



# OBJECT ORIENTED ANALYSIS & DESIGN DATA STRUCTURES & ALGORITHMS

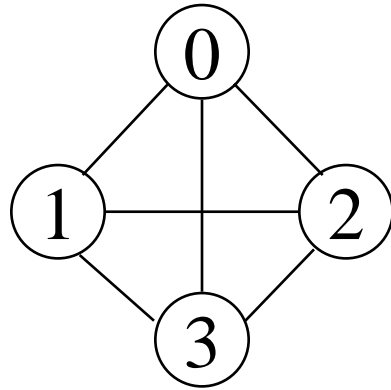
## GRAPHS

# Definition

- A graph  $G$  consists of two sets
  - a finite, nonempty set of vertices  $V(G)$
  - a finite, possible empty set of edges  $E(G)$
  - $G(V, E)$  represents a graph
- An undirected graph is one in which the pair of vertices in a edge is unordered,  $(v_0, v_1) = (v_1, v_0)$
- A directed graph is one in which each edge is a directed pair of vertices,  $\langle v_0, v_1 \rangle \neq \langle v_1, v_0 \rangle$

tail  $\longrightarrow$  head

# Examples for Graph



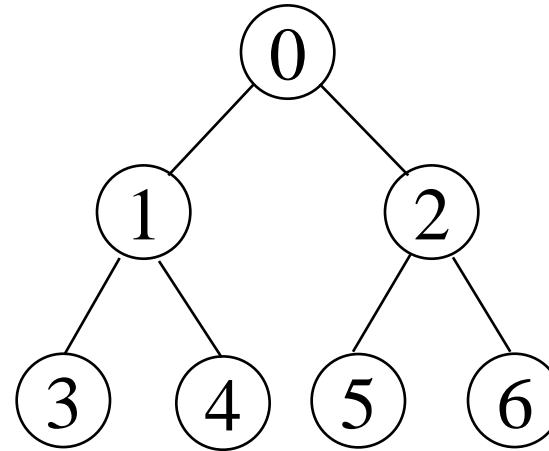
$G_1$

complete graph

$$V(G_1) = \{0, 1, 2, 3\}$$

$$V(G_2) = \{0, 1, 2, 3, 4, 5, 6\}$$

$$V(G_3) = \{0, 1, 2\}$$



$G_2$

incomplete graph

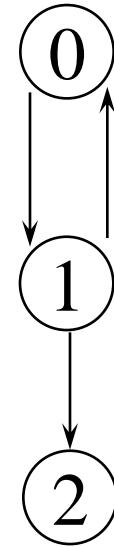
$$E(G_1) = \{(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)\}$$

$$E(G_2) = \{(0, 1), (0, 2), (1, 3), (1, 4), (2, 5), (2, 6)\}$$

$$E(G_3) = \{<0, 1>, <1, 0>, <1, 2>\}$$

complete undirected graph:  $n(n-1)/2$  edges

complete directed graph:  $n(n-1)$  edges



$G_3$

# Complete Graph

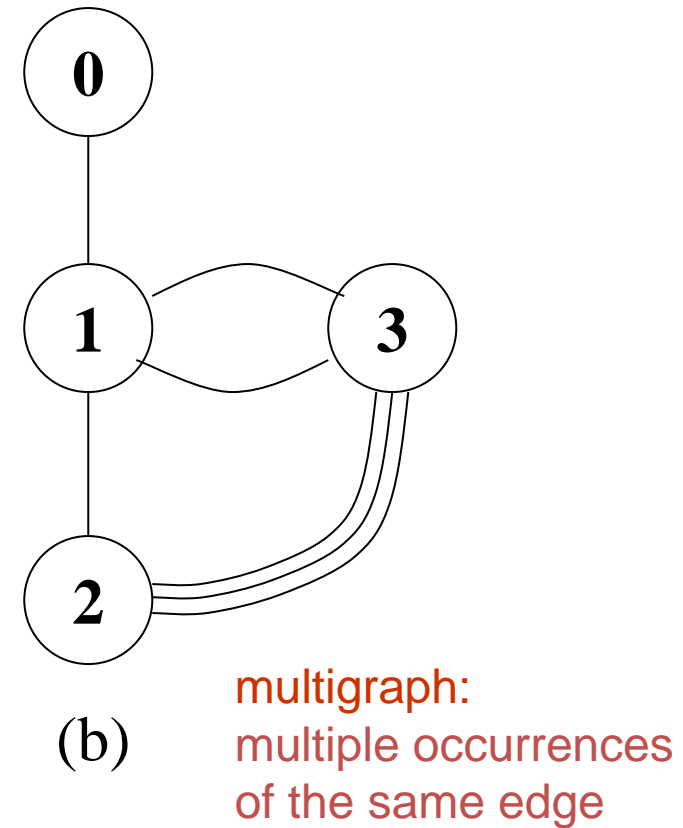
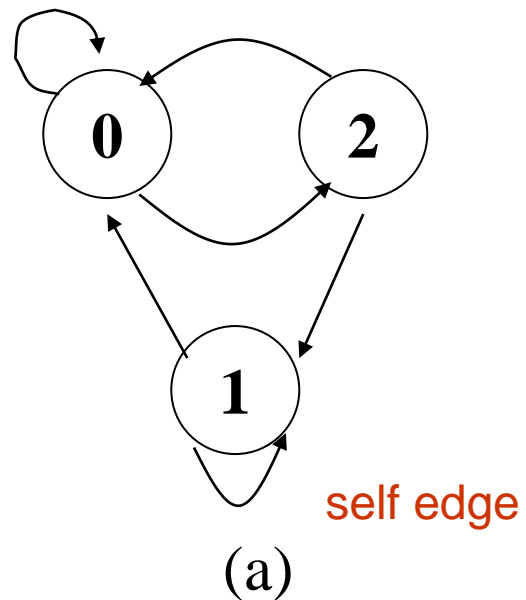
- A complete graph is a graph that has the maximum number of edges
  - for undirected graph with  $n$  vertices, the maximum number of edges is  $\frac{n(n-1)}{2}$
  - for directed graph with  $n$  vertices, the maximum number of edges is  $n(n-1)$
  - example:  $G_1$  is a complete graph

# Adjacent and Incident

- If  $(v_0, v_1)$  is an edge in an undirected graph,
  - $v_0$  and  $v_1$  are adjacent
  - The edge  $(v_0, v_1)$  is incident on vertices  $v_0$  and  $v_1$
- If  $\langle v_0, v_1 \rangle$  is an edge in a directed graph
  - $v_0$  is adjacent to  $v_1$ , and  $v_1$  is adjacent from  $v_0$
  - The edge  $\langle v_0, v_1 \rangle$  is incident on  $v_0$  and  $v_1$



## Example of a graph with feedback loops and a multigraph

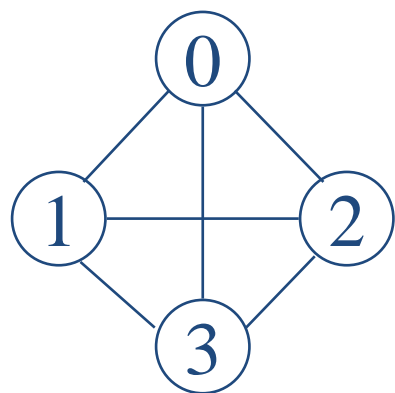




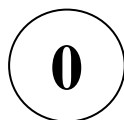
## Subgraph and Path

- A subgraph of  $G$  is a graph  $G'$  such that  $V(G')$  is a subset of  $V(G)$  and  $E(G')$  is a subset of  $E(G)$
- A path from vertex  $v_p$  to vertex  $v_q$  in a graph  $G$ , is a sequence of vertices,  $v_p, v_{i1}, v_{i2}, \dots, v_{in}, v_q$ , such that  $(v_p, v_{i1}), (v_{i1}, v_{i2}), \dots, (v_{in}, v_q)$  are edges in an undirected graph
- The length of a path is the number of edges on it

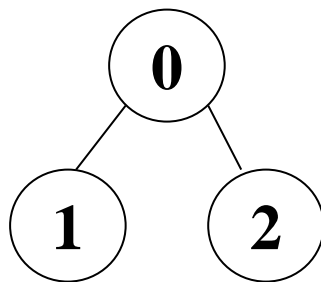
## subgraphs of $G_1$ and $G_3$



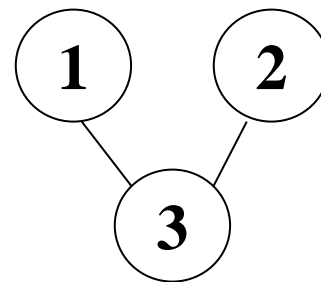
$G_1$



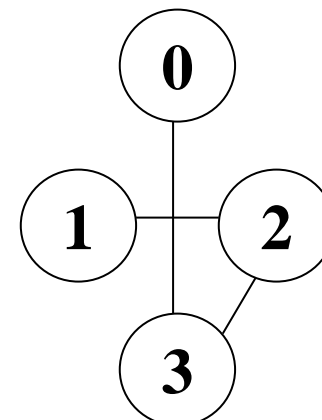
(i)



(ii)



(iii)

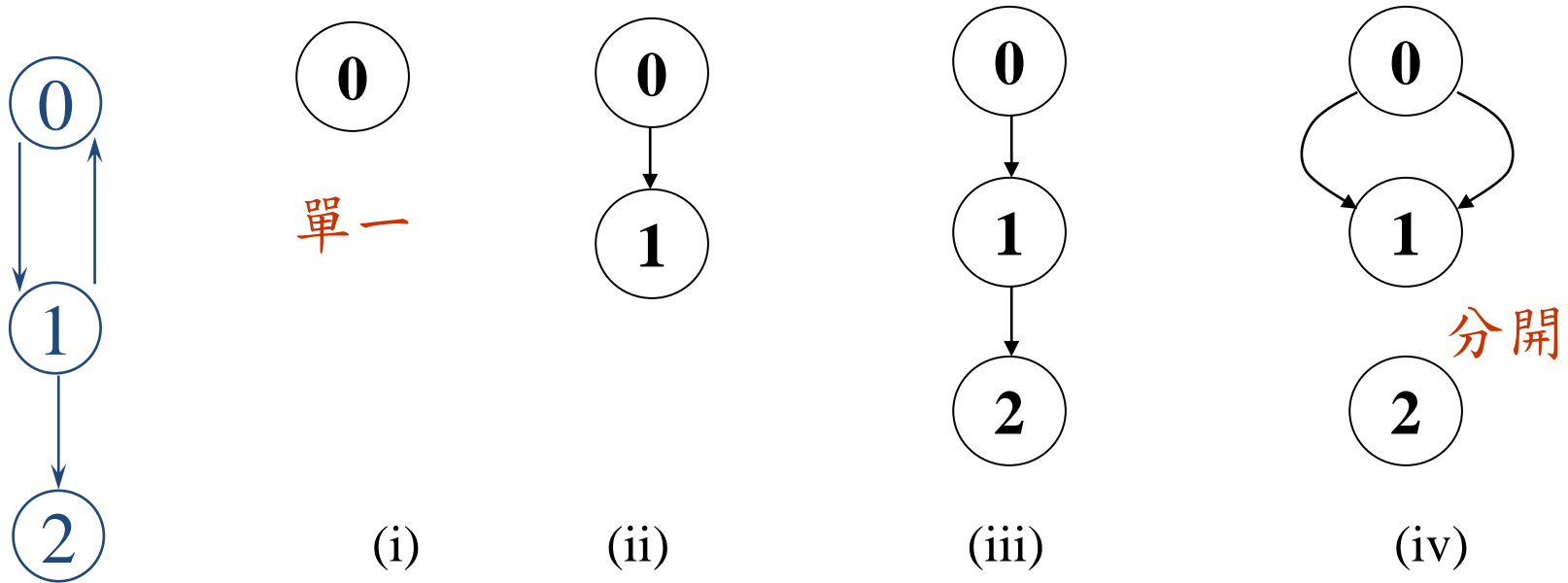


(iv)

(a) Some of the subgraph of  $G_1$



## subgraphs of $G_1$ and $G_3$ (contd.)



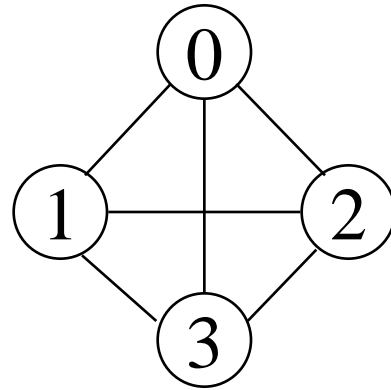
$G_3$

(b) Some of the subgraph of  $G_3$

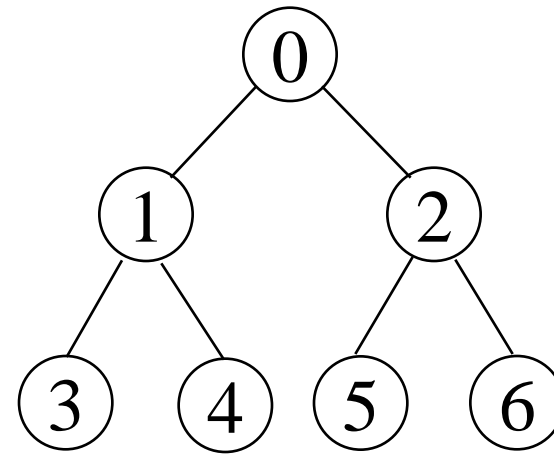
# Simple Path and Style

- A **simple path** is a path in which all vertices, except possibly the **first and the last**, are **distinct**
- A **cycle** is a simple **path** in which the **first and the last vertices** are the same
- In an **undirected** graph  $G$ , two vertices,  **$v_0$  and  $v_1$** , are connected if there is a path in  **$G$  from  $v_0$  to  $v_1$**
- An **undirected graph** is **connected** if, for every pair of distinct vertices  $v_i, v_j$ , there is a **path from  $v_i$  to  $v_j$**

