Announcements

- Weekly Reading Assignment:

  Chapter 22

- Homework #5 due on Nov. 17, 2005

Graphs

- A collection of *vertices* or *nodes*, connected by a collection of *edges*.
- Applicable to many applications where there is some “connection” or “relationship” or “interaction” between pairs of objects
  - network communication & transportation
  - VLSI design & logic circuit design
  - surface meshes in CAD/CAM & GIS
  - path planning for autonomous agents
  - precedence constraints in scheduling

Basic Definitions

- ***Directed Graph*** **(or *digraph*)** $G = (V, E)$ **consists of a finite set** V**, called vertices or nodes, and** E**, a set of *ordered* pairs, called edges of** G**.** E **is a binary relation on** V**. Self-loops are allowed. Multiple edges are not allowed, though** (v, w) **and** (w, v) **are distinct edges.**

- ***Undirected Graph*** **(or *graph*)** $G = (V, E)$ **consists of a finite set** V **of vertices, and a set** E **of *unordered* pairs of distinct vertices, called edges of** G**. No self-loops are allowed.**

Examples of Digraphs & Graphs

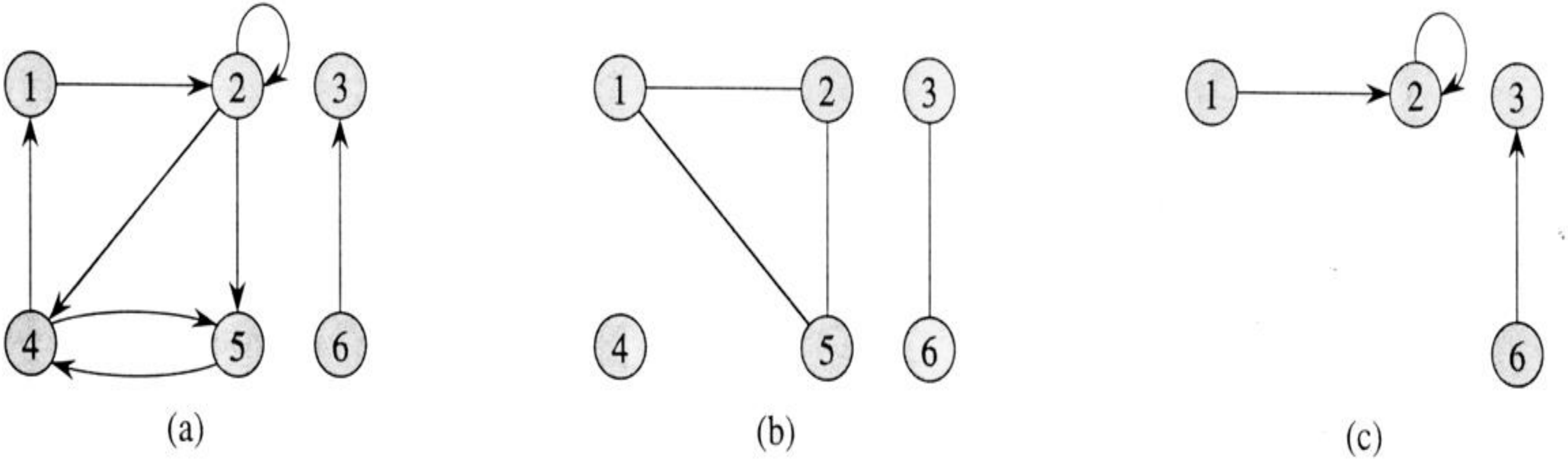


Figure 5.2 Directed and undirected graphs. **(a)** A directed graph $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{(1,2), (2,2), (2,4), (2,5), (4,1), (4,5), (5,4), (6,3)\}$. The edge $(2,2)$ is a self-loop. **(b)** An undirected graph $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{(1,2), (1,5), (2,5), (3,6)\}$. The vertex 4 is isolated. **(c)** The subgraph of the graph in part (a) induced by the vertex set $\{1, 2, 3, 6\}$.

