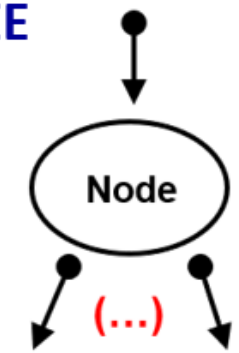
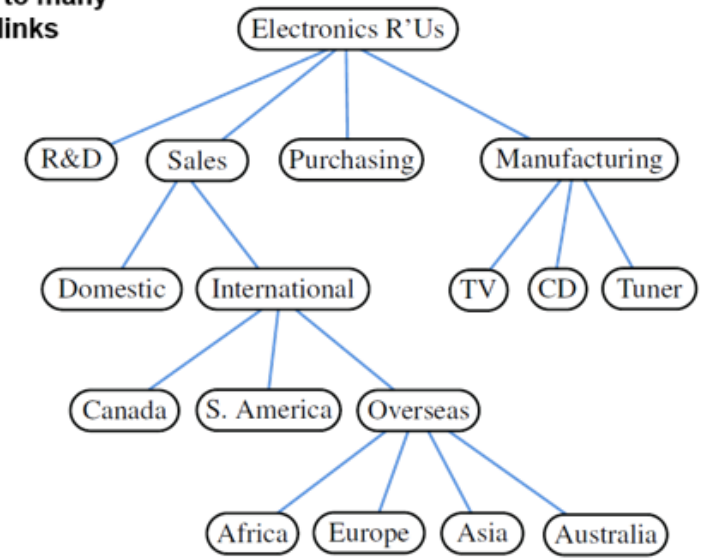
UNIVERSITY OF CALGARY

Introduction to Graphs

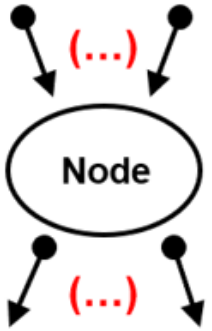
TREE



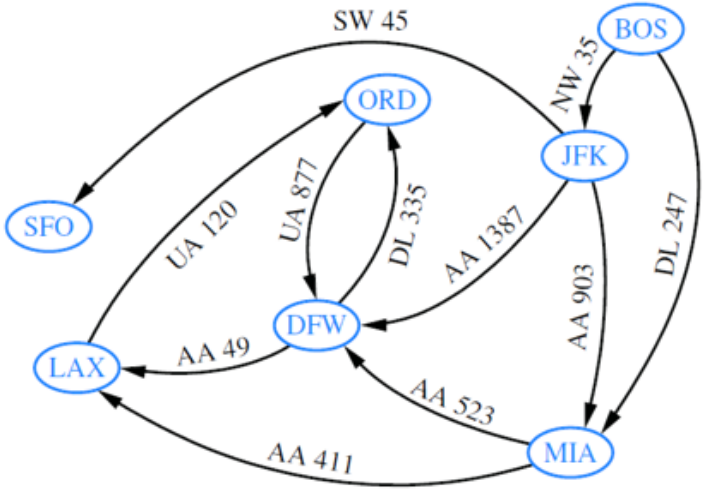
"one to many" links



GRAPH



"many to many" links



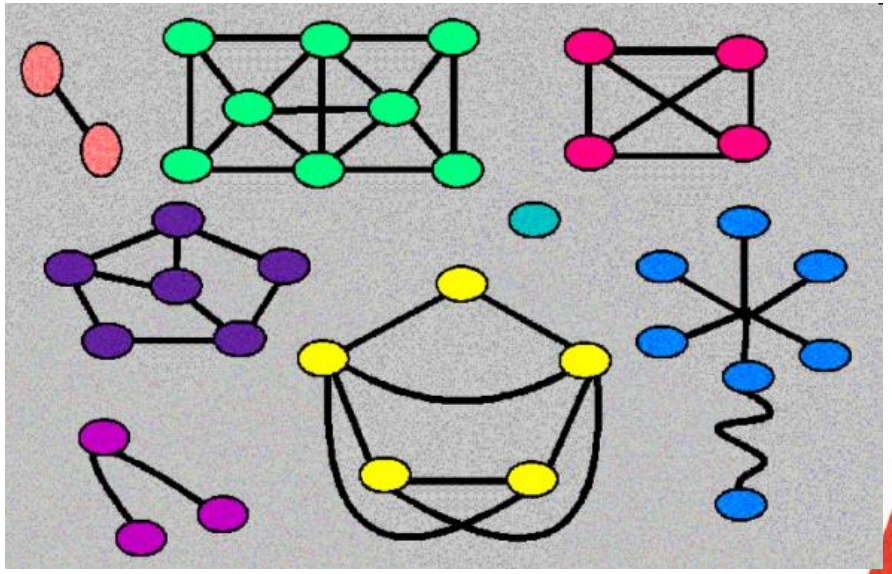
What is a Graph?

- A **graph** *G* is an ordered pair of sets [*V,E*] where

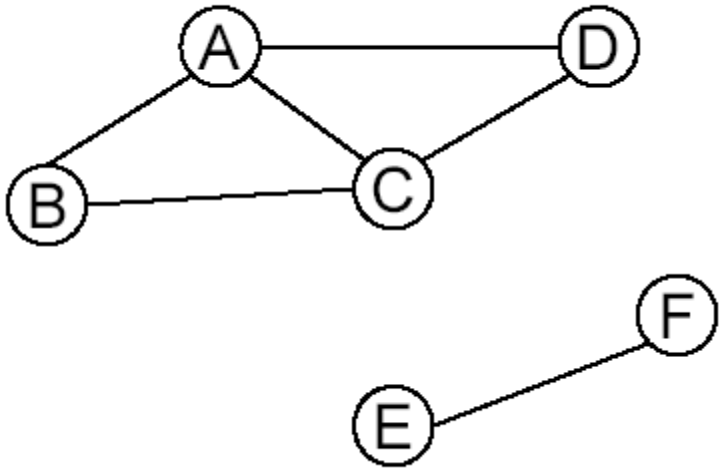
$V = \{v_1, v_2, ..., v_i, ..., v_n\}$ is a set of ***vertices*** (i.e., nodes)

$E = \{e_1, e_2, ..., e_i, ..., e_m\}$ is a set of ***edges*** (i.e., links)

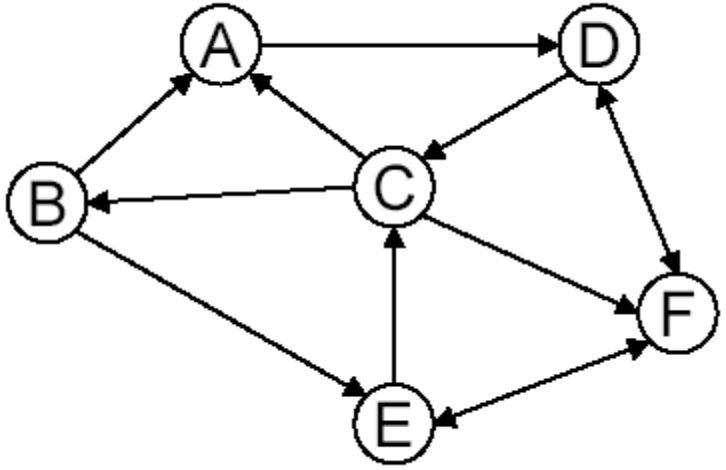
- Data structure heavily used in computing problems.



What is a Graph?



Undirected Graph

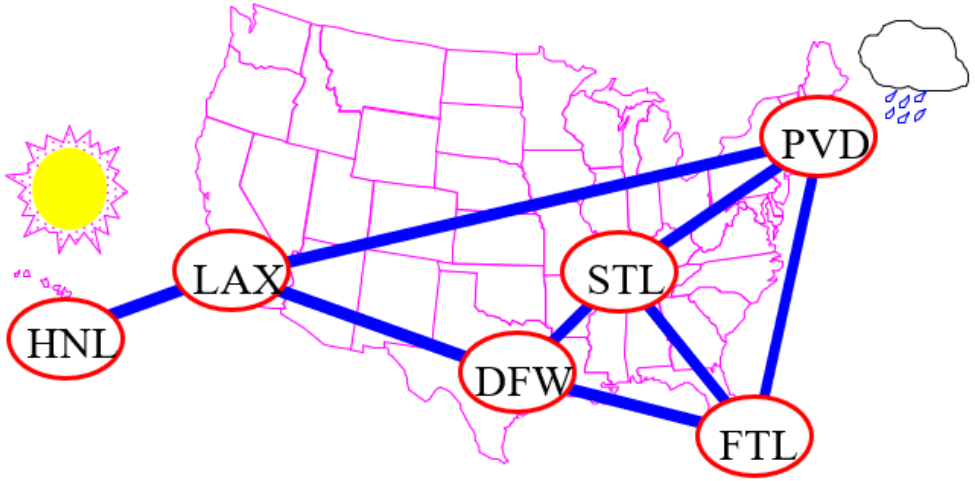


Directed Graph (digraph)

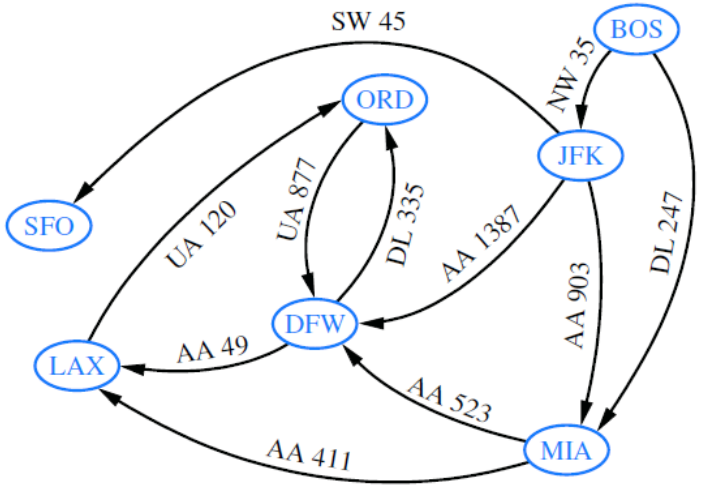
Applications

- Networks:

  (roads, flights, communications)

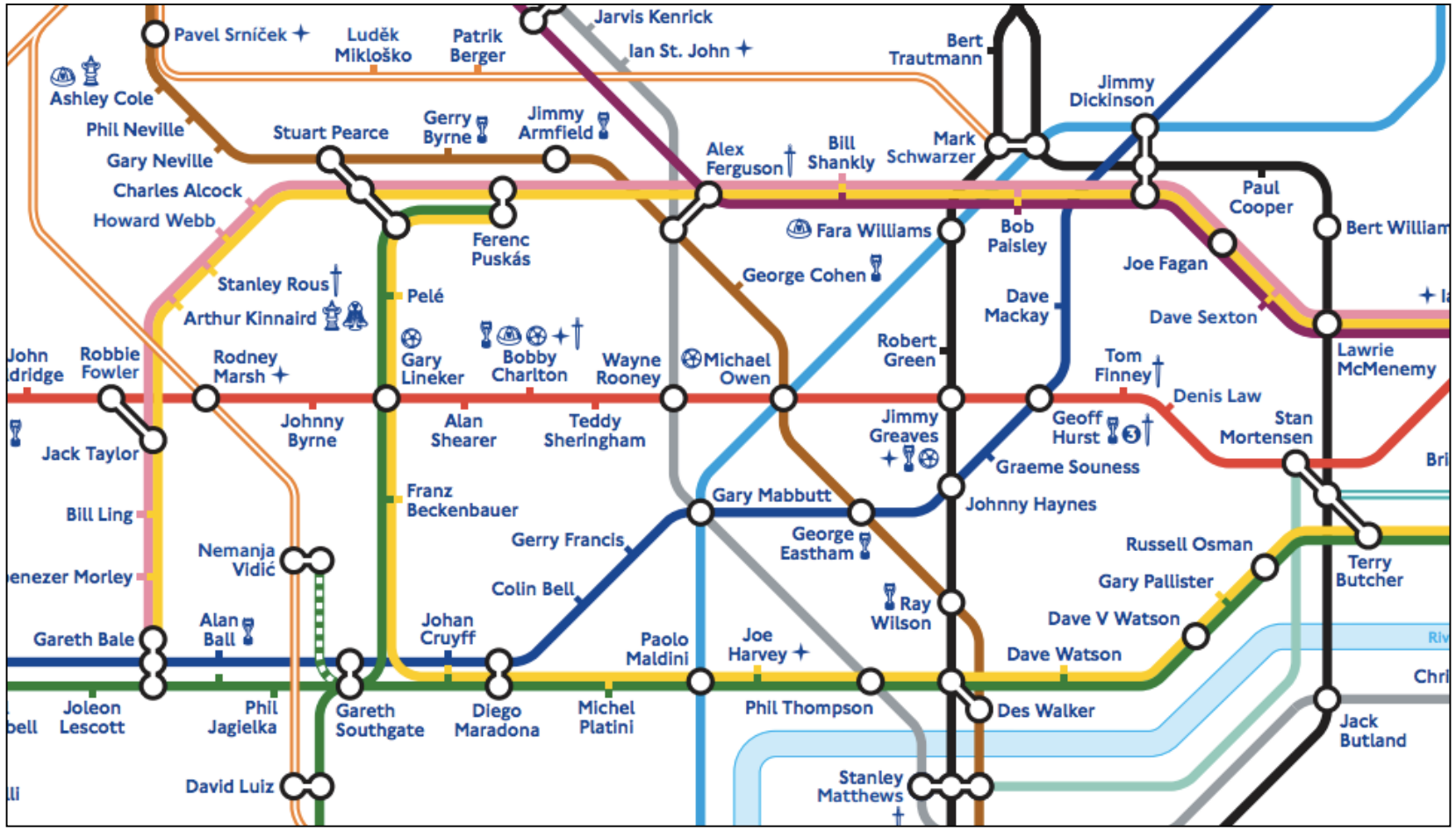


Undirected Graph



Flight Connections

Directed Graph (digraph)

