# Design and Analysis of Algorithms

- ***Course Code***: BCSE204L
- ***Course Type***: Theory (ETH)
- ***Slot***: A1+TA1 & & A2+TA2
- ***Class ID***: VL2023240500901
  VL2023240500902

A1+TA1

| Day | Start | End |
|---|---|---|
| Monday | 08:00 | 08:50 |
| Wednesday | 09:00 | 09:50 |
| Friday | 10:00 | 10:50 |

A2+TA2

| Day | Start | End |
|---|---|---|
| Monday | 14:00 | 14:50 |
| Wednesday | 15:00 | 15:50 |
| Friday | 16:00 | 16:50 |

# Syllabus- Module 4

| Module:4 | Graph Algorithms | 6 hours |
| --- | --- | --- |

All pair shortest path: Bellman Ford Algorithm, Floyd-Warshall Algorithm
Network Flows: Flow Networks, Maximum Flows: Ford-Fulkerson, Edmond-Karp, Push Re-label Algorithm – Application of Max Flow to maximum matching problem

Types of Shortest Path Algorithms

1. Single-source shortest path algorithms
2. All-pairs shortest path algorithms

## All-pairs shortest path algorithms

Given a graph $G$, with vertices $V$, edges $E$ with weight function

w(u, v) $=w_{u,v}$ return the shortest path from $u$ to $v$ for all $(u,v)$ in $V$.

**All Pair Source Shortest Path (APSP) Algorithm**:

- **Purpose**: Finding the shortest path between all pairs of nodes in the graph.
- **Common Algorithm**:
    - Floyd-Warshall Algorithm is a popular choice for solving APSP problem. It's based on dynamic programming and works efficiently for dense graphs.

# Difference between single-source shortest path (SSSP) and all-pairs shortest path (APSP) algorithms

## Single-Source Shortest Path (SSSP)

- SSSP algorithms focus on finding the shortest path from a single source vertex to all other vertices in the graph.
- The output of an SSSP algorithm provides the shortest paths from one specific source vertex to all other vertices in the graph.
- Examples of SSSP algorithms include Dijkstra's algorithm and the Bellman-Ford algorithm.

## All-Pairs Shortest Path (APSP):

- APSP algorithms aim to find the shortest path between every pair of vertices in the graph.
- The output of an APSP algorithm provides the shortest paths between every pair of vertices in the graph.
- Examples of APSP algorithms include Floyd-Warshall algorithm and Johnson's algorithm.

# Difference between single-source shortest path (SSSP) and all-pairs shortest path (APSP) algorithms

In a graph with N vertices

**Single-Source Shortest Path (SSSP)**

- There is only one source node.
- There are N - 1 destination nodes (all other vertices except the source vertex), considering each node as a destination from the perspective of the source vertex.

**All-Pairs Shortest Path (APSP):**

- The total number of pairs of vertices is N * (N - 1), considering all possible combinations of pairs with repetitions.
- Each vertex serves as a source for $N-1$ other vertices, resulting in $N\times(N-1)$ total distinct pairs.

# ***All-pairs shortest path algorithms***

The ***Floyd-Warshall*** algorithm and ***Johnson's algorithm*** are two commonly used approaches for solving the all-pairs shortest path problem, each with its own strengths and weaknesses.

**Floyd-Warshall Algorithm**:

- This algorithm employs dynamic programming to efficiently compute the shortest paths between all pairs of vertices in a weighted graph.
- It works efficiently for dense graphs, where the number of edges is close to the maximum possible number of edges.
- It has a time complexity of $O(V^3)$, where V is the number of vertices in the graph.
- It can handle graphs with negative edge weights, making it suitable for a broader range of applications.

**Johnson's Algorithm**:

- Johnson's algorithm is more suitable for sparse graphs, where the number of edges is much smaller than the maximum possible number of edges.
- It combines Dijkstra's algorithm with Bellman-Ford algorithm to efficiently find all-pairs shortest paths in a graph with potentially negative edge weights.
- Johnson's algorithm has a time complexity of $O(V^2 \log V + VE)$, where V is the number of vertices and E is the number of edges. It may be more efficient than Floyd-Warshall for sparse graphs due to its lower asymptotic running time.

