Elementary

Graph Algorithms

PART-1

1

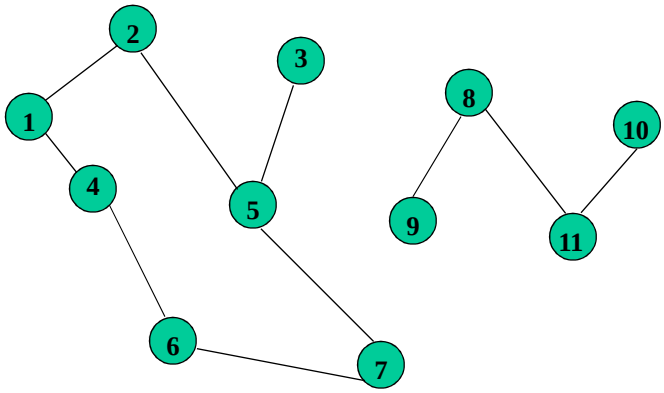
Outlines: Graphs Part-1

- Applications on Graphs
  - Communication Networks
  - Driving Distance /Time Map
  - Street Map
  - Computer Networks
- Graph Terminology
  - Undirected vs. Directed
  - Adjacent Vertices
  - Incident Edges
  - Path & Simple Path
  - Loops
  - Cycles & Simple Cycles
  - Weighted Graph
  - Complete Graph
  - Subgraph
  - Connected Graph & Component
  - Strongly vs. Weakly Connected
  - Symmetric Graph
  - Tree vs. Forest
  - End vertices (End points)
  - Parallel edges
  - Number of Edges
  - Sum of Vertex Degrees
  - In-degree and Out-degree
- Graph Representation
  - Adjacency Matrix
  - Adjacency Lists

2

Applications: Communication Network

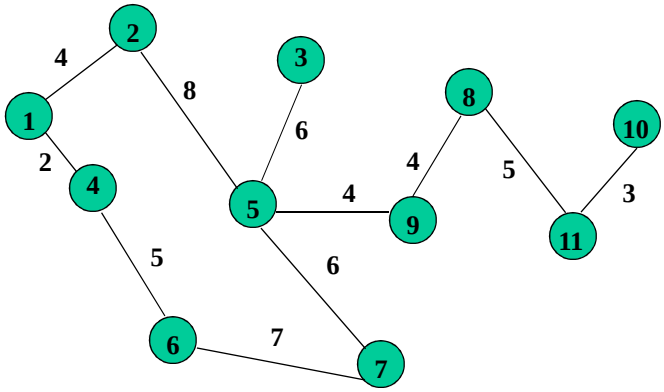
- *vertex* = city, *edge* = communication link



3

Driving Distance/Time Map

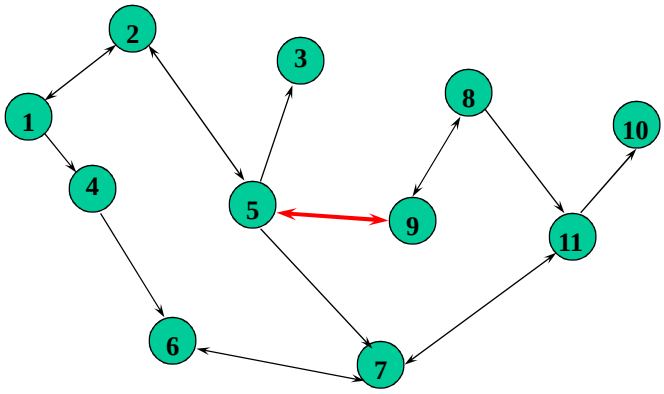
- *vertex* = city,
- *edge* weight = driving distance/time



4

Street Map

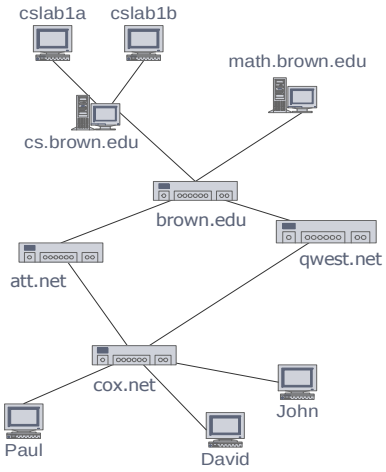
- Some streets are one way
- A ***bidirectional*** link represented by 2 directed edge
  - (5, 9) (9, 5)



