CS 2022 : DATA STRUCTURES & ALGORITHMS

Lecture 8: Graphs (Introduction & Searching)

Malaka Walpola

Outline

- Terminology
- Graph Representations
  - Adjacency List
  - Adjacency Matrix
- Searching in Graphs
  - Breath First search (BFS)
  - Depth First search (DFS)

Learning Outcomes

- After successfully studying contents covered in this lecture, students should be able to,
  - explain the terminology used in the graphs
  - explain the different graph representation mechanisms and their characteristics
  - explain graph search strategies and their characteristics

Introduction

Terminology

- Graph G = (V, E)
  - V = set of vertices
  - E = set of edges $\subseteq (V \times V)$
  - $|E|$ - Number of Edges
    - $|E| \le |V|^2$
  - $(u, v) \in E$: vertex **v** is **adjacent** to vertex **u**

Types of Graphs

- Undirected
  - Edges are undirected
- Directed
  - Edges are directed
- Weighted
  - Edges have weights

Terminology Cont…

- Directed Graphs
  - In-degree of a vertex
  - Out- degree of a vertex
- Undirected/Directed Graphs
  - Degree of a vertex
- Path Between Two Vertices
  - Path is simple if no vertex is repeated (no circle)

Terminology Cont…

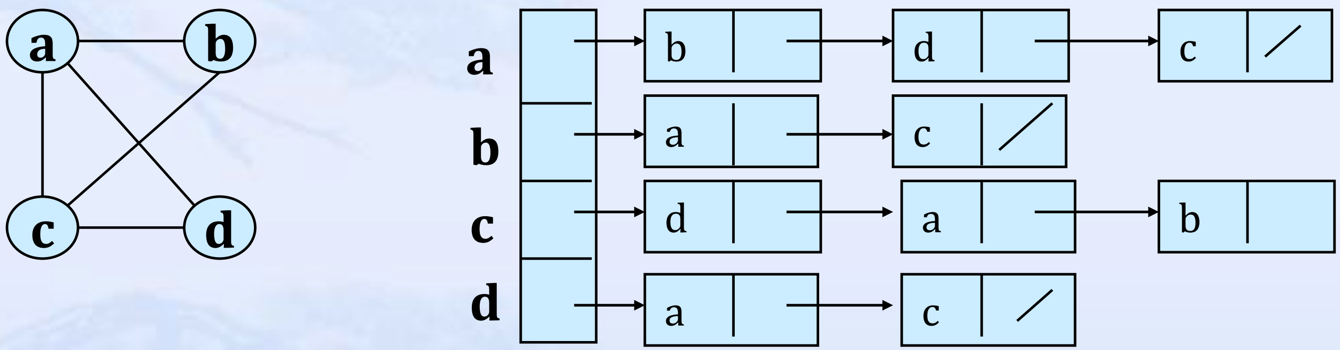
- Path Cost of a Weighted Graph
- Vertex **v** is Reachable from Vertex **u**
  - There is a path from u to v
- Length of a Path
  - Number of edges in path
- Subgraph

Terminology Cont…

- Dense
  - Edges dense ($|E| \approx |V|^2$)
- Sparse
  - Edges sparse ($|E| << |V|^2$)
- Connected/Strongly Connected
  - There is a path between every pair of vertices
  - $|E| \geq |V| - 1$.
  - If $|E| = |V| - 1$, then G is a tree

Graph Representation

- Adjacency Lists



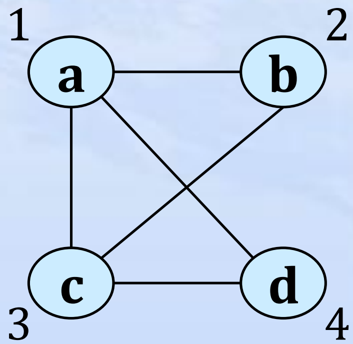
- One List Per Vertex
- For u $\in$ V, Adj[u] Consists of all Vertices Adjacent to u

