

## All-Pairs Shortest Paths

**Input:** A directed graph  $G = (V, E)$  where each edge  $(v_i, v_j)$  has a weight  $w(i, j)$ .

**Output:** A “shortest” path from  $u$  to  $v$ , for all  $u, v \in V$ .

**Weight of path:** Given a path  $p = \langle v_1, \dots, v_k \rangle$ , its weight is:

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1}) \quad (1)$$

**“shortest” path** = path of minimum weight. We use  $\sigma(u, v)$  to denote this minimum weight.

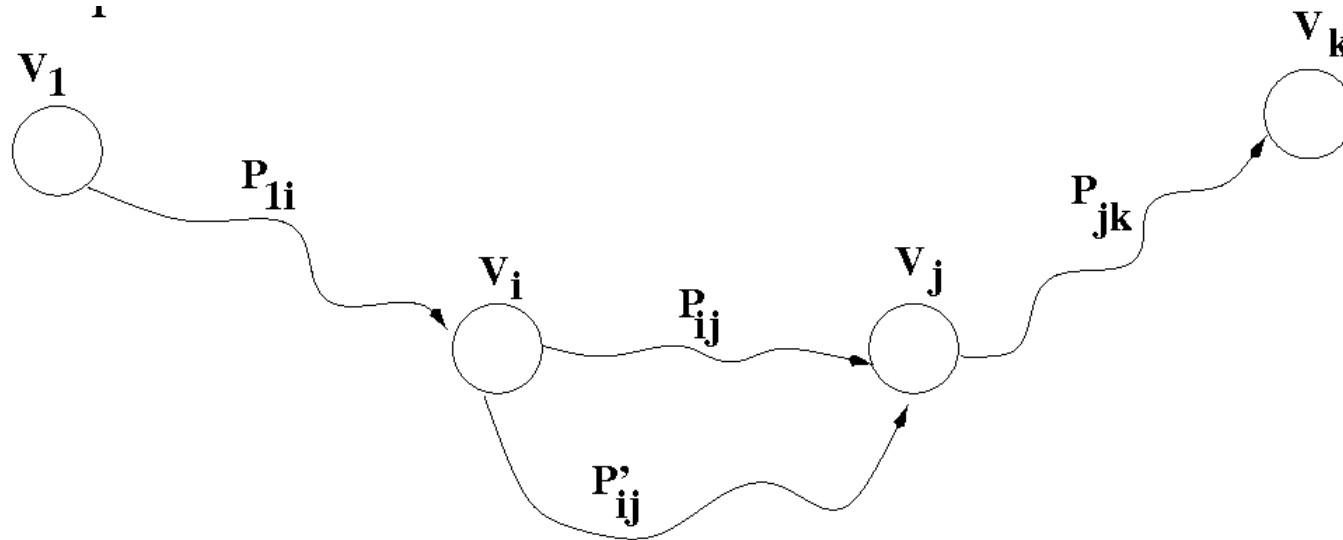
## Different variants of shortest path problems

- **Single-pair shortest path (SPSP):**  
Find a shortest path from  $u$  to  $v$ .
- **Single-source shortest paths (SSSP):**  
Find a shortest path from source  $s$  to all vertices  $v \in V$ .
  - solved with a Greedy algorithm (Dijkstra's).
- **All-pairs shortest paths (APSP):**  
Find a shortest path from  $u$  to  $v$  for all  $u, v \in V$ .
  - solved with a Dynamic Programming algorithm (Floyd-Warshall).
- Both algorithms need the Optimal Substructure property.

## Properties of shortest paths: Optimal Substructure

**Lemma 24.1:** Subpaths of shortest paths are shortest paths.

**Proof:** Cut and paste:



If some subpath were not a shortest path, we could substitute the shorter subpath and create an even shorter total path.

## All-Pairs Shortest Paths (APSP)

- **All-pairs shortest paths (APSP):** Find a shortest path from  $u$  to  $v$  for all  $u, v \in V$ .
  - The output has size  $O(V^2)$ , so we cannot hope to design a better than  $O(V^2)$ -time algorithm.
  - We can solve the problem simply by running Dijkstra's algorithm  $|V|$  times. It takes  $O(V \lg V)$  time, if the min-priority queue is implemented using a binary heap.

