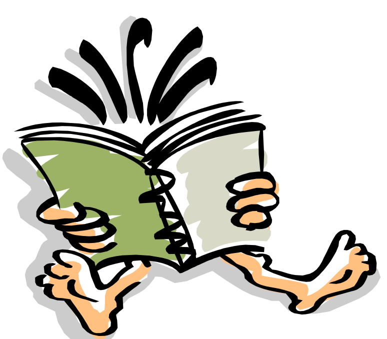
Analysis of Algorithms

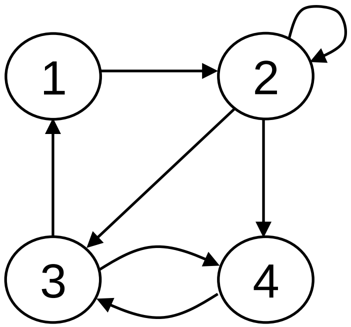
Graphs

Appendix B4, Appendix B5.1
Chapter 22



Graphs

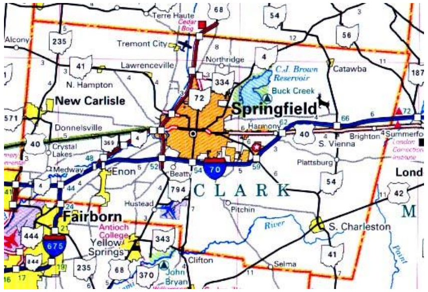
Definition = a set of nodes (vertices) with edges (links) between them.



- $G = (V, E)$ - graph
- V = set of vertices $|V| = n$
- E = set of edges $|E| = m$
  - Binary relation on V
  - Subset of $V \times V = \{(u,v): u \in V, v \in V\}$

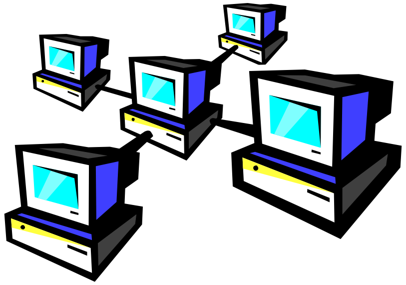
Applications

- Applications that involve not only a set of items, but also the connections between them

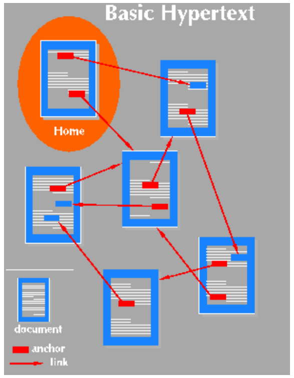


Maps

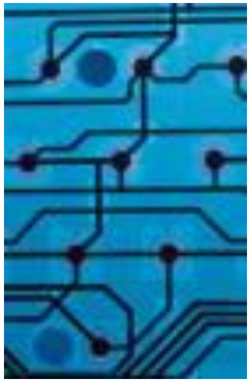
Schedules



Computer networks



Hypertext

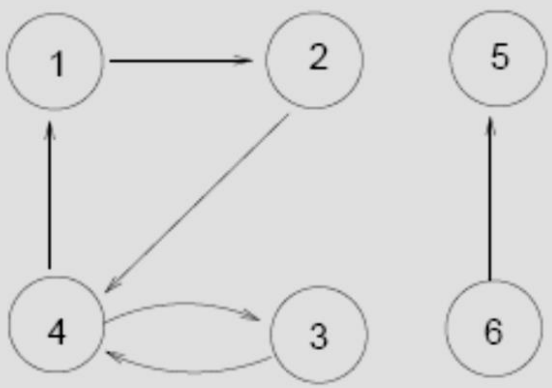


Circuits

Terminology

- Directed vs Undirected graphs

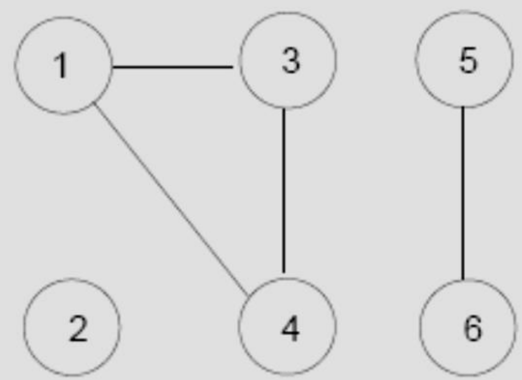
Directed graphs (digraphs)
(ordered pairs of vertices)



in-degree of v: # edges enetring v
out-degree of v: # edges leaving v

v is **adjacent** to u if there is an edge (u,v)