5

Computer Networks

- Electronic Circuits
  - Printed Circuit Board
  - Integrated Circuits (ICs)
- Computer networks
  - Local Area Network (LAN)
  - Internet
  - Web

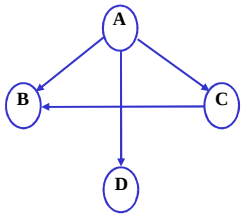


6

Graph Terminology

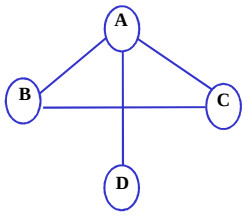
- A graph $G = (V, E)$
  - V = set of vertices (nodes or points)
  - E = set of edges (arcs or lines) = subset of $V \times V$
  - Thus $|E| \leq |V|^2 = O(|V|^2)$
- In an *undirected graph:*
  - $edge(u, v) = edge(v, u)$
- In a *directed* graph:
  - $edge(u, v)$ goes from vertex u to vertex v, notated $u \rightarrow v$
  - $edge(u, v)$ is **NOT** the same as $edge(v, u)$

Graph Terminology



Directed graph:

$V = \{A, B, C, D\}$
$E = \{(A,B), (A,C), (A,D), (C,B)\}$

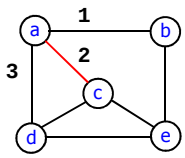


Undirected graph:

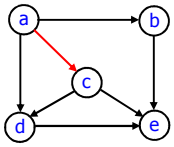
$V = \{A, B, C, D\}$
$E = \{(A,B), (A,C), (A,D), (C,B), (B,A), (C,A), (D,A), (B,C)\}$

Graph Terminology

- ***Adjacent vertices***: connected by an edge
  - Vertex ***v*** is adjacent to ***u*** if and only if **(*u*, *v*)** ∈ ***E***.
  - In an undirected graph with edge **(*u*, *v*)**, and hence **(*v*, *u*)**, ***v*** is adjacent to ***u*** and ***u*** is adjacent to ***v***.
- ***Incident edges***: on a vertex
  - For example, edges:1, 2, and 3 are incident on vertex a.



Vertex *a* is adjacent to *c* and vertex *c* is adjacent to *a*

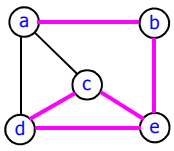


Vertex *c* is adjacent to *a*, but vertex *a* is NOT adjacent to *c*

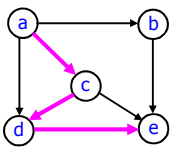
9

Graph Terminology

- *A **Path*** in a graph from *u* to *v* is a **sequence of edges** between vertices $w_0, w_1, \ldots, w_k$, such that $(w_i, w_{i+1}) \in E$, $u = w_0$ and $v = w_k$, for $0 \le i < k$
  - The **length of the path** is *k*, the number of edges on the path



abedce is a path.
cdeb is a path.
bca is NOT a path.

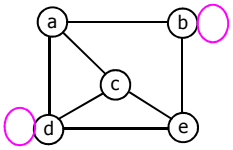


acde is a path.
abec is NOT a path.

10

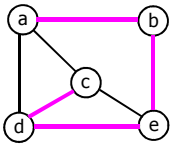
Graph Terminology

- ***Loops***
  - If the graph contains an edge (*v*, *v*) from a vertex to itself, then the path *v*, *v* is sometimes referred to as a ***loop***.



  - The graphs we will consider will generally be loopless.

- ***A simple*** path is a path such that ***all vertices are distinct***, except that the first and last could be the same.

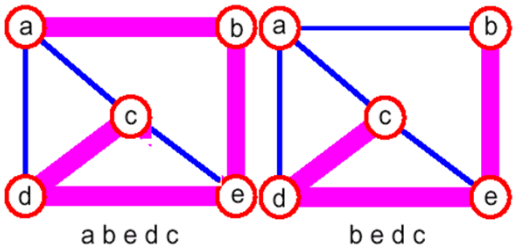


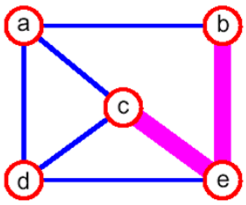
abedc is a simple path.
cdec is a simple path.
abedce is NOT a simple path.

