# *Elementary Graph Algorithms*

***Heejin Park***

*Division of Computer Science and Engineering*

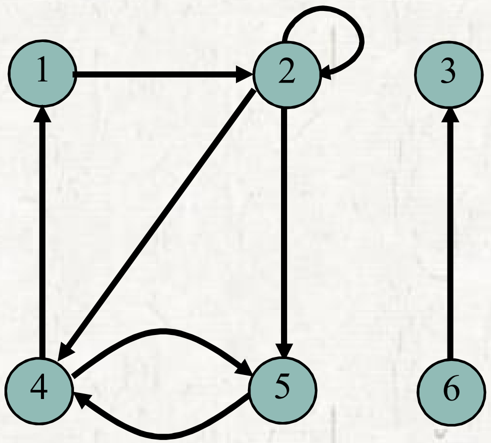
*Hanyang University*

# Contents

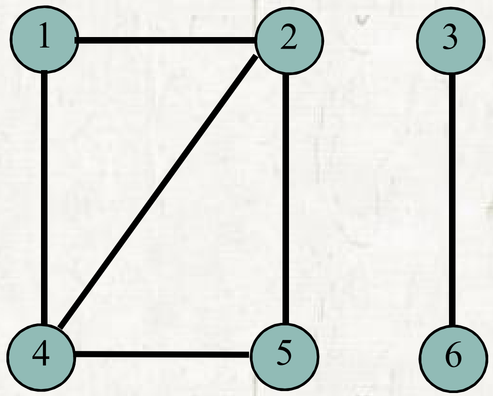
- **Graphs**
  - Graphs basics
  - Graph representation
- **Searching a graph**
  - Breadth-first search
  - Depth-first search
- **Applications of depth-first search**
  - Topological sort

# Graph basics

- A ***graph G*** is a pair (***V***, ***E***)
  where ***V*** is a ***vertex*** set and ***E*** is an ***edge*** set.
- A vertex (node) is a stand-alone object.
  - Represented by a circle.
- An edge **(**link**)** is an object connecting two vertices.
  - Represented by either an arrow or a line.



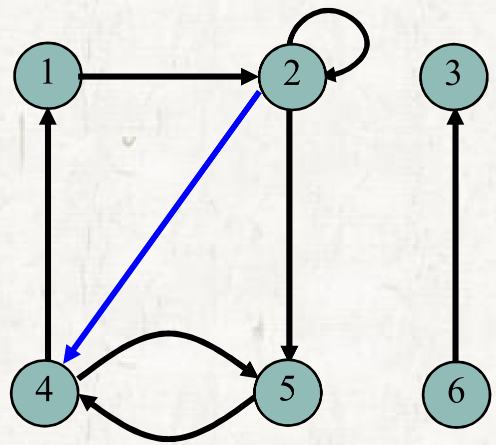

**A directed graph**




**An undirected graph**

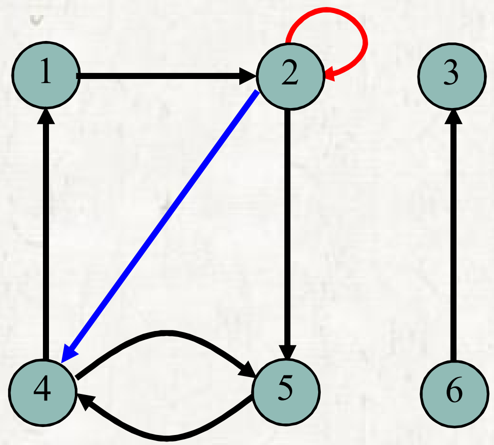
# Graph basics

- A ***directed graph*** (or ***digraph***) is a graph with ***directed edges***.
  - Edges have directions so they are represented by arrows.
  - Each edge ***leaves*** a vertex and ***enters*** a vertex.
    - The blue edge leaves vertex 2 and enters vertex 4.



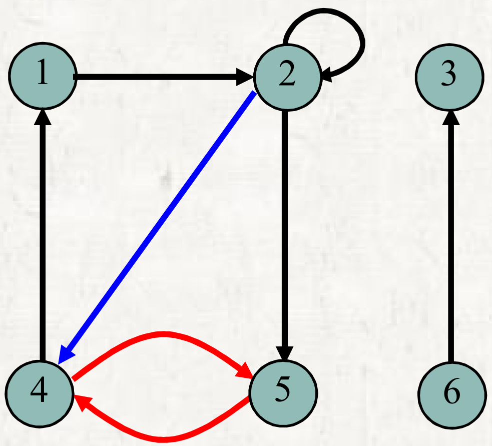
# Graph basics

- An edge leaving a vertex $u$ and entering a vertex $v$ is said it is ***incident from*** $u$ and ***incident to*** $v$.
  - The blue edge is incident from vertex 2 and to vertex 4.
- In a digraph, ***self-loops*** (edges from a vertex to itself) are possible.
  - The red edge is a self-loop.



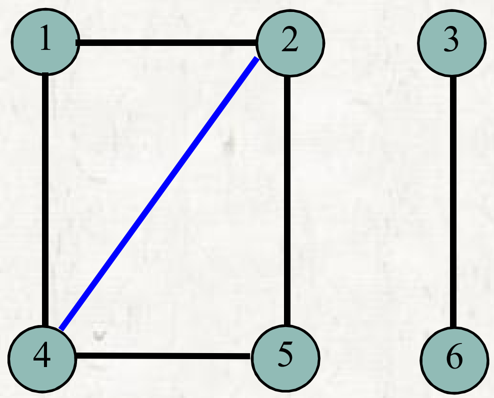
# Graph basics

- Normally, each vertex is identified by a number or a name.
  - $V = \{1, 2, 3, 4, 5, 6\}$
- Each edge is identified by the ***ordered pair of vertices*** it leaves and enters.
  - $E = \{(1,2), (2,2), (2,4), (2,5), (4,1), (4,5), (5,4), (6,3)\}$
- In a digraph, there are at most 2 edges between two vertices.



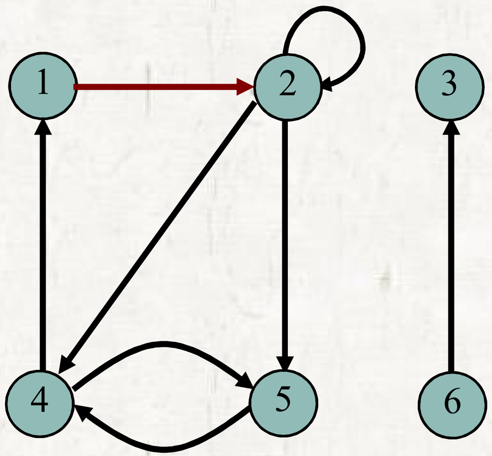
# Graph basics

- An ***undirected graph*** is a graph with ***undirected edges***.
  - Edges have no directions so they are represented by lines.
  - Self-loops are forbidden.
  - Edge $(u,v)$ is the same as edge $(v,u)$.
    - (2,4) = (4,2)
    - The blue edge is ***incident on*** vertices 2 and 4.



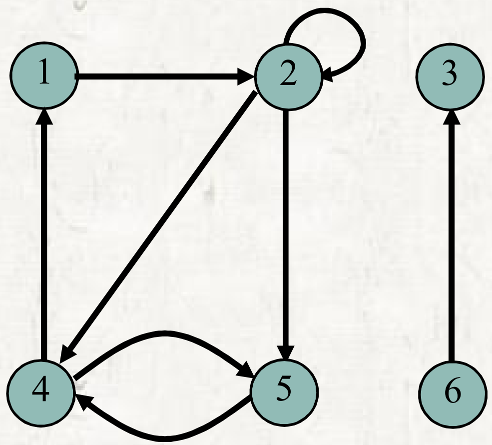
# Graph basics

- ***Adjacency***
  - If ($u$,$v$) is an edge, vertex $v$ is ***adjacent*** to vertex $u$.
  - In an undirected graph, adjacency relation is symmetric.
    - If $u$ is adjacent to $v$, $v$ is adjacent to $u$.
  - In a directed graph, it is not symmetric.
    - Vertex 2 is adjacent to 1.
    - But vertex 1 is not adjacent to 2.



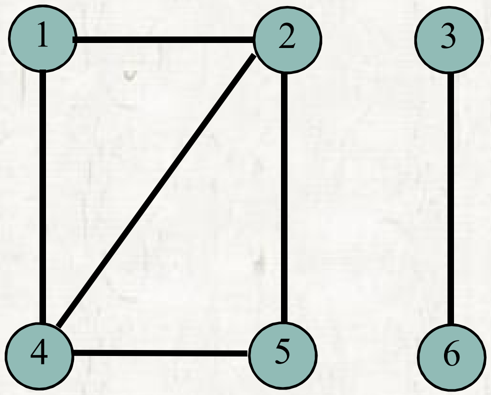
# Graph basics

- ***Degree***
  - The ***out-degree*** of a vertex is the number of edges leaving it.
    - The out-degree of vertex 2 is 3.
  - The ***in-degree*** of a vertex is the number of edges entering it.
    - The in-degree of vertext 2 is 2.
  - ***degree* = out-degree + in-degree.**



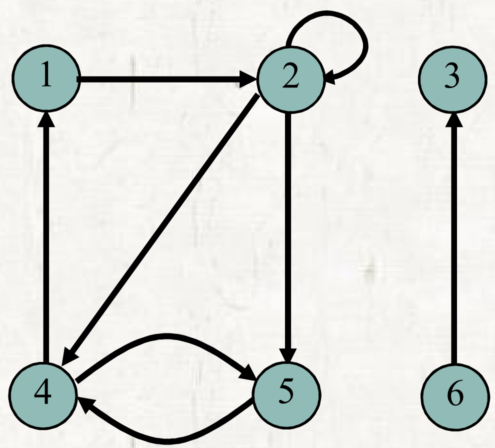
# Graph basics

- ***Degree***
  - In an undirected graph,
    - The out-degree and the in-degree are not defined.
    - Only the degree of a vertex is defined.
  - The degree of vertex 2 is 3.



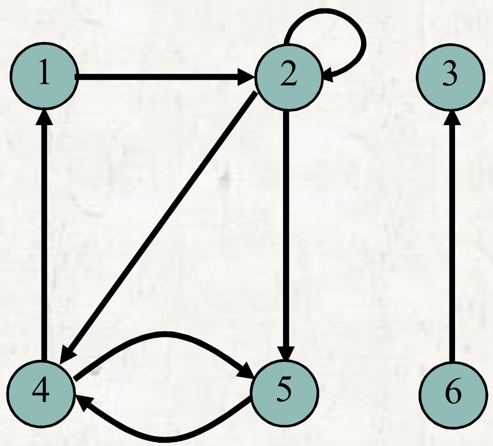
# Graph basics

- ***Path***
  - A path from vertex $u$ to vertex $v$ is a sequence of vertices $<v_0, v_1, v_2, \ldots, v_k>$ where
    - $v_0 = u$, $v_k = v$, and
    - every vertex $v_{i+1}$ $(0 \leq i \leq k\text{-}1)$ is adjacent to $v_i$.
      - There is an edge $(v_i, v_{i+1})$ for all $i$.
    - <1, 2, 4, 5> is a path.
    - <1, 2, 4, 1, 2> is a path.
    - <1, 2, 4, 2> is not a path.



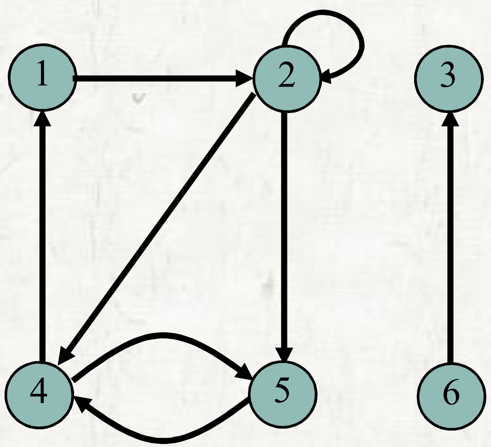
# Graph basics

- ***Path***
  - The ***length*** of a path is the number of edges in the path.
    - The length of a path <1, 2, 4, 5> is 3.
    - If there is a path from vertex *u* to vertex *v*, *v* is called ***reachable*** from *u*.
      - Vertex 5 is reachable from vertex 1.
      - Vertex 3 is not reachable from vertex 1.



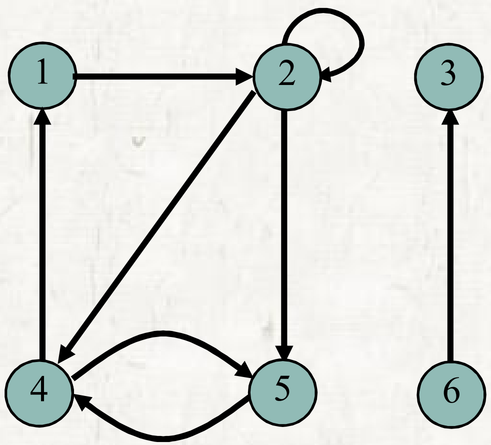
# Graph basics

- ***Simple path***
  - A path is simple if all vertices in the path are distinct.
  - A path <1, 2, 4, 5> is a simple path.
  - A path <1, 2, 4, 1, 2> is not a simple path.



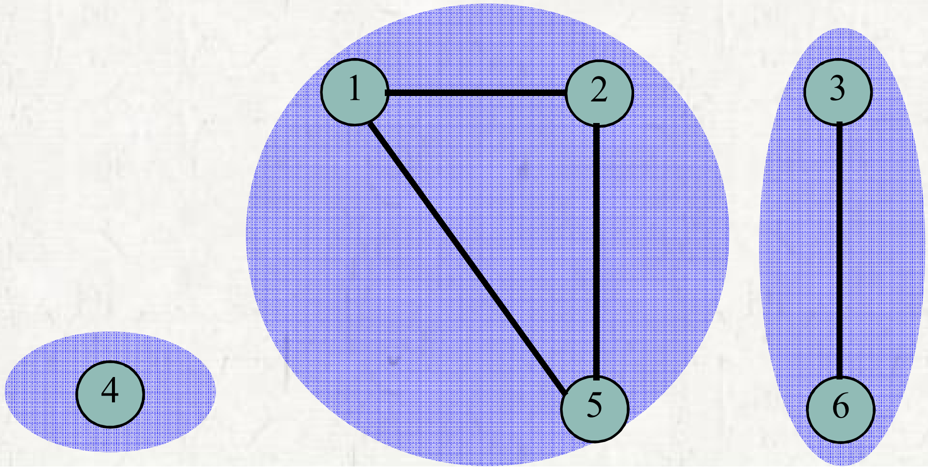
# Graph basics

- ***Cycle* and *simple cycle***
  - A path $<v_0, v_1, v_2, \ldots, v_k>$ is a cycle if $v_0 = v_k$
  - A cycle $<v_0, v_1, v_2, \ldots, v_k>$ is simple if $v_1, v_2, \ldots, v_k$ are distinct.
  - A path <1, 2, 4, 5, 4, 1> is a cycle but it is not a simple cycle.
  - A path <1, 2, 4, 1> is a simple cycle.



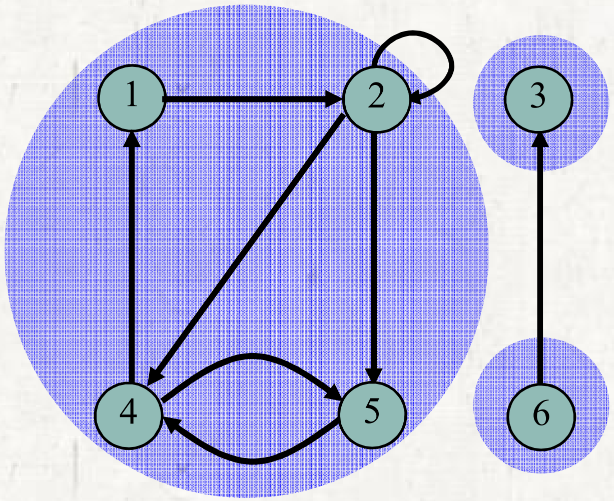
# Graph basics

- ***An acyclic graph***
  - A graph without cycles
- ***A connected graph***
  - An undirected graph is ***connected*** if every pair of vertices is connected by a path.
- ***Connected components***
  - Maximally connected subsets of vertices of an undirected graph.