Definitions

- Vertex w is ***adjacent*** to vertex v if there is an edge (v,w). Given an edge $e = (u,v)$ in an undirected graph, u and v are the endpoints of e and e is ***incident*** on u (or on v). In a digraph, u & v are the ***origin*** and ***destination***. e leaves u and enters v.
- A digraph or graph is ***weighted*** if its edges are labeled with numeric values.
- In a digraph,
  - ***Out-degree*** of v: number of edges coming out of v
  - ***In-degree*** of v: number of edges coming in to v
- In a graph, ***degree*** of v: no. of incident edges to v

Combinatorial Facts

- In a graph
  - $0 \leq e \leq C(n,2) = n\,(n\text{-}1)\,/\,2 \in O(n^2)$
  - $\sum_{v \in V} \deg(v) = 2e$
- In a digraph
  - $0 \leq e \leq n^2$
  - $\sum_{v \in V} \text{in-deg}(v) = \sum_{v \in V} \text{out-deg}(v) = e$

A graph is said to be ***sparse*** if $e \in O(n)$, and ***dense*** otherwise.

Definitions (Path vs. Cycle)

- ***Path***: a sequence of vertices $<v_0, \ldots, v_k>$ s.t. (v_{i-1}, v_i) is an edge for $i = 1$ *to* k, in a digraph. The ***length*** of the path is the number of edges, k.
- w is ***reachable*** from u if there is a path from u to w. A path is ***simple*** if all vertices are distinct.
