

# Advanced Algorithms Analysis and Design

By

Nazir Ahmad Zafar

# Lecture No 36

## All Pairs Shortest Paths

# Today Covered

- All Pairs Shortest Paths
- Algorithms
  - Matrix Multiplication
  - The Floyd-Warshall Algorithm
- Time Complexity
- Conclusion

# All-Pairs Shortest Path (APSP): Approach

- In all-pair shortest path problems, graph  $G$  given as
  - Directed, weighted with weight function  $w : E \rightarrow \mathbb{R}$
  - where  $w$  is a function from edge set to real-valued weights
- Our objective is to find shortest paths, for all pair of vertices  $u, v \in V$ ,

## Approach

- All-pair shortest path problem can be solved by running single source shortest path in  $|V|$  times, by taking each vertex as a source vertex.
- Now there are two cases.

# Edges are non-negative

## Case 1

- Then use Dijkstra's algorithm
- Linear array of min-priority queue takes  
 $O(V^3)$
- Binary min-heap of min-priority queue,  
 $O(VE \lg V)$
- Fibonacci heap of min-priority queue takes  
 $O(V^2 \lg V + VE)$

# Negative weight edges are allowed

## Case 2

- Bellman-Ford algorithm can be used when negative weight edges are present
- In this case, the running time is  $O(V^2E)$
- However if the graph is dense then the running time is  $O(V^4)$

## Note

- Unlike single-source shortest path algorithms, most algorithms of all pair shortest problems use an adjacency-matrix representation
- Let us define adjacency matrix representation.

# Adjacency Matrix Representation

## Assumptions

- Vertices are numbered from 1, 2, . . . ,  $|V|$
- In this way input is an  $n \times n$  matrix
- $W$  represents edges weights of  $n$ -vertex directed weighted graph  $G$  i.e.,

$W = (w_{ij})$ , where

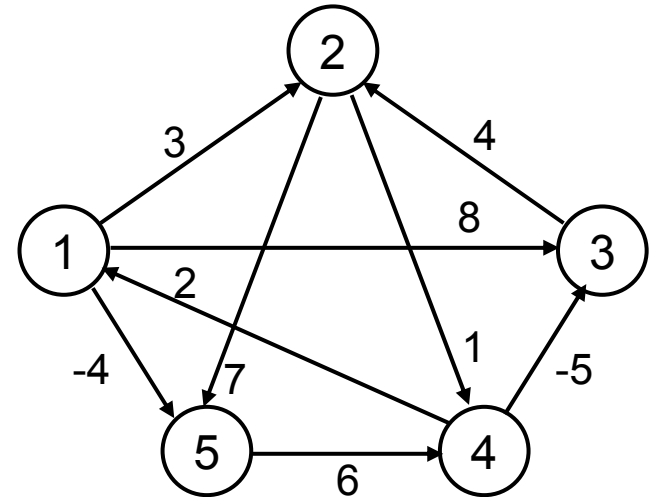
$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ \text{the weight of directed edge } (i, j) & \text{if } i \neq j \text{ and } (i, j) \in E, \\ \infty & \text{if } i \neq j \text{ and } (i, j) \notin E \end{cases}$$

# Shortest Path and Solution Representation

- For a moment we assume that negative-weight edges are allowed, but input graph contains no negative-weight cycle
- The tabular output of all-pairs shortest-paths algorithms will be presented as an  $n \times n$  matrix  $D = (d_{ij})$ , where entry  $d_{ij}$  contains the weight of a shortest path from vertex  $i$  to vertex  $j$ . And the
- A **Predecessor Matrix**  $\Pi = (\pi_{ij})$ , where
$$\pi_{ij} = \text{NIL}, \quad \text{if either } i = j \text{ or no path from } i \text{ to } j$$
$$\pi_{ij} = \text{predecessor of } j \text{ on some shortest path from } i, \text{ otherwise}$$

# Example: All-Pairs Shortest Path

- **Given**
  - Directed graph  $G = (V, E)$
  - Weight function  $w : E \rightarrow \mathbf{R}$
- **Compute**
  - The shortest paths between all pairs of vertices in a graph
  - Representation of result:
    - $5 \times 5$  matrix of shortest-path distances  $\delta(u, v)$
    - $5 \times 5$  matrix of predecessor sub-graph