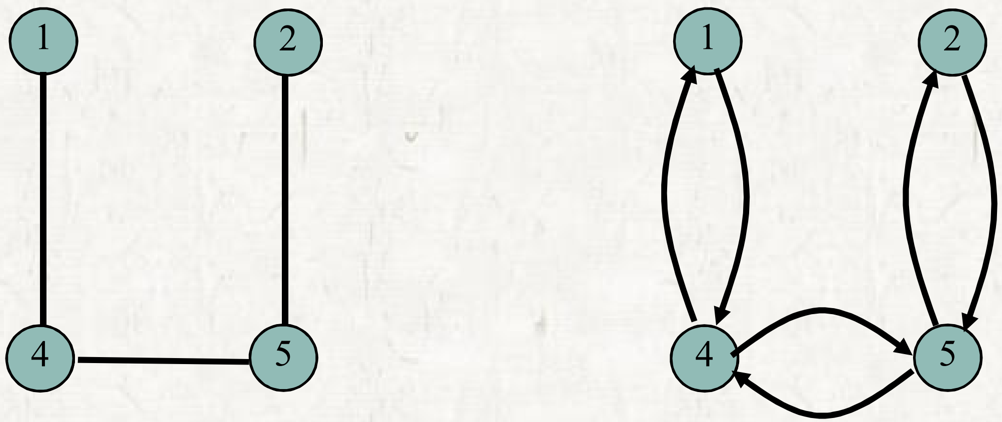
# Graph basics

- ***Strongly connected***
  - A directed graph is ***strongly connected*** if every pair of vertices is reachable from each other.
- ***Strongly connected components***
  - Maximally strongly connected subsets of vertices in a directed graph.



# Graph basics

- ***Directed version*** of an undirected graph
  - Replace each undirected edge ($u$,$v$) by two directed edges ($u$,$v$) and ($v$,$u$).



# Graph basics

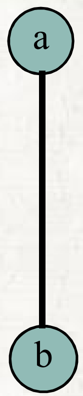
- ***Undirected version*** of a directed graph
  - Replace each directed edge $(u,v)$ by an undirected edge $(u,v)$

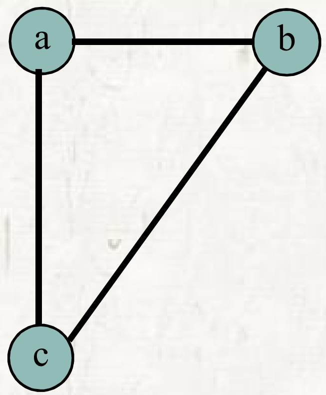
# Graph basics

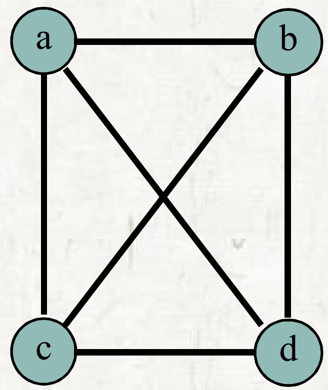
- Undirected graph $G$ → directed ver. $G'$ → undirected ver. $G''$
  - Are $G$ and $G''$ the same?
- Directed graph $G$ → undirected ver. $G'$ → directed ver. $G''$
  - Are $G$ and $G''$ the same?

# Graph basics

- ***A complete graph***
  - An undirected graph in which every pair of vertices is adjacent.



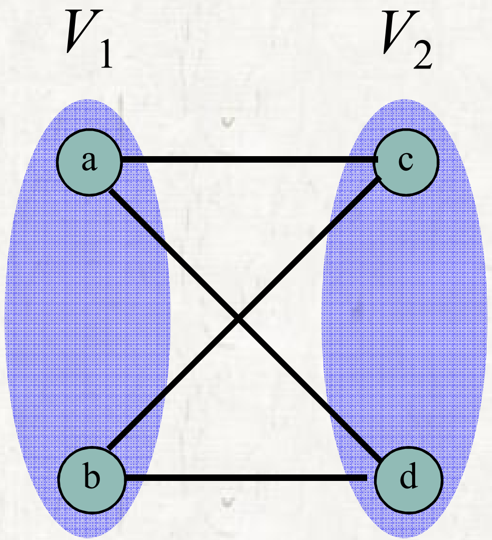




  - The number of edges with *n* vertices?

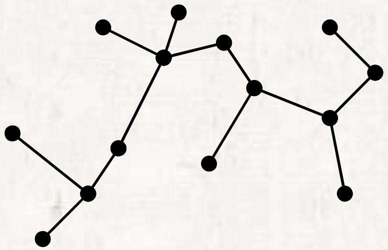
# Graph basics

- ***A bipartite graph***
  - An undirected graph $G = (V,E)$ in which $V$ can be partitioned into two sets $V_1$ and $V_2$ such that for each edge $(u,v)$, either $u \in V_1$ and $v \in V_2$ or $u \in V_2$ and $v \in V_1$.

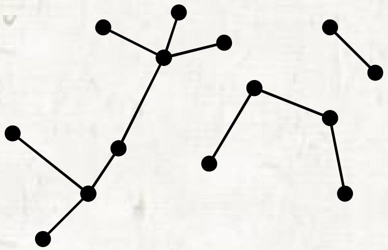



# Graph basics

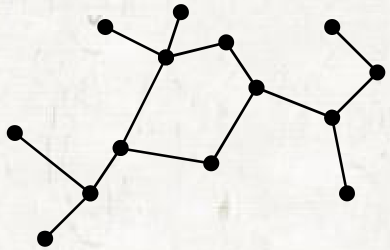
- ***Forest***
  - An acyclic, undirected graph
- ***Tree***
  - A connected forest
  - A connected, acyclic, undirected graph



T



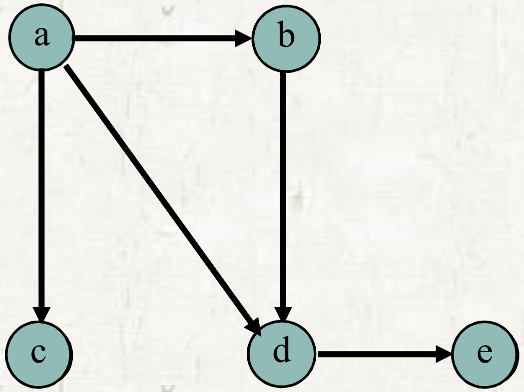
F



X

# Graph basics

- ***Dag***
  - A directed acyclic graph

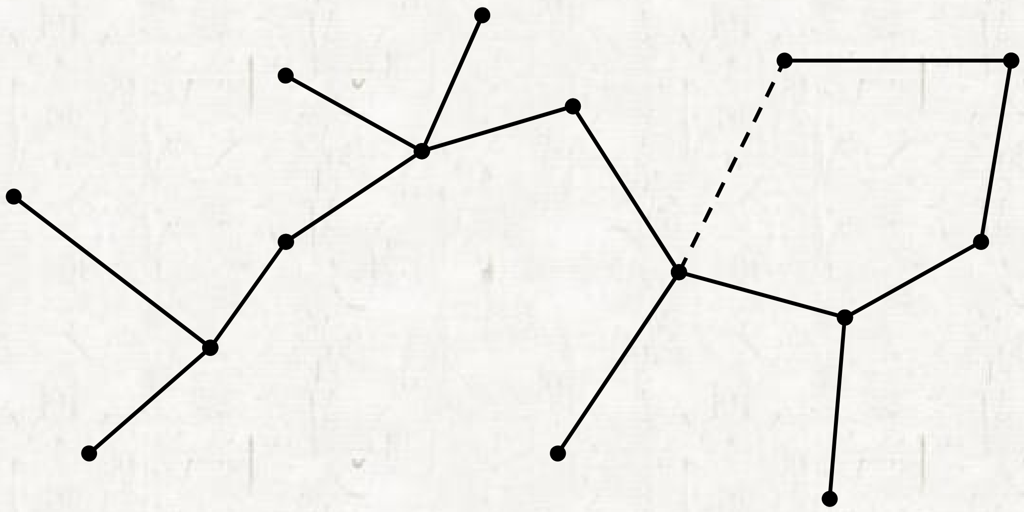



- ***Handshaking lemma***
  - If $G = (V, E)$ is an undirected graph

$$\sum_{v \in V} \text{degree}(v) = 2|E|$$

# Graph basics

- **Tree: connected, acyclic, and undirected graph**
  - Any two vertices are connected by a unique simple path.
  - If any edge is removed, the resulting graph is disconnected.
  - If any edge is added, the resulting graph contains a cycle.
  - $|E| = |V| - 1$



# Graph basics

- ***G* is a tree.**

  = *G* is a connected, acyclic, and undirected graph

  = In G, any two vertices are connected by a unique simple path.

  = *G* is connected, and if any edge is removed, the resulting graph is disconnected.

  = *G* is connected, $|E| = |V| - 1$.

  = *G* is acyclic, $|E| = |V| - 1$.

  = G is acyclic, but if any edge is added, the resulting graph contains a cycle.

# Graph basics

- **The number of edges**
  - Directed graph
    - $|E| \leq |V|^2$ Complete Graph nπn
  - Undirected graph
    - $|E| \leq |V| \, (|V|-1) \, / \, 2$ vC2

# Contents

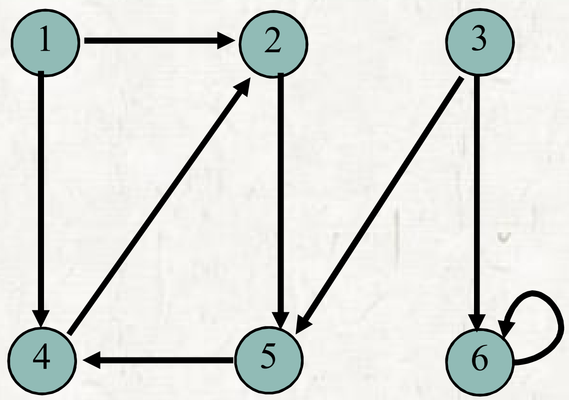
- ***Graphs***
  - *Graphs basics*
  - Graph representation
- **Searching a graph**
  - Breadth-first search
  - Depth-first search
- **Applications of depth-first search**
  - Topological sort

# Graph representation

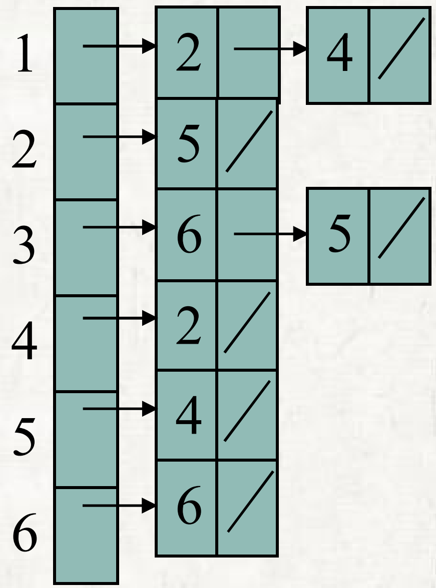
- **Representations of graphs**
  - Adjacency-list representation
  - Adjacency-matrix representation

# Graph representation

- **Adjacency-list representation**
  - An array of $|V|$ lists, one for each vertex.
  - For vertex $u$, its adjacency list contains all vertices adjacent to $u$.

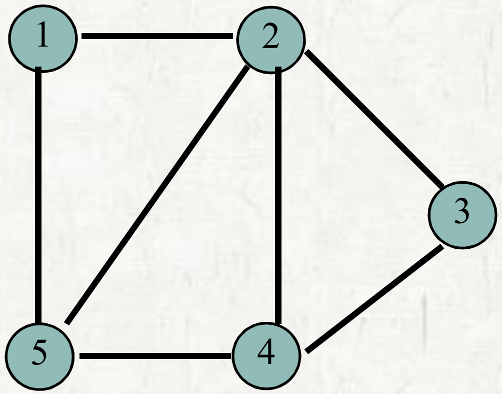



**A directed graph**

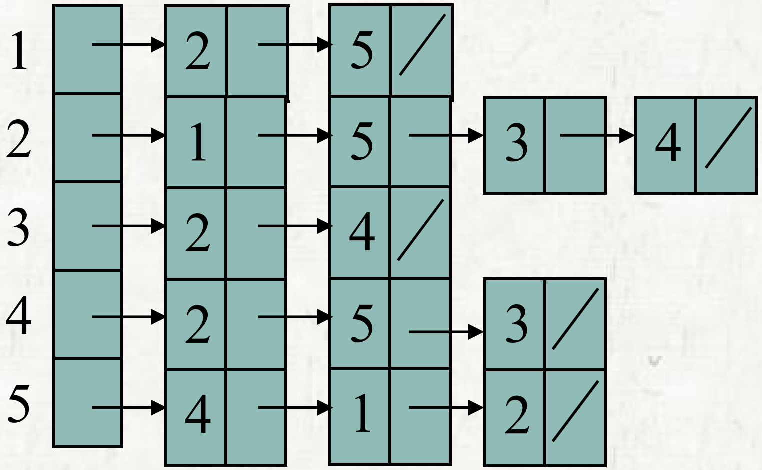


# Graph representation

- **Adjacency-list representation**
  - For an undirected graph, its directed version is stored.



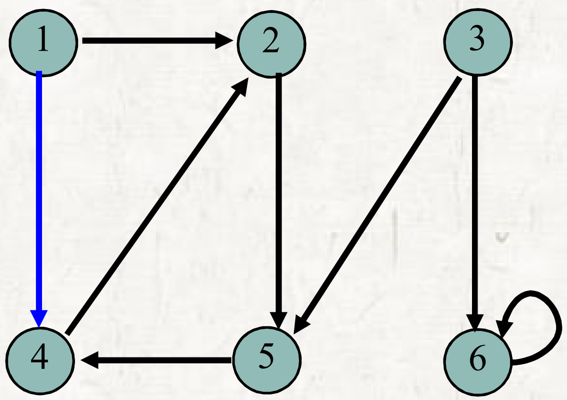

**An undirected graph**




  - $\Theta(V + E)$ space

# Graph representation

- **Adjacency-matrix representation**
  - $|V| \times |V|$ matrix: $\Theta(V^2)$ space
  - Entry $(i,j)$ is 1 if there is an edge and 0 otherwise.

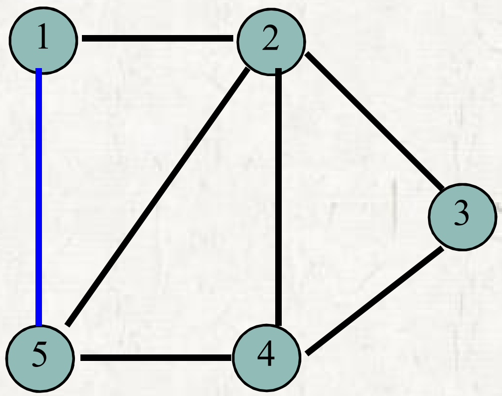



**A directed graph**

| | 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 |

# Graph representation

- **Adjacency-matrix representation**
  - $|V| \times |V|$ matrix
  - Entry $(i,j)$ is 1 if there is an edge and 0 otherwise.

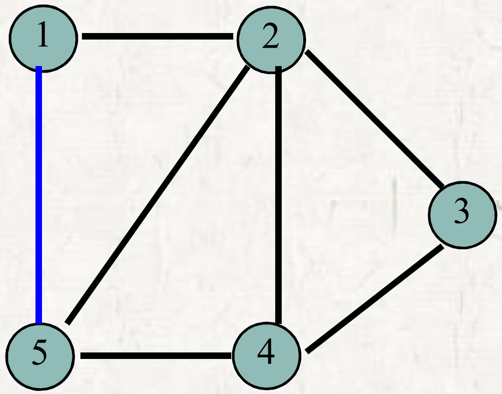



**An undirected graph**

| | 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 |

# Graph representation

- **Adjacency-matrix representation**
  - For an undirected graph, there is a symmetry along the main diagonal of its adjacency matrix.
  - Storing the lower matrix is enough.