- ***Cycle***: a path containing at least 1 edge and for which $v_0 = v_k$, in a digraph. A cycle is ***simple*** if, in addition, all vertices are distinct.
- For *graphs*, the definitions are the same, but a ***simple cycle*** must visit $\geq$ 3 distinct vertices.

History on Cycles/Paths

- ***Eulerian cycle*** **is a cycle (not necessarily simple) that visits every edge of a graph exactly once.**
- ***Hamiltonian cycle (path)*** **is a cycle (path in a directed graph) that visits every vertex exactly once.**

Definitions (Connectivity)

- ***Acyclic***: if a graph contains no simple cycles
- ***Connected***: if every vertex of a graph can reach every other vertex
- ***Connected***: every pair of vertices is connected by a path
- ***Connected Components***: equivalence classes of vertices under "is reachable from" relation
- ***Strongly connected***: for any 2 vertices, they can reach each other in a digraph
- $G = (V, E)$ & $G' = (V', E')$ are ***isomorphic***, if $\exists$ a bijection $f : V \rightarrow V'$ s.t. $v, u \in E$ iff $(f(v), f(u)) \in E'$.

Examples for Isomorphic Graphs

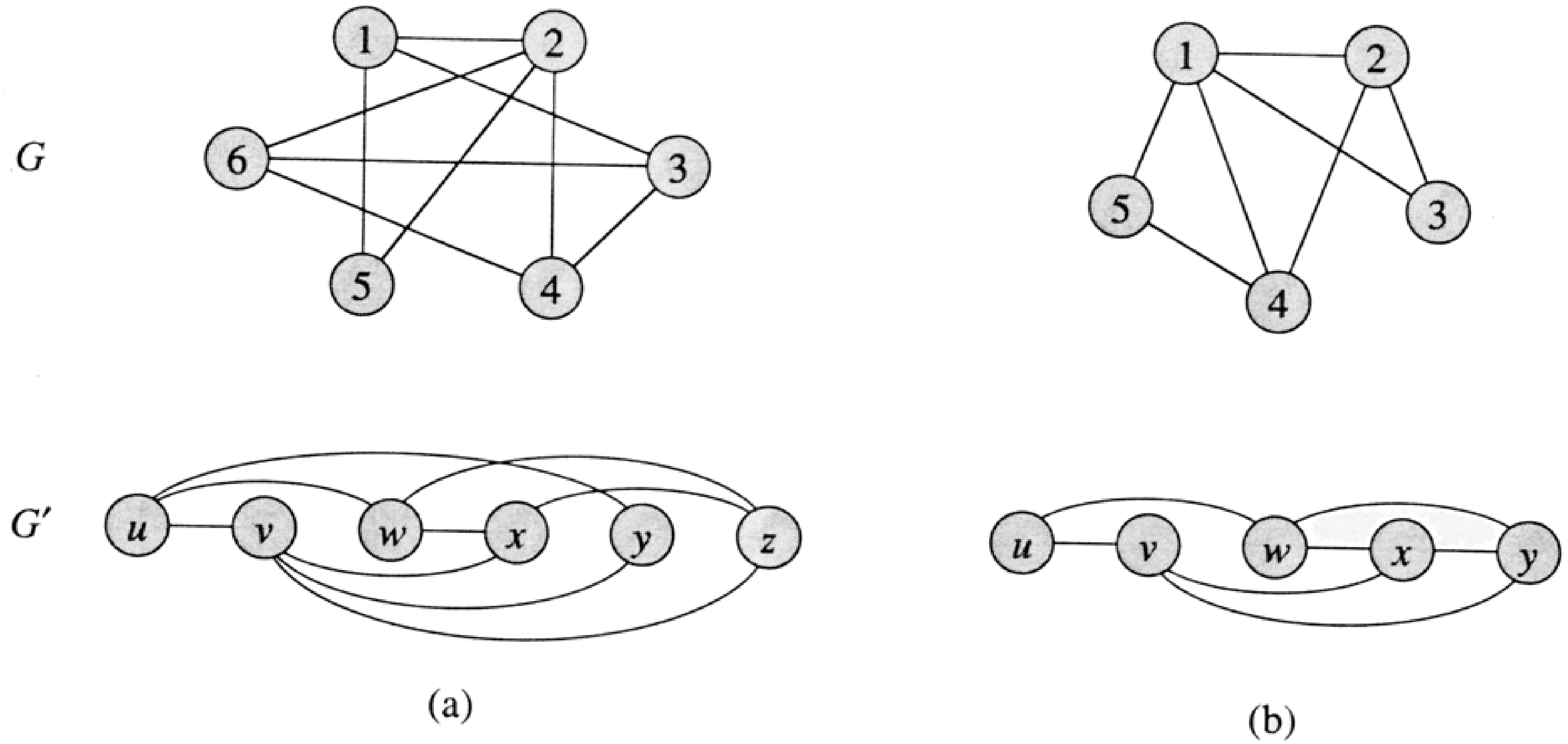
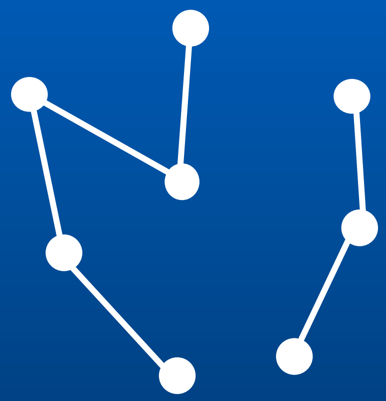