Undirected graphs
(unordered pairs of vertices)

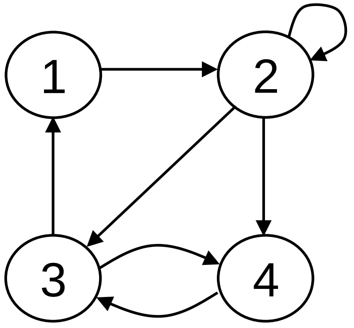


degree of v: # edges incident on v

v is **adjacent** to u and u is **adjacent** to v if there is and edge (u,v)

Terminology (cont'd)

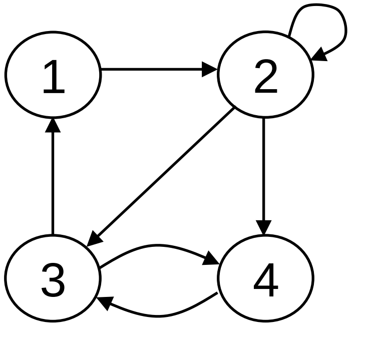
- Complete graph
  - A graph with an edge between each pair of vertices
- Subgraph
  - A graph (V', E') such that V'⊆V and E'⊆E
- Path from v to w
  - A sequence of vertices <v_0, v_1, …, v_k> such that v_0=v and v_k=w
- Length of a path
  - Number of edges in the path



path from v_1 to v_4
<v_1, v_2, v_4>

Terminology (cont'd)

- w is **reachable** from v
  - If there is a path from v to w
- **Simple** path
  - All the vertices in the path are distinct
- Cycles
  - A path $<v_0, v_1, \ldots, v_k>$ forms a cycle if $v_0=v_k$ and $k \geq 2$
- Acyclic graph
  - A graph without any cycles



cycle from v_1 to v_1
$<v_1, v_2, v_3, v_1>$

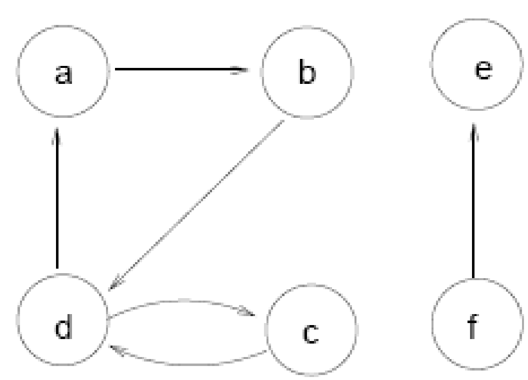
Terminology (cont'd)