# Structure of Output: Sub-graph for each row

- For each vertex  $i \in V$  the **Predecessor Subgraph** of  $G$  for  $i$  is defined as  $G_{\pi, i} = (V_{\pi, i}, E_{\pi, i})$ , where

$$V_{\pi, i} = \{j \in V : \pi_{ij} \neq \text{NIL}\} \cup \{i\} \text{ and}$$

$$E_{\pi, i} = \{(i, j) : j \in V_{\pi, i} - \{i\}\}$$

- $G_{\pi, i}$  is shortest pat tree as was in single source shortest path problem

# Printing Output

PRINT-ALL-PAIRS-SHORTEST-PATH ( $\Pi, i, j$ )

```
1  if  $i = j$ 
2      then print  $i$ 
3      else if  $\pi_{ij} = \text{NIL}$ 
4          then print "no path from"  $i$  "to"  $j$  "exists"
5          else PRINT-ALL-PAIRS-SHORTEST-
              PATH( $\Pi, i, \pi_{ij}$ )
6              print  $j$ 
```

## Shortest Paths and Matrix Multiplication

# Shortest Paths and Matrix Multiplication

- Here we present a dynamic-programming algorithm for all-pairs shortest paths on a directed graph  $G = (V, E)$ .
- Each major loop of dynamic program will invoke an operation very similar to multiplication of two matrices, and algorithm looks like repeated matrix multiplication
- At first we will develop  $\Theta(V^4)$ -time algorithm and then improve its running time to  $\Theta(V^3 \lg V)$ .
- Before we go for dynamic solution, let us have a review of steps involved in dynamic-programming algorithms.

# Steps in Dynamic Programming

Steps on dynamic-programming algorithm are

- Characterize the structure of an optimal solution.
- Recursively define value of an optimal solution
- Computing value of an optimal solution in bottom-up
- Constructing optimal solution from computed information

**Note:**

Steps 1-3 are for optimal value while step 4 is for computing optimal solution

# 1. Structure of an Optimal Solution

- Consider shortest path  $p$  from vertex  $i$  to  $j$ , and suppose that  $p$  contains at most  $m$  edges
  - If  $i = j$ , then  $p$  has weight 0 and no edges
  - If  $i$  and  $j$  are distinct, then decompose path  $p$  into  $i \xrightarrow{p'} k \rightarrow j$ , where path  $p'$  contains at most  $m - 1$  edges and it is a shortest path from vertex  $i$  to vertex  $k$ , and

Hence  $\delta(i, j) = \delta(i, k) + w_{kj}$ .

## 2. A Recursive Solution

- Let  $l_{ij}^{(m)}$  = minimum weight of path  $i$  to  $j$  at most  $m$  edges
  - $m = 0$ , there is shortest path  $i$  to  $j$  with no edges  $\Leftrightarrow i = j$ , thus

$$l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$$

- $m \geq 1$ , compute using  $l_{ij}^{(m-1)}$  and adjacency matrix  $w$

$$\begin{aligned} l_{ij}^{(m)} &= \min \left( l_{ij}^{(m-1)}, \min_{1 \leq k \leq n} \left\{ l_{ik}^{(m-1)} + w_{kj} \right\} \right) \\ &= \min_{1 \leq k \leq n} \left\{ l_{ik}^{(m-1)} + w_{kj} \right\} \end{aligned}$$

- The actual shortest-path weights are therefore given by

$$\delta(i, j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \dots$$

### 3. Compute Shortest Path Weights Bottom-up

- Input matrix  $W = (w_{ij})$ ,
- Suppose that,  $L^{(m)} = (l_{ij}^{(m)})$ , where,  $m = 1, 2, \dots, n - 1$
- Compute series of matrices  $L^{(1)}, L^{(2)}, \dots, L^{(n-1)}$ ,
- Objective function,  $L^{(n-1)}$ , at most  $n-1$  edges in each path

#### Note

- Observe that  $l_{ij}^{(1)} = w_{ij}$ , for all  $i, j \in V$ , and so  $L^{(1)} = W$
- Heart of the algorithm is : given matrices  $L^{(m-1)}$  and  $W$ , and compute the matrix  $L^{(m)}$
- That is, it extends shortest paths computed so far by one more edge.