Adjacency Lists

- Space Requirement $\Theta(V+E)$
  - Space-efficient, when a graph is sparse
- Determining If an Edge $(u,v) \in G$ Is Not Efficient.
  - Have to search in u's adjacency list
    - $\Theta(\text{degree}(u))$ time
    - $\Theta(V)$ in the worst case

Graph Representation

- Adjacency Matrices
  - A - $|V| \times |V|$ Matrix
  - $A[i, j] = a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$



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

Adjacency Matrix

- Space: $\Theta(V^2)$
  - Not memory efficient for sparse graphs
- Time:
  - To list all vertices adjacent to u: $\Theta(V)$
  - To determine if $(u, v) \in E$: $\Theta(1)$
- Can Store Weights Instead of Bits for Weighted Graphs

Searching in Graphs

Searching in Graphs

- Searching a Graph
  - Systematically follow the edges of a graph to visit the vertices of the graph
  - Used to discover the structure of a graph
- Standard Graph-Searching Algorithms
  - Breadth-first Search (BFS)
  - Depth-first Search (DFS)

Breadth-First Search

- Expands the Frontier Between the Discovered and Undiscovered Vertices Uniformly Across the Breadth of the Frontier
  - A vertex is “discovered” the first time it is encountered during the search
  - A vertex is “finished” if all vertices adjacent to it have been discovered

Breadth-First Search

- Colors the vertices to keep track of progress
  - White – Undiscovered
  - Gray – Discovered but not finished
  - Black – Finished

