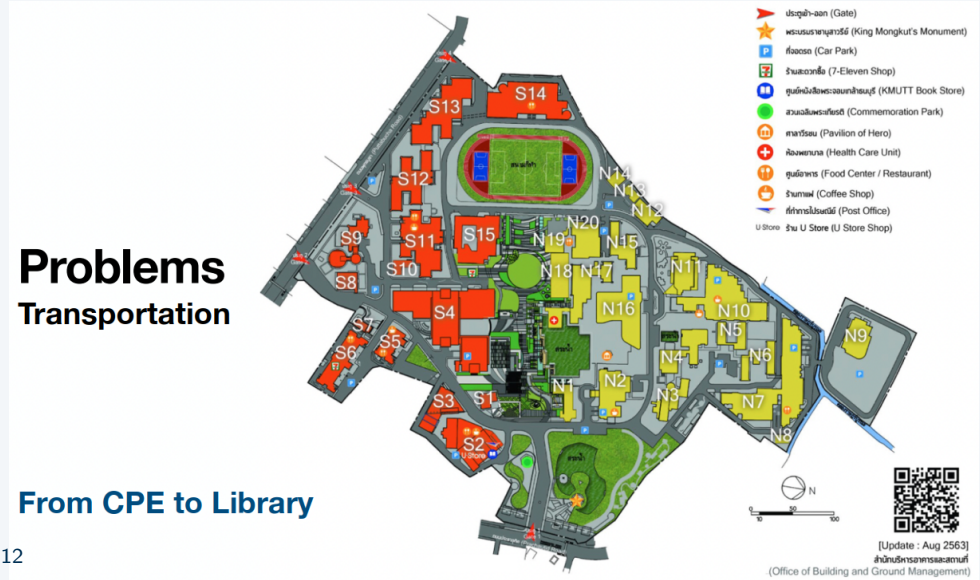
Graphs

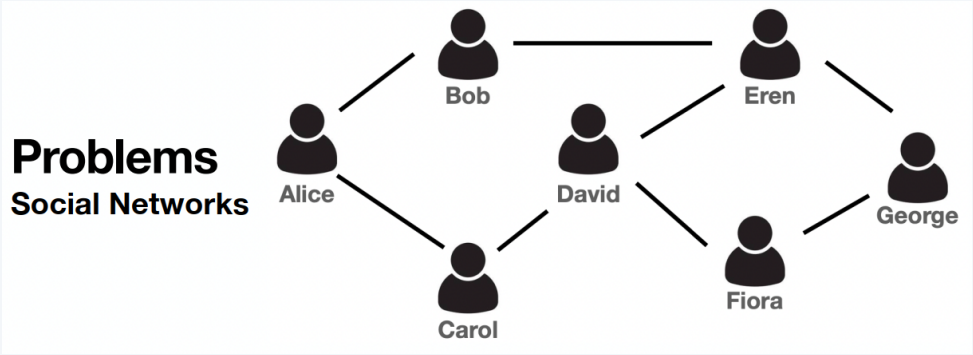


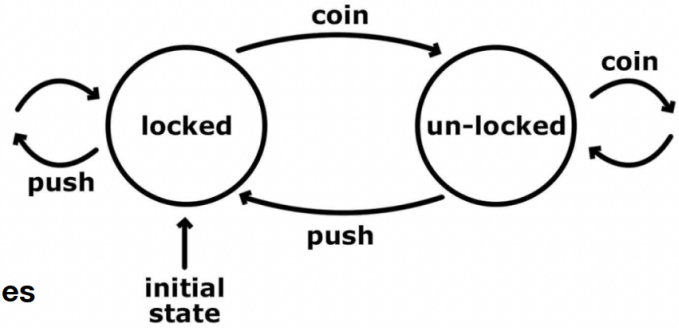
Problems

Transportation

From CPE to Library

Examples

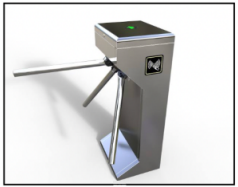

Problems
Social Networks
Bob
Eren
Alice
David
George
Fiora
Carol



Problems

Finite State Machines (FSM)

