CPE 112 Programming with
Data Structures

Graph Algorithms

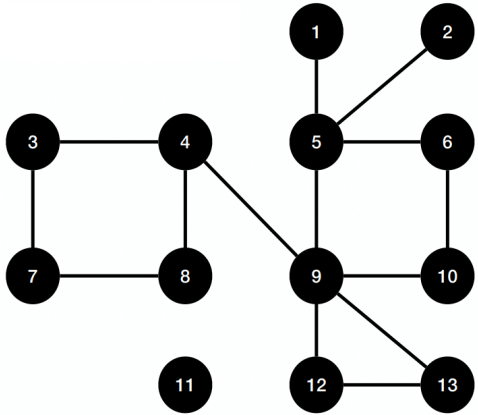
Dr. Piyanit Ua-areemitr
Dr. Taweechai Nuntawisuttiwong

Contents

Graph Traversal Algorithms

Graph Traversal

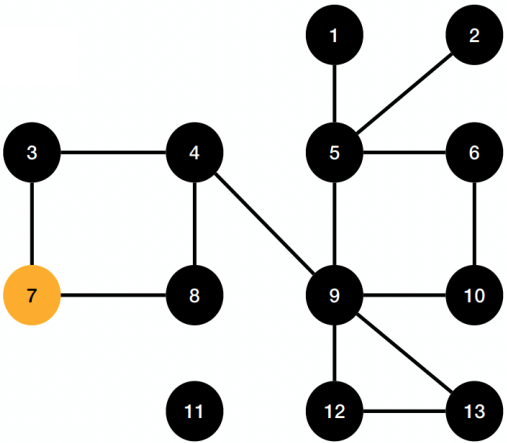
- The method of examining the nodes and edges of graph.
- Standard methods
  - Depth-first search
  - Bread-first search



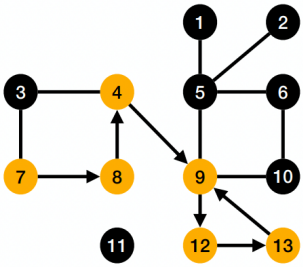
Depth-first Search

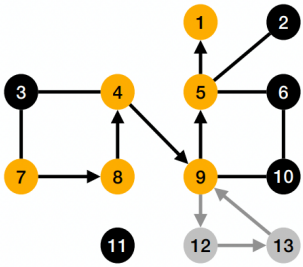
- Search in depth until cannot go further
- Backtrack & continue

