算法基础
Foundation of Algorithms

主讲人 徐云

Fall 2018, USTC

Part 1 Foundation

Part 2 Sorting and Order Statistics

Part 3 Data Structure

Part 4 Advanced Design and Analysis Techniques

Part 5 Advanced Data Structures

Part 6 Graph Algorithms

- chap 22 Elementary Graph Algorithms
- chap 23 Minimum Spanning Trees
- chap 24 Single-Source Shortest Paths
- chap 25 All-Pairs Shortest Paths

Part 7 Selected Topics

Part 8 Supplement

Part 6 Graph Algorithms

22 Elementary Graph Algorithms

23 Minimum Spanning Trees

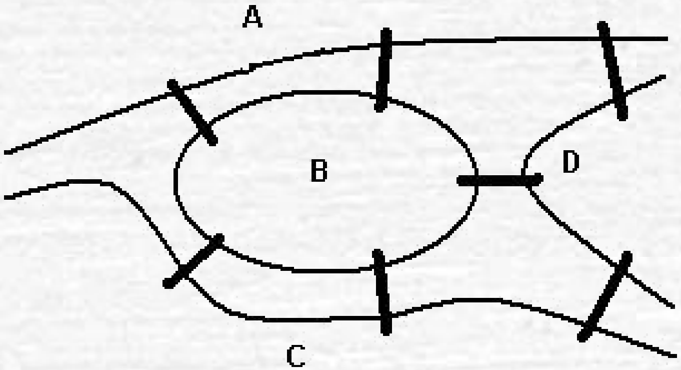
24 Single-Source Shortest Paths

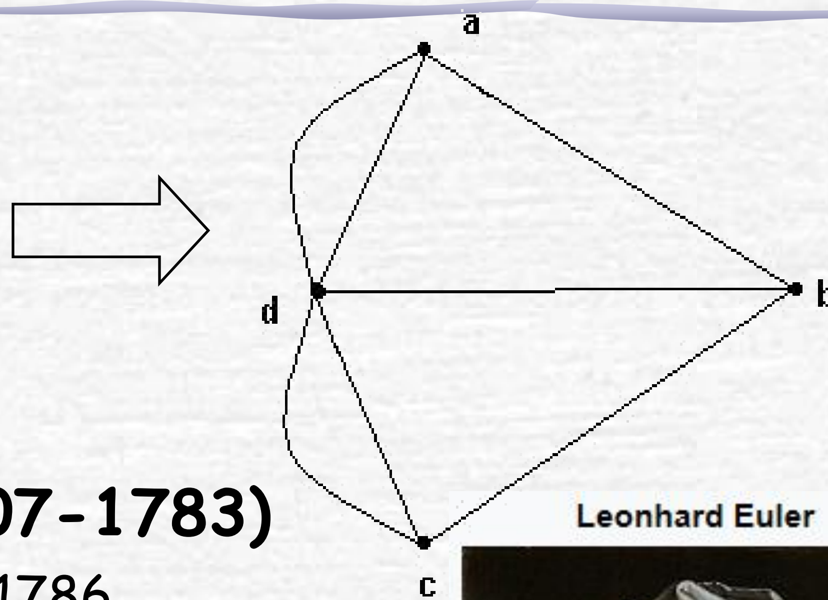
25 All-Pairs Shortest Paths

22 Elementary Graph Algorithms

- Background and History
- Graph Foundations
- Breadth-first Search (BFS)
- Depth-first Search (DFS)

Background and History (1)

