MODULE 2-4 [CPE11202]
Graph

Assc. Prof. Dr. Natasha Dejdumrong

KING MONGKUT'S UNIVERSITY
OF TECHNOLOGY THONBURI

Outlines

- Graph Definition
- Graph Terminology
- Degree of Graph
- Graph Representation

Graph

★ A graph is an abstract data structure that is used to implement the mathematical concept of graphs.

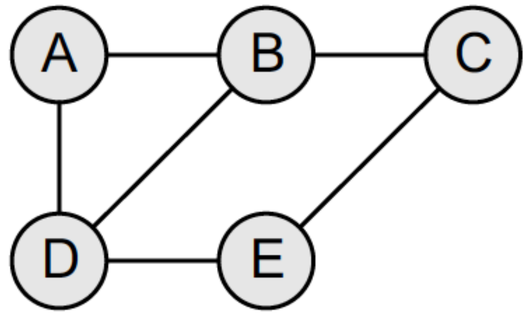
★ It is basically a collection of vertices (also called nodes) and edges that connect these vertices.

★ A graph is often viewed as a generalization of the tree structure, where instead of having a purely parent-to-child relationship between tree nodes, any kind of complex relationship can exist.

Graph : Definition

★ A graph G is defined as an ordered set (V, E), where V(G) represents the set of vertices and E(G) represents the edges that connect these vertices.

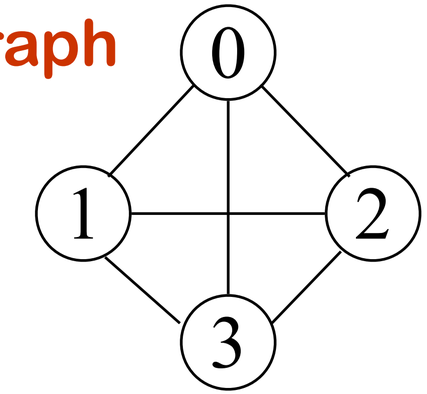
★ Figure shows a graph with
V(G) = {A, B, C, D and E} and
E(G) = {(A, B), (B, C), (A, D), (B, D), (D, E), (C, E)}.



Note that there are five vertices or nodes and six edges in the graph.

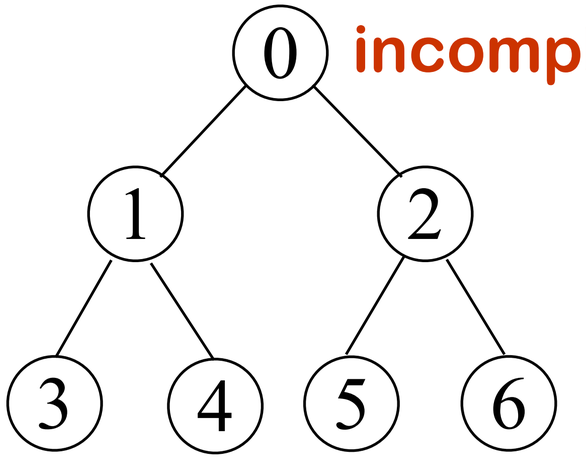
Graph : Examples of Graphs

complete graph

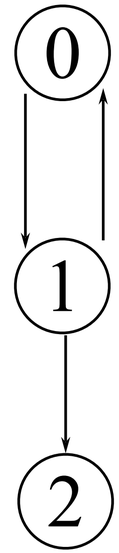


G_1

incomplete graph



G_2



G_3

$V(G_1)=\{0,1,2,3\}$
$V(G_2)=\{0,1,2,3,4,5,6\}$
$V(G_3)=\{0,1,2\}$

$E(G_1)=\{(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)\}$
$E(G_2)=\{(0,1),(0,2),(1,3),(1,4),(2,5),(2,6)\}$
$E(G_3)=\{<0,1>,<1,0>,<1,2>\}$

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

Graph : Why is it useful?

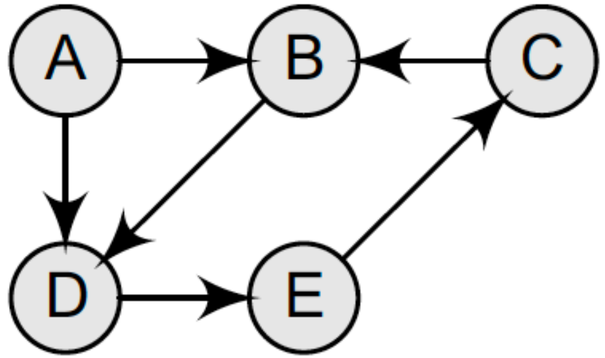
★ Graphs are widely used to model any situation where entities or things are related to each other in pairs.

★ For example, the following information can be represented by graphs:

- ★ Family trees in which the member nodes have an edge from parent to each of their children.
- ★ Transportation networks in which nodes are airports, intersections, ports, etc. The edges can be airline flights, one-way roads, shipping routes, etc.

Directed Graph vs Undirected Graph

- A graph can be directed or undirected. In an undirected graph, edges do not have any direction associated with them. That is, if an edge is drawn between nodes A and B, then the nodes can be traversed from A to B as well as from B to A.
- Look at the figure which shows a directed graph.

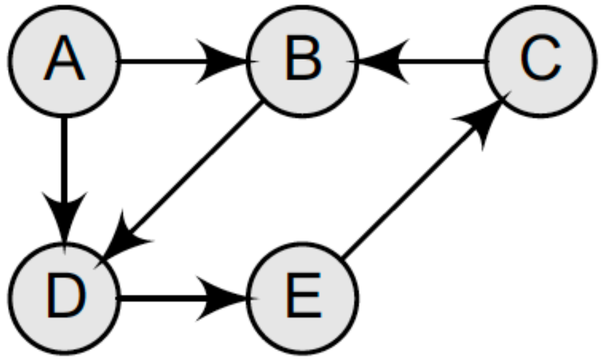


Directed Graph vs Undirected Graph

★ An undirected graph is one in which the pair of vertices in a edge is unordered, (v0, v1) = (v1,v0).

★ A directed graph is one in which each edge is a directed pair of vertices, (v0, v1) != (v1, v0).

★ In a directed graph, edges form an ordered pair. If there is an edge from A to B, then there is a path from A to B but not from B to A. The edge (A, B) is said to initiate from node A (also known as initial node) and terminate at node B (terminal node).



Graph: Degree

★ Adjacent nodes or neighbours For every edge, e = (u, v) that connects nodes u and v, the nodes u and v are the end-points and are said to be the adjacent nodes or neighbours.

★ Degree of a node Degree of a node u, deg(u), is the total number of edges containing the node u.

★ If deg(u) = 0, it means that u does not belong to any edge and such a node is known as an isolated node.