## Connected



$G_1$



$G_2$

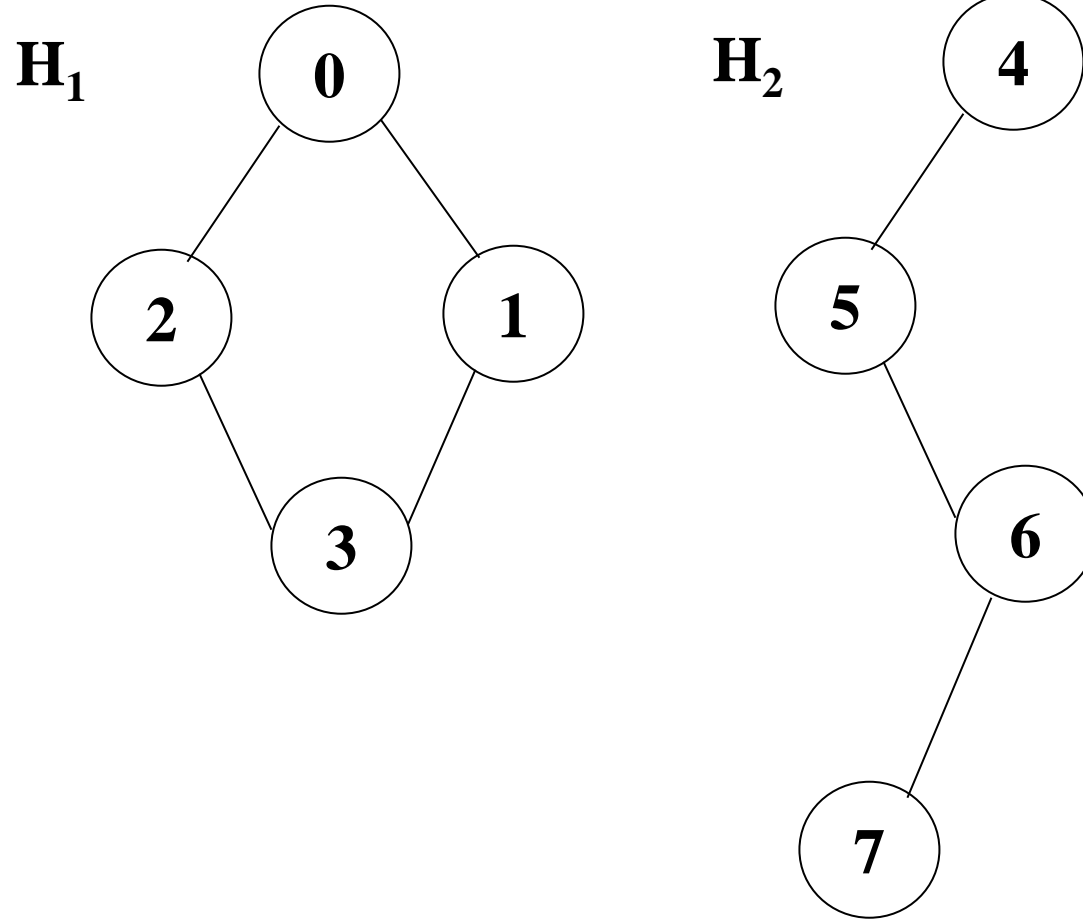
tree (acyclic graph)

# Connected Component

- A connected component of an undirected graph is a maximal connected subgraph.
- A tree is a graph that is connected and acyclic.
- A directed graph is strongly connected if there is a directed path from  $v_i$  to  $v_j$  and also from  $v_j$  to  $v_i$ .
- A strongly connected component is a maximal subgraph that is strongly connected.

# A graph with two connected components

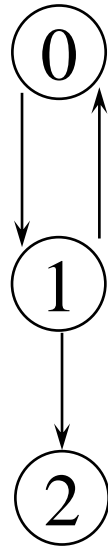
connected component (maximal connected subgraph)



$G_4$  (not connected)

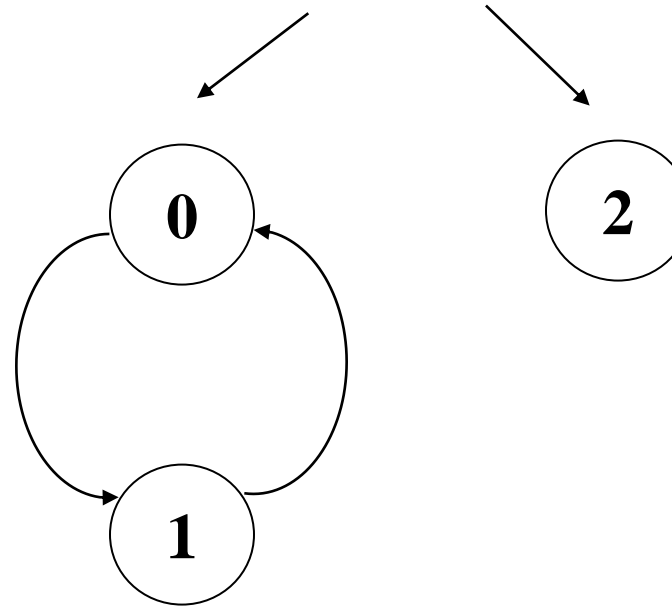
# Strongly connected components of $G_3$

not strongly connected



$G_3$

strongly connected component  
(maximal strongly connected subgraph)



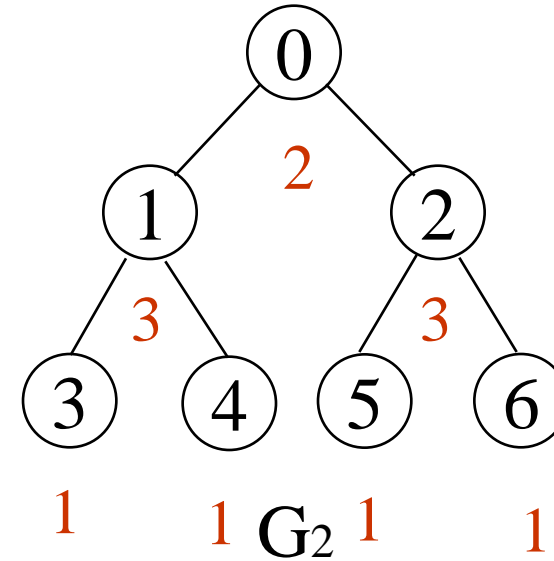
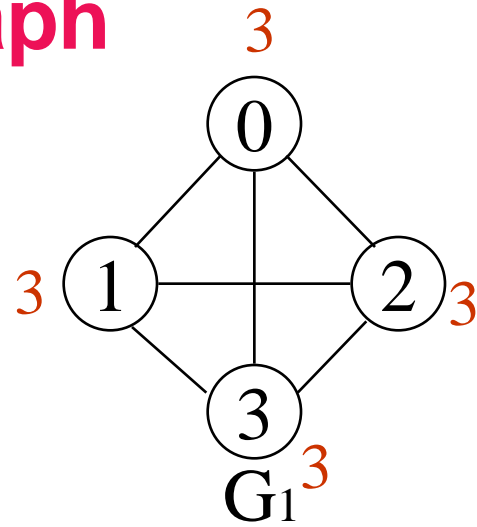


# Degree

- The **degree** of a **vertex** is the **number of edges incident to that vertex**
- For **directed graph**,
  - the **in-degree** of a vertex  $v$  is the number of edges that have  $v$  as the head
  - the **out-degree** of a vertex  $v$  is the number of edges that have  $v$  as the tail
  - if  $d_i$  is the **degree of a vertex  $i$**  in a graph  $G$  with  **$n$  vertices** and  **$e$  edges**, the **number of edges** is

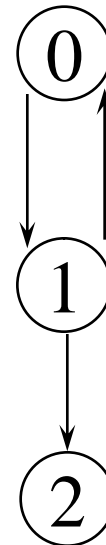
$$e = \left( \sum_{i=0}^{n-1} d_i \right) / 2$$

# undirected graph



directed graph

in-degree  
out-degree



in: 1, out: 1

in: 1, out: 2

in: 1, out: 0

$G_3$

# ADT for Graph

structure Graph is

**objects:** a nonempty set of vertices and a set of undirected edges, where each edge is a pair of vertices

**functions:** for all  $graph \in Graph$ ,  $v$ ,  $v_1$  and  $v_2 \in Vertices$

*Graph* **Create()** ::= return an empty graph

*Graph* **InsertVertex**(*graph*, *v*) ::= return a graph with *v* inserted. *v* has no incident edge.

*Graph* **InsertEdge**(*graph*, *v1*, *v2*) ::= return a graph with new edge between *v1* and *v2*

*Graph* **DeleteVertex**(*graph*, *v*) ::= return a graph in which *v* and all edges incident to it are removed

*Graph* **DeleteEdge**(*graph*, *v1*, *v2*) ::= return a graph in which the edge (*v1*, *v2*) is removed

*Boolean* **IsEmpty**(*graph*) ::= if (*graph* == empty graph) return TRUE  
else return FALSE

*List* **Adjacent**(*graph*, *v*) ::= return a list of all vertices that are adjacent to *v*

# Graph Representations

- Adjacency Matrix
- Adjacency Lists
- Adjacency Multilists

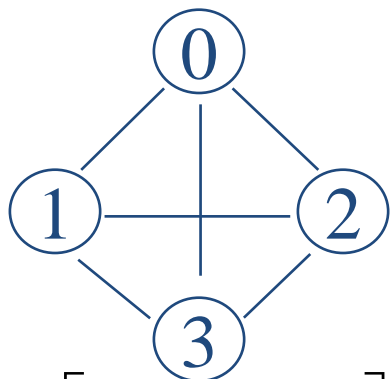


# Adjacency Matrix

- Let  $G=(V,E)$  be a graph with  $n$  vertices.
- The adjacency matrix of  $G$  is a two-dimensional  $n$  by  $n$  array, say  $\text{adj\_mat}$
- If the edge  $(v_i, v_j)$  is in  $E(G)$ ,  $\text{adj\_mat}[i][j]=1$
- If there is no such edge in  $E(G)$ ,  $\text{adj\_mat}[i][j]=0$
- The adjacency matrix for an undirected graph is symmetric; the adjacency matrix for a digraph need not be symmetric



# Examples for Adjacency Matrix



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

$G_1$

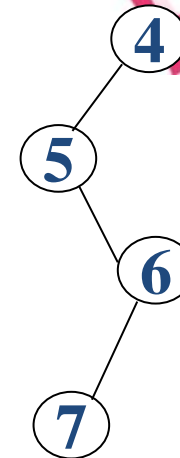
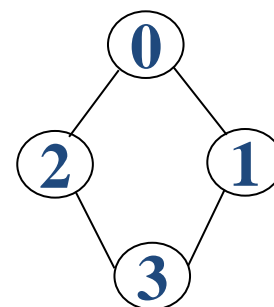
undirected:  
 $n^2/2$   
 directed:  $n^2$



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

$G_2$

symmetric



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

$G_4$

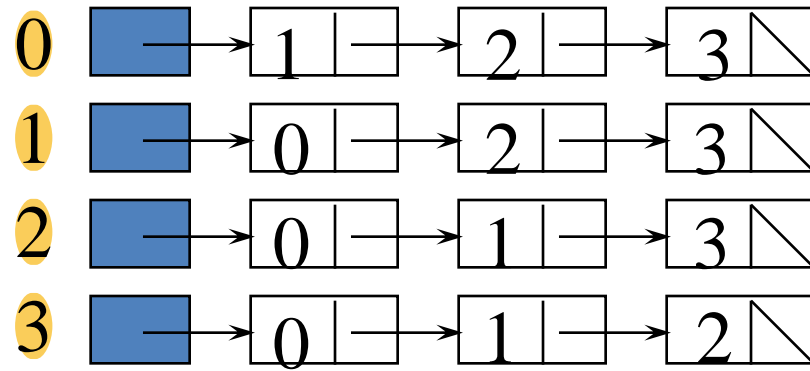
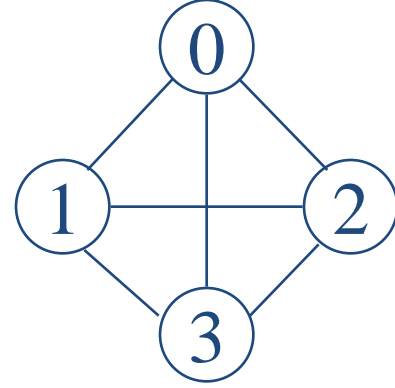