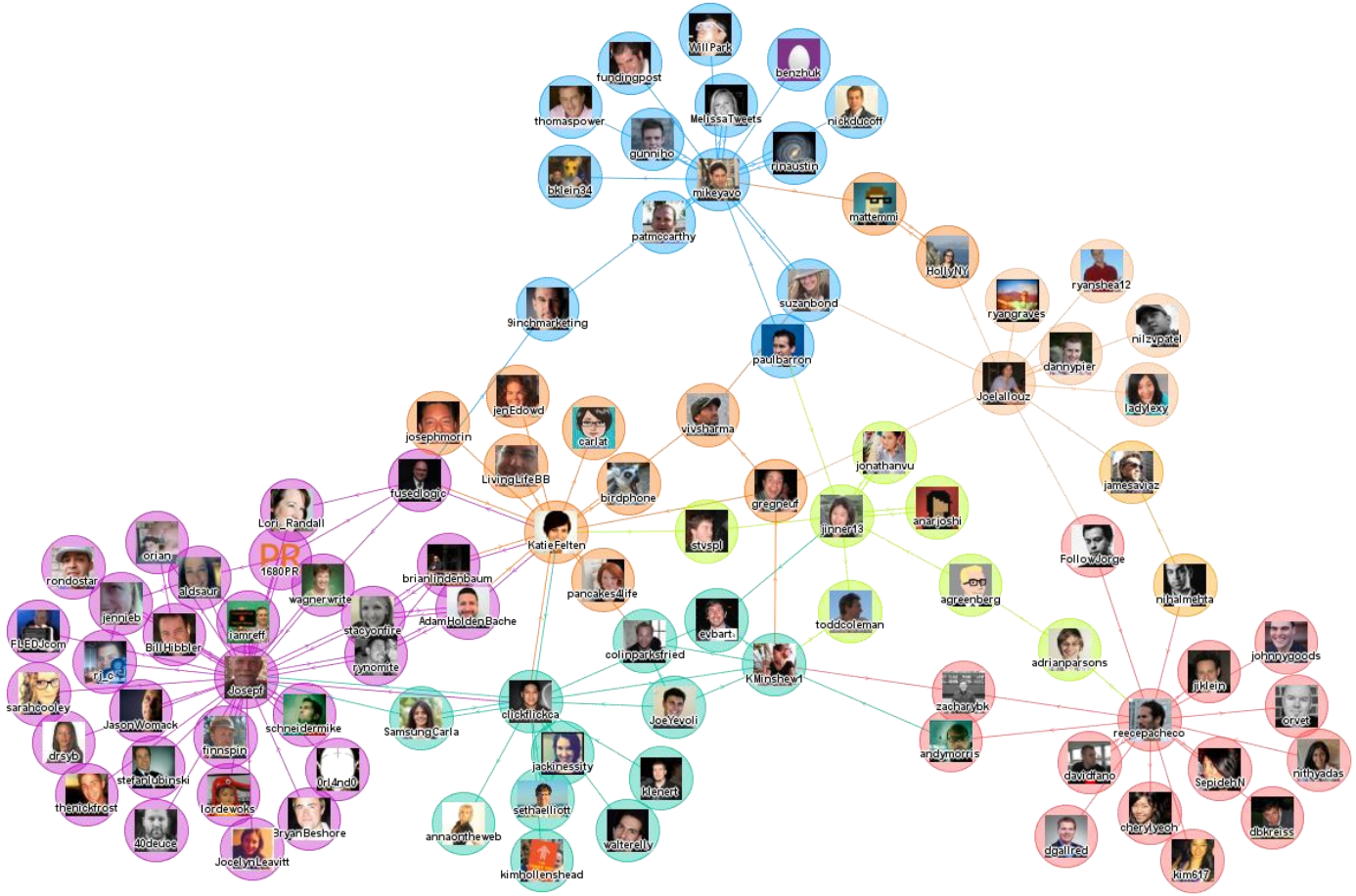


London Tube Map

Applications

UNIVERSITY OF CALGARY

Social Network



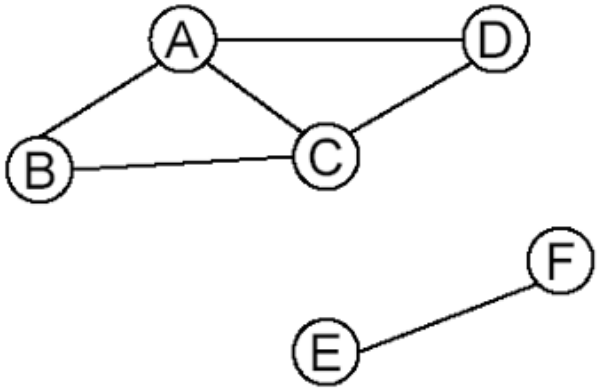
What is a Graph?

- A graph G = (V,E) is composed of:

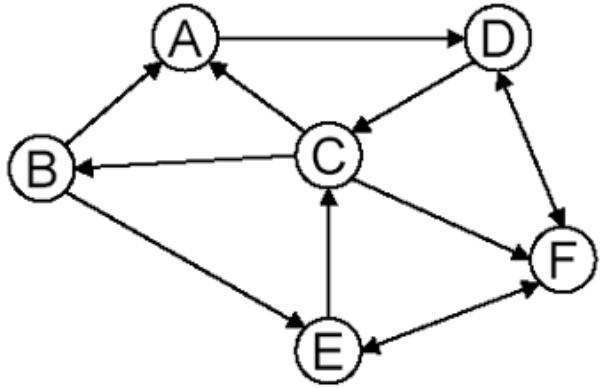
  V: set of vertices (i.e., *nodes*)

  E: set of edges (i.e., *links*) connecting the vertices in V

- An edge e = (u,v) is a pair of vertices
- Example:



Undirected Graph



Directed Graph (digraph)

Vertices

- ***V*** represents the number of vertices in the graph

- ***V(G)*** is a set of vertices or nodes which can represent an object that needs to be “connected”

Edges

- a distinct pair of vertices
- indicates a valid/existing connection between two vertices

- ***E*** → the number of edges in the graph
- ***E(G)*** → a set consisting of a pair of vertices

- *an edge which connects the same vertex is called a loop*

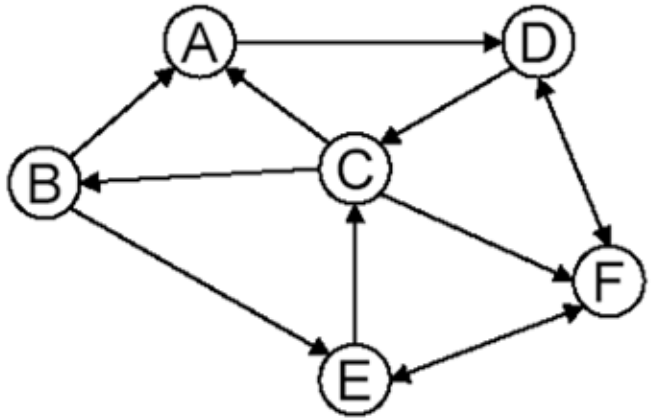
Types of Graphs

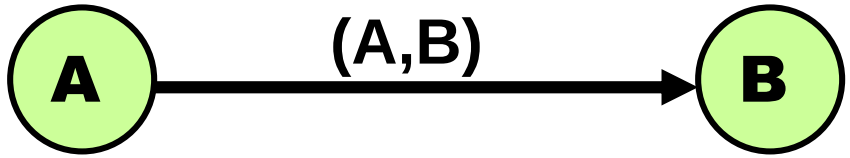
UNIVERSITY OF CALGARY

- Directed Graphs
- Undirected Graphs
- Weighed Graphs
- Trees

Directed Graph or Digraph

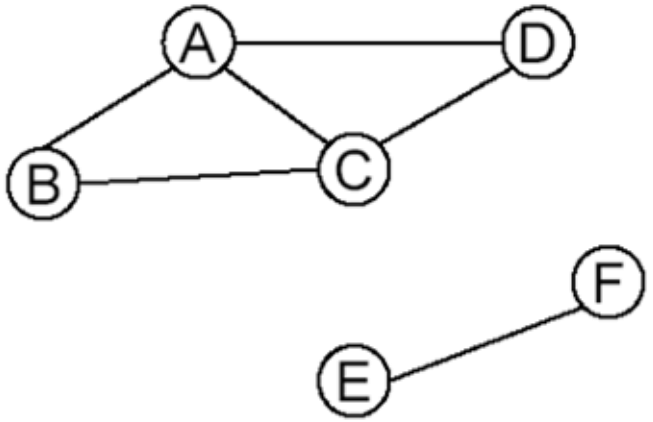
- There is an order of the pair of vertices in the set of edges
- The connecting lines are usually represented with an arrow





Undirected Graph

- The order of the vertices in the pair of vertices in the set of edges does not matter

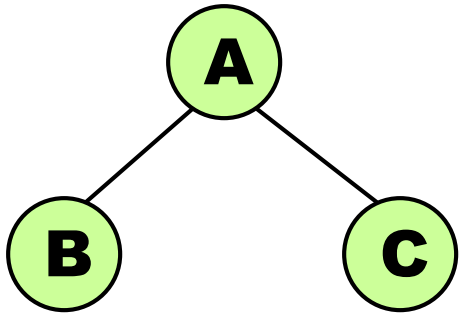


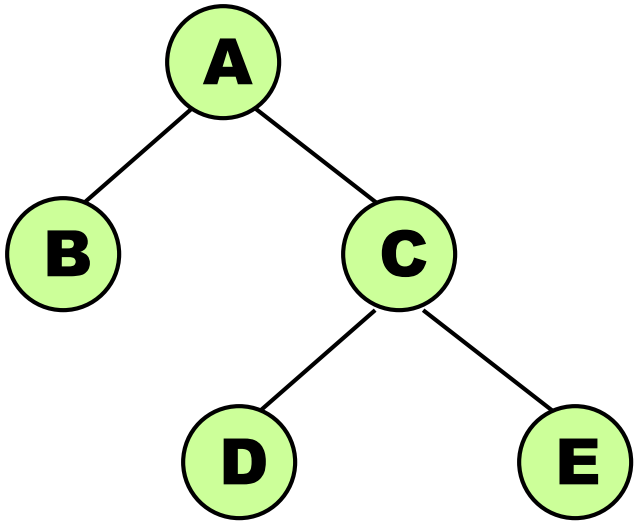
Weighted Graph

- Each edge has an associated weight which could indicate cost, distance, time, etc. between two adjacent vertices

Tree

- A connected graph with no cycles

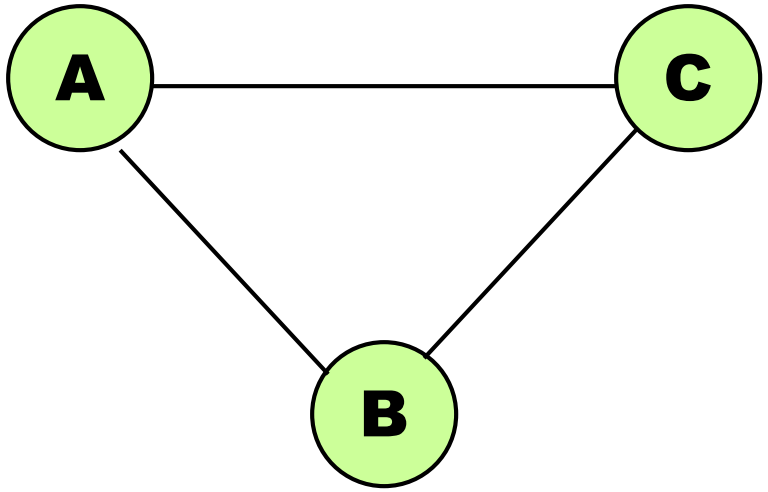




Examples of a Tree

Adjacent Vertices

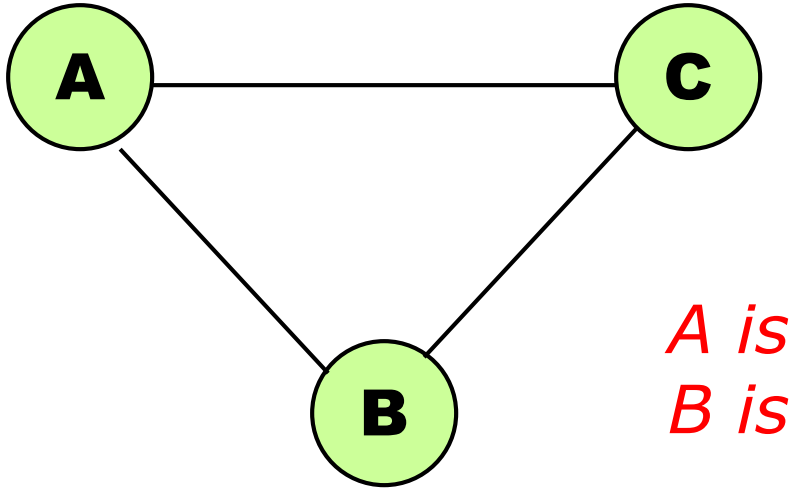
- if two vertices are joined by an edge they are said to be ***adjacent***



Adjacent Vertices:
A & C
A & B
B & C

Incident Vertices

- if an edge connects two vertices, the vertices are said to be ***incident*** to the edge

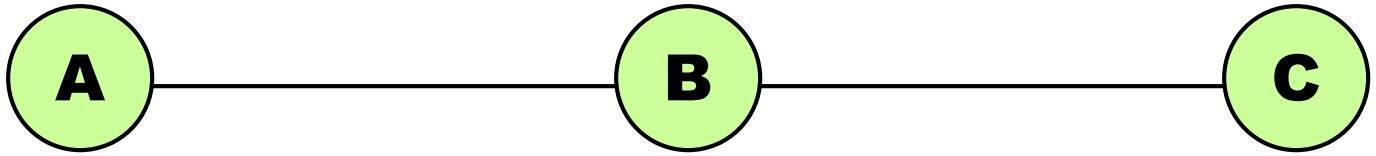


A is incident to edge AC
B is incident to edge BA

Degree of a Vertex

- The degree of a vertex ***v*** is the number of vertices adjacent to it (or the number of edges incident to it)

- Represented as ***deg(v) or degree(v)***



deg(A) = 1 *deg(B) = 2* *deg(C) = 1*

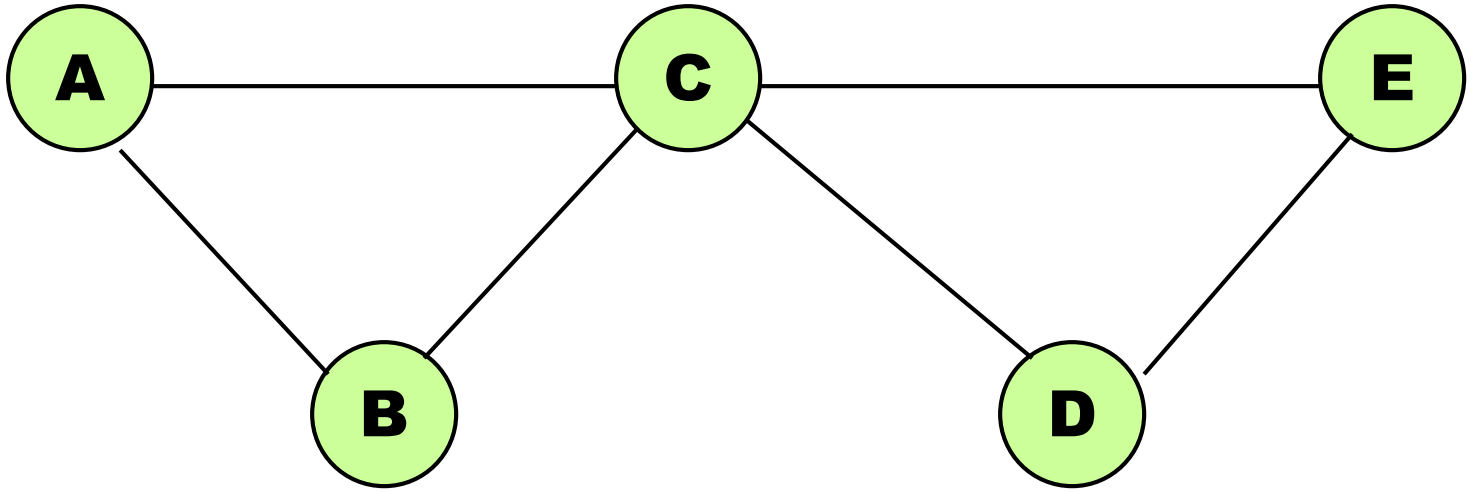
Paths

- Given a graph ***G*** which includes the vertices ***x*** and ***y***.
- A path from ***x*** to ***y*** length ***k*** is a sequence of ***k*** distinct edges