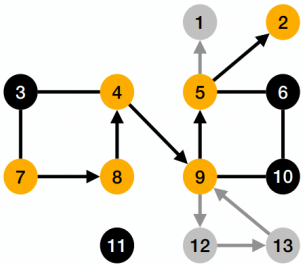
Start from node 7



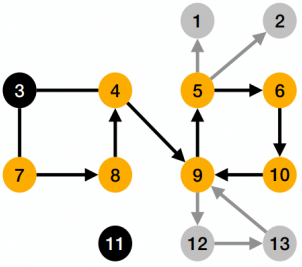
Depth-first Search

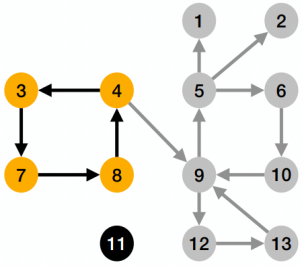


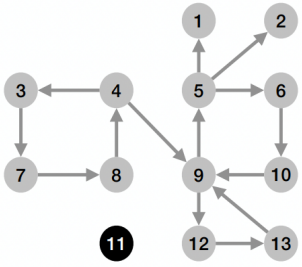




Depth-first Search



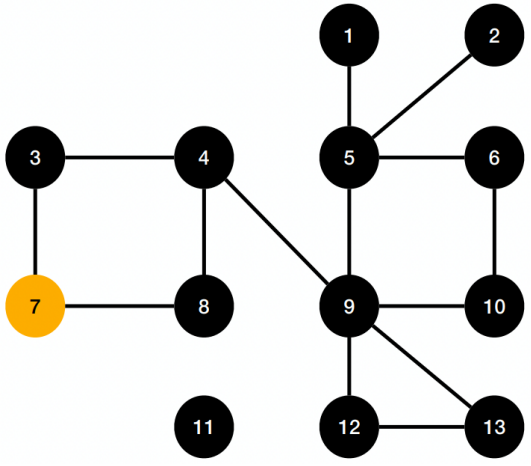




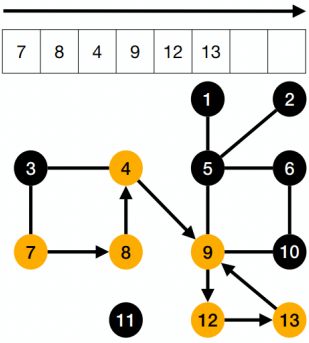
State of Nodes

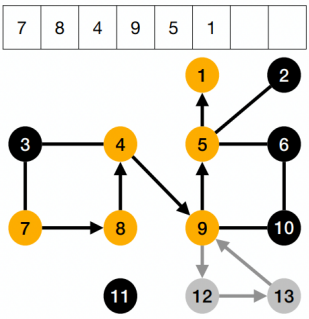
Status	State	Description
1	**Ready**	The initial state of the node N .
2	**Waiting**	Node N is place on the stack/queue and waiting to be processed.
3	**Processed**	Node N has been completely processed.

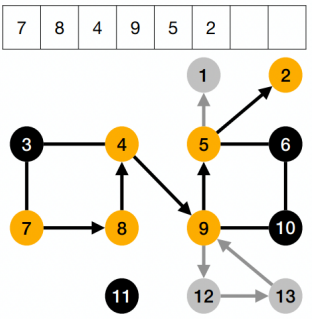
Depth-first Search



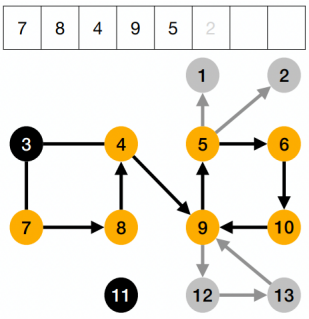
Depth-first Search

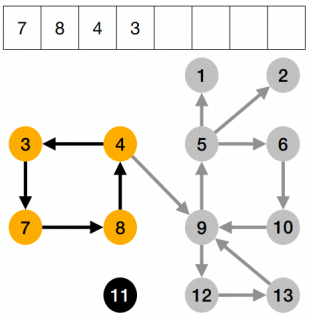


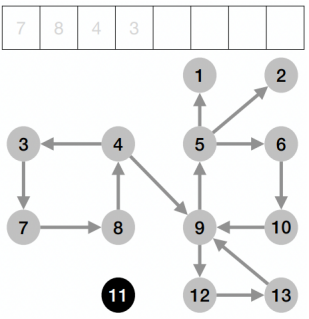




Depth-first Search







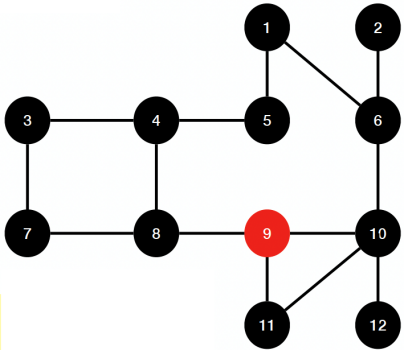
Depth-first Search

Push the first vertex onto the stack
Mark this vertex as **visited**
REPEAT
  Visit the next vertex **adjacent** to the one on top of the stack
  Push this vertex onto the stack
  Mark this vertex as visited
  IF there is no vertex to visit
    Pop this vertex off the stack
    Mark this vertex as processed
  END IF
UNTIL the stack is empty

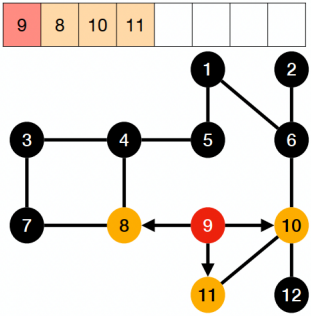
Breadth-first Search

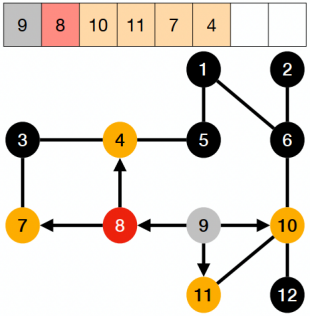
- Explore neighbors
- Move to the next level neighbors