## Merits of Adjacency Matrix

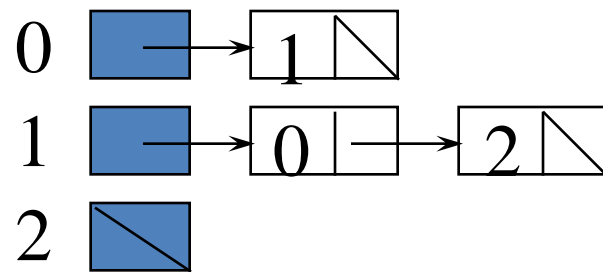
- From the adjacency matrix, to determine the connection of vertices is easy
- The degree of a vertex is
- For a digraph, the row sum is the out\_degree, while the column sum is the in\_degree

$$\sum_{j=0}^{n-1} adj\_mat[i][j]$$

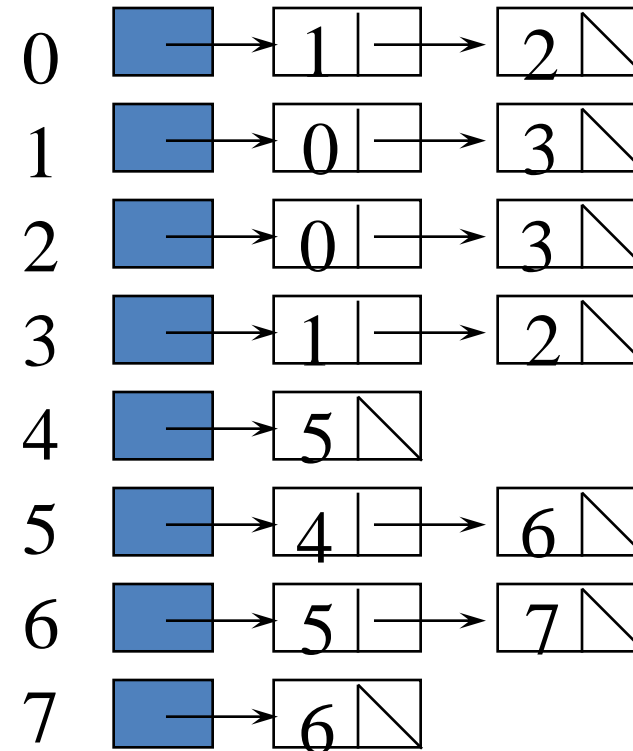
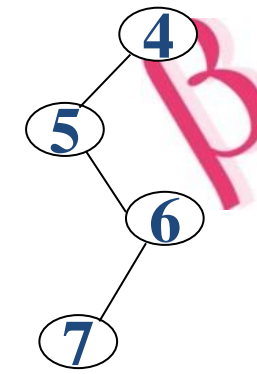
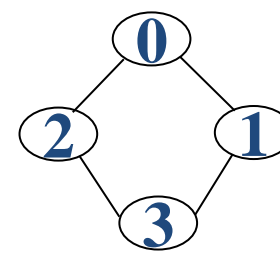
$$ind(v_i) = \sum_{j=0}^{n-1} A[j, i] \quad outd(v_i) = \sum_{j=0}^{n-1} A[i, j]$$



$G_1$



$G_3$



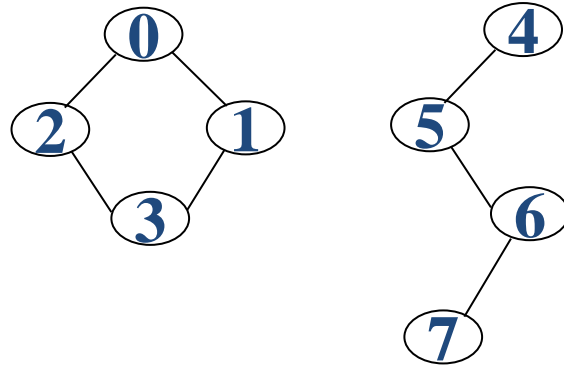
$G_4$

An undirected graph with  $n$  vertices and  $e$  edges  $\Rightarrow$   $n$  head nodes and  $2e$  list nodes

## Interesting Operations

- degree of a vertex in an undirected graph  
# of nodes in adjacency list
- # of edges in a graph  
determined in  $O(n+e)$
- out-degree of a vertex in a directed graph  
# of nodes in its adjacency list
- in-degree of a vertex in a directed graph  
traverse the whole data structure

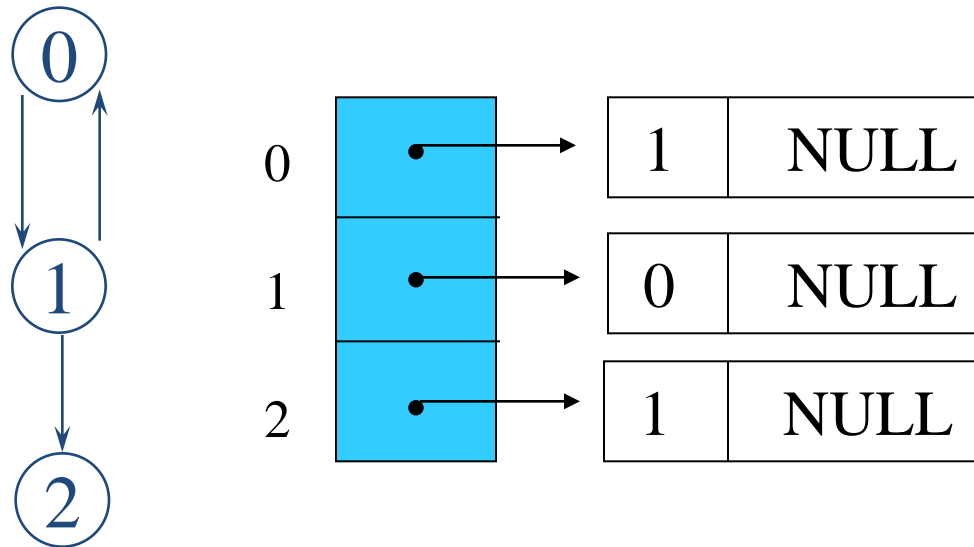
# Compact Representation



node[0] ... node[n-1]: starting point for vertices  
 node[n]:  $n+2e+1$   
 node[n+1] ... node[n+2e]: head node of edge

[0]	9		[8]	23		[16]	2
[1]	11	0	[9]	1	4	[17]	5
[2]	13		[10]	2	5	[18]	4
[3]	15	1	[11]	0		[19]	6
[4]	17		[12]	3	6	[20]	5
[5]	18	2	[13]	0		[21]	7
[6]	20		[14]	3	7	[22]	6
[7]	22	3	[15]	1			

## Inverse adjacency list for G3



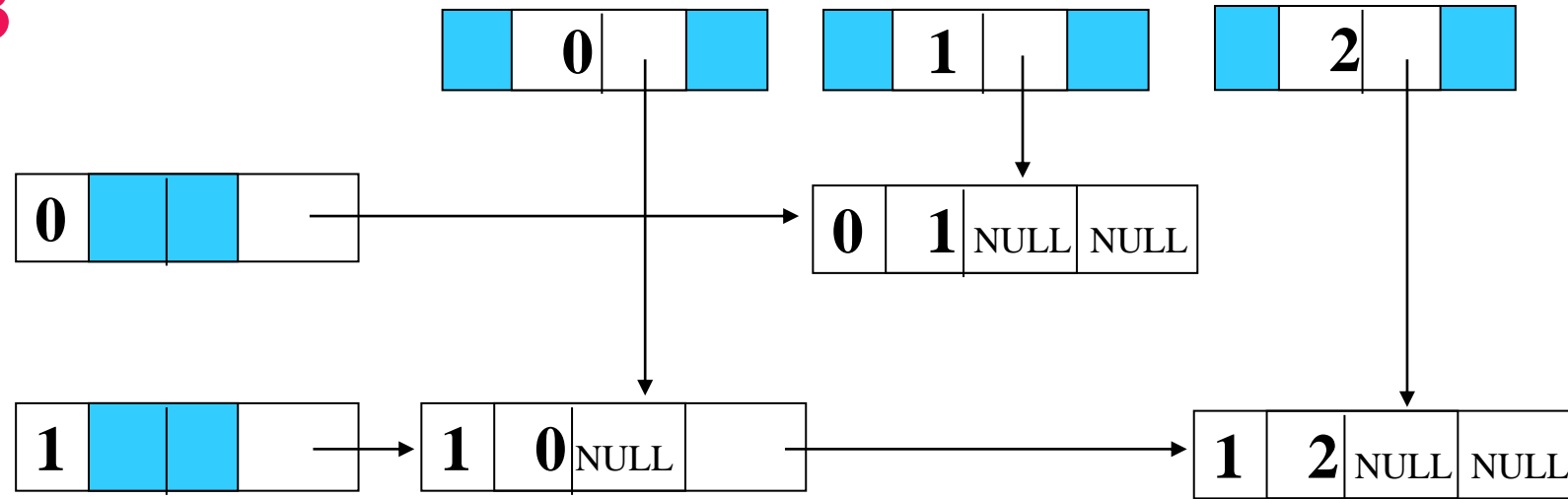
Determine in-degree of a vertex in a fast way.

## Alternate node structure for adjacency lists

tail	head	column link for head	row link for tail
------	------	----------------------	-------------------



# Orthogonal representation for graph G3

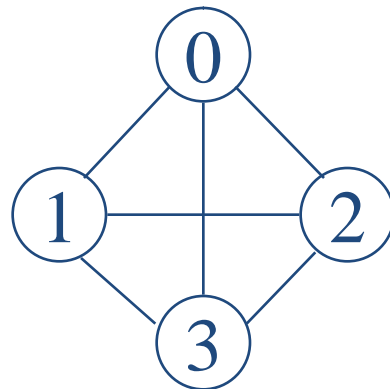
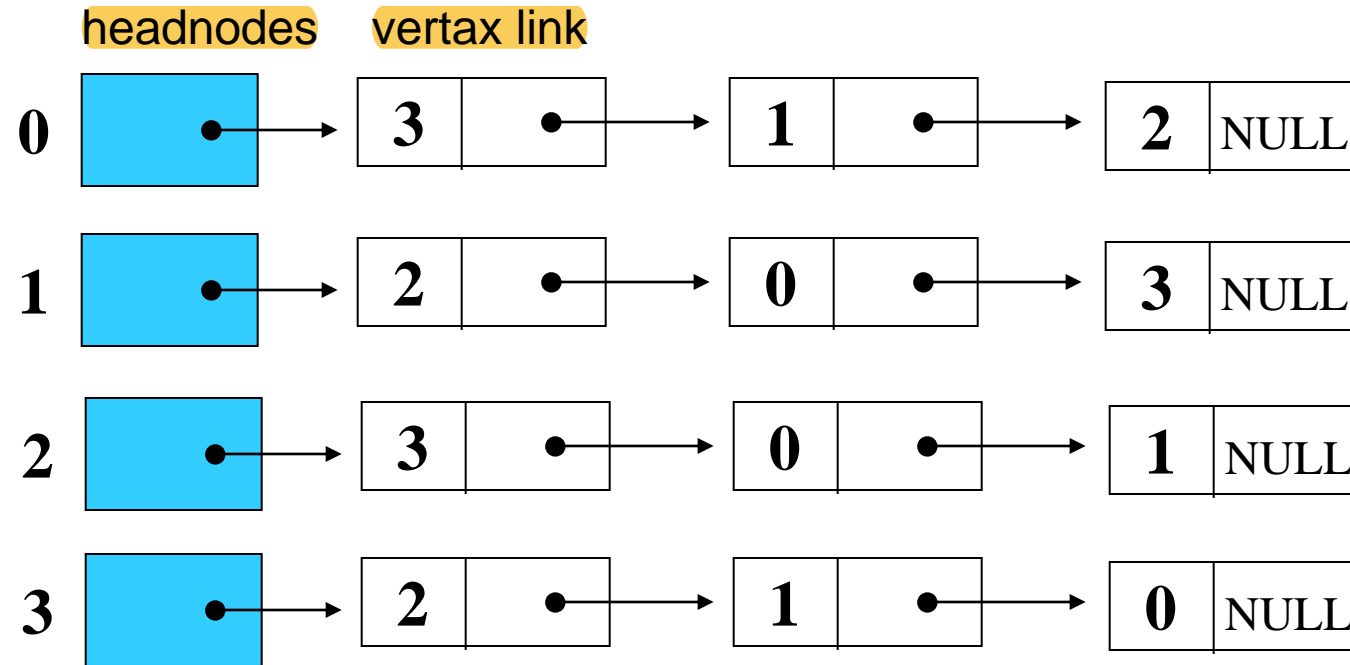


$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$



# Alternate order adjacency list for G1

Order is of no significance.