Connected and Strongly Connected

directed graphs

strongly connected: every two vertices are reachable from each other

strongly connected components : all possible strongly connected subgraphs

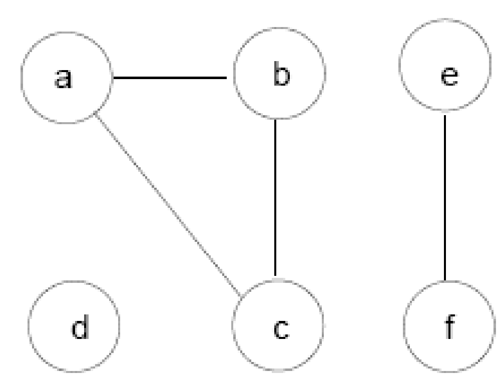


strongly connected components: {a,b,c,d} { e} {f}

undirected graphs

connected: every pair of vertices is connected by a path

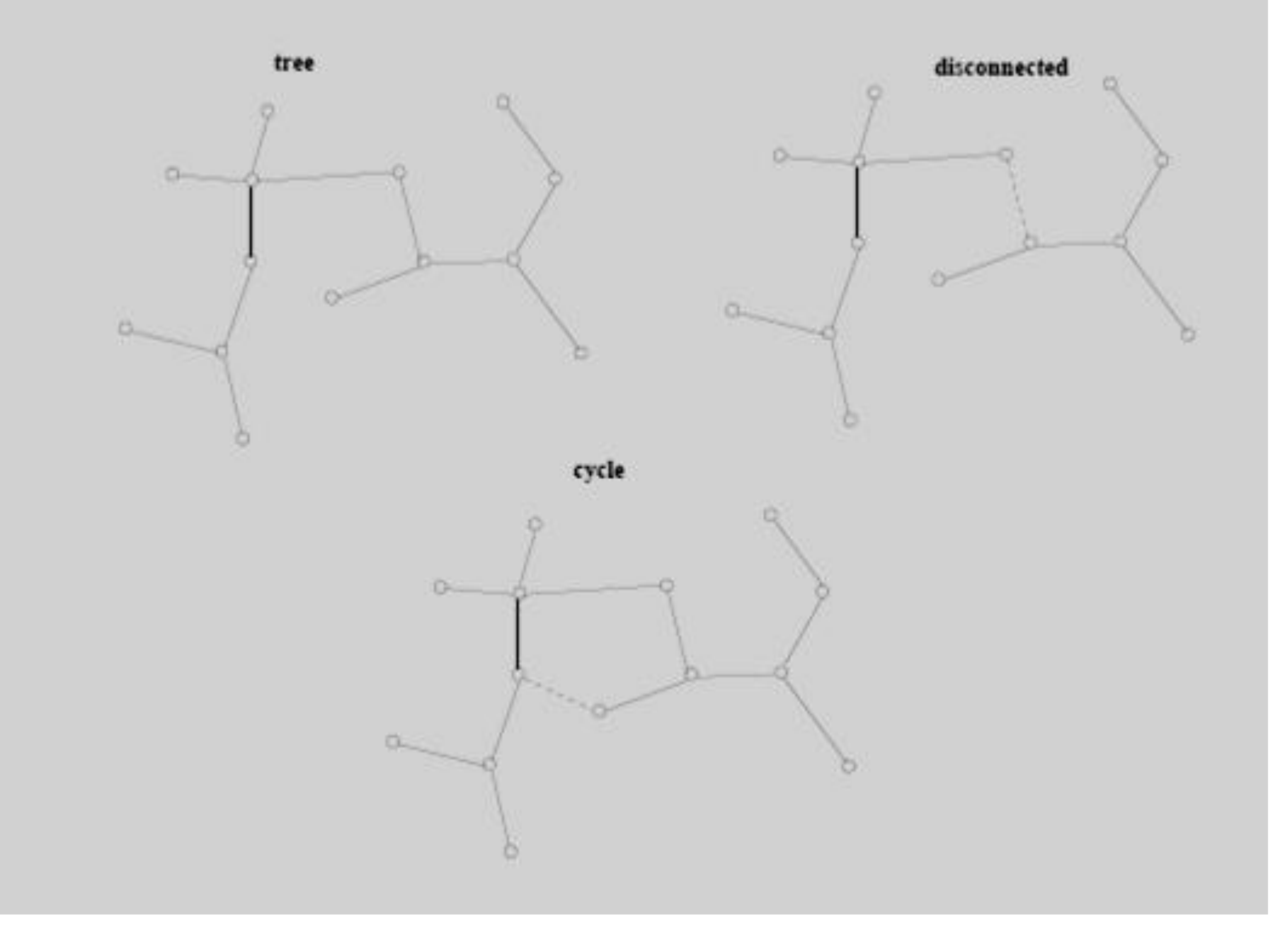
connected components: all possible connected subgraphs



connected components: {a,b,c} {d} {e,f}

Terminology (cont'd)

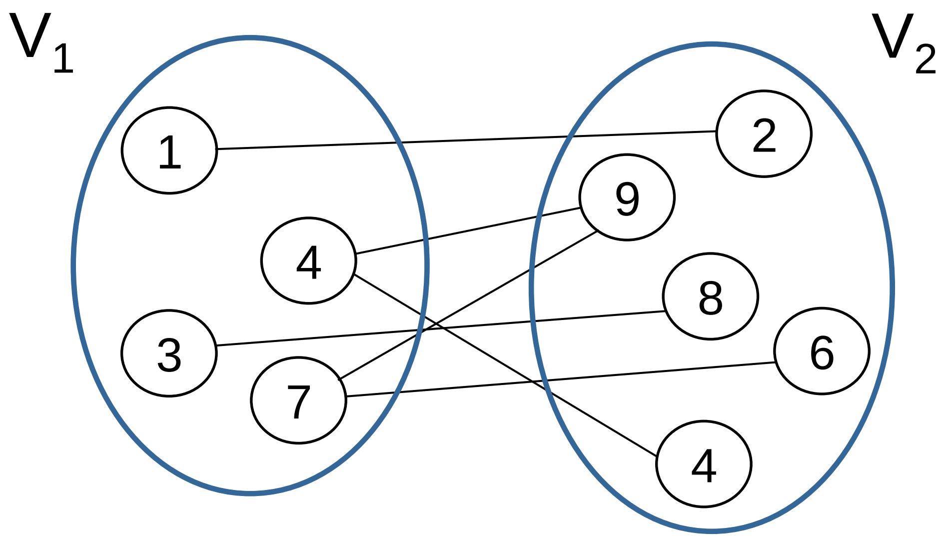
- A **tree** is a connected, acyclic undirected graph