11

Graph Terminology

- Simple path: no repeated vertices



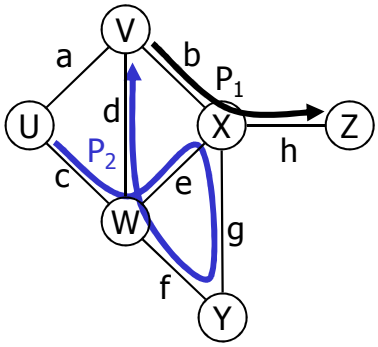


b e c

12

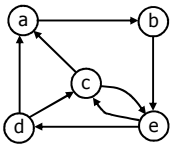
Graph Terminology

- **Path**
  - sequence of alternating vertices and edges
  - begins with a vertex
  - ends with a vertex
  - each edge is preceded and followed by its endpoints
- **Simple path**
  - path such that all its **vertices** and edges are **distinct**.
- **Examples**
  - P_1 = (V, X, Z) is a simple path.
  - P_2 = (U, W, X, Y, W, V) is a path that is not simple.



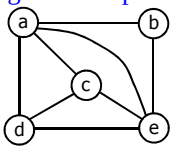
Graph Terminology

- ***Cycles***
  - A ***cycle*** in a **directed graph** is a **path** of length at **least 2** such that the **first** vertex on the path is the same as the **last** one; if the path is **simple**, then the cycle is a ***simple cycle***.



abeda is a simple cycle.
abeceda is a cycle, but is NOT a simple cycle.
abedc is NOT a cycle.

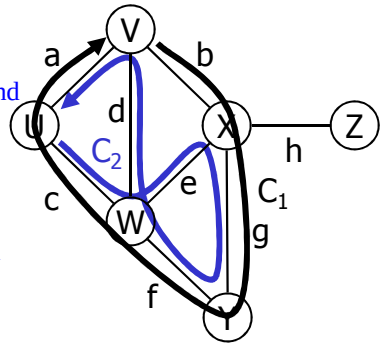
  - A ***cycle*** in a undirected graph
    - A path of length at **least 3** such that the **first** vertex on the path is the same as the **last** one.
    - The edges on the path are **distinct**.



aba is NOT a cycle.
abedceda is NOT a cycle.
abedcea is a cycle, but NOT simple.
abea is a simple cycle.

Graph Terminology

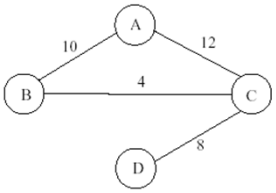
- **Cycle**
  - circular sequence of alternating vertices and edges
  - each edge is preceded and followed by its endpoints
- Simple cycle
  - cycle such that all its **vertices** and edges are **distinct**
- Examples
  - C_1 = (V, X, Y, W, U, V) is a simple cycle
  - C_2 = (U, W, X, Y, W, V, U) is a cycle that is not simple



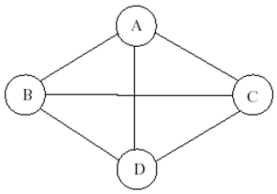
15

Graph Terminology

- If each edge in the graph carries a value, then the graph is called ***weighted graph***.
  - A weighted graph is a graph $G = (V, E, W)$, where each edge, $e \in E$ is assigned a real valued weight, $W(e)$.
- A ***complete graph*** is a graph with an edge between every pair of vertices.
  - A graph is called ***complete graph*** if every vertex is adjacent to every other vertex.



weighted graph

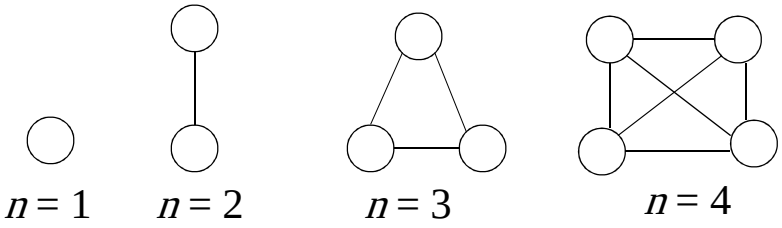


complete graph

16

Graph Terminology

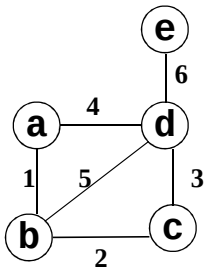
- Complete Undirected Graph
  - has all possible edges