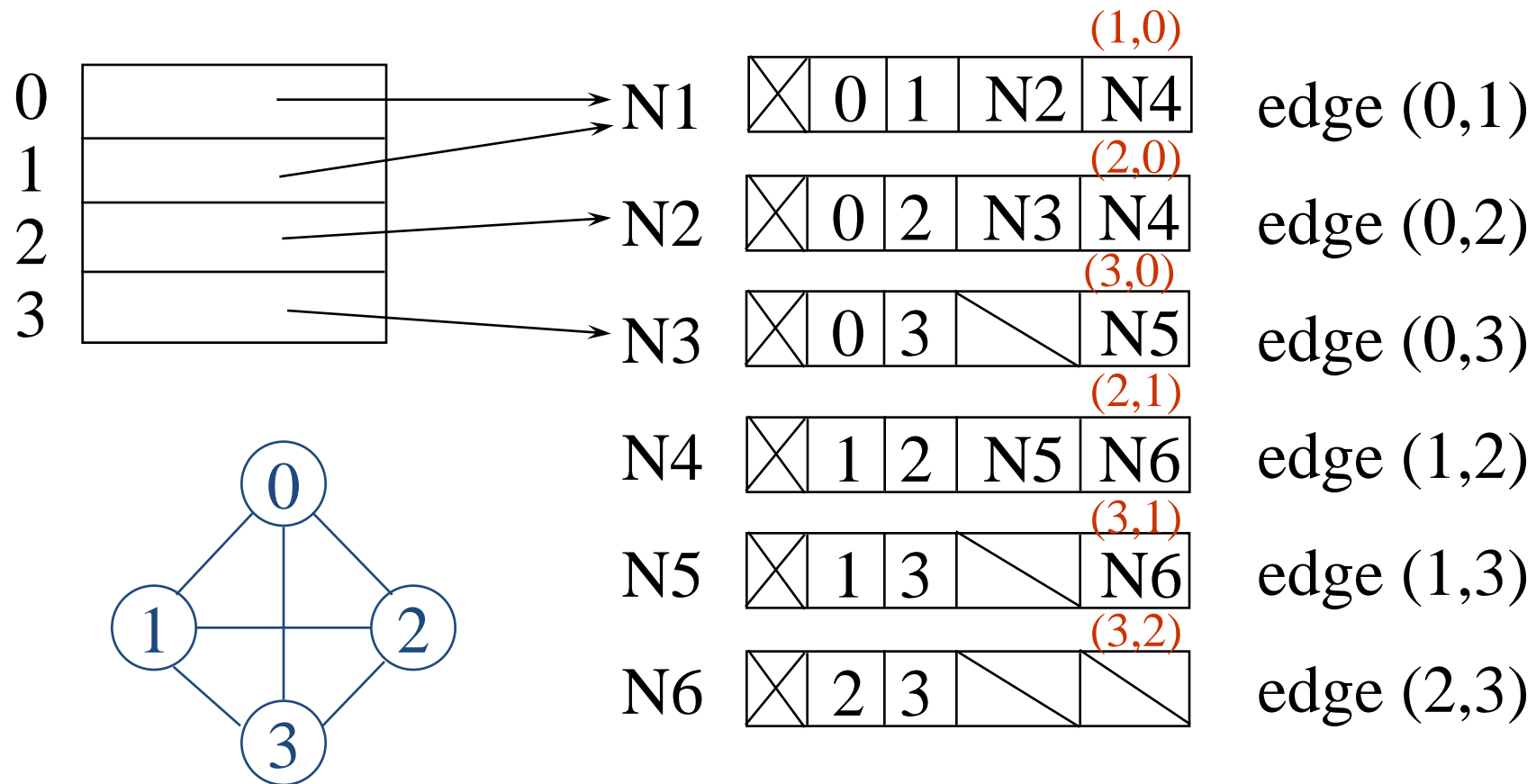
# Adjacency Multilists

- An edge in an undirected graph is represented by two nodes in adjacency list representation.
- Adjacency Multilists  
lists in which nodes may be shared among several lists.  
(an edge is shared by two different paths)

marked	vertex1	vertex2	path1	path2
--------	---------	---------	-------	-------

# Example for Adjacency Multlists

Lists: vertex 0: M1->M2->M3, vertex 1: M1->M4->M5  
vertex 2: M2->M4->M6, vertex 3: M3->M5->M6



six edges

# Some Graph Operations

- Traversal

Given  $G=(V,E)$  and vertex  $v$ , find all  $w \in V$ , such that  $w$  connects  $v$ .

Depth First Search (DFS)

preorder tree traversal

Breadth First Search (BFS)

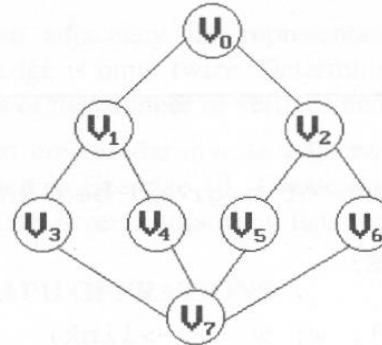
level order tree traversal

- Connected Components
- Spanning Trees

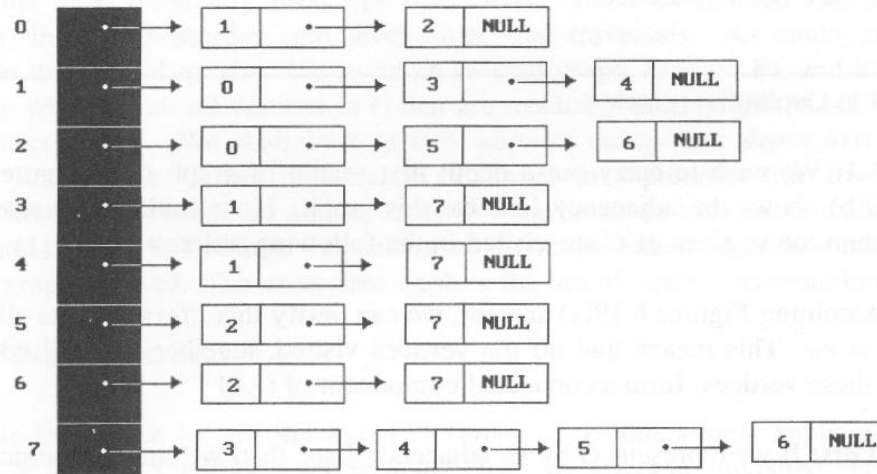


# Graph G and its adjacency lists

depth first search: v0, v1, v3, v7, v4, v5, v2, v6



(a)



(b)

breadth first search: v0, v1, v2, v3, v4, v5, v6, v7

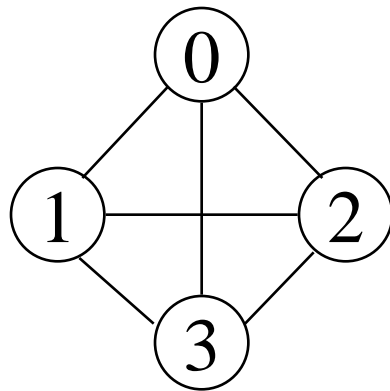


# Spanning Trees

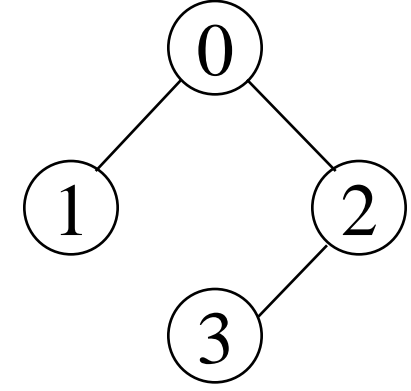
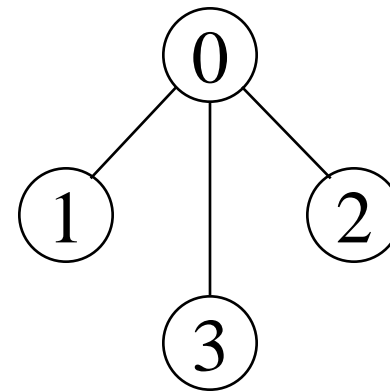
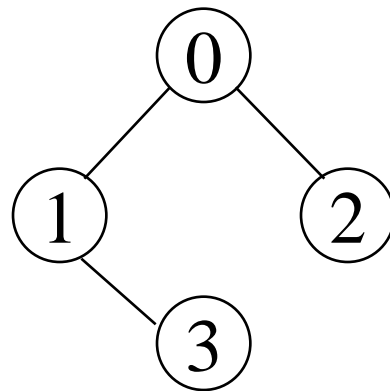
- When graph  $G$  is connected, a depth first or breadth first search starting at any vertex will visit all vertices in  $G$
- A spanning tree is any tree that consists solely of edges in  $G$  and that includes all the vertices
- $E(G): T$  (tree edges) +  $N$  (nontree edges)

where  $T$ : set of edges used during search  
 $N$ : set of remaining edges

## Examples of Spanning Tree



**G<sub>1</sub>**

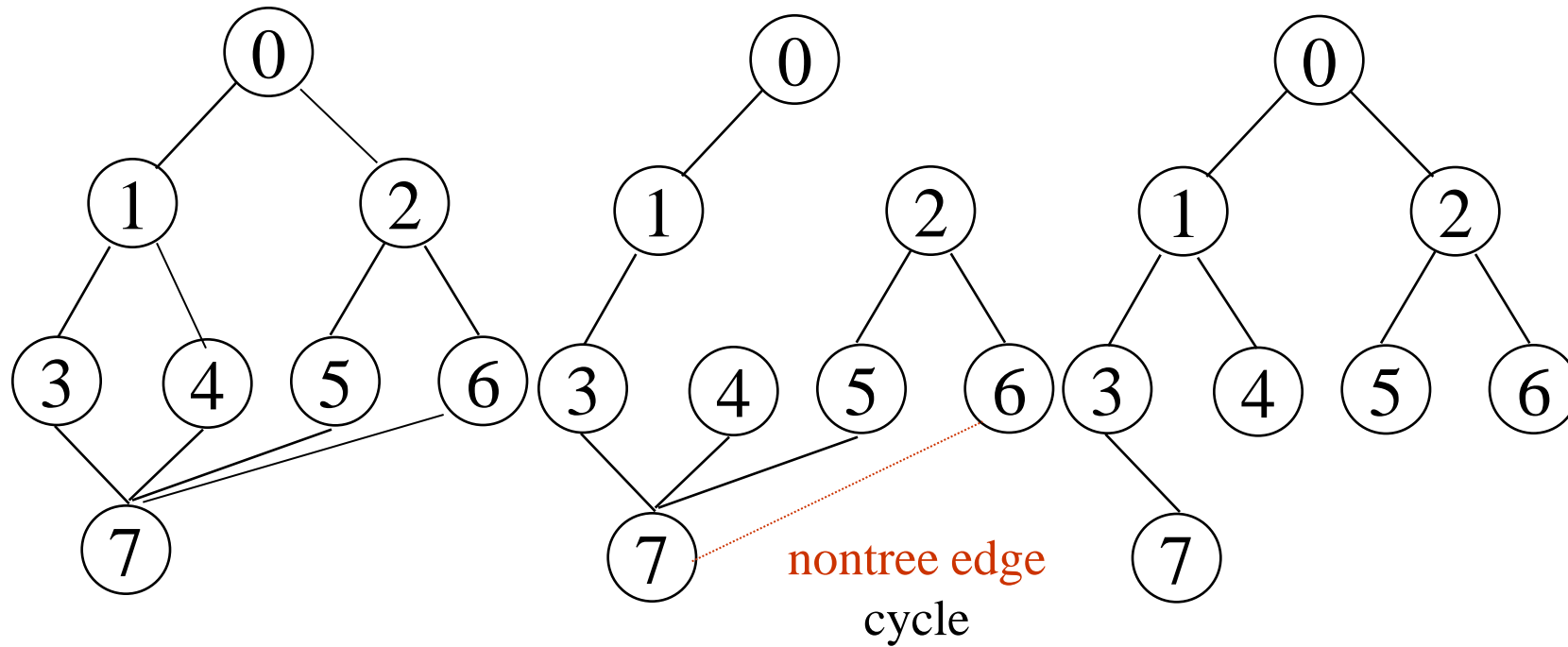


Possible spanning trees

# Spanning Trees

- Either dfs or bfs can be used to create a spanning tree
  - When dfs is used, the resulting spanning tree is known as a depth first spanning tree
  - When bfs is used, the resulting spanning tree is known as a breadth first spanning tree
- While adding a nontree edge into any spanning tree, this will create a cycle

## DFS VS BFS Spanning Tree



DFS Spanning

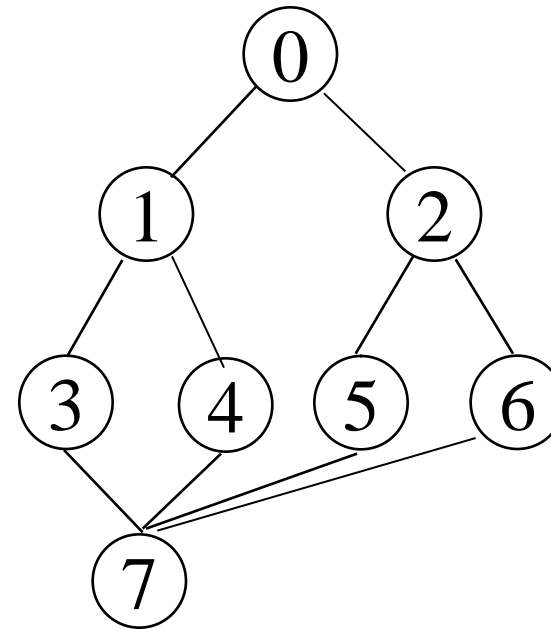
BFS Spanning

A spanning tree is a **minimal subgraph**,  $G'$ , of  $G$  such that  $V(G')=V(G)$  and  $G'$  is **connected**.

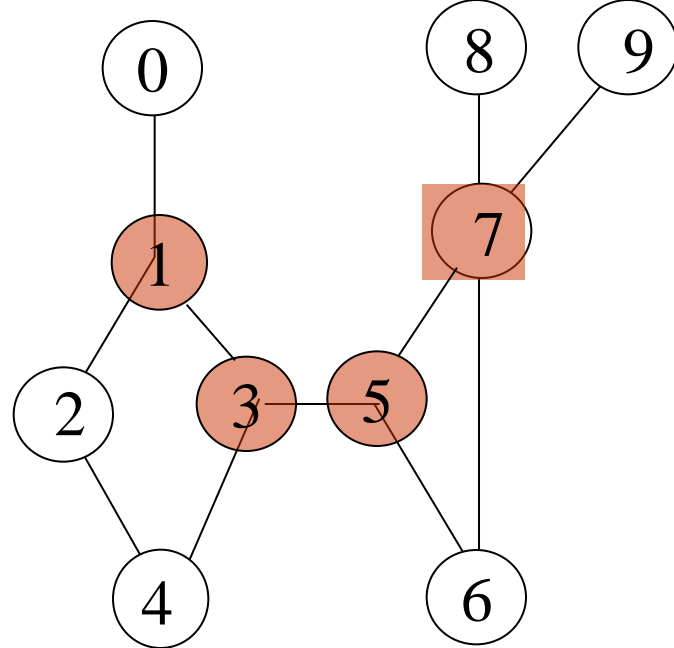
Any connected graph with  $n$  vertices must have at least  $n-1$  edges.