Terminology (cont'd)

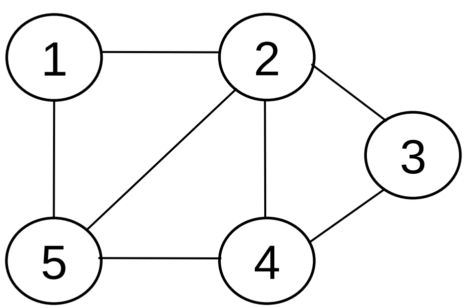
- A **bipartite graph** is an undirected graph

$G = (V, E)$ in which $V = V_1 + V_2$ and there are edges only between vertices in V_1 and V_2

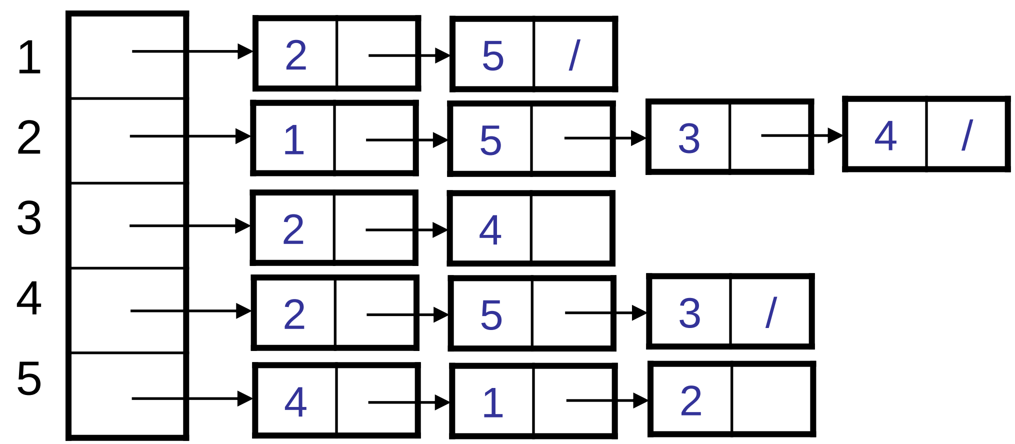


Graph Representation

- **Adjacency list representation** of G = (V, E)
  - An array of |V| lists, one for each vertex in V
  - Each list `Adj[u]` contains all the vertices v that are adjacent to u (i.e., there is an edge from u to v)
  - Can be used for both directed and undirected graphs

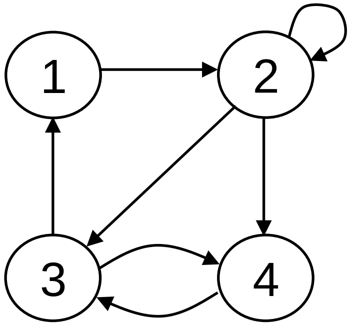


Undirected graph

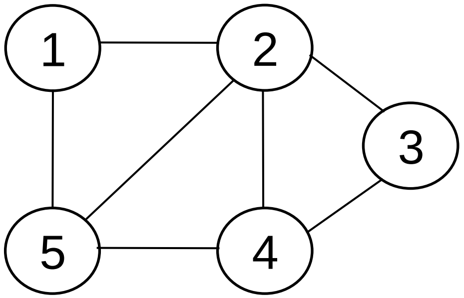


Properties of Adjacency-List Representation

- Sum of "lengths" of all adjacency lists
  - Directed graph: $|E|$
    - edge (u, v) appears only once (i.e., in the list of **u**)
  - Undirected graph: $2|E|$
    - edge (u, v) appears twice (i.e., in the lists of both **u** and **v**)



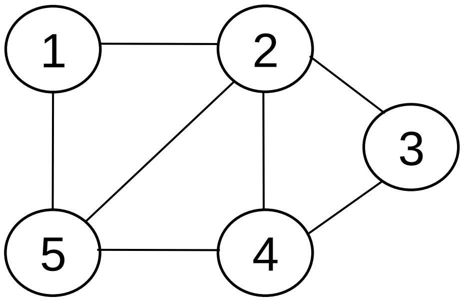
Directed graph



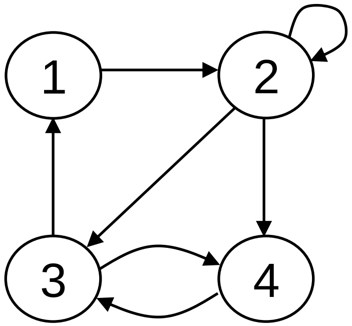
Undirected graph

Properties of Adjacency-List Representation

- Memory required
  - $\Theta(V + E)$
- Preferred when
  - The graph is **sparse**: $|E| << |V|^2$
  - We need to quickly determine the nodes adjacent to a given node.
- Disadvantage
  - No quick way to determine whether there is an edge between node u and v
- Time to determine if $(u, v) \in E$:
  - $O(\text{degree}(u))$
- Time to list all vertices adjacent to u:
  - $\Theta(\text{degree}(u))$