Graph Terminology

★ Regular graph It is a graph where each vertex has the same number of neighbours. That is, every node has the same degree.

★ A regular graph with vertices of degree k is called a k–regular graph or a regular graph of degree k. Figure 13.3 shows regular graphs.

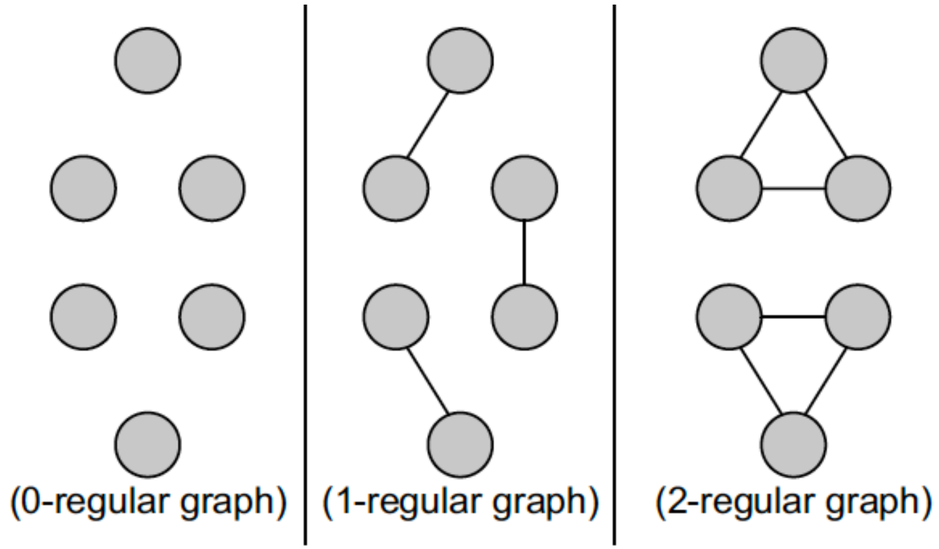


Figure 13.3 Regular graphs

Graph Terminology

★ Path A path P written as P = {v0, v1, v2, ..., vn), of length n from a node u to v is defined as a sequence of (n+1) nodes.
Here, u = v0, v = vn and vi–1 is adjacent to vi for i = 1, 2, 3, ..., n.

★ Closed path A path P is known as a closed path if the edge has the same end-points. That is, if v0 = vn.

★ Simple path A path P is known as a simple path if all the nodes in the path are distinct with an exception that v0 may be equal to vn. If v0 = vn, then the path is called a closed simple path.

Graph Terminology

★ Cycle A path in which the first and the last vertices are same.
A simple cycle has no repeated edges or vertices (except the first and last vertices).

★ Connected graph A graph is said to be connected
if for any two vertices (u, v) in V there is a path from u to v.
That is to say that there are no isolated nodes in a connected graph.

★ A connected graph that does not have any cycle is called a tree.
Therefore, a tree is treated as a special graph (Refer Fig. 13.4(b)).

Graph Terminology

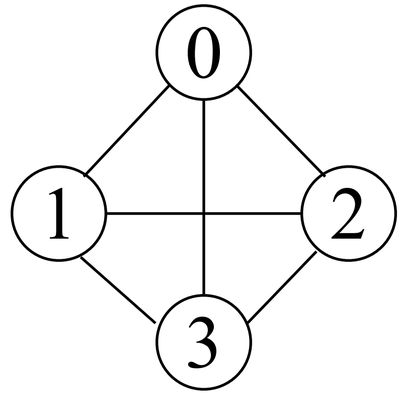
★ Complete graph A graph G is said to be complete if all its nodes are fully connected and has the maximum number of edges.
That is, there is a path from one node to every other node in the graph.

A complete graph has n(n–1)/2 edges, where n is the number of nodes in G.

★For undirected graph with n vertices, the maximum number of edges is n(n-1)/2

★For directed graph with n vertices, the maximum number of edges is n(n-1)

★Example: G1 is a complete graph



complete graph G_1

Graph Terminology

★ Labelled graph or weighted graph A graph is said to be labelled if every edge in the graph is assigned some data. In a weighted graph, the edges of the graph are assigned some weight or length. The weight of an edge denoted by w(e) is a positive value which indicates the cost of traversing the edge. Figure 13.4(c) shows a weighted graph.

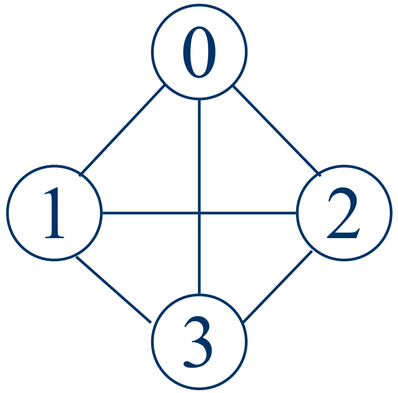
Subgraph and Path

★ 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).

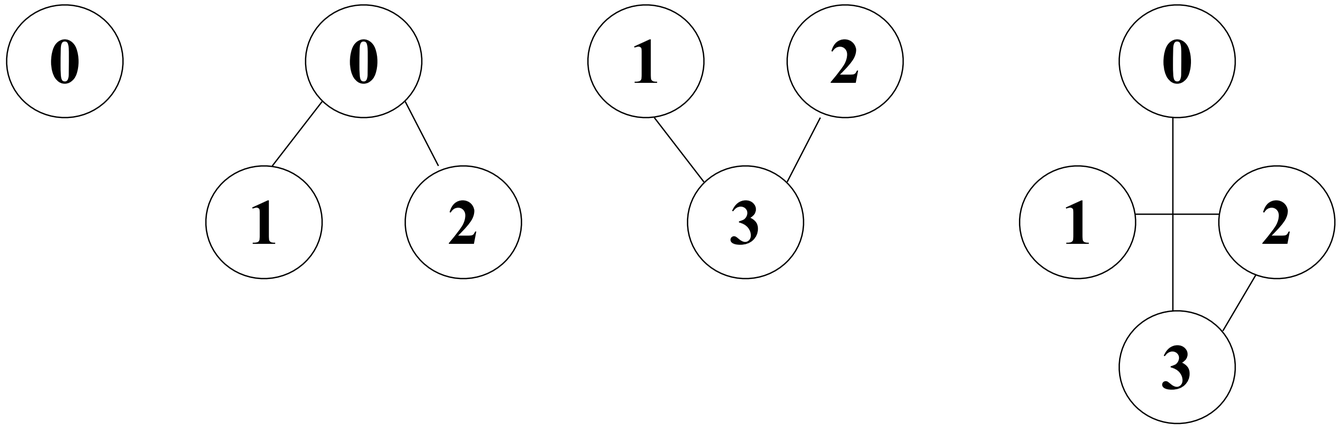
★ A path from vertex v_p to vertex v_q in a graph G, is a sequence of vertices, v_p, v_{i1}, v_{i2}, ..., v_{in}, v_q, such that (v_p, v_{i1}), (v_{i1}, v_{i2}), ..., (v_{in}, v_q) are edges in an undirected graph

★ The length of a path is the number of edges on it.