$e_1, e_2, .., e_n$ such that:

$e1 = (x, x_1), e2 = (x_1, x_2), \ldots e3 = (e_{k-1}, y)$

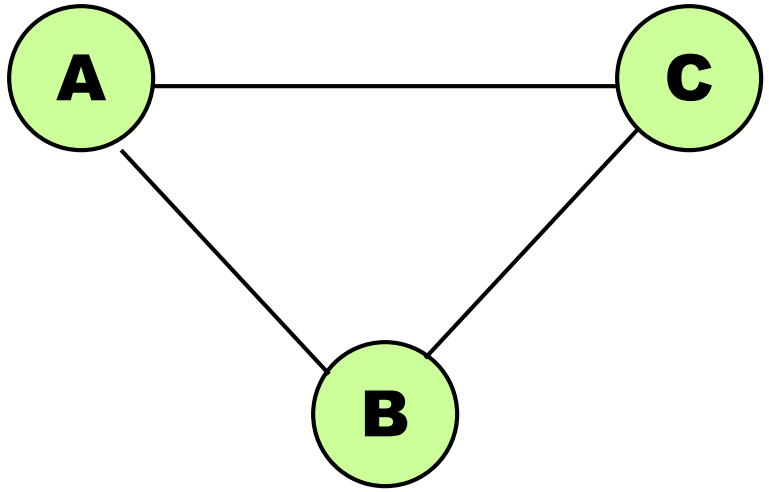
Path



Path from A to D: A-B-C-D
Path from B to E: B-A-C-D-E

Cycle

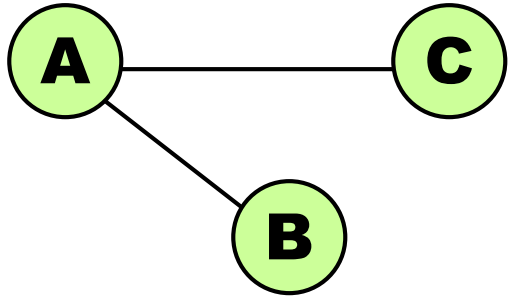
- If ***x*** and ***y*** are the same vertex, then we say that the path from ***x*** to ***y*** is a cycle



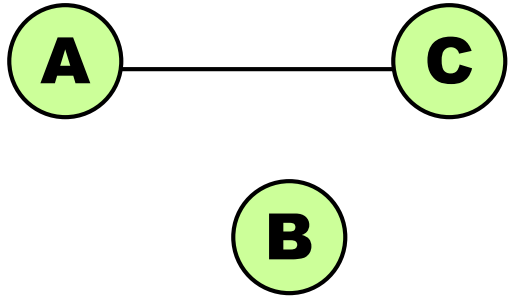
Cycles:
A-B-C-A
C-B-A-C

Connectivity

- A graph ***G*** is connected if for every pair of vertices ***x*** and ***y***, there exists a path from ***x*** to ***y***



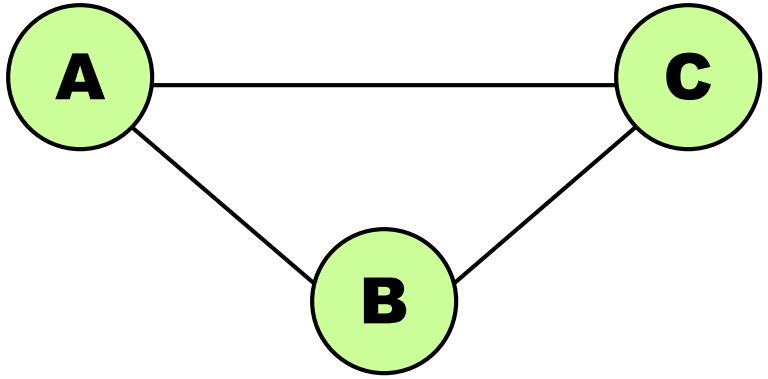
Connected Graph



Disconnected Graph

Distance

- The distance ***d(x,y)*** from ***x*** to ***y*** is the smallest number of edges in a path from ***x*** to ***y***

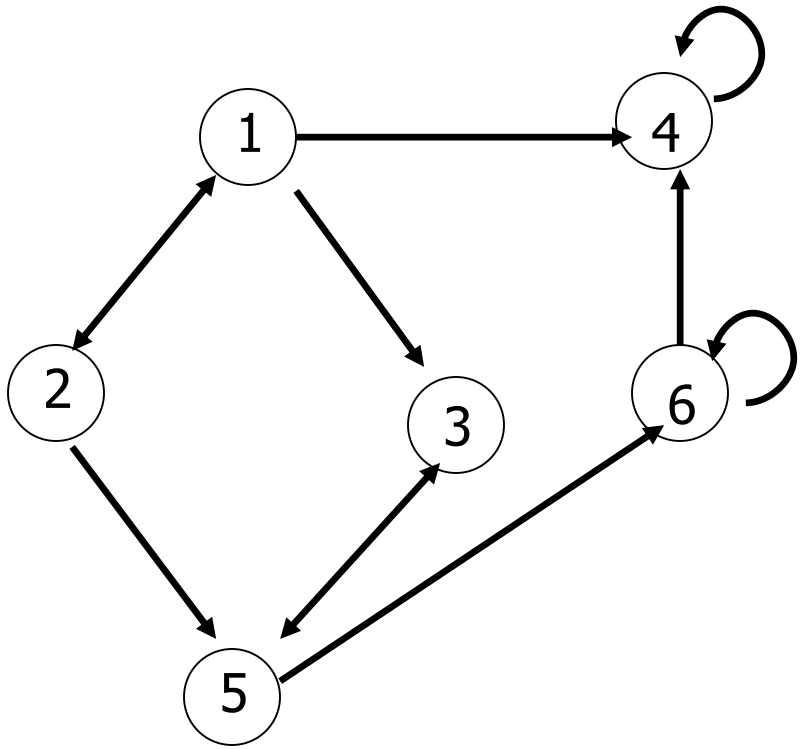


d(A,C) = 1
d(B,C) = 1

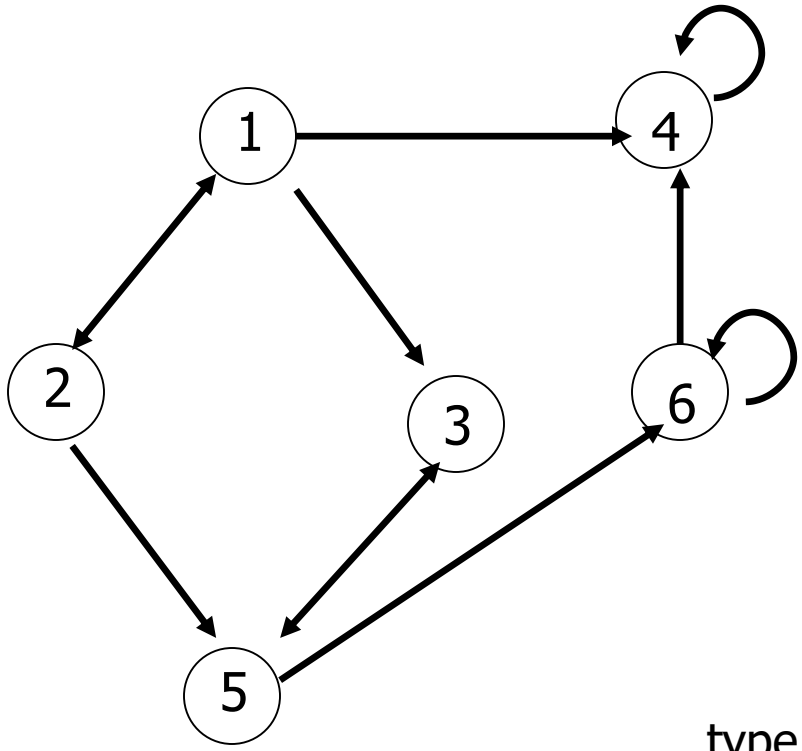
Data Structures for Graphs

- Adjacency List
- Adjacency Matrix

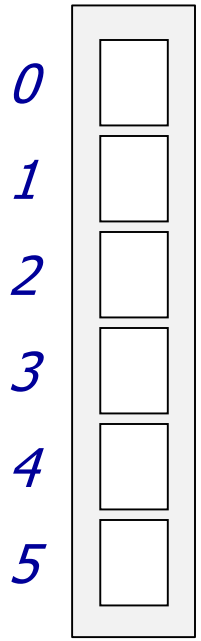
Adjacency List Structure



Adjacency List Structure



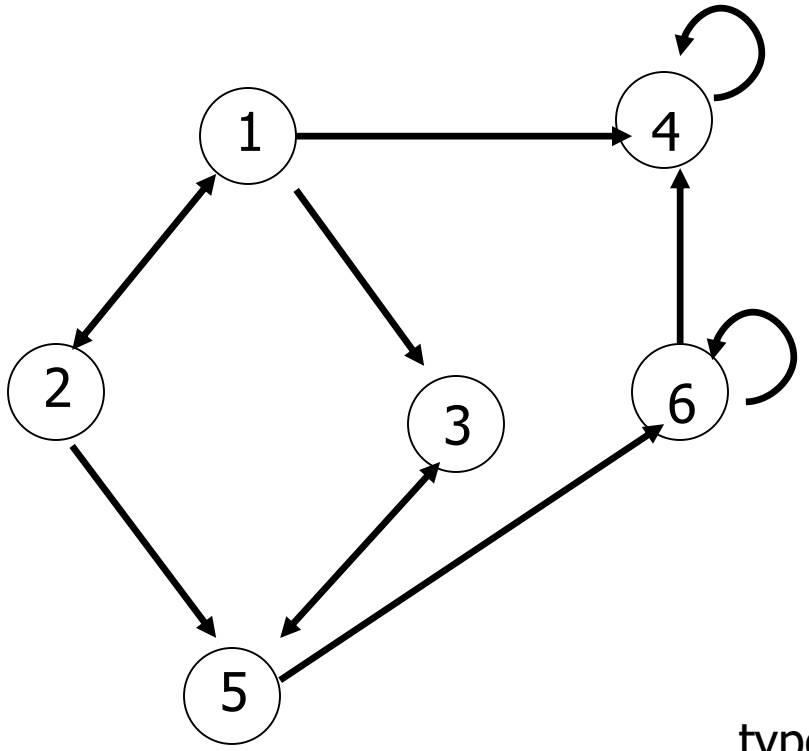
Static 1-D array
of type VERTEX



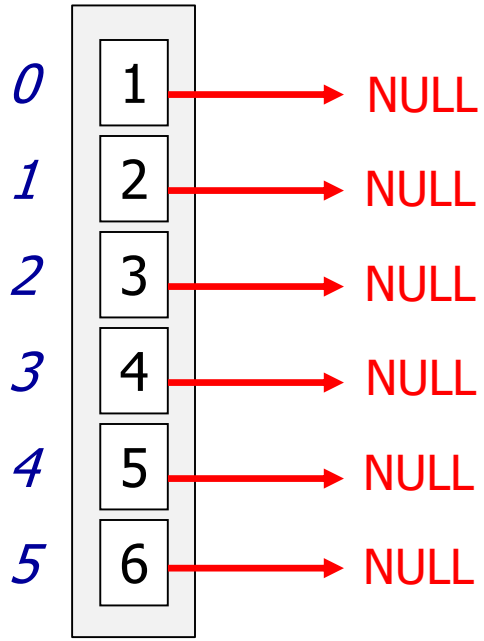
type VERTEX (i.e. node)