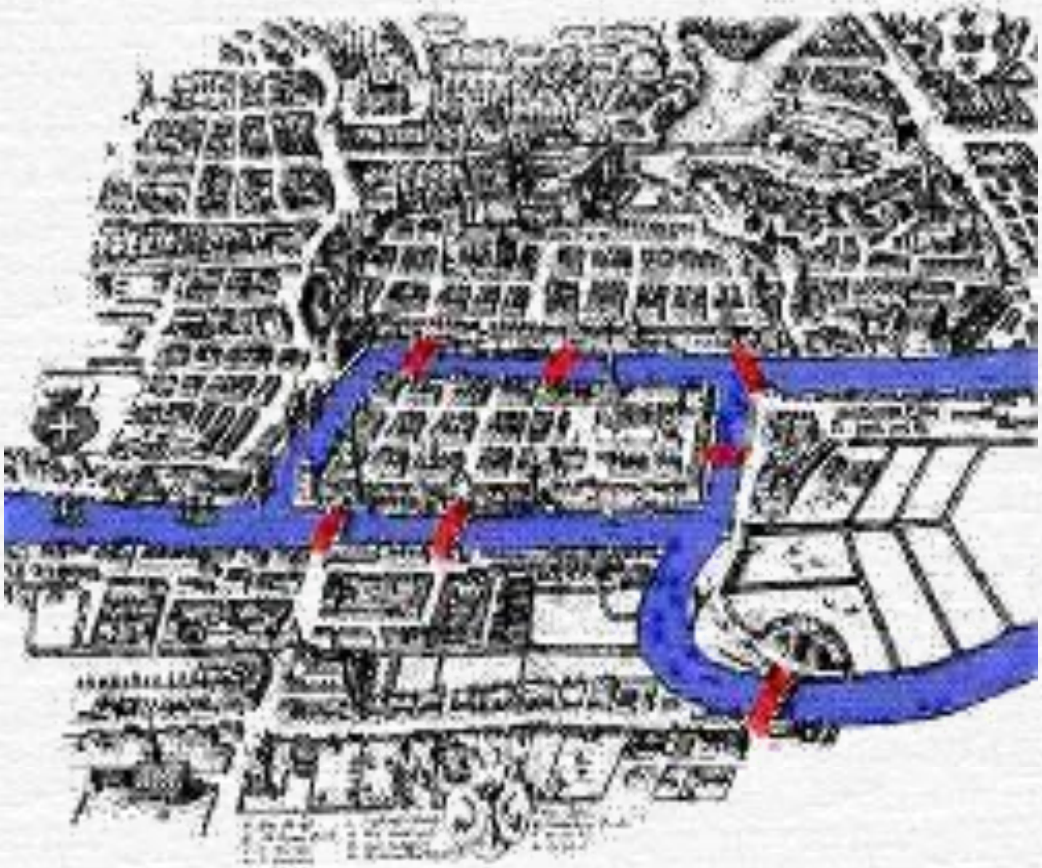




- 1736: **Leonhard Euler(1707-1783)**
  - Basel, 1707-St. Petersburg, 1786.
  - He wrote *A solution to a problem concerning the geometry of a place.* First paper in graph theory.
- Problem of the Königsberg bridges:
  - Starting and ending at the same point, is it possible to cross all seven bridges just once and return to the starting point?

Leonhard Euler

Background and History（2）



The map of Konigsberg in the eighteenth century, showing the river and the seven bridges that inspired Euler to introduce the first graph, creating graph theory.

Graph Definitions (1)

- Graph $G = (V, E)$
  - V = set of vertices; E = set of edges $\subseteq (V \times V)$;
- Types of graphs
  - Undirected: edge $(u, v) = (v, u)$; for all v, $(v, v) \notin E$ (No self loops.)
  - Directed: (u, v) is edge from u to v, denoted as $u \rightarrow v$. Self loops are allowed.
  - Weighted: each edge has an associated weight, given by a weight function $w: E \rightarrow \mathbf{R}$.
  - Dense: $|E| \approx |V|^2$.
  - Sparse: $|E| \ll |V|^2$.
- $|E| = O(|V|^2)$

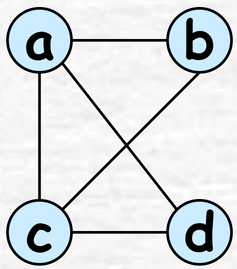
Graph Definitions (2)

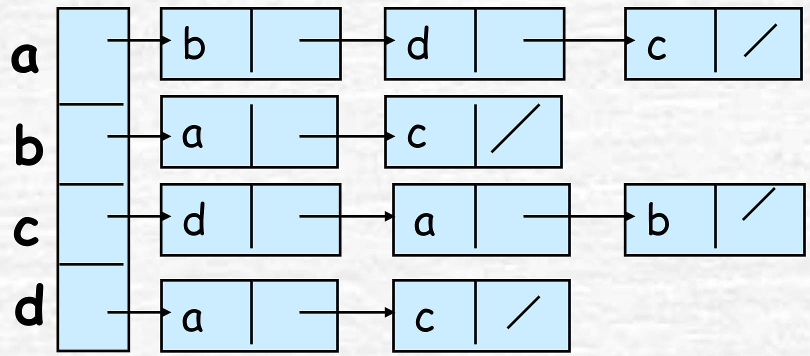
- If $(u, v) \in E$, then vertex v is adjacent to vertex u.
- Adjacency relationship is:
  - Symmetric if G is undirected.
  - Not necessarily so if G is directed.
- If G is connected:
  - There is a path between every pair of vertices.
  - $|E| \geq |V| - 1$.
  - Furthermore, if $|E| = |V| - 1$, then G is a tree.

- Other definitions in Appendix B (B.4 and B.5) as needed.

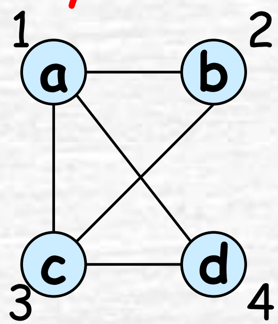
Representation of Graphs

- Two standard ways:
  - Adjacency Lists.





  - Adjacency Matrix.



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

- Their advantages and disadvantages.

Graph Traversals

- Some applications involving graph G:
  - Is G connected?
  - Does G contain a cycle?
  - Is G a tree?
  - Is G bipartite?
  - Find connected components.
  - Topological sorting
  - Is directed G strongly connected?

Breadth-first Search (BFS)

- *Goal*
  - *Systematically explores* the edges of G to *visit each node* of G reachable from s. And,
  - based on all vertices at distance k from s, *discovers any vertices at distance k + 1* from s.

- *Basic idea*
  - Use a *First-In-First-Out (FIFO) queue* to implement the *frontier (前沿点)* or gray nodes later.
  - Expand the *frontier* between already discovered and undiscovered vertices one step at a time.

- *Thinking:* which one is better?
  - $O(|V|+|E|)$ time via adjacency list, and $O(|V|^2)$ via adjacency matrix, generally.
  - Depend on the density/sparseness of the graph.

BFS: Method

- **Input:** Graph $G = (V, E)$, either directed or undirected, and *source vertex* $s \in V$.
- **Output:**
  - Builds *breadth-first tree* with root s that contains all reachable vertices. And,
  - $d[v]$ = distance (shortest, or pathsmallest # of edges) from s to v, for all $v \in V$. $d[v] = \infty$ if v is not reachable from s.
  - $\pi[v] = u$ such that (u, v) is last edge on shortest path $s \rightsquigarrow v$, which u is v's predecessor.

Definitions:

- Path between vertices u and v: Sequence of vertices $(v_1, v_2, \ldots, v_k)$ such that $u = v_1$ and $v = v_k$, and $(v_i, v_{i+1}) \in E$, for all $1 \leq i \leq k-1$.
- Length of the path: Number of edges in the path.

