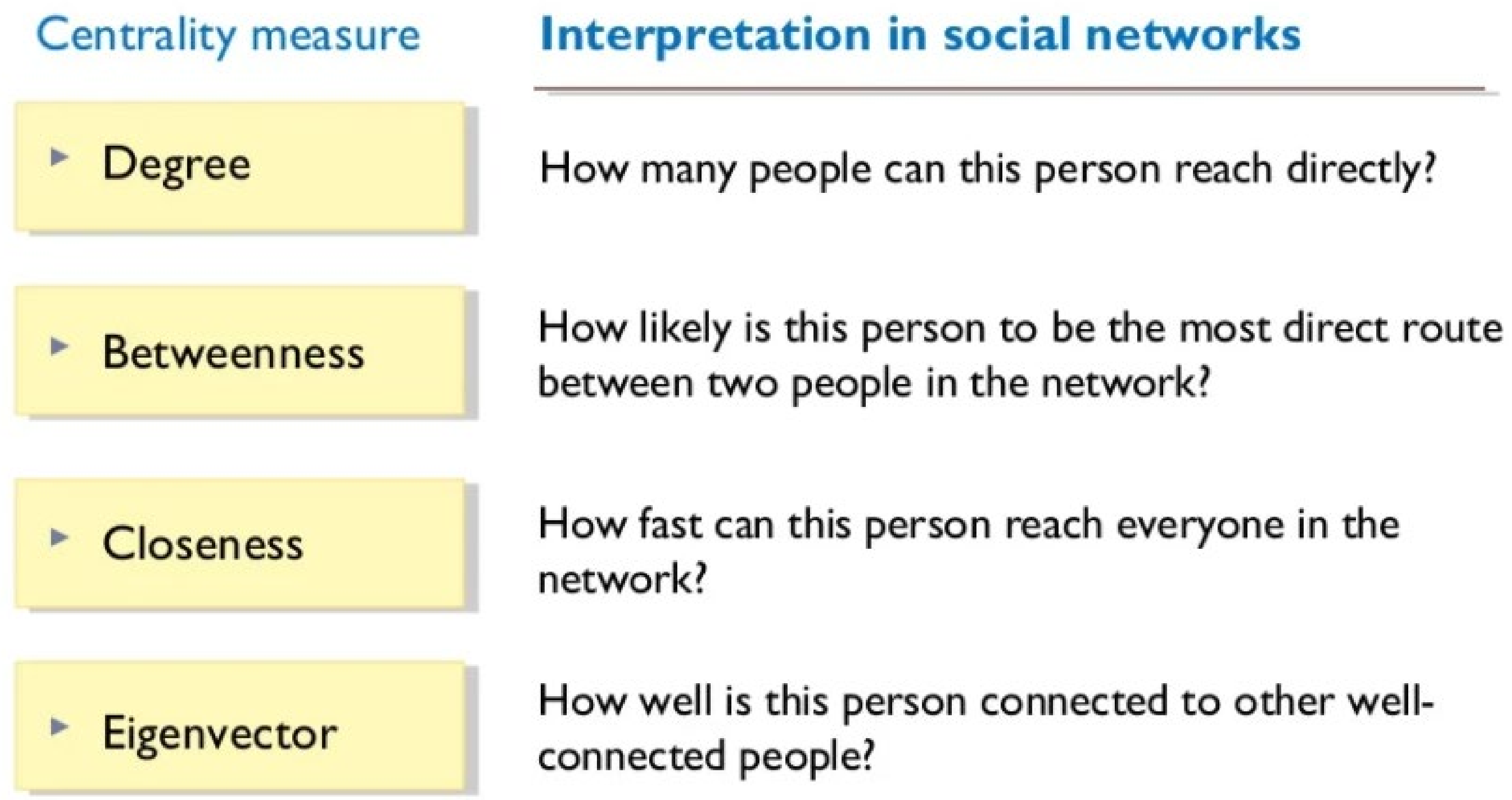
Social Networking
الشبكات الاجتماعية
ITMC 413

إعداد

أ.منار سامي عريف

Interpretation of measures (1)

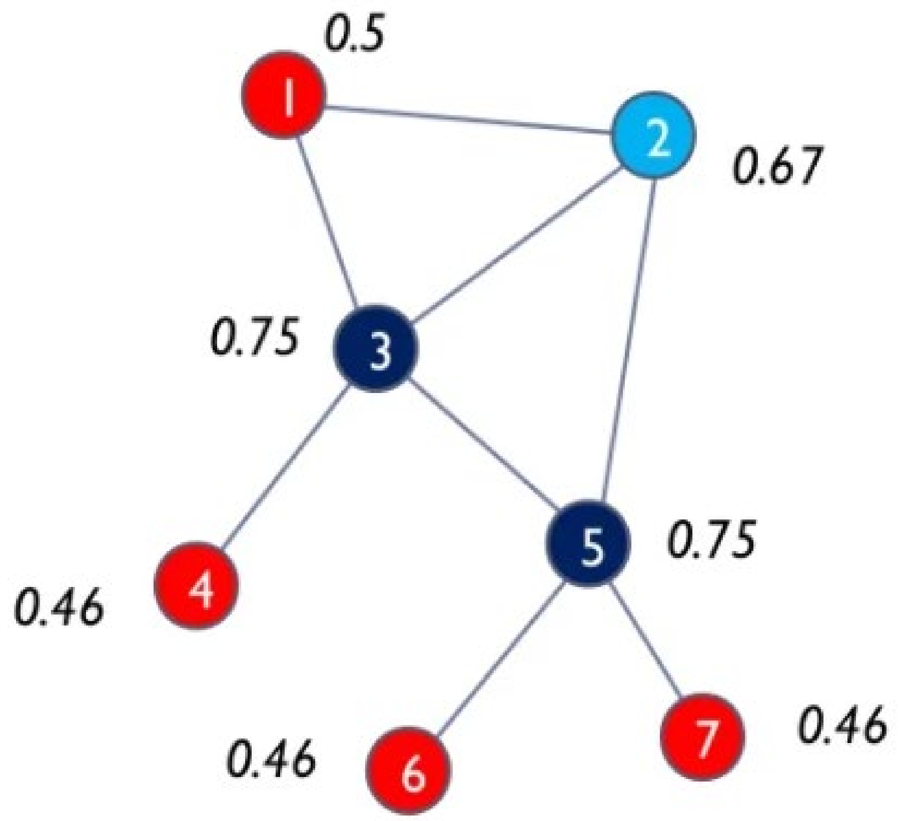


Centrality measure	Interpretation in social networks
Degree	How many people can this person reach directly?
Betweenness	How likely is this person to be the most direct route between two people in the network?
Closeness	How fast can this person reach everyone in the network?
Eigenvector	How well is this person connected to other well-connected people?

Closeness centrality

$$C_C(i) = \frac{n-1}{\sum_{j=1}^{n} d(i,j)}$$

- Calculate the mean length of all shortest paths from a node to all other nodes in the network (i.e. how many hops on average it takes to reach every other node)
- Take the reciprocal of the above value so that higher values are 'better' (indicate higher closeness) like in other measures of centrality
- It is a measure of *reach*, i.e. the speed with which information can reach other nodes from a given starting node



Nodes 3 and 5 have the highest (i.e. best) closeness, while node 2 fares almost as well

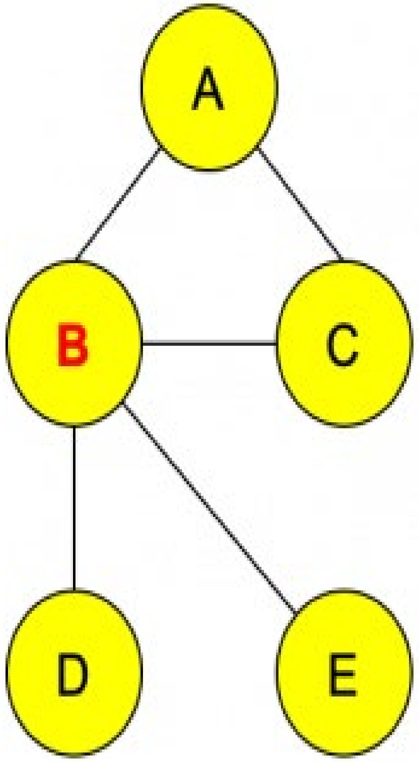
Note: Sometimes closeness is calculated without taking the reciprocal of the mean shortest path length. Then lower values are 'better'.

Closeness centrality

- is a useful measure that estimates how fast the flow of information would be through a given node to other nodes.
- Closeness centrality measures how short the shortest paths are from node *i* to all nodes. It is usually expressed as the normalized inverse of the sum of the topological distances in the graph.
- This sum is also known as the farness of the nodes. Sometimes closeness centrality is also expressed simply as the inverse the farness.

$$CC(i) = \frac{N-1}{\sum_j d(i,j)}$$

where
$i \neq j$,
d_{ij} is the length of the shortest path between nodes i and j in the network,
N is the number of nodes.



$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 1 & 2 & 2 \\ B & 1 & 0 & 1 & 1 & 1 \\ C & 1 & 1 & 0 & 2 & 2 \\ D & 2 & 1 & 2 & 0 & 2 \\ E & 2 & 1 & 2 & 2 & 0 \end{array}$$

farness $\sum_{j=1}^{n} d(i,j)$	$CC(i) = \frac{N-1}{\sum_j d(i,j)}$
6	(5-1)/6= 0.67
4	1.00
6	0.67
7	0.57
7	0.57

N = 5 (# of nodes)

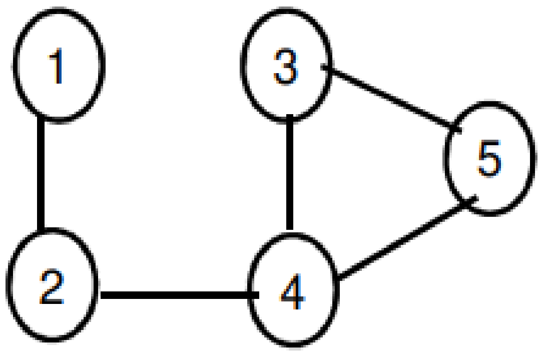
Eigenvector Centrality

is an algorithm that measures
the transitive influence of nodes. Relationships originating from high-scoring nodes contribute more to the score of a node than connections from low-scoring nodes. A high eigenvector score means that a node is connected to many nodes who themselves have high scores.

Eigenvector Centrality

- The algorithm computes the eigenvector associated with the largest absolute eigenvalue. To compute that eigenvalue, the algorithm applies the power iteration approach .
- Within each iteration, the centrality score for each node is derived from the scores of its incoming neighbors. In the power iteration method, the eigenvector is normalized after each iteration, leading to normalized results by default.
- Power iteration is a very simple algorithm, but it may converge slowly .

EigenVector Centrality Example (1)



$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Let X0 = $\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$

Iteration 1

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 3 \\ 2 \end{bmatrix} \equiv \begin{bmatrix} 0.213 \\ 0.426 \\ 0.426 \\ 0.639 \\ 0.426 \end{bmatrix}$$

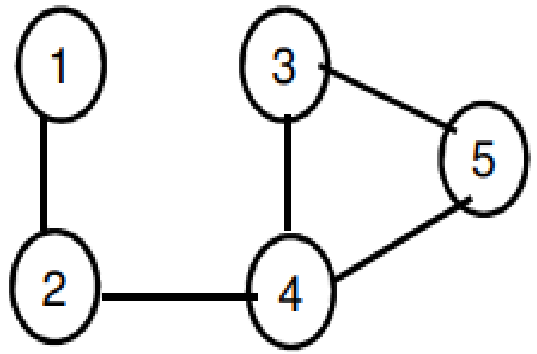
Normalized Value = 4.69

Iteration 2

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0.213 \\ 0.426 \\ 0.426 \\ 0.639 \\ 0.426 \end{bmatrix} = \begin{bmatrix} 0.426 \\ 0.852 \\ 1.065 \\ 1.278 \\ 1.065 \end{bmatrix} \equiv \begin{bmatrix} 0.195 \\ 0.389 \\ 0.486 \\ 0.584 \\ 0.486 \end{bmatrix}$$

Normalized Value = 2.19

EigenVector Centrality Example (1)



$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Let X0 = $\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$

Iteration 3

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0.195 \\ 0.389 \\ 0.486 \\ 0.584 \\ 0.486 \end{bmatrix} = \begin{bmatrix} 0.389 \\ 0.779 \\ 1.07 \\ 1.361 \\ 1.07 \end{bmatrix} \equiv \begin{bmatrix} 0.176 \\ 0.352 \\ 0.484 \\ 0.616 \\ 0.484 \end{bmatrix}$$

Normalized Value = 2.21

Iteration 4

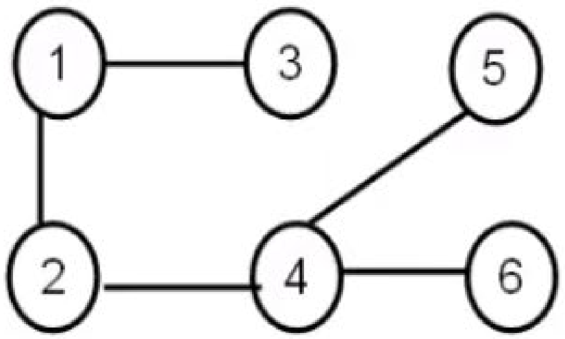
$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0.176 \\ 0.352 \\ 0.484 \\ 0.616 \\ 0.484 \end{bmatrix} = \begin{bmatrix} 0.352 \\ 0.792 \\ 1.100 \\ 1.320 \\ 1.100 \end{bmatrix}$$

Normalized Value = 2.21 converges

Eigen Vector Centrality

$$\begin{array}{c|c} 1 & 0.176 \\ 2 & 0.352 \\ 3 & 0.484 \\ 4 & 0.616 \\ 5 & 0.484 \end{array}$$

