if there is a path of length  $\ell+1$  from  $i$  to  $j$
  - For some  $k$ ,  $A^\ell[i, k] = 1$ ,  $A[k, j] = 1$
- $A^+ = A + A^2 + \dots + A^{n-1}$

## An alternative approach

- $B^k[i, j] = 1$  if there is path from  $i$  to  $j$  via vertices  $\{0, 1, \dots, k-1\}$ 
  - Constraint applies only to intermediate vertices between  $i$  and  $j$
  - $B^0[i, j] = 1$  if there is a direct edge
  - $B^0 = A$



# Transitive closure

- Adjacency matrix  $A$  represents paths of length 1
- Matrix multiplication,  $A^2 = A \times A$ 
  - $A^2[i, j] = 1$  if there is a path of length 2 from  $i$  to  $j$
  - For some  $k$ ,  $A[i, k] = A[k, j] = 1$
- In general,  $A^{\ell+1} = A^\ell \times A$ ,
  - $A^{\ell+1}[i, j] = 1$  if there is a path of length  $\ell+1$  from  $i$  to  $j$
  - For some  $k$ ,  $A^\ell[i, k] = 1$ ,  $A[k, j] = 1$
- $A^+ = A + A^2 + \dots + A^{n-1}$

## An alternative approach

- $B^k[i, j] = 1$  if there is path from  $i$  to  $j$  via vertices  $\{0, 1, \dots, k-1\}$ 
  - Constraint applies only to intermediate vertices between  $i$  and  $j$
  - $B^0[i, j] = 1$  if there is a direct edge
  - $B^0 = A$
- $B^{k+1}[i, j] = 1$  if
  - $B^k[i, j] = 1$  — can already reach  $j$  from  $i$  via  $\{0, 1, \dots, k-1\}$
  - $B^k[i, k] = 1$  and  $B^k[k, j] = 1$  — use  $\{0, 1, \dots, k-1\}$  to go from  $i$  to  $k$  and then from  $k$  to  $j$

# Warshall's Algorithm

- $B^k[i, j] = 1$  if there is path from  $i$  to  $j$  via vertices  $\{0, 1, \dots, k-1\}$
- $B^0[i, j] = A[i, j]$ 
  - Direct edges, no intermediate vertices
- $B^{k+1}[i, j] = 1$  if
  - $B^k[i, j] = 1$ , or
  - $B^k[i, k] = 1$  and  $B^k[k, j] = 1$
- The algorithm on the left also computes transitive closure — **Warshall's algorithm**

# Warshall's Algorithm

- $B^k[i, j] = 1$  if there is path from  $i$  to  $j$  via vertices  $\{0, 1, \dots, k-1\}$
- $B^0[i, j] = A[i, j]$ 
  - Direct edges, no intermediate vertices
- $B^{k+1}[i, j] = 1$  if
  - $B^k[i, j] = 1$ , or
  - $B^k[i, k] = 1$  and  $B^k[k, j] = 1$
- The algorithm on the left also computes transitive closure — **Warshall's algorithm**
- $B^n[i, j] = 1$  if there is some path from  $i$  to  $j$  with intermediate vertices in  $\{0, 1, \dots, n-1\}$

# Warshall's Algorithm

- $B^k[i, j] = 1$  if there is path from  $i$  to  $j$  via vertices  $\{0, 1, \dots, k-1\}$
- $B^0[i, j] = A[i, j]$ 
  - Direct edges, no intermediate vertices
- $B^{k+1}[i, j] = 1$  if
  - $B^k[i, j] = 1$ , or
  - $B^k[i, k] = 1$  and  $B^k[k, j] = 1$
- The algorithm on the left also computes transitive closure — **Warshall's algorithm**
- $B^n[i, j] = 1$  if there is some path from  $i$  to  $j$  with intermediate vertices in  $\{0, 1, \dots, n-1\}$
- $B^n = A^+$