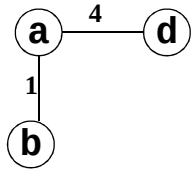
17

Graph Terminology

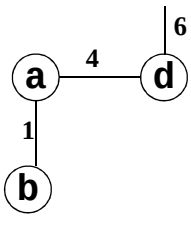
- Subgraph: subset of vertices and edges forming a graph
  - A graph $G_s = (V_s, E_s)$ is a ***subgraph*** of a graph $G = (V, E)$ if $V_s \subseteq V$, $E_s \subseteq E$, and $E_s \subseteq V_s \times V_s$.



G



is a sub-graph



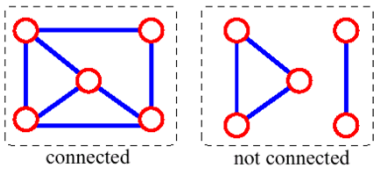
Not a sub-graph

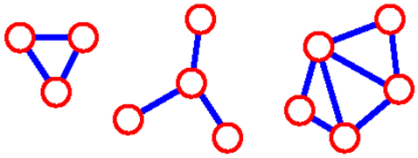
since, $E_s = \{1,4,6\} \not\subset V_s \times V_s = \{1,4\}$

18

Graph Terminology

- connected graph: any two vertices are connected by some path
  - An undirected graph is ***connected*** if, for every pair of vertices *u* and *v* there is a path from *u* to *v*.
- connected component: maximal connected subgraph. E.g., the graph below has 3 connected components





19

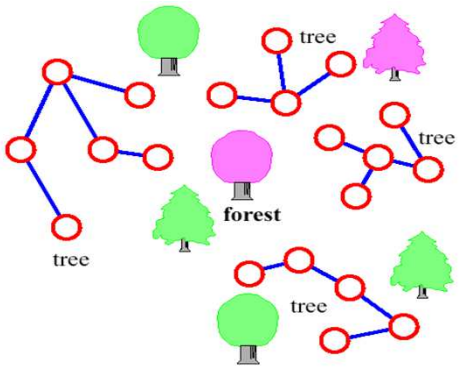
Graph Terminology

- A *directed graph* is ***strongly connected*** if every two vertices *are reachable from each other*.
  - Note that in a directed graph, the existence of a path from *u* to *v* does not imply there is a path from *v* to *u*.
- A directed graph that is *not strongly connected*, but the ***underlying graph is connected***, is called ***weakly connected***.
- A ***symmetric digraph*** is a directed graph, $G = (V, E)$, such that if $(u, v) \in E$, then $(v, u) \in E$.

20

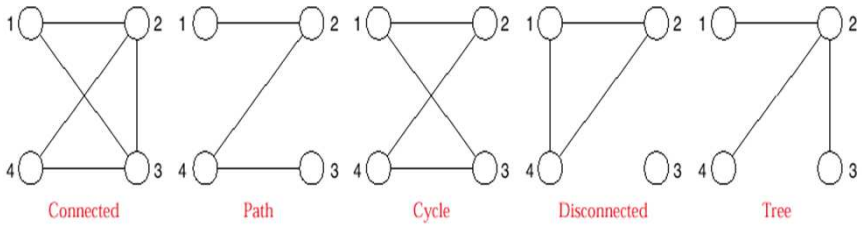
Graph Terminology

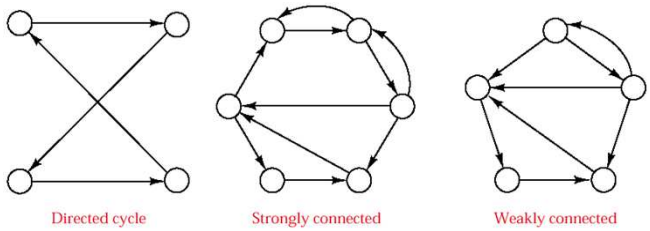
- tree - connected undirected graph without cycles
- forest - collection of trees