Subgraph of G1



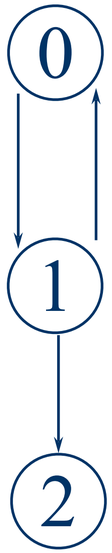
G_1



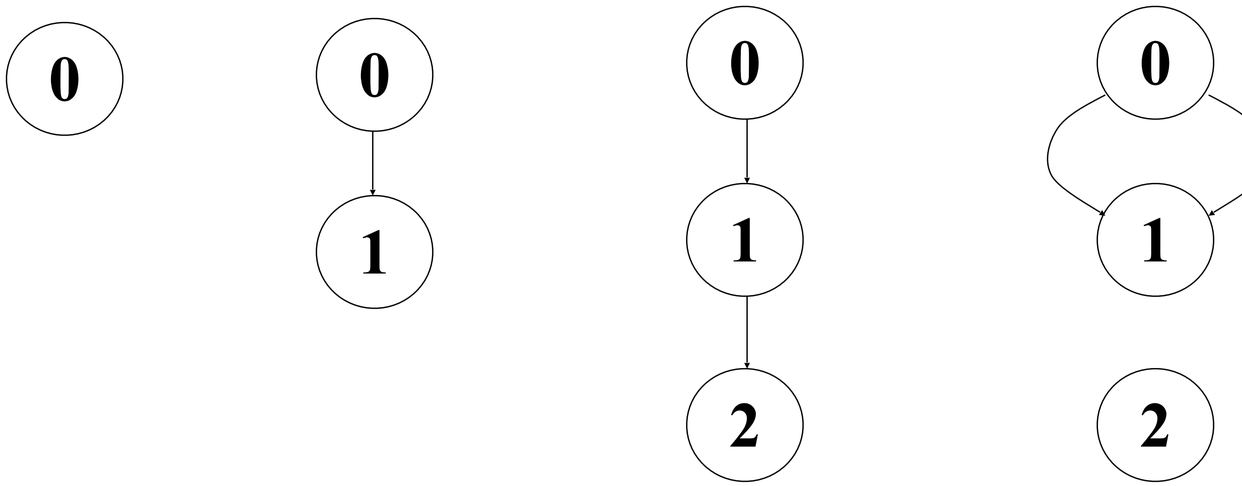
(i) (ii) (iii) (iv)

(a) Some of the subgraph of G_1

Subgraph of G3



G_3



(b) Some of the subgraph of G_3

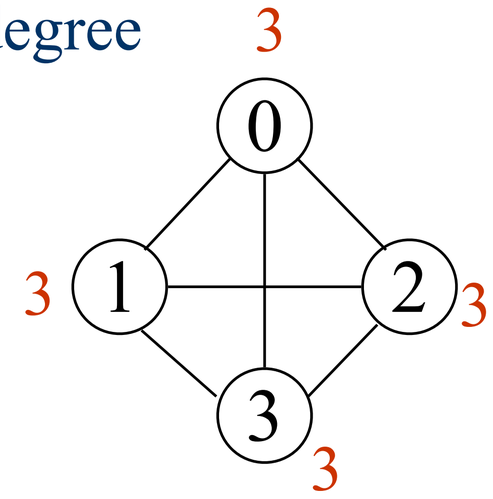
Degree

- 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 di is the degree of a vertex i in a graph G with n vertices and e edges, the number of edges is $$e = (\sum_{0}^{n-1} d_i) / 2$$

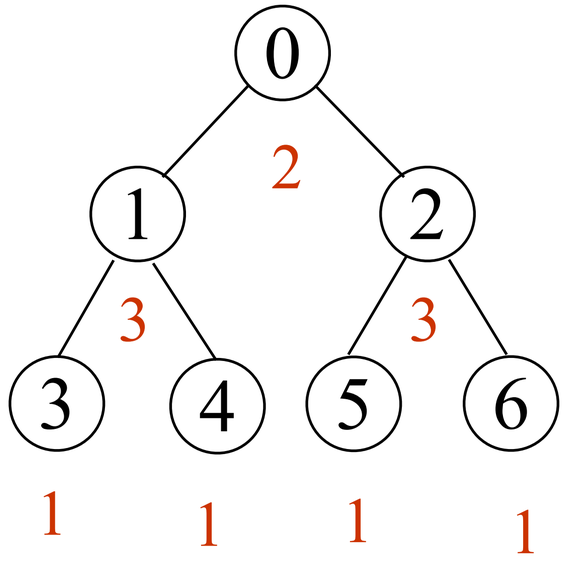
Degree

Undirected graph

degree

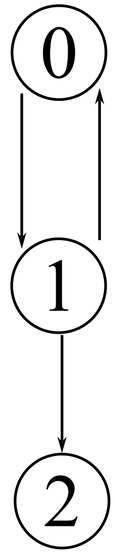


G_1



G_2

Directed graph in-degree
out-degree



in:1, out: 1

in: 1, out: 2

in: 1, out: 0

G_3

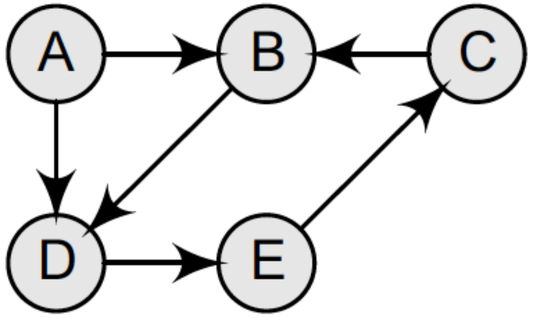
Graph Representation

- Adjacency Matrix
- Adjacency Lists
- Adjacency Multi-Lists

Adjacency Matrix

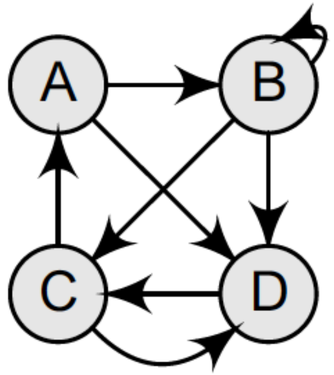
- Let G=(V,E) be a graph with n vertices.
- The adjacency matrix of G is a two-dimensional n by n array, say adj_mat
- If the edge (v_i, v_j) is in E(G), adj_mat[I][j]=1
- If there is no such edge in E(G), adj_mat[I][j]=0
- The adjacency matrix for an undirected graph is symmetric; the adjacency matrix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