**An 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 |

# Graph representation

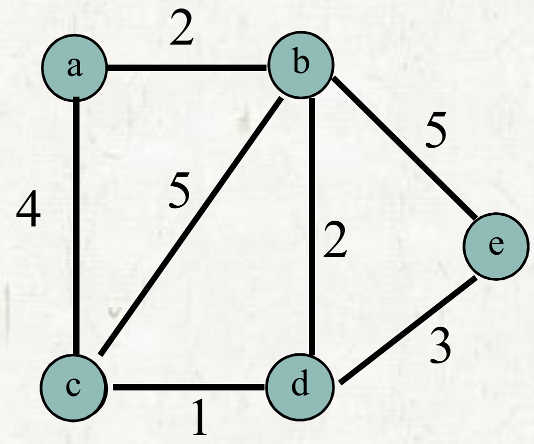
- **Comparison of adjacency list an adjacency matrix**
  - Storage
    - If $G$ is sparse, adjacency list is better.
      - because $|E| < |V|^2$.
    - If $G$ is dense, adjacency matrix is better.
      - because adjacency matrix uses only one bit for an entry.
  - Edge present test: does an edge $(i,j)$ exist?
    - Adjacency matrix: $\Theta(1)$ time.
    - Adjacency list: $O(V)$ time.

# Graph representation

- **Comparison of adjacency list and adjacency matrix**
  - Listing or visiting all edges
    - Adjacency matrix: $\Theta(V^2)$ time.
    - Adjacency list: $\Theta(V + E)$ time.

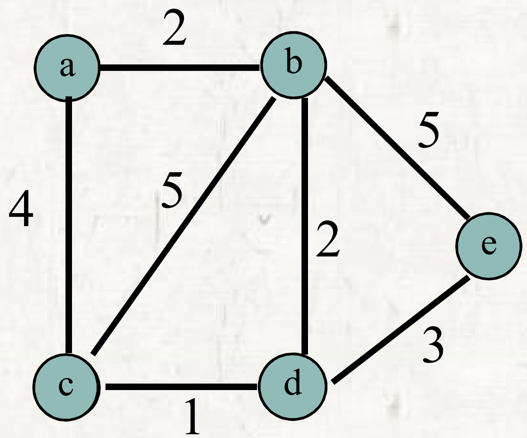
# Graph representation

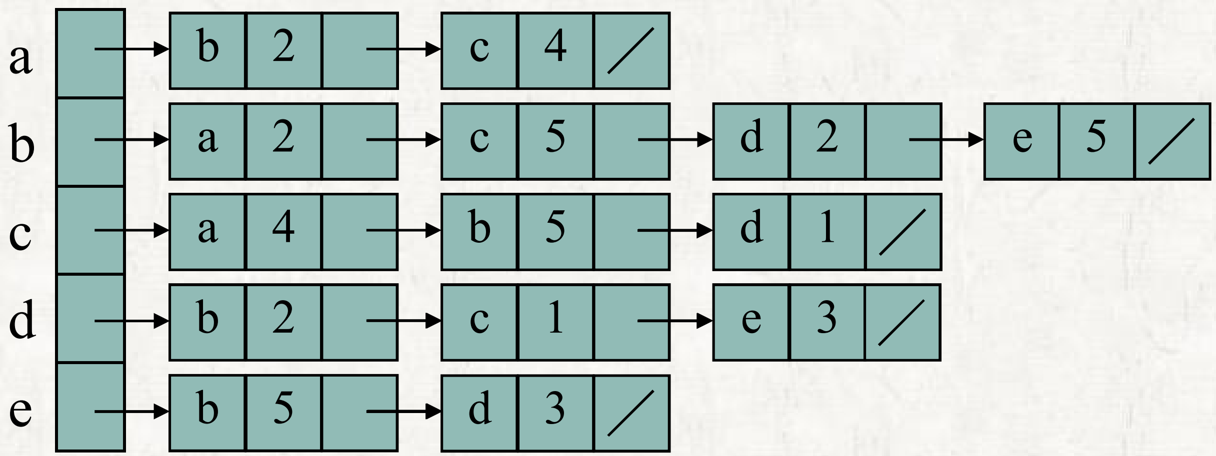
- **Weighted graph**
  - Edges have weights.



# Graph representation

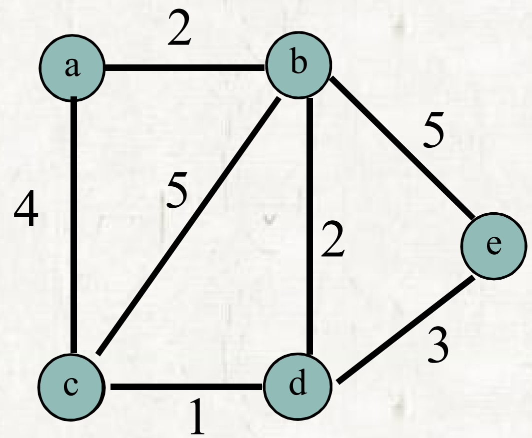
- **Weighted graph representation**
  - adjacency list



# Graph representation

- **Weighted graph representation**
  - adjacency matrix
    - $\Theta(V^2)$ space




| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | 0 | 2 | 4 | 0 | 0 |
| b | 2 | 0 | 5 | 2 | 5 |
| c | 4 | 5 | 0 | 1 | 0 |
| d | 0 | 2 | 1 | 0 | 3 |
| e | 0 | 5 | 0 | 3 | 0 |

# Graph representation

- **Transpose of a matrix**
  - The ***transpose*** of a matrix $A = (a_{ij})$ is
  - $A^T = (a_{ij}^T)$ where $a_{ij}^T = a_{ji}$
  - An undirected graph is its own transpose: $A = A^{\mathrm{T}}$.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$$

방향 바뀜

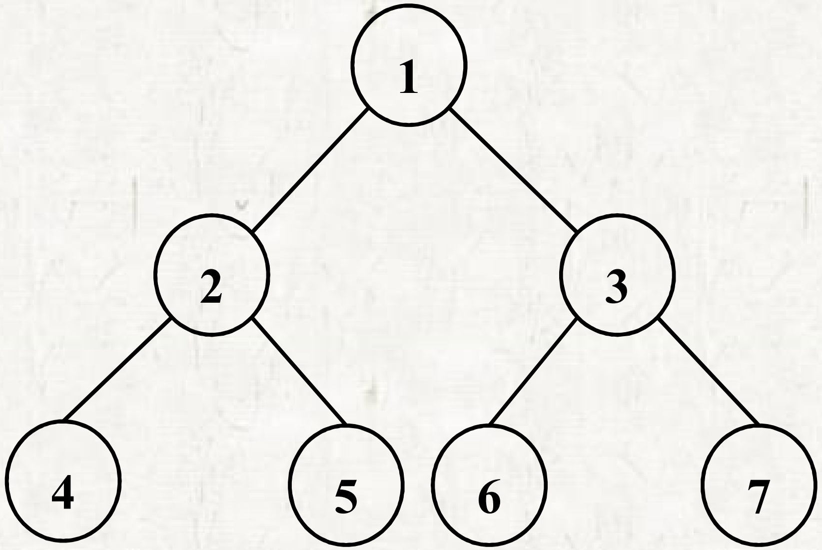
# Self-study

- **Exercise 22.1-3**
  - The transpose of a directed graph
- **Exercise 22.1-4**
  - Removing duplicate edges in a multigraph in $O(V+E)$ time.
- **Exercise 22.1-6**
  - Universal sink detection in $O(V)$ time.

# Contents

- ***Graphs***
  - *Graphs basics*
  - *Graph representation*
- **Searching a graph**
  - Breadth-first search
  - Depth-first search
- **Applications of depth-first search**
  - Topological sort

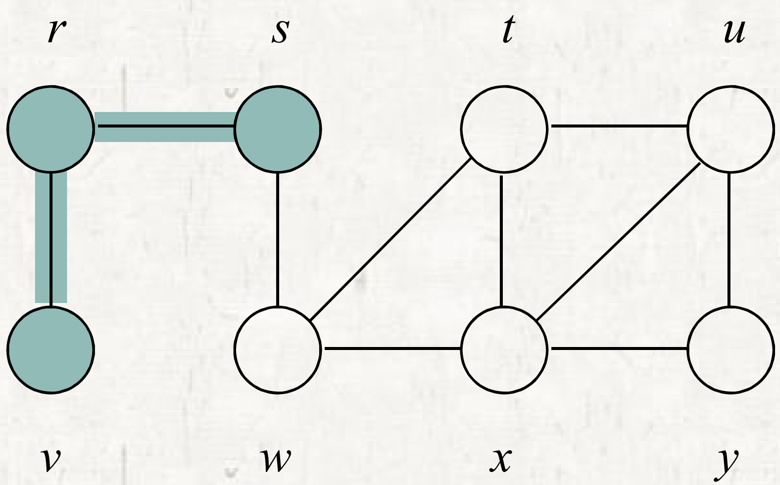
# Searching a tree




- **Breadth-first search**
- **Depth-first search**

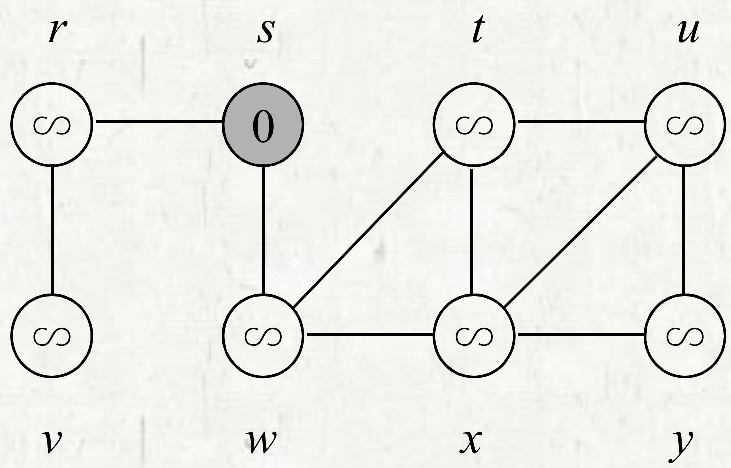
# Breadth-first search

- ***Distance***
  - Distance from *u* to *v*
    - The number of edges in the shortest path from *u* to *v*.
    - The distance from *s* to *v* is 2.



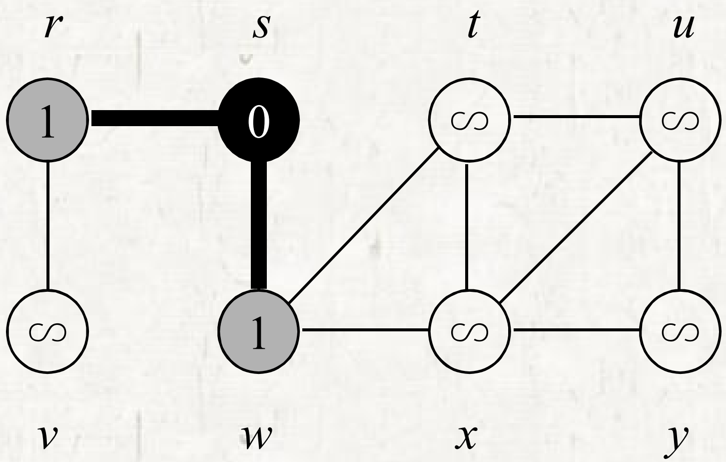
# Breadth-first search

- **Breadth-first search**
  - Given a graph $G = (V, E)$ and a ***source*** vertex $s$, it explores the edges of $G$ to "discover" every reachable vertex from $s$.
  - It discovers vertices in the increasing order of distance from the source. It first discovers all vertices at distance 1, then 2, and etc.



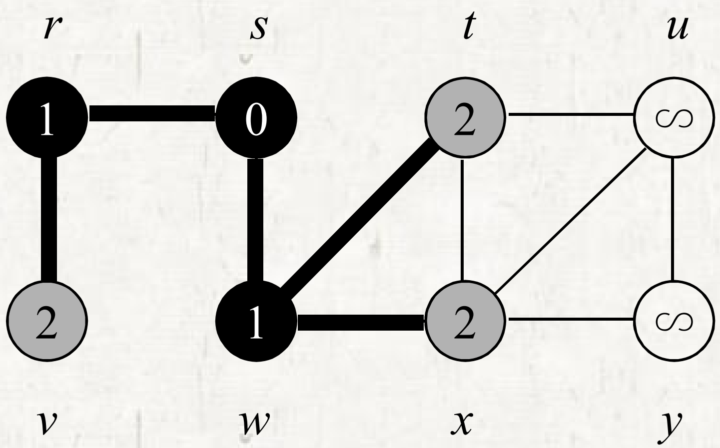
# Breadth-first search

- **Breadth-first search**
  - Given a graph $G = (V, E)$ and a ***source*** vertex $s$, it explores the edges of $G$ to "discover" every reachable vertex from $s$.
  - It discovers vertices in the increasing order of distance from the source. It first discovers all vertices at distance 1, then 2, and etc.



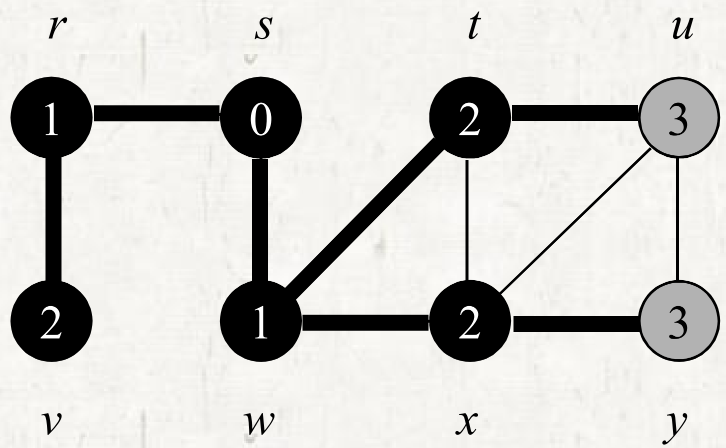
# Breadth-first search

- **Breadth-first search**
  - Given a graph $G = (V, E)$ and a ***source*** vertex $s$, it explores the edges of $G$ to "discover" every reachable vertex from $s$.
  - It discovers vertices in the increasing order of distance from the source. It first discovers all vertices at distance 1, then 2, and etc.



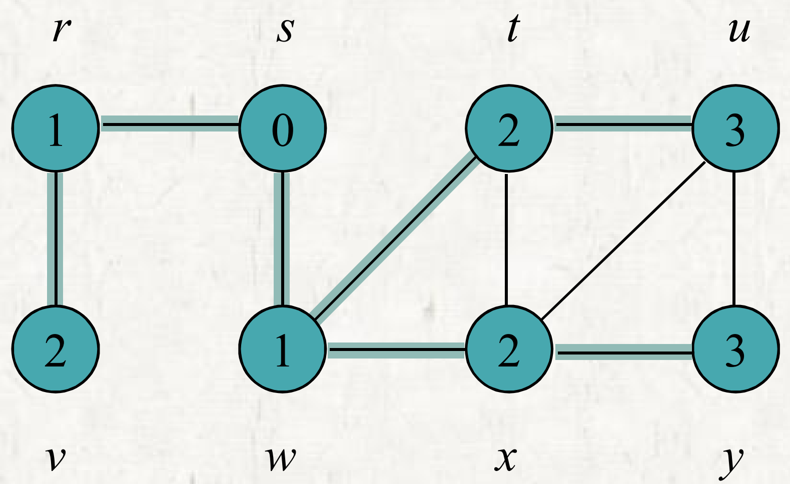
# Breadth-first search

- **Breadth-first search**
  - Given a graph $G = (V, E)$ and a ***source*** vertex $s$, it explores the edges of $G$ to "discover" every reachable vertex from $s$.
  - It discovers vertices in the increasing order of distance from the source. It first discovers all vertices at distance 1, then 2, and etc.