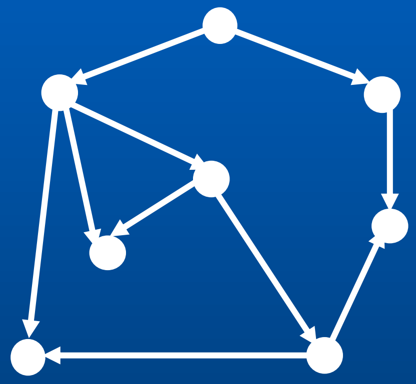
Figure 5.3 **(a)** A pair of isomorphic graphs. The vertices of the top graph are mapped to the vertices of the bottom graph by $f(1) = u$, $f(2) = v$, $f(3) = w$, $f(4) = x$, $f(5) = y$, $f(6) = z$. **(b)** Two graphs that are not isomorphic, since the top graph has a vertex of degree 4 and the bottom graph does not.

Free Trees, Forests, and DAG's

Free Tree



Forest



DAG

Graph Representations

Let $G = (V, E)$ be a digraph with $n = |V|$ & $e = |E|$

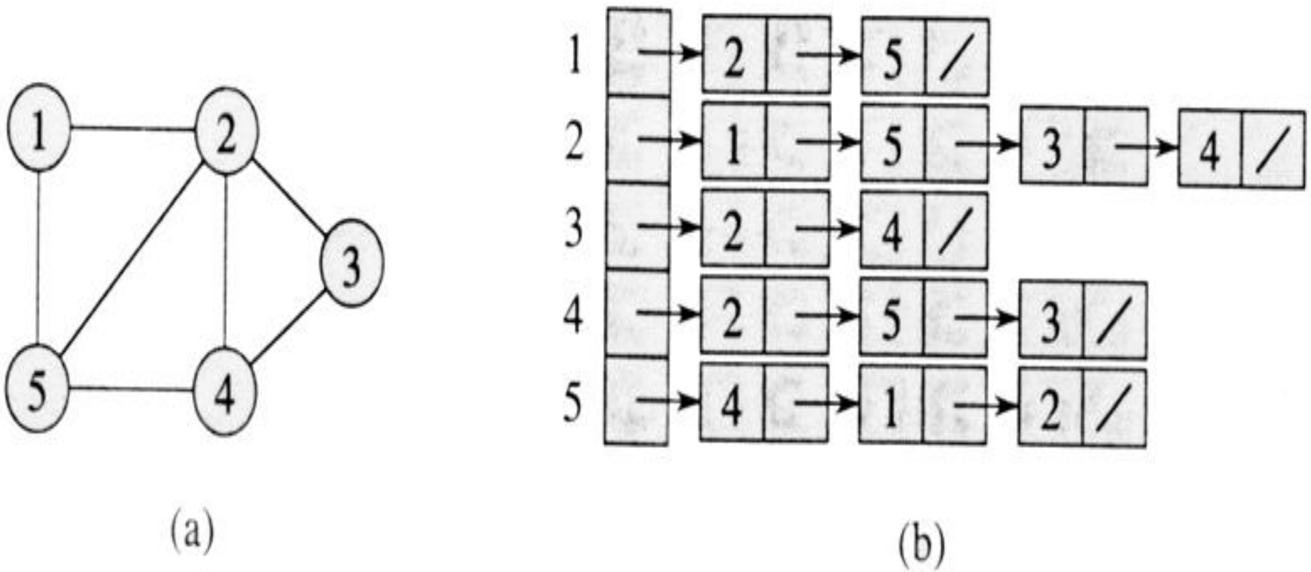
- ***Adjacency Matrix***: a $n \times n$ matrix for $1 \leq v,w \leq n$

  $$A[v,w] = 1 \text{ if } (v,w) \in E \text{ and } 0 \text{ otherwise}$$

  If digraph has weights, store them in matrix.

- ***Adjacency List***: an array $Adj[1 \ldots n]$ of pointers where for $1 \leq v \leq n$, $Adj[v]$ points to a linked list containing the vertices which are adjacent to v. If the edges have weights then they may also be stored in the linked list elements.

Example for Graphs



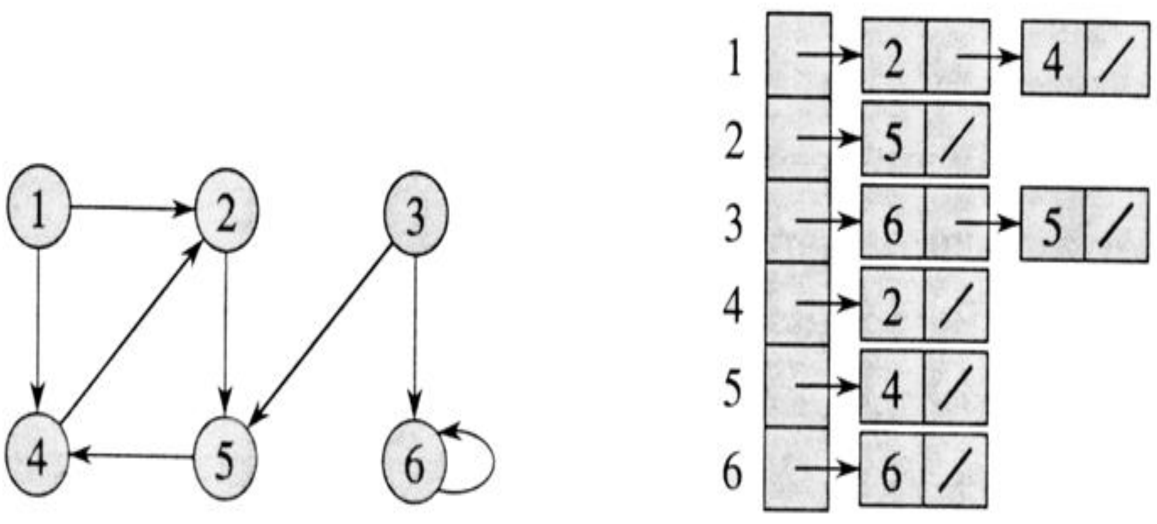
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)

Figure 23.1 Two representations of an undirected graph. **(a)** An undirected graph G having five vertices and seven edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

NOTE: it is common to include cross links between corresponding edges, when you need to mark the edges you visit before. E.g. $(v,w) = (w,v)$

Example for Digraphs



(a) (b)

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

(c)

Figure 23.2 Two representations of a directed graph. **(a)** A directed graph G having six vertices and eight edges. **(b)** An adjacency-list representation of G. **(c)** The adjacency-matrix representation of G.