21

Graph Terminology

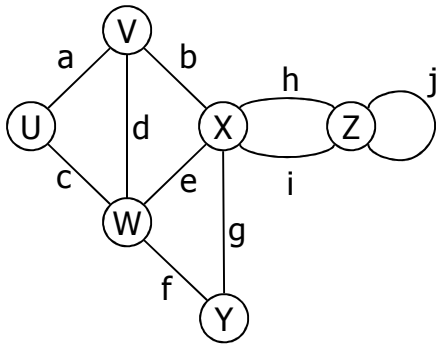




22

Graph Terminology

- ***End vertices*** (or ***endpoints***) of an ***edge***
  - **U** and **V** are the endpoints of **a**
- **Edges incident** on a vertex
  - **a**, **d**, and **b** are incident on **V**
- **Adjacent vertices**
  - **U** and **V** are adjacent
- **Degree of a vertex**
  - **X** has degree 5
- **Parallel edges**
  - **h** and **i** are parallel edges
- **Self-loop**
  - **j** is a self-loop

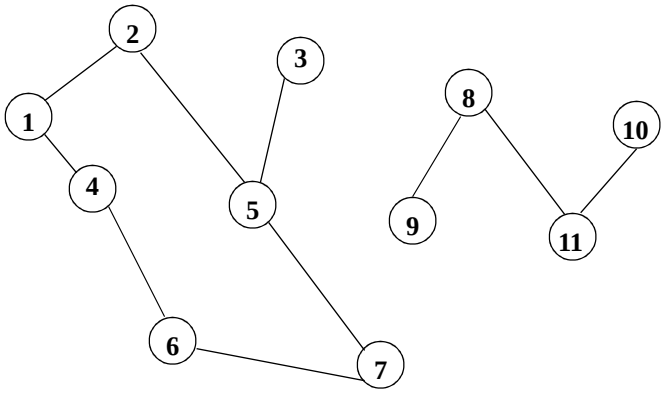


Number of Edges

- Each edge is of the form (u, v), $u \neq v$
- Number of such pairs in an n vertex graph is $n(n-1)$
- **Undirected Graph**
  - Since *edge*(u, v) is the same as *edge*(v, u), the number of edges in a ***complete undirected*** graph is $n(n-1)/2$
  - Number of edges in an ***undirected*** graph is $\leq n(n-1)/2$
- **Directed Graph**
  - Since *edge*(u, v) is not the same as *edge*(v, u), the number of edges in a ***complete directed*** graph is $n(n-1)$
  - Number of edges in a **directed** graph is $\leq n(n-1)$

Vertex Degree

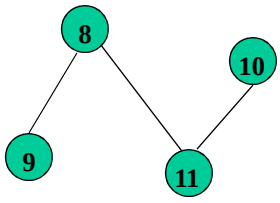
- degree (of a **vertex**): # of adjacent vertices
  - degree(2) = 2, degree(5) = 3, degree(3) = 1



25

Sum of Vertex Degrees (undirected graph)

- Sum of degrees of all vertex = $2|E|$
  - Since adjacent vertices each count the adjoining edge, it will be counted twice

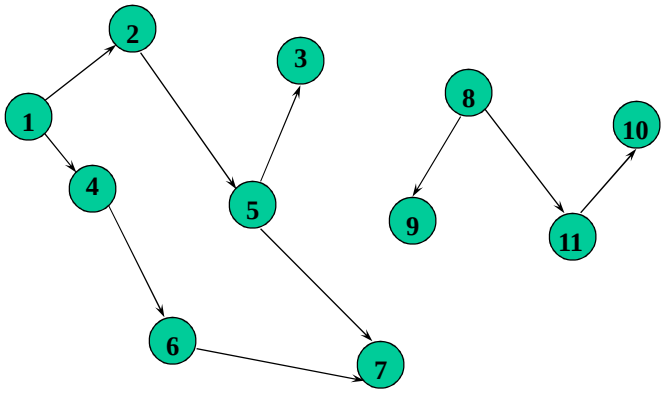


$$\sum_{v \in V} \deg(v) = 2(\#\ \text{of edges})$$

26

In-Degree of a Vertex (digraph)