A **biconnected graph** is a **connected graph** that has no **articulation points**.

biconnected graph

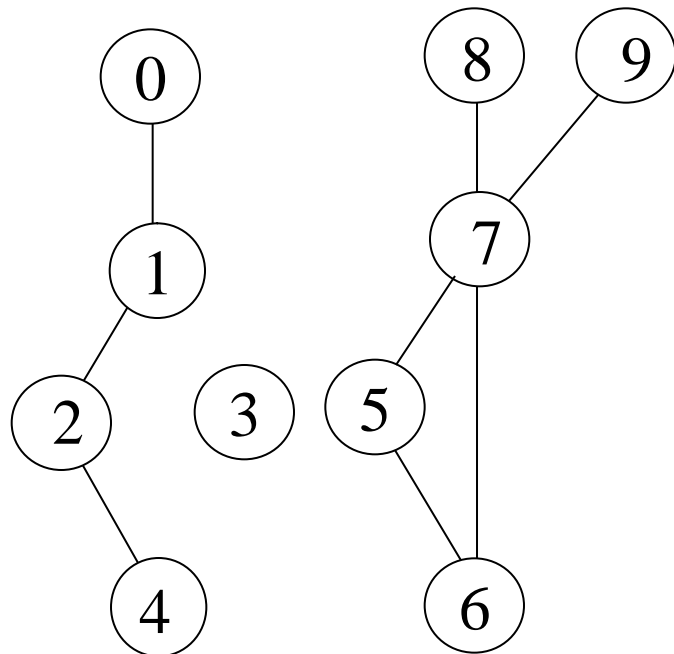




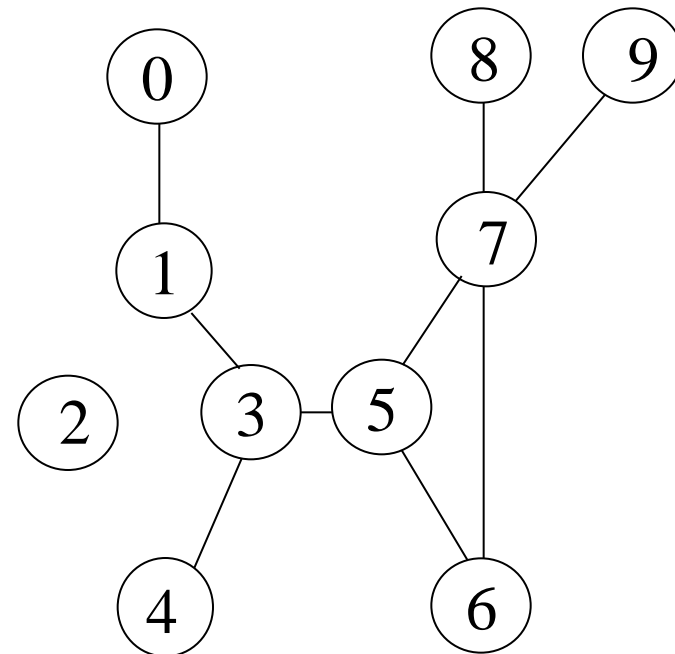


connected graph

two connected components

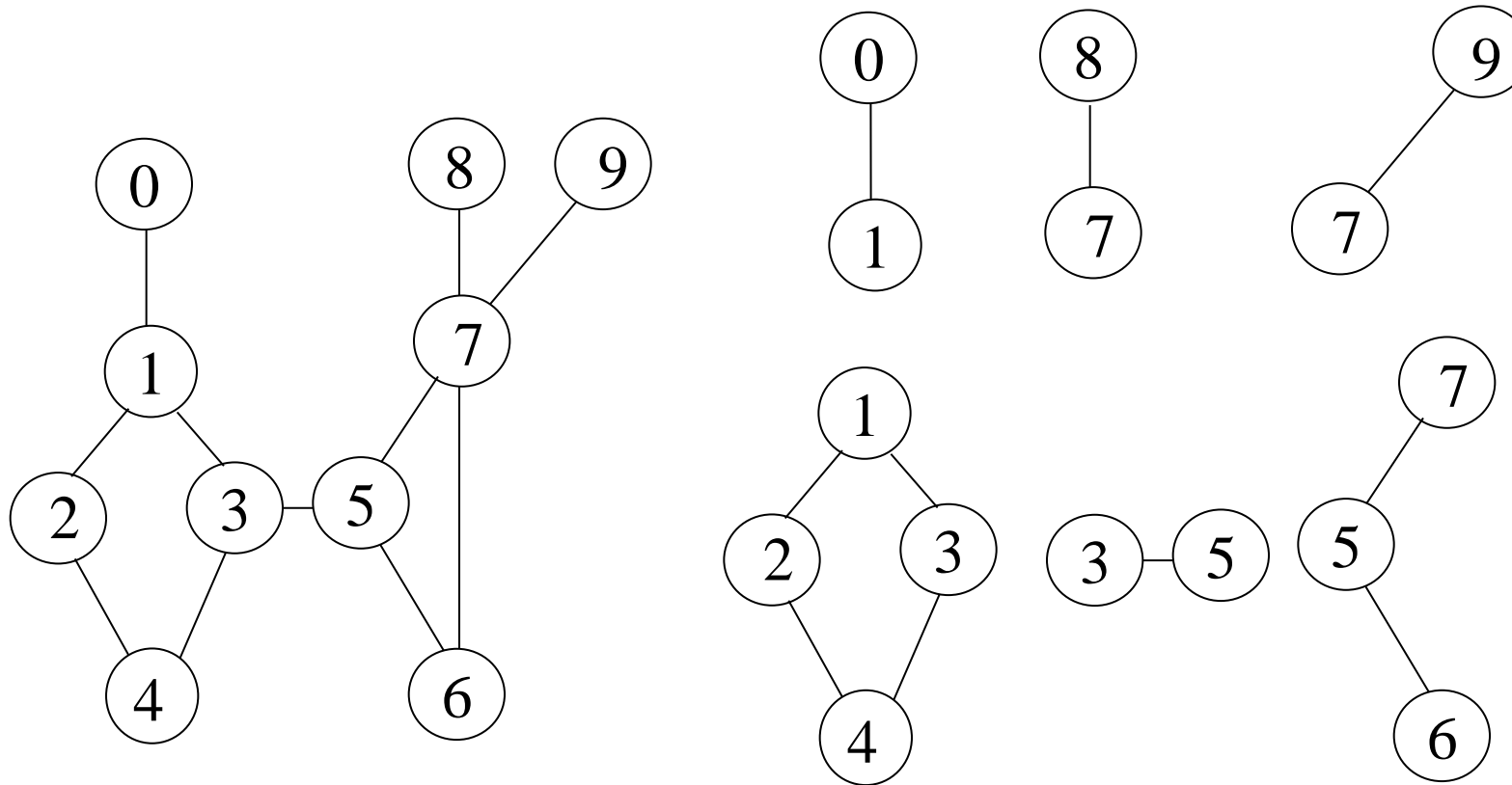


one connected graph



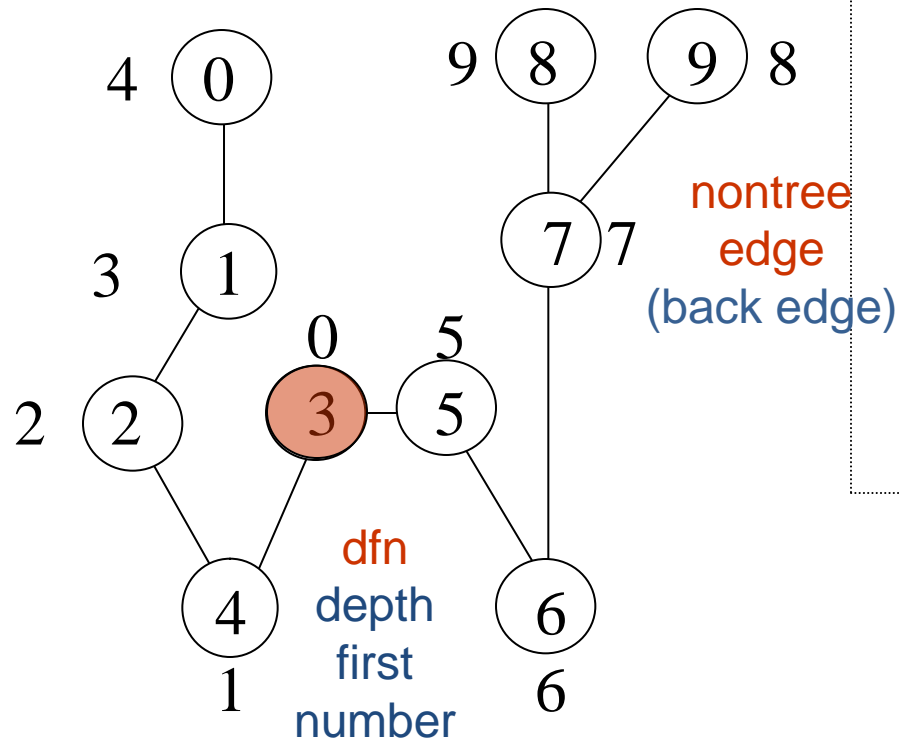


**Biconnected component:** a maximal connected subgraph  $H$   
(no subgraph that is both biconnected and properly contains  $H$ )

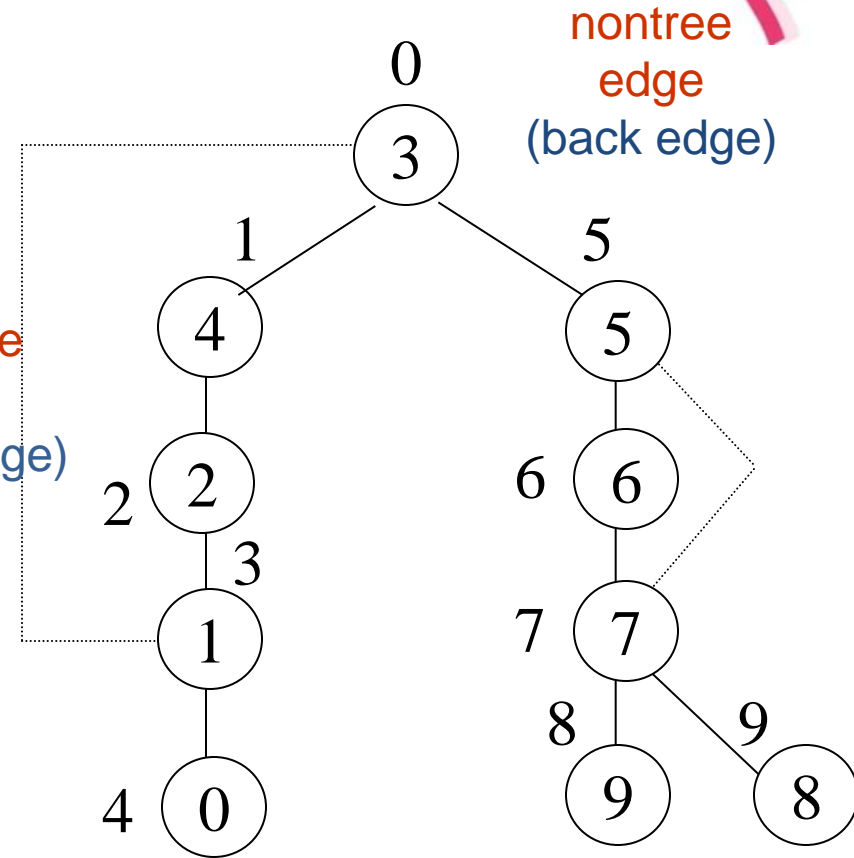


biconnected components

Find biconnected component of a connected undirected graph by **depth first spanning tree**



(a) depth first spanning tree



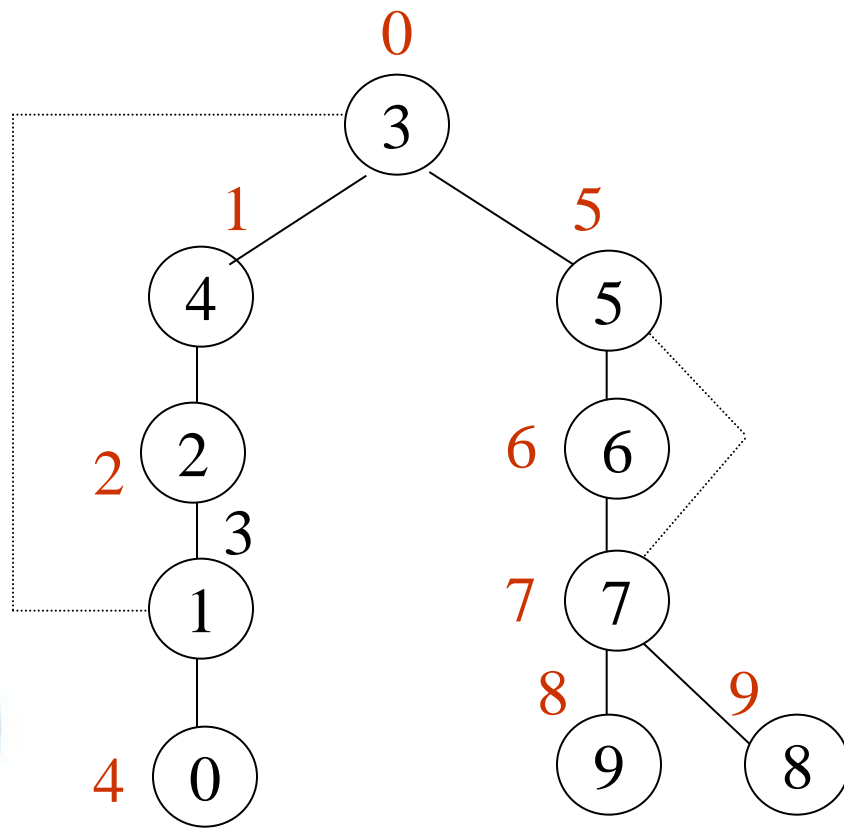
Why is cross edge impossible?