Source: www.smashingmagazine.com



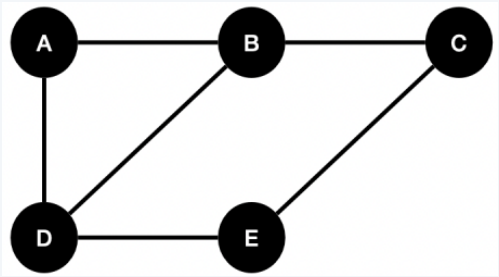
Source: www.alibaba.com

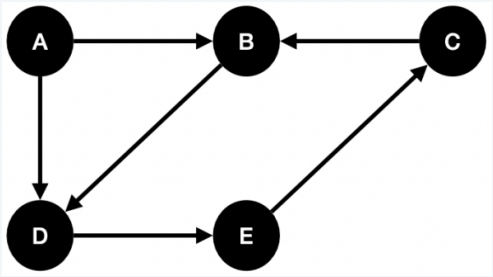
Graph

Definition

A graph G is defined as an ordered set (V, E), where

- $V(G)$ represents the set of vertices,
- $E(G)$ represents the edges that connect these vertices.





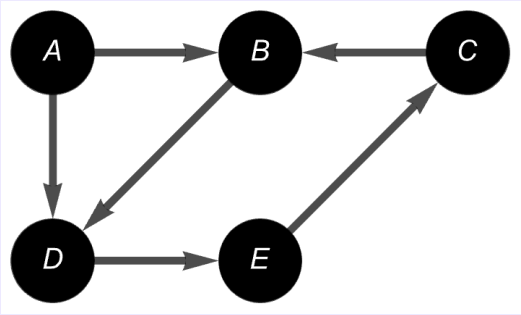
Representation of Graphs

Representation of Graphs

- Sequential representation: adjacency matrix
- Linked representation: adjacency list
- Adjacency multi-list: extension of linked representation

Adjacency Matrix

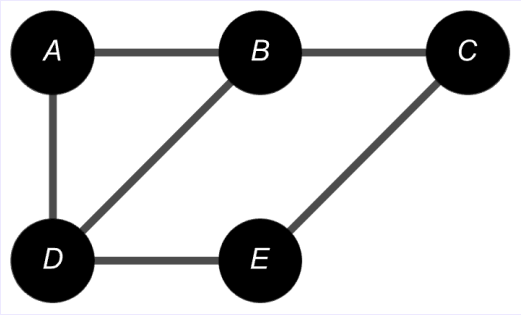
Directed graph



$$
\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array}
\begin{array}{c}
\begin{array}{ccccc} A & B & C & D & E \end{array} \\
\left[\begin{array}{ccccc}
0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0
\end{array}\right]
\end{array}
$$

Adjacency Matrix

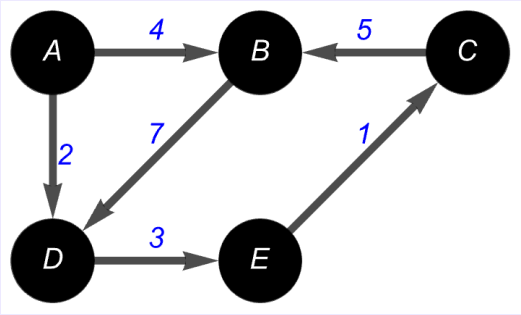
Undirected graph



	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	1	0
C	0	1	0	0	1
D	1	1	0	0	1
E	0	0	1	1	0