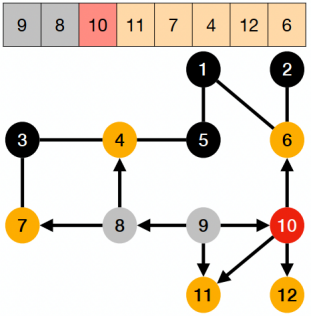
Start from node 9



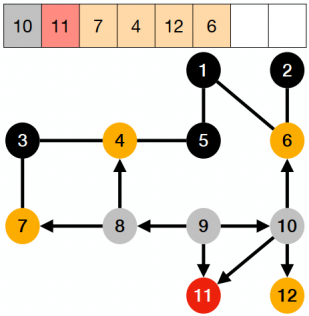
Breadth-first Search

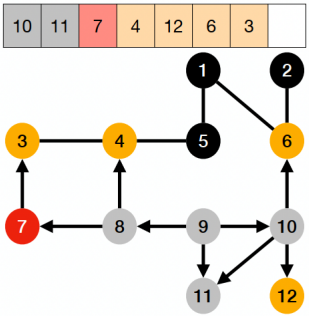


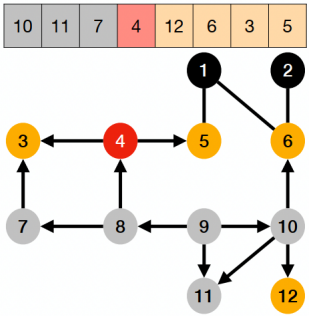




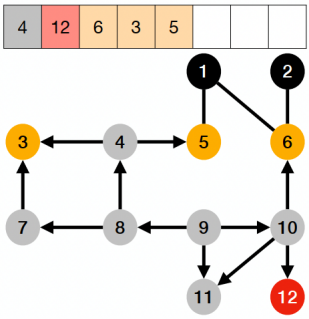
Breadth-first Search

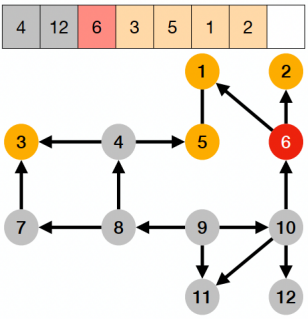


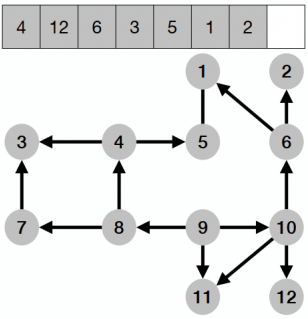




Breadth-first Search







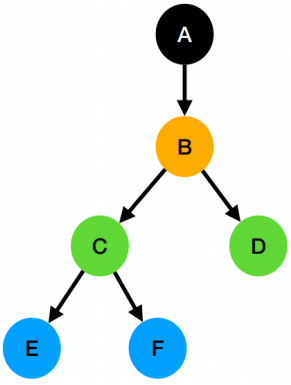
Breadth-first Search

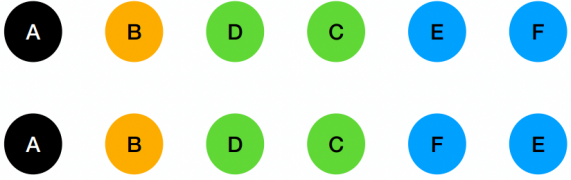
Enqueue the first vertex into the queue
Mark this vertex as visited
REPEAT
  Visit the next vertex adjacent to the first vertex
  Mark this vertex as visited
  Enqueue this vertex into the queue
UNTIL all adjacent vertices visited
REPEAT
  Dequeue next vertex from the queue
  Mark as processed
  REPEAT
    Visit the next unvisited vertex adjacent to that at the front of the queue
    Mark this vertex as visited
    Enqueue this vertex into the queue
  UNTIL all adjacent vertices visited
UNTIL the queue is empty

Topological Sorting

Topological Sorting

- A linear ordering of nodes in a graph
- Each node comes before all nodes to which it has outbound edges