# Warshall's Algorithm

- $B^k[i, j] = 1$  if there is path from  $i$  to  $j$  via vertices  $\{0, 1, \dots, k-1\}$
- $B^0[i, j] = A[i, j]$ 
  - Direct edges, no intermediate vertices
- $B^{k+1}[i, j] = 1$  if
  - $B^k[i, j] = 1$ , or
  - $B^k[i, k] = 1$  and  $B^k[k, j] = 1$
- The algorithm on the left also computes transitive closure — **Warshall's algorithm**
- $B^n[i, j] = 1$  if there is some path from  $i$  to  $j$  with intermediate vertices in  $\{0, 1, \dots, n-1\}$
- $B^n = A^+$
- We adapt Warshall's algorithm to compute all-pairs shortest paths

# Floyd-Warshall Algorithm

- Let  $SP^k[i, j]$  be the length of the shortest path from  $i$  to  $j$  via vertices  $\{0, 1, \dots, k-1\}$
- $SP^0[i, j] = W[i, j]$ 
  - No intermediate vertices, shortest path is weight of direct edge
  - Assume  $W[i, j] = \infty$  if  $(i, j) \notin E$

# Floyd-Warshall Algorithm

- Let  $SP^k[i, j]$  be the length of the shortest path from  $i$  to  $j$  via vertices  $\{0, 1, \dots, k-1\}$
- $SP^0[i, j] = W[i, j]$ 
  - No intermediate vertices, shortest path is weight of direct edge
  - Assume  $W[i, j] = \infty$  if  $(i, j) \notin E$
- $SP^{k+1}[i, j]$  is the minimum of
  - $SP^k[i, j]$   
Shortest path using only  $\{0, 1, \dots, k-1\}$
  - $SP^k[i, k] + SP^k[k, j]$   
Combine shortest path from  $i$  to  $k$  and  $k$  to  $j$

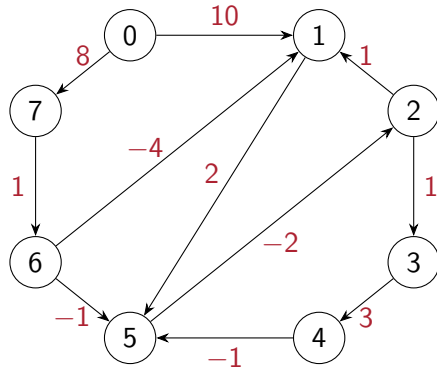
# Floyd-Warshall Algorithm

- Let  $SP^k[i, j]$  be the length of the shortest path from  $i$  to  $j$  via vertices  $\{0, 1, \dots, k-1\}$
- $SP^0[i, j] = W[i, j]$ 
  - No intermediate vertices, shortest path is weight of direct edge
  - Assume  $W[i, j] = \infty$  if  $(i, j) \notin E$
- $SP^{k+1}[i, j]$  is the minimum of
  - $SP^k[i, j]$   
Shortest path using only  $\{0, 1, \dots, k-1\}$
  - $SP^k[i, k] + SP^k[k, j]$   
Combine shortest path from  $i$  to  $k$  and  $k$  to  $j$
- $SP^n[i, j] = 1$  is the length of the shortest path overall from  $i$  to  $j$ 
  - Intermediate vertices lie in  $\{0, 1, \dots, n-1\}$



# Floyd-Warshall Algorithm

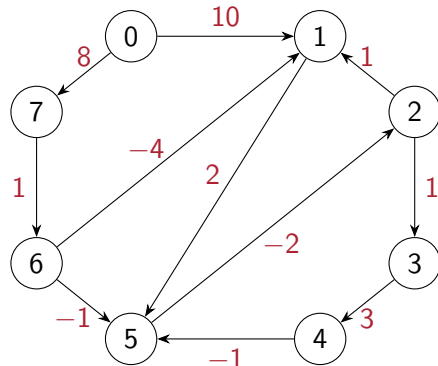
$SP^0$	0	1	2	3	4	5	6	7
0	$\infty$	10	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	8
1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	2	$\infty$	$\infty$
2	$\infty$	1	$\infty$	1	$\infty$	$\infty$	$\infty$	$\infty$
3	$\infty$	$\infty$	$\infty$	$\infty$	3	$\infty$	$\infty$	$\infty$
4	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	-1	$\infty$	$\infty$
5	$\infty$	$\infty$	-2	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
6	$\infty$	-4	$\infty$	$\infty$	$\infty$	-1	$\infty$	$\infty$
7	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	1	$\infty$