Adjacency Matrix

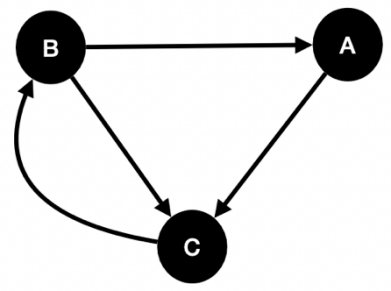
Weighted graph



	A	B	C	D	E
A	0	4	0	2	0
B	0	0	0	7	0
C	0	5	0	0	0
D	0	0	0	0	3
E	0	0	1	0	0

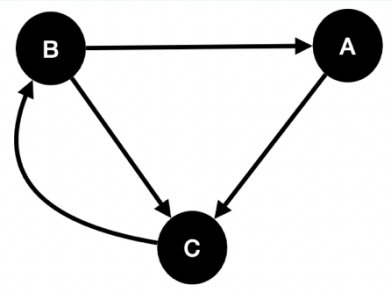
Path Matrix

For a matrix M^k, each element $m_{ij} = n$ where n is number of the paths (length $= k$) from v_i to v_j.



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

Path Matrix



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

$$M^2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

$$M^3 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Path Matrix

Transitive Closure

$$M_L = M^1 + M^2 + \ldots + M^k \text{ or } M_P = M^1 \wedge M^2 \wedge \ldots \wedge M^k$$

$$ML = M^1 + M^2 + M^3 = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 2 & 3 \\ 1 & 2 & 2 \end{bmatrix}$$

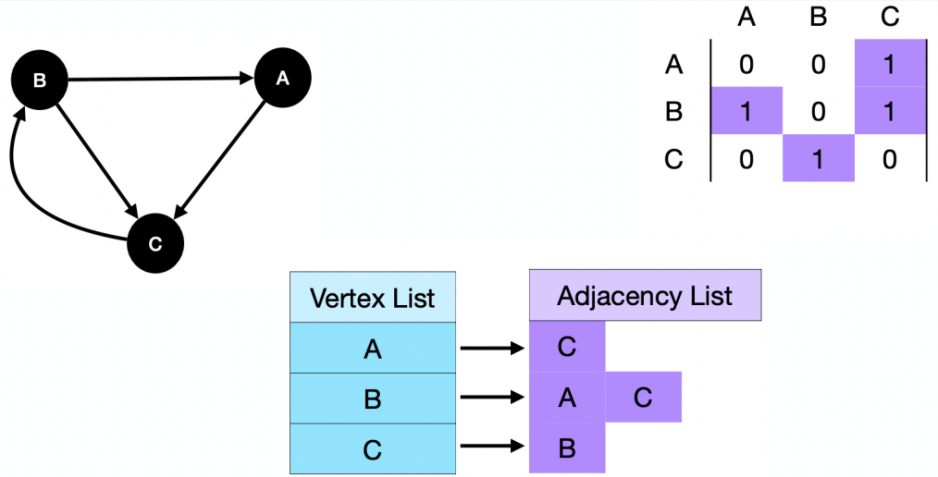
$$ML = M^1 \wedge M^2 \wedge M^3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Adjacency Matrix