EigenVector Centrality Example (2)



$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Let X0 = $\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$

Iteration 1

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 1 \\ 3 \\ 1 \\ 1 \end{bmatrix} \equiv \begin{bmatrix} 0.447 \\ 0.447 \\ 0.224 \\ 0.671 \\ 0.224 \\ 0.224 \end{bmatrix}$$

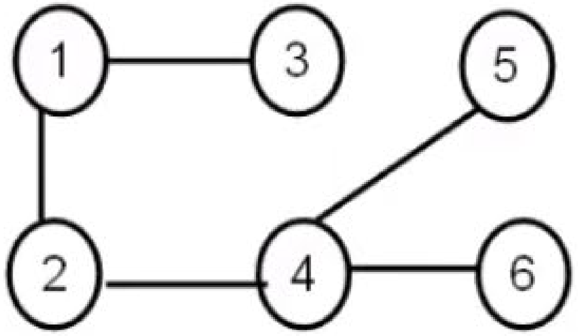
Normalized Value = 4.472

Iteration 2

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.447 \\ 0.447 \\ 0.224 \\ 0.671 \\ 0.224 \\ 0.224 \end{bmatrix} = \begin{bmatrix} 0.671 \\ 0.671 \\ 0.447 \\ 0.895 \\ 0.671 \\ 0.671 \end{bmatrix} \equiv \begin{bmatrix} 0.401 \\ 0.401 \\ 0.267 \\ 0.535 \\ 0.401 \\ 0.401 \end{bmatrix}$$

Normalized Value = 1.674

EigenVector Centrality Example (2)



$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\text{Let } X0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Iteration 3

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.401 \\ 0.401 \\ 0.267 \\ 0.535 \\ 0.401 \\ 0.401 \end{bmatrix} = \begin{bmatrix} 0.668 \\ 0.936 \\ 0.401 \\ 1.203 \\ 0.535 \\ 0.535 \end{bmatrix} \equiv \begin{bmatrix} 0.357 \\ 0.500 \\ 0.214 \\ 0.643 \\ 0.286 \\ 0.286 \end{bmatrix}$$

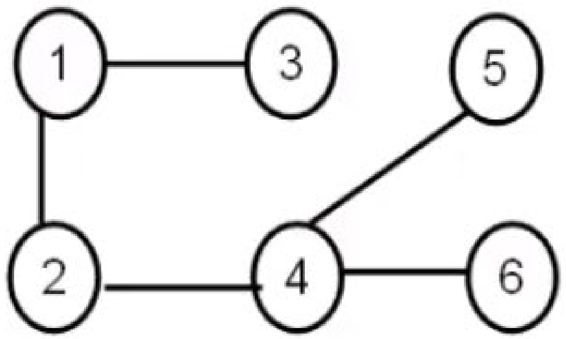
Normalized Value = 1.872

Iteration 4

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.357 \\ 0.500 \\ 0.214 \\ 0.643 \\ 0.286 \\ 0.286 \end{bmatrix} = \begin{bmatrix} 0.714 \\ 1.000 \\ 0.357 \\ 1.072 \\ 0.643 \\ 0.643 \end{bmatrix} \equiv \begin{bmatrix} 0.376 \\ 0.526 \\ 0.188 \\ 0.564 \\ 0.338 \\ 0.338 \end{bmatrix}$$

Normalized Value = 1. 901

EigenVector Centrality Example (2)



$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\text{Let } X0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Iteration 5

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.376 \\ 0.526 \\ 0.188 \\ 0.564 \\ 0.338 \\ 0.338 \end{bmatrix} = \begin{bmatrix} 0.714 \\ 0.940 \\ 0.376 \\ 1.202 \\ 0.564 \\ 0.564 \end{bmatrix} \equiv \begin{bmatrix} 0.376 \\ 0.494 \\ 0.198 \\ 0.632 \\ 0.297 \\ 0.297 \end{bmatrix}$$

Normalized Value = 1. 901 converges

EigenVector Centrality

$$\begin{bmatrix} 0.376 \\ 0.494 \\ 0.198 \\ 0.632 \\ 0.297 \\ 0.297 \end{bmatrix}$$

Node Ranking

4
2
1
5
6
3

Note that we typically stop when the EigenVector values converge.
For exam purposes, we will Stop when the Normalized value converges.

EigenVector Centrality for Directed Graph

For directed graphs, we can use the Eigen Vector
of a node (based ”importance“centrality to evaluate the
of a ”prestige“on the out-degree Eigen Vector) and the
node (through the in-degree Eigen Vector)

A node is considered to be more important if it has out-
going links to nodes that in turn have a larger out-degree
(i.e., more out-going links)

sah ti fi ,”prestige“A node is considered to have a higher
regral a evah sevlesmeht taht sedon morf sknil gnimoc-ni
sknil gnimoc-ni erom ,.e.i) eerged-ni

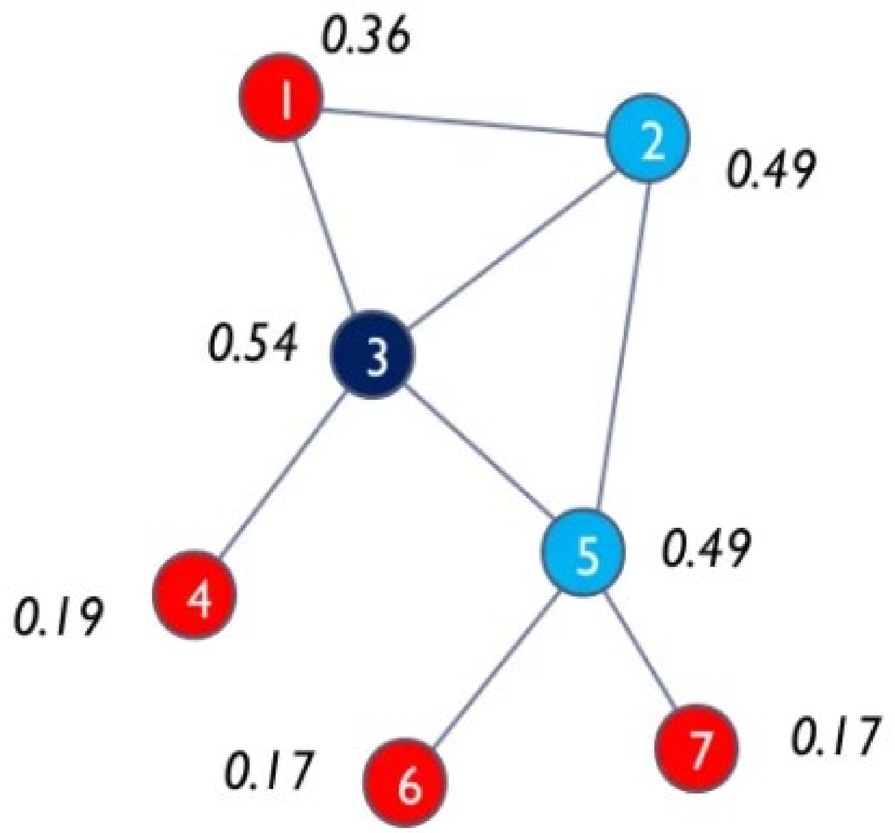
Why is Eigenvector Centrality not commonly used for directed graphs ?

- Adjacency matrix is asymmetric…use left or right leading eigenvector
- Any vertex within degree zero has centrality value zero and “passes ”that value to all vertices to which it points.

Eigenvector centrality

- A node's eigenvector centrality is proportional to the sum of the eigenvector centralities of all nodes directly connected to it
- In other words, a node with a high eigenvector centrality is connected to other nodes with high eigenvector centrality
- This is similar to how Google ranks web pages: links from highly linked-to pages count more
- Useful in determining who is connected to the most connected nodes

Node 3 has the highest eigenvector centrality, closely followed by 2 and 5



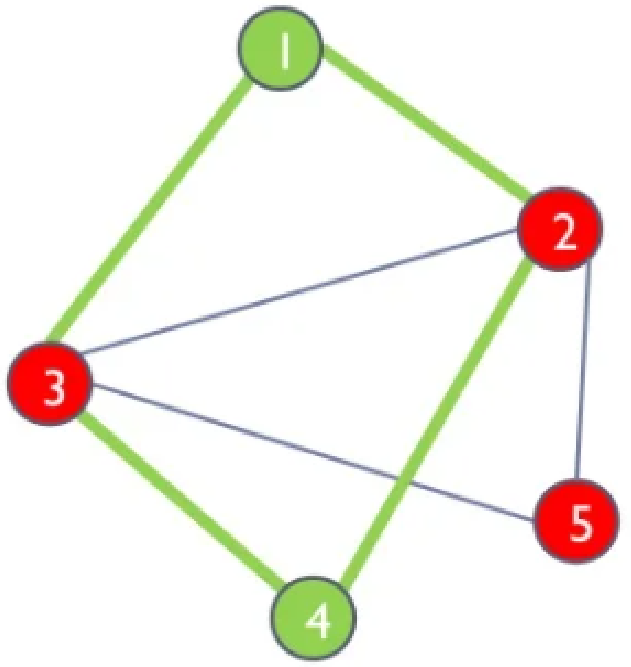
Note: The term 'eigenvector' comes from mathematics (matrix algebra), but it is not necessary for understanding how to interpret this measure

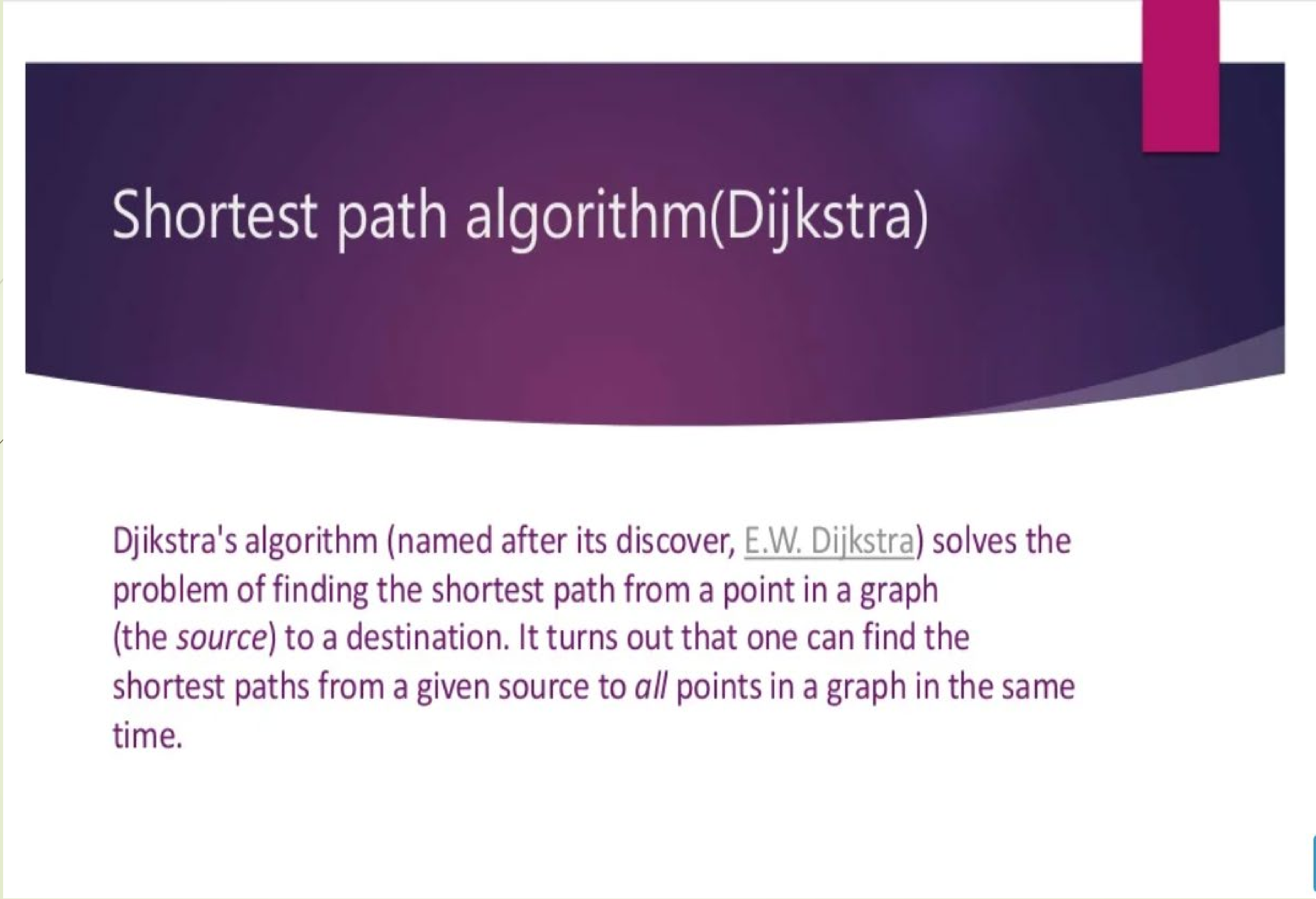
Paths and shortest paths

🟩 Shortest path(s)

- A *path* between two nodes is any sequence of non-repeating nodes that connects the two nodes
- The *shortest path* between two nodes is the path that connects the two nodes with the shortest number of edges (also called the *distance* between the nodes)
- In the example to the right, between nodes 1 and 4 there are two shortest paths of length 2: {1,2,4} and {1,3,4}
- Other, longer paths between the two nodes are {1,2,3,4}, {1,3,2,4}, {1,2,5,3,4} and {1,3,5,2,4} (the longest paths)
- Shorter paths are desirable when speed of communication or exchange is desired (often the case in many studies, but sometimes not, e.g. in networks that spread disease)