# Floyd-Warshall Algorithm

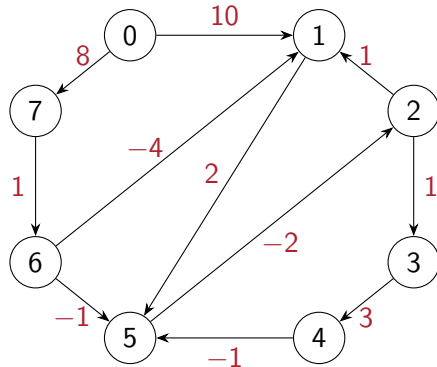
$SP^0$	0	1	2	3	4	5	6	7
0	$\infty$	10	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	8
1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	2	$\infty$	$\infty$
2	$\infty$	1	$\infty$	1	$\infty$	$\infty$	$\infty$	$\infty$
3	$\infty$	$\infty$	$\infty$	$\infty$	3	$\infty$	$\infty$	$\infty$
4	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	-1	$\infty$	$\infty$
5	$\infty$	$\infty$	-2	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
6	$\infty$	-4	$\infty$	$\infty$	$\infty$	-1	$\infty$	$\infty$
7	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	1	$\infty$

$SP^1$	0	1	2	3	4	5	6	7
0	$\infty$	10	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	8
1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	2	$\infty$	$\infty$
2	$\infty$	1	$\infty$	1	$\infty$	$\infty$	$\infty$	$\infty$
3	$\infty$	$\infty$	$\infty$	$\infty$	3	$\infty$	$\infty$	$\infty$
4	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	-1	$\infty$	$\infty$
5	$\infty$	$\infty$	-2	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
6	$\infty$	-4	$\infty$	$\infty$	$\infty$	-1	$\infty$	$\infty$
7	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	1	$\infty$



# Floyd-Warshall Algorithm

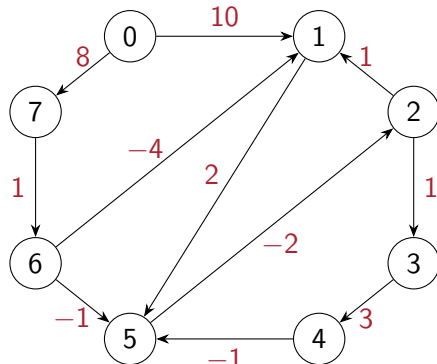
$SP^1$	0	1	2	3	4	5	6	7
0	$\infty$	10	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	8
1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	2	$\infty$	$\infty$
2	$\infty$	1	$\infty$	1	$\infty$	$\infty$	$\infty$	$\infty$
3	$\infty$	$\infty$	$\infty$	$\infty$	3	$\infty$	$\infty$	$\infty$
4	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	-1	$\infty$	$\infty$
5	$\infty$	$\infty$	-2	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
6	$\infty$	-4	$\infty$	$\infty$	$\infty$	-1	$\infty$	$\infty$
7	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	1	$\infty$