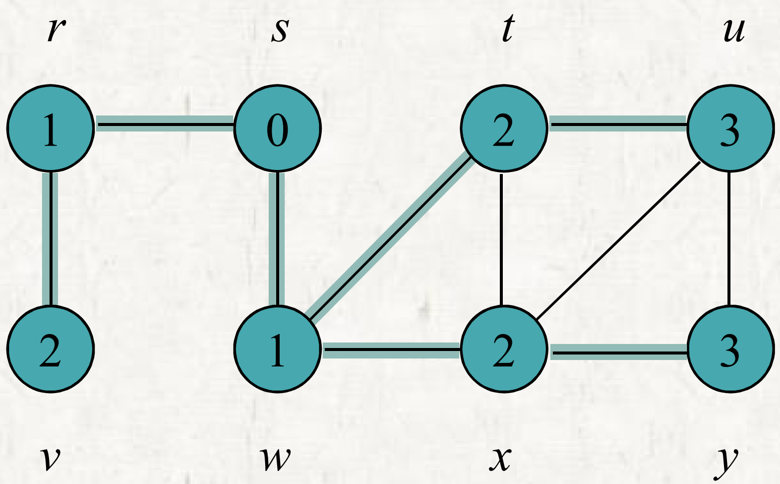
# Breadth-first search

- **Breadth-first search**
  - It also computes
    - the distance of vertices from the source: $u.d = 3$
    - the predecessor of vertices: $u.\pi = t$



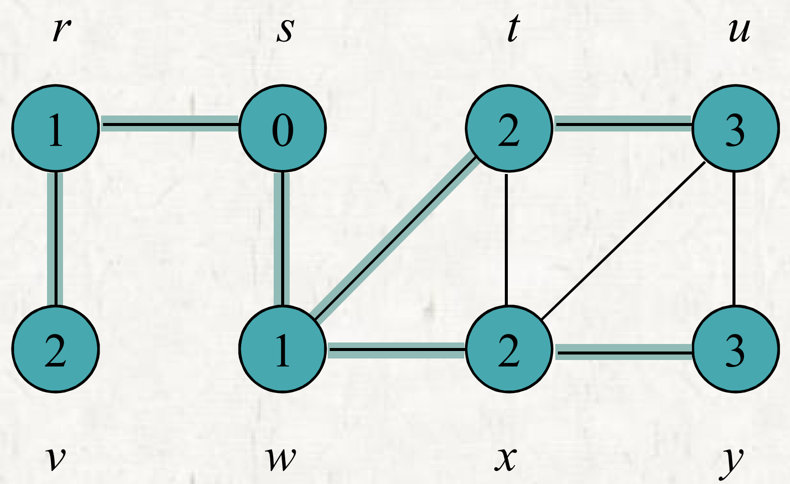
# Breadth-first search

- The ***predecessor subgraph*** of $G$ as $G_\pi = (V_\pi, E_\pi)$,
  - $V_\pi = \{v \in V: v.\pi \neq \text{NIL}\} \cup \{s\}$
  - $E_\pi = \{(v.\pi, v) : v \in V_\pi - \{s\}\}$.



# Breadth-first search

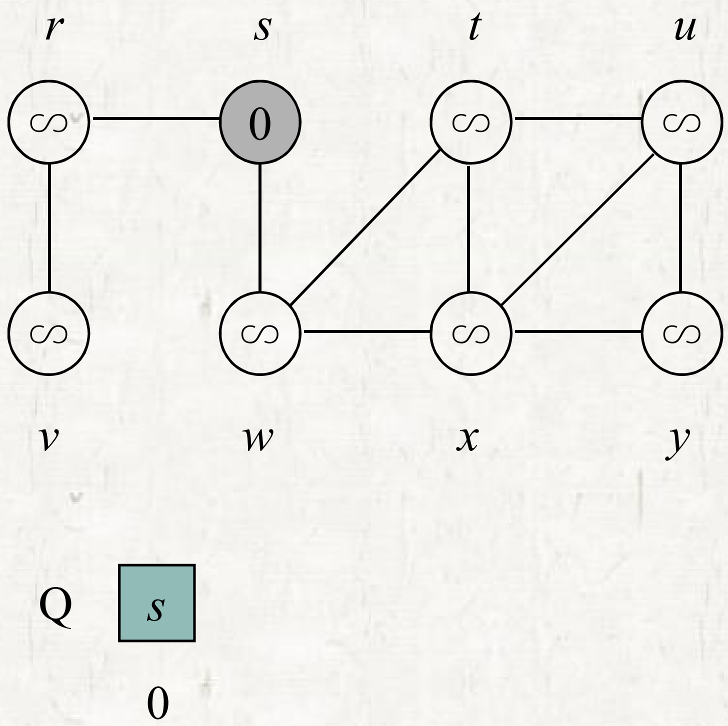
- The predecessor subgraph $G_\pi$ is a ***breadth-first tree.***
  - since it is connected and $|E_\pi| = |V_\pi|$ -1.
  - The edges in $E_\pi$ are called ***tree edges***.