Examples of Adjacency Matrix



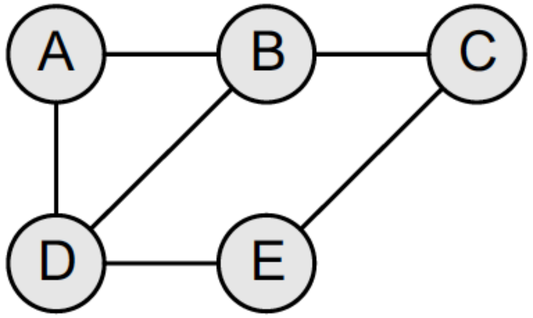
	A	B	C	D	E
A	0	1	0	1	0
B	0	0	0	1	0
C	0	1	0	0	0
D	0	0	0	0	1
E	0	0	1	0	0

(a) Directed graph



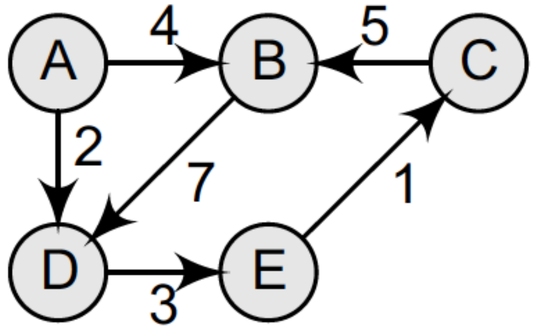
	A	B	C	D
A	0	1	0	1
B	0	1	1	1
C	1	0	0	1
D	0	0	1	0

(b) Directed graph with loop



	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

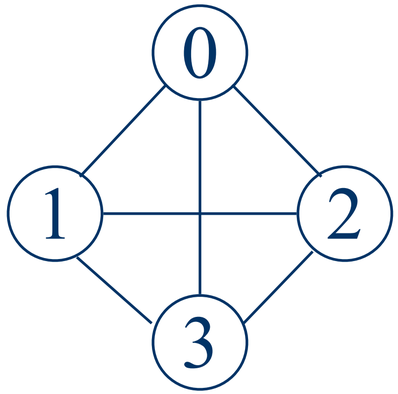
(c) Undirected graph



	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

(d) Weighted graph

Examples of Adjacency Matrix

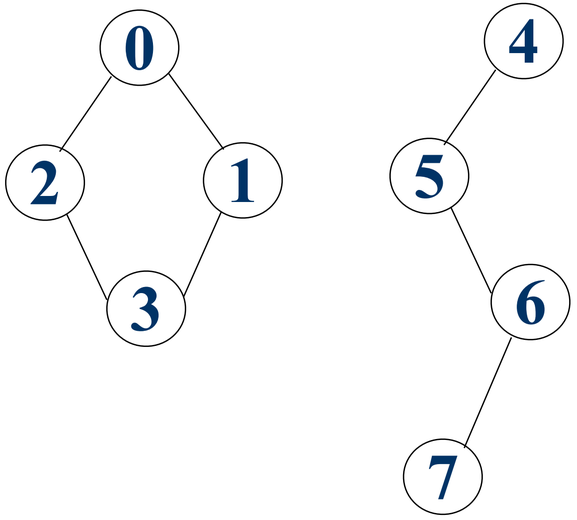


G_1

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

G_2

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



G_4

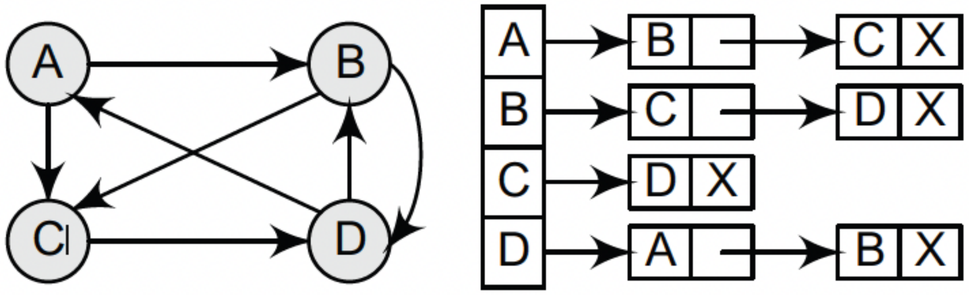
$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 &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

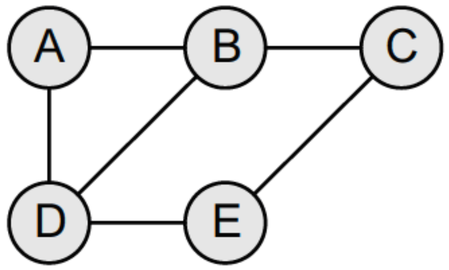
symmetric

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

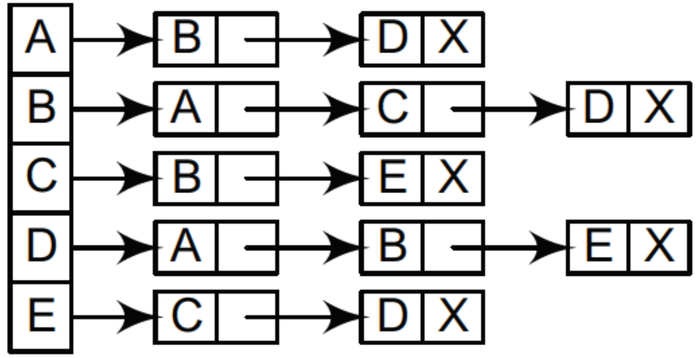
Adjacency List

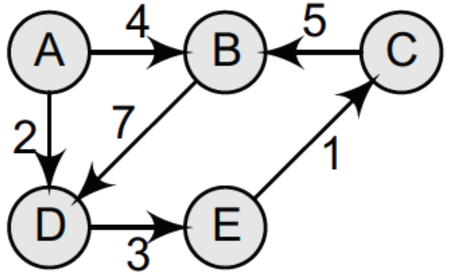
★ Each row in adjacency matrix is represented as an adjacency list.