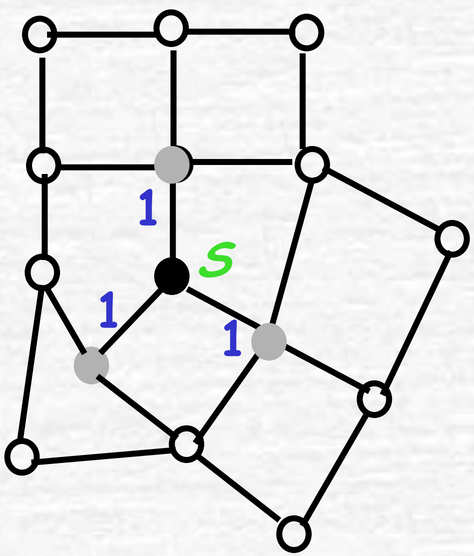
Path is simple if no vertex is repeated.

BFS: Coloring the Nodes

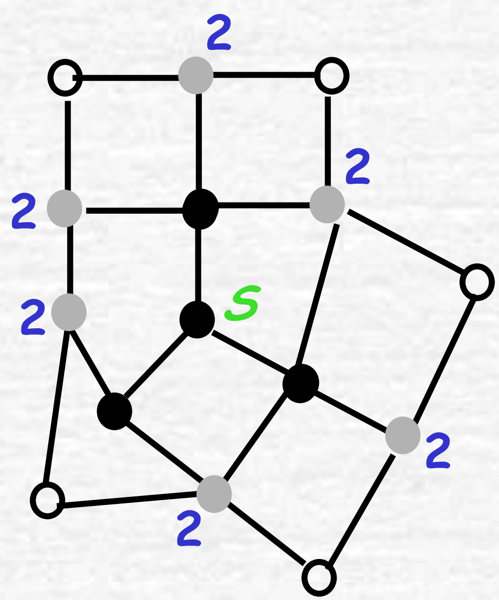
- To ease illustration, we use colors (white, gray and black) to denote the state of the node during the search.
  - *White* – Undiscovered.
  - *Gray* – Discovered but not finished.
  - *Black* – Finished.
- All nodes change color in order: white → gray →black.