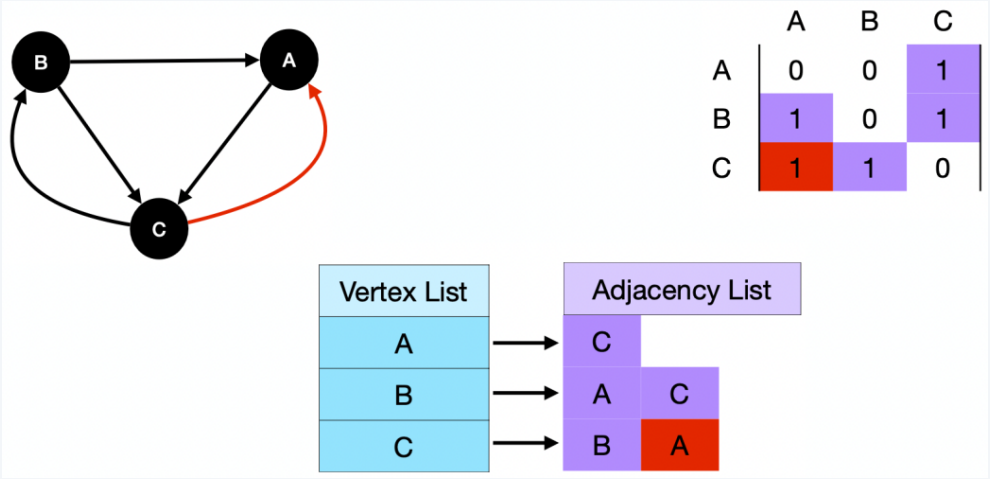
Summary

- Easy to understand.
- Handle a fixed number of vertices.
- Could take more memory than other methods because there are cells in the matrix for every possible combination of vertices.

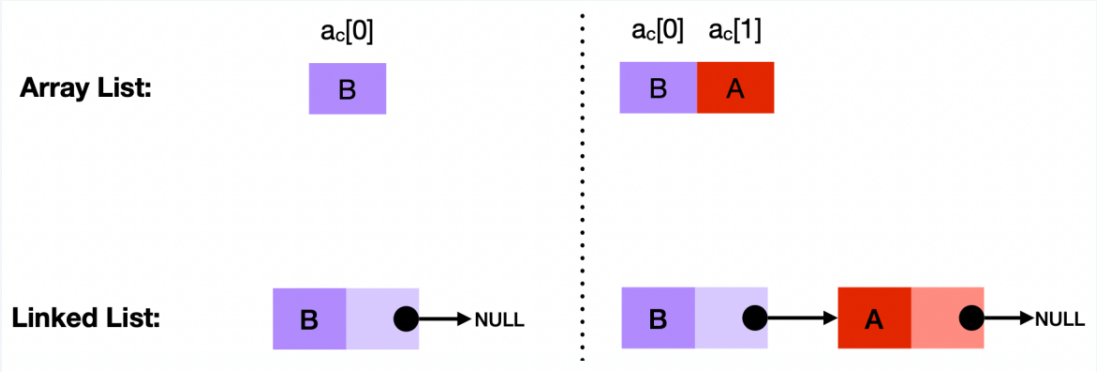
Adjacency List



Add New Edge



Array vs Linked List



Shortest Path Algorithms

Shortest Path Algorithm

- Minimum spanning Tree
- Dijkstra's algorithm
- Warshall's algorithm

Minimum Spanning Trees

Spanning Tree

A spanning tree of a connected, undirected graph G is a sub-graph of G which is a tree that connects all the vertices together.

Minimum spanning tree(MST)

A minimum spanning tree is defined as a spanning tree with weight less than of equal to the weight of every other spanning tree.