(b)

If  $u$  is an ancestor of  $v$  then  $\text{dfn}(u) < \text{dfn}(v)$ .

*dfn* and *low* values for *dfs* spanning tree with *root* = 3

Vertax	0	1	2	3	4	5	6	7	8	9
<i>dfn</i>	4	3	2	0	1	5	6	7	9	8
<i>low</i>	4	0	0	0	0	5	5	5	9	8



- The root of a depth first spanning tree is an articulation point iff it has at least two children.

- Any other vertex  $u$  is an articulation point iff it has at least one child  $w$  such that we cannot reach an ancestor of  $u$  using a path that consists of

(1) only  $w$  (2) descendants of  $w$  (3) single back edge.

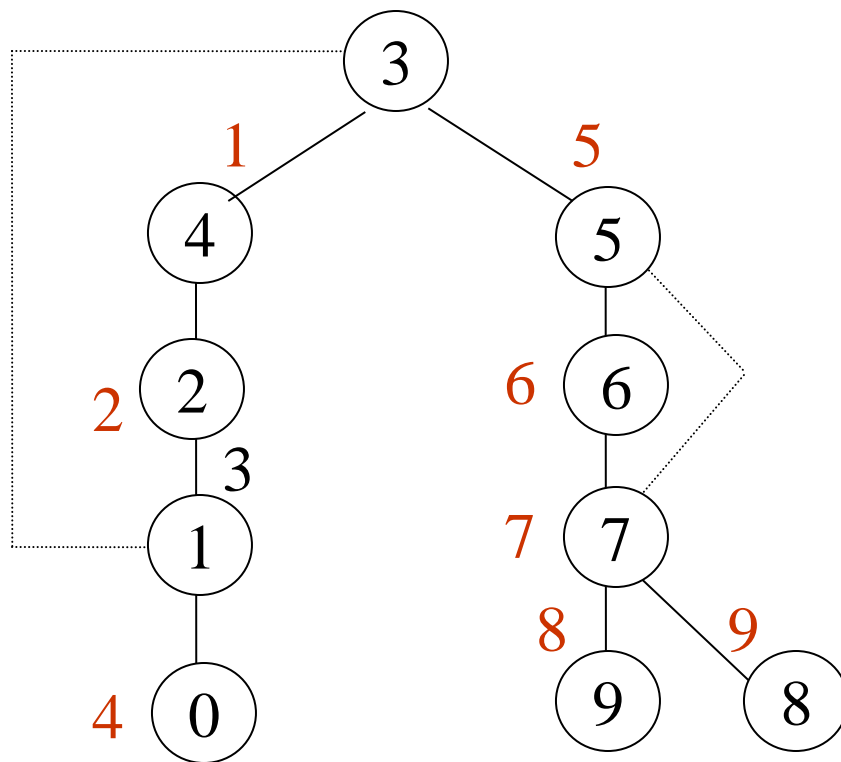
$\text{low}(u) = \min\{\text{dfn}(u),$   
 $\min\{\text{low}(w) \mid w \text{ is a child of } u\},$   
 $\min\{\text{dfn}(w) \mid (u, w) \text{ is a back edge}\}$

$u$ : articulation point  
 $\text{low}(\text{child}) \geq \text{dfn}(u)$

vertex	dfn	low	child	low_child	low:dfn
0	4	4 (4,n,n)	null	null	null:4
1	3	0 (3,4,0)	0	4	$4 \geq 3$ •
2	2	0 (2,0,n)	1	0	$0 < 2$
3	0	0 (0,0,n)	4,5	0,5	$0,5 \geq 0$ •
4	1	0 (1,0,n)	2	0	$0 < 1$
5	5	5 (5,5,n)	6	5	$5 \geq 5$ •
6	6	5 (6,5,n)	7	5	$5 < 6$
7	7	5 (7,8,5)	8,9	9,8	$9,8 \geq 7$ •
8	9	9 (9,n,n)	null	null	null, 9
9	8	8 (8,n,n)	null	null	null, 8









# Minimum Cost Spanning Tree

- The cost of a spanning tree of a weighted undirected graph is the sum of the costs of the edges in the spanning tree
- A minimum cost spanning tree is a spanning tree of least cost
- Three different algorithms can be used
  - Kruskal
  - Prim
  - Sollin

Select  $n-1$  edges from a weighted graph of  $n$  vertices with minimum cost.

## Greedy Strategy

- An optimal solution is constructed in stages
- At each stage, the best decision is made at this time
- Since this decision cannot be changed later, we make sure that the decision will result in a feasible solution
- Typically, the selection of an item at each stage is based on a least cost or a highest profit criterion

## Kruskal's Idea

- Build a minimum cost spanning tree  $T$  by adding edges to  $T$  one at a time
- Select the edges for inclusion in  $T$  in nondecreasing order of the cost
- An edge is added to  $T$  if it does not form a cycle
- Since  $G$  is connected and has  $n > 0$  vertices, exactly  $n-1$  edges will be selected



0 ~~10~~ 5

2 ~~12~~ 3

1 ~~14~~ 6

1 ~~16~~ 2

3 ~~18~~ 6

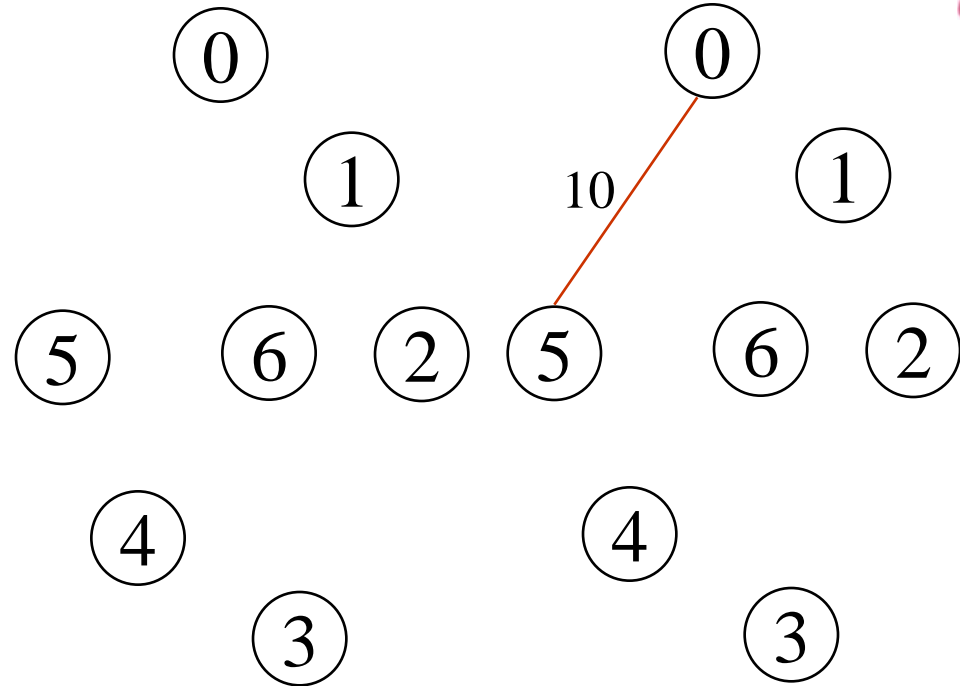
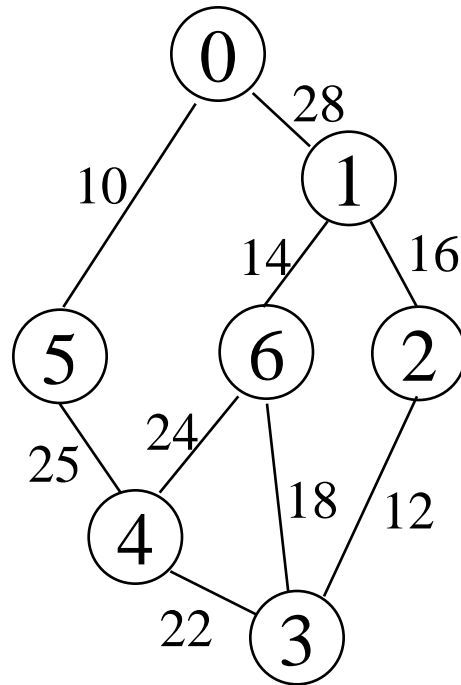
3 ~~22~~ 4

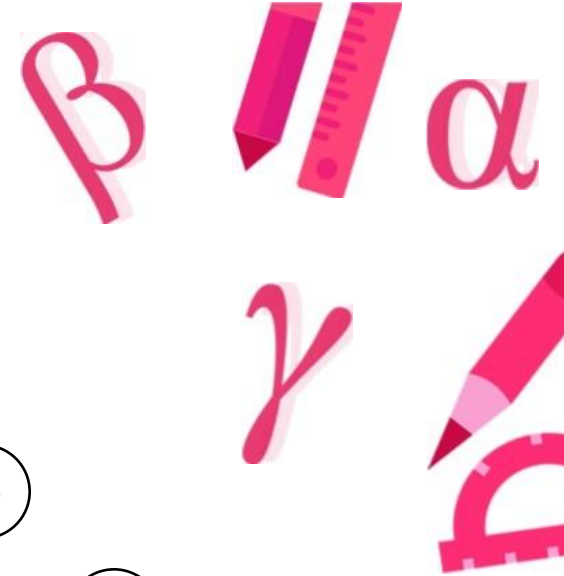
4 ~~24~~ 6

4 ~~25~~ 5

0 ~~28~~ 1

## Examples for Kruskal's Algorithm





$$0 - 10 - 5$$

$$2 - 12 - 3$$

$$1 - 14 - 6$$

$$1 - 16 - 2$$

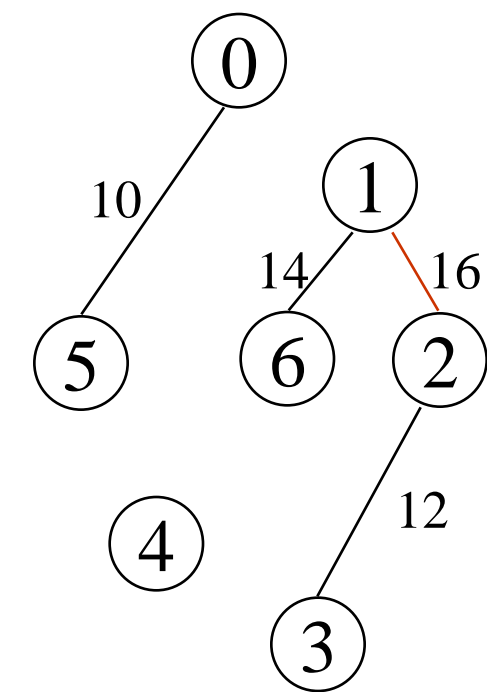
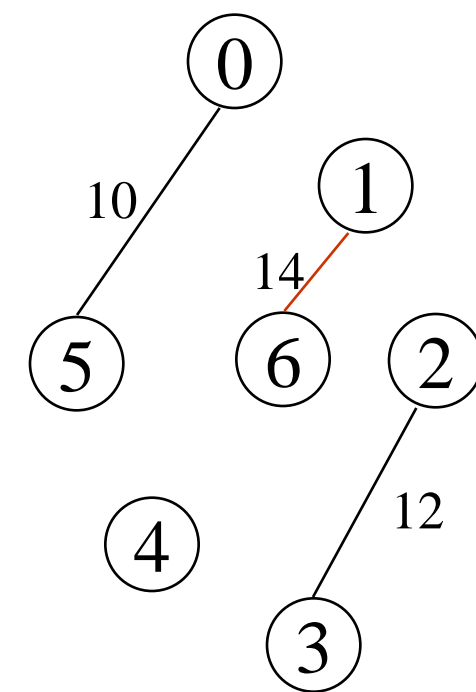
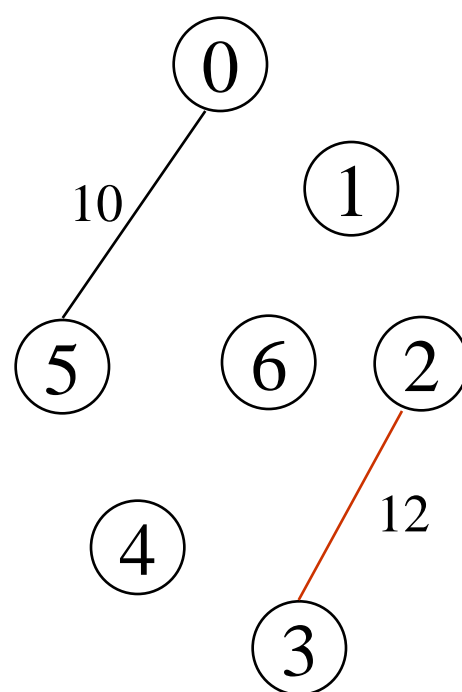
$$3 - 18 - 6$$