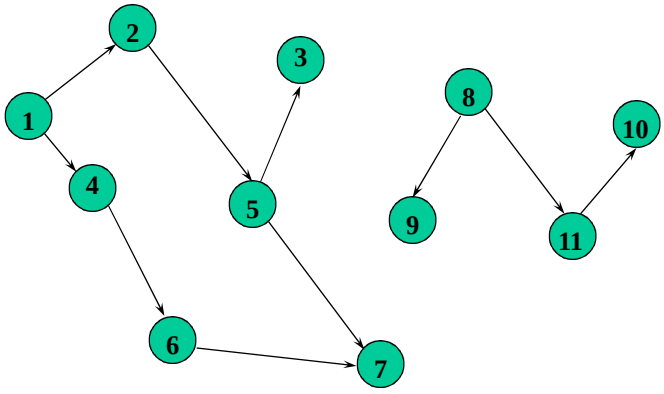
- **in-degree** is number of incoming edges
  - indegree(2) = 1, indegree(8) = 0



27

Out-Degree of a Vertex (digraph)

- **out-degree** is number of outbound edges
  - outdegree(2) = 1, outdegree(8) = 2



28

Sum of In- and Out-Degrees (digraph)

- Each edge contributes 1 to the **in-degree** of some vertex and 1 to the **out-degree** of some other vertex
- Sum of in-degrees = sum of out-degrees = $|E|$,
  - where $|E|$ is the number of edges in the digraph

29

Graphs

- We will typically express running times in terms of
  - $|V|$ = number of vertices, and
  - $|E|$ = number of edges
  - If $|E| \approx |V|^2$ the graph is ***dense***
  - If $|E| \approx |V|$ the graph is ***sparse***
- If you know you are dealing with dense or sparse graphs, different data structures may make sense

30

Graph Representation

- Adjacency Matrix
- Adjacency Lists

31

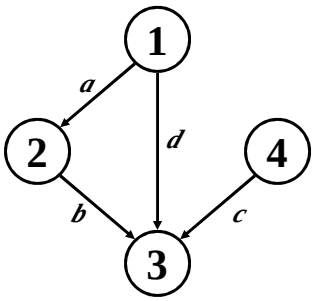
Adjacency Matrix

- Assume $V = \{1, 2, \ldots, n\}$
- An ***adjacency matrix*** represents the graph as a $n \times n$ matrix A:
  - $A[i, j]$ = 1 if $edge(i, j) \in E$ (or weight of edge)
    = 0 if $edge(i, j) \notin E$

32

Adjacency Matrix

- Example:

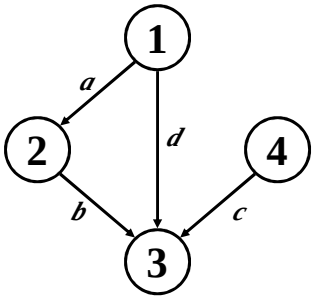


A	1	2	3	4
1				
2				
3			??	
4				

33

Adjacency Matrix

- Example:

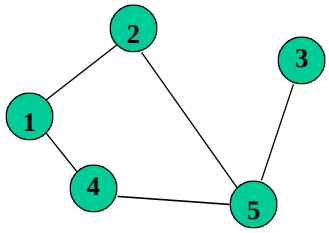


A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

34

Adjacency Matrix

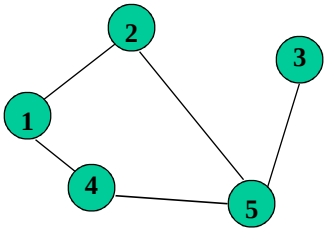
- 0/1 $n \times n$ matrix, where n = # of vertices
- $A(i, j) = 1$ iff (i, j) is an edge



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

35

Adjacency Matrix

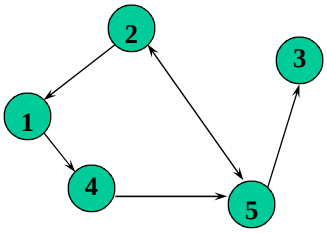


	1	2	3	4	5
1	0	1	0	1	0
2	1	0	0	0	1
3	0	0	0	0	1
4	1	0	0	0	1
5	0	1	1	1	0

- Diagonal entries are zero
- Adjacency matrix of an **undirected graph** is symmetric
  - $A(i, j) = A(j, i)$ for all i and j