Breadth-First Search

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

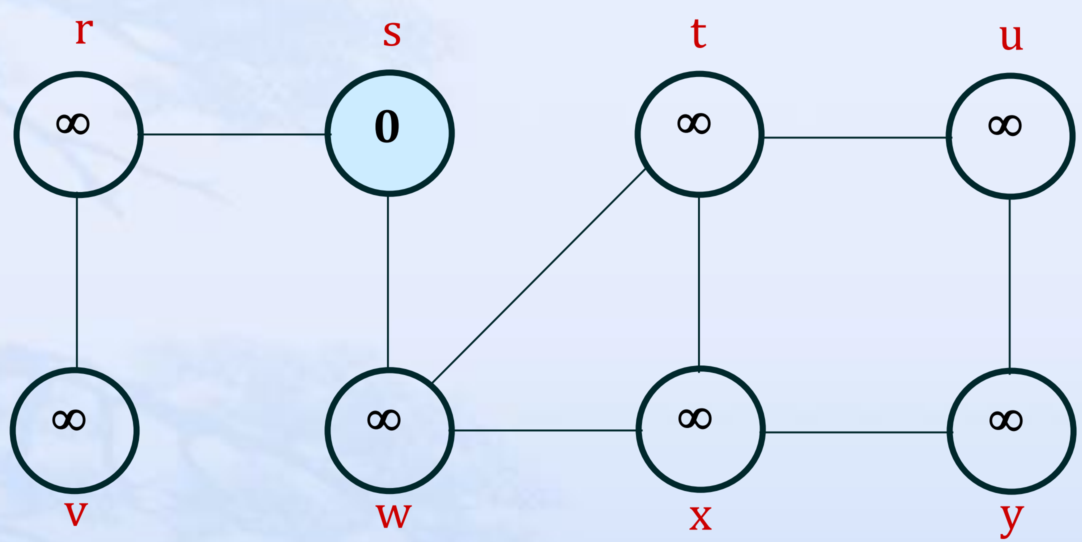
Breadth-First Search

```
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