$$3 - 22 - 4$$

$$4 - 24 - 6$$

$$4 - 25 - 5$$

$$0 - 28 - 1$$



↓ + 3 — 6  
cycle

0 ~~10~~ 5

2 ~~12~~ 3

1 ~~14~~ 6

1 ~~16~~ 2

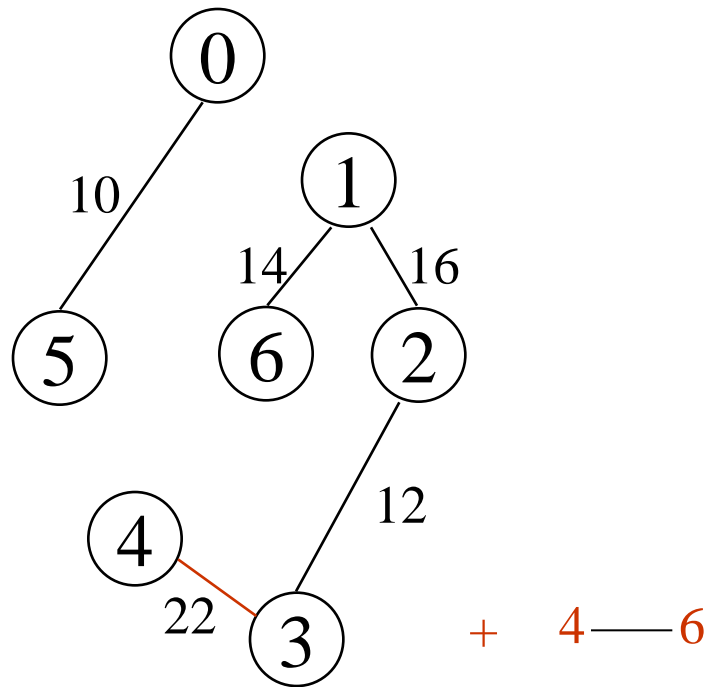
3 ~~18~~ 6

3 ~~22~~ 4

4 ~~24~~ 6

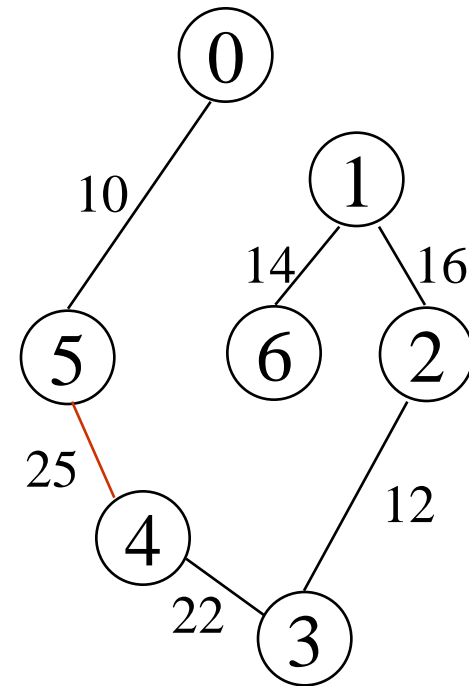
4 ~~25~~ 5

0 ~~28~~ 1



+ 4 — 6

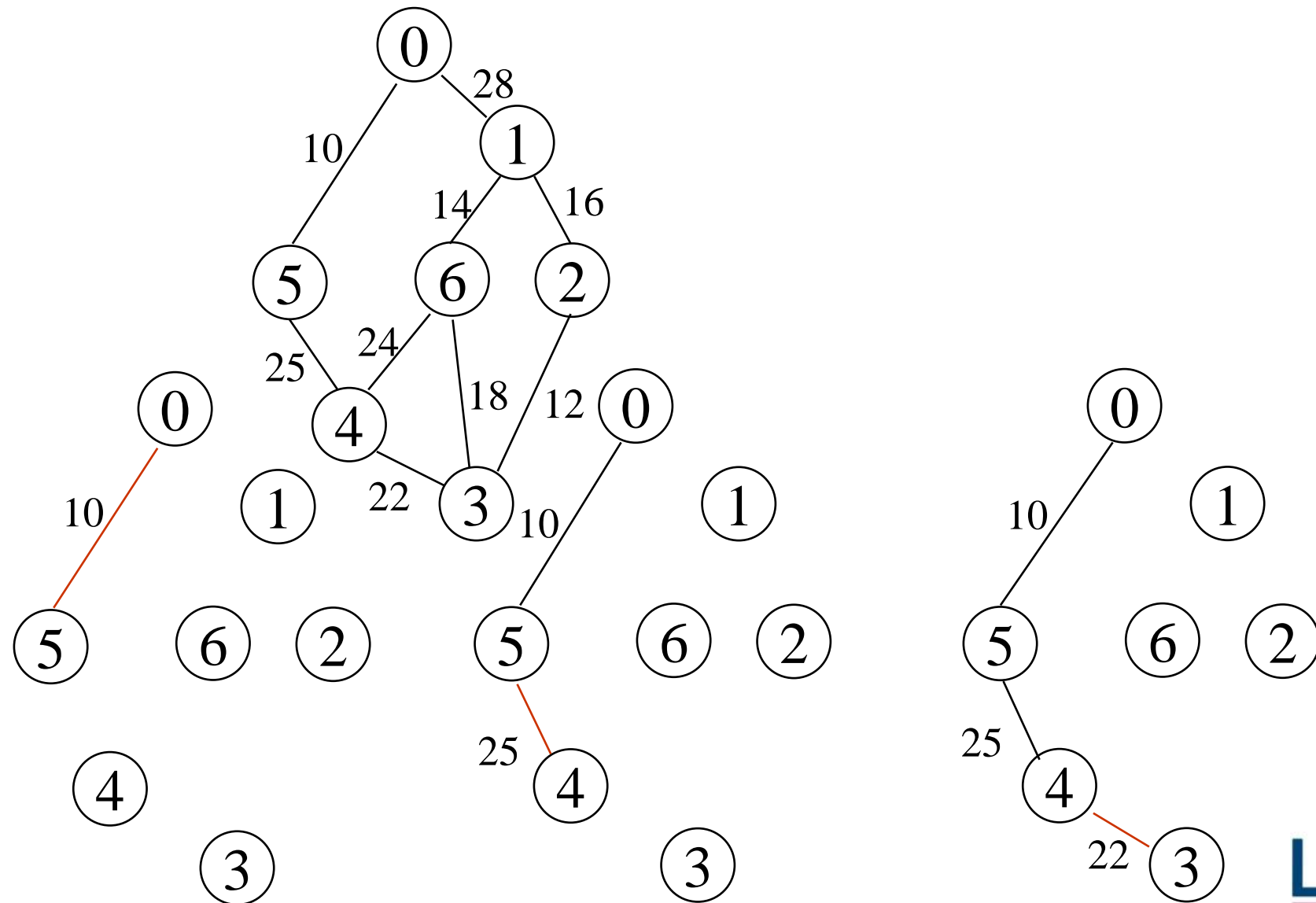
cycle

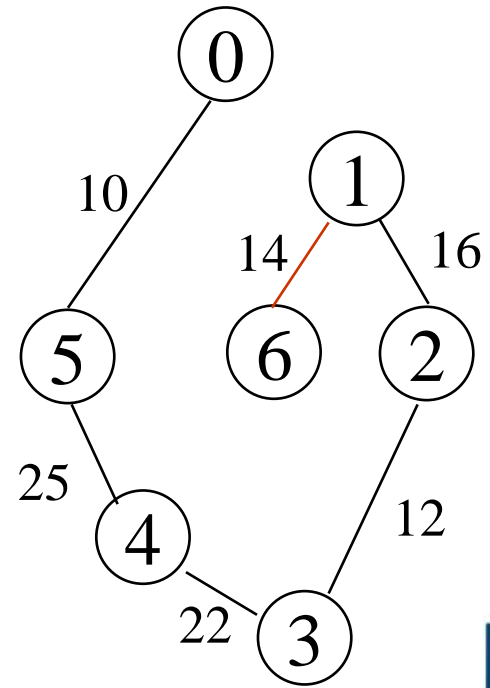
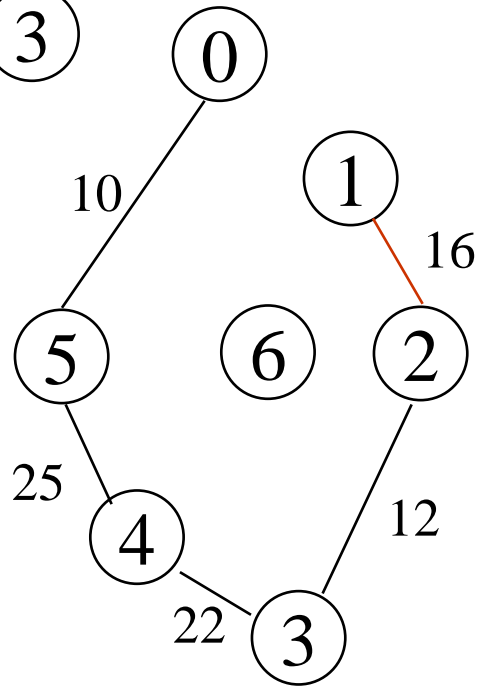
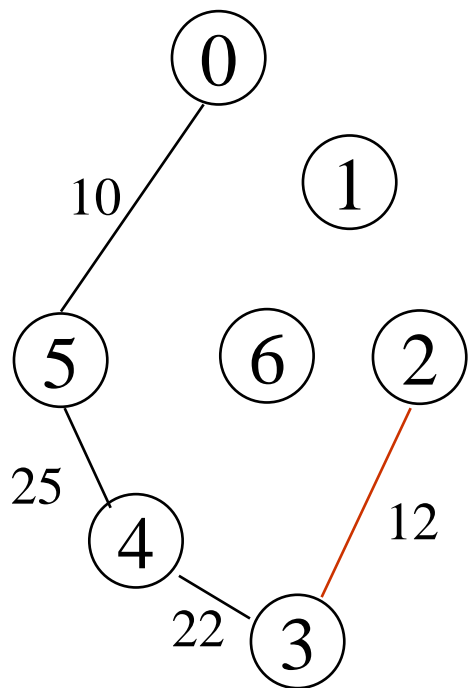
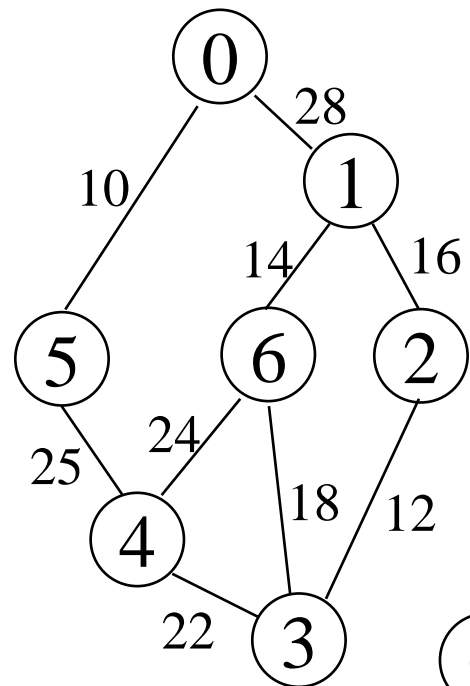