vertex_number: int edge (i.e. link): singly linked list of type NODE

Adjacency List Structure



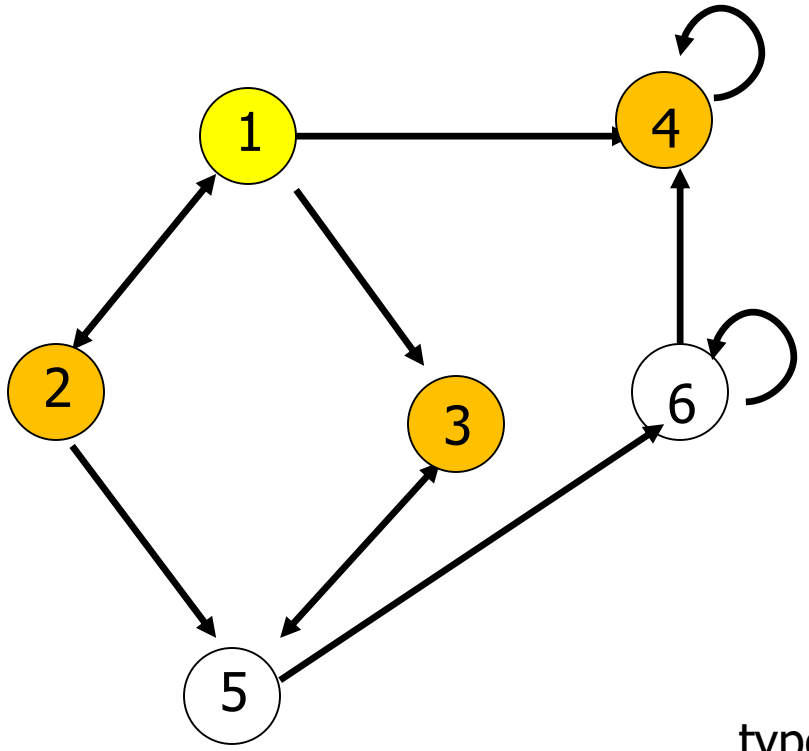
Static 1-D array
of type VERTEX



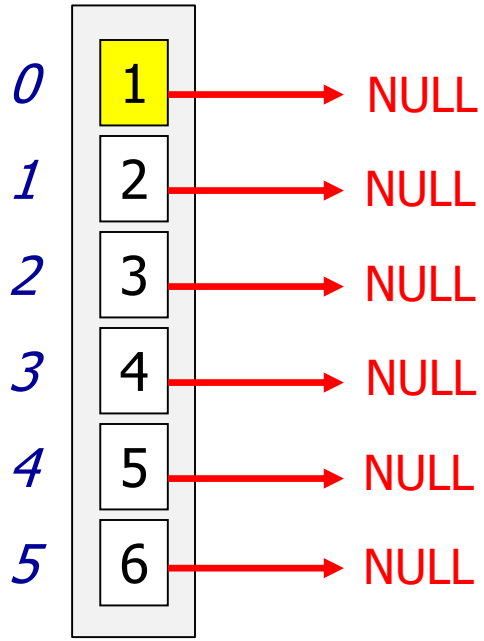
type VERTEX

vertex_number: int edge: singly linked list of type VERTEX

Adjacency List Structure



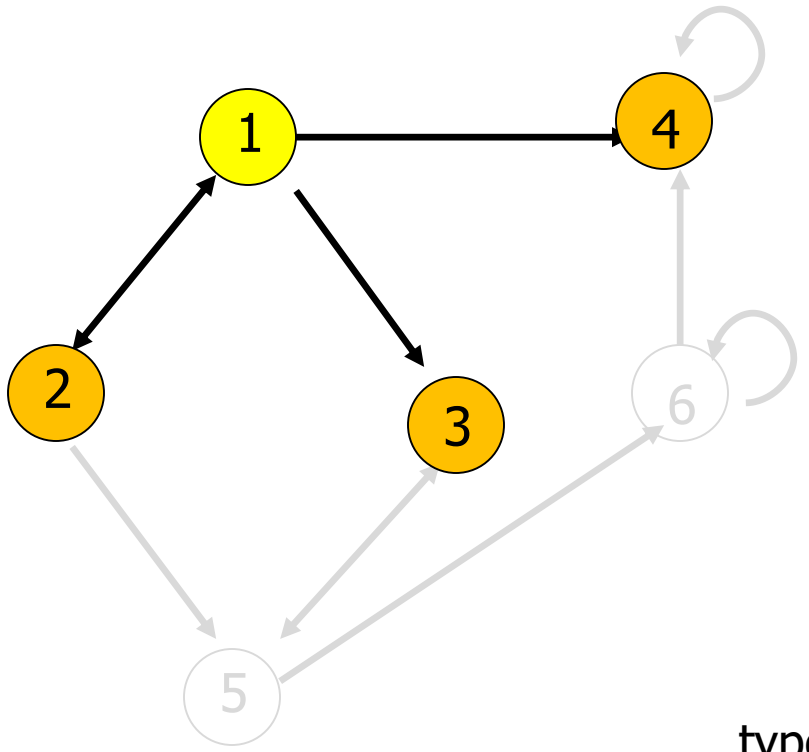
Static 1-D array
of type VERTEX



type VERTEX

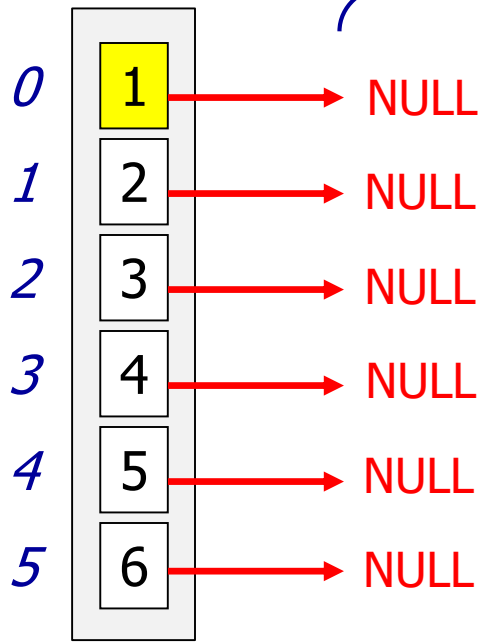
vertex_number: int edge: singly linked list of type VERTEX