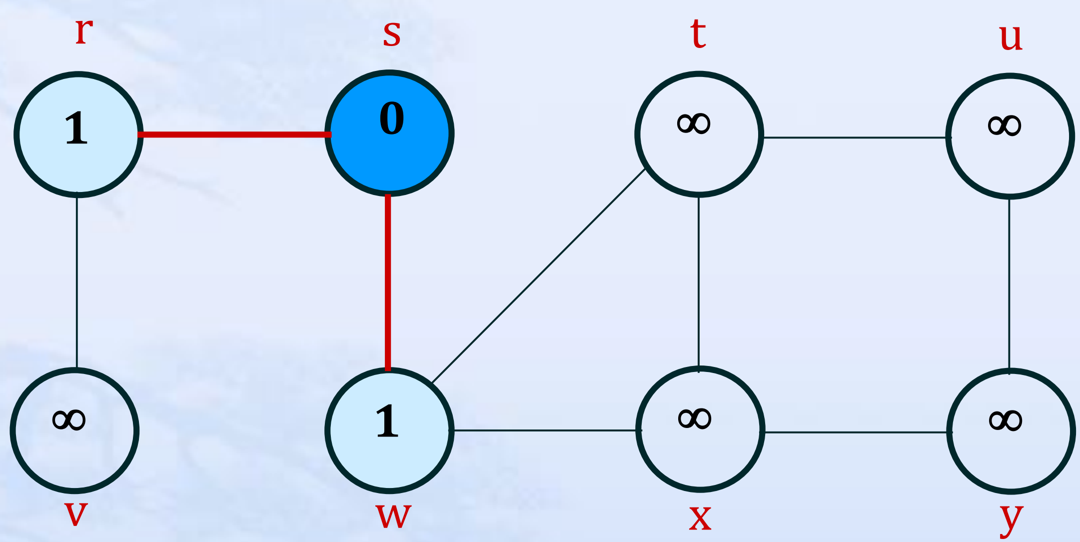
Q: a queue of discovered vertices
color[v]: color of v
d[v]: distance from s to v
π[v]: predecessor of v

BFS Example



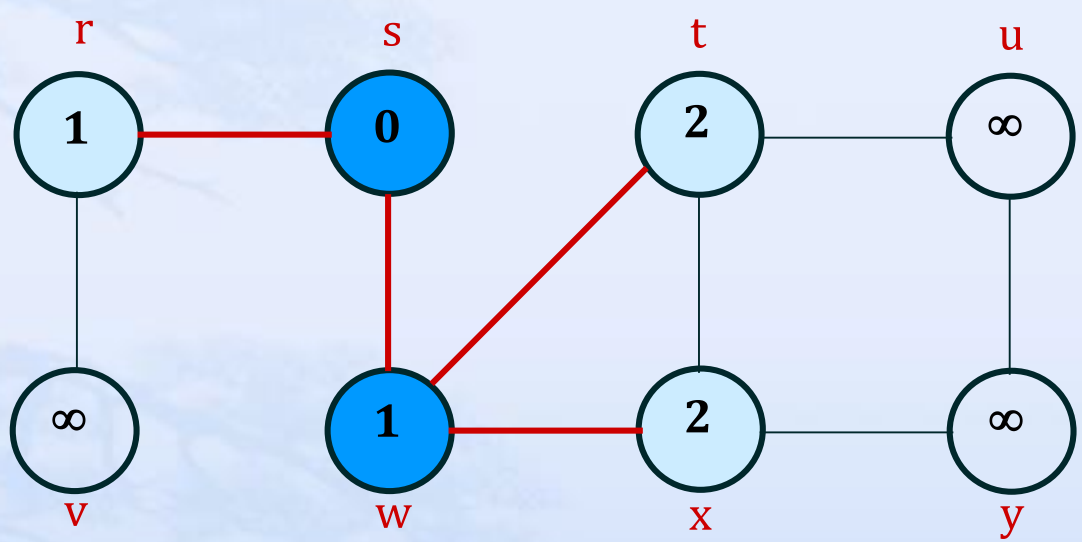
Q: s
0

BFS Example