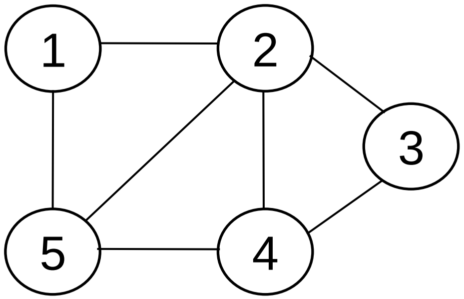
Undirected graph



Directed graph

Graph Representation

- **Adjacency matrix representation** of G = (V, E)
  - Assume vertices are numbered 1, 2, … $|V|$
  - The representation consists of a matrix $A_{|V| \times |V|}$:
  - $a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$



Undirected graph

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

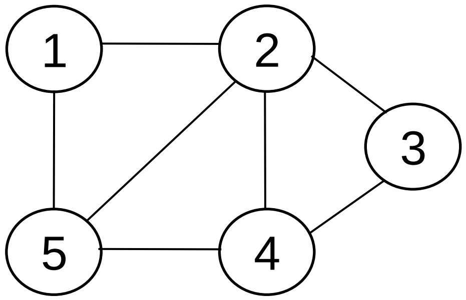
For undirected graphs, matrix A is symmetric:

$a_{ij} = a_{ji}$

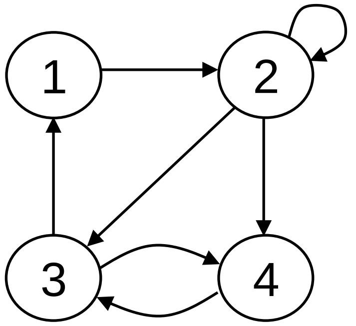
$A = A^T$

Properties of Adjacency Matrix Representation

- Memory required
  - $\Theta(V^2)$, independent on the number of edges in G
- Preferred when
  - The graph is **dense:** $|E|$ is close to $|V|^2$
  - We need to quickly determine if there is an edge between two vertices
- Time to determine if $(u, v) \in E$:
  - $\Theta(1)$
- Disadvantage
  - No quick way to determine the vertices adjacent to another vertex
- Time to list all vertices adjacent to u:
  - $\Theta(V)$