36

Adjacency Matrix



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

- Diagonal entries are zero
- Adjacency matrix of a **digraph** need not be symmetric

37

Adjacency Matrix

- *How much **storage** does the adjacency matrix require?*
  - Answer: $O(|V|^2)$
- $O(|V|)$ time to find vertex degree and/or ***vertices*** **adjacent** to a given vertex
- *What is the **minimum amount of storage** needed by an adjacency matrix representation of an **undirected** graph with 4 vertices?* $(|V|-1)|V|/2$
  - Answer: 6 bits
  - Undirected graph → matrix is symmetric
  - No self-loops → don't need diagonal

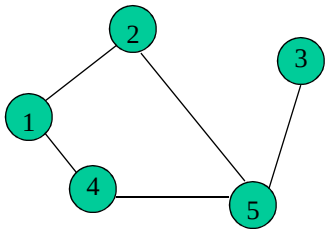
38

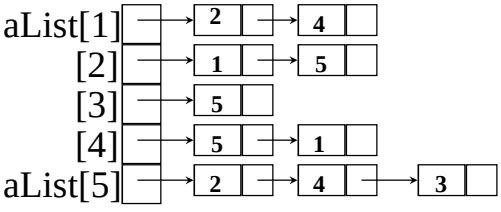
Adjacency Matrix

- The adjacency **matrix** is a **dense** representation
  - Usually too much storage for large graphs
  - But can be very efficient for **small** graphs
  - If graph is **unweighted**, there is an additional advantage in storage for the adjacency matrix presentation; rather than using one word of computer memory for each matrix entry, the adjacency matrix uses only one bit per entry.
  - Prefer to be a **directed** graph.
- Most large interesting graphs are **sparse**
  - For this reason the ***adjacency list*** is often a more appropriate representation

Adjacency List

- Adjacency list: for each vertex $v \in V$, store a list of vertices adjacent to v.
- Adjacency list for vertex i is a **linear list** of vertices adjacent from vertex i
- Each adjacency list is a **chain**.

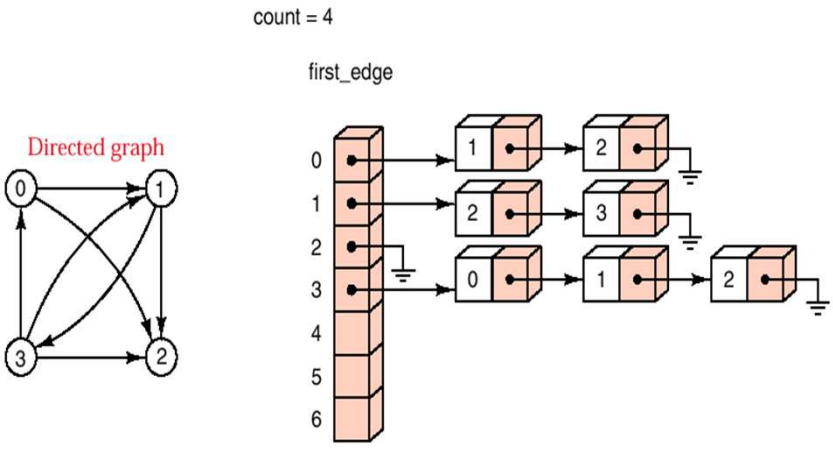




# of chain nodes = $2|E|$ (undirected graph)

# of chain nodes = $|E|$ (digraph)

Adjacency List



41

Adjacency List

- How much storage is required?
  - The *degree* of a vertex v = # incident edges
    - Directed graphs have in-degree and out-degree
  - For directed graphs:
    - # of items in adjacency lists is Σ out-degree(v) = $|E|$
    - takes $\Theta(V + E)$ storage (*Why?*)
  - For undirected graphs:
    - # items in adjacency lists is Σ degree(v) = $2|E|$
    - also $\Theta(V + E)$ storage
- So: Adjacency lists take $O(V + E)$ storage

42