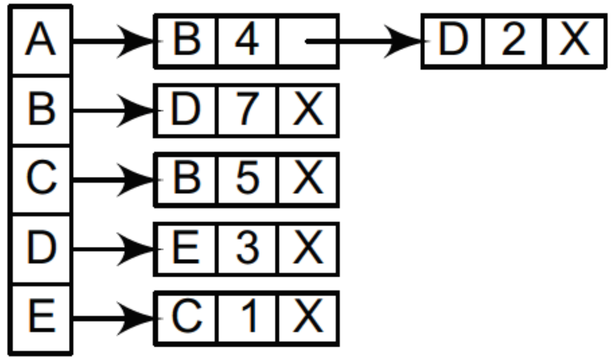


(Undirected graph)





(Weighted graph)

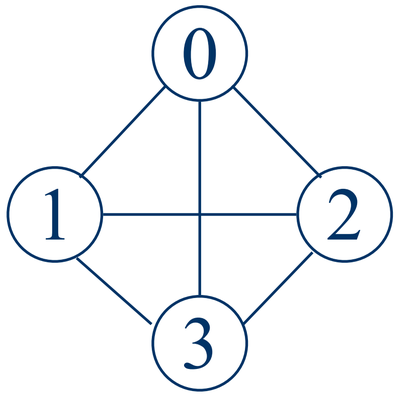


Data Structure of Adjacency List

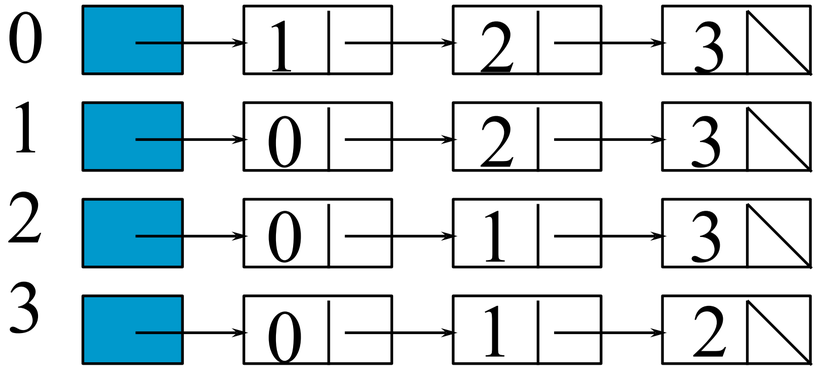
★ Each row in adjacency matrix is represented as an adjacency list.

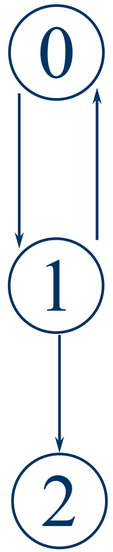
```
#define MAX_VERTICES 50
typedef struct node *node_pointer;
typedef struct node {
    int vertex;
    struct node *link;
};

node_pointer graph[MAX_VERTICES];
int n=0; /* vertices currently in use */
```

Data Structure of Adjacency List

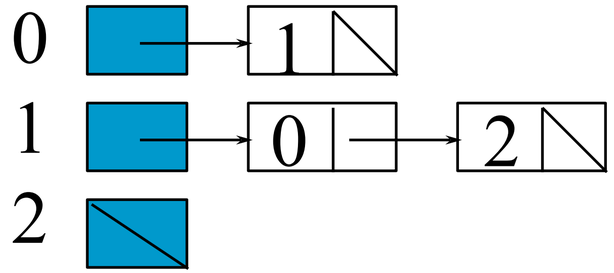


G_1



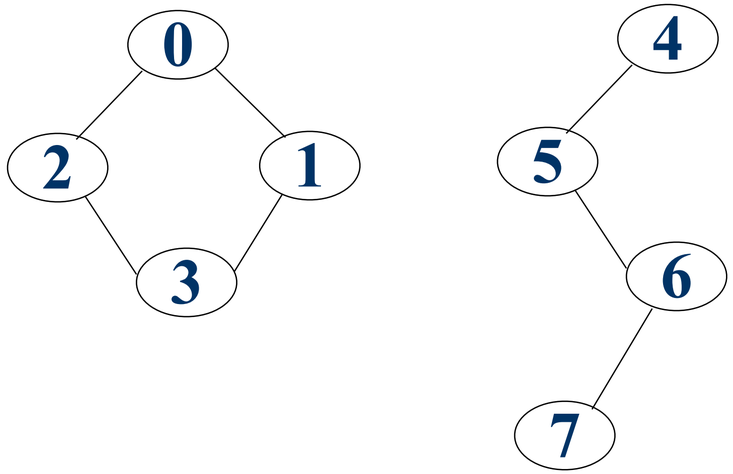


G_3

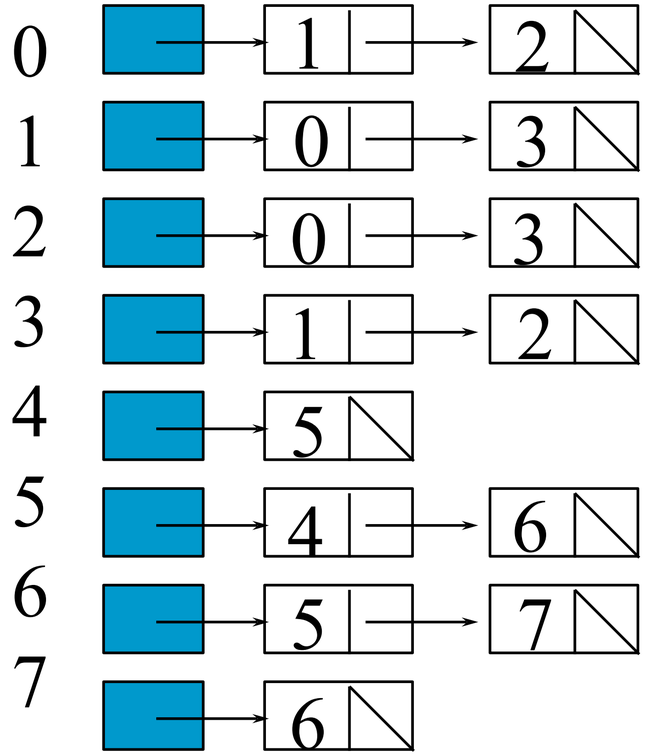


An undirected graph with n vertices and e edges ==> n head nodes and 2e list nodes