Undirected graph

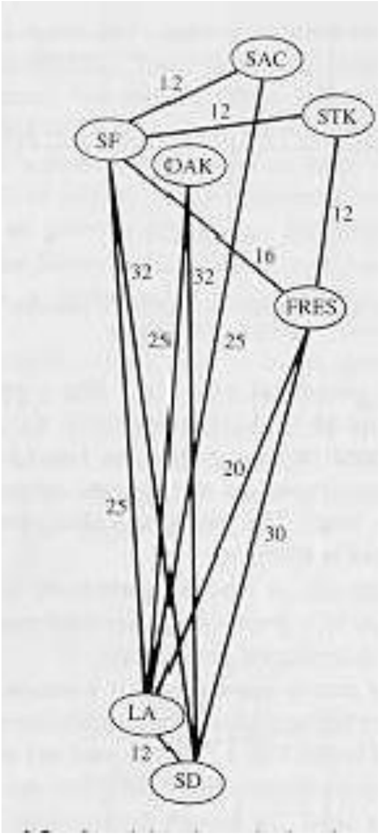


Directed graph

Weighted Graphs

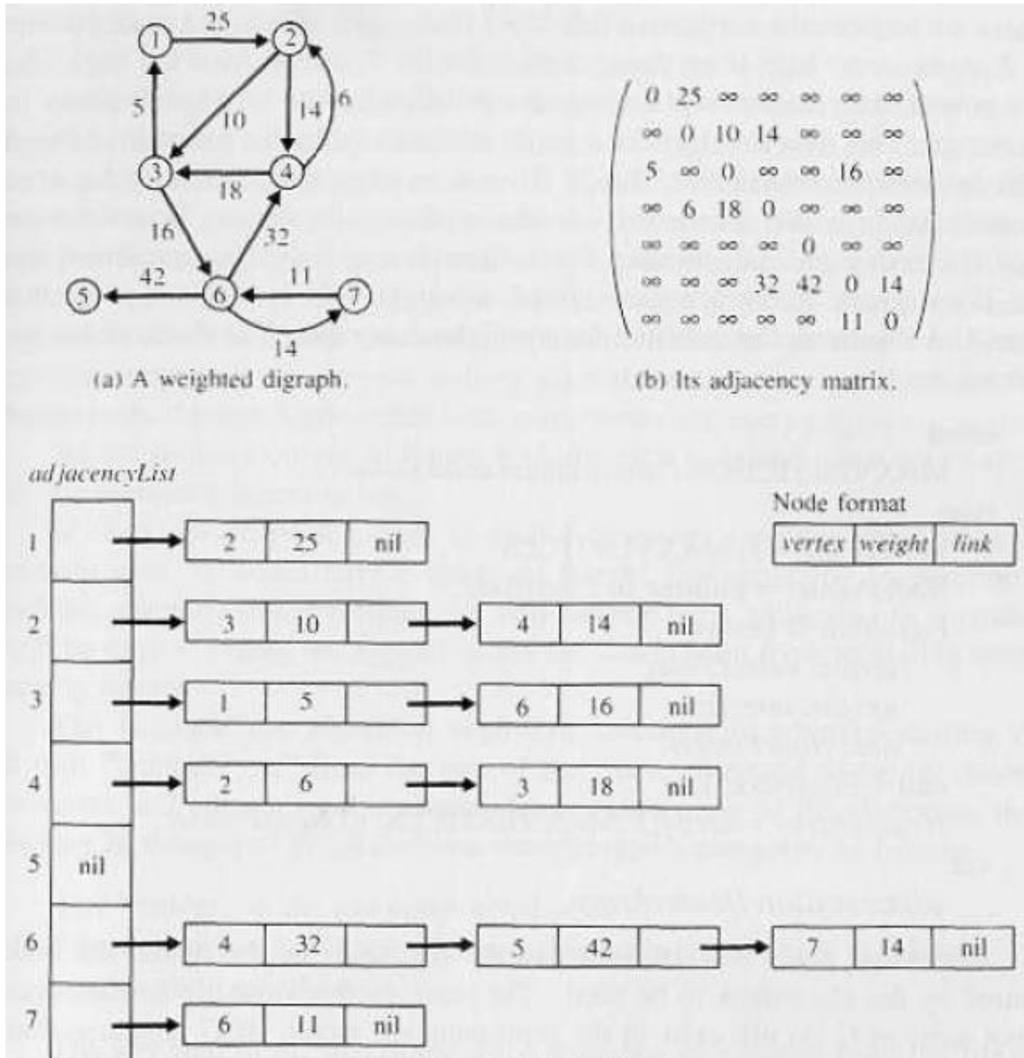
- Graphs for which each edge has an associated weight $w(u, v)$

$$w: E \rightarrow R, \text{ weight function}$$



- Storing the weights of a graph
  - Adjacency list:
    - Store $w(u,v)$ along with vertex v in u's adjacency list
  - Adjacency matrix:
    - Store $w(u, v)$ at location (u, v) in the matrix

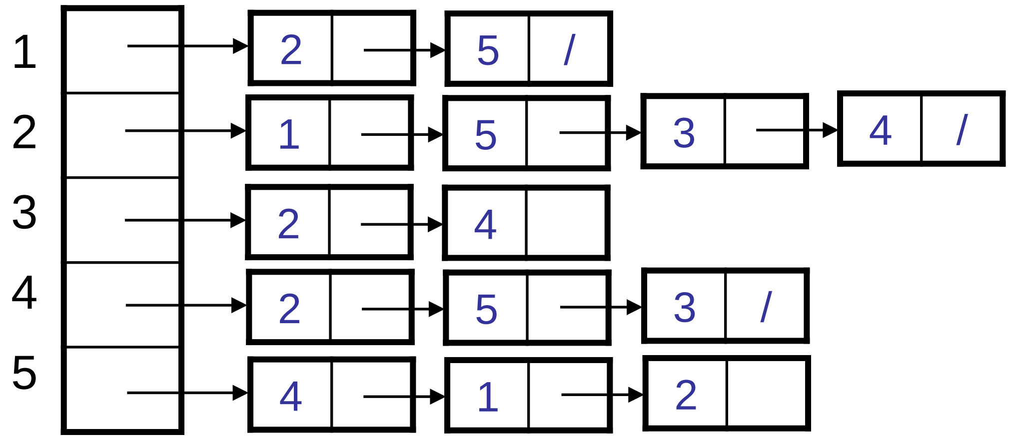
Weighted Graphs



(a) A weighted digraph. (b) Its adjacency matrix.