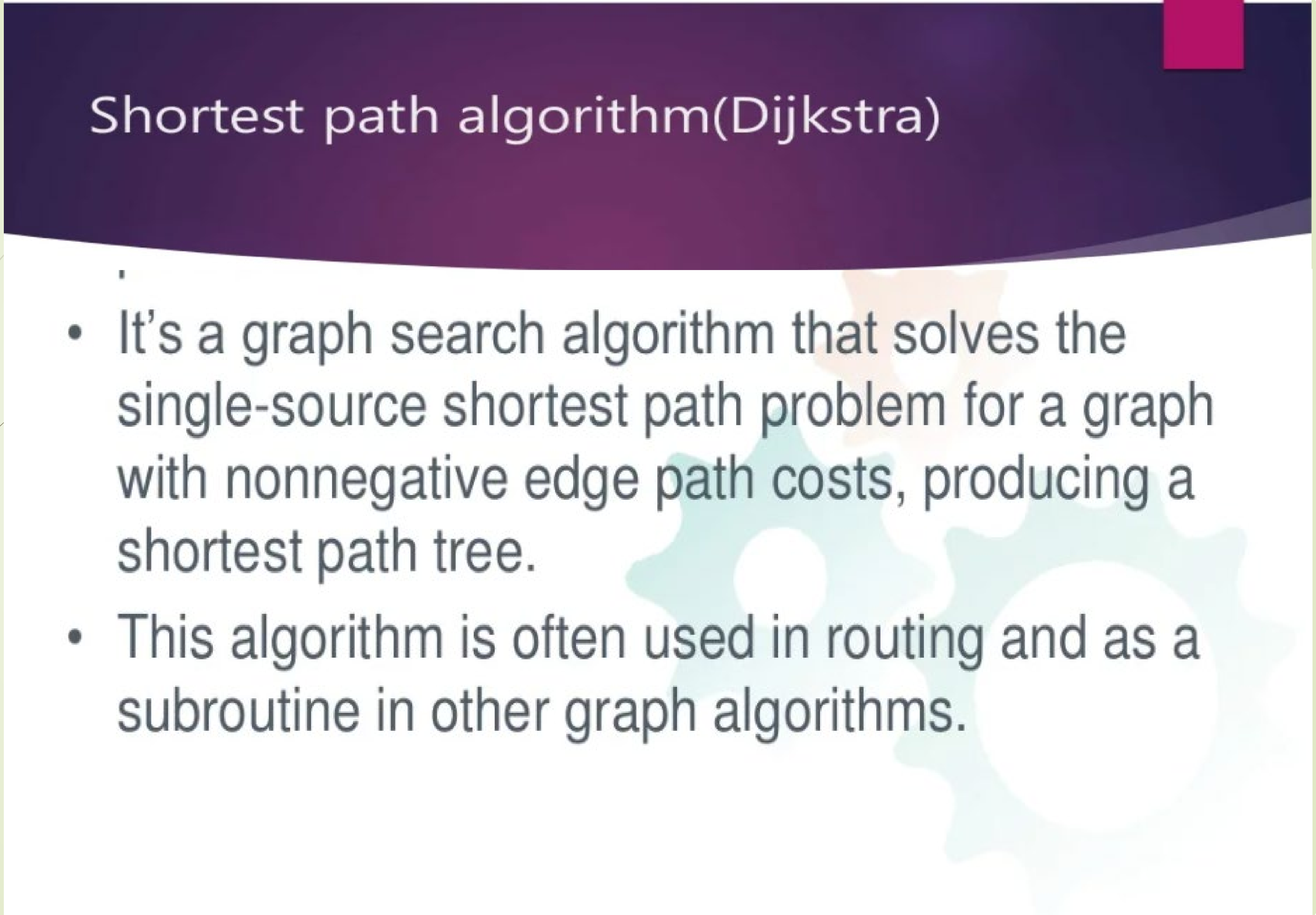
Hypothetical graph





Shortest path algorithm(Dijkstra)

Djikstra's algorithm (named after its discover, E.W. Dijkstra) solves the problem of finding the shortest path from a point in a graph (the *source*) to a destination. It turns out that one can find the shortest paths from a given source to *all* points in a graph in the same time.



Shortest path algorithm(Dijkstra)

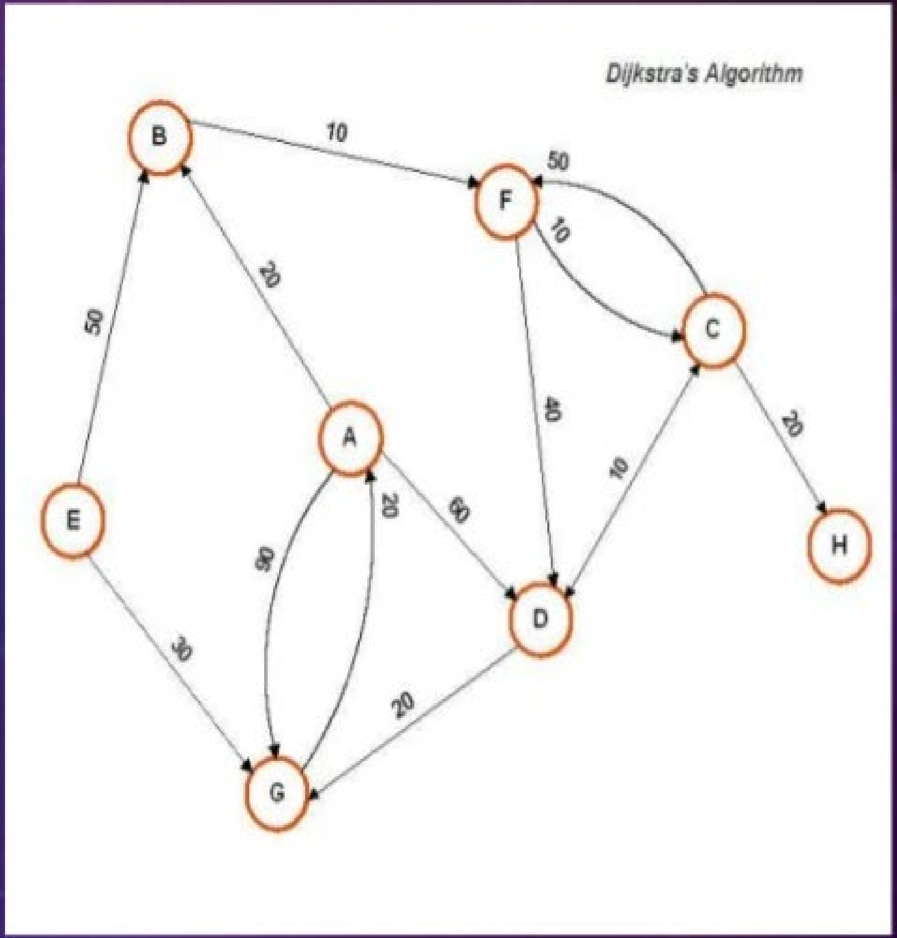
- It’s a graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path tree.
- This algorithm is often used in routing and as a subroutine in other graph algorithms.

How it works ?

- This algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities.
- According to this algorithm, to solve a given problem, we need to solve different parts of problems.

Graph Algorithm

- In this interconnected 'Vertex' we'll use 'Dijkstra's Algorithm'.
- To use this algorithm in this network we have to start from a decided vertex and then continue to others.



Dijkstra’s algorithm (Greedy)

- ❑ Dijkstra's algorithm has many variants but the most common one is to find the shortest paths from the source vertex to all other vertices in the graph.

Single Source Shortest Path Algorithm

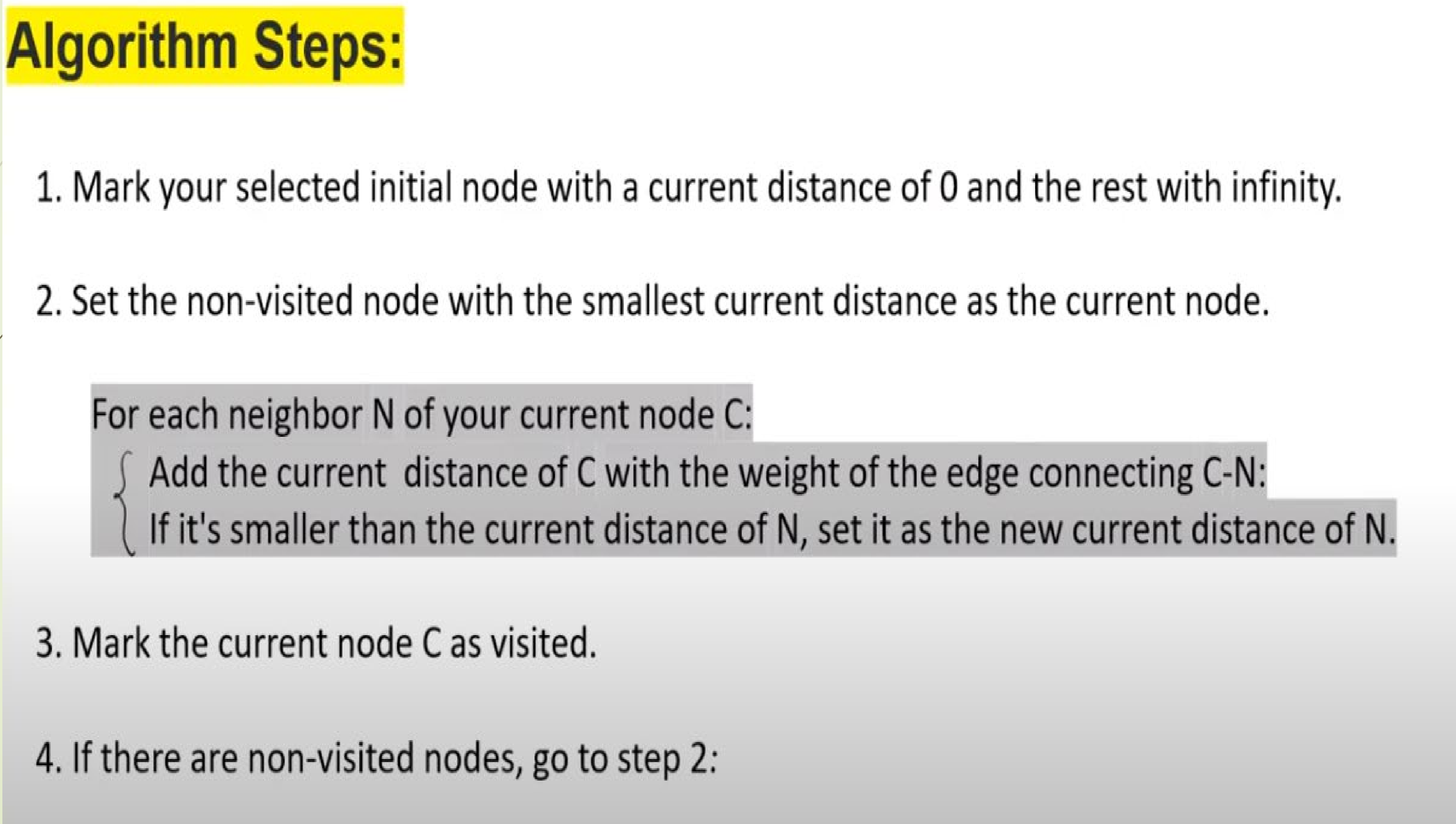
- ❑ Greedy strategy can be applied:

  Problem is solved in stages by taking one step and considering one output at a time to get the optimal solution.

- Main idea:
  - Shortest path vertex is selected next.
  - Update the shortest path of other vertices.

- ❑ Any vertex can be selected as a source
- ❑ **It can be applied to directed and undirected connected Graphs**
  - Dijkstra doesn’t work for Graphs with negative weight edges.

Shortest path algorithm(Dijkstra)



Algorithm Steps:

1. Mark your selected initial node with a current distance of 0 and the rest with infinity.

2. Set the non-visited node with the smallest current distance as the current node.

    For each neighbor N of your current node C:

    $\left\{\begin{array}{l}\text{Add the current distance of C with the weight of the edge connecting C-N:}\\ \text{If it's smaller than the current distance of N, set it as the new current distance of N.}\end{array}\right.$

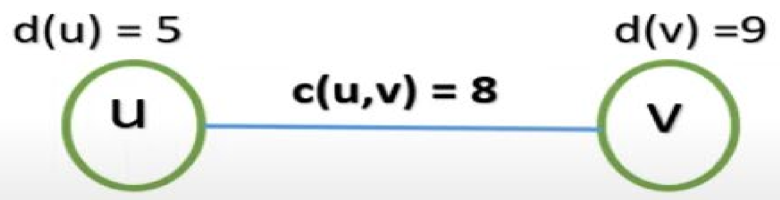
3. Mark the current node C as visited.