cost = 10 + 25 + 22 + 12 + 16 + 14

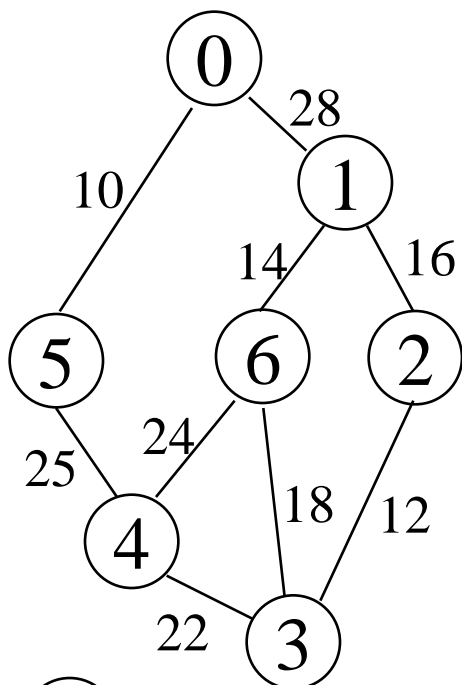


# Examples for Prim's Algorithm

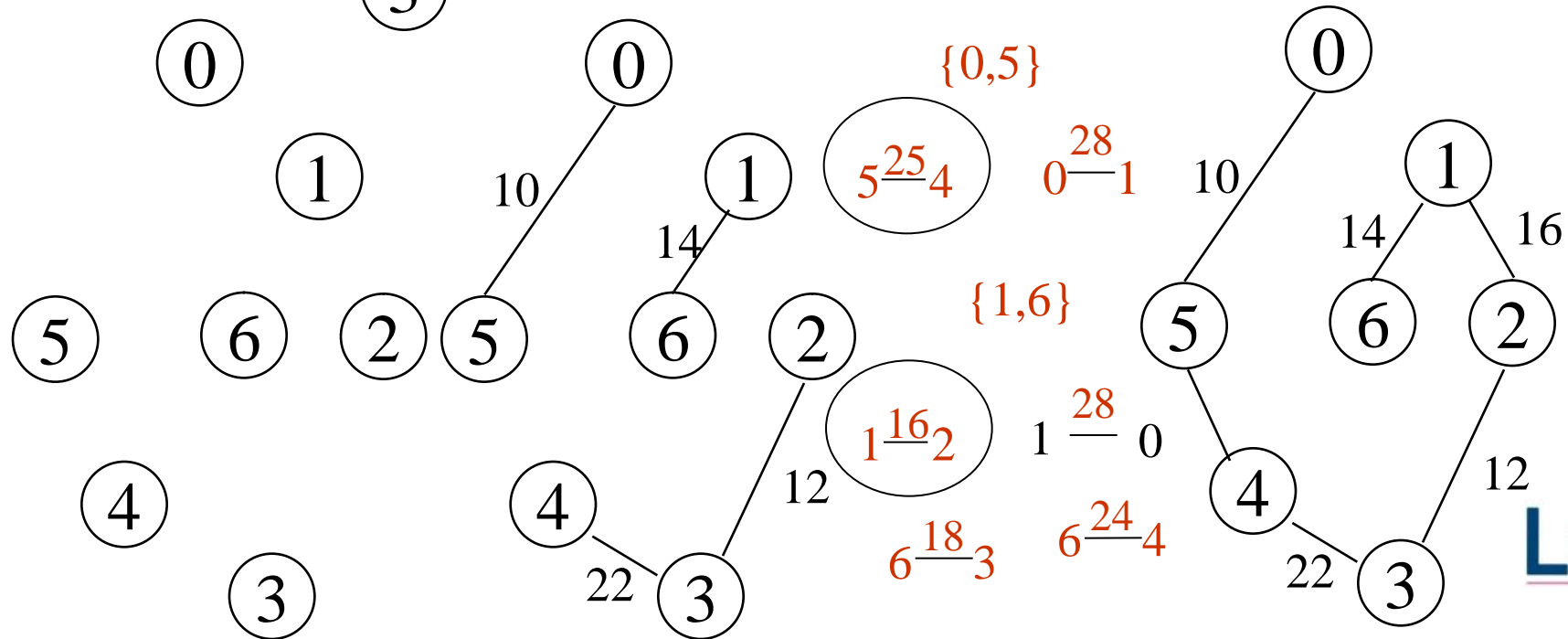




# Sollin's Algorithm

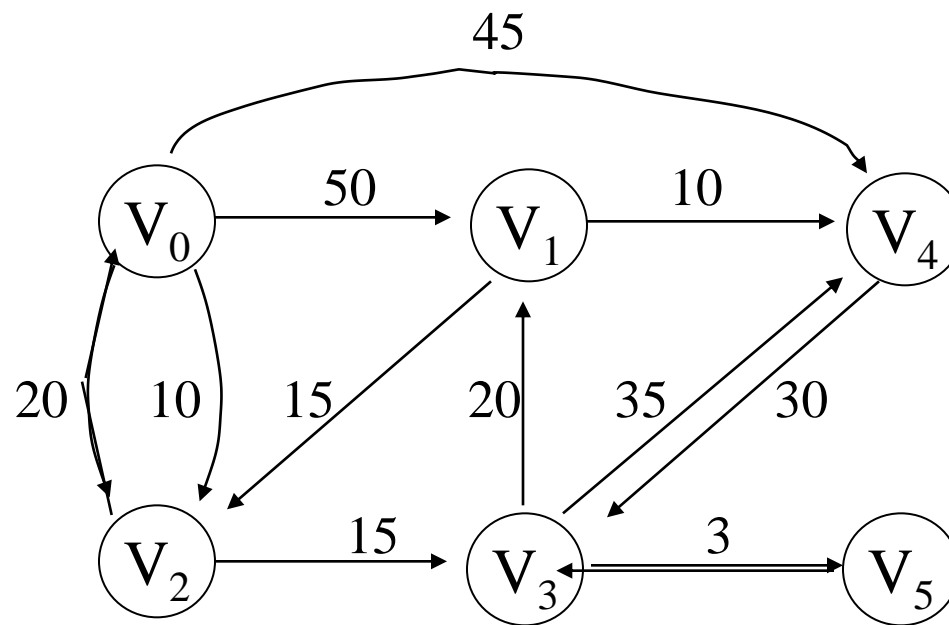


vertex	edge
0	0 -- 10 --> 5, 0 -- 28 --> 1
1	1 -- 14 --> 6, 1 -- 16 --> 2, 1 -- 28 --> 0
2	2 -- 12 --> 3, 2 -- 16 --> 1
3	3 -- 12 --> 2, 3 -- 18 --> 6, 3 -- 22 --> 4
4	4 -- 22 --> 3, 4 -- 24 --> 6, 5 -- 25 --> 5
5	5 -- 10 --> 0, 5 -- 25 --> 4
6	6 -- 14 --> 1, 6 -- 18 --> 3, 6 -- 24 --> 4



# Single Source All Destinations

Determine the shortest paths from  $v_0$  to all the remaining vertices.

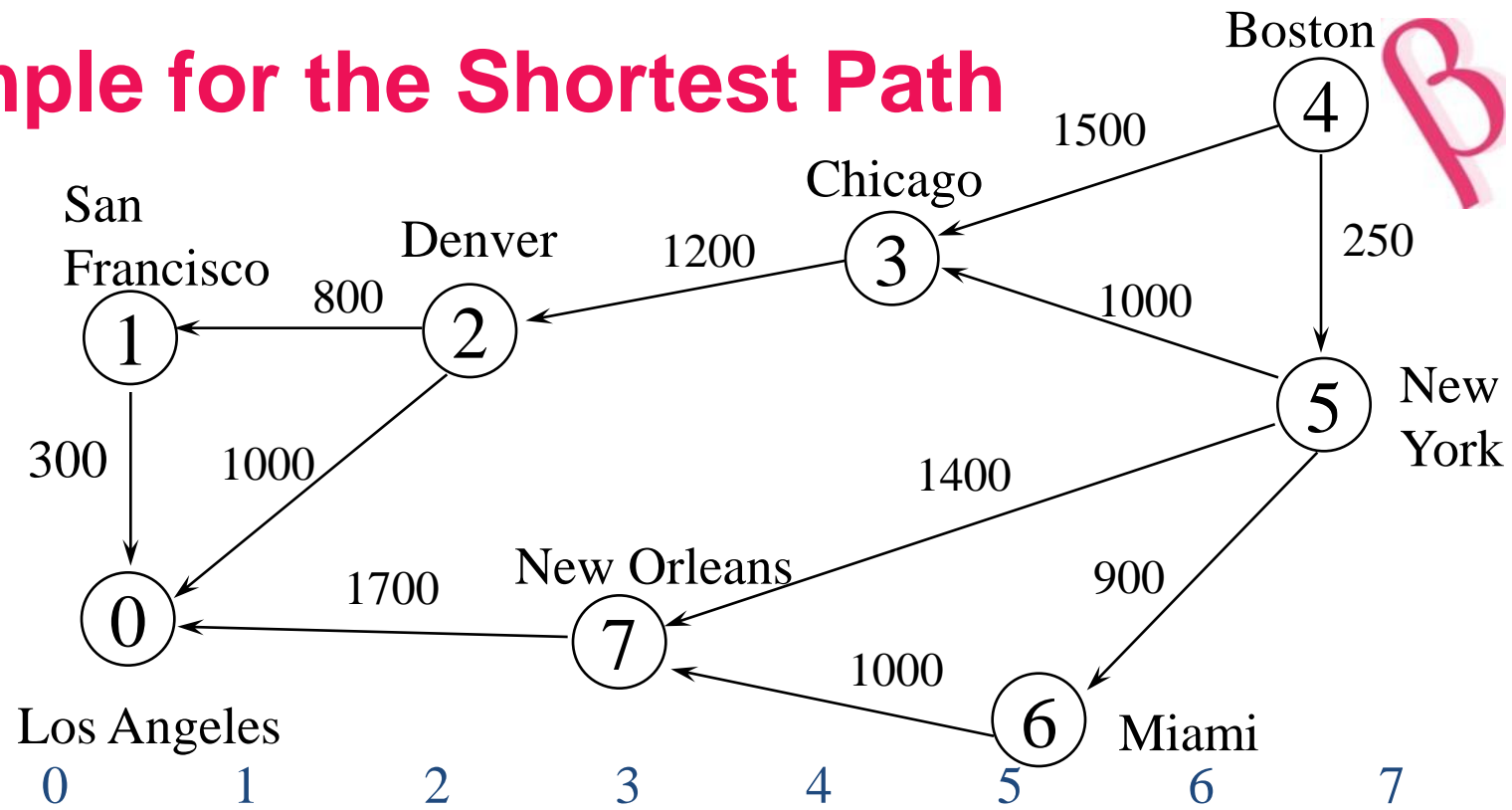


(a)

path	length
1) $v_0 v_2$	10
2) $v_0 v_2 v_3$	25
3) $v_0 v_2 v_3 v_1$	45
4) $v_0 v_4$	45

(b)

# Example for the Shortest Path

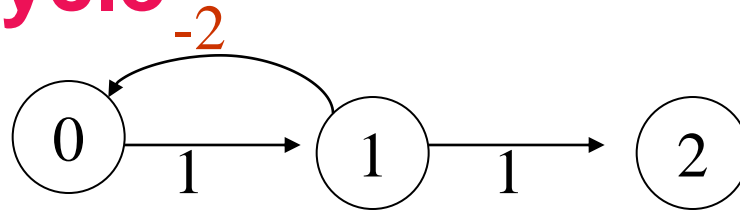


	0	1	2	3	4	5	6	7
0	0							
1	300	0						
2	1000	800	0					
3			1200	0				
4				1500	0	250		
5				1000		0	900	1400
6							0	1000
7	1700							0

Cost adjacency matrix



## Graph with negative cycle



(a) Directed graph

$$\begin{bmatrix} 0 & 1 & \infty \\ -2 & 0 & 1 \\ \infty & \infty & 0 \end{bmatrix}$$

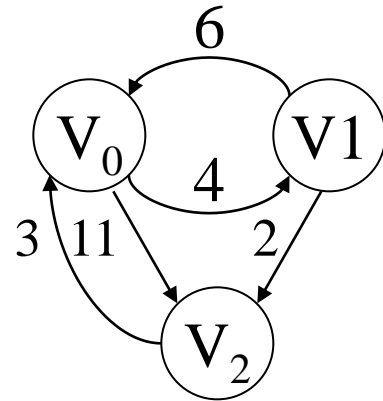
(b)  $A^{-1}$

The length of the shortest path from vertex 0 to vertex 2 is  $-\infty$ .

$$0, 1, 0, 1, 0, 1, \dots, 0, 1, 2$$



## Directed graph and its cost matrix



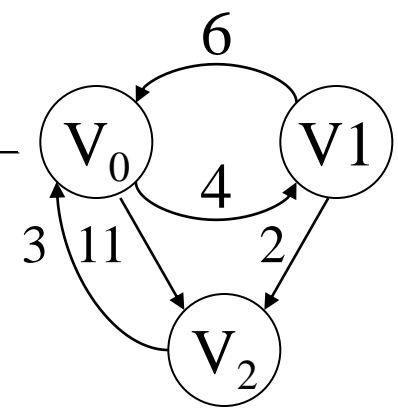
(a) Digraph G

	0	1	2
0	0	4	11
1	6	0	2
2	3	$\infty$	0

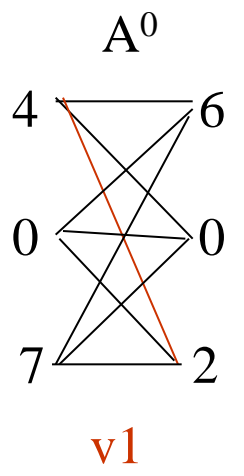
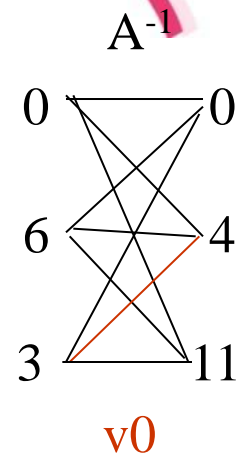
(b) Cost adjacency matrix for G



$A^{-1}$	0	1	2
0	0	4	11
1	6	0	2
2	3	$\infty$	0

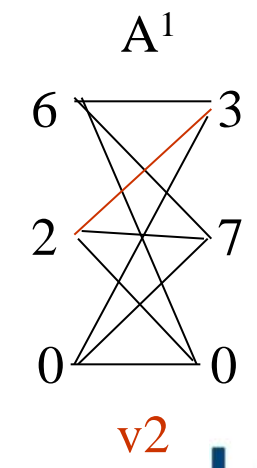


$A^0$	0	1	2
0	0	4	11
1	6	0	2
2	3	7	0



$A^1$	0	1	2
0	0	4	6
1	6	0	2
2	3	7	0

$A^2$	0	1	2
0	0	4	6
1	5	0	2
2	3	7	0



# Transitive Closure

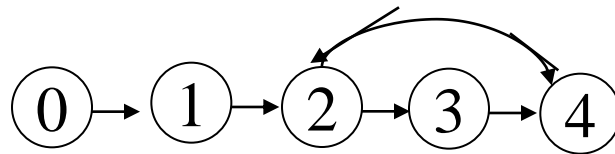
Goal: given a graph with unweighted edges, determine if there is a path from  $i$  to  $j$  for all  $i$  and  $j$ .

(1) Require positive path ( $> 0$ ) lengths.

transitive closure matrix

(2) Require nonnegative path ( $\geq 0$ ) lengths.

reflexive transitive closure matrix



(a) Digraph G

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

(b) Adjacency matrix A for G

$$A^+ = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

cycle

(c) transitive closure matrix  $A^+$

There is a path of length  $> 0$

$$A^* = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

reflexive

(d) reflexive transitive closure matrix  $A^*$

There is a path of length  $\geq 0$



**Thank You!**