## The Floyd-Warshall algorithm: Step (i)

Step (i): *Characterize the structure of the APSP solution.*

- Definition: An **intermediate vertex** of a simple path  $p = \langle v_1, v_2, \dots, v_l \rangle$  is any vertex of  $p$  other than  $v_1$  and  $v_l$ , i.e., any vertex in the set  $\{v_2, v_3, \dots, v_{l-1}\}$ .
- Define  $d_{ij}^{(k)}$  to be the weight of a shortest path  $p$  from  $i$  to  $j$  for which all intermediate vertices are in the set  $\{1, 2, \dots, k\}$  (*similar to second DP approach to the Coin-Changing problem*).
- Depending on whether or not  $k$  is an intermediate vertex on  $p$ , we have two cases:

## The Floyd-Warshall algorithm: Step (i)

Two cases:

Case (1): If the shortest path  $p$  (from  $i$  to  $j$  going through vertices with indices  $\leq k$ ) does not go through the vertex  $k$ , then:

$$d_{ij}^{(k)} = d_{ij}^{(k-1)}.$$

Case (2): If the shortest path  $p$  goes through vertex  $k$ , then:

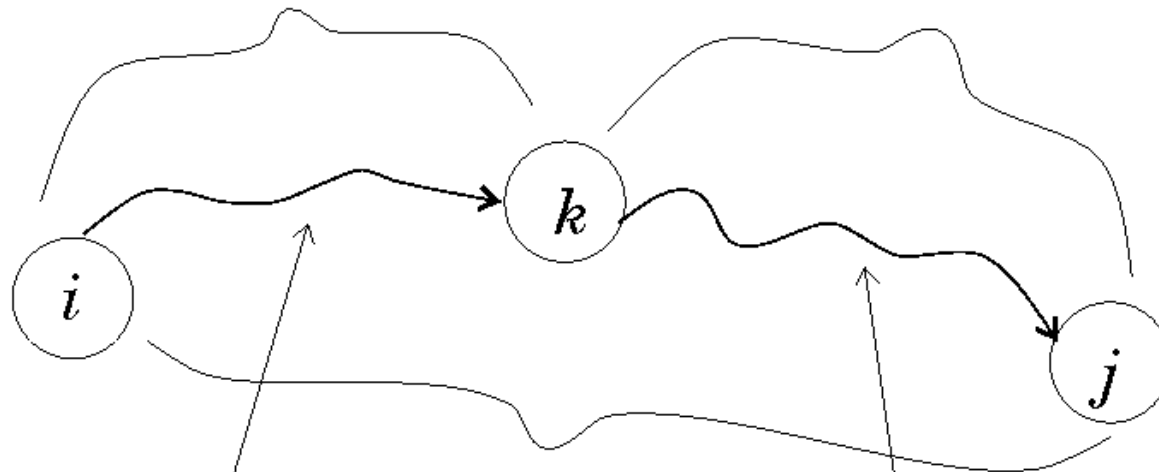
$$d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}.$$

Therefore,  $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$ .

# The Floyd-Warshall algorithm: Step (i)

vertices in  $\{1, 2, \dots, k-1\}$

vertices in  $\{1, 2, \dots, k-1\}$



all intermediate vertices in  $\{1, 2, \dots, k\}$

$$d_{ik}^{k-1}$$

$$d_{kj}^{k-1}$$

## The Floyd-Warshall algorithm: Step (ii)

Step (ii): *Recursively define the value of an optimal solution.*

- **Boundary conditions:** for  $k = 0$ , a path from vertex  $i$  to  $j$  with no intermediate vertex numbered higher than 0 has no intermediate vertices at all, hence  $d_{ij}^{(0)} = w_{ij}$ .
- **Recursive formulation:**

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1 \end{cases}$$

$D^{(n)} = (d_{ij}^{(n)})$  is the solution for this APSP problem:

---

$$d_{ij}^{(n)} = \sigma(i, j).$$

## The Floyd-Warshall algorithm: Step (iii)

Step (iii): *Compute the shortest-path weights bottom up.*

FLOYD-WARSHALL( $W, n$ )

{

$D^{(0)} = W;$

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  $D^{(n)};$

}

Complexity:  $\Theta(n^3)$ .

## The Floyd-Warshall algorithm: Step (iv)

Step (iv): *Constructing the shortest paths.*

Need to compute the **predecessor matrix**  $\Pi$ , in which  $\pi_{ij}$  is the predecessor of vertex  $j$  on a shortest path from vertex  $i$ .