# Algorithm: Extension from $L^{(m-1)}$ to $L^{(m)}$

## EXTEND-SHORTEST-PATHS ( $L, W$ )

```
1   $n \leftarrow \text{rows}[L]$ 
2  let  $L' = (l'_{ij})$  be an  $n \times n$  matrix
3  for  $i \leftarrow 1$  to  $n$ 
4      do for  $j \leftarrow 1$  to  $n$ 
5          do  $l'_{ij} \leftarrow \infty$ 
6              for  $k \leftarrow 1$  to  $n$ 
7                  do
8  return  $L'$   $l'_{ij} \leftarrow \min(l'_{ij}, l_{ik} + w_{kj})$ 
```

**Total Running Time =  $\Theta(n^3)$**

# Algorithm is Similar to Matrix Multiplication

## MATRIX-MULTIPLY ( $A, B$ )

```
1   $n \leftarrow \text{rows}[A]$ 
2  let  $C$  be an  $n \times n$  matrix
3  for  $i \leftarrow 1$  to  $n$ 
4      do for  $j \leftarrow 1$  to  $n$ 
5          do  $c_{ij} \leftarrow 0$ 
6              for  $k \leftarrow 1$  to  $n$ 
7                  do  $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$ 
8  return  $C$ 
```

<b>Total Running Time = <math>\Theta(n^3)</math></b>
--

# Complete but Slow Algorithm

## SLOW-ALL-PAIRS-SHORTEST-PATHS ( $W$ )

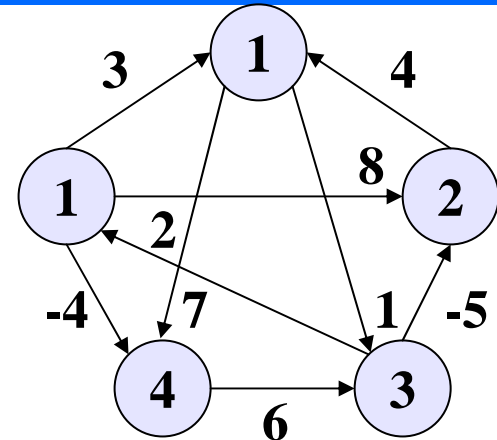
```
1   $n \leftarrow \text{rows } [W]$ 
2   $L^{(1)} \leftarrow W$ 
3  for  $m \leftarrow 2$  to  $n - 1$ 
4      do  $L^{(m)} \leftarrow \text{EXTEND-SHORTEST-PATHS}$   

          $(L^{(m-1)}, W)$ 
5  return  $L^{(n-1)}$ 
```

<b>Total Running Time = <math>\Theta(n^4)</math></b>
--

# Example

$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$



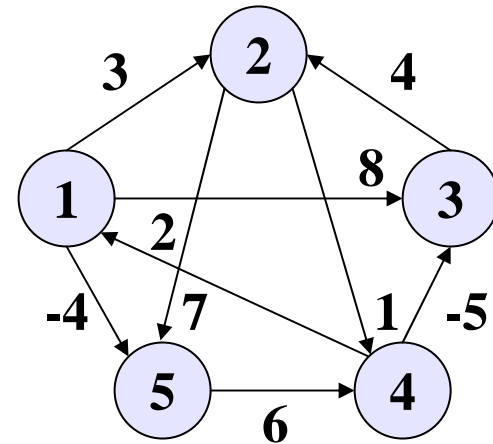
$$L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

# Example

The reader may verify that  $L^{(4)} = L^{(5)} = L^{(6)} = \dots$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$



# Improving Running Time

- Running time of previous algorithm is very high and needs improvement.
- The goal is not to compute all the  $L^{(m)}$  matrices but only computation of matrix  $L^{(n-1)}$  is of interest
- As Matrix multiplication associative,  $L^{(n-1)}$  can be calculated with only  $\lceil \lg(n-1) \rceil$  matrix products as.

$$L^{(1)} = W,$$

$$L^{(2)} = W^2 = W \cdot W,$$

$$L^{(4)} = W^4 = W^2 \cdot W^2,$$

$$L^{(8)} = W^8 = W^4 \cdot W^4,$$

⋮

$$L^{n-1} = L^{2^{\lceil \lg(n-1) \rceil}} = W^{2^{\lceil \lg(n-1) \rceil}}$$

# Improved Algorithm

## FASTER-ALL-PAIRS-SHORTEST-PATHS ( $W$ )

```
1   $n \leftarrow \text{rows}[W]$ 
2   $L^{(1)} \leftarrow W$ 
3   $m \leftarrow 1$ 
4  while  $m < n - 1$ 
5      do  $L^{(2m)} \leftarrow \text{EXTEND-SHORTEST-PATHS}$ 
         $(L^{(m)}, L^{(m)})$ 
6       $m \leftarrow 2m$ 
7  return  $L^{(m)}$ 
```

**Total Running Time =  $\Theta(n^3 \lg n)$**