**Note**: Dijkstra's algorithm is also sometimes used to solve the all-pairs shortest path problem by simply running it on all vertices in V. Again, this requires all edge weights to be positive.

# Relation between the single-source shortest path (SSSP) and all pair shortest path (APSP)

- In SSSP, you find the shortest path from a single source node to all other nodes in the graph. This process typically involves algorithms like Dijkstra's algorithm or Bellman-Ford algorithm.

- In APSP, you find the shortest path between every pair of nodes in the graph. This means you're essentially running SSSP from every vertex in the graph.

- **The relationship you mentioned is correct**: if you were to ***run SSSP from every vertex in the graph, you would effectively obtain the APSP***. However, this approach is computationally inefficient, especially for large graphs, because you're essentially repeating the same calculations multiple times.

# Relation between the single-source shortest path (SSSP) and all pair shortest path (APSP)

- How
  - If you take the Dijkstra algorithm
  - Dijkstra algorithm using heap (i.e., binary min-heap, note: there are other heap algorithm called Fibonacci heap), the time complexity is $O(E \log V)$
  - If this Dijkstra algorithm is applied to every vertex, then total time complexity for all pair shortest path using the Dijkstra algorithm is $O(V\ E \log V)$
  - We already know that number of edges E is $O(V^2)$.
    - Therefore, In the case of a dense graph,

      we can also write $O(V\ E \log V)$ as $O(V\ V^2 \log V)$.

      That is $O(V^3 \log V)$

# Relation between the single-source shortest path (SSSP) and all pair shortest path (APSP)

- But we know that the Diskastra algorithm problem is that it cannot work with negative weight edges.
- That's why we have to go for the bellman ford algorithm (i.e., if you have the graph with negative weight edges)

  Those time complexity is $O(VE)$

- If we run this bellman ford algorithm on every vertex (there are V vertices),

  $O(V* VE)=O(V^2E)$

- We know that E is the order of $V^2$ (In the dense graph)

  Then $O(V^2\ V^2)=O(V^4)$

These two time complexities of all pair shortest paths are high.

Using dynamic programming, we can implement it in a better way.

**How dynamic programming could be applied**

# APSP: Floyd Warshall Algorithm:

- **Floyd warshall algorithm** algorithm has capable of handling both positive and negative edge weights in a graph.
- This algorithm works for both the directed and undirected weighted graphs. But, there should be no negative edge cycles in the Floyd warshall algorithm's input graph because this algorithm is incapable of detecting negative edge cycles.
- The Floyd-Warshall algorithm is indeed unable to detect negative edge cycles in the input graph. If the graph contains such cycles, it may produce incorrect results.
  - Specifically, the presence of a negative edge cycle leads to the phenomenon where the shortest path between some pairs of nodes becomes negative infinity (-∞).

# APSP: Floyd Warshall Algorithm: Analysis

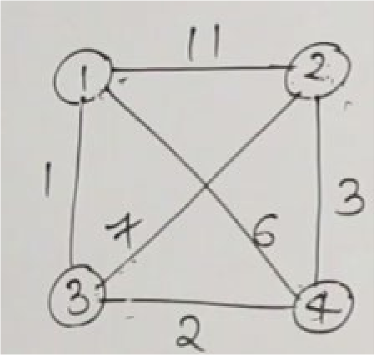
- Floyd-Warhshall algorithm is also called as **Floyd's algorithm, Roy-Floyd algorithm, Roy-Warshall algorithm, or WFI algorithm**.

- This algorithm follows the **dynamic programming** approach to find the shortest paths.

Recurrence Relation:

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

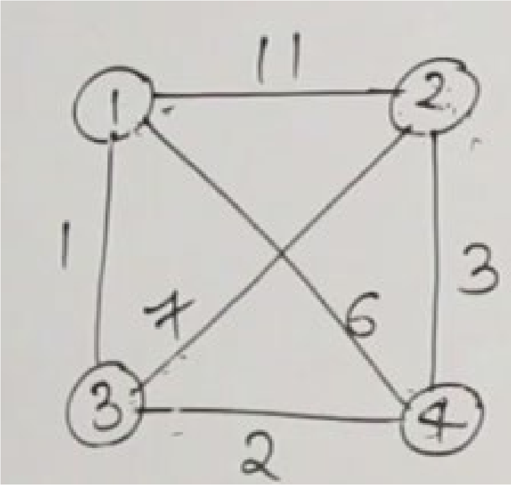
Time Complexity: **O(V^3)**




$$D^{0} = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 11 & 1 & 6 \\ 2 & 11 & 0 & 7 & 3 \\ 3 & 1 & 7 & 0 & 2 \\ 4 & 6 & 3 & 2 & 0 \end{array}$$