BFS for Shortest Paths

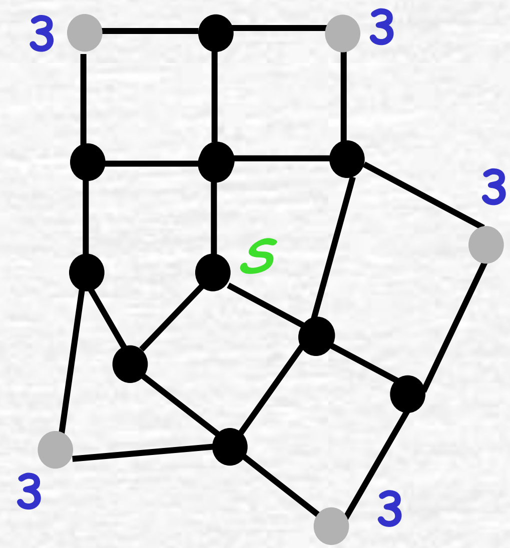
First iteration



Second iteration



Third iteration



● Finished   ● Discovered   ○ Undiscovered

Algorithm of BFS

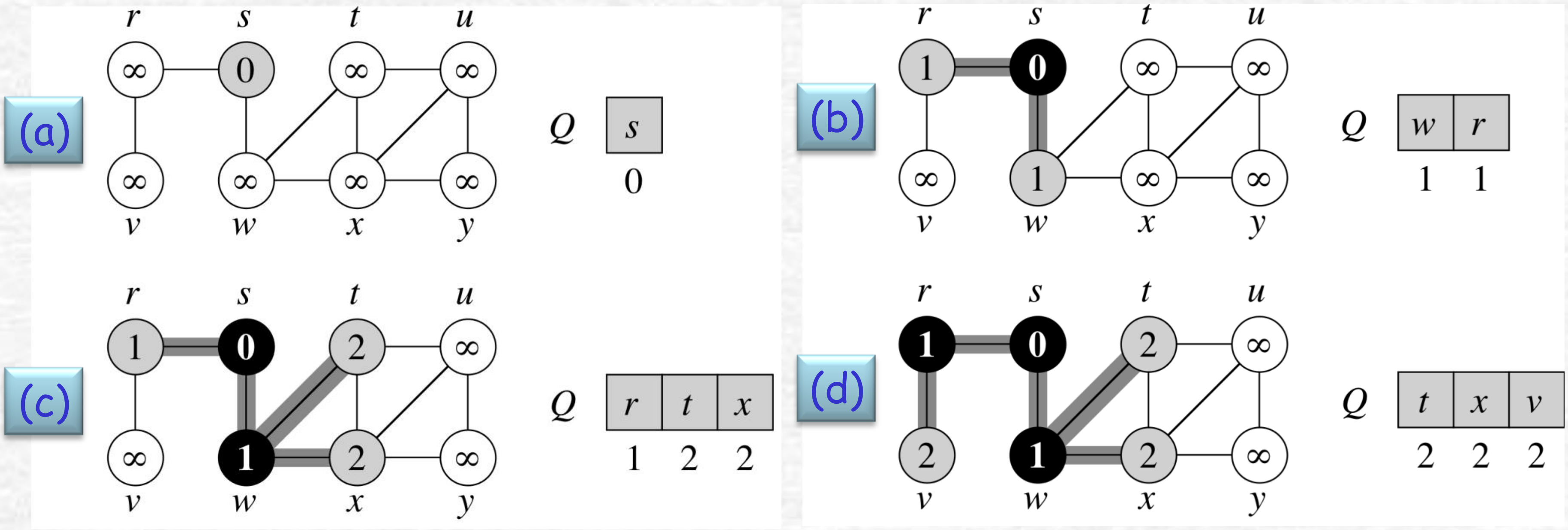
BFS(G,s)

```
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

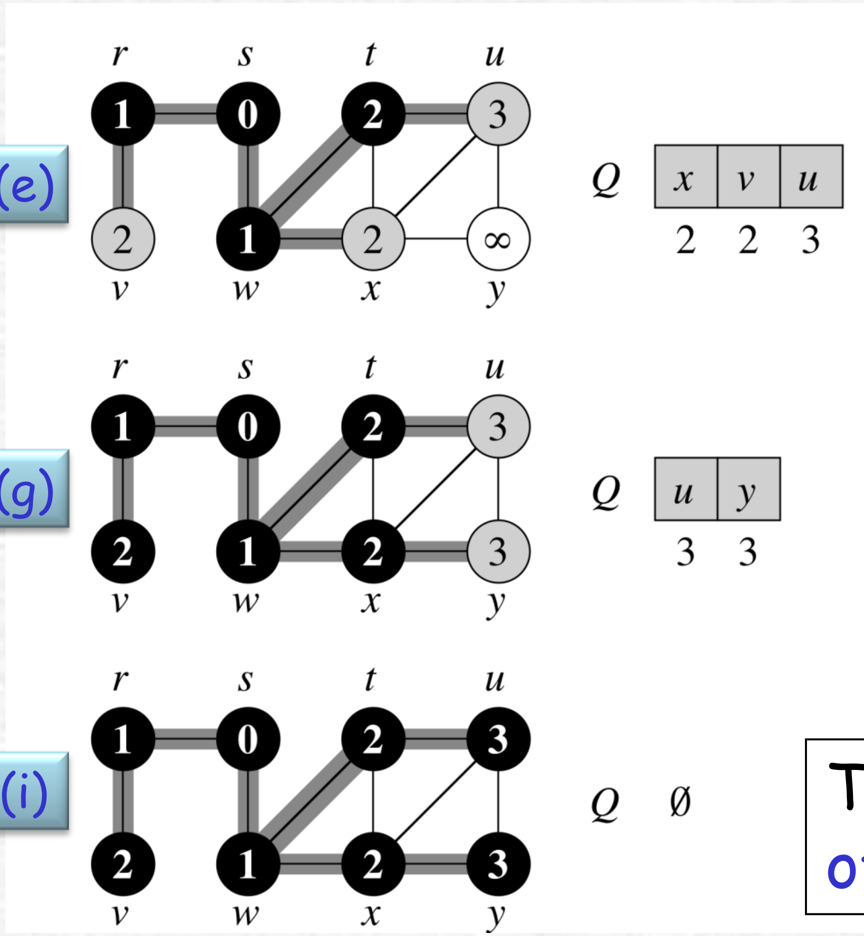
Q: a queue of discovered vertexes
color[*v*]: color of v
d[*v*]: distance from s to v
π[*u*]: predecessor of v

BFS: Example (1)

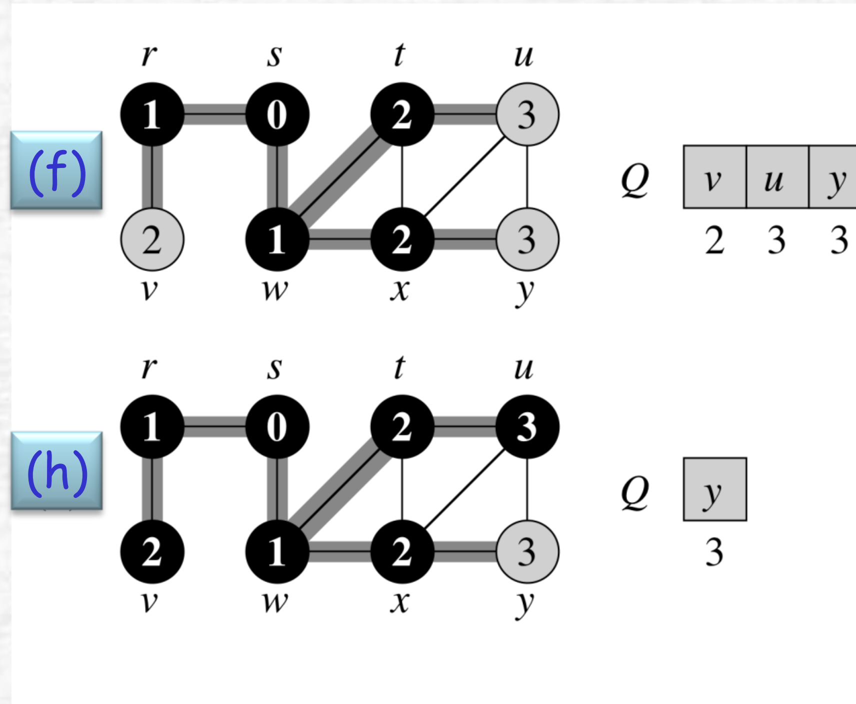


BFS: Example (2)

(e) (g) (i)



(f) (h)



The shaded edges are edges of the breadth first tree.

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.
- Correctness Proof
  - We omit for BFS and DFS later.

BFS Tree

- For a graph $G = (V, E)$ with source s, the *predecessor subgraph* of G is $G_\pi = (V_\pi, E_\pi)$ where
  - $V_\pi = \{ v \in V : \pi[v] \neq \text{NIL} \} \cup \{s\}$
  - $E_\pi = \{ (\pi[v], v) \in E : v \in V_\pi - \{s\} \}$
- 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*.
  $|E_\pi| = |V_\pi| - 1$.