Data Structure of Adjacency List



G_4



Adjacency Multi-List

- 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

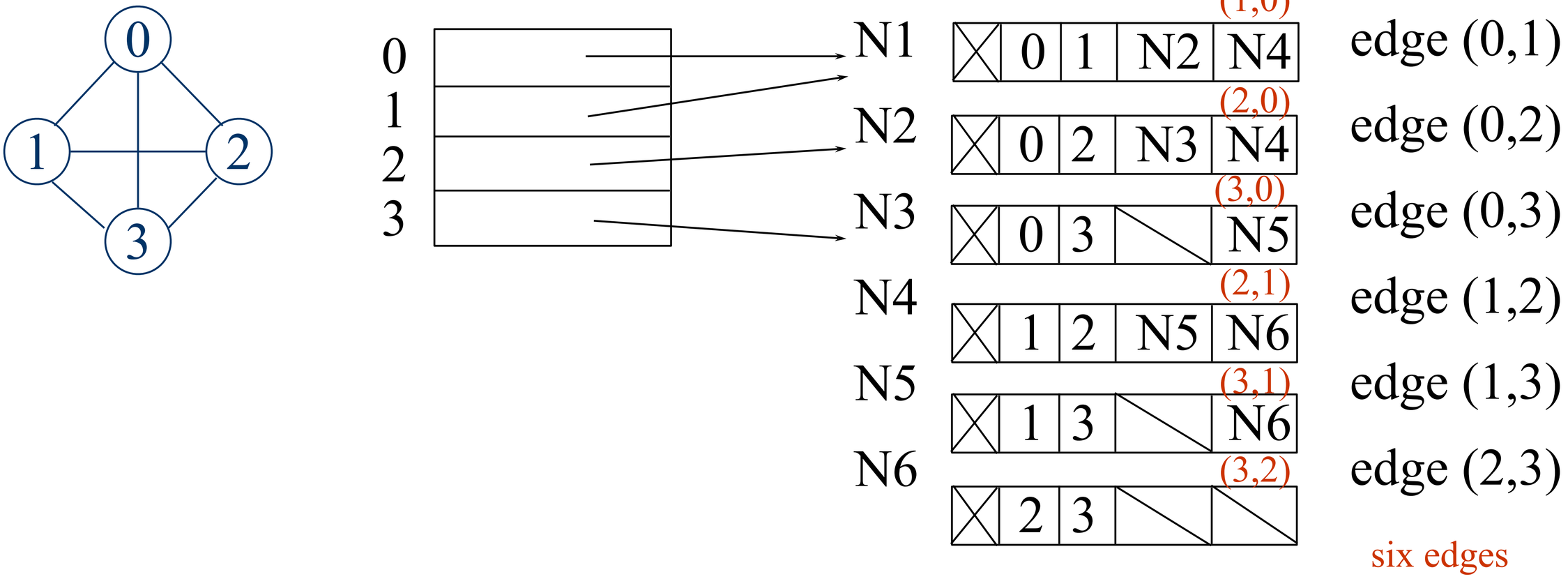
Adjacency Multi-List

- 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 of Adjacency Multi-List

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



Data Structure of Adjacency Multi-List

```
typedef struct edge *edge_pointer;
typedef struct edge {
     short int marked;
     int vertex1, vertex2;
     edge_pointer path1, path2;
};
edge_pointer graph[MAX_VERTICES];
```

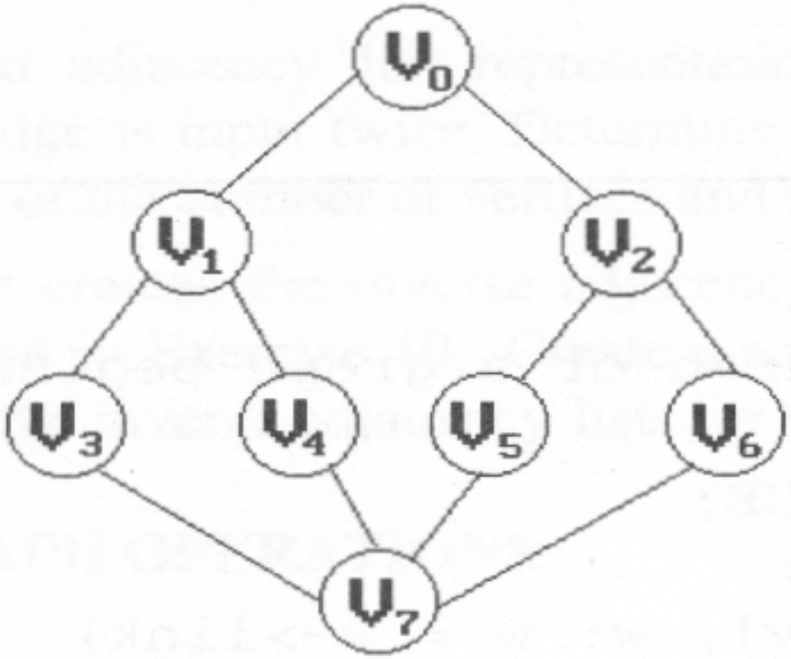
Graph Operations

- Traversal

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

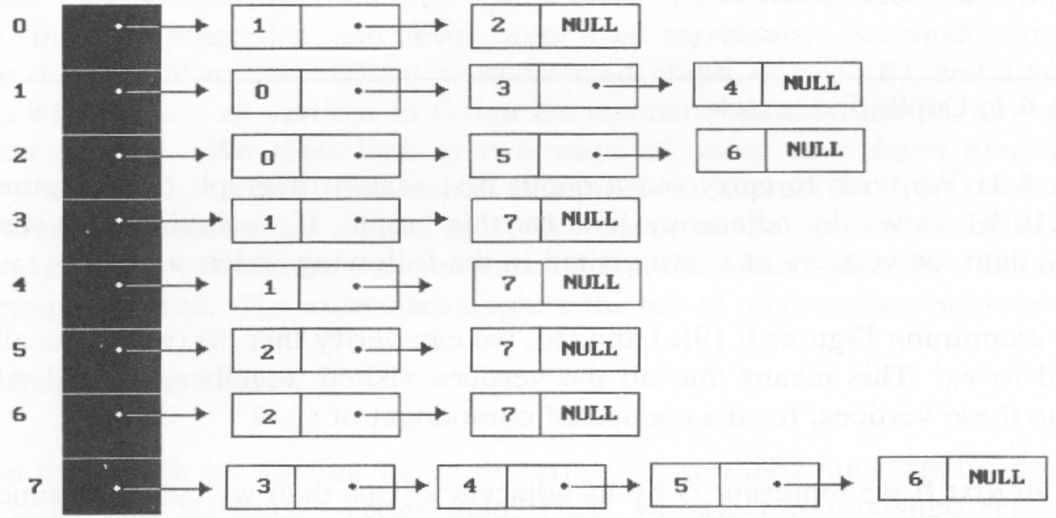
  - Depth First Search (DFS) preorder tree traversal
  - Breadth First Search (BFS) level order tree traversal
- Connected Components
- Spanning Trees

Depth First (DFS) vs Breadth First (BFS) Searches



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

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



Depth First Search (DFS)

```
#define FALSE 0
#define TRUE 1
short int visited[MAX_VERTICES];
```

```
void dfs(int v)
{
  node_pointer w;
  visited[v]= TRUE;
  printf("%5d", v);
  for (w=graph[v]; w; w=w->link)
    if (!visited[w->vertex])
      dfs(w->vertex);
}
```

Data structure
adjacency list: O(e)
adjacency matrix: $O(n^2)$

Breadth First Search (BFS)

```
typedef struct queue *queue_pointer;
typedef struct queue {
     int vertex;
     queue_pointer link;
};
void addq(queue_pointer *,
          queue_pointer *, int);
int deleteq(queue_pointer *);
```