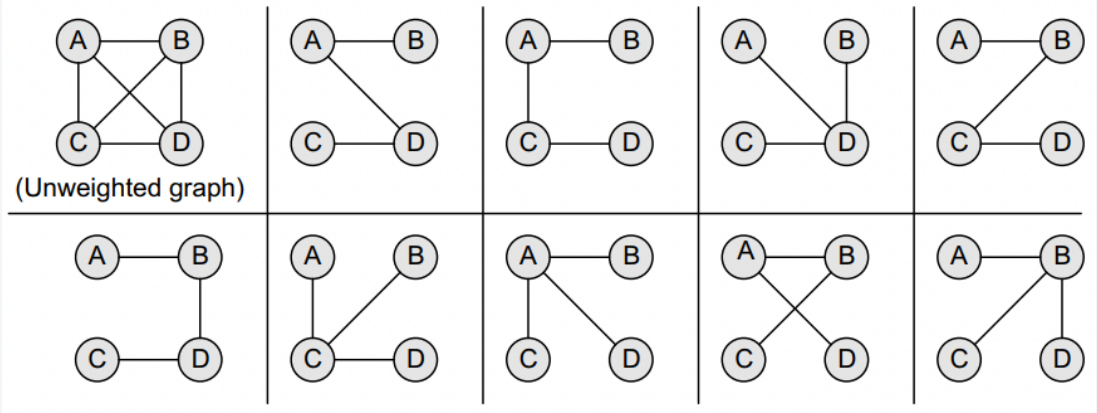
Minimum Spanning Trees

Properties

- Possible multiplicity
- Uniqueness
- Minimum-cost subgraph
- Cycle property
- Usefulness
- Simplicity

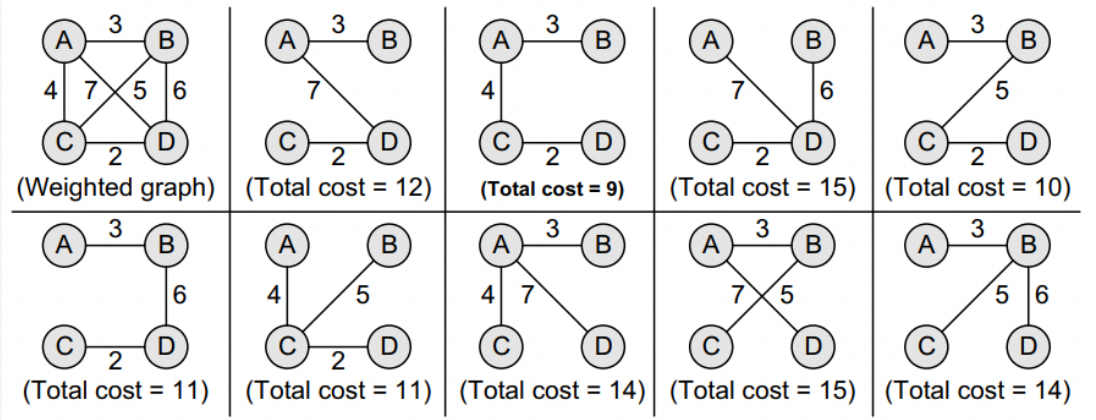
Minimum Spanning Trees

Unweighted graph and its spanning trees



Minimum Spanning Trees

Weighted graph and its spanning trees



Minimum Spanning Trees

Applications

- Design networks.
- Find airline routes.
- Find the cheapest way to connect terminals.
- Apply in routing algorithm.

Prim's Algorithm

Prim's algorithm is a greedy algorithm that is used to form a minimum spanning tree for a connected weighted undirected graph.