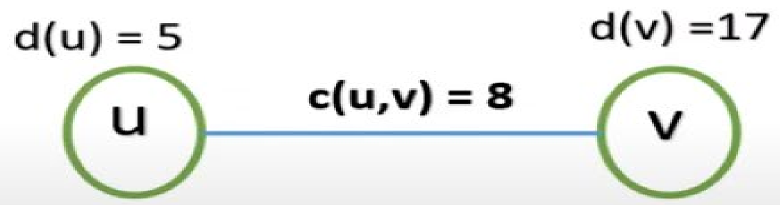
4. If there are non-visited nodes, go to step 2:

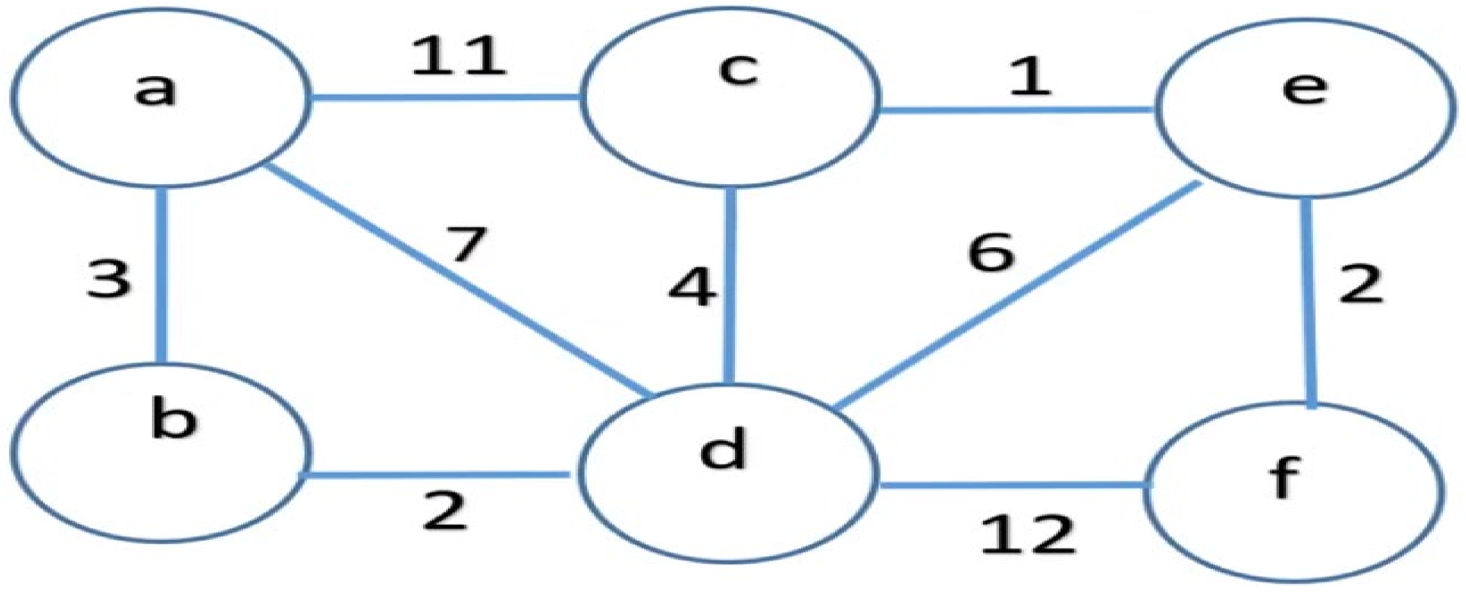
Relaxation Process

If (d(u) + c(u,v) < d(v)) Then

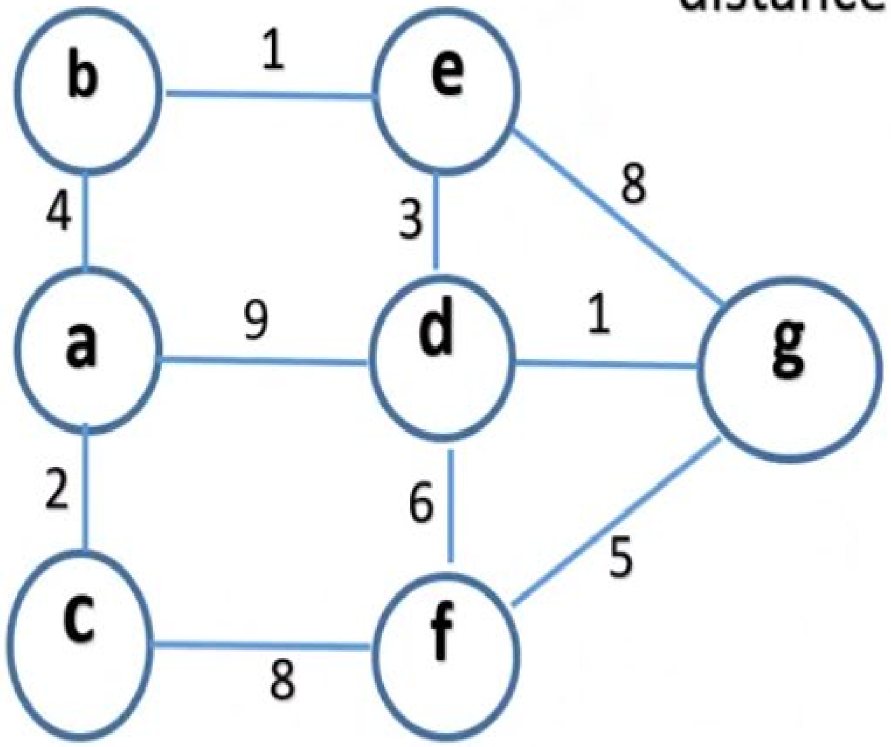
d(v) = d(u) + c(u, v)







Shortest Path in a Graph -Dijkstra's algorithm



V = {a, b, c, d, e, f, g)

distance map

ab	ac	ad	be	cf	de	df	dg	eg	fg
4	2	9	1	8	3	6	1	8	5

Heap map

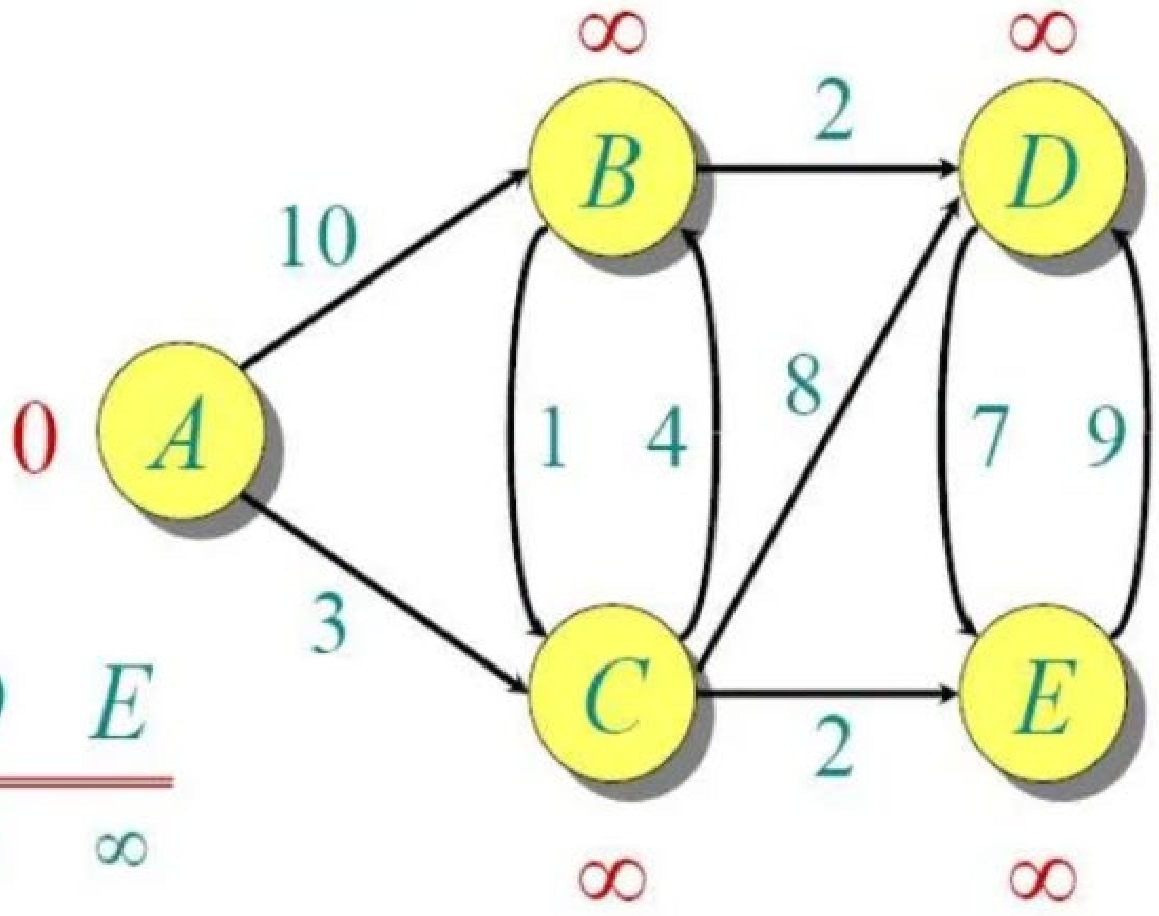
a	0
b	4
c	2
d	8
e	5
f	10
g	9

Path map

v	P
a	-
b	a
c	a
d	e
e	b
f	c
g	d

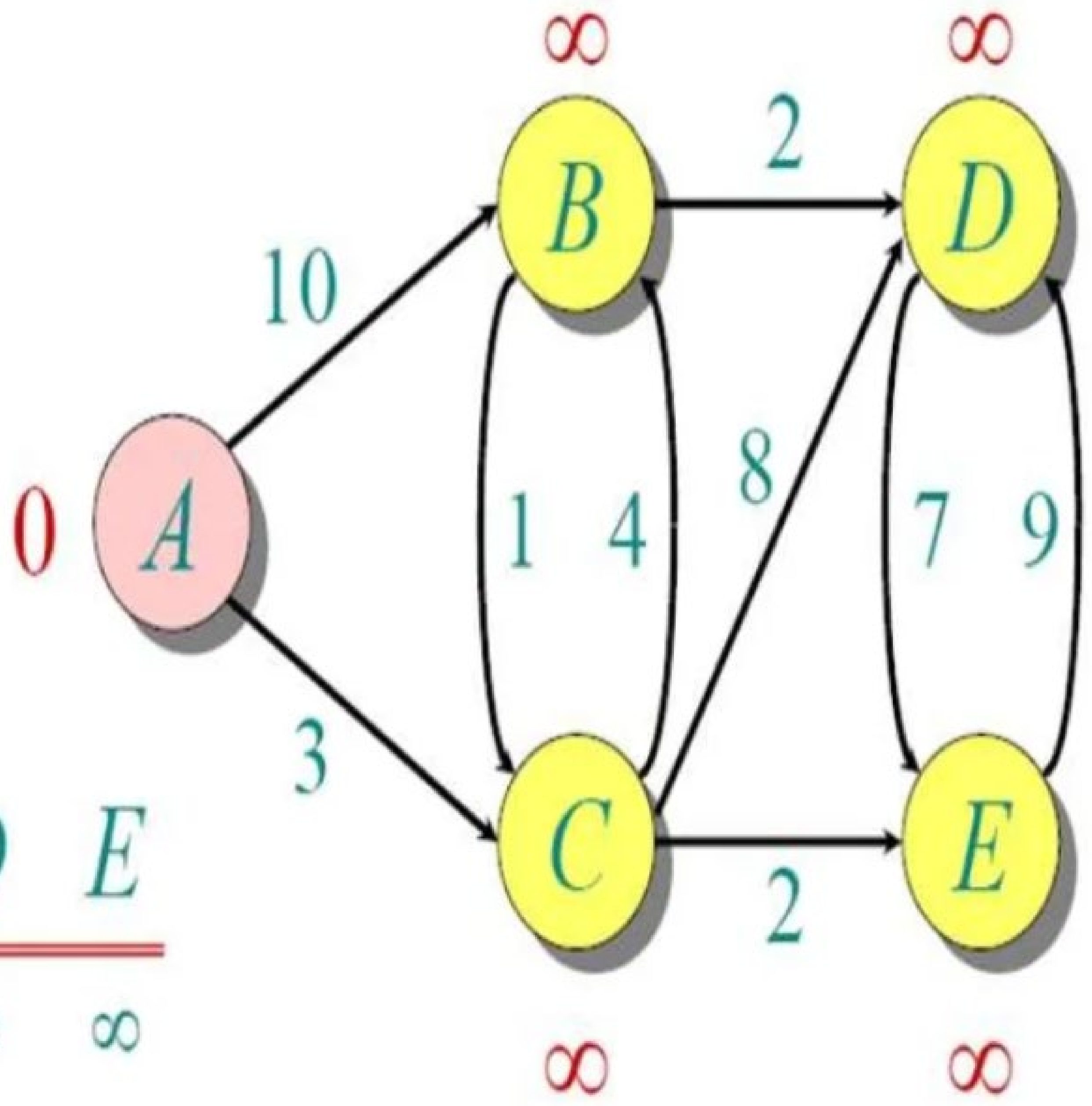
Dijkstra Animated Example

Initialize:

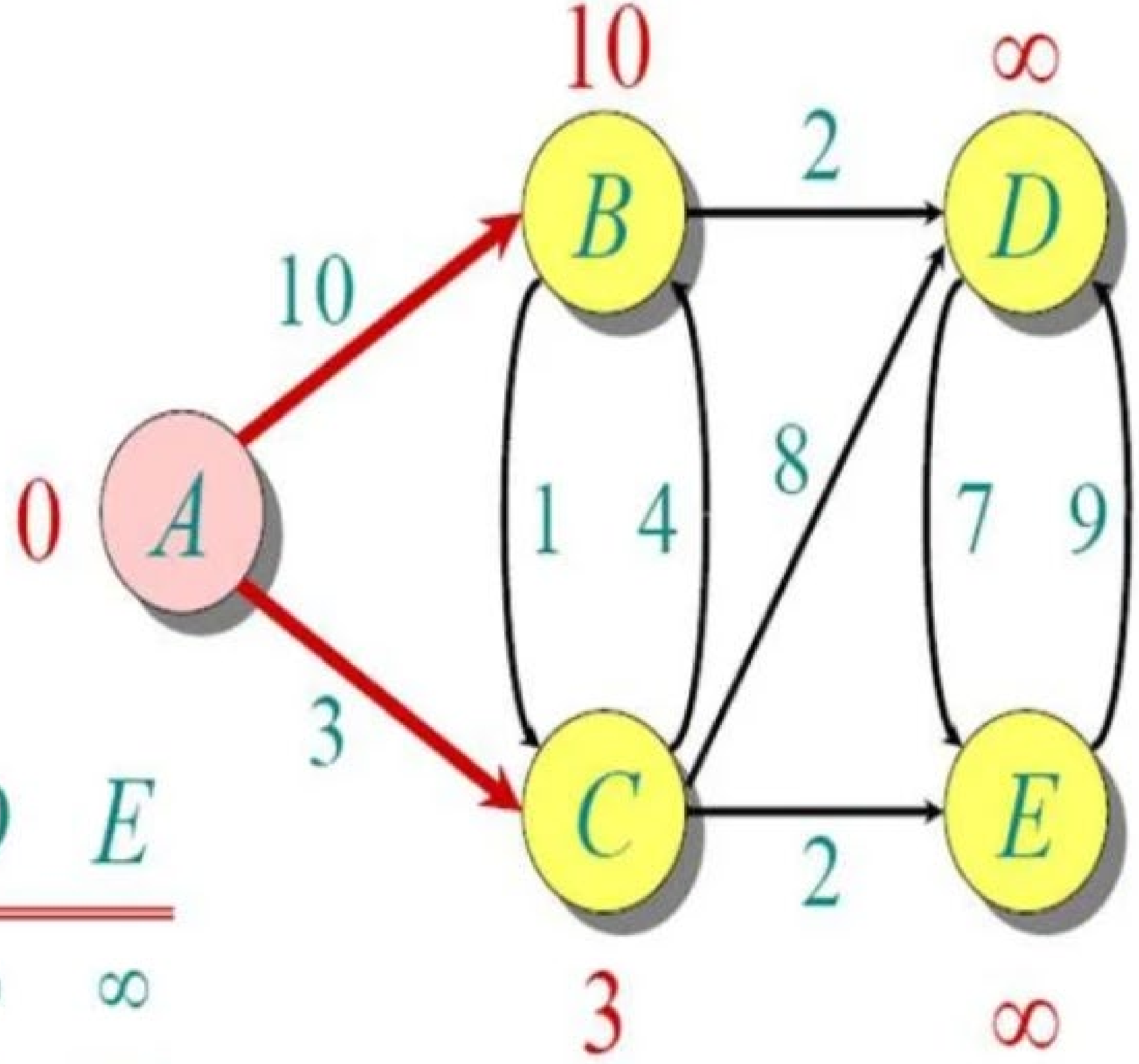


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

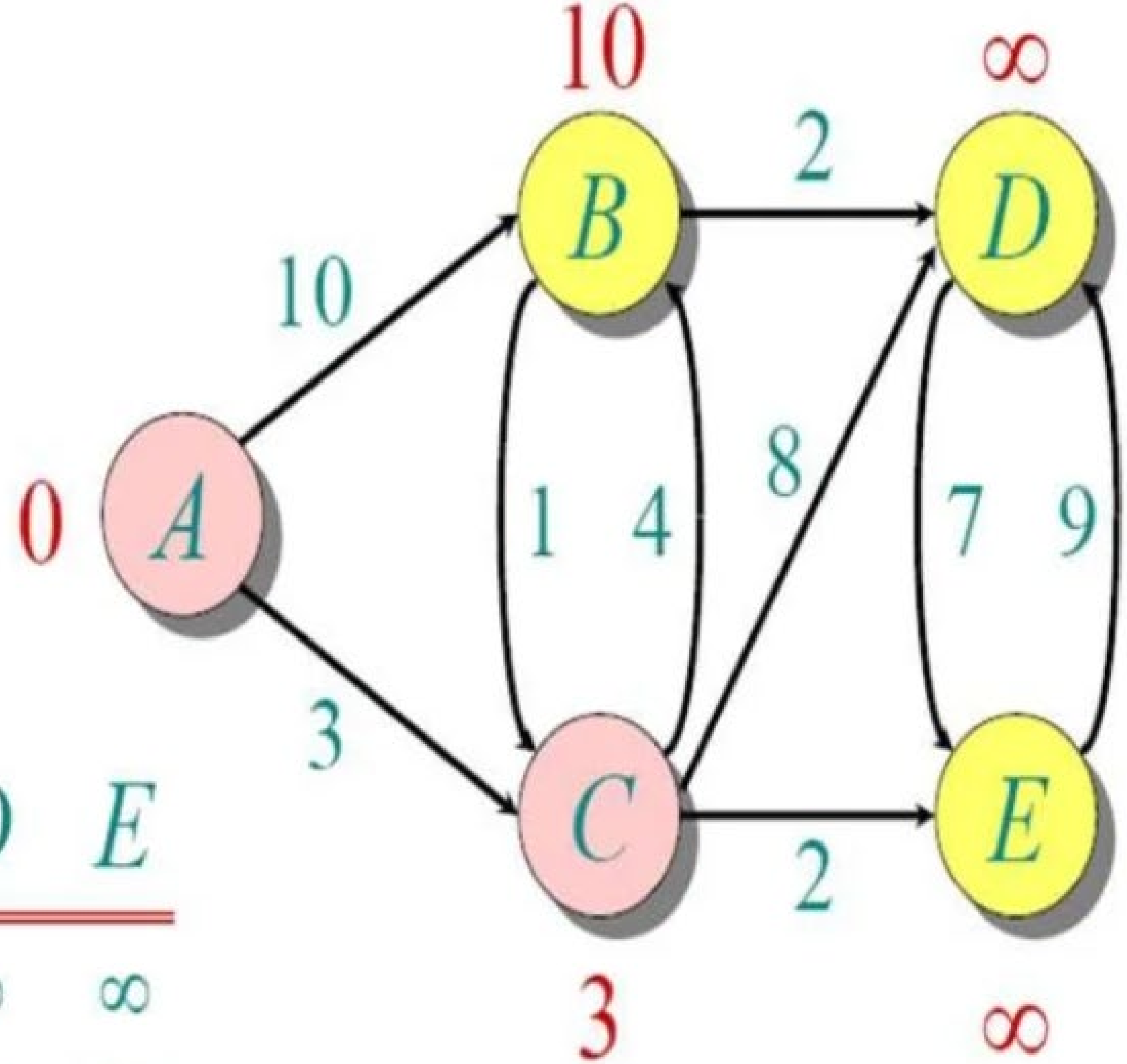


Q:	A	B	C	D	E
	0	∞	∞	∞	∞



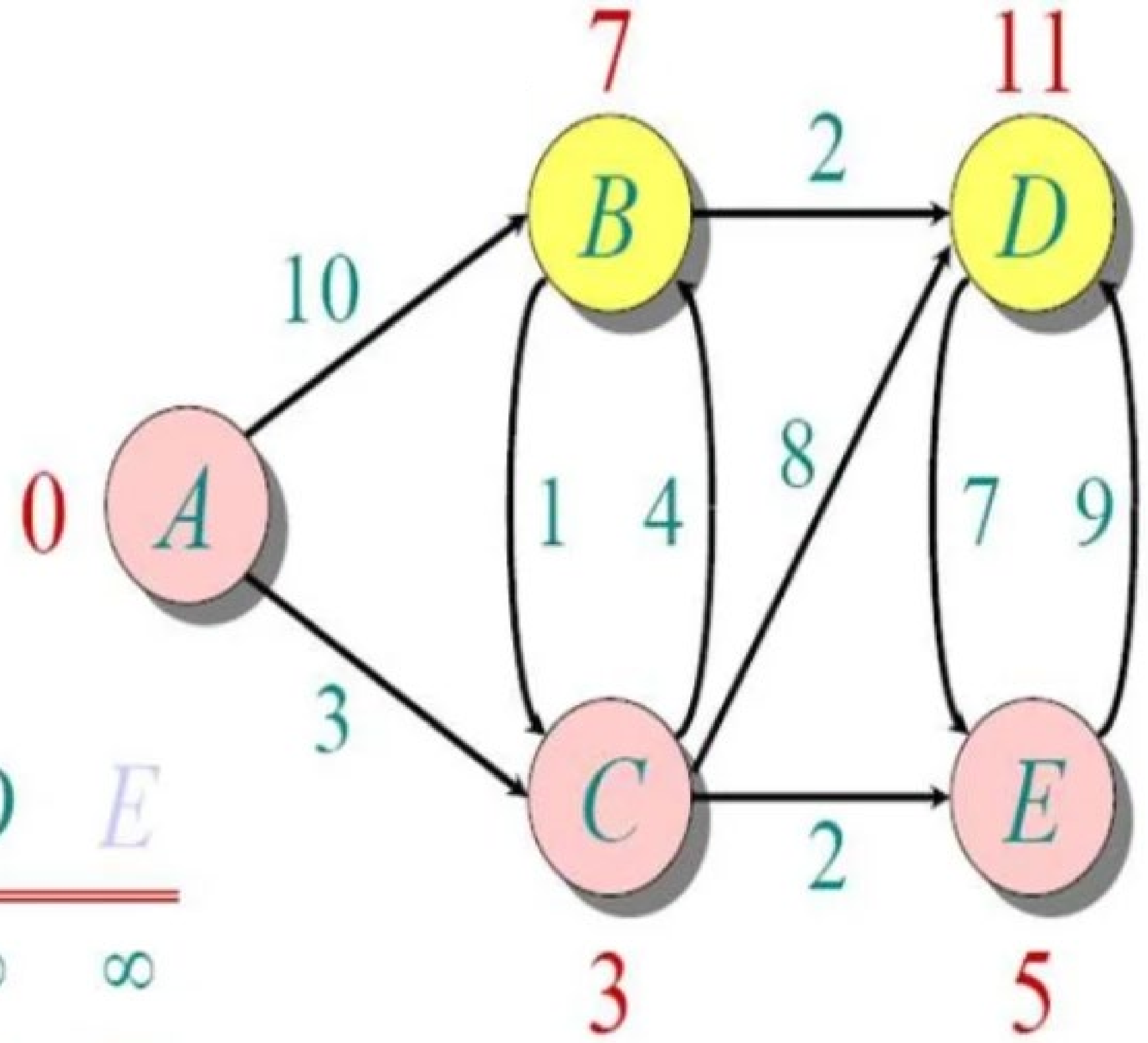
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }



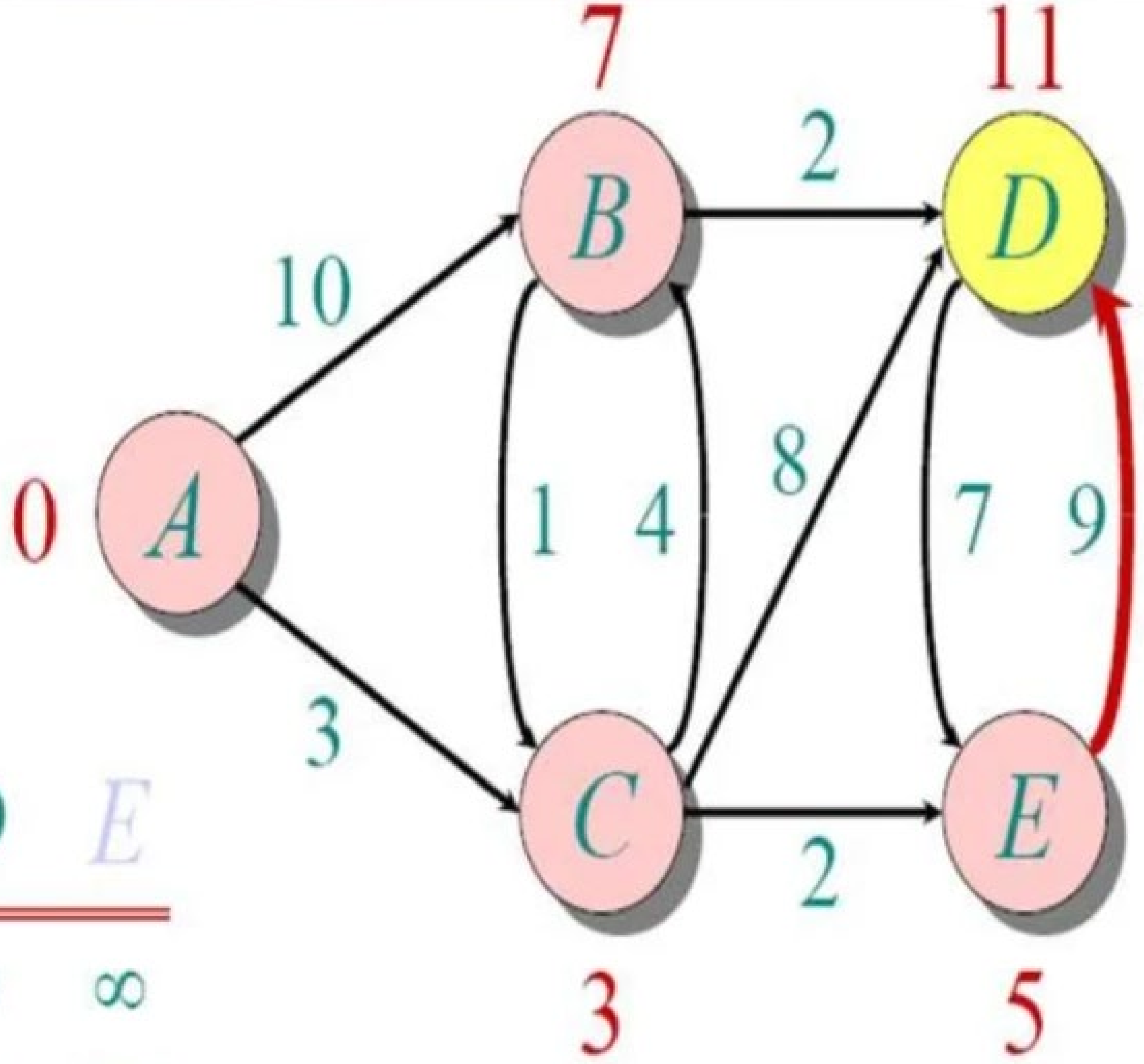
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }