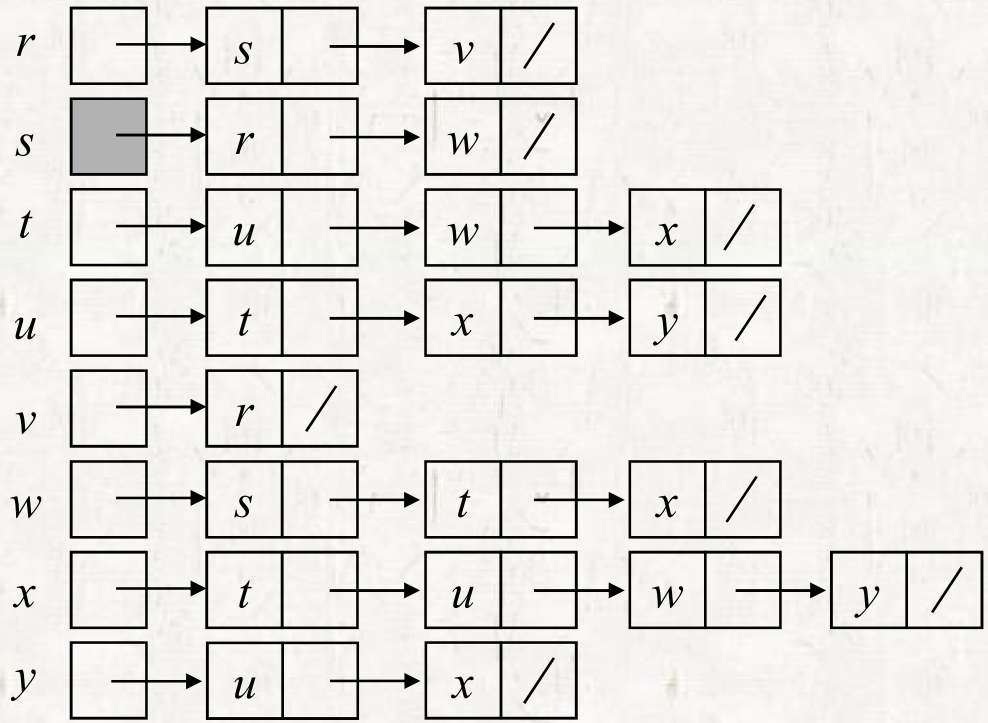


# Breadth-first search

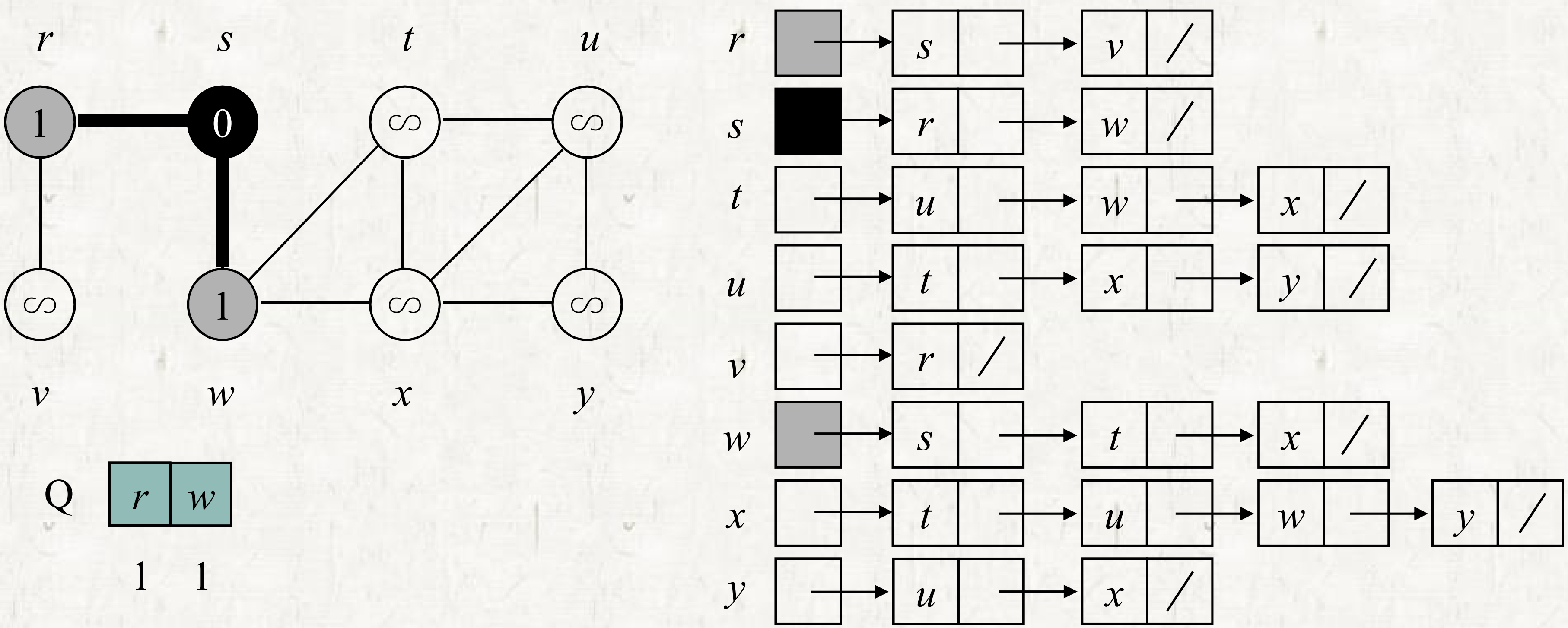
```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = Ø
ENQUEUE(Q, s)
while Q ≠ Ø
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

# Breadth-first search



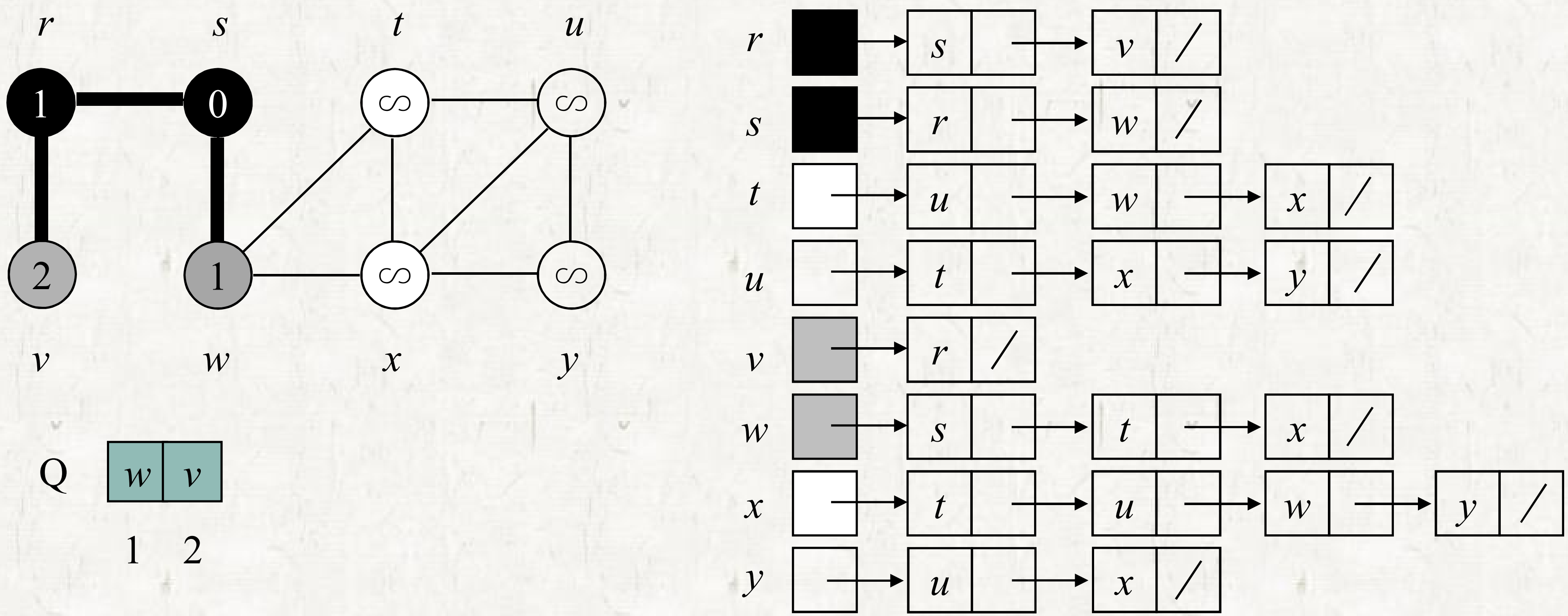



# Breadth-first search

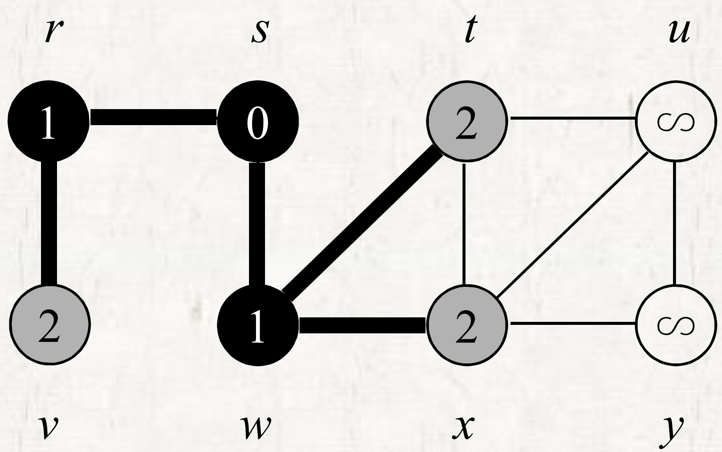



- white: not discovered (not entered the Q)
- gray: discovered (in the Q)
- black: finished (out of the Q)

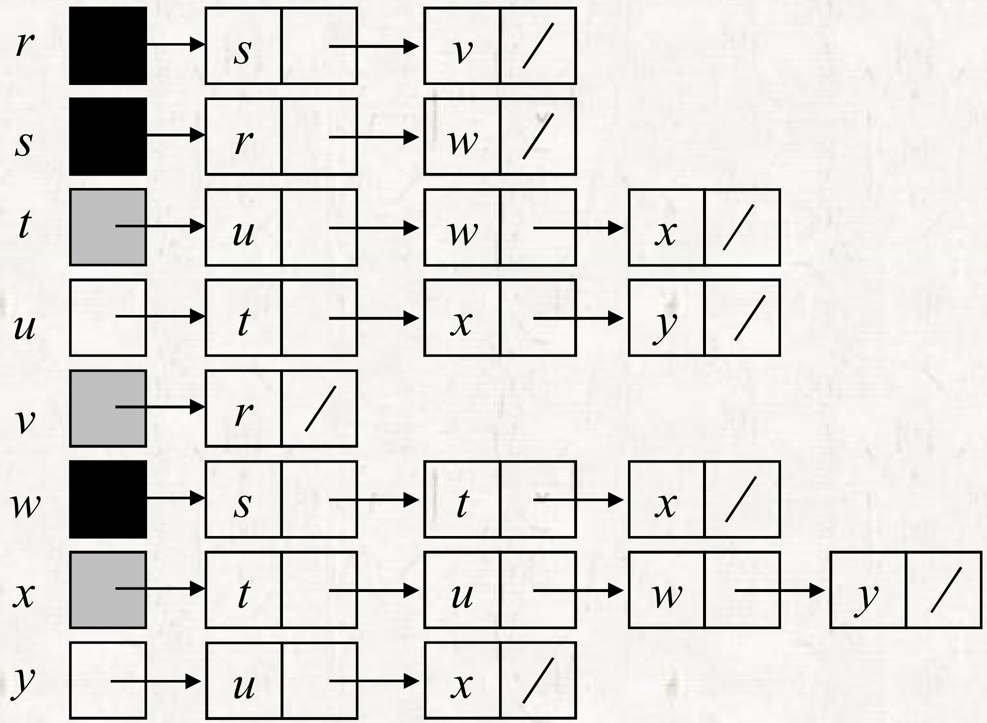
# Breadth-first search



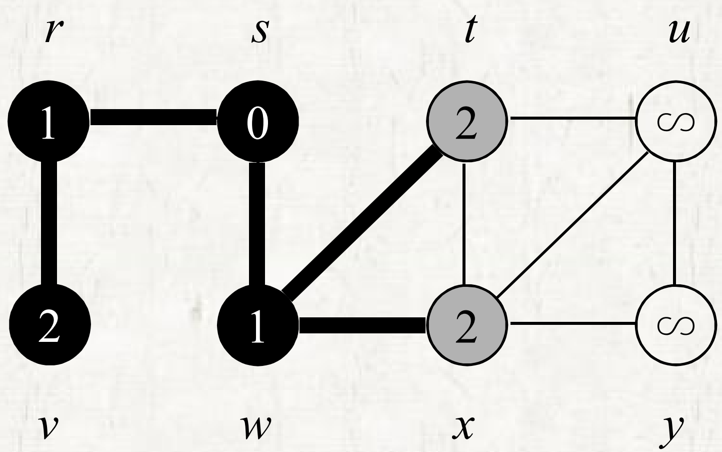
# Breadth-first search




| Q | $v$ | $t$ | $x$ |
|---|---|---|---|
| | 2 | 2 | 2 |

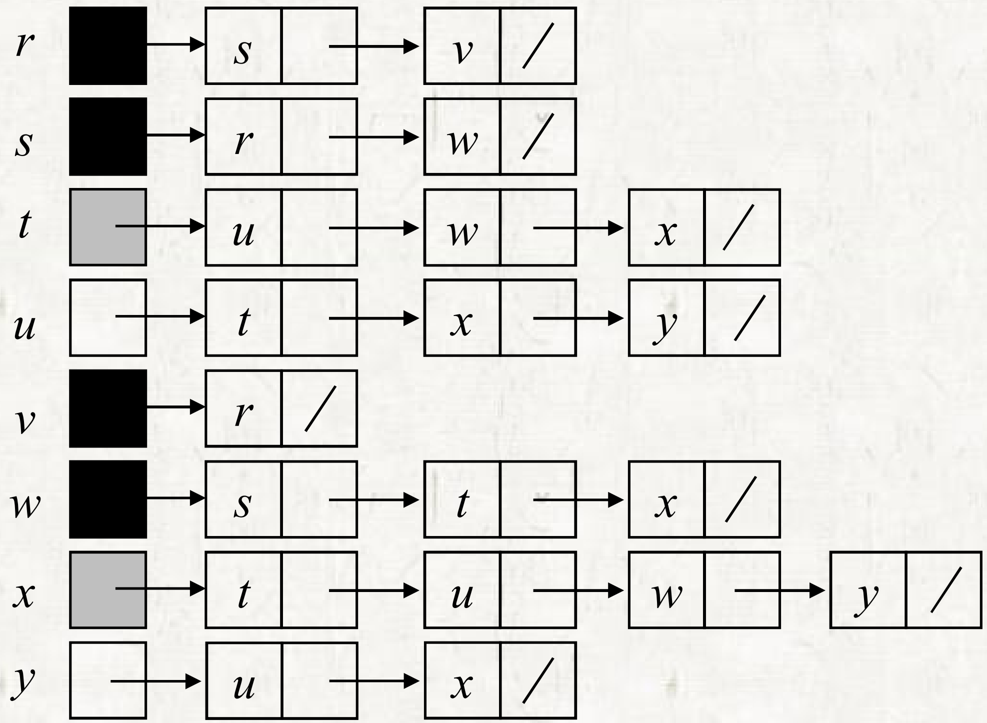


# Breadth-first search

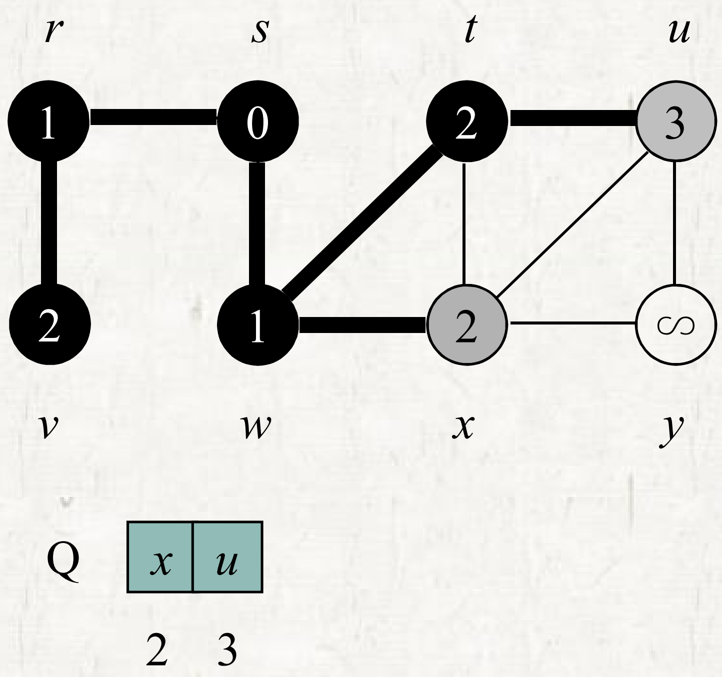


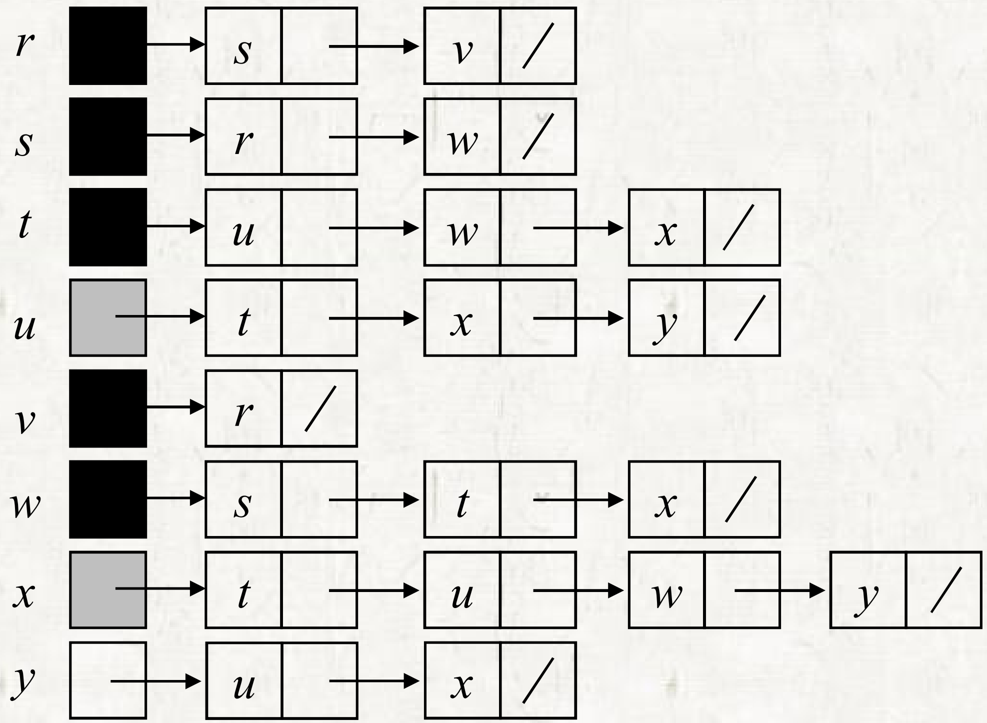

Q

| t | x |
|---|---|
| 2 | 2 |

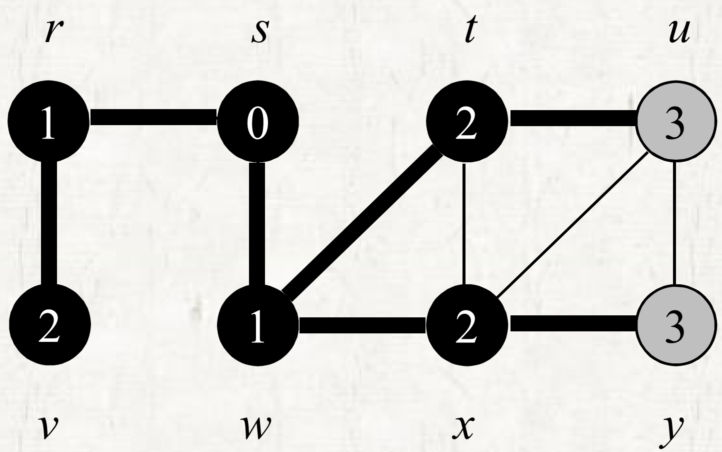


# Breadth-first search

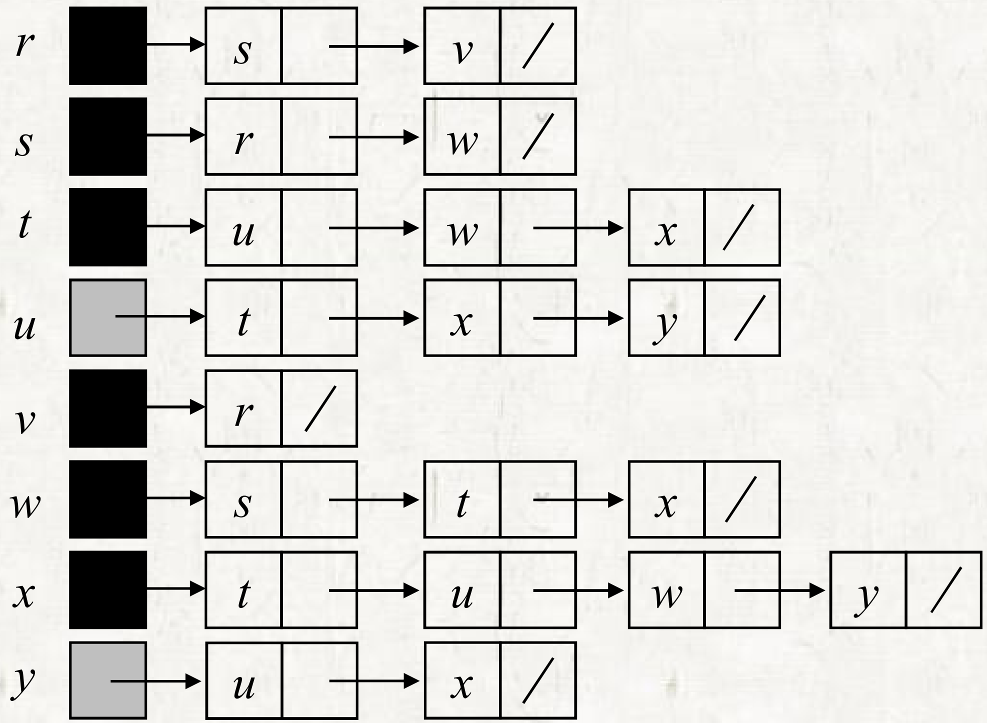


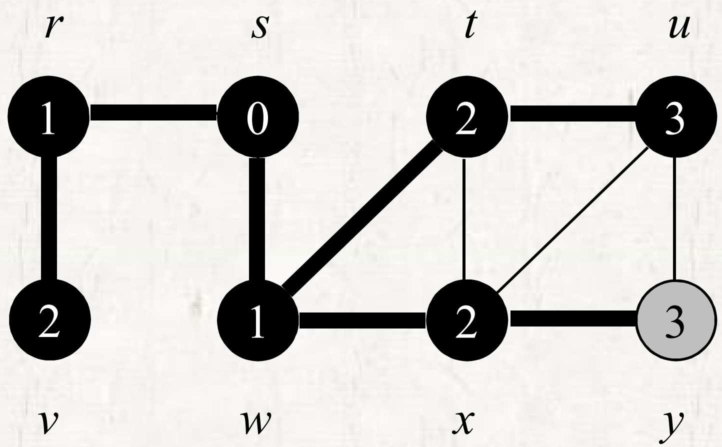
# Breadth-first search

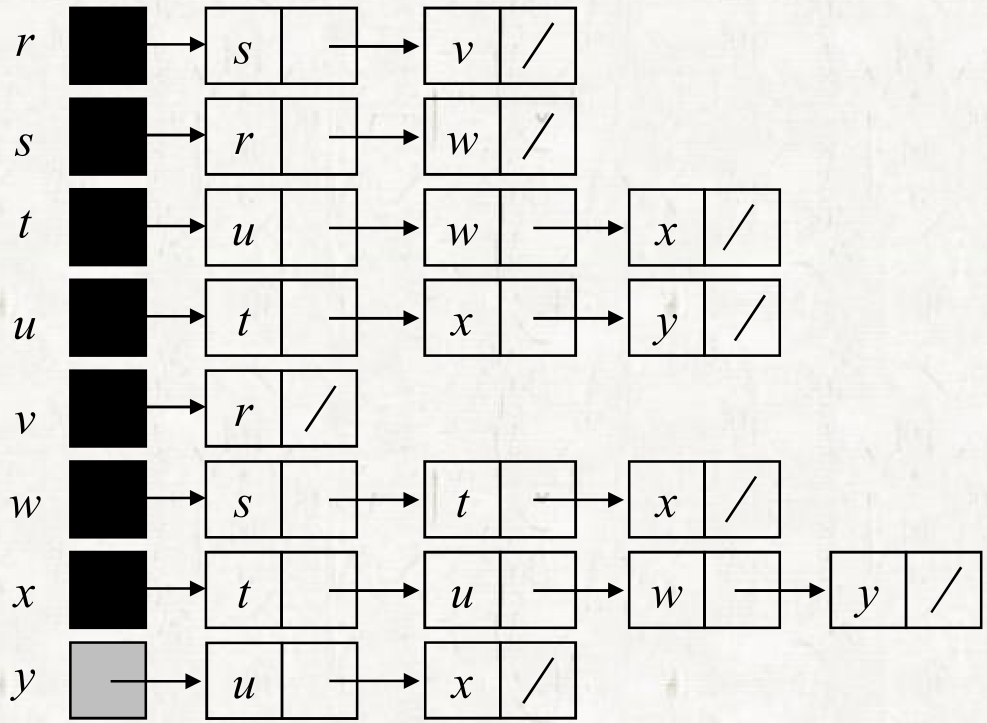



| Q | *u* | *y* |
|---|---|---|
| | 3 | 3 |



# Breadth-first search

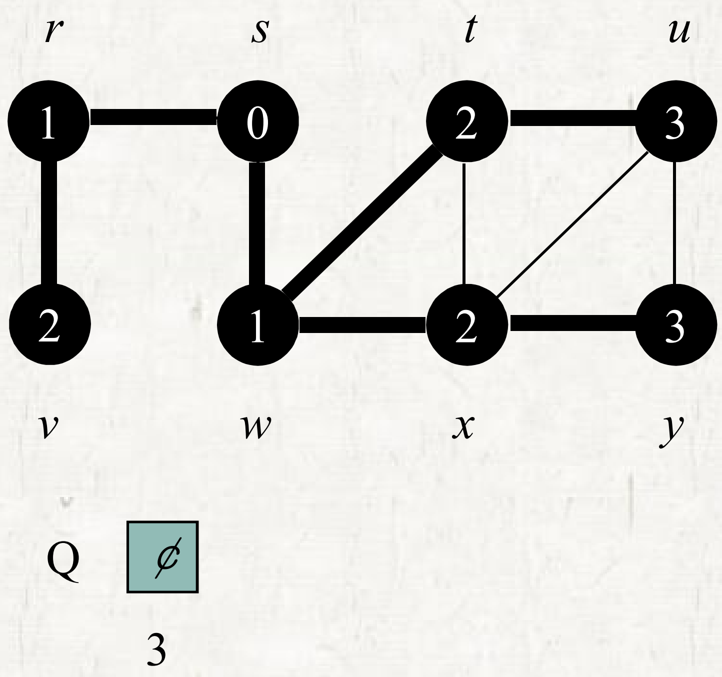


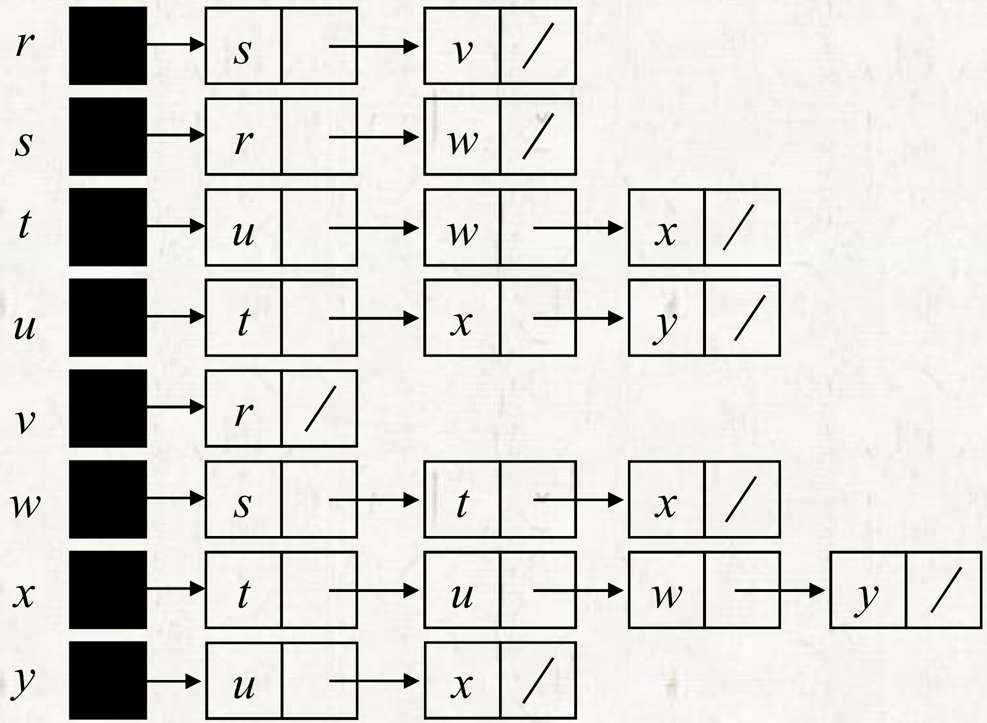




Q y

3

# Breadth-first search



# Breadth-first search

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d  = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = Ø
ENQUEUE(Q, s)
while Q ≠ Ø
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if  v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

# Breadth-first search

- **Running time**
  - **Initialization**: $\Theta(V)$
  - **Exploring the graph**: $O(V + E)$
    - A vertex is examined at most once.
    - An edge is explored at most twice.
  - **Overall:** $O(V + E)$

# Self-study

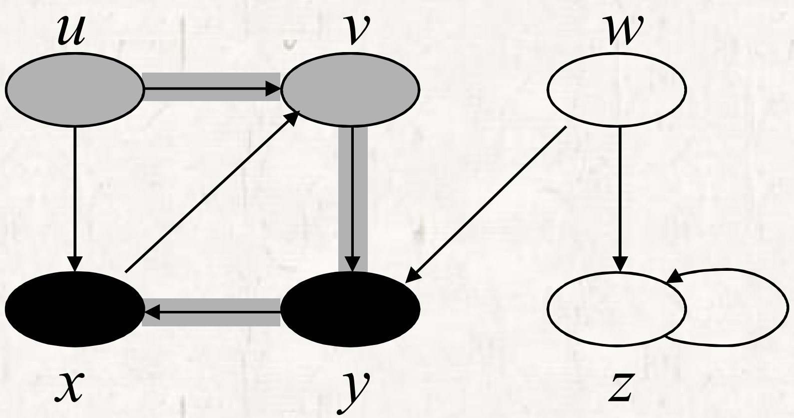
- **Exercise 22.2-4 (22.2-3 in the 2nd ed.)**