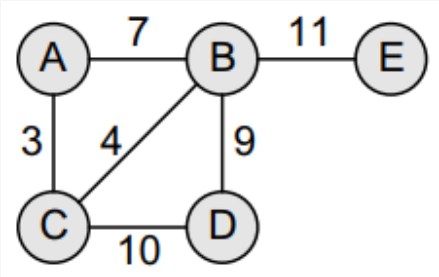
Three sets of vertices

- Tree vertices
- Fringe vertices
- Unseen vertices

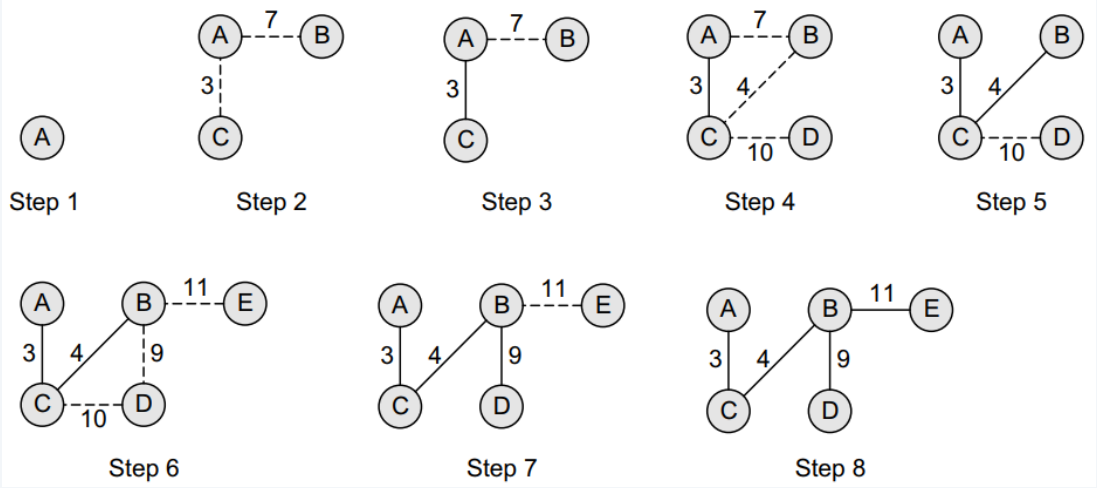
Prim's Algorithm

```
Step 1: Select a starting vertex
Step 2: Repeat Steps 3 and 4 until there are fringe vertices
Step 3:     Select an edge e connecting the tree vertex and
            fringe vertex that has minimum weight
Step 4:     Add the selected edge and the vertex to the
            minimum spanning tree T
        [END OF LOOP]
Step 5: EXIT
```

Example



Example


A
A
7
B
3
C
Step 1
Step 2
A
7
B
3
C
Step 3
A
7
B
3
4
C
10
D
Step 4
A
B
3
4
C
10
D
Step 5
A
B
11
E
3
4
9
C
10
D
Step 6
A
B
11
E
3
4
9
C
D
Step 7
A
B
11
E
3
4
9
C
D
Step 8

Kruskal's Algorithm

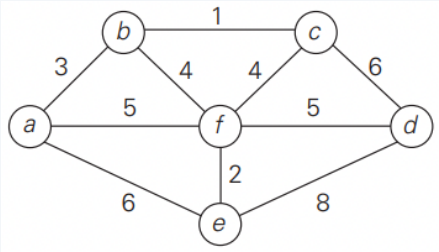
- Kruskal's algorithm is used to find the minimum spanning tree for a connected weighted graph.
- If the graph is not connected, then it finds a minimum spanning forest.

Kruskal's Algorithm

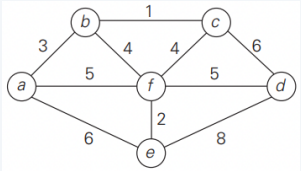
ALGORITHM *Kruskal(G)*

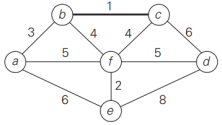
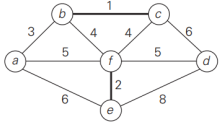
//Kruskal's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E \rangle$
//Output: E_T, the set of edges composing a minimum spanning tree of G
sort E in nondecreasing order of the edge weights $w(e_{i_1}) \leq \cdots \leq w(e_{i_{|E|}})$
$E_T \leftarrow \varnothing$; $ecounter \leftarrow 0$ //initialize the set of tree edges and its size
$k \leftarrow 0$ //initialize the number of processed edges
while $ecounter < |V| - 1$ **do**
    $k \leftarrow k + 1$
    if $E_T \cup \{e_{i_k}\}$ is acyclic
        $E_T \leftarrow E_T \cup \{e_{i_k}\}$; $ecounter \leftarrow ecounter + 1$
return E_T

Example

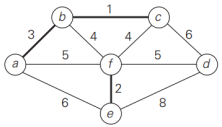
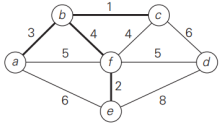
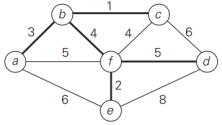