Depth-first Search (DFS)

- *Goal*
  - *Systematically explore* every vertex and edge of G.
  - Go "deeper" whenever possible.

- *Method: Until there are no more undiscovered nodes*
  - *Pick an undiscovered node* and *start a depth first search* from it.
  - The search proceeds *from the most recently discovered node to discover new nodes*.
  - *When the last discovered node v is fully explored*, *backtrack* to the node used to discover v. Eventually, the start node is fully explored.

- *Remark*
  - Don't require a pre-specified source node in textbook.
  - Output varies depending on the nodes order.

DFS: Method

- Input: $G = (V, E)$, directed or undirected. No source vertex given!
- Output:
  - 2 timestamps on each vertex. Integers between 1 and $2|V|$.
    - *d[v] = discovery time* (v turns from white to gray)
    - *f[v] = finishing time* (v turns from gray to black)
  - $\pi[v]$: predecessor of v is $\pi[v]$, such that v was discovered during the scan of $\pi[v]$'s adjacency list.
- Uses the same coloring scheme for vertices as BFS.

DFS: Pseudo-code

<u>DFS(*G*)</u>

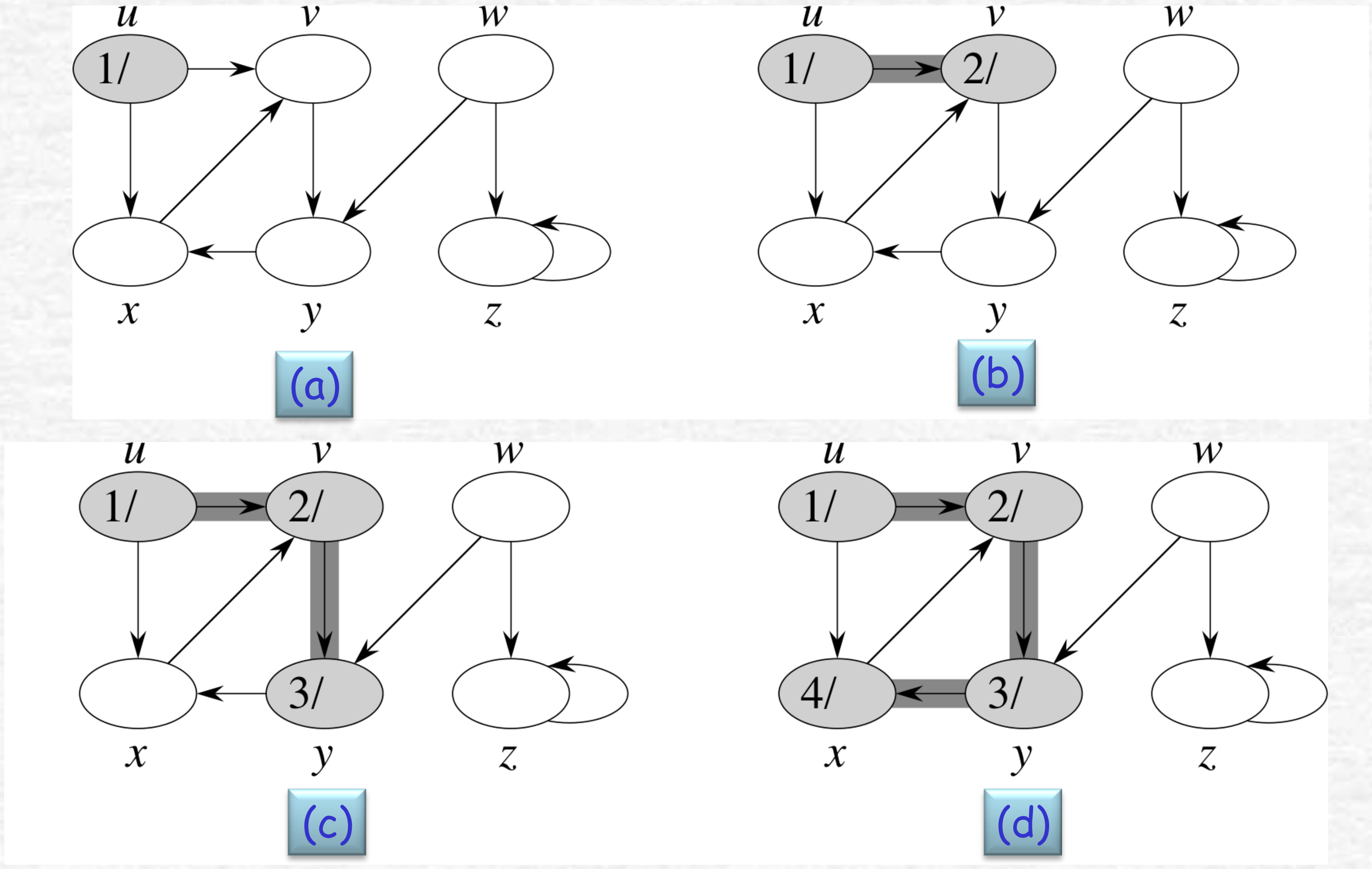
for each vertex $u \in V[G]$
    do *color*$[u] \leftarrow$ white
        $\pi[u] \leftarrow$ NIL
time $\leftarrow 0$
for each vertex $u \in V[G]$
    do if *color*$[u]$ = white
        then DFS-Visit(u)

Uses a global timestamp *time*.

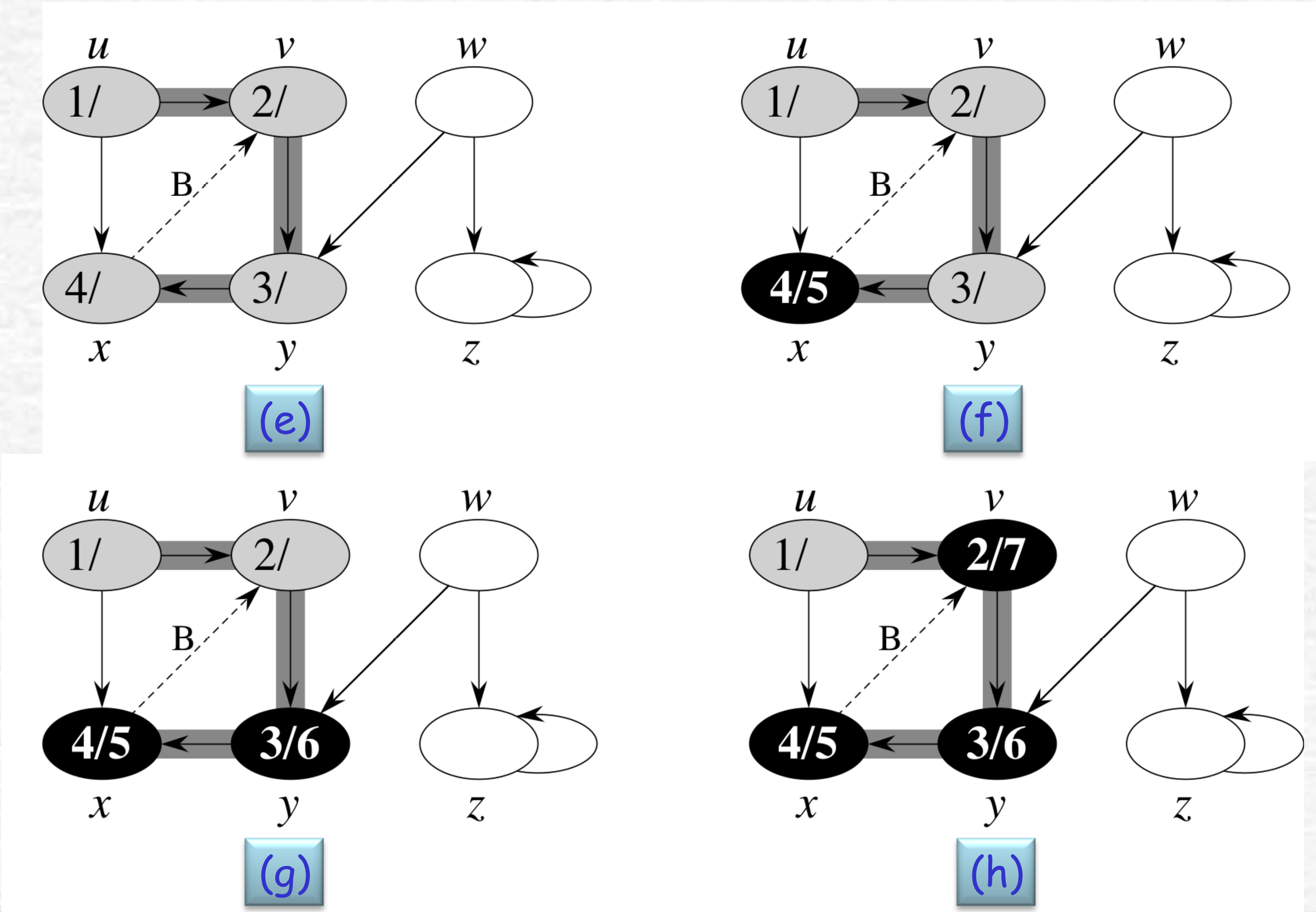
DFS-Visit(*u*)

color$[u] \leftarrow$ GRAY // White vertex u has been discovered
time $\leftarrow$ *time* + 1
$d[u] \leftarrow$ *time*
for each $v \in Adj[u]$
    do if *color*$[v]$ = WHITE
        then $\pi[v] \leftarrow u$
            DFS-Visit(v)
color$[u] \leftarrow$ BLACK // Blacken u; it is finished.
$f[u] \leftarrow$ *time* $\leftarrow$ *time* + 1

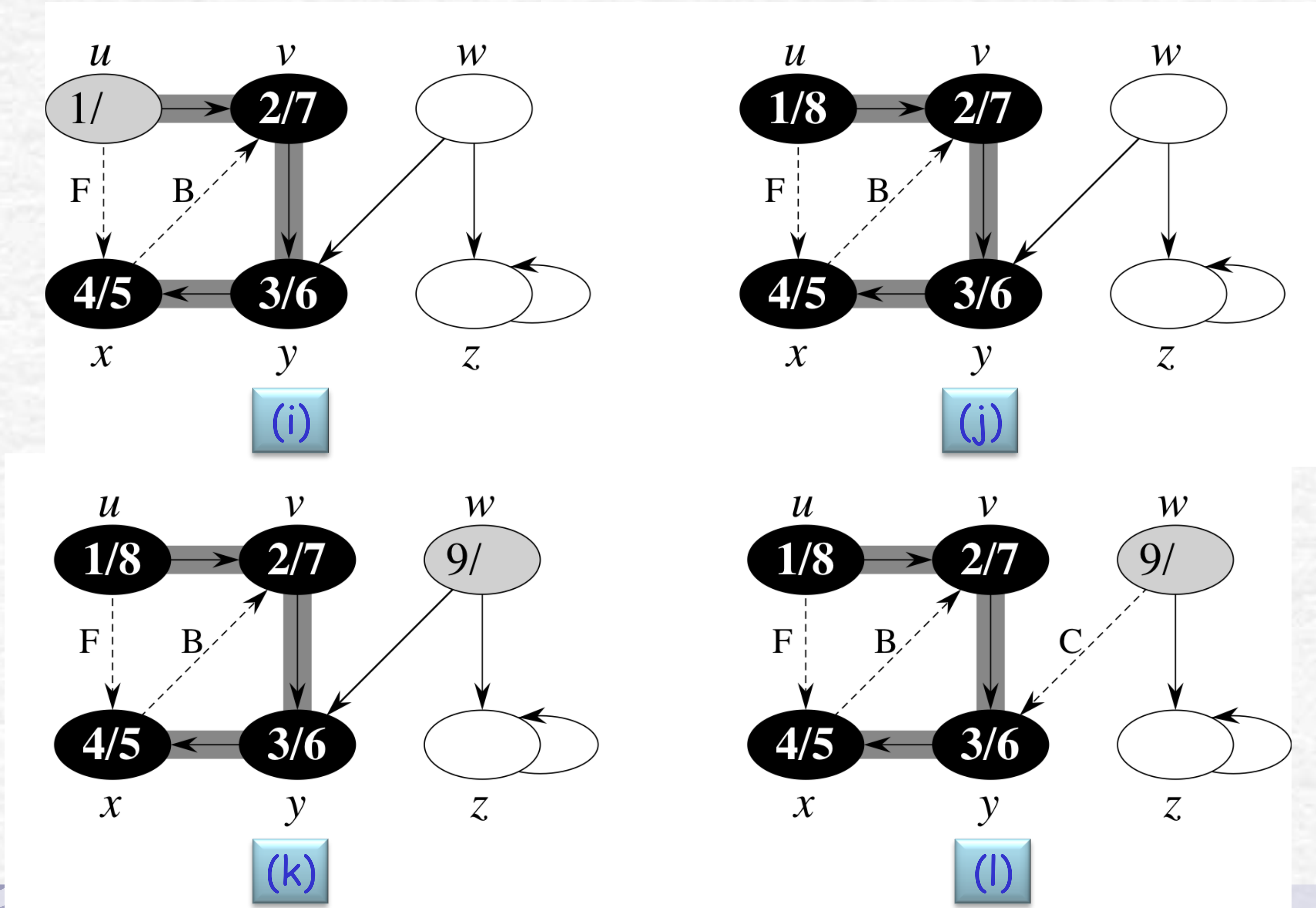
DFS: Example (1)



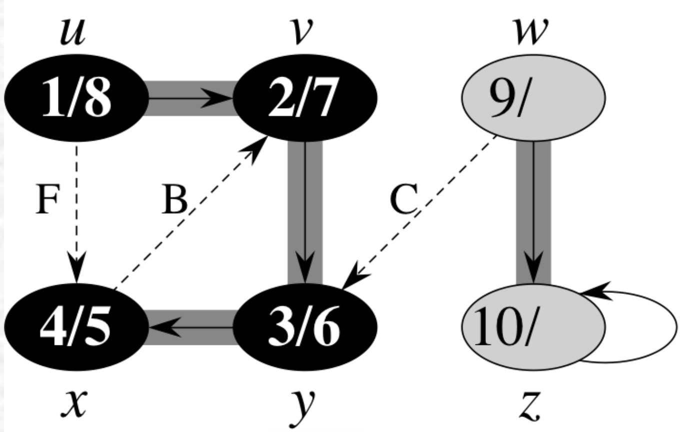
DFS: Example (2)



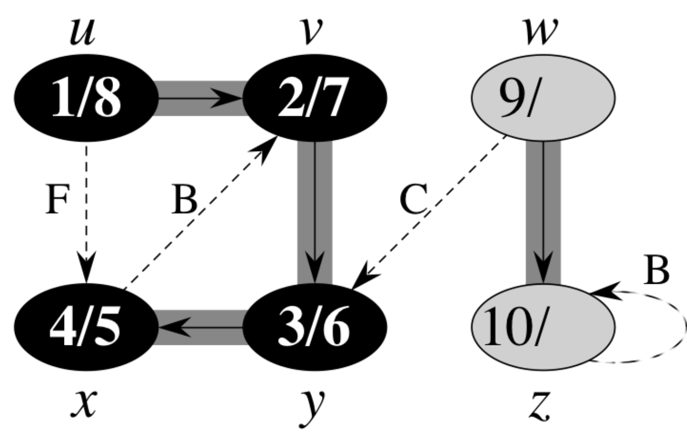
DFS: Example (3)



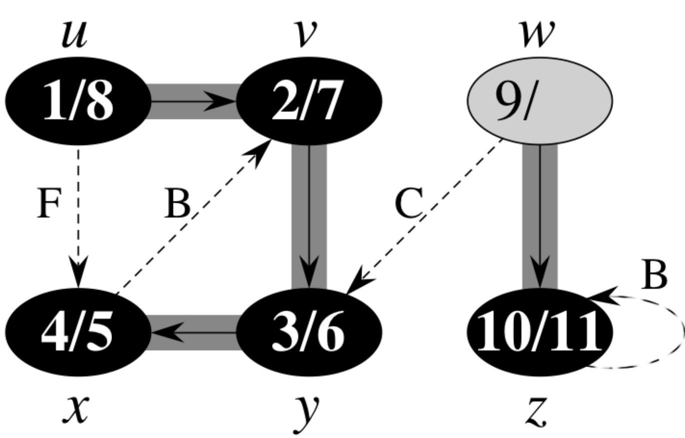
DFS: Example (4)



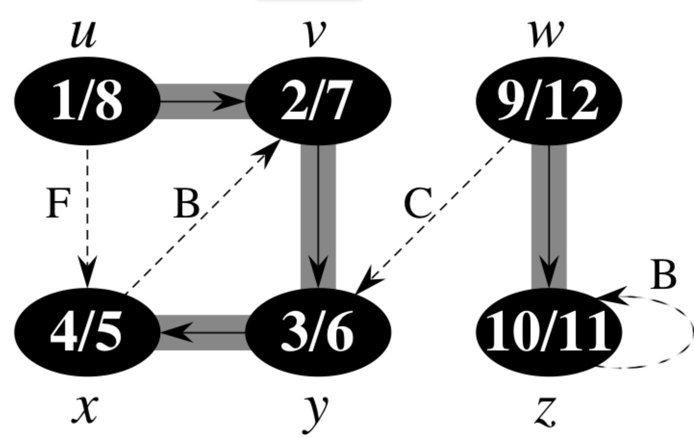
(m)



(n)



(o)



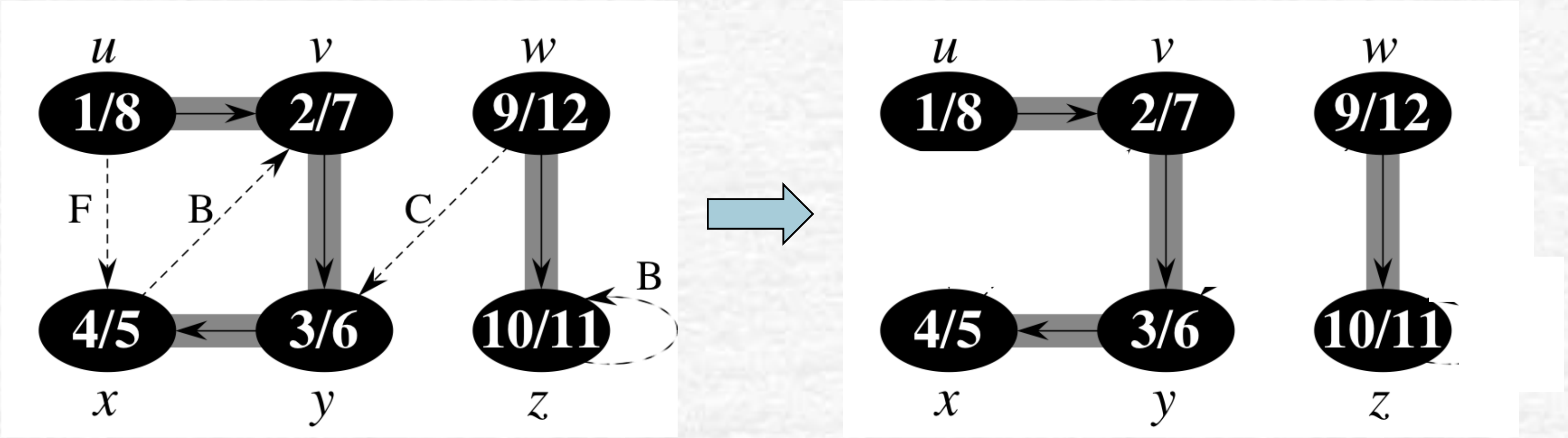
(p)