- Set of the pairs in Distances in which there is no node between any pair. (Simple, the distance of path having a single edge.)

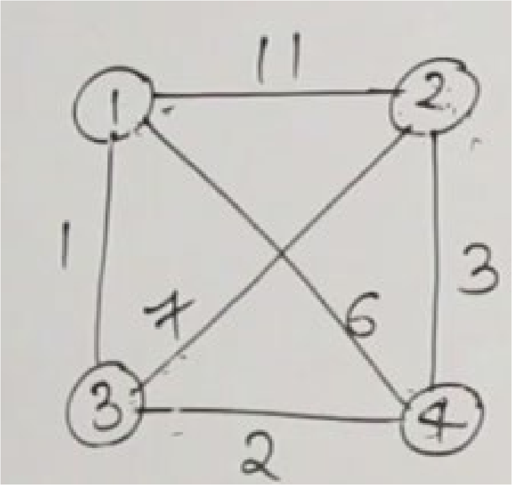
- <u>This is the **base condition**</u>: which is the smallest problem going evaluation.
    - Using this base condition, we are going to increase the problem size to compute the big problem.
    - The smallest size of this problem is the shortest path distance in which there is only one edge included (i.e., no node included in between source and destination).




$$D^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 11 & 1 & 6 \\ 11 & 0 & 7 & 3 \\ 1 & 7 & 0 & 2 \\ 6 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

$$D^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 11 & 1 & 6 \\ 11 & 0 & 7 & 3 \\ 1 & 7 & 0 & 2 \\ 6 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

- Give the path a that uses node 1.
- Compute the shortest path, which one goes through only node 1, if it will be able to do better.



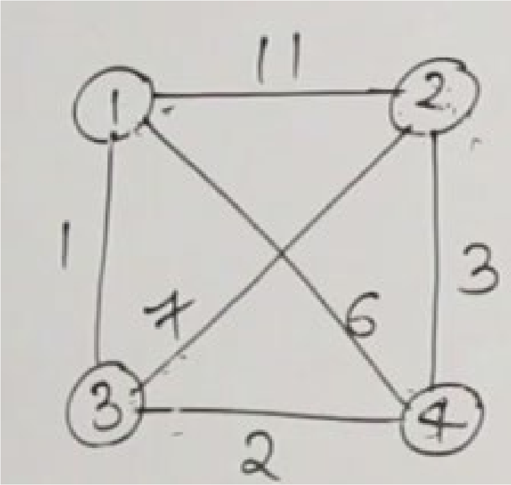

$$D^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 11 & 1 & 6 \\ 11 & 0 & 7 & 3 \\ 1 & 7 & 0 & 2 \\ 6 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

$$D^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 11 & 1 & 6 \\ 11 & 0 & 7 & 3 \\ 1 & 7 & 0 & 2 \\ 6 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

$\{1,2\}$

$$D^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 11 & 1 & 6 \\ 11 & 0 & 7 & 3 \\ 1 & 7 & 0 & 2 \\ 6 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

- Make a path that includes both nodes 1 and 2.
- Set of all minimum path distances between each pair, with the possibility of allowing both vertex 1 and vertex 2 if it can do better.
- It can have both vertices 1 and 2, any one, or none at all.
- It takes D1 to compute.




$$D^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 11 & 1 & 6 \\ 2 & 11 & 0 & 7 & 3 \\ 3 & 1 & 7 & 0 & 2 \\ 4 & 6 & 3 & 2 & 0 \end{array}$$