Finding Shortest Paths

- Given an undirected graph and source vertex s, the length of a path in a graph (without edge weights) is the number of edges on the path. Find the shortest path from s to each other vertex in the graph.

- Brute-Force: enumerate all simple paths starting from s and keep track of the shortest path arriving at each vertex. There may be $n!$ simple paths in a graph…

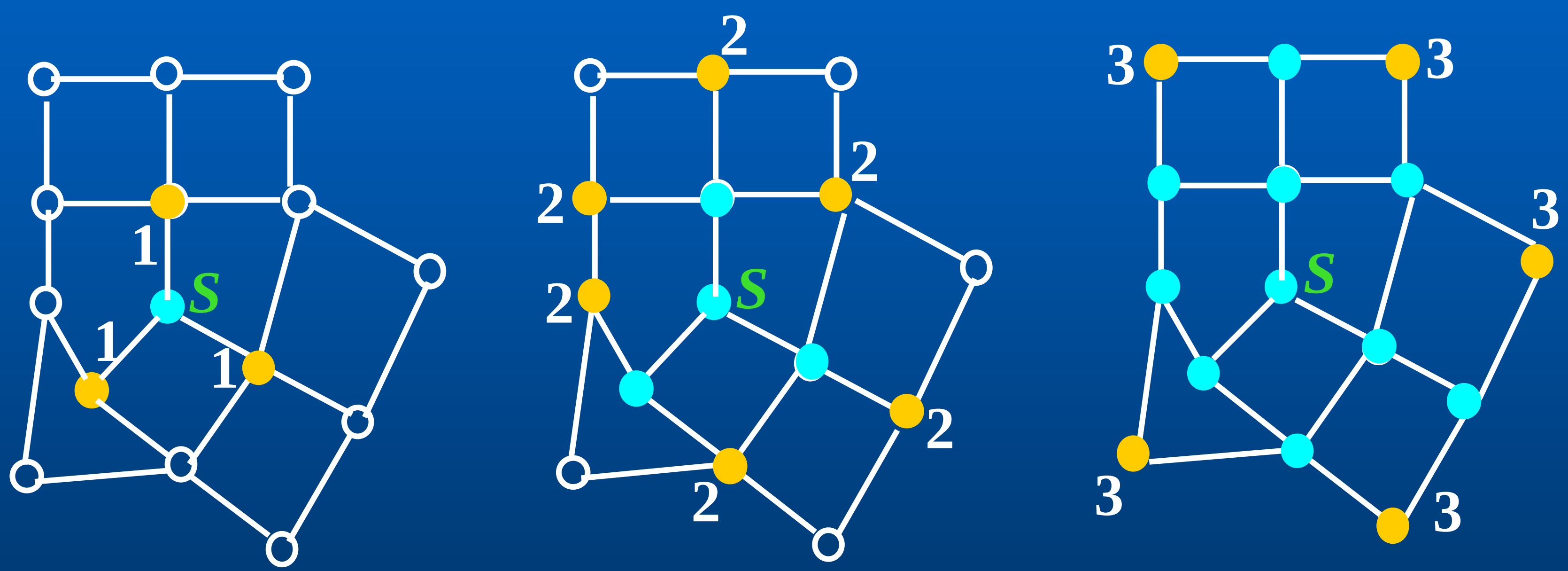
Breadth-First-Search (BFS)

- **Given:**
  - $G = (V, E)$
  - **A distinguished *source* vertex**
- **Systematically explores the edges of G to discover every vertex that is reachable from s**
  - **Computes (shortest) distance from s to all reachable vertices**
  - **Produces a *breadth-first-tree* with root s that contains all reachable vertices**

Breadth-First-Search (BFS)

- BFS colors each vertex:
  - white -- undiscovered
  - gray -- discovered but “not done yet”
  - black -- all adjacent vertices have been discovered