# Floyd-Warshall Algorithm

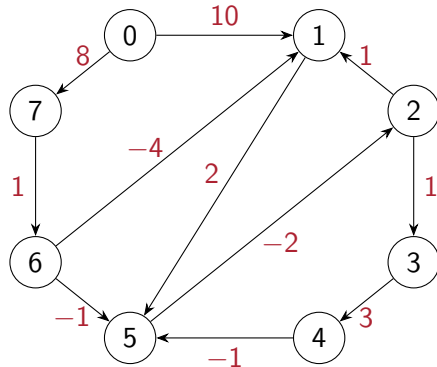
$SP^1$	0	1	2	3	4	5	6	7
0	$\infty$	10	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	8
1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	2	$\infty$	$\infty$
2	$\infty$	1	$\infty$	1	$\infty$	$\infty$	$\infty$	$\infty$
3	$\infty$	$\infty$	$\infty$	$\infty$	3	$\infty$	$\infty$	$\infty$
4	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	-1	$\infty$	$\infty$
5	$\infty$	$\infty$	-2	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
6	$\infty$	-4	$\infty$	$\infty$	$\infty$	-1	$\infty$	$\infty$
7	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	1	$\infty$

$SP^2$	0	1	2	3	4	5	6	7
0	$\infty$	10	$\infty$	$\infty$	$\infty$	12	$\infty$	8
1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	2	$\infty$	$\infty$
2	$\infty$	1	$\infty$	1	$\infty$	3	$\infty$	$\infty$
3	$\infty$	$\infty$	$\infty$	$\infty$	3	$\infty$	$\infty$	$\infty$
4	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	-1	$\infty$	$\infty$
5	$\infty$	$\infty$	-2	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
6	$\infty$	-4	$\infty$	$\infty$	$\infty$	-2	$\infty$	$\infty$
7	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	1	$\infty$



# Floyd-Warshall Algorithm

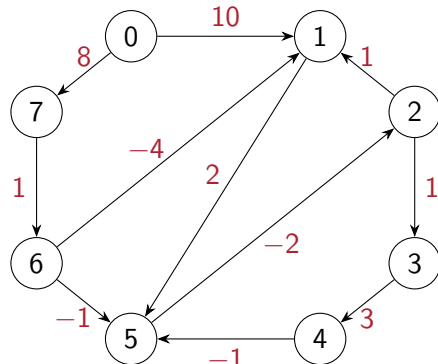
$SP^2$	0	1	2	3	4	5	6	7
0	$\infty$	10	$\infty$	$\infty$	$\infty$	12	$\infty$	8
1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	2	$\infty$	$\infty$
2	$\infty$	1	$\infty$	1	$\infty$	3	$\infty$	$\infty$
3	$\infty$	$\infty$	$\infty$	$\infty$	3	$\infty$	$\infty$	$\infty$
4	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	-1	$\infty$	$\infty$
5	$\infty$	$\infty$	-2	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
6	$\infty$	-4	$\infty$	$\infty$	$\infty$	-2	$\infty$	$\infty$
7	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	1	$\infty$



# Floyd-Warshall Algorithm

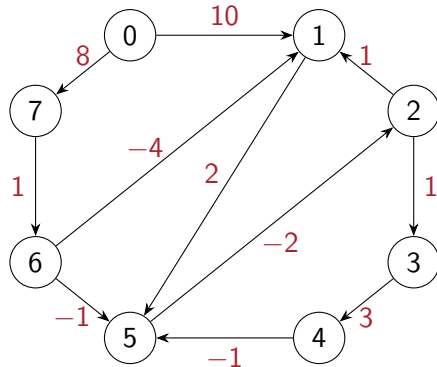
$SP^2$	0	1	2	3	4	5	6	7
0	$\infty$	10	$\infty$	$\infty$	$\infty$	12	$\infty$	8
1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	2	$\infty$	$\infty$
2	$\infty$	1	$\infty$	1	$\infty$	3	$\infty$	$\infty$
3	$\infty$	$\infty$	$\infty$	$\infty$	3	$\infty$	$\infty$	$\infty$
4	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	-1	$\infty$	$\infty$
5	$\infty$	$\infty$	-2	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
6	$\infty$	-4	$\infty$	$\infty$	$\infty$	-2	$\infty$	$\infty$
7	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	1	$\infty$