  - The running time of BFS
    with adjacency matrix representation.
- **Exercise 22.2-6 (22.2-5 in the 2nd ed.)**
  - Impossible breadth-first trees.
- **Exercise 22.2-7 (22.2-6 in the 2nd ed.)**
  - Rivalry

# Contents

- ***Graphs***
  - *Graphs basics*
  - *Graph representation*
- ***Searching a graph***
  - *Breadth-first search*
  - Depth-first search
- **Applications of depth-first search**
  - Topological sort

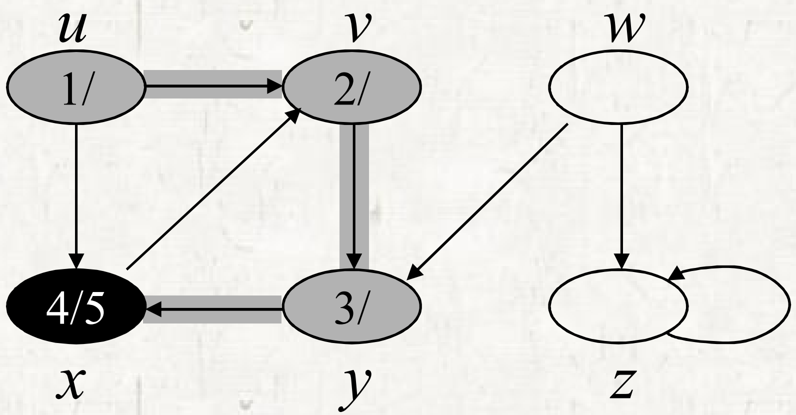
# Depth-first search

- ***Colors of vertices***
  - Each vertex is initially ***white***. (not discovered)
  - The vertex is ***grayed*** when it is ***discovered.***
  - The vertex is ***blackened*** when it is ***finished***, that is, when its adjacency list has been examined completely.

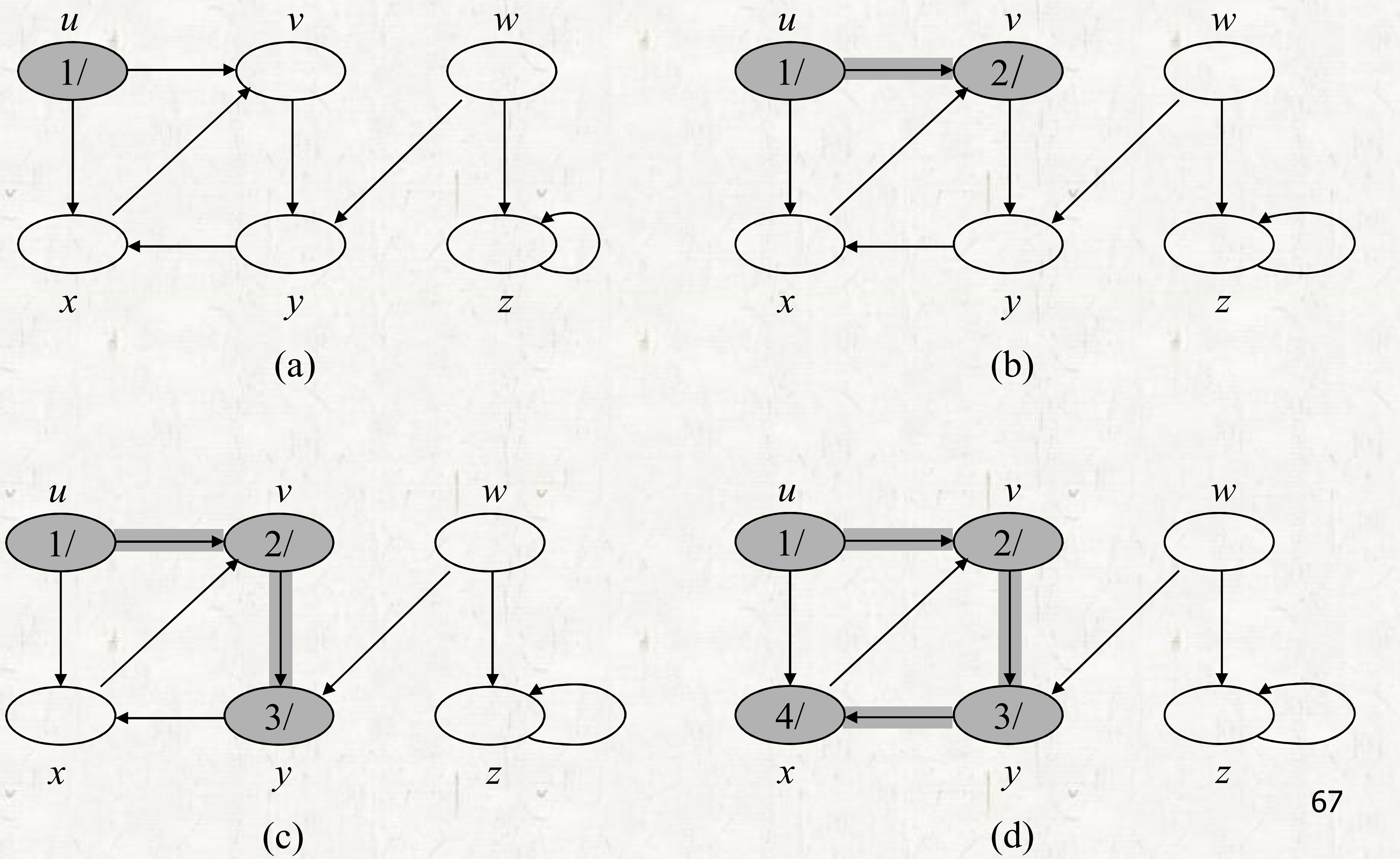


# Depth-first search

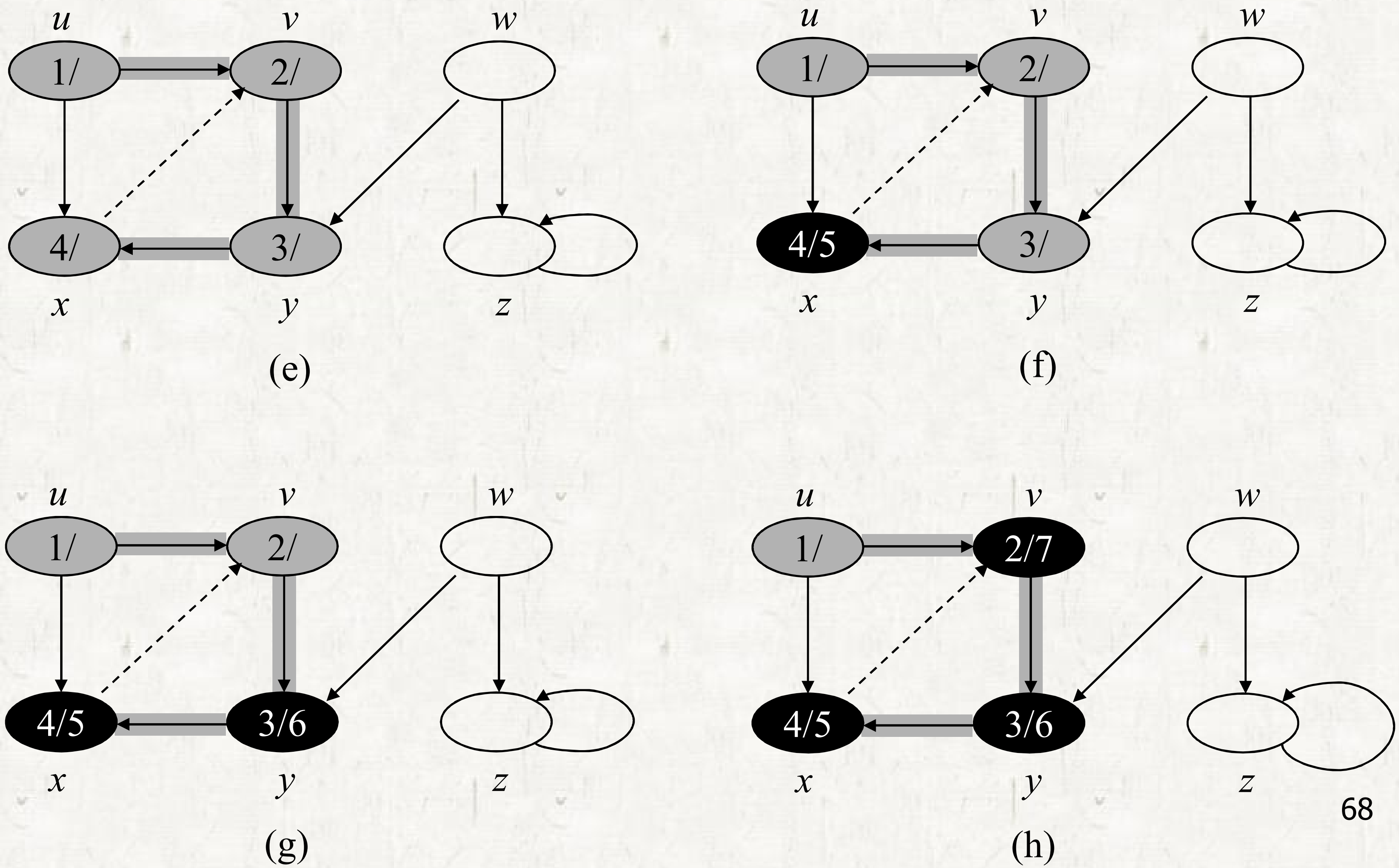
- ***Timestamps***
  - Each vertex *v* has two timestamps.
    - *v.d*: ***discovery time*** (when *v* is grayed)
    - *v.f*: ***finishing time*** (when *v* is blacken)



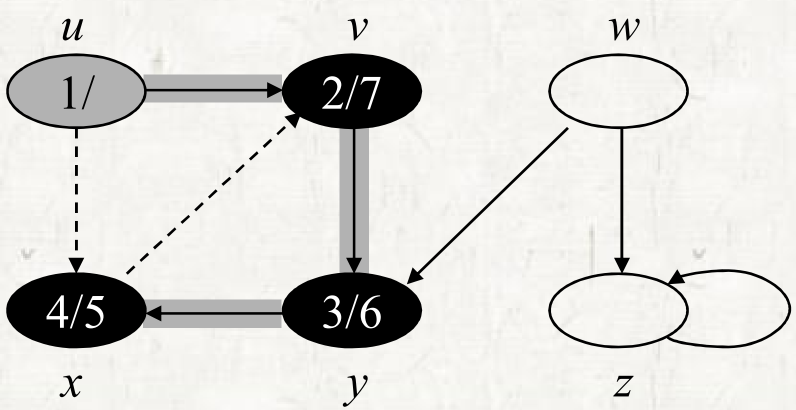
# Depth-first search



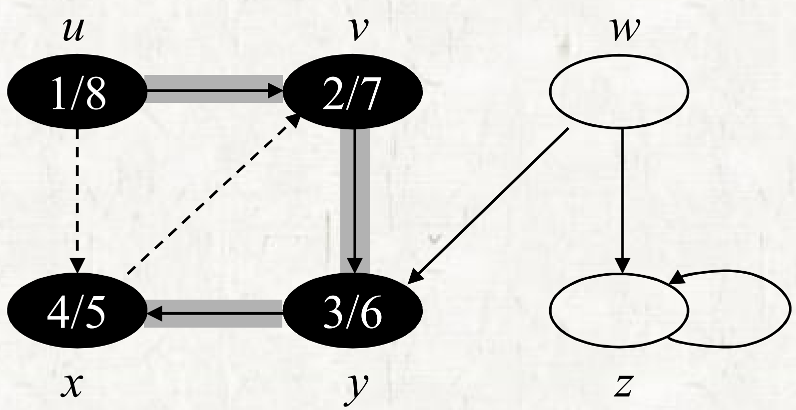
# Depth-first search



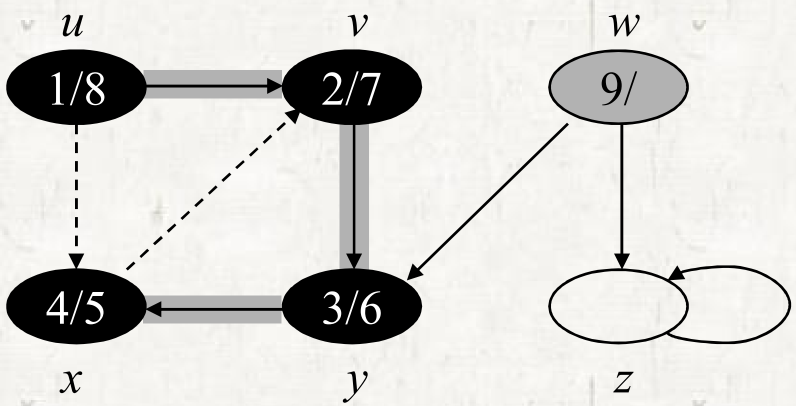
# Depth-first search



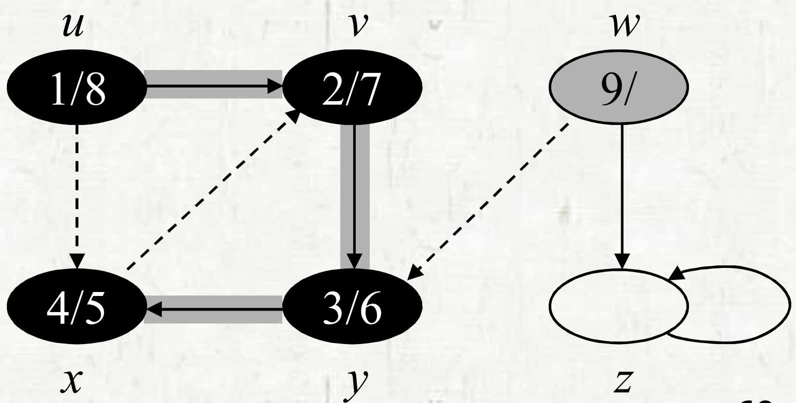

(i)




(j)

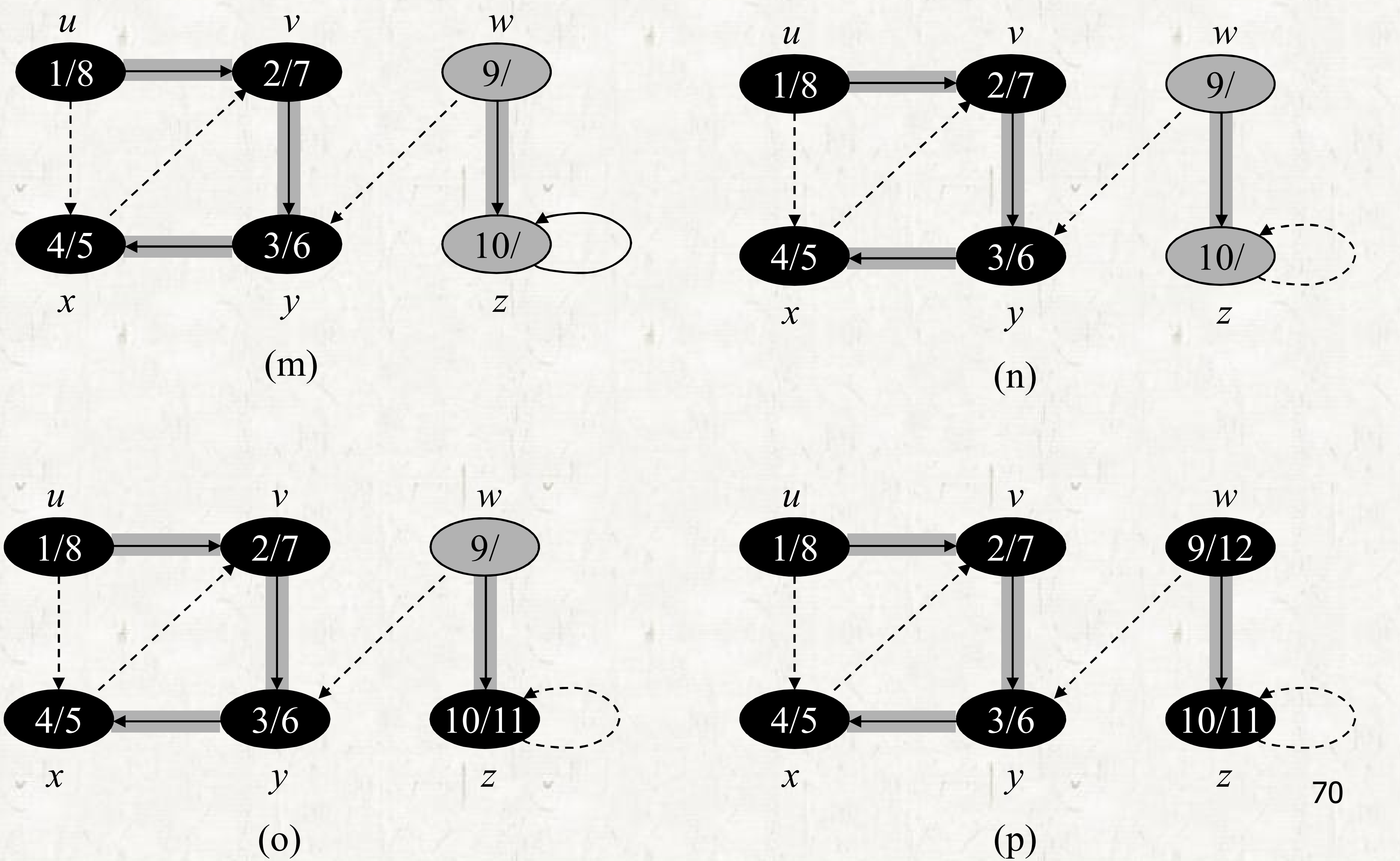



(k)



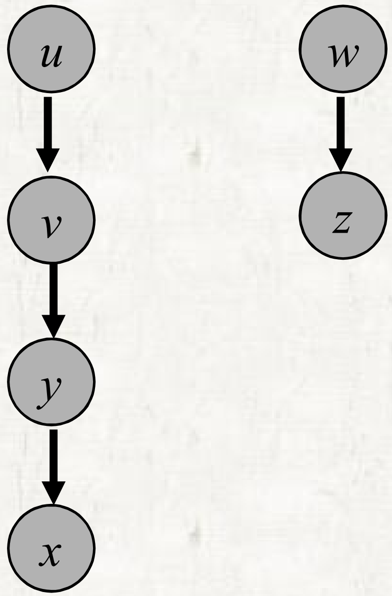

(l)

# Depth-first search



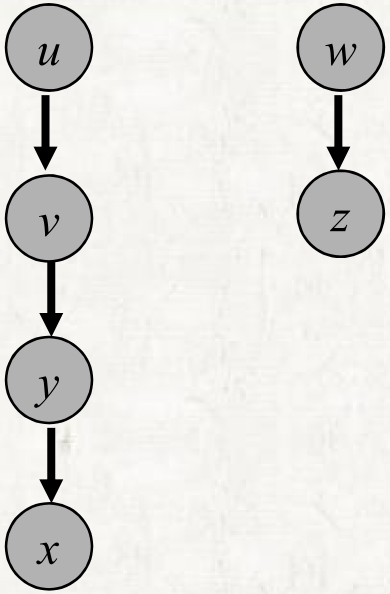
# Depth-first search

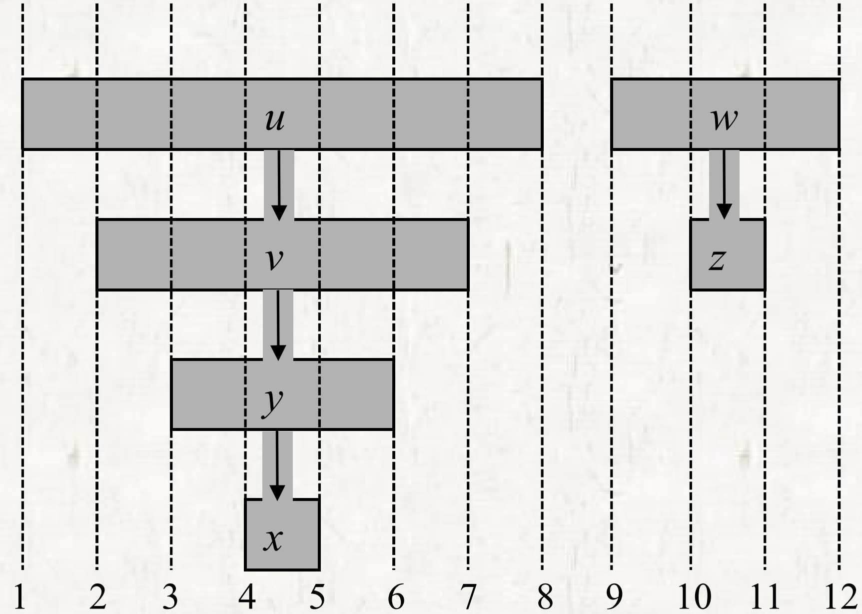
- The ***predecessor subgraph*** is a ***depth-first forest***.



# Depth-first search

- ***Parenthesis theorem (for gray interval)***
  - ***Inclusion***: The ancestor's includes the descendants'.
  - ***Disjoint***: Otherwise.

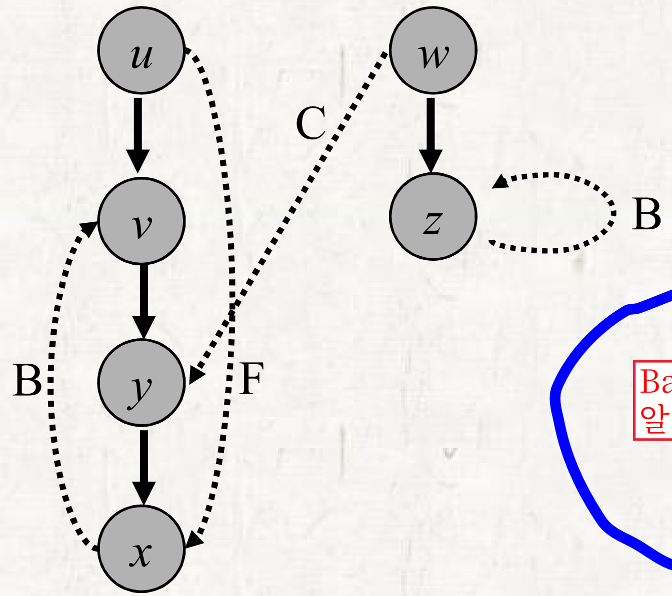


# Depth-first search

## Classification of edges

- *Tree edges*
- *Back edges*
- *Forward edges*
- *Cross edges*




Back edge로 하면