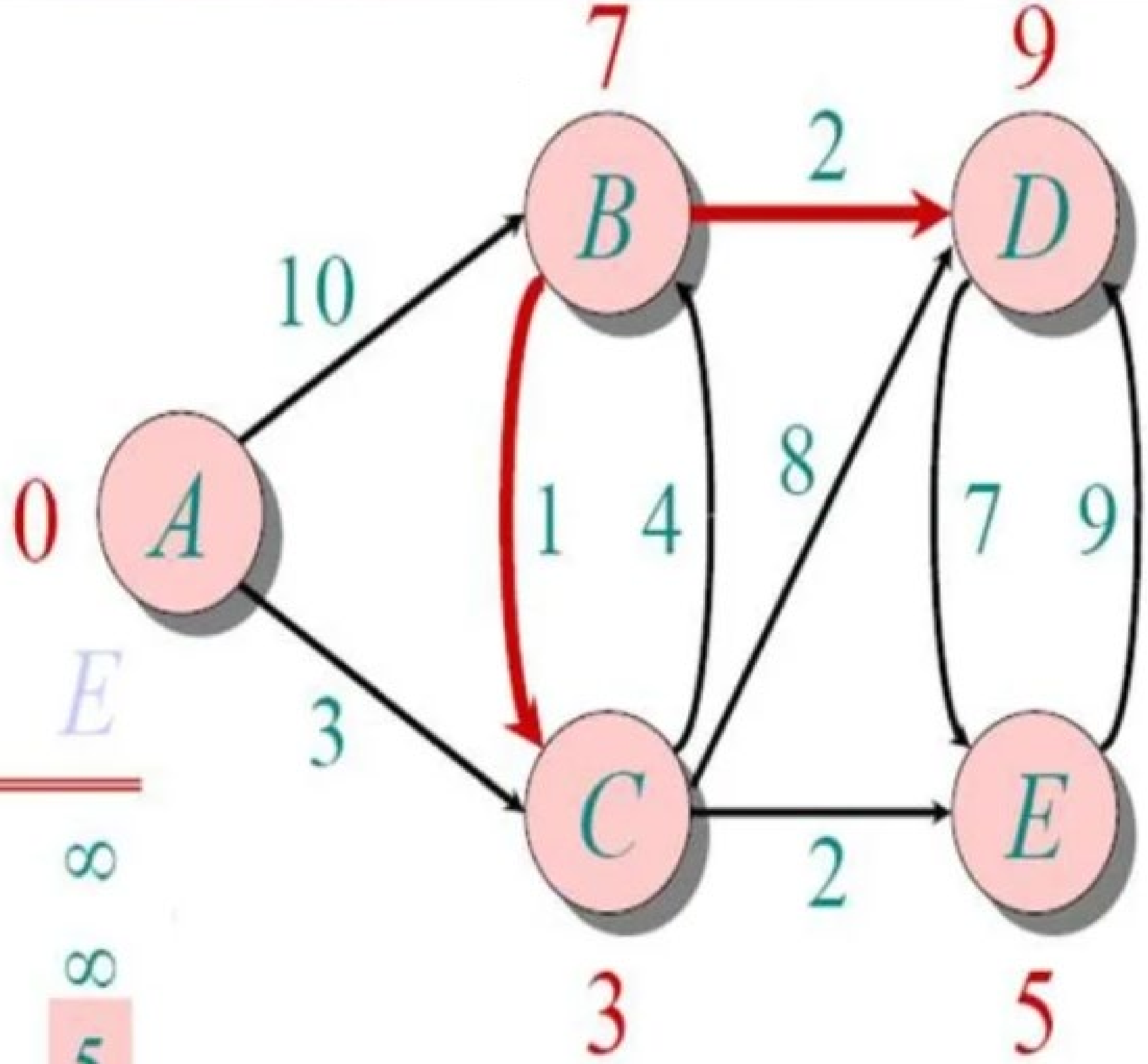
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

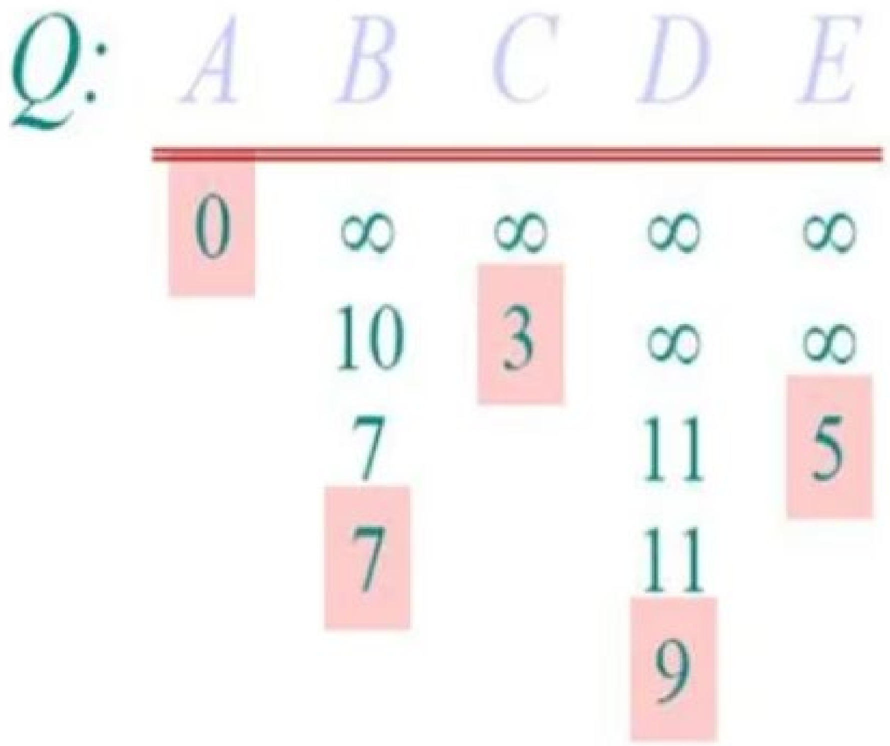
S: { A, C, E }



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E, B }



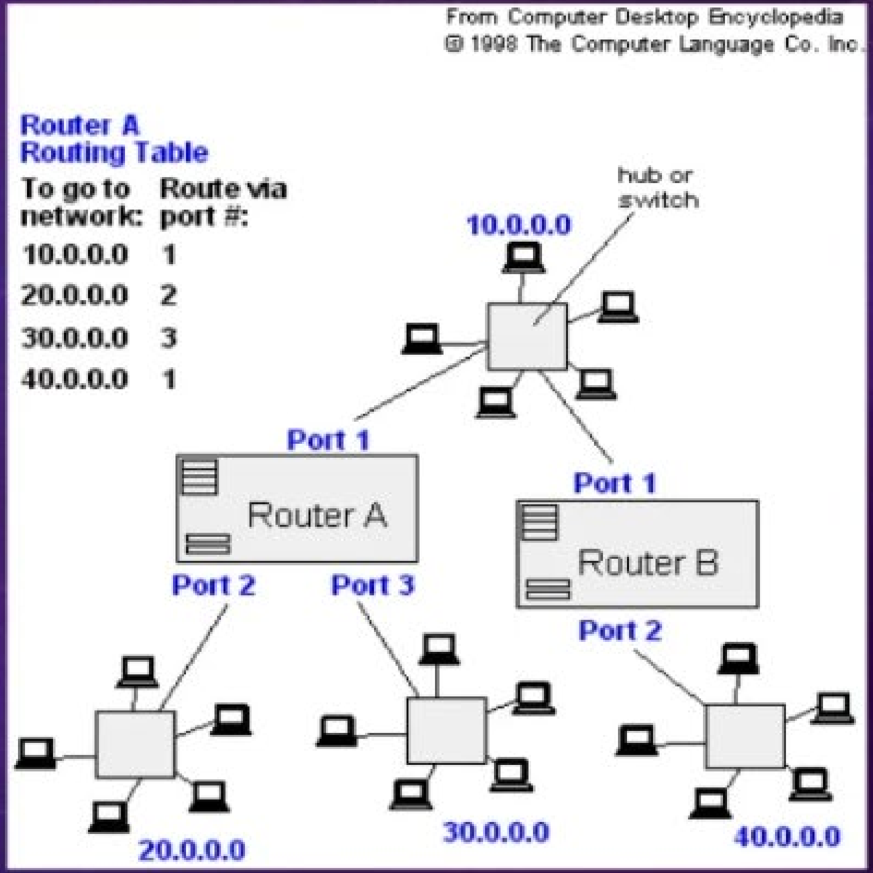


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { A, C, E, B, D }

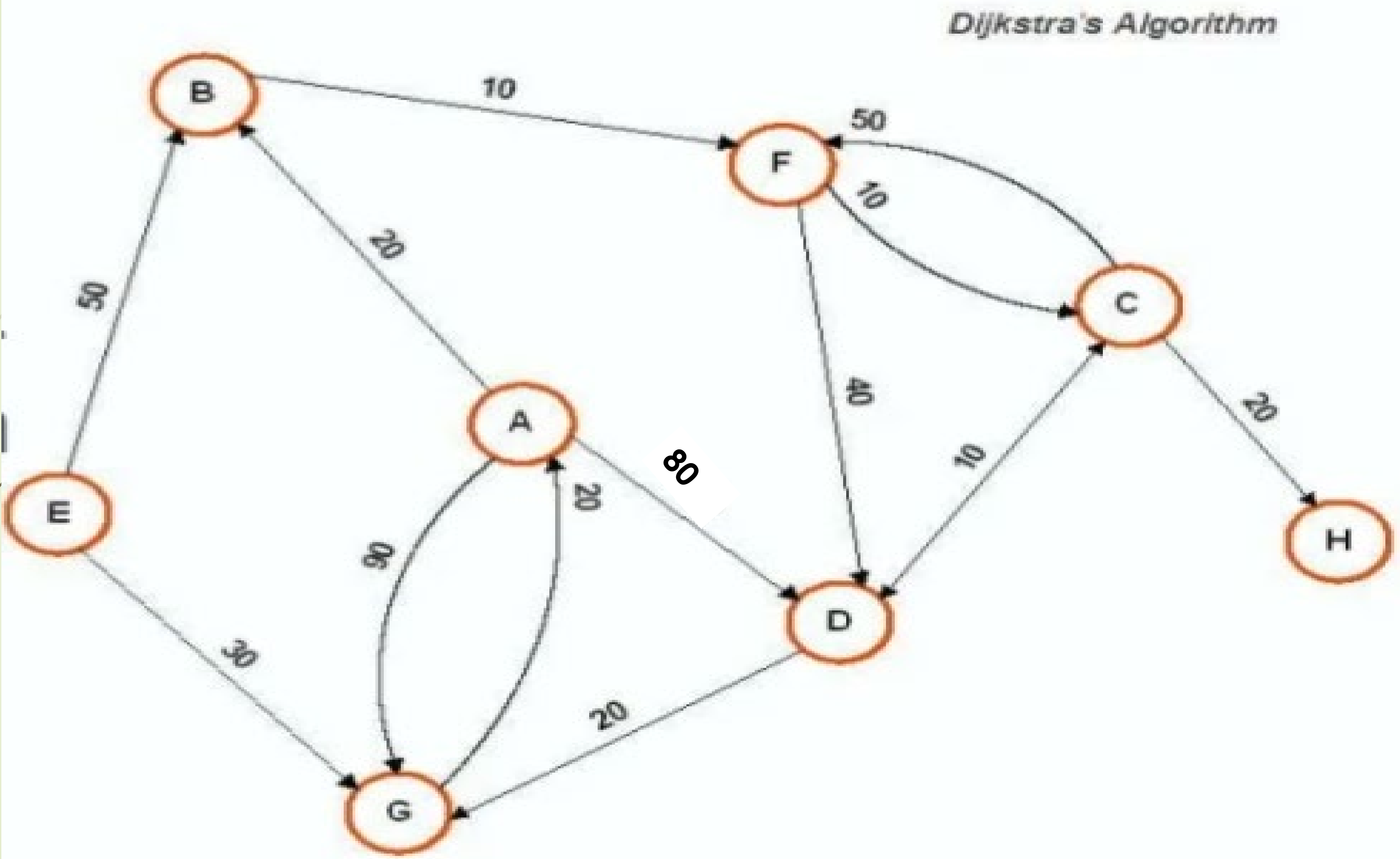
Application :

- Traffic Information Systems are most prominent use
- Mapping (Map Quest, Google Maps)
- Routing Systems

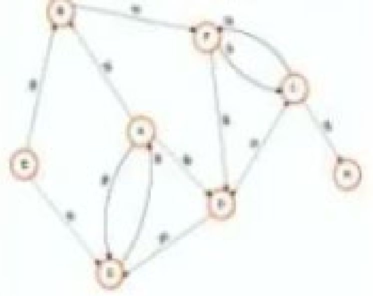

From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.