$SP^3$	0	1	2	3	4	5	6	7
0	$\infty$	10	$\infty$	$\infty$	$\infty$	12	$\infty$	8
1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	2	$\infty$	$\infty$
2	$\infty$	1	$\infty$	1	$\infty$	3	$\infty$	$\infty$
3	$\infty$	$\infty$	$\infty$	$\infty$	3	$\infty$	$\infty$	$\infty$
4	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	-1	$\infty$	$\infty$
5	$\infty$	-1	-2	-1	$\infty$	1	$\infty$	$\infty$
6	$\infty$	-4	$\infty$	$\infty$	$\infty$	-2	$\infty$	$\infty$
7	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	1	$\infty$



# Floyd-Warshall Algorithm

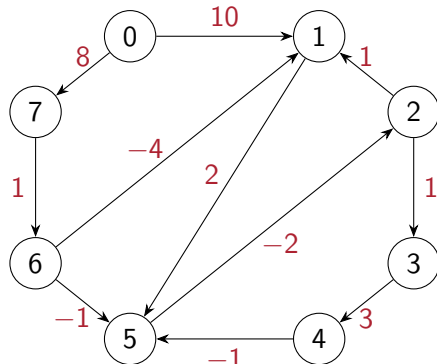
$SP^7$	0	1	2	3	4	5	6	7
0	$\infty$	10	10	11	14	12	$\infty$	8
1	$\infty$	1	0	1	4	2	$\infty$	$\infty$
2	$\infty$	1	1	1	4	3	$\infty$	$\infty$
3	$\infty$	1	0	1	3	2	$\infty$	$\infty$
4	$\infty$	-2	-3	-2	1	-1	$\infty$	$\infty$
5	$\infty$	-1	-2	-1	2	1	$\infty$	$\infty$
6	$\infty$	-4	-4	-3	0	-2	$\infty$	$\infty$
7	$\infty$	-3	-3	-2	1	-1	1	$\infty$



# Floyd-Warshall Algorithm

$SP^7$	0	1	2	3	4	5	6	7
0	$\infty$	10	10	11	14	12	$\infty$	8
1	$\infty$	1	0	1	4	2	$\infty$	$\infty$
2	$\infty$	1	1	1	4	3	$\infty$	$\infty$
3	$\infty$	1	0	1	3	2	$\infty$	$\infty$
4	$\infty$	-2	-3	-2	1	-1	$\infty$	$\infty$
5	$\infty$	-1	-2	-1	2	1	$\infty$	$\infty$
6	$\infty$	-4	-4	-3	0	-2	$\infty$	$\infty$
7	$\infty$	-3	-3	-2	1	-1	1	$\infty$

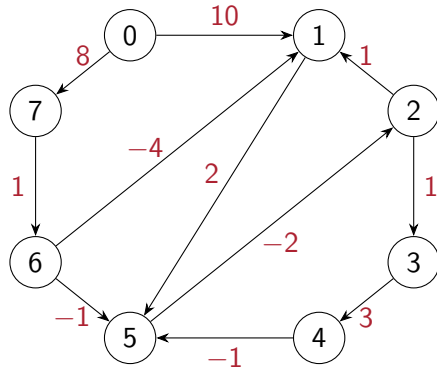
$SP^8$	0	1	2	3	4	5	6	7
0	$\infty$	5	5	6	9	7	9	8
1	$\infty$	1	0	1	4	2	$\infty$	$\infty$
2	$\infty$	1	1	1	4	3	$\infty$	$\infty$
3	$\infty$	1	0	1	3	2	$\infty$	$\infty$
4	$\infty$	-2	-3	-2	1	-1	$\infty$	$\infty$
5	$\infty$	-1	-2	-1	2	1	$\infty$	$\infty$
6	$\infty$	-4	-4	-3	0	-2	$\infty$	$\infty$
7	$\infty$	-3	-3	-2	1	-1	1	$\infty$