알고리즘이 편해짐

# Depth-first search

## Classification of edges

- ***Tree edges***: Edges in the depth-first forest.

White edge를 방문할때

- ***Back edges***: Those edges ($u$, $v$) connecting a vertex $u$ to an ancestor $v$ in a depth-first tree. Self-loops are considered to be back edges.

interval이 포함되고 finishing time이 많은 쪽으로.

- ***Forward edges***: Those edges ($u$, $v$) connecting a vertex $u$ to a descendant $v$ in a depth-first tree.

Black edge로 갈때 interval이 포함 될 때

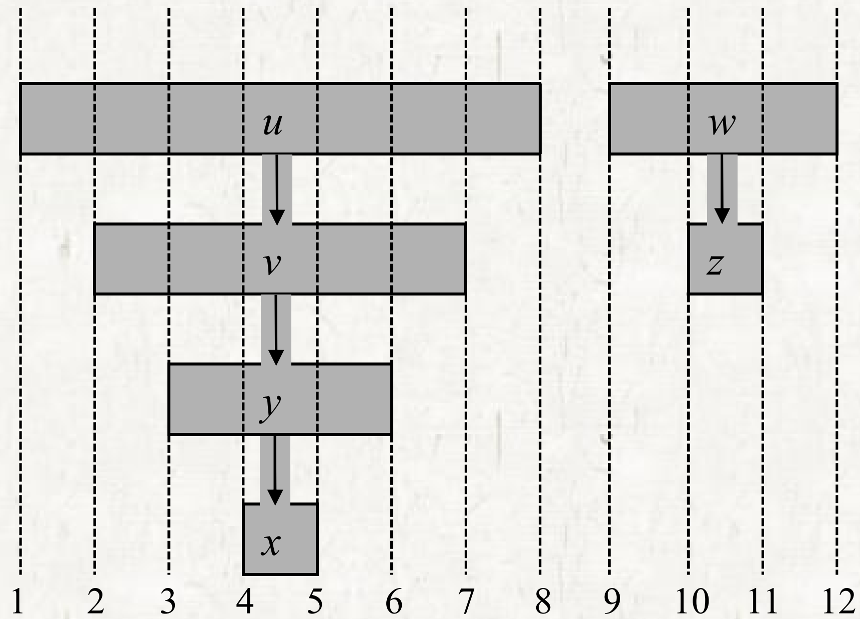
- ***Cross edges***: All other edges. They can go between vertices in the same depth-first tree, as long as one vertex is not an ancestor of the other, or they can go between vertices in different depth-first trees.

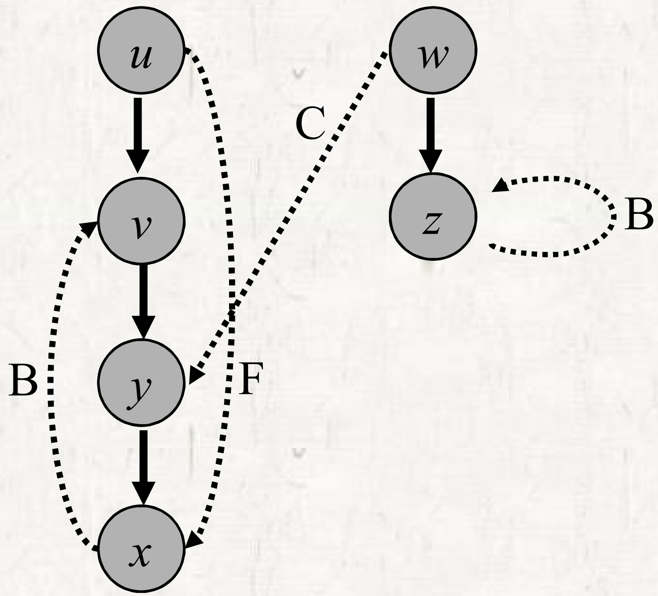
Black edge로 갈때 interval이 포함 되지 않을 때

# Depth-first search

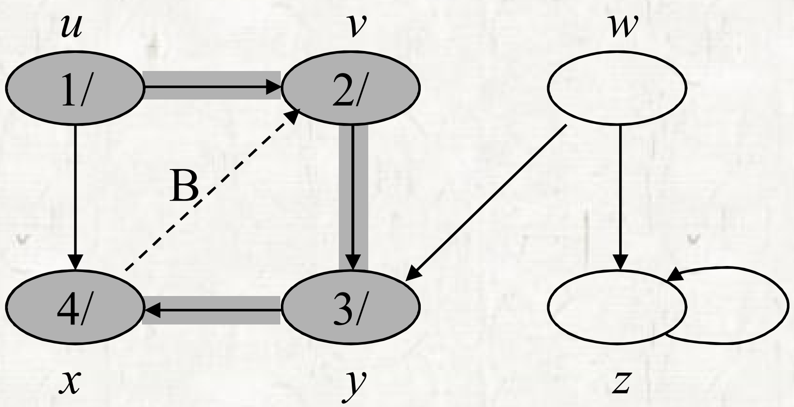
- **Classification by the DFS algorithm**
  - Each edge $(u, v)$ can be classified by the color of the vertex $v$ that is reached when the edge is first explored:
    - ***white*** indicates a tree edge,
    - ***gray*** indicates a back edge, and
    - ***black*** indicates a forward or cross edge.
  - Forward and cross edges are classified by ***the inclusion of gray intervals of u and v***.

# Depth-first search

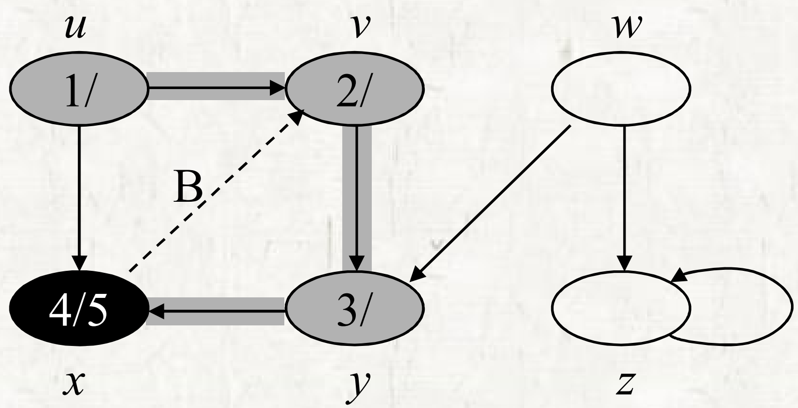


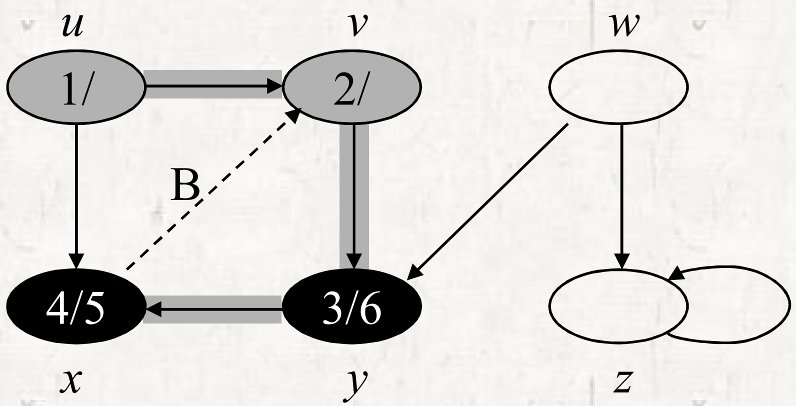
# Depth-first search



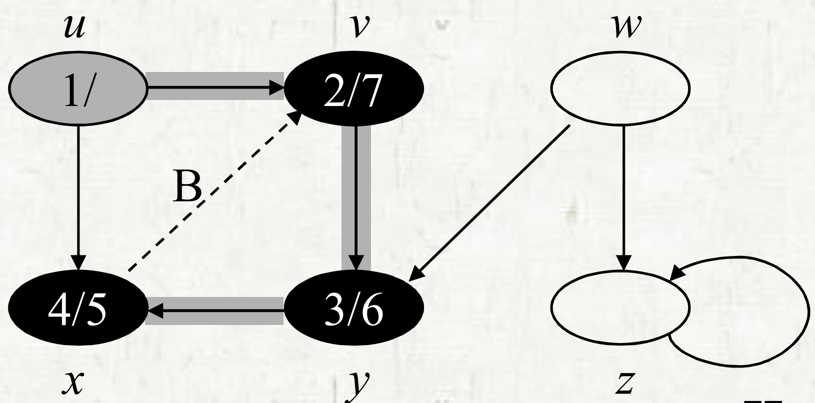

(e)




(f)

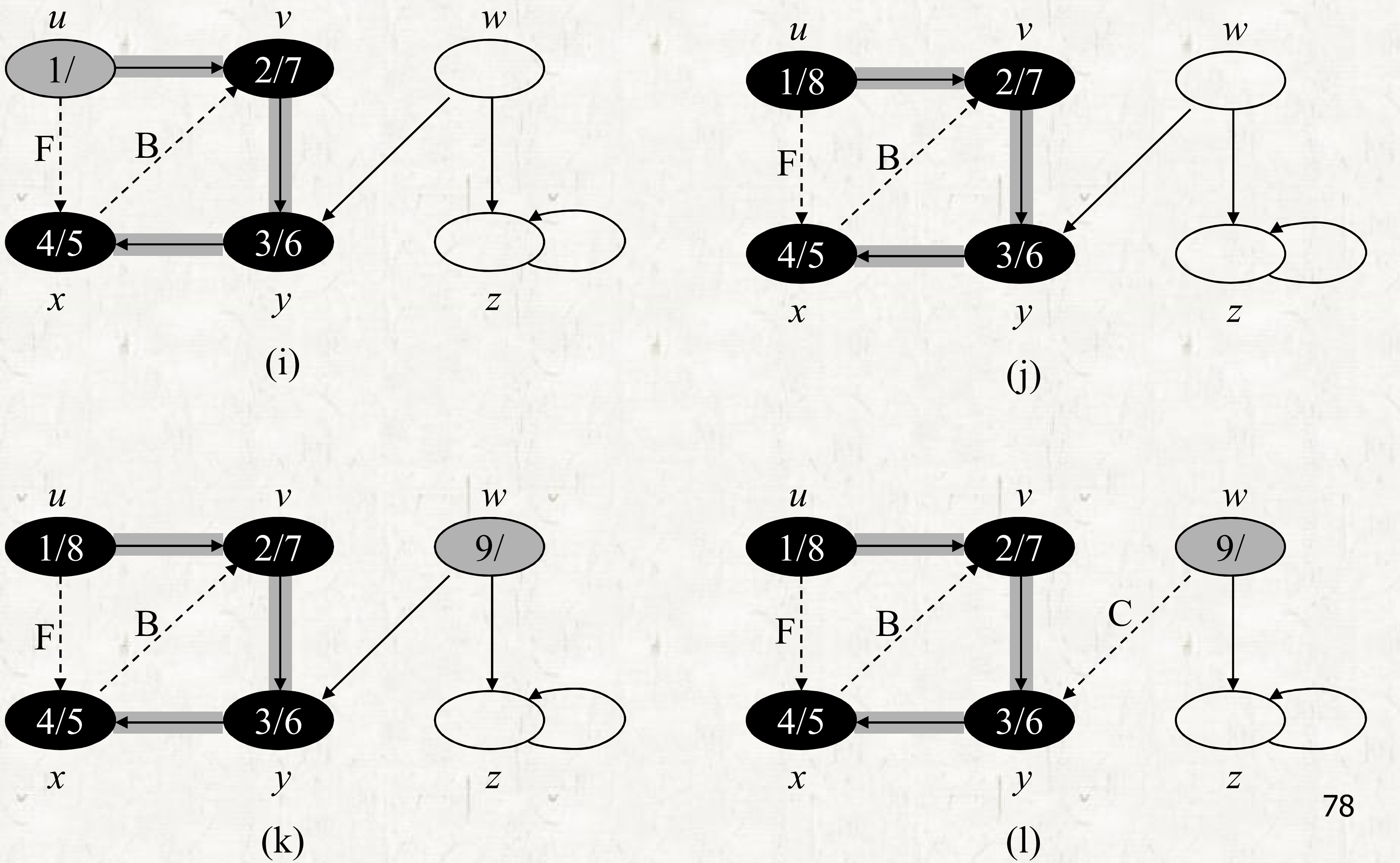



(g)

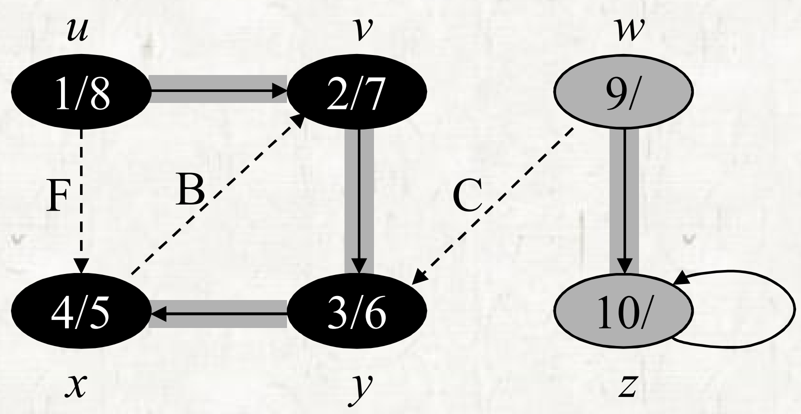



(h)

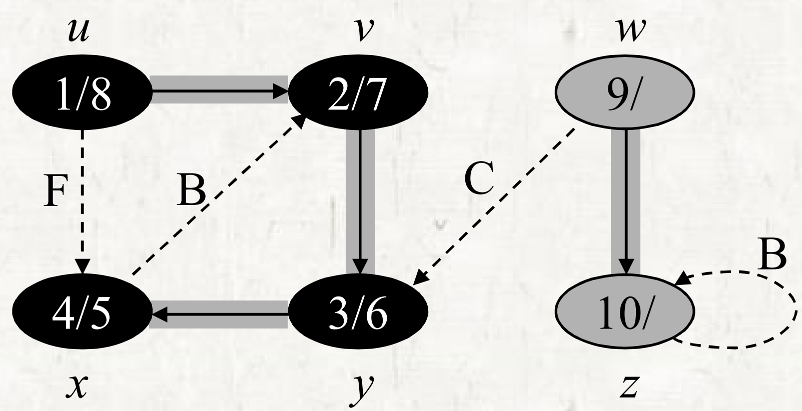
# Depth-first search



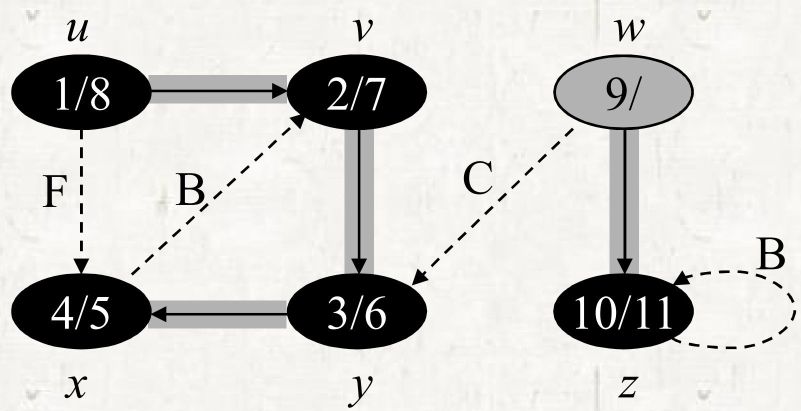
# Depth-first search



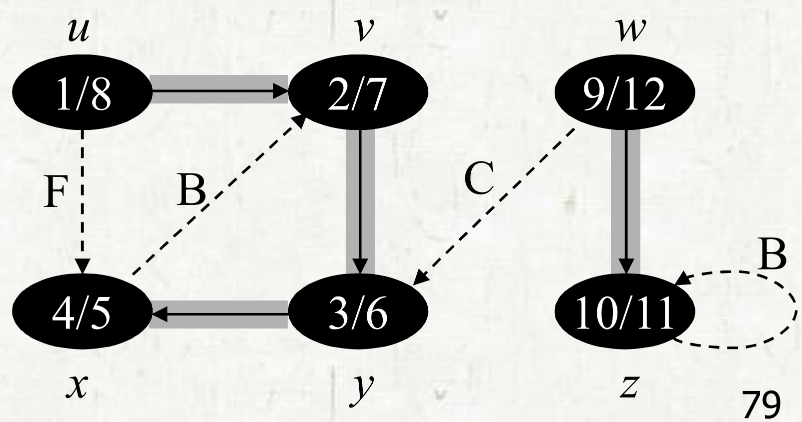

(m)




(n)




(o)




(p)

# Depth-first search

- In a depth-first search of an ***undirected graph***, every edge of G is either a ***tree edge*** or a ***back edge***.