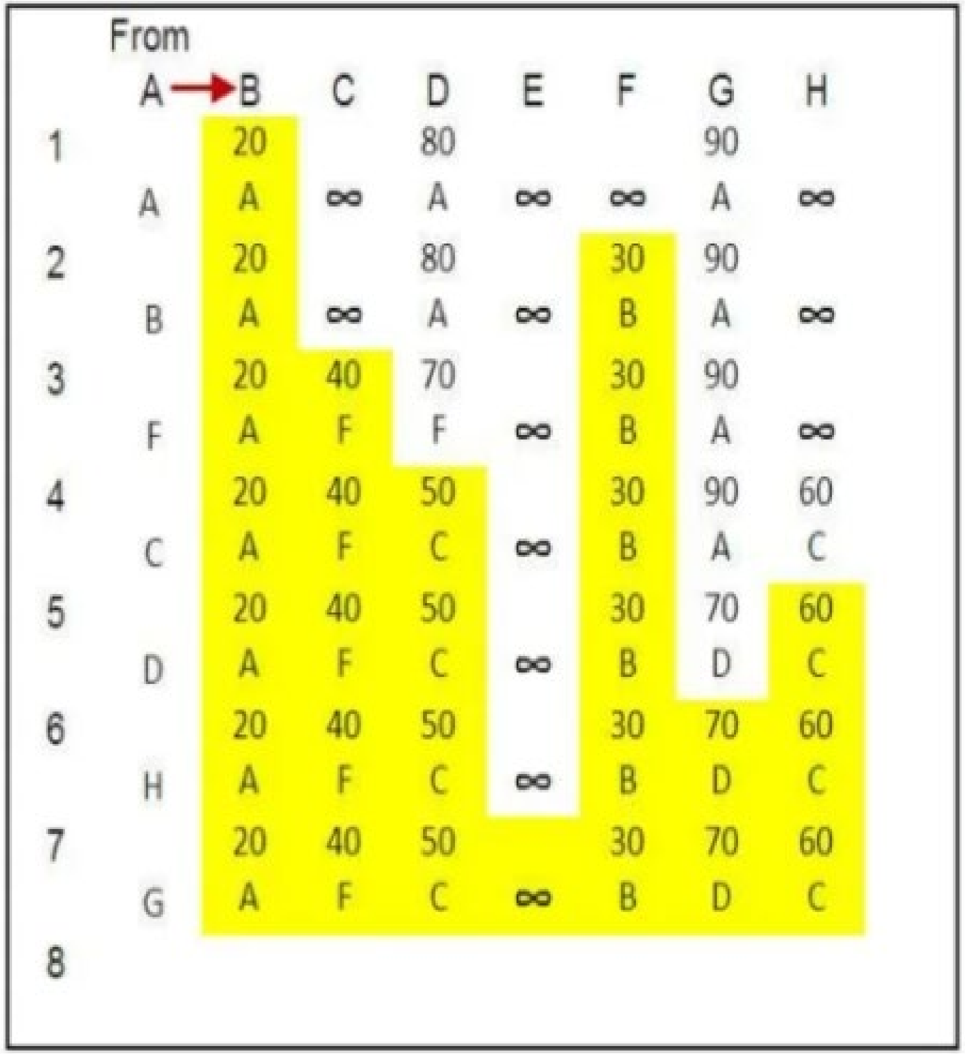
Router A
Routing Table

To go to network:	Route via port #:
10.0.0.0	1
20.0.0.0	2
30.0.0.0	3
40.0.0.0	1

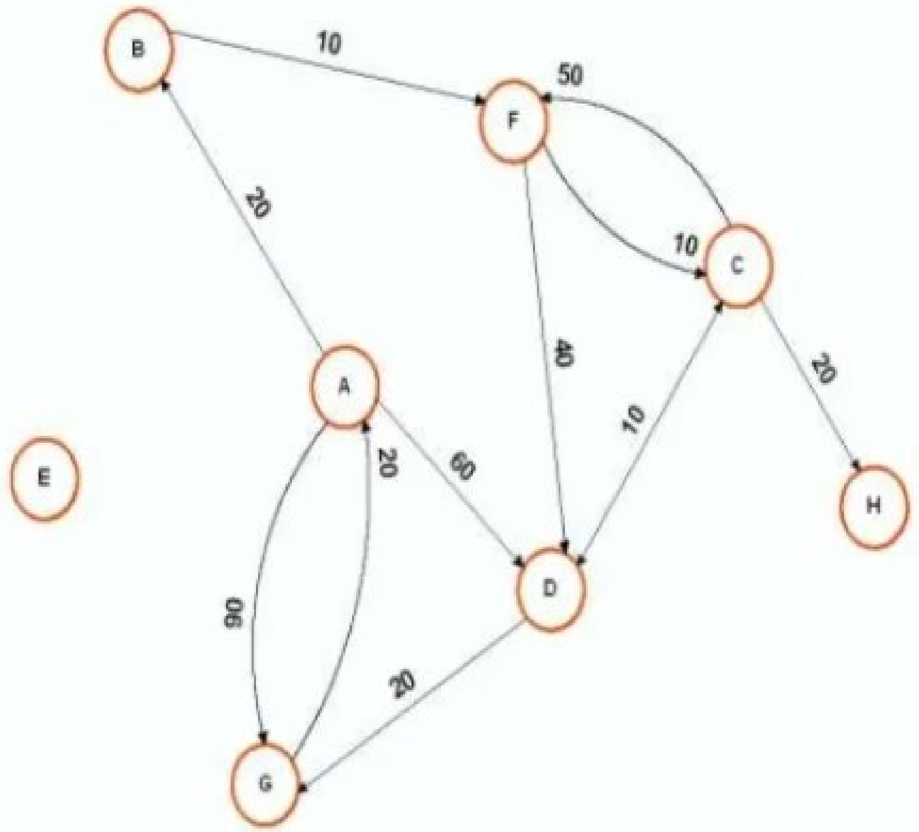

Dijkstra's Algorithm
B
F
C
A
E
D
G
H
10
50
10
20
50
40
10
20
80
20
90
30
20

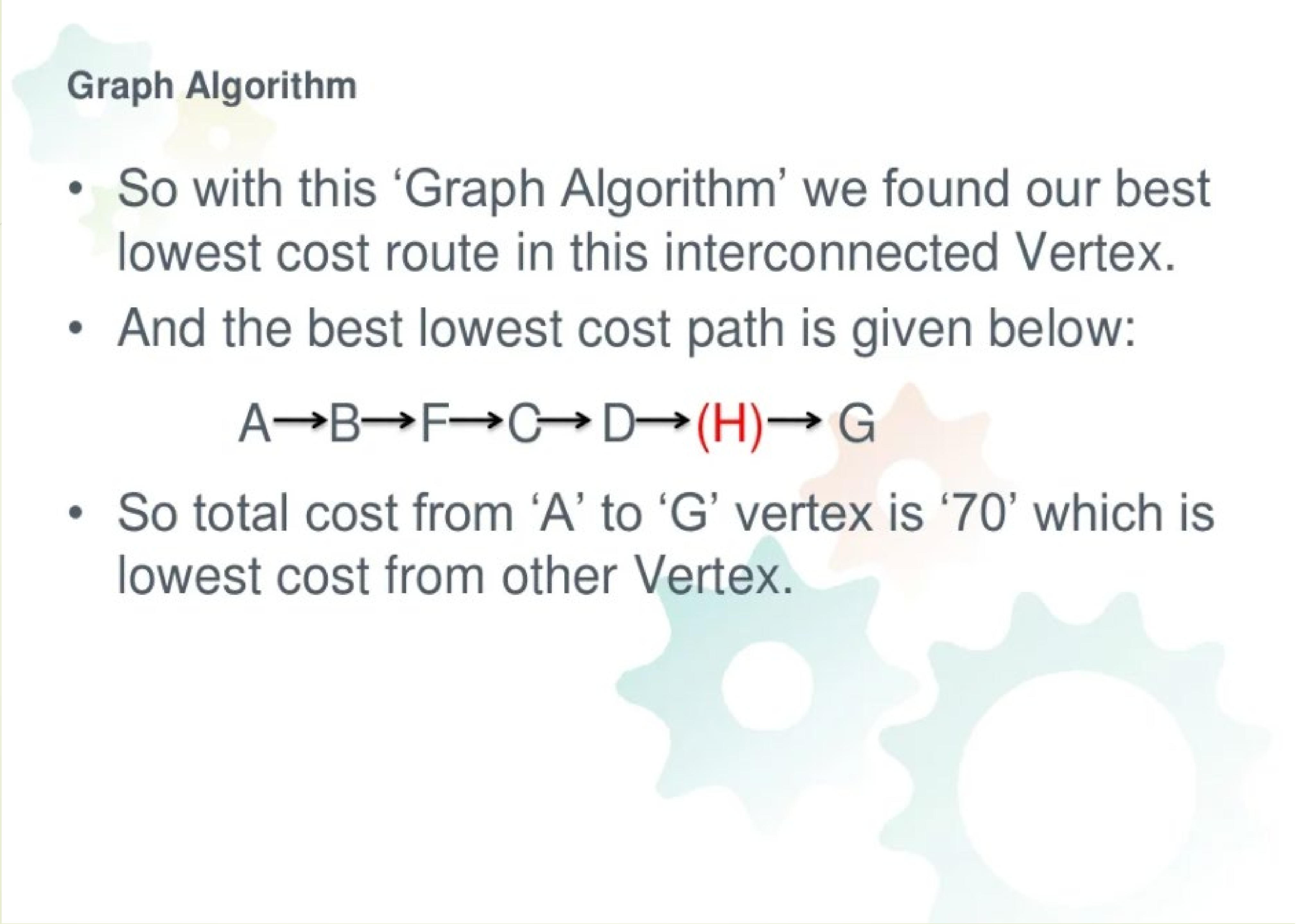
Graph Algorithm





	From							
	A →	B	C	D	E	F	G	H
1		20		80			90	
	A	A	∞	A	∞	∞	A	∞
2		20		80		30	90	
	B	A	∞	A	∞	B	A	∞
3		20	40	70		30	90	
	F	A	F	F	∞	B	A	∞
4		20	40	50		30	90	60
	C	A	F	C	∞	B	A	C
5		20	40	50		30	70	60
	D	A	F	C	∞	B	D	C
6		20	40	50		30	70	60
	H	A	F	C	∞	B	D	C
7		20	40	50		30	70	60
	G	A	F	C	∞	B	D	C
8								





Graph Algorithm

- So with this ‘Graph Algorithm’ we found our best lowest cost route in this interconnected Vertex.
- And the best lowest cost path is given below:

$$A \rightarrow B \rightarrow F \rightarrow C \rightarrow D \rightarrow (H) \rightarrow G$$

- So total cost from ‘A’ to ‘G’ vertex is ‘70’ which is lowest cost from other Vertex.

Bellman-Ford Algorithm خوارزمية بيلمان فورد

- BF is a single source shortest path algorithm
- BF is slower than Dijkstra's Algorithm, it works in the cases when the weight of the edge is negative, and it also finds negative weight cycle in the graph.

The idea of this algorithm is to relax all the edges of the graph one-by-one in some random order at most (n-1) times.

Bellman-Ford Algorithm خوارزمية بيلمان فورد

Algorithm Steps

S - Starting node   E # of edges   V # of nodes

D Array that tracks the best distance from S to all nodes

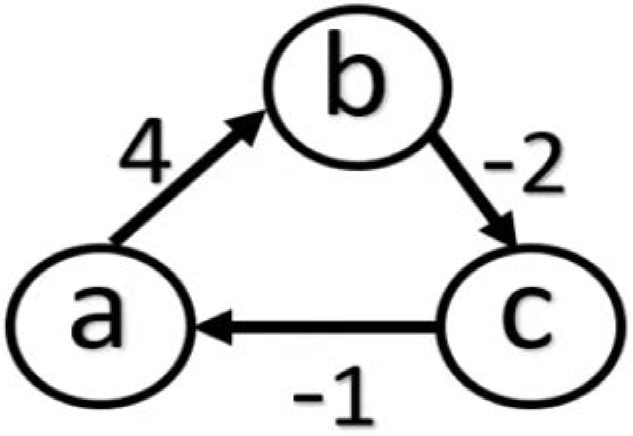
- Set D[S] to 0
- Set every entry in D to ∞
- Relax each edge V-1 times
-

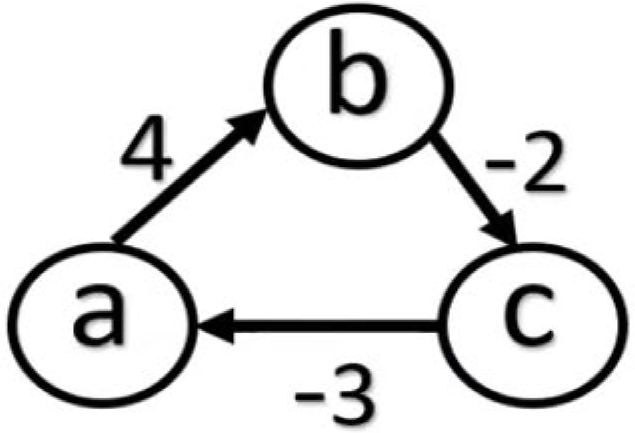
(ab), (ac), (bd), (cd), (be), (df), (fe), (bf), (cb)

$$\begin{tikzcd}
a \arrow[r, "9"] \arrow[d, "7"'] & b \arrow[r, "3"] \arrow[d, "1"'] \arrow[dr, "-1"] & e \\
c \arrow[ur, "-2"] \arrow[r, "2"'] & d \arrow[r, "1"'] & f \arrow[u, "1"']
\end{tikzcd}$$

	a	b	c	d	e	f
initially	0	∞	∞	∞	∞	∞
1	0	5	7	9	11	8
2	0	5	7	6	8	4
3	0	5	7	6	5	4
4	0	5	7	6	5	4
5						