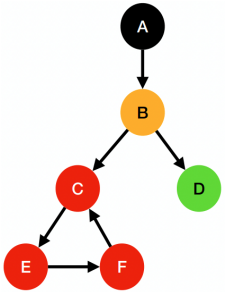


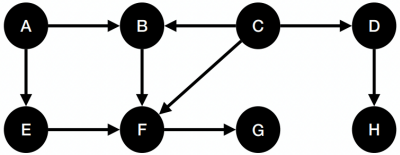


Topological Sorting

- A linear ordering of nodes in a graph
- Each node comes before all nodes to which it has outbound edges

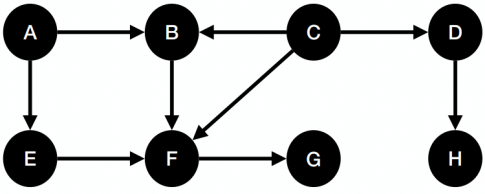
A topological sort of a DAG G: An ordering of the vertices such that if G contains an edge (u, v), then u appears before v in the ordering.



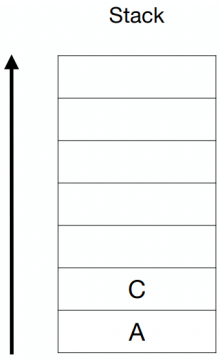


Topological Sorting

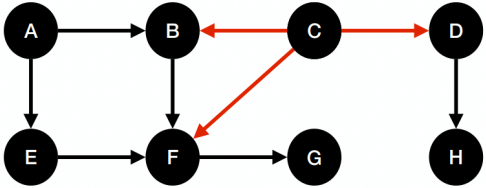
- Use stack or queue



Vertex	In-degree
A	0
B	2
C	0
D	1
E	1
F	3
G	1
H	1



Topological Sorting



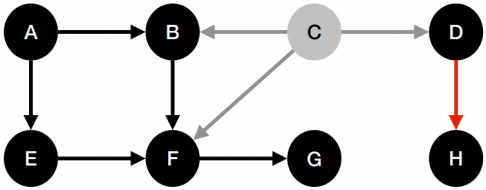
C

Vertex	In-degree
A	0
B	~~2~~ 1
C	0
D	~~1~~ 0
E	1
F	~~3~~ 2
G	1
H	1

Stack

D
A

Topological Sorting

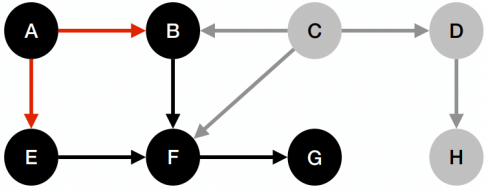


Vertex	In-degree
A	0
B	1
C	0
D	0
E	1
F	2
G	1
H	~~1~~ 0

Stack

H
A

Topological Sorting

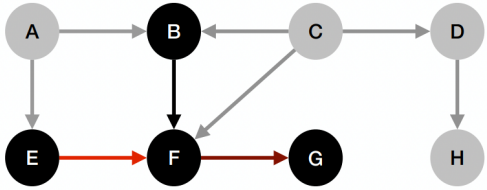


Vertex	In-degree
A	0
B	~~1~~ 0
C	0
D	0
E	~~1~~ 0
F	2
G	1
H	0

Stack

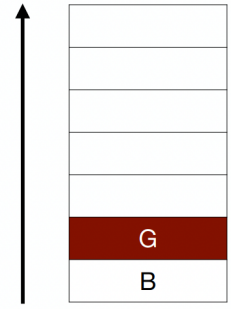
E
B

Topological Sorting

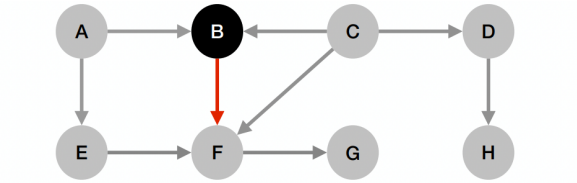


Vertex	In-degree
A	0
B	0
C	0
D	0
E	0
F	~~2~~ 1
G	~~1~~ 0
H	0

Stack

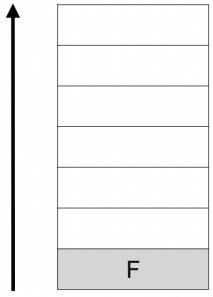


Topological Sorting



Vertex	In-degree
A	0
B	0
C	0
D	0
E	0
F	~~1~~ 0
G	0
H	0

Stack



Topological Sorting

Pseudocode

```
Set up the in-degree array from the graph
Add any vertices with in-degree zero to the stack

REPEAT
   Remove first vertex and add it to the ordering
   FOR EACH edge from that vertex
      Reduce the in-degree for the vertex the edge goes to
      IF the in-degree for that vertex is zero
         Put it in the queue
      END IF
UNTIL the stack is not empty
```