Adjacency List Structure



Static 1-D array
of type VERTEX

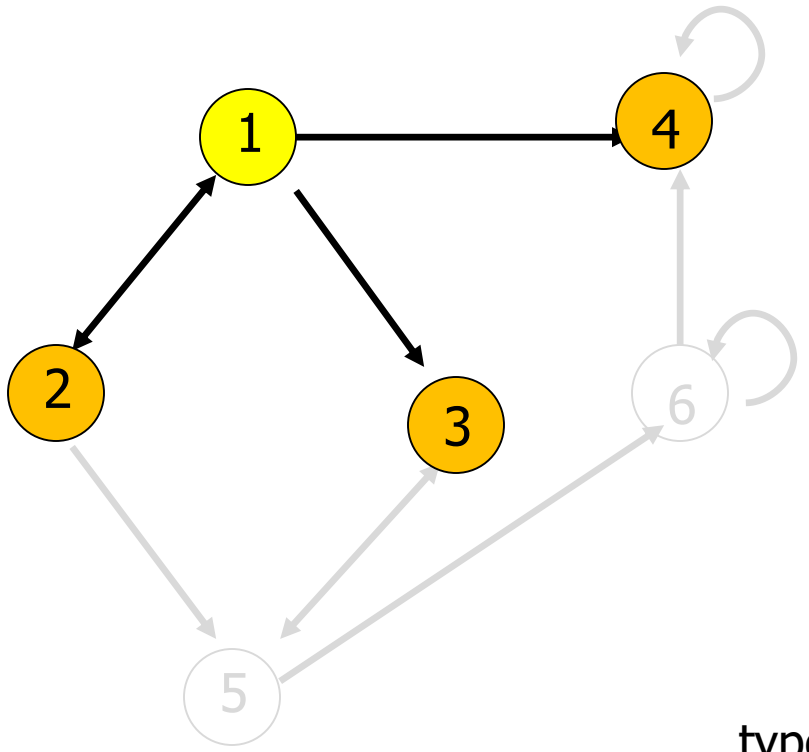
List of vertices
adjacent to vertex #1

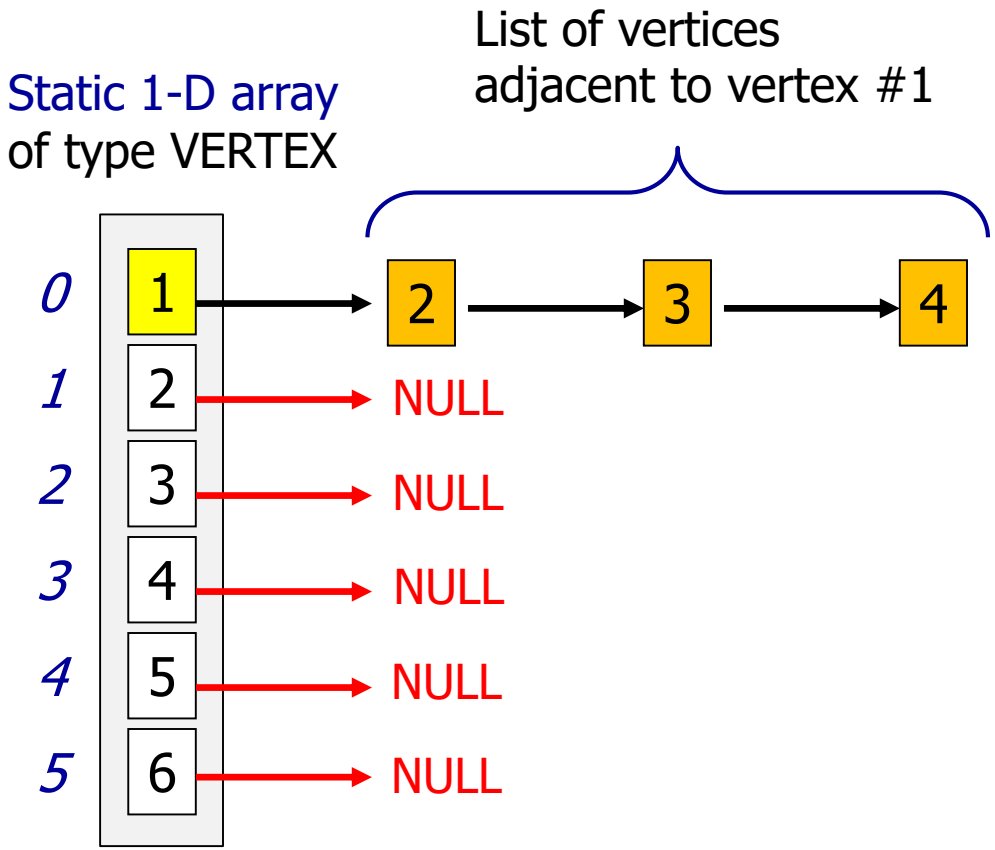


type VERTEX

vertex_number: int edge: singly linked list of type VERTEX

Adjacency List Structure

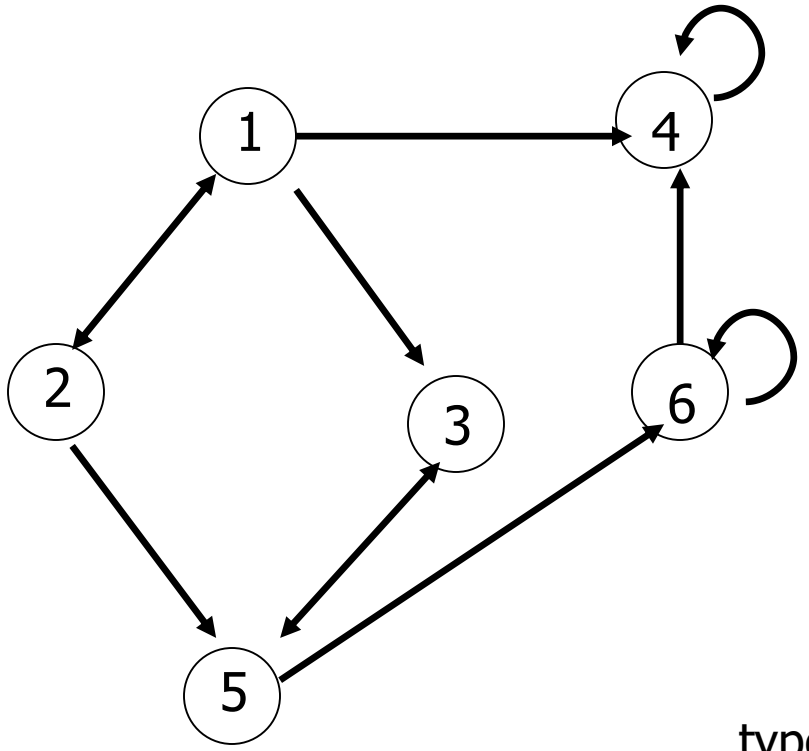




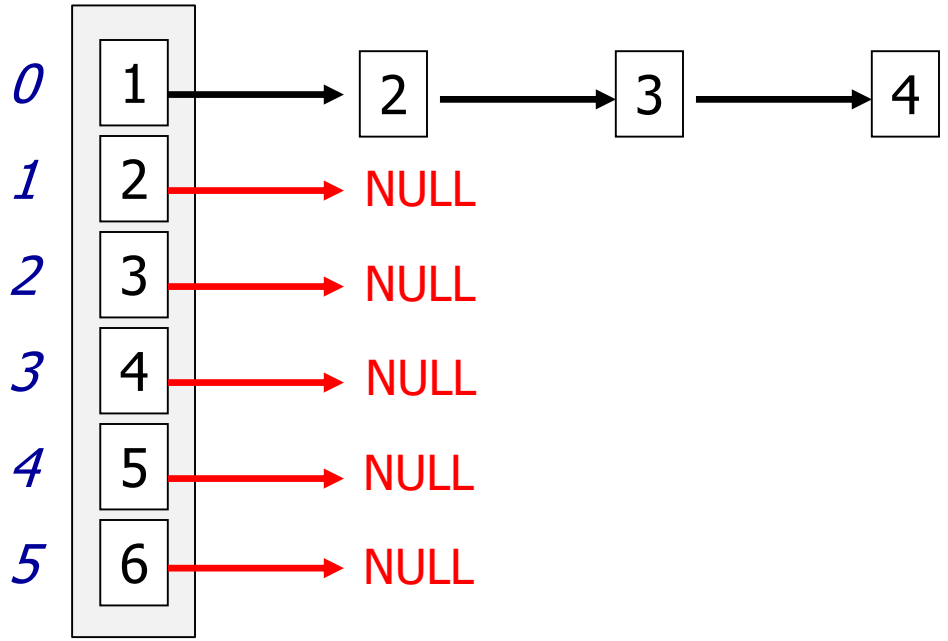
type VERTEX

vertex_number: int edge: singly linked list of type VERTEX

Adjacency List Structure



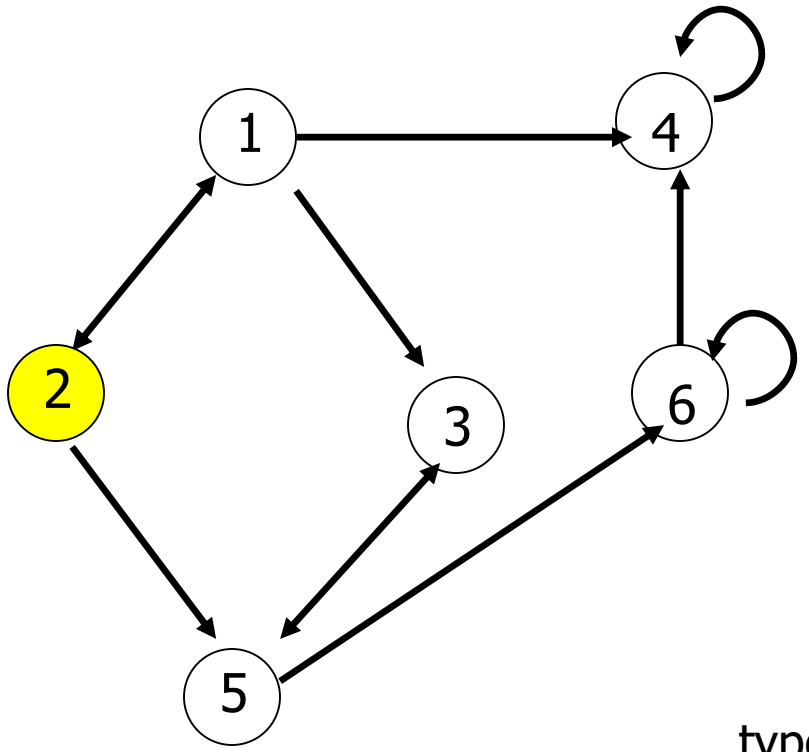
Static 1-D array
of type VERTEX



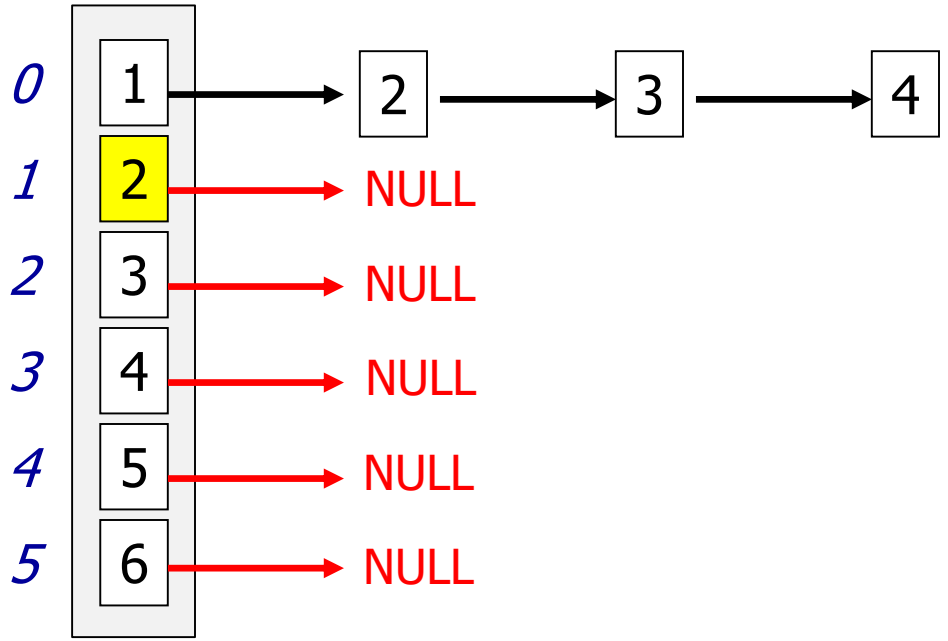
type VERTEX

vertex_number: int edge: singly linked list of type VERTEX

Adjacency List Structure



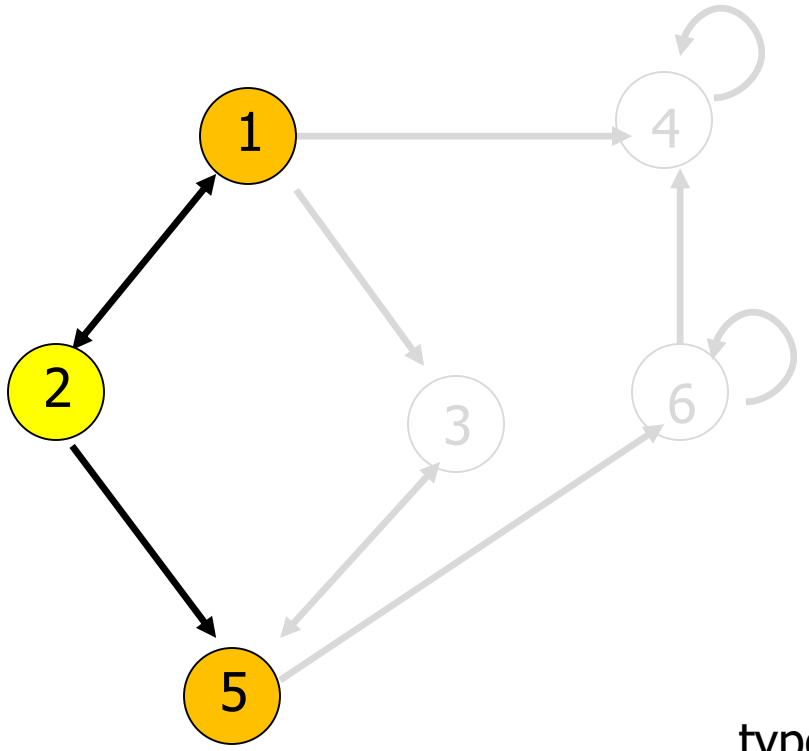
Static 1-D array
of type VERTEX



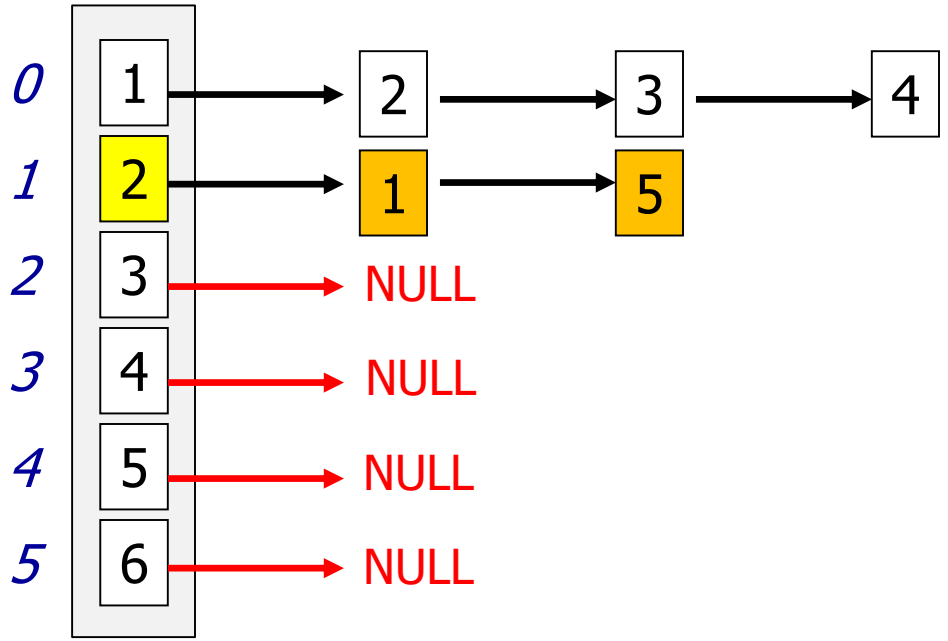
type VERTEX

vertex_number: int edge: singly linked list of type VERTEX

Adjacency List Structure



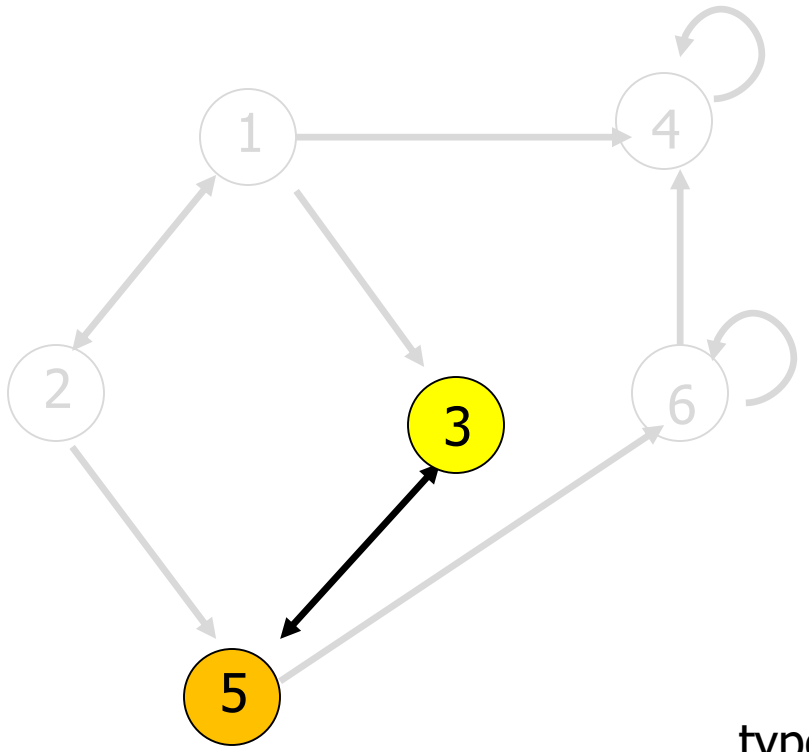
Static 1-D array
of type VERTEX



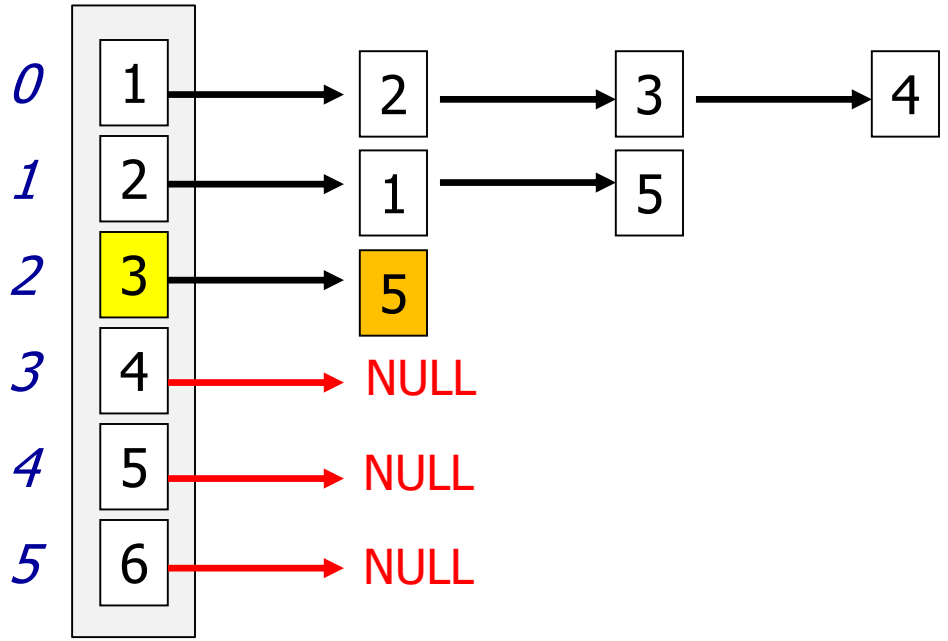
type VERTEX

vertex_number: int edge: singly linked list of type VERTEX

Adjacency List Structure