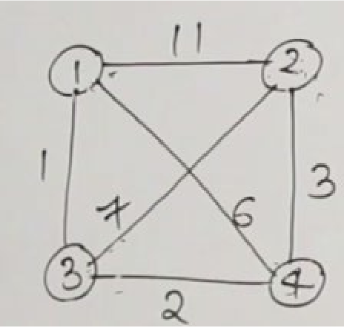
$$D^1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 11 & 1 & 6 \\ 2 & 11 & 0 & 7 & 3 \\ 3 & 1 & 7 & 0 & 2 \\ 4 & 6 & 3 & 2 & 0 \end{array}$$

$\{1,2\}$

$$D^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 11 & 1 & 6 \\ 2 & 11 & 0 & 7 & 3 \\ 3 & 1 & 7 & 0 & 2 \\ 4 & 6 & 3 & 2 & 0 \end{array}$$

$\{1,2,3\}$

$$D^3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & 1 & 3 \\ 2 & 8 & 0 & 7 & 3 \\ 3 & 1 & 7 & 0 & 2 \\ 4 & 3 & 3 & 2 & 0 \end{array}$$




$$D^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 11 & 1 & 6 \\ 2 & 11 & 0 & 7 & 3 \\ 3 & 1 & 7 & 0 & 2 \\ 4 & 6 & 3 & 2 & 0 \end{array}$$

$$D^1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 11 & 1 & 6 \\ 2 & 11 & 0 & 7 & 3 \\ 3 & 1 & 7 & 0 & 2 \\ 4 & 6 & 3 & 2 & 0 \end{array}$$

$\{1,2\}$

$$D^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 11 & 1 & 6 \\ 2 & 11 & 0 & 7 & 3 \\ 3 & 1 & 7 & 0 & 2 \\ 4 & 6 & 3 & 2 & 0 \end{array}$$

$\{1,2,3\}$

$$D^3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & 1 & 3 \\ 2 & 8 & 0 & 7 & 3 \\ 3 & 1 & 7 & 0 & 2 \\ 4 & 3 & 3 & 2 & 0 \end{array}$$

$\{1,2,3,4\}$

$$D^4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 6 & 1 & 3 \\ 2 & 6 & 0 & 5 & 3 \\ 3 & 1 & 5 & 0 & 2 \\ 4 & 3 & 3 & 2 & 0 \end{array}$$

For each consideration of one node in between pair, we have $n^2$ problems

If we have n nodes are there, then $n(n^2)$ problems are there.

$n(n^2)=O(n^3)$

Space required is two matrices in case of best case.

Given matrix-D:
0 11 1 6
11 0 7 3
1 7 0 2
6 3 2 0

This matrix represents the distances between vertices in a graph. Each row and column correspond to a vertex, and the value at the intersection represents the distance between those vertices.

| | |
|---|---|
| D0 =<br>0 11 1 6<br>11 0 7 3<br>1 7 0 2<br>6 3 2 0 | This is the initial matrix where only the direct edge weights are considered. |

| | |
|---|---|
| D1 =<br>0 11 1 6<br>11 0 7 3<br>1 7 0 2<br>6 3 2 0 | **Iteration 1 (D1):**<br>No change occurs because we're considering paths that include only node 1, and there are no such paths that can improve the distances. |

| | |
|---|---|
| D2 =<br>0 11 1 3<br>11 0 7 3<br>1 7 0 2<br>6 3 2 0 | **Iteration 2 (D2):**Here, we update the distances matrix to consider paths that include nodes 1 and 2. For example, the distance between vertices 1 and 3 (originally 1) is updated to 1+2=3 because now we can go from 1 to 2 (distance 11) and then from 2 to 3 (distance 2), which is shorter |

| | |
|---|---|
| D3 =<br>0 8 1 3<br>8 0 7 3<br>1 7 0 2<br>3 3 2 0 | **Iteration 3 (D3):**Continuing the process, we update the distances matrix to include paths that go through nodes 1, 2, and 3. For example, the distance between vertices 1 and 3 (originally 3) is updated to 1+7=8 because now we can go from 1 to 2 (distance 11), then from 2 to 3 (distance 7), and finally from 3 to 1 (distance 1), which is shorter. |

| | |
|---|---|
| D4 =<br>0 6 1 3<br>6 0 5 3<br>1 5 0 2<br>3 3 2 0 | **Iteration 4 (D4):**Finally, we update the distances matrix to include paths that go through nodes 1, 2, 3, and 4. For example, the distance between vertices 1 and 3 (originally 8) is updated to 1+5=6 because now we can go from 1 to 2 (distance 6), then from 2 to 3 (distance 5), and finally from 3 to 1 (distance 1), which is shorter. |