# Floyd-Warshall Algorithm

$SP^8$	0	1	2	3	4	5	6	7
0	$\infty$	5	5	6	9	7	9	8
1	$\infty$	1	0	1	4	2	$\infty$	$\infty$
2	$\infty$	1	1	1	4	3	$\infty$	$\infty$
3	$\infty$	1	0	1	3	2	$\infty$	$\infty$
4	$\infty$	-2	-3	-2	1	-1	$\infty$	$\infty$
5	$\infty$	-1	-2	-1	2	1	$\infty$	$\infty$
6	$\infty$	-4	-4	-3	0	-2	$\infty$	$\infty$
7	$\infty$	-3	-3	-2	1	-1	1	$\infty$



# Implementation

- Shortest path matrix  $SP$  is  $n \times n \times (n + 1)$

```
def floydwarshall(WMat):  
    (rows,cols,x) = WMat.shape  
    infinity = np.max(WMat)*rows*rows+1  
    SP = np.zeros(shape=(rows,cols,cols+1))  
    for i in range(rows):  
        for j in range(cols):  
            SP[i,j,0] = infinity  
  
    for i in range(rows):  
        for j in range(cols):  
            if WMat[i,j,0] == 1:  
                SP[i,j,0] = WMat[i,j,1]  
  
    for k in range(1,cols+1):  
        for i in range(rows):  
            for j in range(cols):  
                SP[i,j,k] = min(SP[i,j,k-1],  
                                SP[i,k-1,k-1]+SP[k-1,j,k-1])  
    return(SP[:, :, cols])
```

# Implementation

- Shortest path matrix  $SP$  is  $n \times n \times (n + 1)$
- Initialize  $SP[i, j, 0]$  to edge weight  $W(i, j)$ , or  $\infty$  if no edge

```
def floydwarshall(WMat):  
    (rows, cols, x) = WMat.shape  
    infinity = np.max(WMat)*rows*rows+1  
    SP = np.zeros(shape=(rows, cols, cols+1))  
    for i in range(rows):  
        for j in range(cols):  
            SP[i, j, 0] = infinity  
  
    for i in range(rows):  
        for j in range(cols):  
            if WMat[i, j, 0] == 1:  
                SP[i, j, 0] = WMat[i, j, 1]  
  
    for k in range(1, cols+1):  
        for i in range(rows):  
            for j in range(cols):  
                SP[i, j, k] = min(SP[i, j, k-1],  
                                   SP[i, k-1, k-1]+SP[k-1, j, k-1])  
    return(SP[:, :, cols])
```

# Implementation

- Shortest path matrix  $SP$  is  $n \times n \times (n + 1)$
- Initialize  $SP[i, j, 0]$  to edge weight  $W(i, j)$ , or  $\infty$  if no edge
- Update  $SP[i, j, k]$  from  $SP[i, j, k - 1]$  using the Floyd-Warshall update rule

```
def floydwarshall(WMat):  
    (rows, cols, x) = WMat.shape  
    infinity = np.max(WMat)*rows*rows+1  
    SP = np.zeros(shape=(rows, cols, cols+1))  
    for i in range(rows):  
        for j in range(cols):  
            SP[i, j, 0] = infinity  
  
    for i in range(rows):  
        for j in range(cols):  
            if WMat[i, j, 0] == 1:  
                SP[i, j, 0] = WMat[i, j, 1]  
  
    for k in range(1, cols+1):  
        for i in range(rows):  
            for j in range(cols):  
                SP[i, j, k] = min(SP[i, j, k-1],  
                                   SP[i, k-1, k-1]+SP[k-1, j, k-1])  
  
    return(SP[:, :, cols])
```

# Implementation

- Shortest path matrix  $SP$  is  $n \times n \times (n + 1)$
- Initialize  $SP[i, j, 0]$  to edge weight  $W(i, j)$ , or  $\infty$  if no edge
- Update  $SP[i, j, k]$  from  $SP[i, j, k - 1]$  using the Floyd-Warshall update rule
- Time complexity is  $O(n^3)$

```
def floydwarshall(WMat):  
    (rows,cols,x) = WMat.shape  
    infinity = np.max(WMat)*rows*rows+1  
    SP = np.zeros(shape=(rows,cols,cols+1))  
    for i in range(rows):  
        for j in range(cols):  
            SP[i,j,0] = infinity  
  
    for i in range(rows):  
        for j in range(cols):  
            if WMat[i,j,0] == 1:  
                SP[i,j,0] = WMat[i,j,1]  
  
    for k in range(1,cols+1):  
        for i in range(rows):  
            for j in range(cols):  
                SP[i,j,k] = min(SP[i,j,k-1],  
                                SP[i,k-1,k-1]+SP[k-1,j,k-1])  
  
    return(SP[:, :, cols])
```