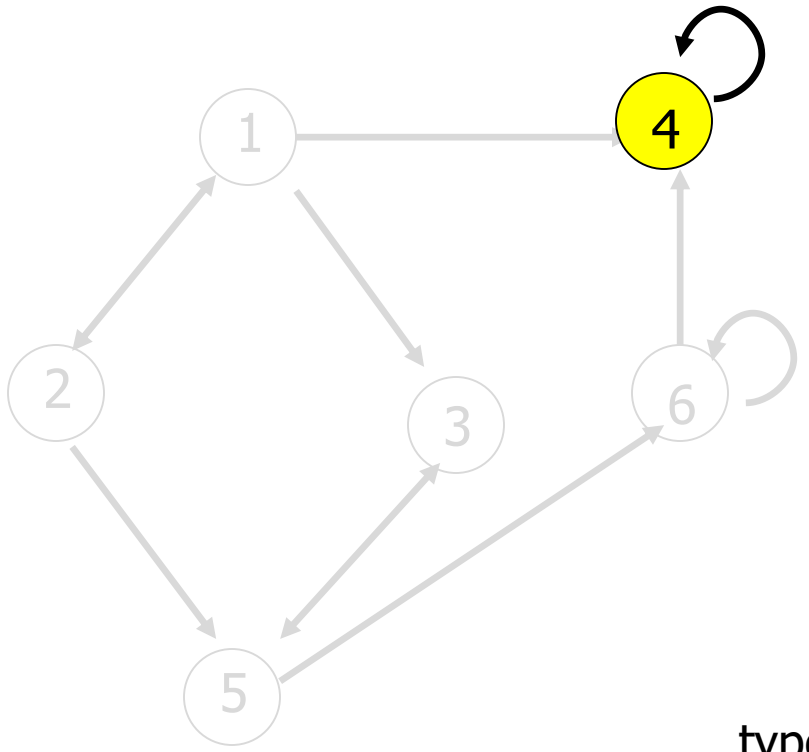
Static 1-D array
of type VERTEX



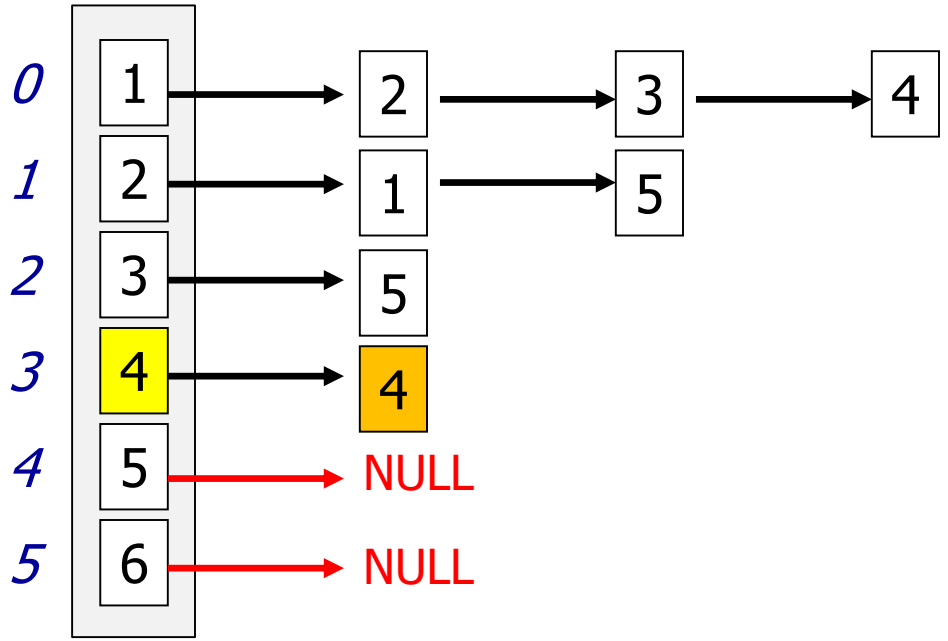
type VERTEX

vertex_number: int
edge: singly linked list of type VERTEX

Adjacency List Structure



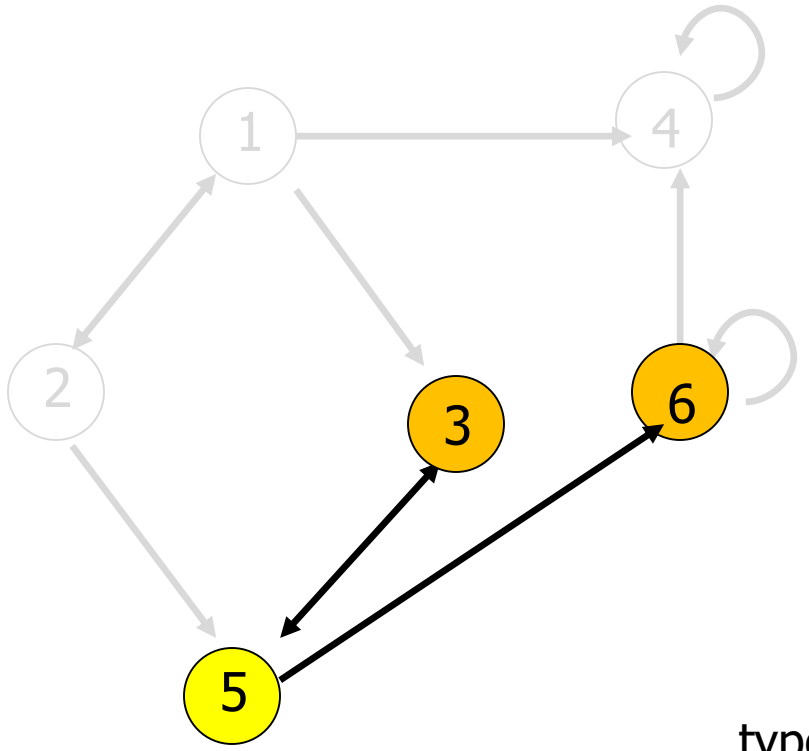
Static 1-D array
of type VERTEX



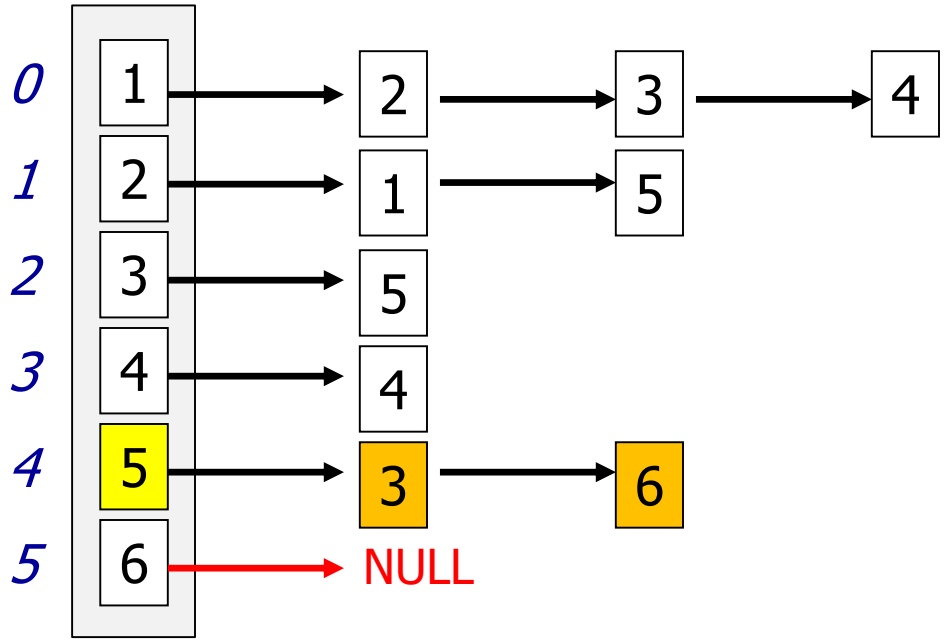
type VERTEX

vertex_number: int edge: singly linked list of type VERTEX

Adjacency List Structure



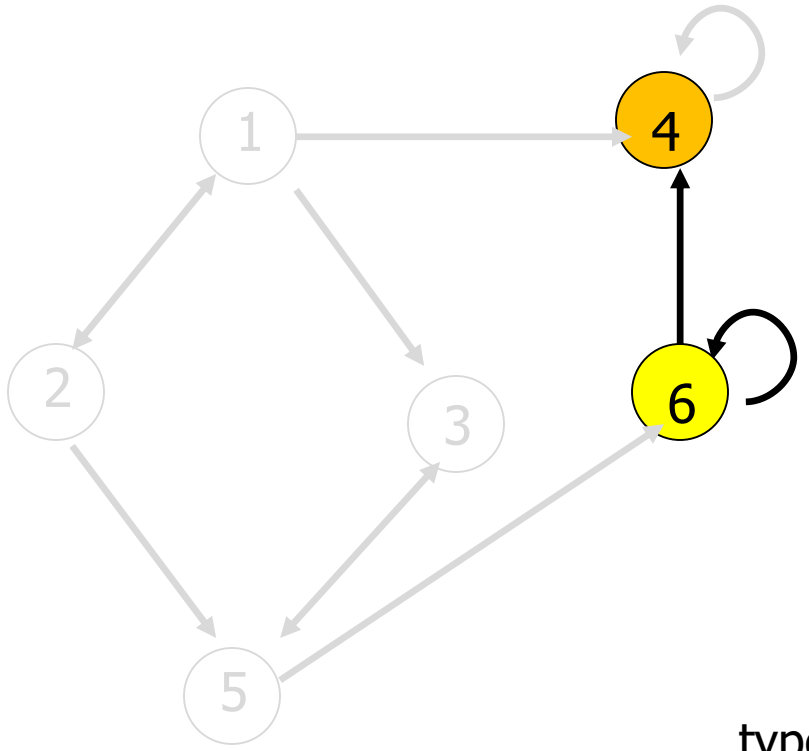
Static 1-D array
of type VERTEX



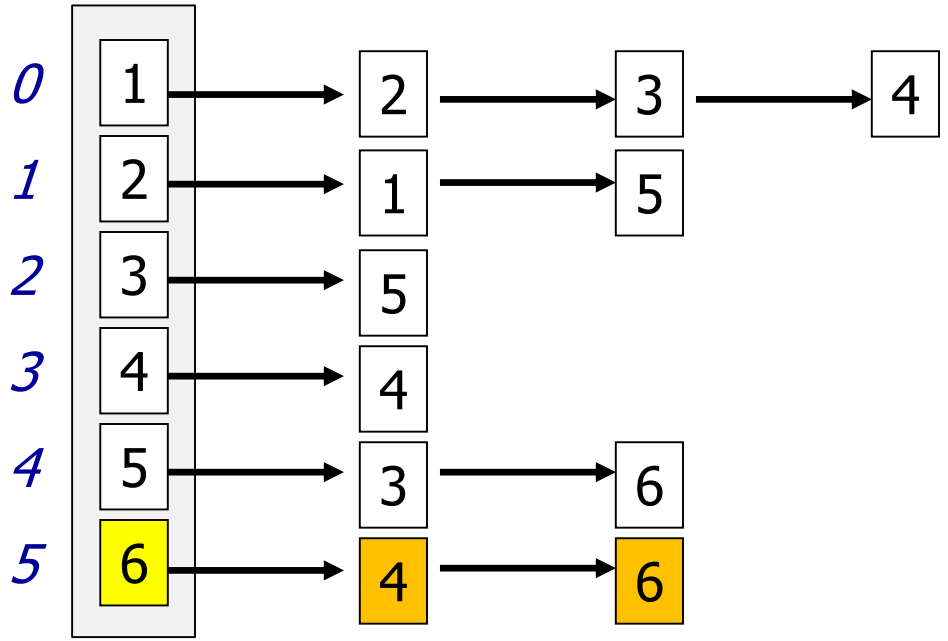
type VERTEX

vertex_number: int edge: singly linked list of type VERTEX

Adjacency List Structure



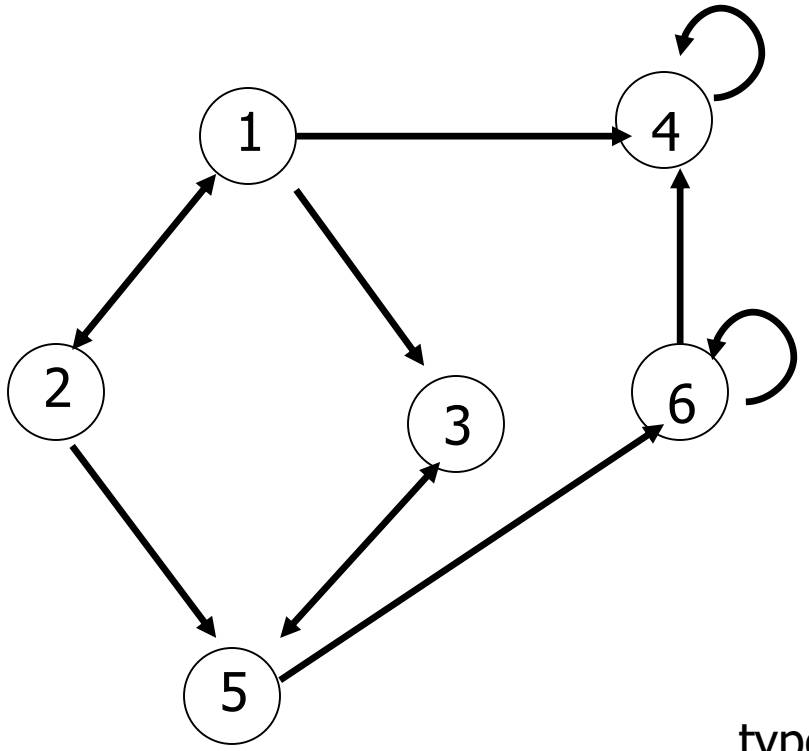
Static 1-D array
of type VERTEX



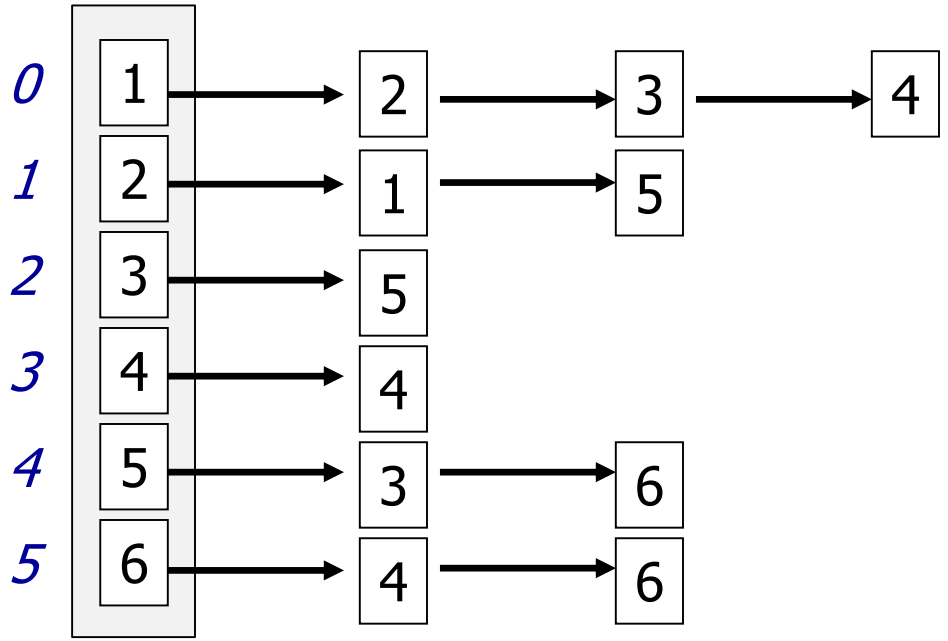
type VERTEX

vertex_number: int edge: singly linked list of type VERTEX

Adjacency List Structure



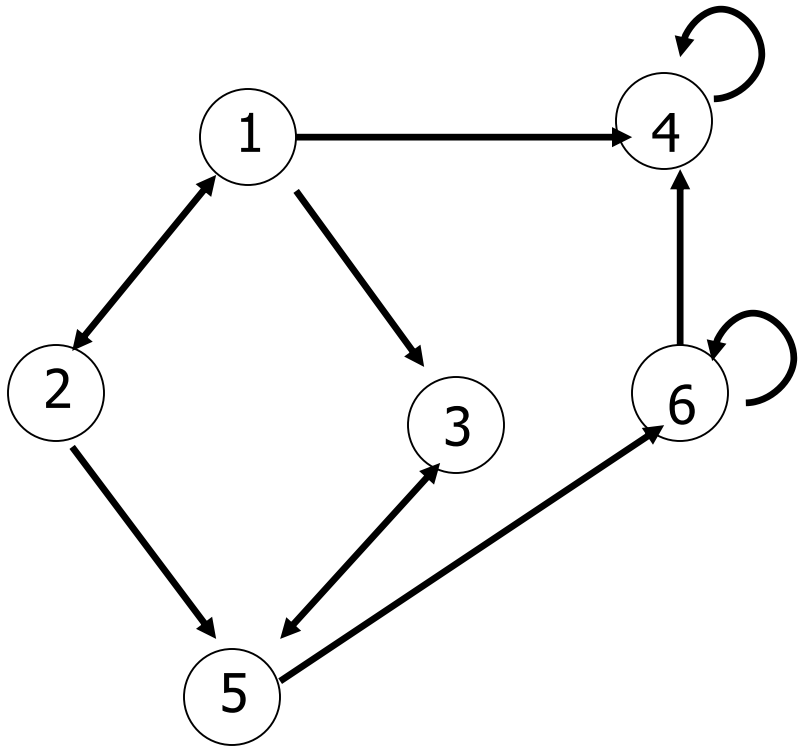
Static 1-D array
of type VERTEX



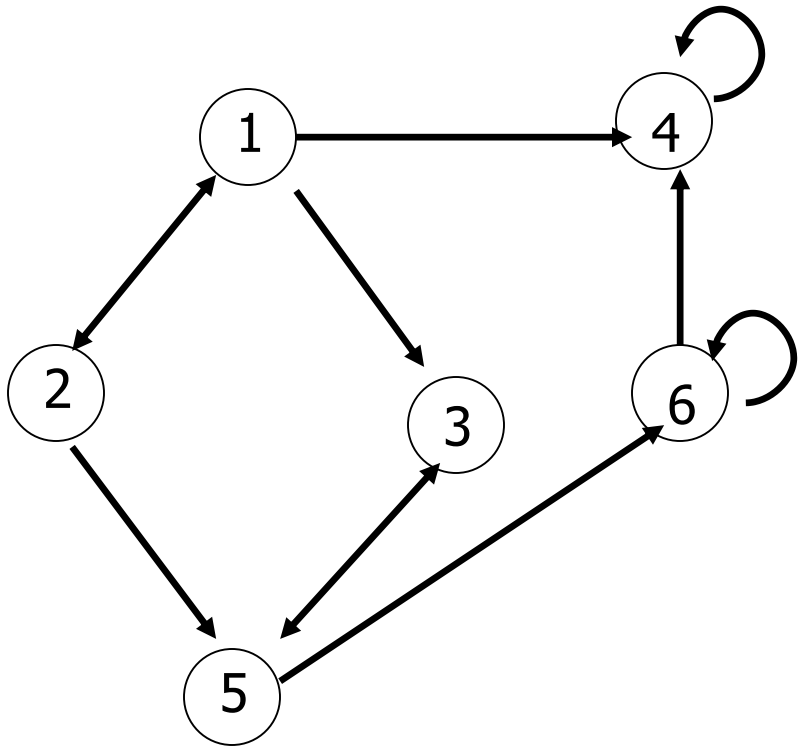
type VERTEX

vertex_number: int edge: singly linked list of type VERTEX

Adjacency Matrix Structure



Adjacency Matrix Structure