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

FLOYD-WARSHALL($\omega$)

{

1) $n = \omega.rows$

2) $D^0 = W$

3) for $K = 1$ to $n$

4) let $D^{(k)} = (d_{ij}^{k})$ be a new $n \times n$ matrix

5) for $i = 1$ to $n$

6) for $j = 1$ to $n$

7) $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$

8) return $D^{(n)}$

FLOYD-WARSHALL(W)

{

1) $n = W.rows$

2) $D^{0} = W$

3) for $k = 1$ to $n$

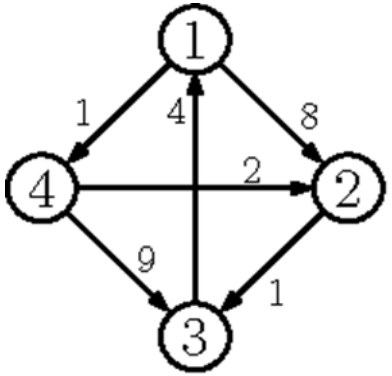
4) let $D^{(k)} = (d_{ij}^{k})$ be a new $n \times n$ matrix

5) for $i = 1$ to $n$

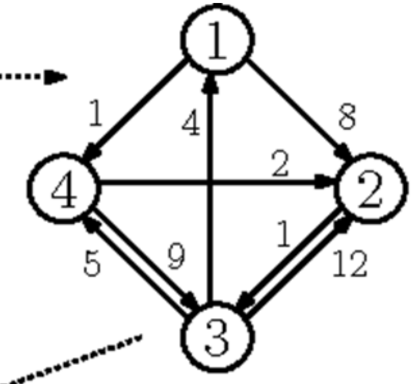
6) for $j = 1$ to $n$

7) $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$

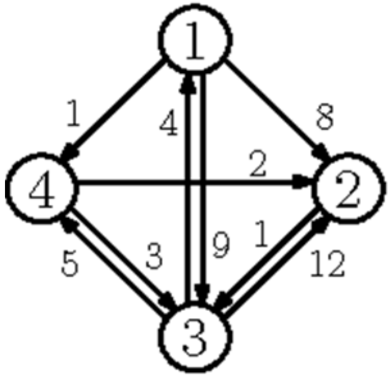
8) return $D^{(n)}$



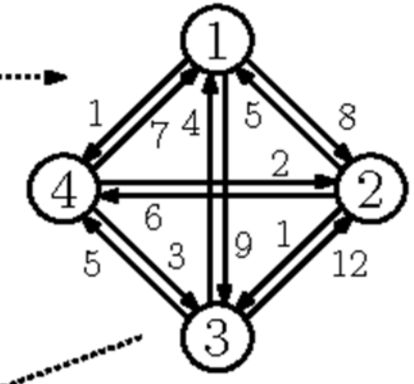

$$d^{(0)} = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix}$$



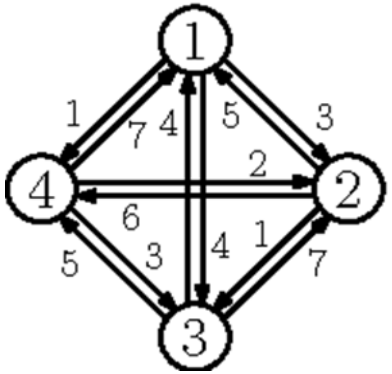