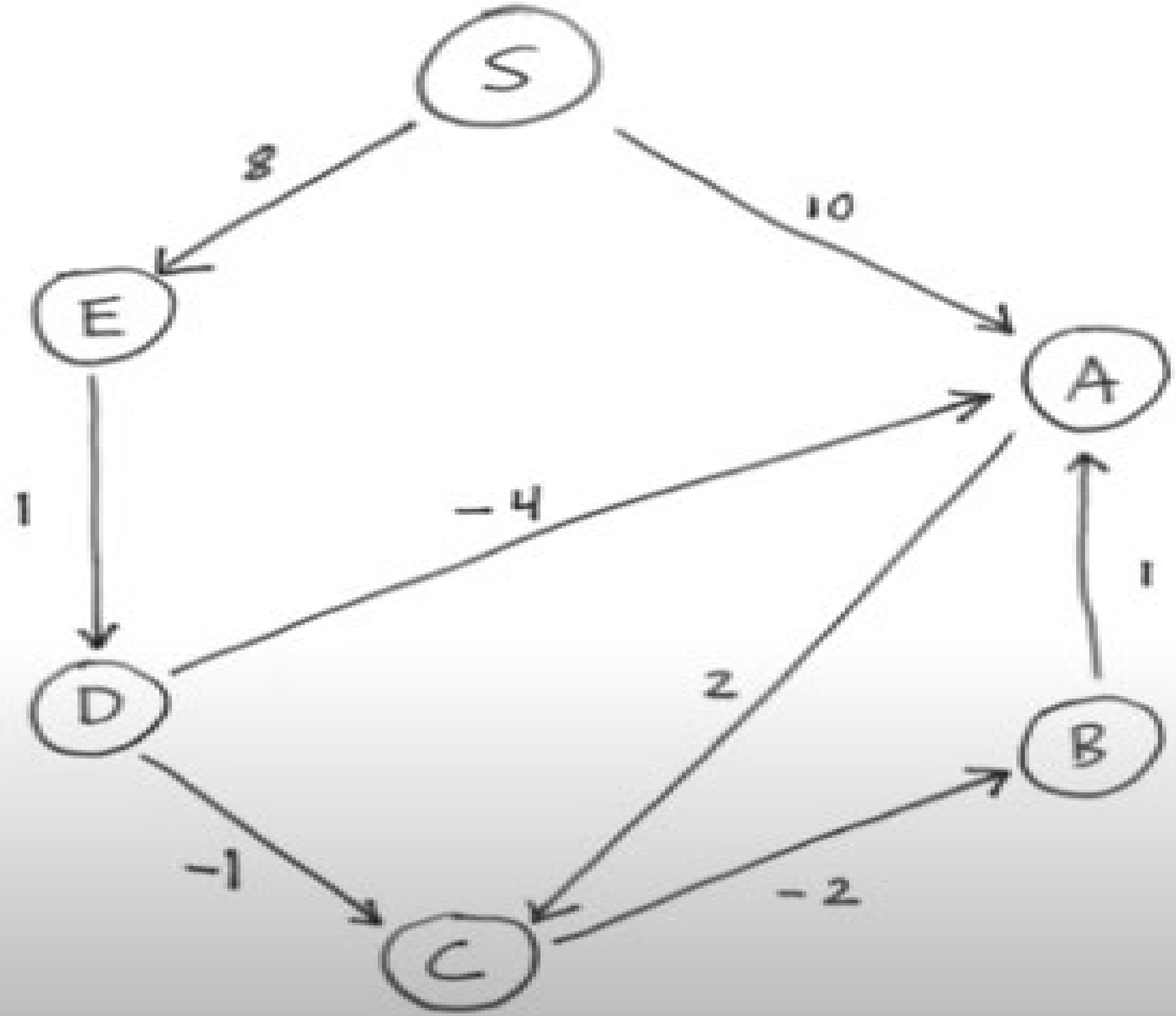
Bellman-Ford Algorithm





Negative weight cycle in the Graph

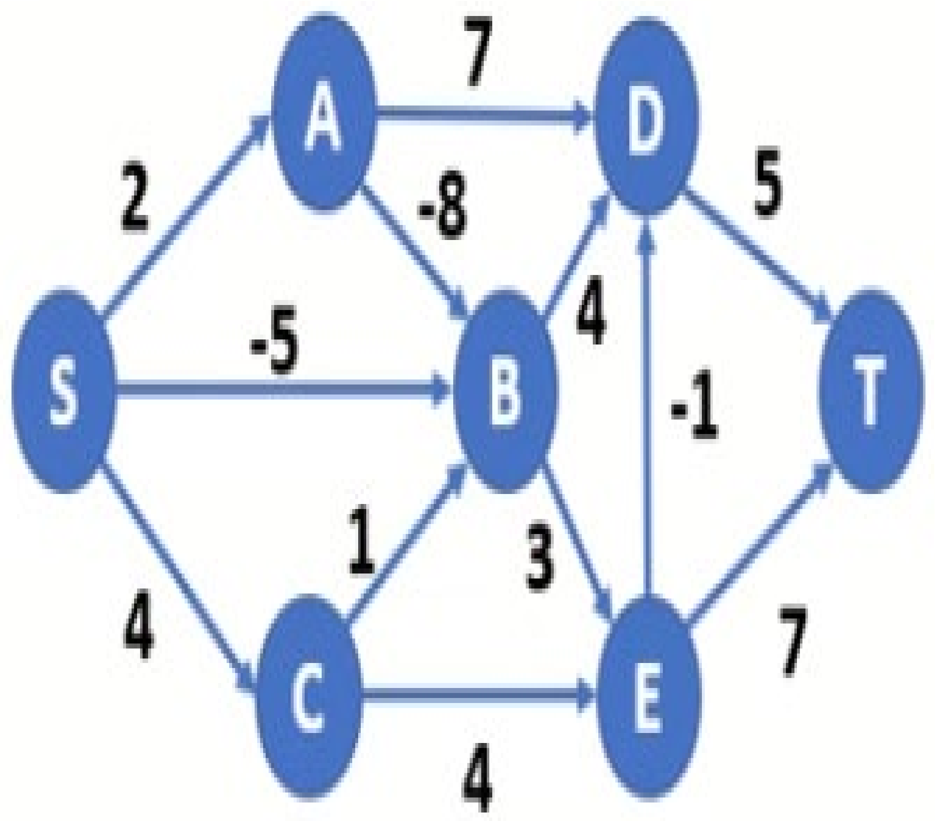
Quiz



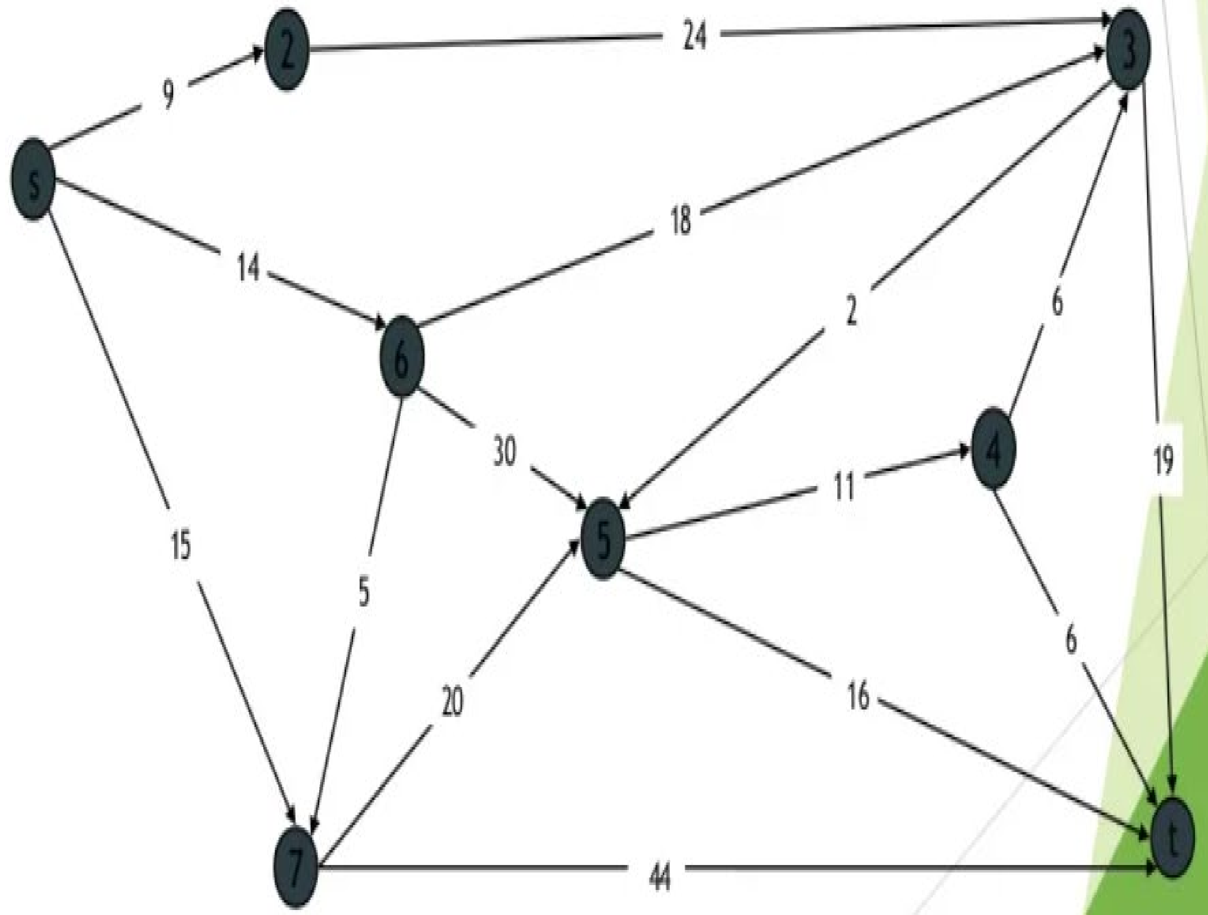
Find shortest path from s To t ?

H.W / Write a program to represent the Dijkstra's algorithm and Bellman-ford Algorithm ?

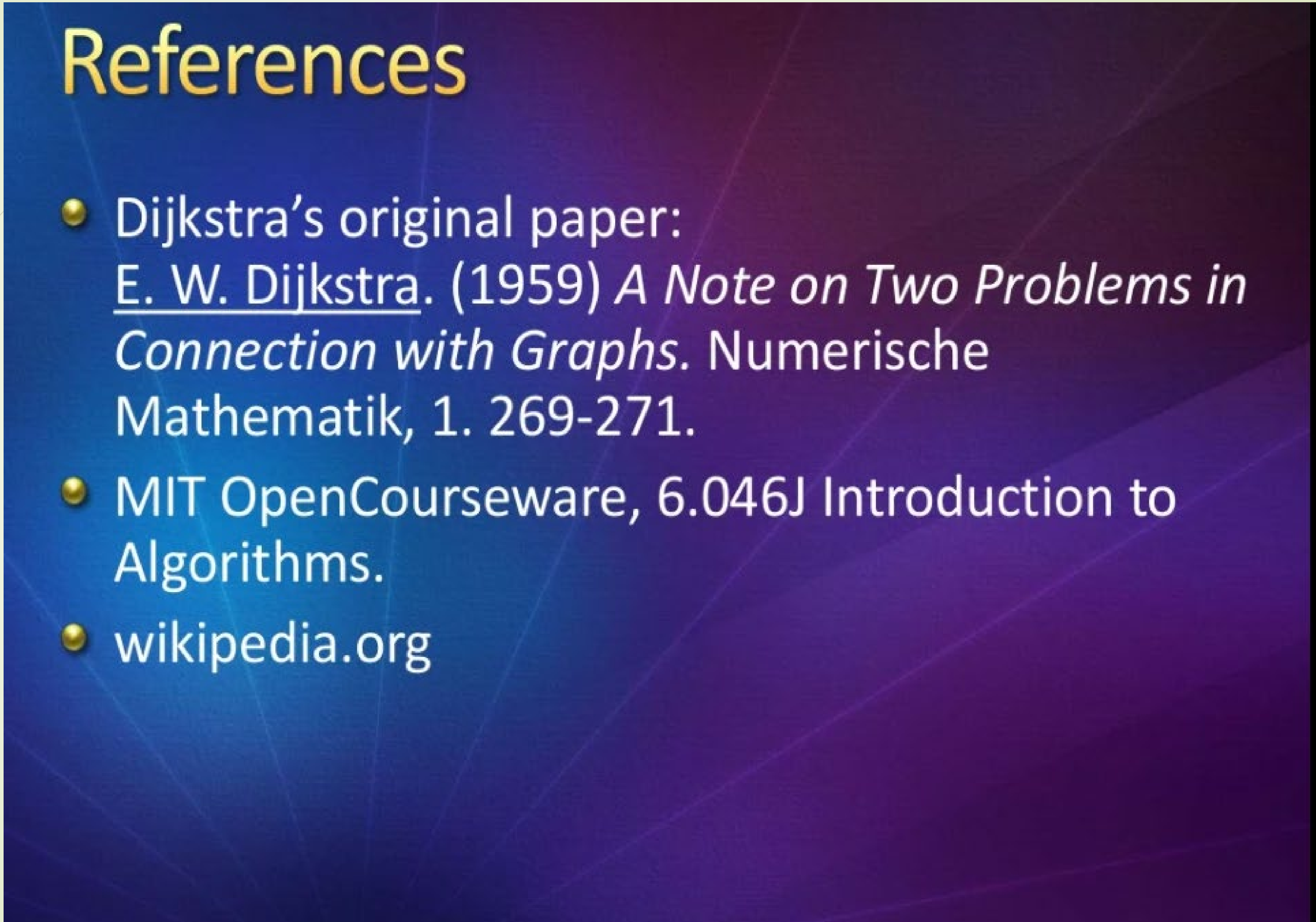
H.W



Bellman-Ford Algorithm



Dijkstra's algorithm



References

- Dijkstra's original paper:
  E. W. Dijkstra. (1959) *A Note on Two Problems in Connection with Graphs*. Numerische Mathematik, 1. 269-271.
- MIT OpenCourseware, 6.046J Introduction to Algorithms.
- wikipedia.org

شكرًا للاستماعِ والمشاركة

:: النجاح يساوي الأهداف ::
و كل ما عداه كماليات

براين تريسي