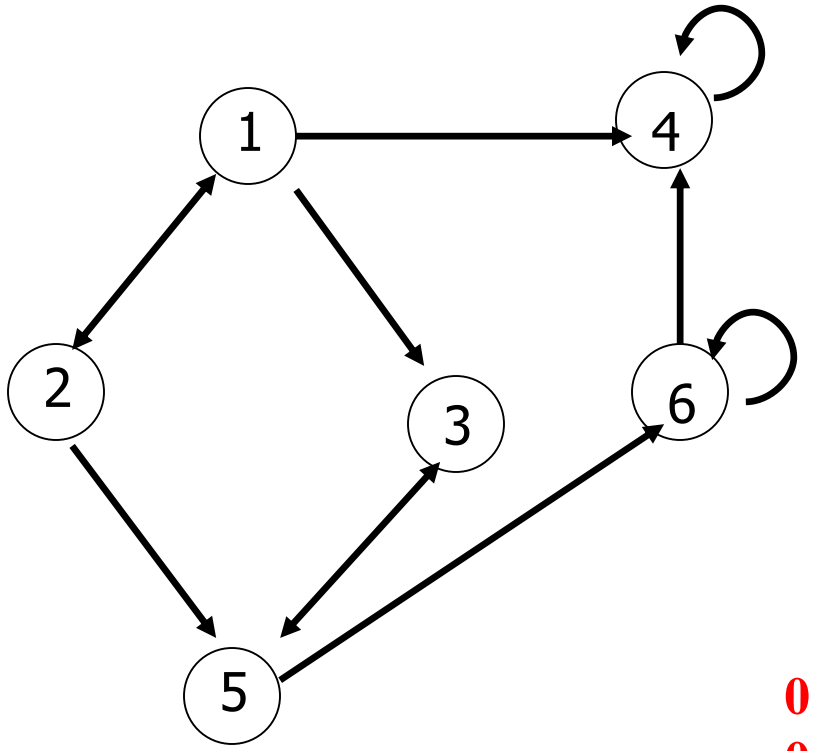


Vertices

	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

Vertices

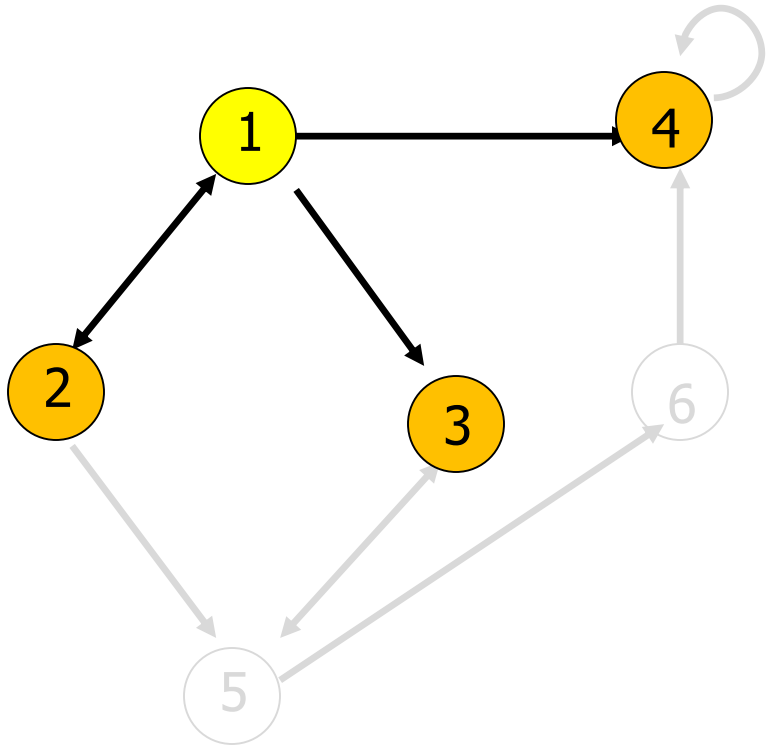
Adjacency Matrix Structure



	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

0 means "no edge linking any of the vertices"
0 means "FALSE"

Adjacency Matrix Structure

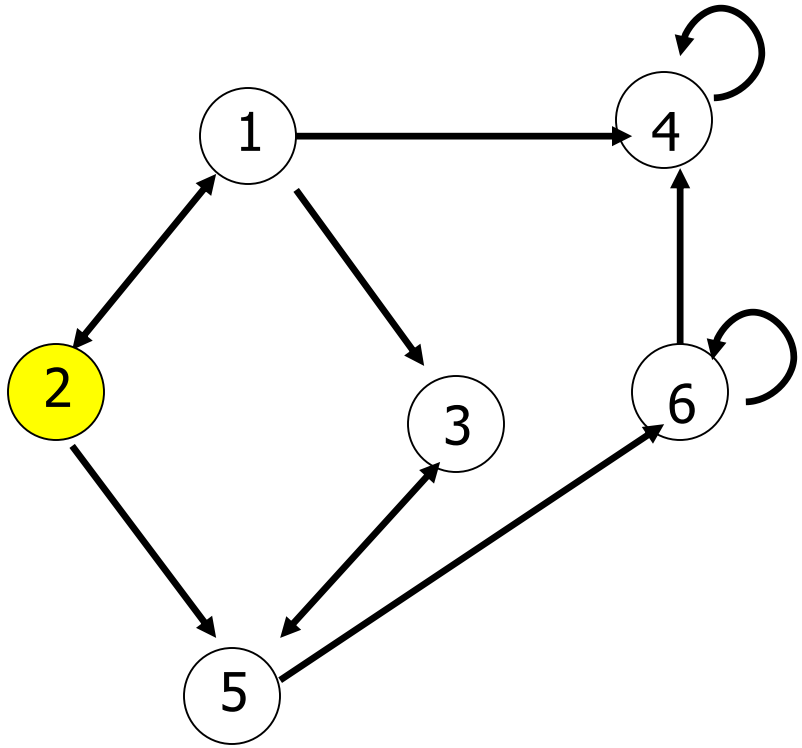


	1	2	3	4	5	6
1	0	**1**	**1**	**1**	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

0 means "no edge linking any of the vertices"
1 means "edge linking vertices"

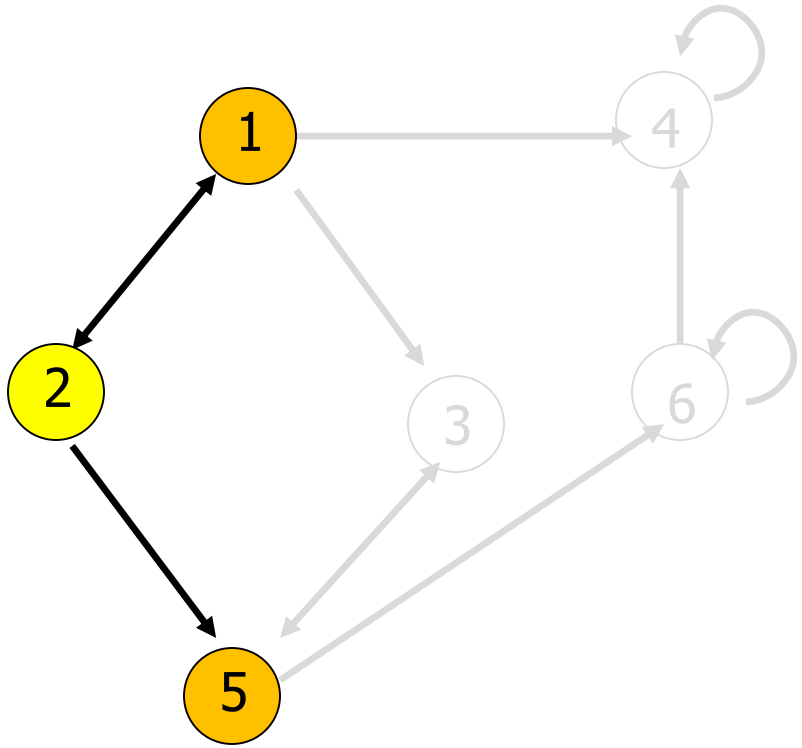
0 means "FALSE"
1 means "TRUE"

Adjacency Matrix Structure



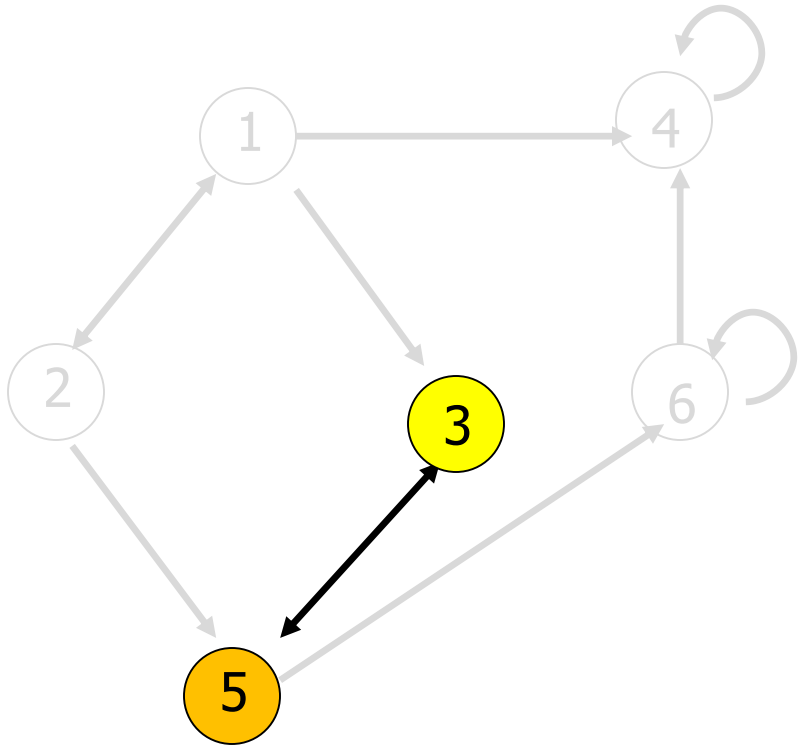
	1	2	3	4	5	6
1	0	1	1	1	0	0
2	1	0	0	0	1	0
3	0	0	0	0	1	0
4	0	0	0	1	0	0
5	0	0	1	0	0	1
6	0	0	0	1	0	1

Adjacency Matrix Structure



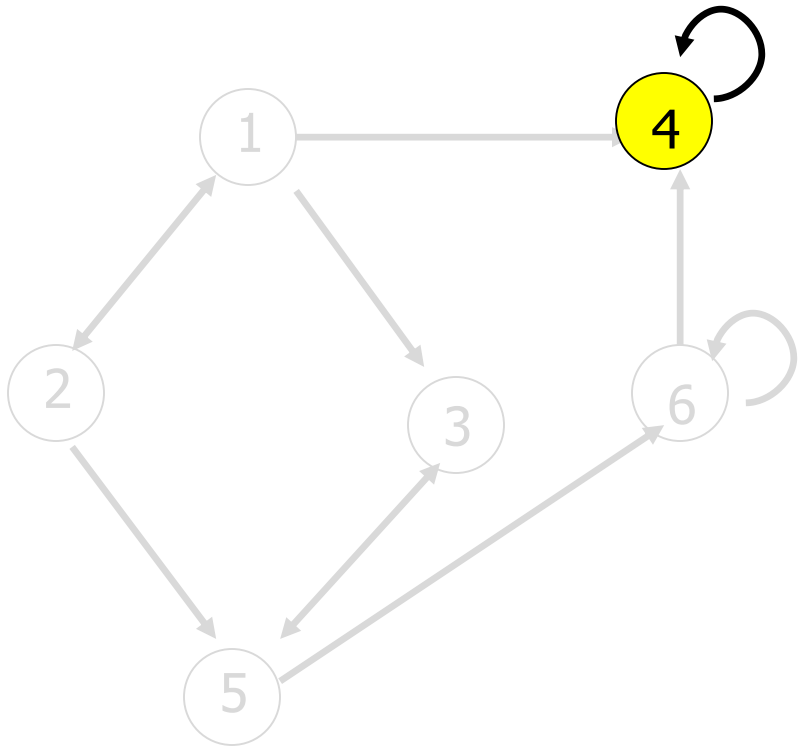
	1	2	3	4	5	6
1	0	1	1	1	0	0
2	1	0	0	0	1	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

Adjacency Matrix Structure



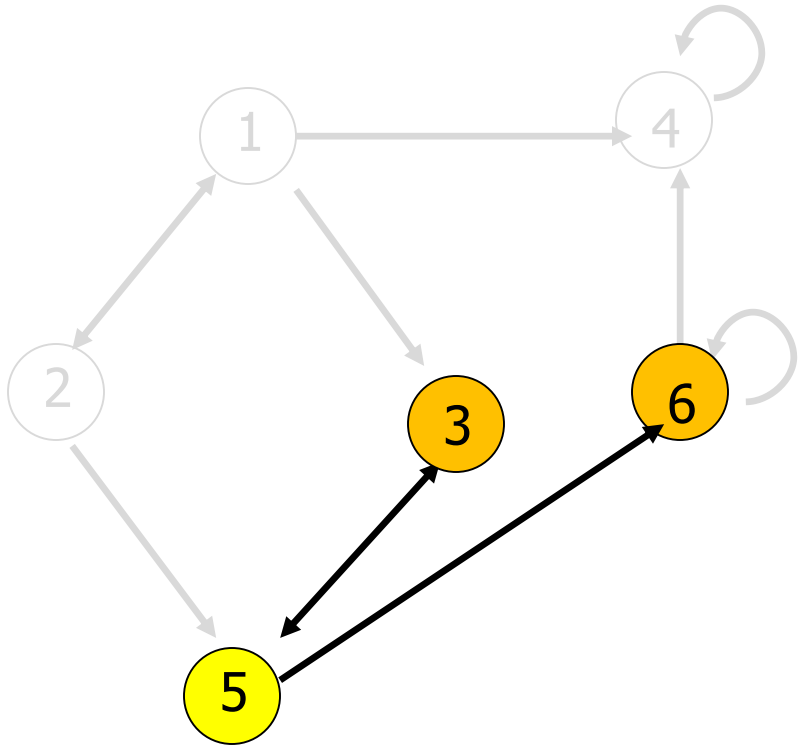
	1	2	3	4	5	6
1	0	1	1	1	0	0
2	1	0	0	0	1	0
3	0	0	0	0	1	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