BFS for Shortest Paths



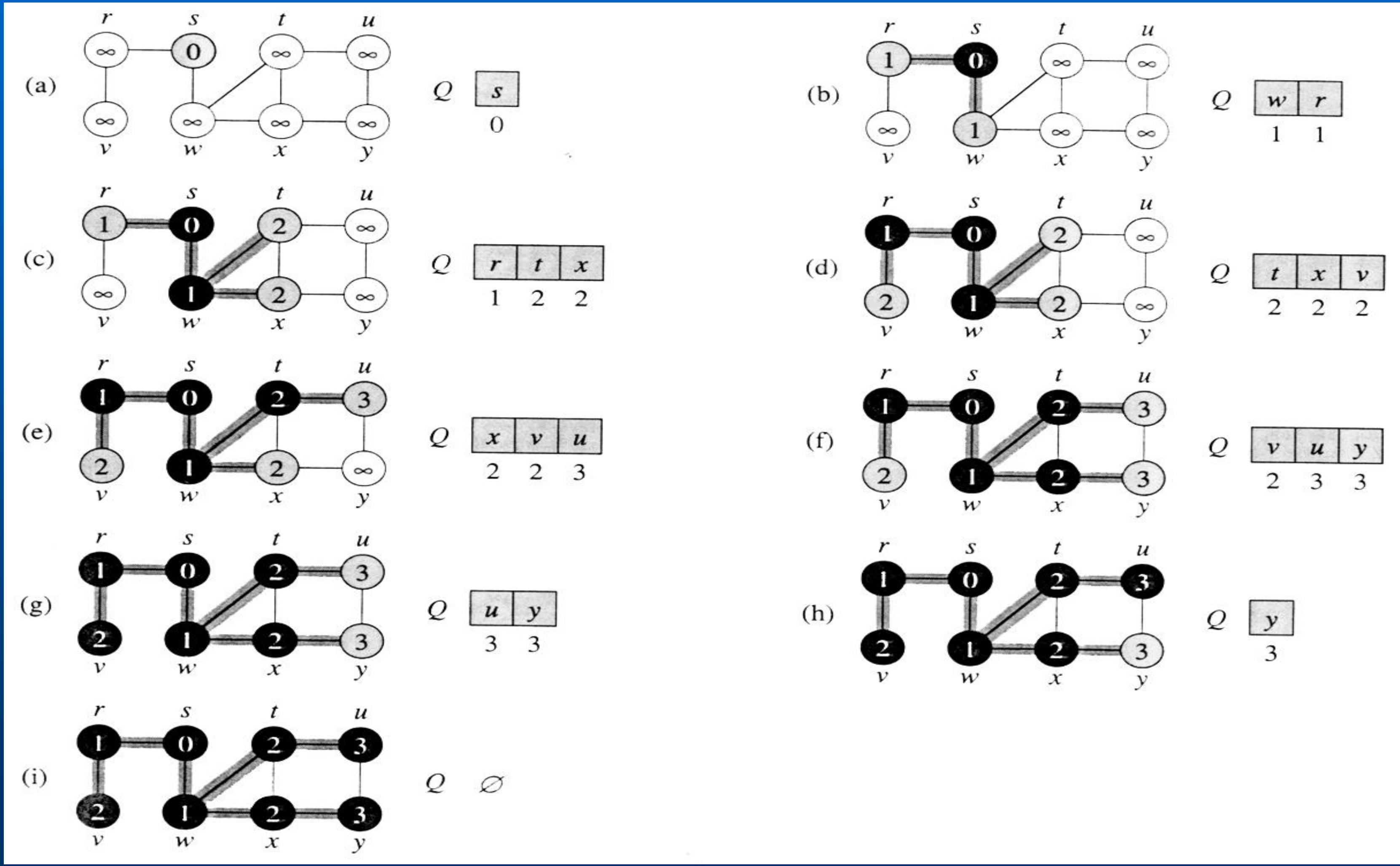
● Finished ● Discovered ○ Undiscovered

```
BFS(G,s)
    for each vertex u in (V[G] \ {s})
            do color[u] ← white
               d[u] ← ∝
               π[u] ← nil
    color[s] ← gray
    d[s] ← 0
    π[s] ← nil
    Q ← Φ
    enqueue(Q,s)
    while Q ≠ Φ
            do u ← dequeue(Q)
                    for each v in Adj[u]
                            do if color[v] = white
                                    then color[v] ← gray
                                         d[v] ← d[u] + 1
                                         π[v] ← u
                                         enqueue(Q,v)
                    color[u] ← black
```

white: undiscovered
gray: discovered
black: finished

Q: a queue of discovered vertices
color[v]: color of v
d[v]: distance from s to v
π[u]: predecessor of v

Operations of BFS on a Graph



Breadth-First Tree

- For a graph $G = (V, E)$ with source s, the *predecessor subgraph* of G is $G_\pi = (V_\pi, E_\pi)$ where
  - $V_\pi = \{v \in V : \pi[v] \neq NIL\} \cup \{s\}$
  - $E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$
- The predecessor subgraph G_π is a *breadth-first tree* if:
  - V_π consists of the vertices reachable from s and
  - for all $v \in V_\pi$, there is a unique simple path from s to v in G_π that is also a shortest path from s to v in G.
- The edges in E_π are called *tree edges*.
  $|E_\pi| = |V_\pi| - 1$

Intuition: Breadth-First Tree

- The predecessor pointers of the BFS define an inverted tree (an acyclic directed graph in which the source is the root, and every other node has a unique path to the root). If we make these edges bidirectional we get a rooted unordered tree called a BFS tree for G.