Q: w r
1 1

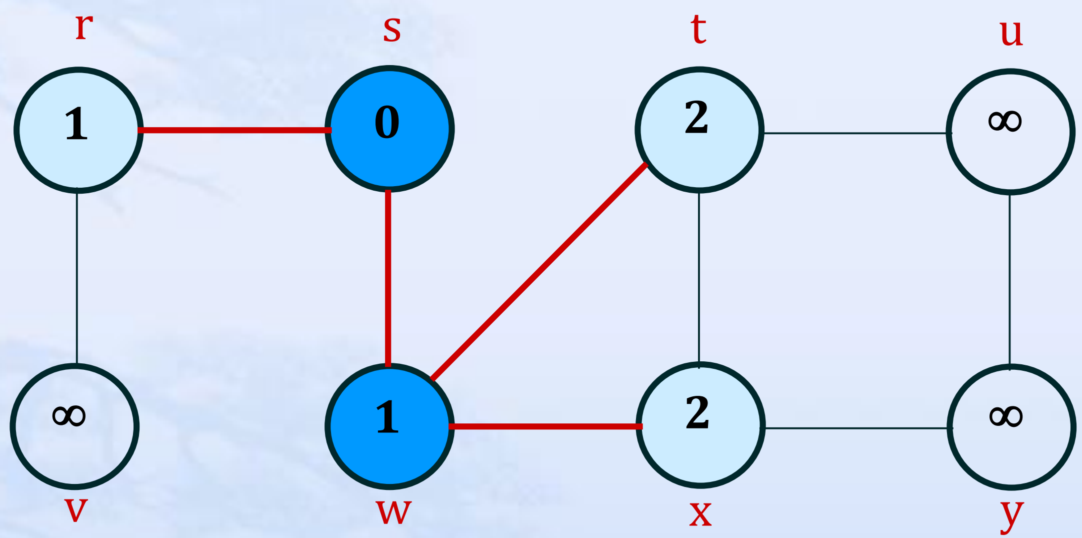
BFS Example



Q: r t x

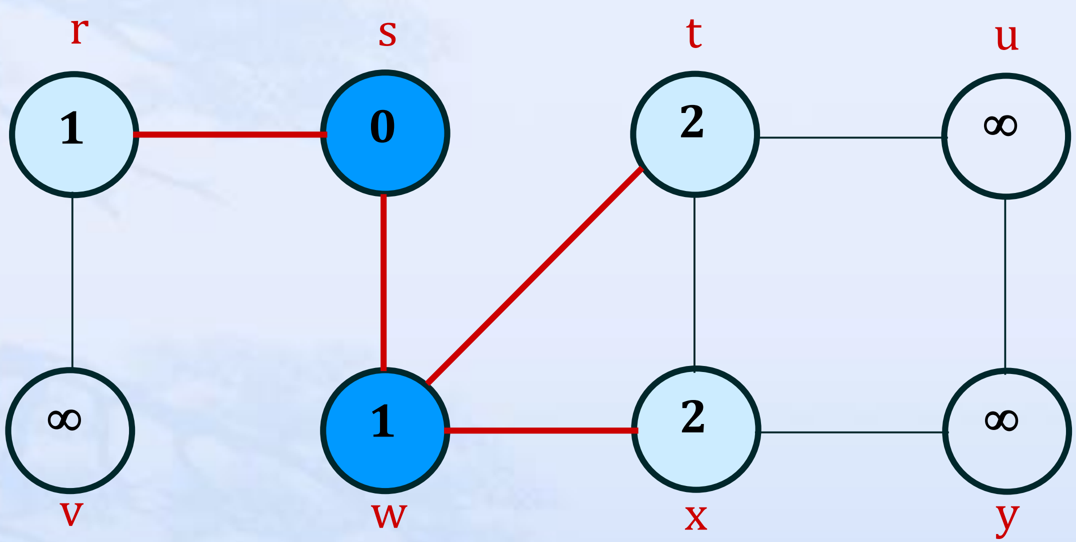
1 2 2

BFS Example



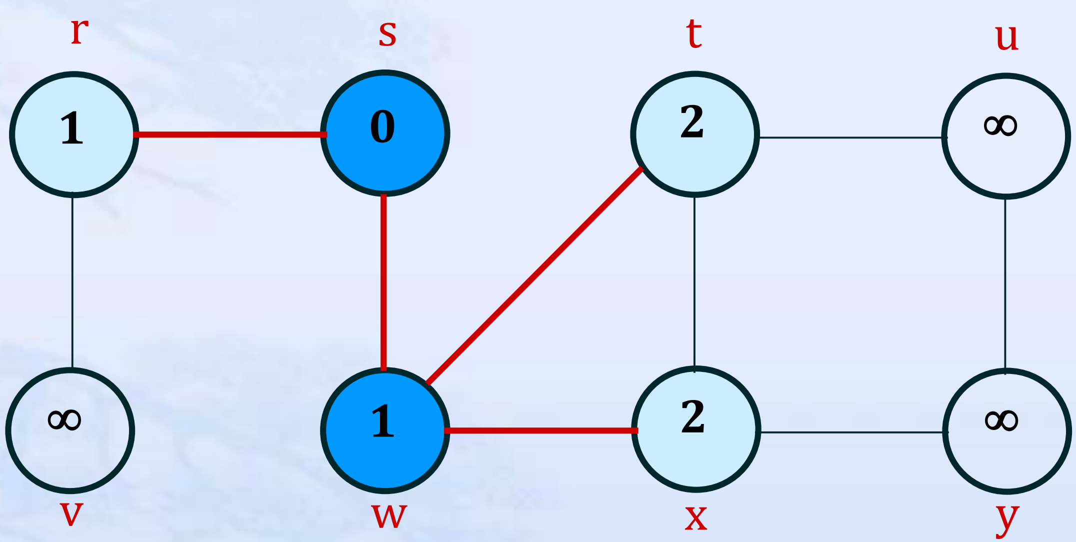
Q: r t x
1 2 2

BFS Example