  - Forward edge?
  - Cross edge?
- Running Time
  - $\Theta(V+E)$

V는 한번씩 방문,
E는 한번씩 탐색

E는 없고 V만 있을
경우

# Self-study

- **Exercise 22.3-5 (22.3-4 in the 2nd ed.)**
  - Edge classification
- **Problem 22-2 *a-d***
  - Articulation points

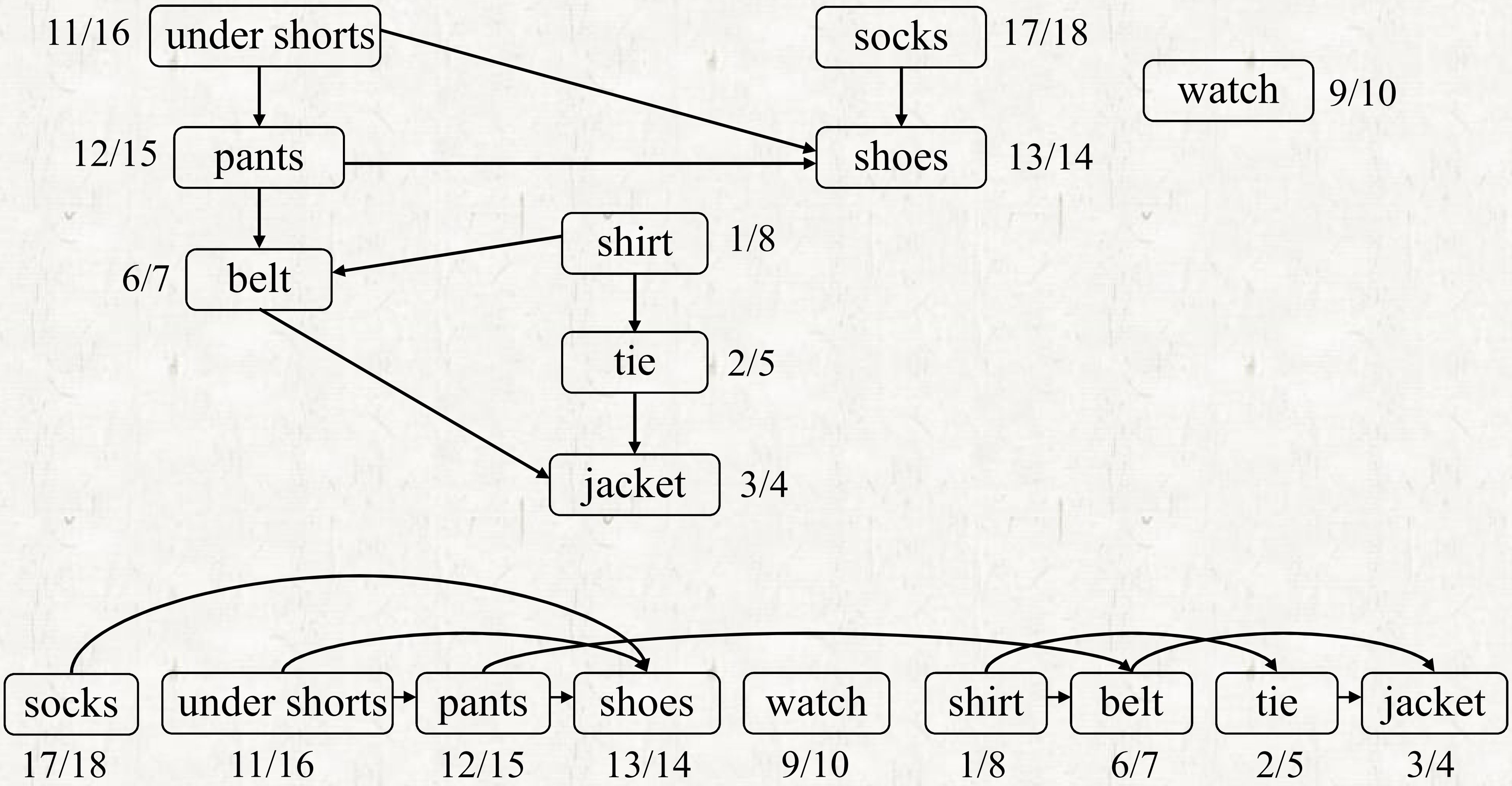
# Contents

- ***Graphs***
  - *Graphs basics*
  - *Graph representation*
- ***Searching a graph***
  - *Breadth-first search*
  - *Depth-first search*
- **Applications of depth-first search**
  - Topological sort

# Topological sort

- ***Definition***
  - Given a DAG (directed acyclic graph), generate a linear ordering of all its vertices such that all edges go from left to right.

# Topological sort






DAG이면 , DFS의 finishing time의 역순.

# Topological sort

- ***Main ideas***
  - Successively place a node from the ***left*** with **0** ***in-degree***.
  - Successively place a node from the ***right*** with **0** ***out-degree***.
  - Run DFS on G and place the nodes from the ***right*** in the ***increasing order of the finishing time***.
  - $\Theta(V+E)$ time

# Topological sort

- ***Correctness***
  - If there is an edge from $u$ to $v$, then $v.f < u.f$.
  - A directed graph $G$ is ***acyclic*** if and only if a depth-first search of G yields ***no back edges***.

finishing time의
역순으로 출력하면
Topological sort.

# Self-study

- **Exercise 22.4-2**
  - Computing the number of simple paths from *s* to *t* in linear time.
- **Exercise 22.4-3**
  - Cycle detection in an undirected graph.
- **Exercise 22.4-5**
  - Another topological sort algorithm.

# Programming Assignment

- ***Depth-first search and its applications***
  - **Exercise 22.3-10 (22.3-9, 2nd ed.) (#1)**
    - Depth-first search with edge classification
  - **Exercise 22.3-12 (22.3-11, 2nd ed.) (#2)**
    - Connected component identification
  - **Topological sort (#3)**
    - The program should detect whether the input is a DAG or not.