- There are potentially many BFS trees for a given graph, depending on where the search starts and in what order vertices are placed on the queue. These edges of G are called tree edges and the remaining edges of G are called cross edges.

Analysis of BFS

- **Initialization takes $O(V)$.**
- **Traversal Loop**
  - **After initialization, each vertex is enqueued and dequeued at most once, and each operation takes $O(1)$. So, total time for queuing is $O(V)$.**
  - **The adjacency list of each vertex is scanned at most once. The sum of lengths of all adjacency lists is $\Theta(E)$.**
- **Summing up over all vertices => total running time of BFS is $O(V+E)$, linear in the size of the adjacency list representation of graph.**

Shortest Paths

- *Shortest-Path distance* $\delta(s, v)$ from s to v is the minimum number of edges in any path from vertex s to vertex v, or else ∞ if there is no path from s to v.
- A path of length $\delta(s, v)$ from s to v is said to be a *shortest path* from s to v.

Lemmas

- Let $G = (V, E)$ be a directed or undirected graph, and let $s \in V$ be an arbitrary vertex. Then, for any edge $(u,v) \in E$, $\delta(s, v) \leq \delta(s, u) + 1$.
- Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$. Then upon termination, for each vertex $v \in V$, the value $d[v]$ computed by BFS satisfies $d[v] \geq \delta(s, v)$.
- Suppose that during the execution of BFS on a graph G, the queue Q contains vertices $(v_1, \ldots, v_r)$, where v_1 is the head of Q and v_r is the tail. Then, $d[v_r] \leq d[v_1] + 1$ and $d[v_i] \leq d[v_{i+1}]$ for $i = 1, 2, \ldots, r-1$.

Correctness of BFS

- Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$. Then, during its execution, BFS discovers every vertex $v \in V$ that is reachable from the source s, and upon termination, $d[v] = \delta(s, v)$ for all $v \in V$. Moreover, for any vertex $v \neq s$ that is reachable from s, one of the shortest paths from s to v is the shortest path from s to $\pi[v]$ followed by the edge $(\pi[v], v)$.

Depth-First-Search (DFS)

- **Explore edges out of the most recently discovered vertex v**
- **When all edges of v have been explored, backtrack to explore edges leaving the vertex from which v was discovered (its *predecessor*)**
- **"Search as deep as possible first"**
- **Whenever a vertex v is discovered during a scan of the adjacency list of an already discovered vertex u, DFS records this event by setting predecessor $\pi[v]$ to u.**

Depth-First Trees

- Coloring scheme is the same as BFS. The predecessor subgraph of DFS is $G_\pi = (V, E_\pi)$ where $E_\pi = \{(\pi[v], v) : v \in V$ and $\pi[v] \neq NIL\}$. The predecessor subgraph G_π forms a ***depth-first forest*** composed of several ***depth-first trees***. The edges in E_π are called ***tree edges***.

- Each vertex u has 2 ***timestamps***: $d[u]$ records when u is first discovered (grayed) and $f[u]$ records when the search finishes (blackens). For every vertex u, $d[u] < f[u]$.