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 $|Adj[v]|$ times. The total cost of executing DFS-Visit is $\sum_{v \in V} |Adj[v]| = \Theta(|E|)$
- Total running time of DFS is $\Theta(|V|+|E|)$.

Depth-First Forest & Depth-First Trees

- DFS produce a *depth-first forest* comprised of *depth-first trees*.



- Each depth-first tree is made of edges (u, v) such that u is gray and v is white when (u, v) is explored.

DFS: Classification of Edges (1)

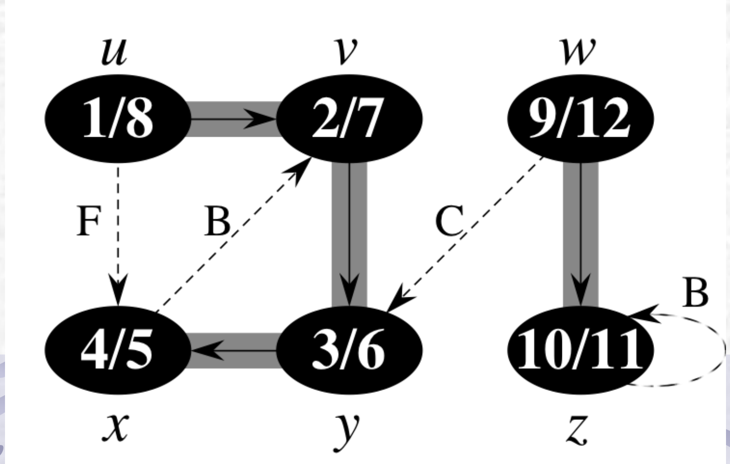
- *Tree edges* – edges belonging to the depth-first forest.
- *Back edges* – non-tree edges from a node to an *ancestor* in a depth first tree.
  - See the edges labeled B in the previous slide.
- *Forward edges* – non-tree edges from a node to a descendant in a depth first tree.
  - See the edge labeled F in the previous slide.
- *Cross edges* – the rest of the edges, can be within a single depth-first tree or between two depth-first trees.
  - See the edge labeled C in the previous slide.

DFS: Classification of Edges (2)

- The type of some edges can be determined when the edges are encountered during DFS.
- When edge (u, v) is first explored, the color of node v determines the type of (u, v):
  - It's a tree edge if v is white.
  - It's a back edge if v is gray.
  - It's a forward or cross edge if v is black.

DFS: Classification of Edges (3)

- Note that *for an undirected graph*, edge (u, v) is the same as edge (v, u).
- In this case, we *classify the edge according to whichever of (u, v) or (v, u) is first encountered* by DFS.
- *Theorem 1*: When a graph is undirected, its edges are either tree edges or back edges.
- *Example*: Treat the graph below as an undirected graph.



- Edge (*x*, *u*) would be encountered before (*u*, *x*), making the edge a back edge.
- Edge (*y*, *w*) would be encountered before (*w*, *y*), making the edge a tree edge.

DFS: Applications

- Is undirected graph G connected?
- Find connected components.
- Does a directed graph G contain a directed cycle?
- Does an undirected graph G contain a cycle?
- Is an undirected graph G a tree?

End of Ch22-25-part1