Problem 1

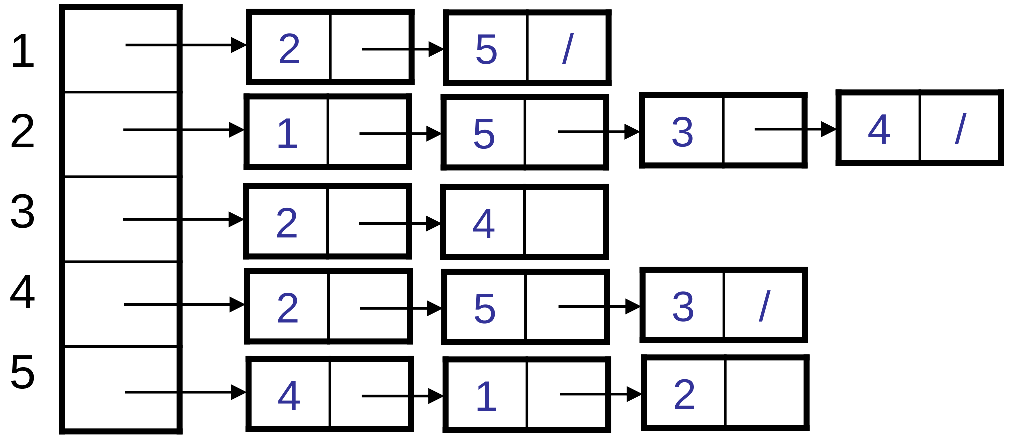
- **(Exercise 22.1-1, page 530)** Given an adjacency-list representation, how long does it take to compute the out-degree of every vertex?
  - For each vertex u, search Adj[u] → Θ(E)



Problem 1

- How long does it take to compute the in-degree of every vertex?
  - For each vertex u,

    search entire list of edges → Θ(VE)



Problem 2

- **(Exercise 22.1-3, page 530)** The transpose of a graph $G=(V,E)$ is the graph $G^T=(V,E^T)$, where $E^T=\{(v,u) \in V \times V: (u,v) \in E\}$. Thus, G^T is G with all edges reversed.

(a) Describe an efficient algorithm for computing G^T from G, both for the adjacency-list and adjacency-matrix representations of G.

(b) Analyze the running time of each algorithm.

Problem 2 (cont'd)

Adjacency matrix

```
for (i=1; i<=V; i++)
  for(j=i+1; j<=V; j++)
    if(A[i][j] && !A[j][i]) {
      A[i][j]=0;
      A[j][i]=1;
    }
```

$O(V^2)$ complexity

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

Problem 2 (cont'd)

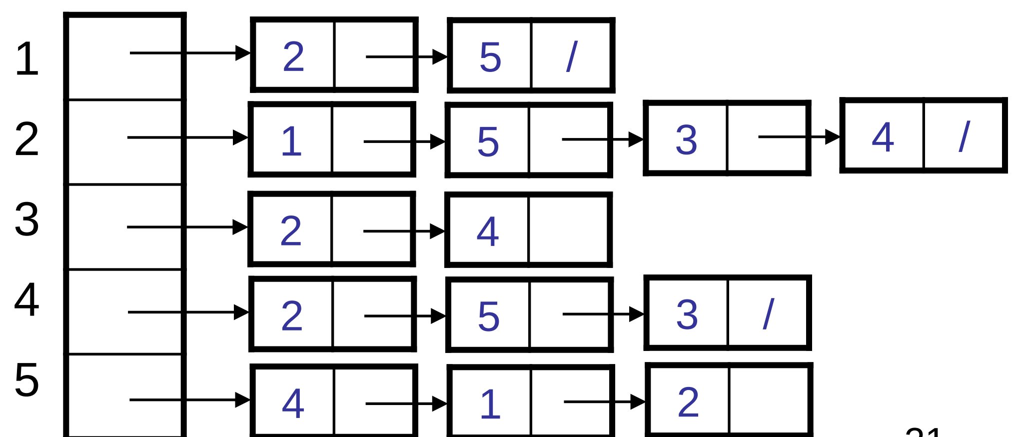
Adjacency list

Allocate V list pointers for G^T (Adj'[]) ← O(V)

for(i=1; i<=V, i++)
    for every vertex v in Adj[i]
        add vertex i to Adj'[v]

← O(E)