# Implementation

- Shortest path matrix  $SP$  is  $n \times n \times (n + 1)$
- Initialize  $SP[i, j, 0]$  to edge weight  $W(i, j)$ , or  $\infty$  if no edge
- Update  $SP[i, j, k]$  from  $SP[i, j, k - 1]$  using the Floyd-Warshall update rule
- Time complexity is  $O(n^3)$
- We only need  $SP[i, j, k - 1]$  to compute  $SP[i, j, k]$

```
def floydwarshall(WMat):  
    (rows, cols, x) = WMat.shape  
    infinity = np.max(WMat)*rows*rows+1  
    SP = np.zeros(shape=(rows, cols, cols+1))  
    for i in range(rows):  
        for j in range(cols):  
            SP[i, j, 0] = infinity  
  
    for i in range(rows):  
        for j in range(cols):  
            if WMat[i, j, 0] == 1:  
                SP[i, j, 0] = WMat[i, j, 1]  
  
    for k in range(1, cols+1):  
        for i in range(rows):  
            for j in range(cols):  
                SP[i, j, k] = min(SP[i, j, k-1],  
                                   SP[i, k-1, k-1]+SP[k-1, j, k-1])  
  
    return(SP[:, :, cols])
```

# Implementation

- Shortest path matrix  $SP$  is  $n \times n \times (n + 1)$
- Initialize  $SP[i, j, 0]$  to edge weight  $W(i, j)$ , or  $\infty$  if no edge
- Update  $SP[i, j, k]$  from  $SP[i, j, k - 1]$  using the Floyd-Warshall update rule
- Time complexity is  $O(n^3)$
- We only need  $SP[i, j, k - 1]$  to compute  $SP[i, j, k]$
- Maintain two “slices”  $SP[i, j]$ ,  $SP'[i, j]$ , compute  $SP'$  from  $SP$ , copy  $SP'$  to  $SP$ , save space

```
def floydwarshall(WMat):  
    (rows, cols, x) = WMat.shape  
    infinity = np.max(WMat)*rows*rows+1  
    SP = np.zeros(shape=(rows, cols, cols+1))  
    for i in range(rows):  
        for j in range(cols):  
            SP[i, j, 0] = infinity  
  
    for i in range(rows):  
        for j in range(cols):  
            if WMat[i, j, 0] == 1:  
                SP[i, j, 0] = WMat[i, j, 1]  
  
    for k in range(1, cols+1):  
        for i in range(rows):  
            for j in range(cols):  
                SP[i, j, k] = min(SP[i, j, k-1],  
                                   SP[i, k-1, k-1]+SP[k-1, j, k-1])  
    return(SP[:, :, cols])
```

# Summary

- Warshall's algorithm is an alternative way to compute transitive closure
  - $B^k[i, j] = 1$  if we can reach  $j$  from  $i$  using vertices in  $\{0, 1, \dots, k-1\}$
- Adapt Warshall's algorithm to compute all pairs shortest paths
  - $SP^k[i, j]$  is the length of the shortest path from  $i$  to  $j$  using vertices in  $\{0, 1, \dots, k-1\}$
  - $SP^n[i, j]$  is the length of the overall shortest path
  - Floyd-Warshall algorithm
- Works with negative edge weights, assuming no negative cycles
- Simple nested loop implementation, time  $O(n^3)$
- Space can be limited to  $O(n^2)$  by reusing two “slices”  $SP$  and  $SP'$