Breadth First Search (BFS)

```
void bfs(int v)
{
  node_pointer w;
  queue_pointer front, rear;
  front = rear = NULL;
  printf("%5d", v);
  visited[v] = TRUE;
  addq(&front, &rear, v);
  while (front) {
    v= deleteq(&front);
    for (w=graph[v]; w; w=w->link)
      if (!visited[w->vertex]) {
        printf("%5d", w->vertex);
        addq(&front, &rear, w->vertex);
        visited[w->vertex] = TRUE;
      }
  }
}
```

adjacency list: O(e)
adjacency matrix: $O(n^2)$

Connected Components

```
void connected(void)
{
    for (i=0; i<n; i++) {
        if (!visited[i]) {
            dfs(i);
            printf("\n");
        }
    }
}
```

adjacency list: O(n+e)
adjacency matrix: $O(n^2)$

Spanning Trees

★ When graph G is connected, a depth first or breadth first search starting at any vertex will visit all vertices in G.

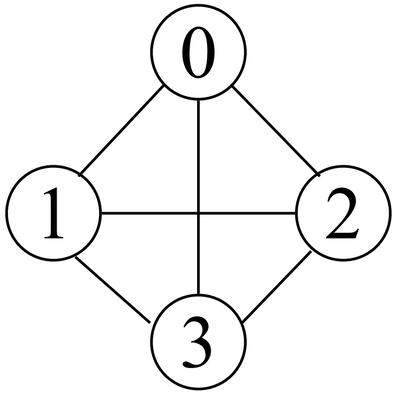
★ A spanning tree is any tree that consists solely of edges in G and that includes all the vertices.

★ E(G): T (tree edges) + N (nontree edges),

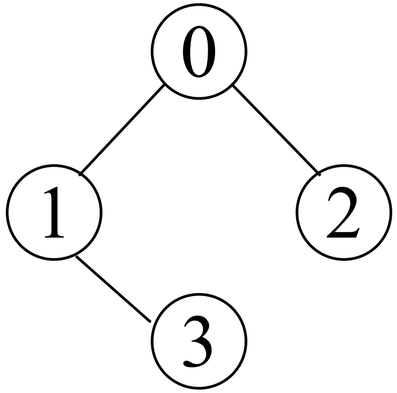
where T: set of edges used during search,

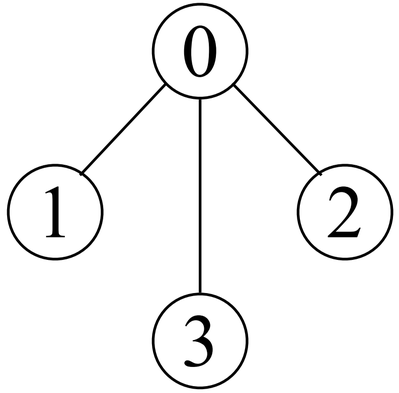
N: set of remaining edges.

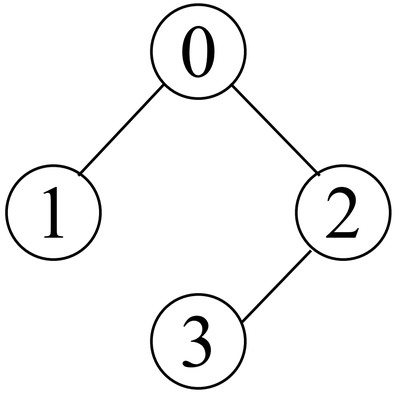
Examples of Spanning Trees



G_1







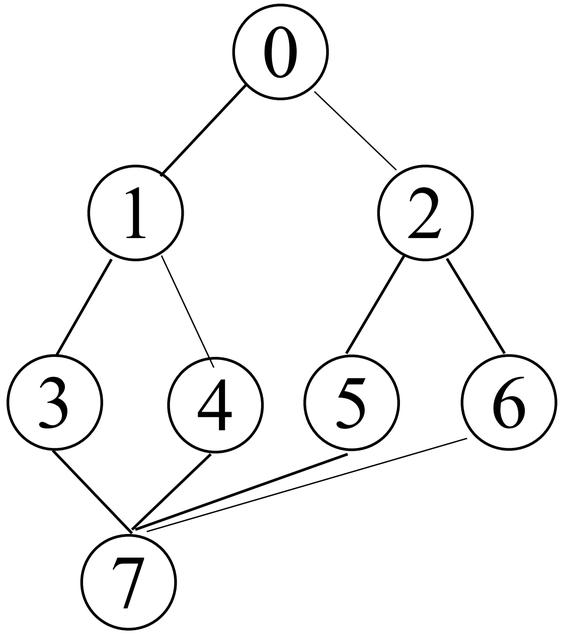
Possible spanning trees

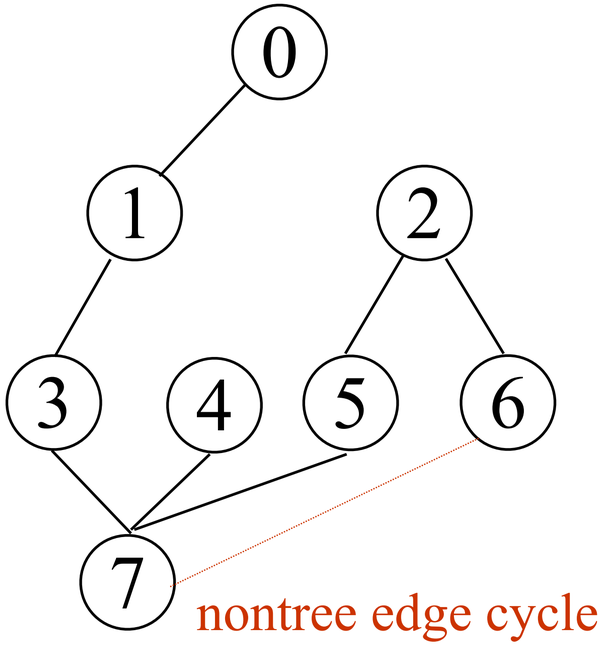
Spanning Tree

- 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.

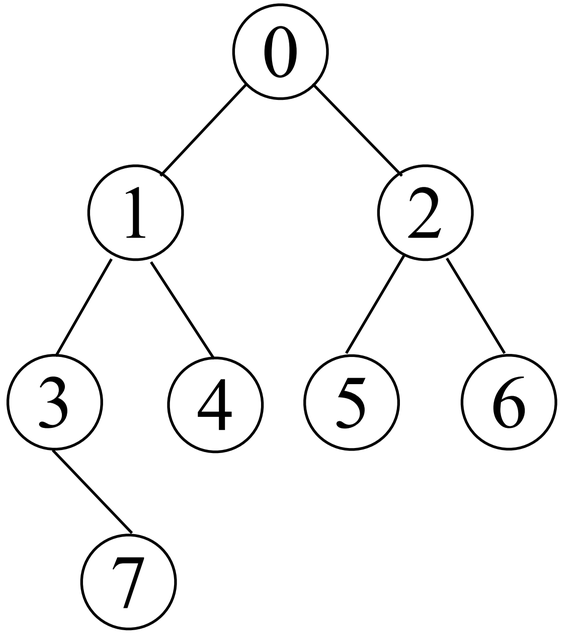
- While adding a non-tree edge into any spanning tree, this will create a cycle.