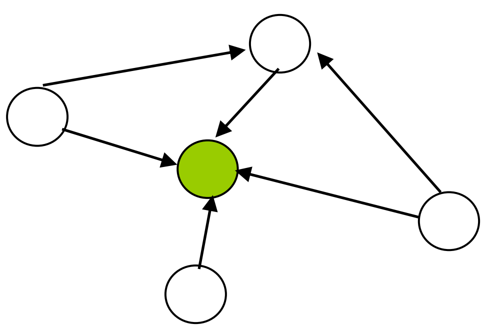
Total time: O(V+E)



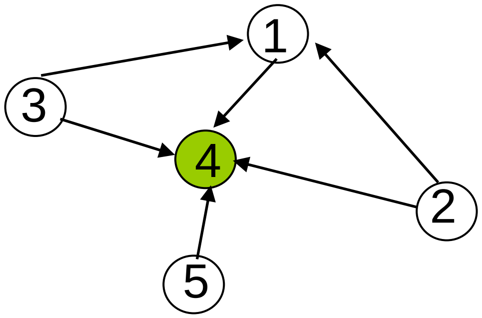
Problem 3

- **(Exercise 22.1-6, page 530)** When adjacency-matrix representation is used, most graph algorithms require time $\Omega(V^2)$, but there are some exceptions. Show that determining whether a directed graph G contains a **universal sink** – a vertex of in-degree |V|-1 and out-degree 0 – an be determined in time O(V).

Example



Problem 3 (cont.)



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

Problem 3 (cont.)

- How many sinks could a graph have?
  - 0 or 1
- How can we determine whether a given vertex u is a universal sink?
  - The u-row must contain 0’s only
  - The u-column must contain 1’s only
  - A[u][u]=0
- How long would it take to determine whether a given vertex u is a universal sink?
  - O(V) time

Problem 3 (cont.)

```
IS-SINK(A, k)
let A be |V| × |V|
for j ← 1 to |V|                  ▷ Check for a 1 in row k
     do if a_kj = 1
           then return FALSE
for i ← 1 to |V|                  ▷ Check for an off-diagonal 0 in column k
     do if a_ik = 0 and i ≠ k
           then return FALSE
return TRUE
```

- How long would it take to determine whether a given graph contains a universal sink if you were to check every single vertex in the graph?

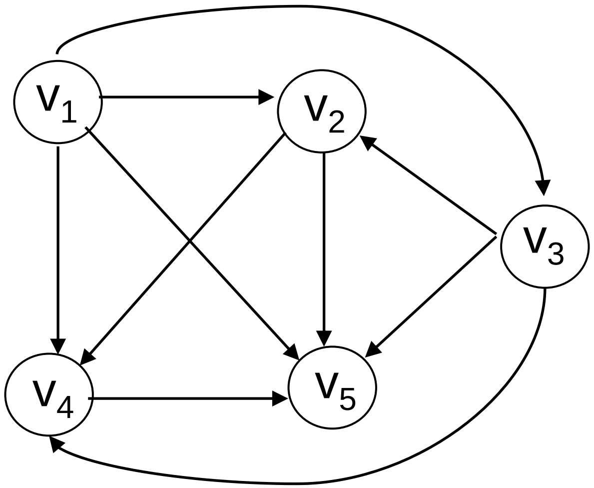
  - $O(V^2)$

Problem 3 (cont.)

- Can you come up with a O(V) algorithm?
- Observations
  - If A[u][v]=1, then u cannot be a universal sink
  - If A[u][v]=0, then v cannot be a universal sink

UNIVERSAL-SINK(A)
let A be $|V| \times |V|$
$i \leftarrow j \leftarrow 1$
while $i \leq |V|$ and $j \leq |V|$
  do if $a_{ij} = 1$
    then $i \leftarrow i + 1$
    else $j \leftarrow j + 1$
$s \leftarrow 0$
if $i > |V|$
  then return "there is no universal sink"
elseif IS-SINK(A, i) = FALSE
  then return "there is no universal sink"
else return i "is a universal sink"

Problem 3 (cont.)



$$\begin{array}{c|c}
 & \begin{array}{ccccc} \mathbf{v_1} & \mathbf{v_2} & \mathbf{v_3} & \mathbf{v_4} & \mathbf{v_5} \end{array} \\
\begin{array}{c} \mathbf{v_1} \\ \mathbf{v_2} \\ \mathbf{v_3} \\ \mathbf{v_4} \\ \mathbf{v_5} \end{array} & \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}
\end{array}$$

- Loop terminates when i>|V| or j>|V|
- Upon termination, the only vertex that could be a sink is i
  - If i > |V|, there is no sink
  - If i < |V|, then j>|V|
    * vertices k where 1 ≤ k < i can not be sinks
    * vertices k where i < k ≤ |V| can not be sinks