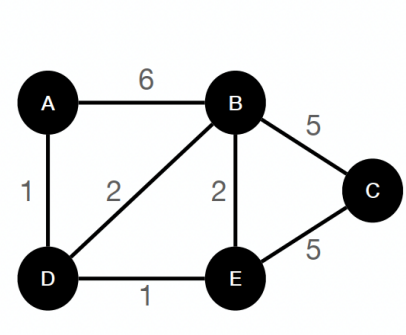
Shortest Path Algorithms

Shortest Path Algorithm

- Dijkstra's algorithm
- Warshall's algorithm

Dijkstra's Algorithm

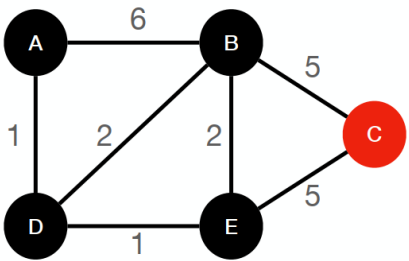
- Non-negative edge weights.
- Find the shortest path from vertex A to every other vertex.



Vertex	Shortest dist. from A	Previous vertex
A	0	-
B	3	D
C	7	E
D	1	A
E	2	D

Dijkstra's Algorithm

From A to C

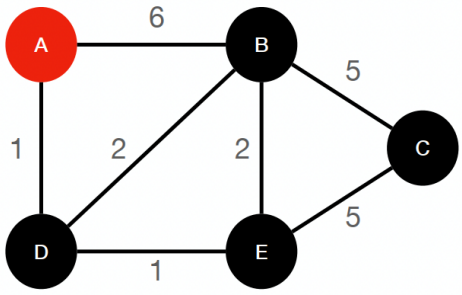


Vertex	Shortest dist. from A	Previous vertex
A	0	-
B	3	D
C	7	E
D	1	A
E	2	D

Dijkstra's Algorithm

- Initial Step: List & Distance from A

Visited = [ ]
Unvisited = [A, B, C, D, E]

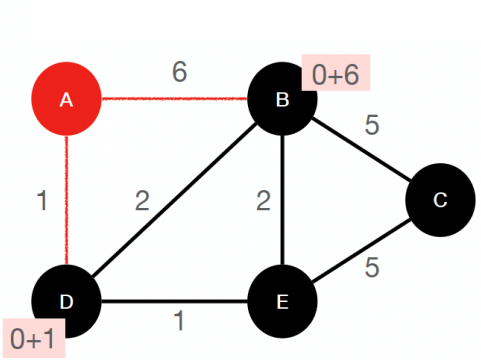


Vertex	Shortest dist. from A	Previous vertex
A	0	-
B	∞	
C	∞	
D	∞	
E	∞	

Dijkstra's Algorithm

- Examine unvisited neighbors (B, D)
- Calculate the distance from the start vertex
- Update the shortest distance

Visited = [ ]
Unvisited = [A, B, C, D, E]

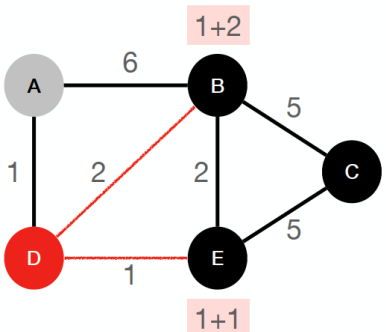


Vertex	Shortest dist. from A	Previous vertex
A	0	-
B	6	A
C	∞	
D	1	A
E	∞	

Dijkstra's Algorithm

- Update Lists
- Visit the unvisited vertex with the smallest known distance (D)

Visited = [A]
Unvisited = [B, C, D, E]