Example



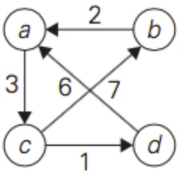
Tree edges	Sorted list of edges	Illustration
	bc ef ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
bc 1	bc **ef** ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	

Example

Tree edges	Sorted list of edges	Illustration
ef 2	bc ef **ab** bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
ab 3	bc ef ab **bf** cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
bf 4	bc ef ab bf cf af **df** ae cd de 1 2 3 4 4 5 5 6 6 8	
df 5		

All-Pairs Shortest-Paths

- Consider a positive weighted connected graph W.
- Find the shortest paths from each vertex to all the others.
- Distance matrix D contains the shortest path length from i^{th} vertex to j^{th} vertex.



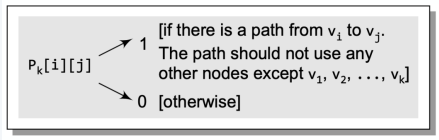
$$W = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array} \qquad D = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$

Warshall's Algorithm

- The path matrix of G can be found as

$$P = A + A^2 + A^3 + \ldots + A^n$$

- Warshall has given a very efficient algorithm to calculate the path matrix.
- Warshall's algorithm defiens metrices $P_0, P_1, P_2, \ldots, P_n$ as



Warshall's Algorithm

- $P_k[i][j]$ is equal to 1 only when either of the two following cases occur:
  - There is a path from v_i to v_j that does not use any other node except $v_1, v_2, \ldots, v_{k-1}$

$$P_{k-1}[i][j] = 1$$

  - There is a path from v_i to v_k and a path from v_k to v_j where all the nodes use $v_1, v_2, \ldots, v_{k-1}$

$$P_{k-1}[i][k] = 1 \text{ AND } P_{k-1}[k][j] = 1$$

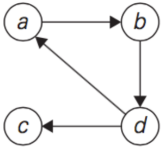
Warshall's Algorithm

The path matrix P_n can be calculated with the formula given as:

$$P_k[i][j] = P_{k-1}[j][j] \vee (P_{k-1}[i][k] \wedge P_{k-1}[k][j])$$

```
Step 1: [INITIALIZE the Path Matrix] Repeat Step 2 for I = 0 to n-1,
        where n is the number of nodes in the graph
Step 2:     Repeat Step 3 for J = 0 to n-1
Step 3:           IF A[I][J] = 0, then SET P[I][J] = 0
                  ELSE P[I][J] = 1
            [END OF LOOP]
      [END OF LOOP]
Step 4: [Calculate the path matrix P] Repeat Step 5 for K = 0 to n-1
Step 5:     Repeat Step 6 for I = 0 to n-1
Step 6:           Repeat Step 7 for J=0 to n-1
Step 7:                     SET P_K[I][J] = P_K-1[I][J] V (P_K-1[I][K]
                            Λ P_K-1[K][J])
Step 8: EXIT
```

Warshall's Algorithm



$R^{(0)} =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

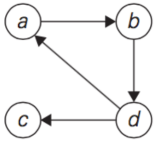
1's reflect the existence of paths with no intermediate vertices ($R^{(0)}$ is just the adjacency matrix); boxed row and column are used for getting $R^{(1)}$.

$R^{(1)} =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	**1**	1	0

1's reflect the existence of paths with intermediate vertices numbered not higher than 1, i.e., just vertex *a* (note a new path from *d* to *b*); boxed row and column are used for getting $R^{(2)}$.

Warshall's Algorithm



$R^{(2)}$ =

	a	b	c	d
a	0	1	0	**1**
b	0	0	0	1
c	0	0	0	0
d	1	1	1	**1**

1's reflect the existence of paths with intermediate vertices numbered not higher than 2, i.e., *a* and *b* (note two new paths); boxed row and column are used for getting $R^{(3)}$.