DFS vs BFS Spanning Tree





DFS Spanning



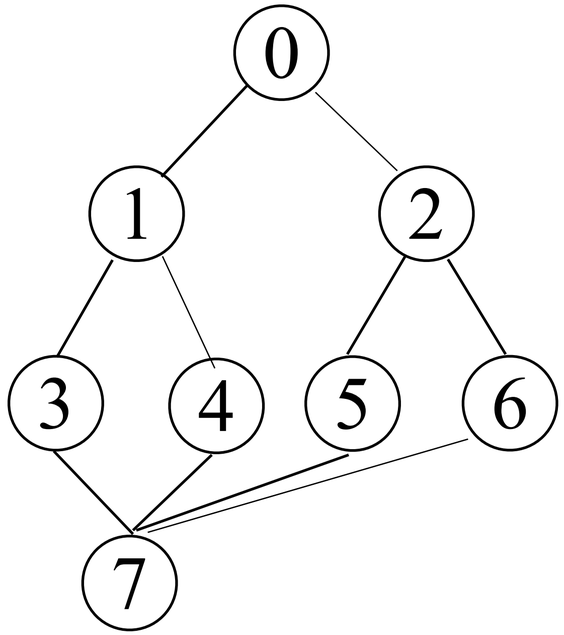
BFS Spanning

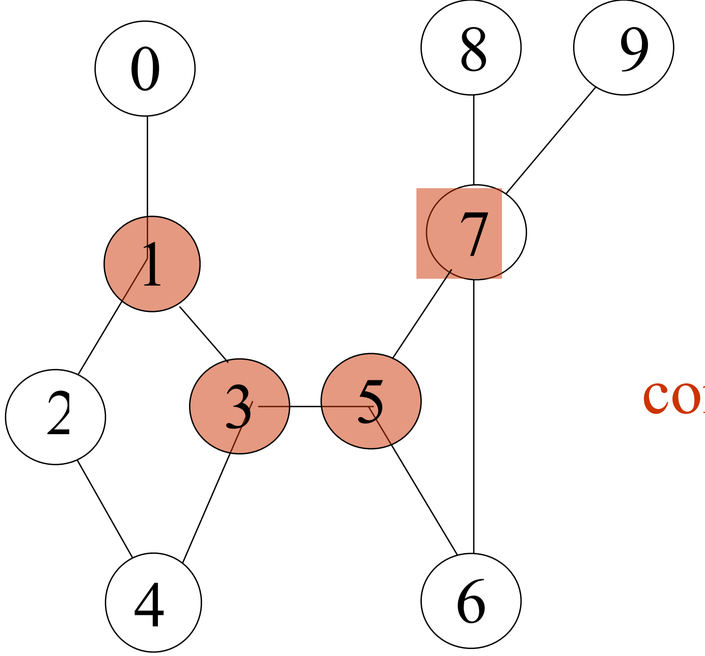
A spanning tree is a minimal subgraph, G', of G such that V(G')=V(G) and G' is connected.

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.

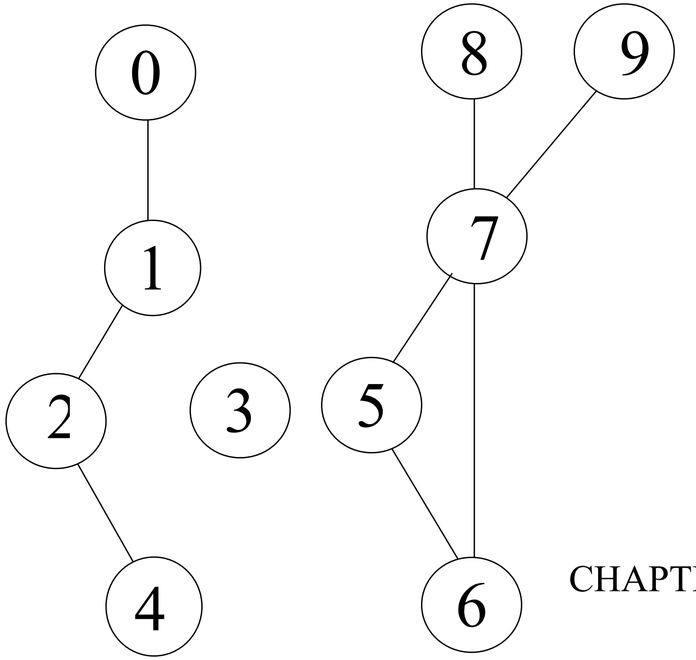
biconnected graph



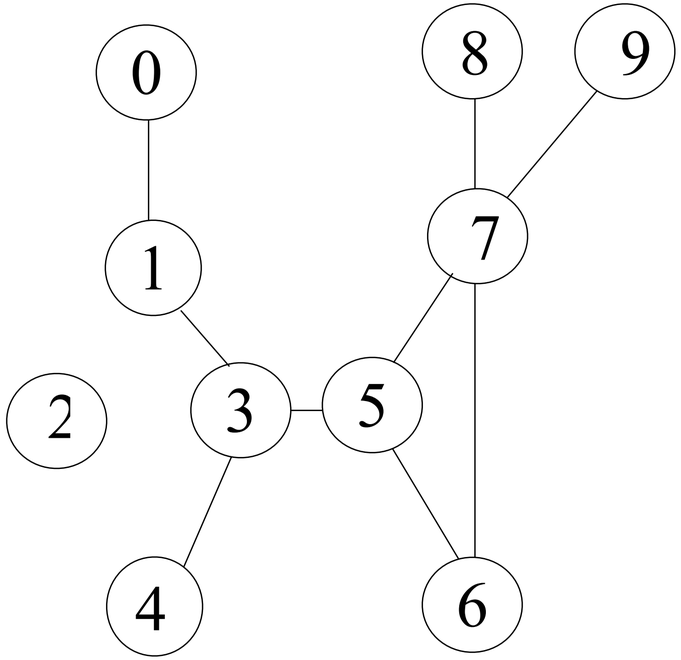


connected graph

two connected components



one connected graph



Spanning Trees

- Traversal

Given G=(V,E) and vertex v, find all w ∈ 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

Question and Answer