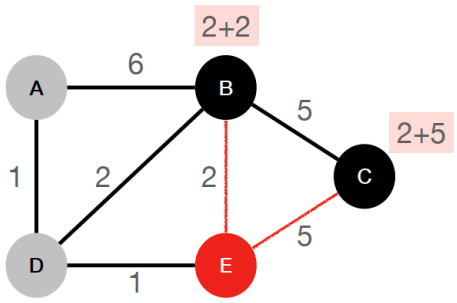


Vertex	Shortest dist. from A	Previous vertex
A	0	-
B	3	D
C	∞	
D	1	A
E	2	D

Dijkstra's Algorithm

- Update Lists
- Visit the unvisited vertex with the smallest known distance (E)

Visited = [A, D]
Unvisited = [B, C, E]

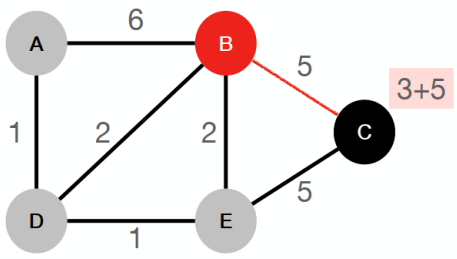


Vertex	Shortest dist. from A	Previous vertex
A	0	-
B	3	D
C	7	E
D	1	A
E	2	D

Dijkstra's Algorithm

- Update Lists
- Visit the unvisited vertex with the smallest known distance (B)

Visited = [A, D, E]
Unvisited = [B, C]

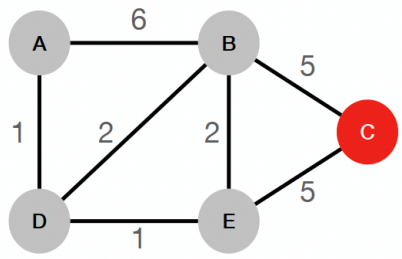


Vertex	Shortest dist. from A	Previous vertex
A	0	-
B	3	D
C	7	E
D	1	A
E	2	D

Dijkstra's Algorithm

- Update Lists
- Visit the unvisited vertex with the smallest known distance (C)

Visited = [A, D, E, B]
Unvisited = [C]



Vertex	Shortest dist. from A	Previous vertex
A	0	-
B	3	D
C	7	E
D	1	A
E	2	D

Dijkstra's Algorithm

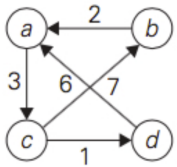
Pseudocode

```
Let distance of start vertex from start vertex = 0
Let distance of all other vertices from start vertex = infinity

REPEAT
  Visit the unvisited vertex with the smallest known distance from the start vertex
  For the current vertex, examine its unvisited neighbors
  For the current vertex, calculate distance of each neighbor from the start vertex
  FOR EACH calculated distance
  IF the calculated distance of a vertex < the known distance
    Update the shortest distance
    Update the previous vertex
  END IF
  Update the lists of visited and unvisited vertices
UNTIL all vertices are visited
```

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.



$$W = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

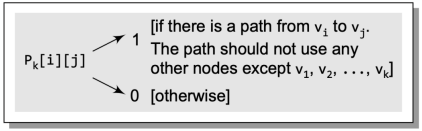
$$D = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$

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 $v_1, v_2, \ldots, v_{k-1}$

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

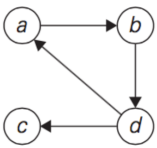
Warshall's Algorithm

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

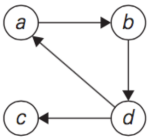
1's reflect the existence of paths with no intermediate vertices ($R^{(0)}$ is just the adjacency matrix); boxed row and column are used for getting $R^{(1)}$.

$R^{(1)} =$

	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

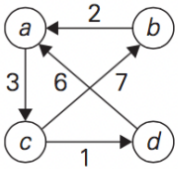
- Warshall's algorithm can be modified to obtain a matrix that gives the shortest paths.
- As an input to the algorithm, we take the adjacency matrix A and replace all the values of A which are zero by infinity (∞).
- We obtain a set of matrices $Q_0, Q_1, Q_2, \ldots, Q_m$ using the formula:

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

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

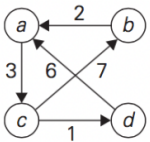
Lengths of the shortest paths with no intermediate vertices ($D^{(0)}$ is simply the weight matrix).

$D^{(1)} =$

	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