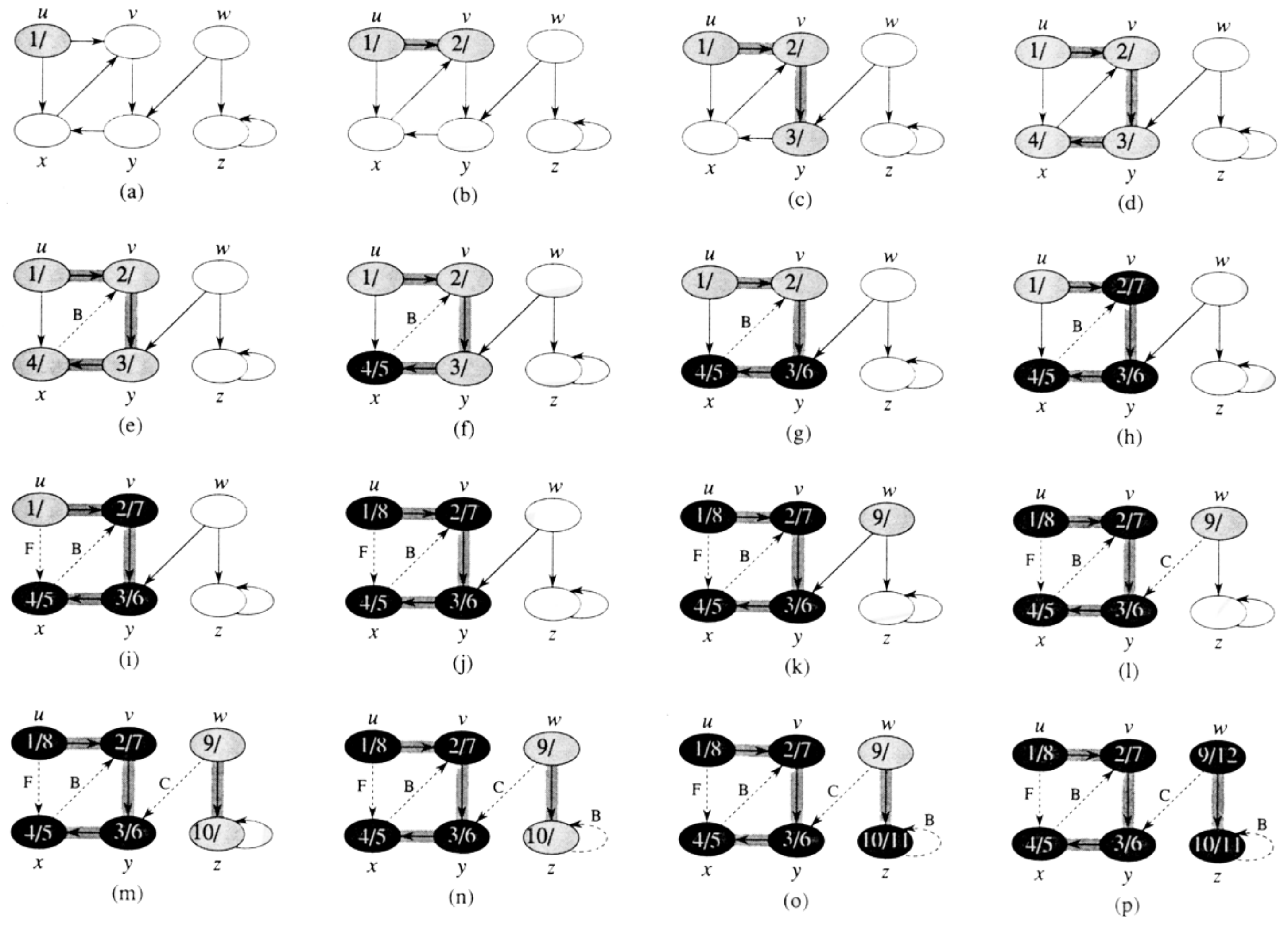
DFS(G)

for each vertex $u \in V[G]$
    do $color[u] \leftarrow$ WHITE
        $\pi[u] \leftarrow$ NIL
$time \leftarrow 0$
for each vertex $u \in V[G]$
    do if $color[v] =$ WHITE
        then DFS-Visit(v)

DFS-Visit(u)

1. $color[u] \leftarrow$ GRAY
   ∇ White vertex u has been discovered
2. $d[u] \leftarrow$ ++$time$
3. for each vertex $v \in Adj[u]$
4. do if $color[v] = \text{WHITE}$
5. then $\pi[v] \leftarrow u$
6. DFS-Visit(v)
7. $color[u] \leftarrow$ BLACK
   ∇ Blacken u; it is finished.
8. $f[u] \leftarrow time$++

Operations of DFS



Analysis of DFS

- Loops on lines 1-2 & 5-7 take $\Theta(V)$ time, excluding time to execute DFS-Visit.
- DFS-Visit is called once for each white vertex $v \in V$ when it's painted gray the first time. Lines 3-6 of DFS-Visit is executed $|\mathrm{Adj}[v]|$ times. The total cost of executing DFS-Visit is $\sum_{v \in V} |\mathrm{Adj}[v]| = \Theta(E)$
- Total running time of DFS is $\Theta(V+E)$.