$$d^{(1)} = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix}$$




$$d^{(2)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{bmatrix}$$



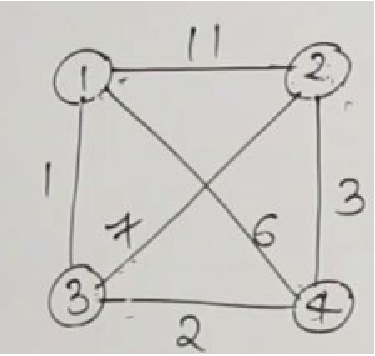

$$d^{(3)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$$




$$d^{(4)} = \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$$

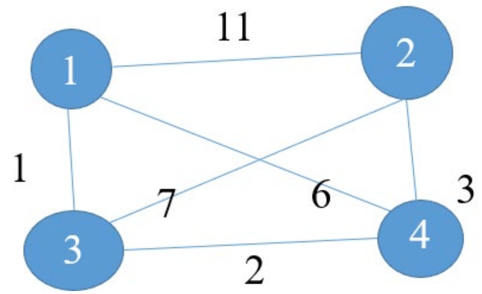
$$\text{final} = \begin{bmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix}$$

# Examples: 1




Given matrix-D:
0 11 1 6
11 0 7 3
1 7 0 2
6 3 2 0

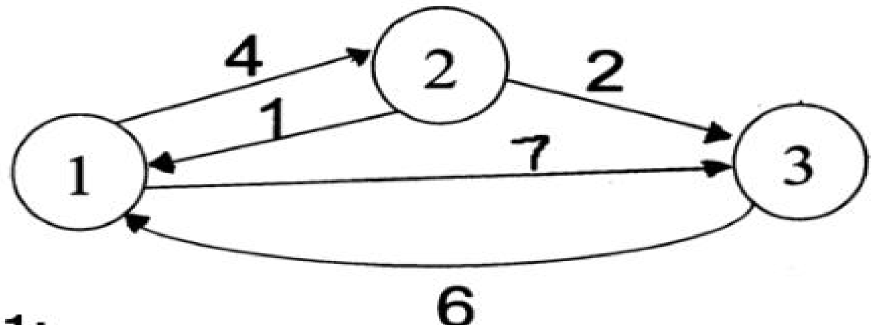
# Examples: 2




Given matrix-D:

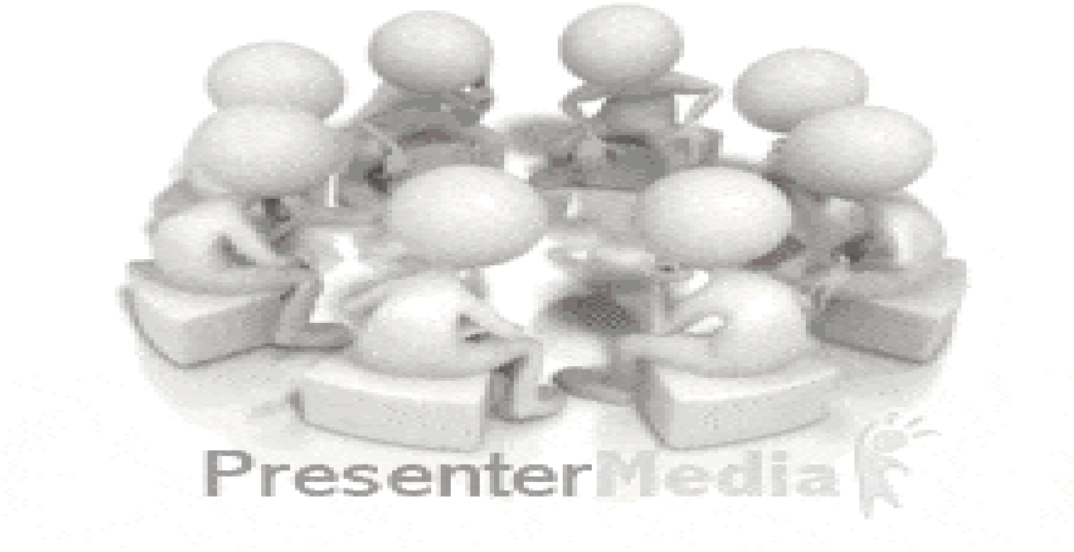
| | | | |
|---|---|---|---|
| 0 | 6 | 1 | 3 |
| 6 | 0 | 5 | 3 |
| 1 | 5 | 0 | 2 |
| 3 | 3 | 2 | 0 |

# Examples: 3




Given matrix-D:
0 4 7
1 0 2
6 inf 0


Any
Question


PresenterMedia


Thank You!


FOR YOUR
ATTENTION