- Compute predecessor matrix  $\Pi$  from the weights matrix  $D$  (Exercise 25.1-6).
- Compute  $\Pi$  online, at the same time with  $D$ :
  - Compute a sequence  $\Pi^{(0)}, \Pi^{(1)}, \dots, \Pi^{(n)}$ , where  $\pi_{ij}^{(k)}$  is defined as the predecessor of vertex  $j$  on a shortest path from vertex  $i$  with all intermediate vertices in  $\{1, 2, \dots, k\}$ .
  - $\Pi = \Pi^{(n)}$ .

## The Floyd-Warshall algorithm: Step (iv)

Recursive formulation of  $\pi_{ij}^{(k)}$ :

- When  $k = 0$ , a shortest path from  $i$  to  $j$  has no intermediate vertices at all. Hence:

$$\pi_{ij}^{(k)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty \end{cases}$$

- When  $k \geq 1$ :
  - If we take the path  $i \rightsquigarrow k \rightsquigarrow j$ , then  $\pi_{ij}^{(k)}$  is the same as the predecessor of  $j$  on the shortest path from  $k$  with intermediate vertices in  $1, 2, \dots, k - 1$ .

$$\pi_{ij}^{(k)} = \pi_{kj}^{(k-1)} \quad \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$$

## The Floyd-Warshall algorithm: Step (iv)

- When  $k \geq 1$ :
  - Otherwise,  $\pi_{ij}^{(k)}$  is the same as the predecessor of  $j$  on the shortest path from  $i$  with intermediate vertices in  $1, 2, \dots, k - 1$ .

$$\pi_{ij}^{(k)} = \pi_{ij}^{(k-1)} \quad \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$$

- Putting these two cases together:

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

## The Floyd-Warshall algorithm: Step (iv)

FLOYD-WARSHALL( $W, n$ )

$D^{(0)} = W;$

INIT-PREDECESSORS( $\Pi^{(0)}$ )

for  $k := 1$  to  $n$

  for  $i := 1$  to  $n$

    for  $j := 1$  to  $n$

      if  $d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$  then

$d_{ij}^{(k)} := d_{ij}^{(k-1)}$

$\pi_{ij}^{(k)} := \pi_{ij}^{(k-1)}$

      else

$d_{ij}^{(k)} := d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

$\pi_{ij}^{(k)} := \pi_{kj}^{(k-1)}$

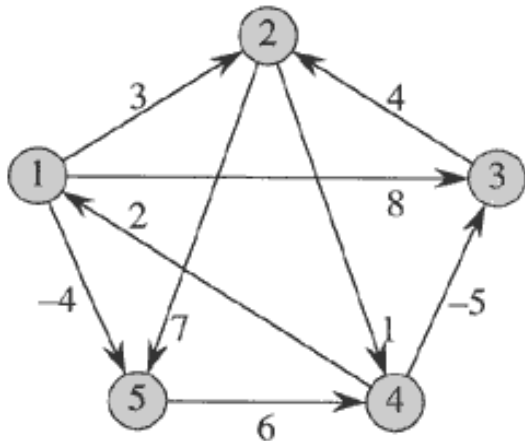
return  $D^{(n)};$

## Printing Shortest Paths with $\Pi$

The predecessor matrix is  $\Pi = \Pi^{(n)}$ . The following recursive procedure prints the shortest path between vertices  $i$  and  $j$ , using  $\Pi$ :

```
PRINT-ALL-PAIRS-SHORTEST-PATHS( $\Pi, i, j$ )
  if  $i = j$  then
    print  $i$ 
  else
    if  $\pi_{ij} = NIL$  then
      print "no path from"  $i$  " to "  $j$ 
    else
      PRINT-ALL-PAIRS-SHORTEST-PATHS( $\Pi, i, \pi_{ij}$ )
  print  $j$ 
```

# APSP: Example

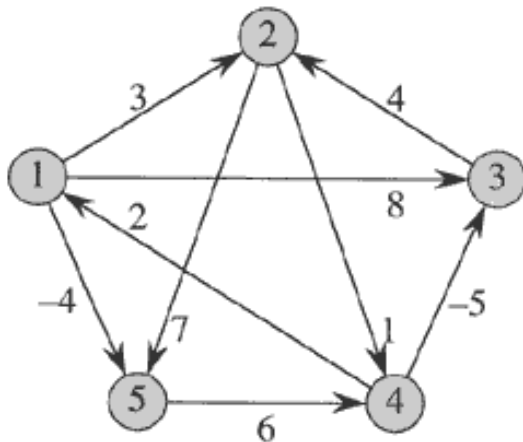


$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

# APSP: Example



$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$