Adjacency Matrix Structure



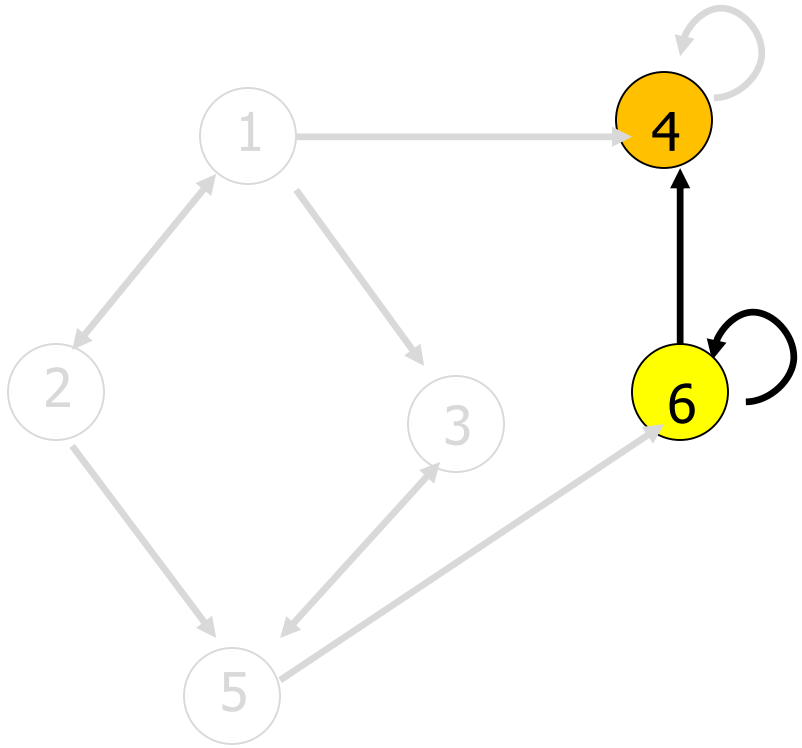
	1	2	3	4	5	6
1	0	1	1	1	0	0
2	1	0	0	0	1	0
3	0	0	0	0	1	0
4	0	0	0	1	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

Adjacency Matrix Structure



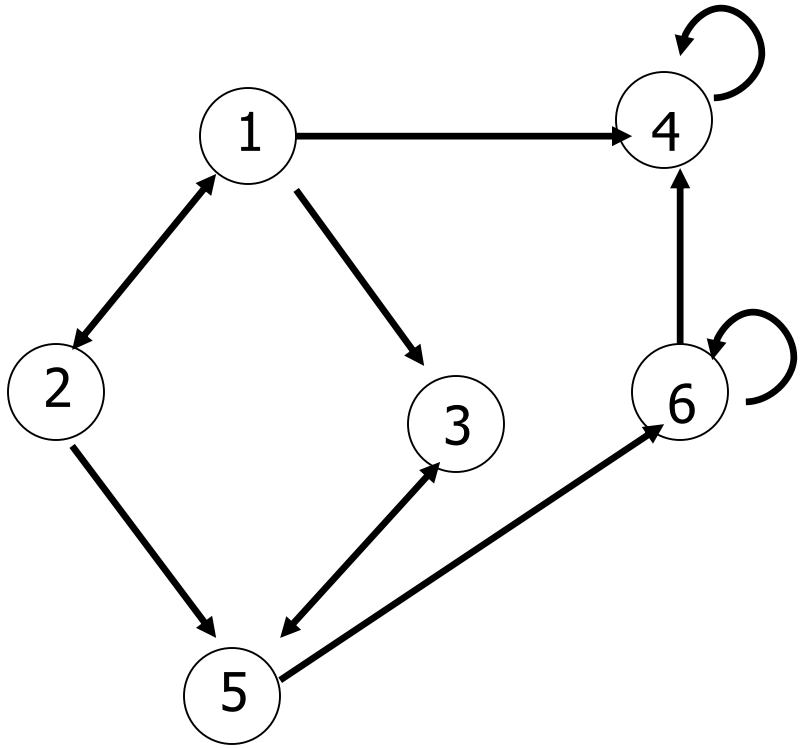
	1	2	3	4	5	6
1	0	1	1	1	0	0
2	1	0	0	0	1	0
3	0	0	0	0	1	0
4	0	0	0	1	0	0
5	0	0	**1**	0	0	**1**
6	0	0	0	0	0	0

Adjacency Matrix Structure



	1	2	3	4	5	6
1	0	1	1	1	0	0
2	1	0	0	0	1	0
3	0	0	0	0	1	0
4	0	0	0	1	0	0
5	0	0	1	0	0	1
6	0	0	0	**1**	0	**1**

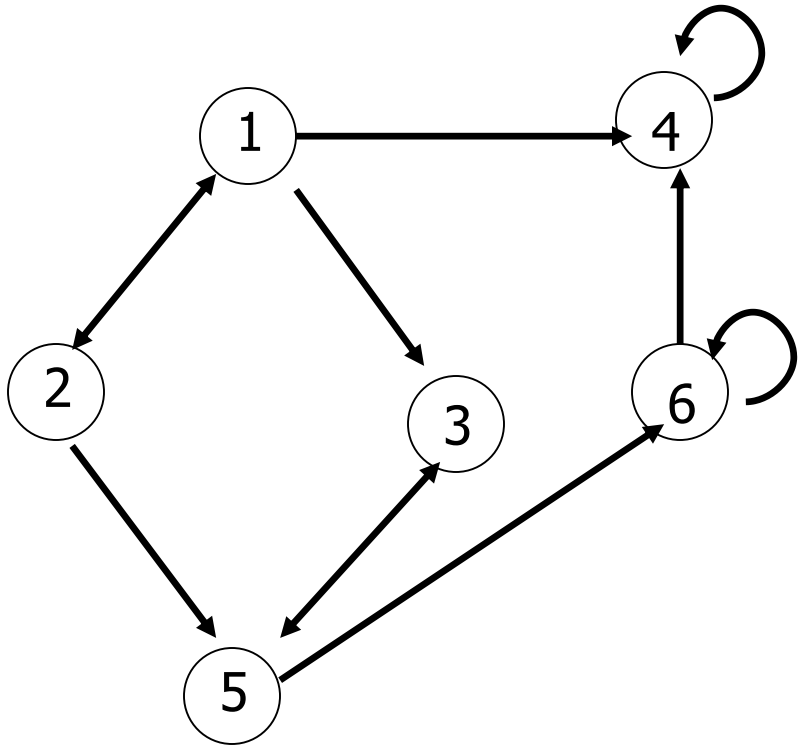
Adjacency Matrix Structure



	1	2	3	4	5	6
1	0	1	1	1	0	0
2	1	0	0	0	1	0
3	0	0	0	0	1	0
4	0	0	0	1	0	0
5	0	0	1	0	0	1
6	0	0	0	1	0	1

0 means "FALSE"
1 means "TRUE"

Adjacency Matrix Structure



	1	2	3	4	5	6
1	F	T	T	T	F	F
2	T	F	F	F	T	F
3	F	F	F	F	T	F
4	F	F	F	T	F	F
5	F	F	T	F	F	T
6	F	F	F	T	F	T

0 means “FALSE”
1 means “TRUE”

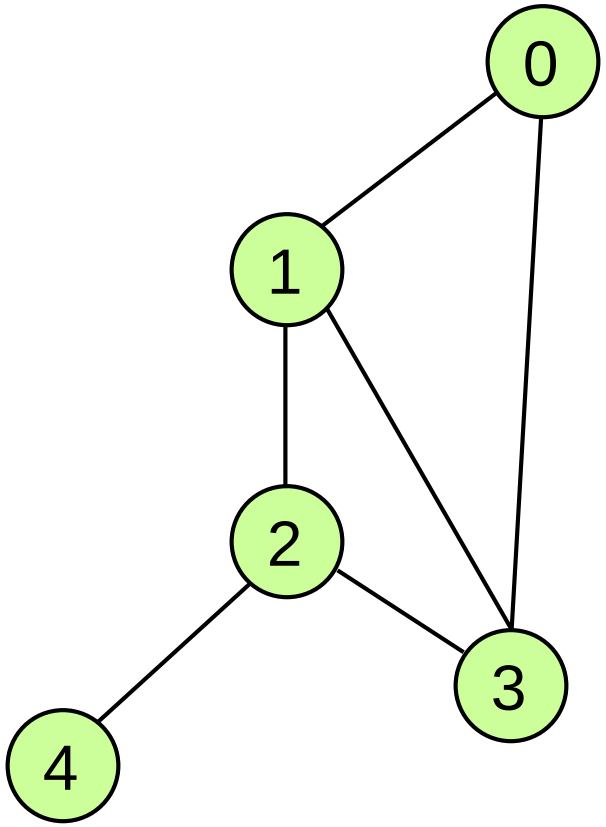
Comparison

UNIVERSITY OF CALGARY

- The adjacency matrix is great for adjacency check O(1)! (but slow on vertex add/removal).

- The adjacency list is great for sparse/few edge graphs (uses heap/pointers but does save space for edges that don't exist.) Also rather efficient to add and query, vertices.

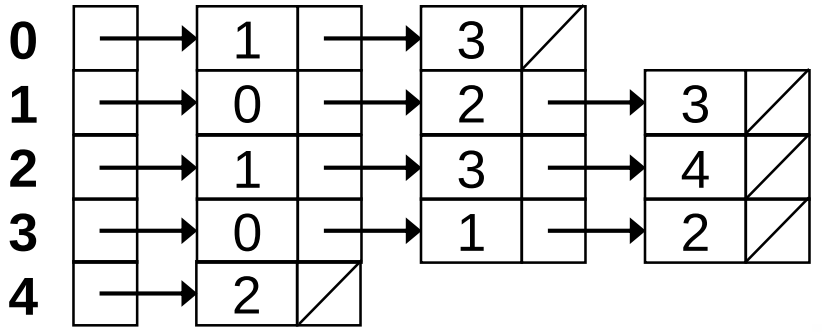
Unweighted, Undirected Graph



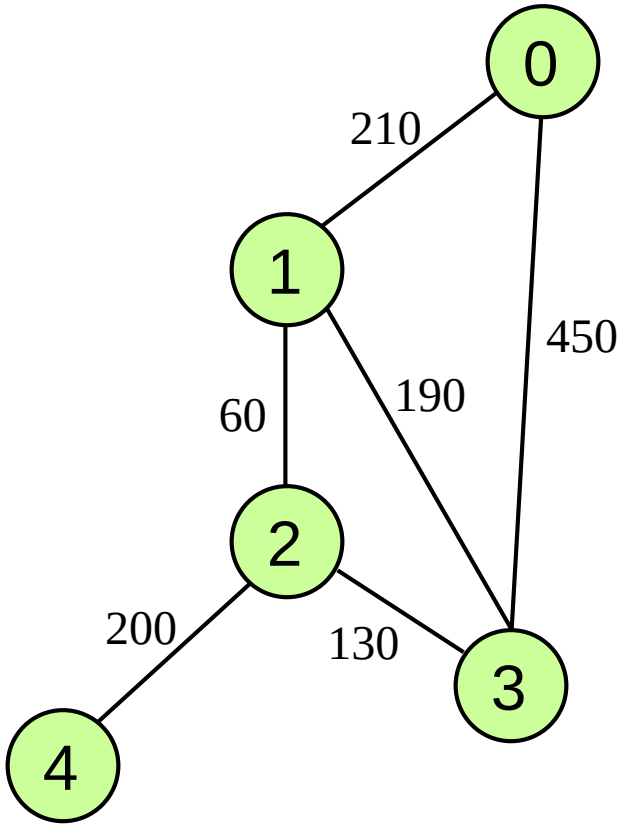
Adjacency matrix

	0	**1**	**2**	**3**	**4**
0	0	1	0	1	0
1	1	0	1	1	0
2	0	1	0	1	1
3	1	1	1	0	0
4	0	0	1	0	0

Adjacency list

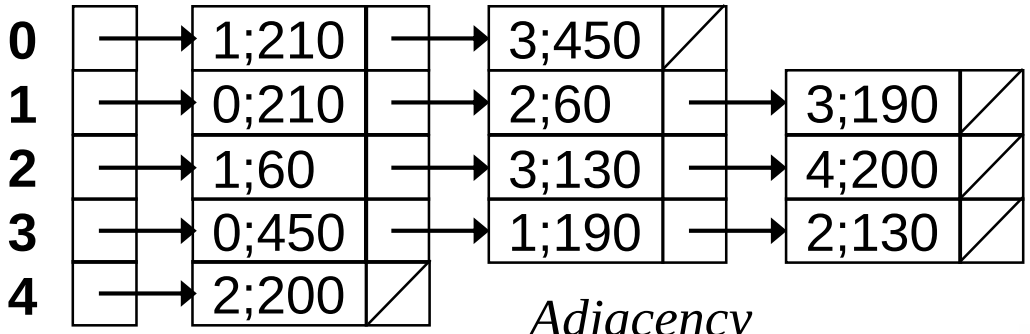


Weighted, Undirected Graph



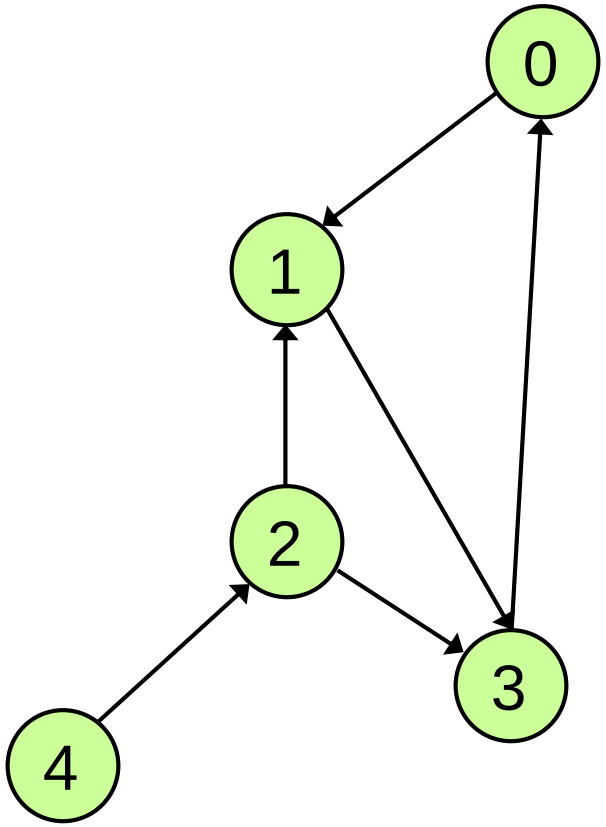
	0	1	2	3	4
0	0	210	0	450	0
1	210	0	60	190	0
2	0	60	0	130	200
3	450	190	130	0	0
4	0	0	200	0	0

Adjacency matrix



Adjacency list

Unweighted, Directed Graph



Adjacency matrix

	0	**1**	**2**	**3**	**4**
0	0	1	0	0	0
1	0	0	0	1	0
2	0	1	0	1	0
3	1	0	0	0	0
4	0	0	1	0	0

Adjacency list