$R^{(3)}$ =

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

1's reflect the existence of paths with intermediate vertices numbered not higher than 3, i.e., *a, b,* and *c* (no new paths); boxed row and column are used for getting $R^{(4)}$.

$R^{(4)}$ =

	a	b	c	d
a	**1**	1	**1**	1
b	**1**	**1**	**1**	1
c	0	0	0	0
d	1	1	1	1

1's reflect the existence of paths with intermediate vertices numbered not higher than 4, i.e., *a, b, c,* and *d* (note five new paths).

Floyd's Algorithm

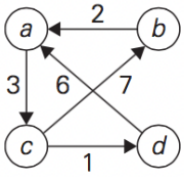
- Warshall's algorithm can be modified to obtain a matrix that gives the shortest paths.
- As an input to the algorithm, we take the adjacency matrix A and replace all the values of A which are zero by infinity (∞).
- We obtain a set of matrices $Q_0, Q_1, Q_2, \ldots, Q_m$ using the formula:

$$Q_k[i][j] = Min(M_{k-1}[j][j], M_{k-1}[i][k] + M_{k-1}[k][j])$$

Floyd's Algorithm

```
Step 1: [Initialize the Shortest Path Matrix, Q] Repeat Step 2 for I = 0
        to n-1, where n is the number of nodes in the graph
Step 2:     Repeat Step 3 for J = 0 to n-1
Step 3:           IF A[I][J] = 0, then SET Q[I][J] = Infinity (or 9999)
                  ELSE Q[I][J] = A[I][j]
            [END OF LOOP]
      [END OF LOOP]
Step 4: [Calculate the shortest path matrix Q] Repeat Step 5 for K = 0
        to n-1
Step 5:     Repeat Step 6 for I = 0 to n-1
Step 6:           Repeat Step 7 for J=0 to n-1
Step 7:                       IF Q[I][J] <= Q[I][K] + Q[K][J]
                              SET Q[I][J] = Q[I][J]
                        ELSE SET Q[I][J] = Q[I][K] + Q[K][J]
                        [END OF IF]
            [END OF LOOP]
      [END OF LOOP]
      [END OF LOOP]
Step 8: EXIT
```

Floyd's Algorithm



$D^{(0)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

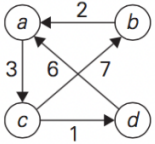
Lengths of the shortest paths with no intermediate vertices ($D^{(0)}$ is simply the weight matrix).

$D^{(1)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	**5**	∞
c	∞	7	0	1
d	6	∞	**9**	0

Lengths of the shortest paths with intermediate vertices numbered not higher than 1, i.e., just *a* (note two new shortest paths from *b* to *c* and from *d* to *c*).

Floyd's Algorithm



$D^{(2)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	**9**	7	0	1
d	6	∞	9	0

Lengths of the shortest paths with intermediate vertices numbered not higher than 2, i.e., a and *b* (note a new shortest path from *c* to *a*).

$D^{(3)} =$

	a	b	c	d
a	0	**10**	3	**4**
b	2	0	5	**6**
c	9	7	0	1
d	6	**16**	9	0

Lengths of the shortest paths with intermediate vertices numbered not higher than 3, i.e., a, *b,* and *c* (note four new shortest paths from *a* to *b,* from a to *d,* from *b* to *d,* and from *d* to *b*).

$D^{(4)} =$

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	**7**	7	0	1
d	6	16	9	0

Lengths of the shortest paths with intermediate vertices numbered not higher than 4, i.e., a, *b, c,* and *d* (note a new shortest path from *c* to *a*).

Applications of Graphs

Applications of Graphs

- Circuit networks
- Transport networks
- Maps
- Program flow analysis
- A graph of a particular concept: shortest paths, project planning, etc.
- Flowcharts or control-flow graphs
- State transition diagrams
- Activity network diagram (known as an arrow diagram or program evaluation review technique): project management tool