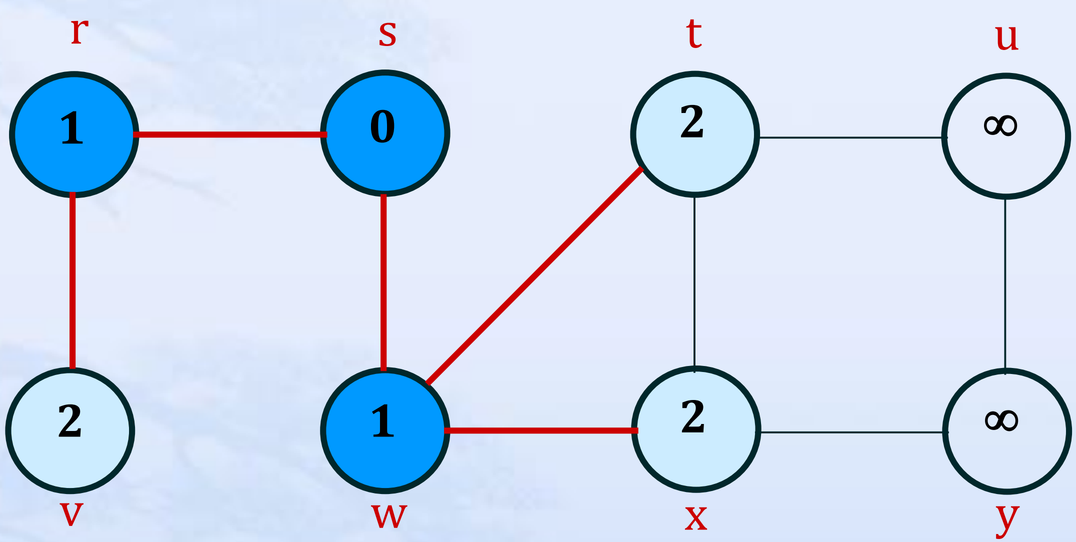
Q:	r	t	x
	1	2	2

BFS Example



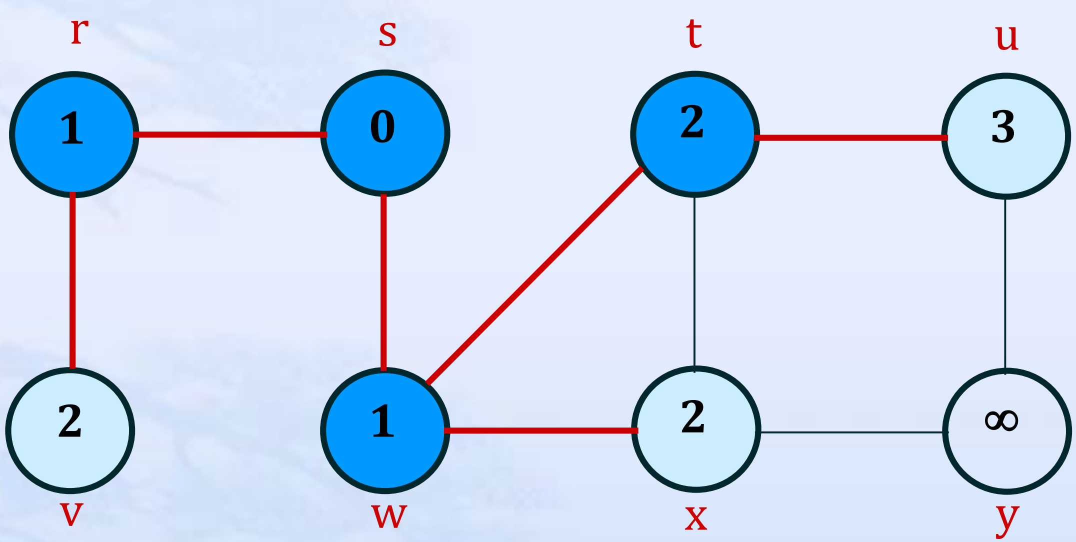
Q: r t x
1 2 2

BFS Example



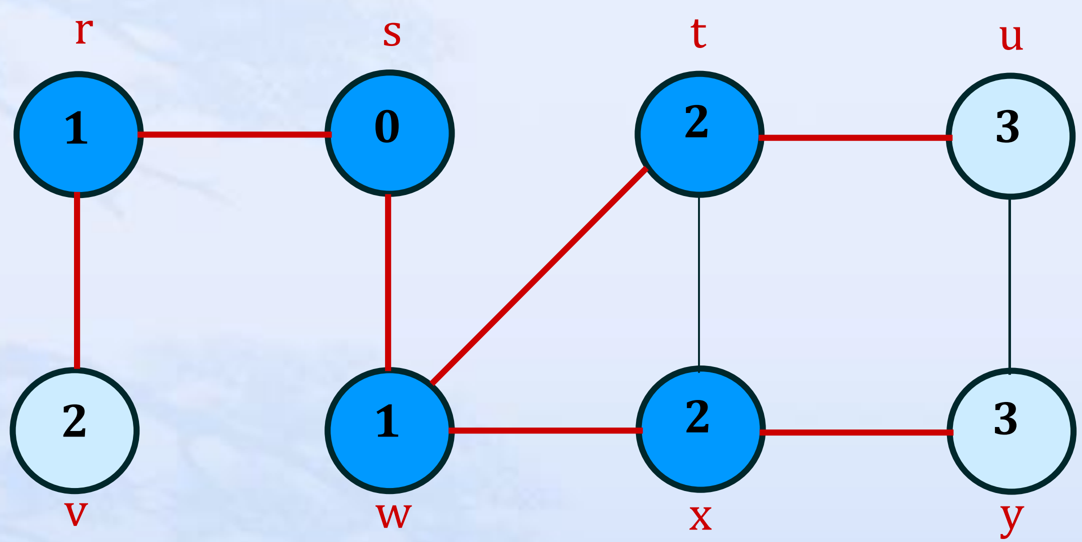
Q: t x v
2 2 2

BFS Example



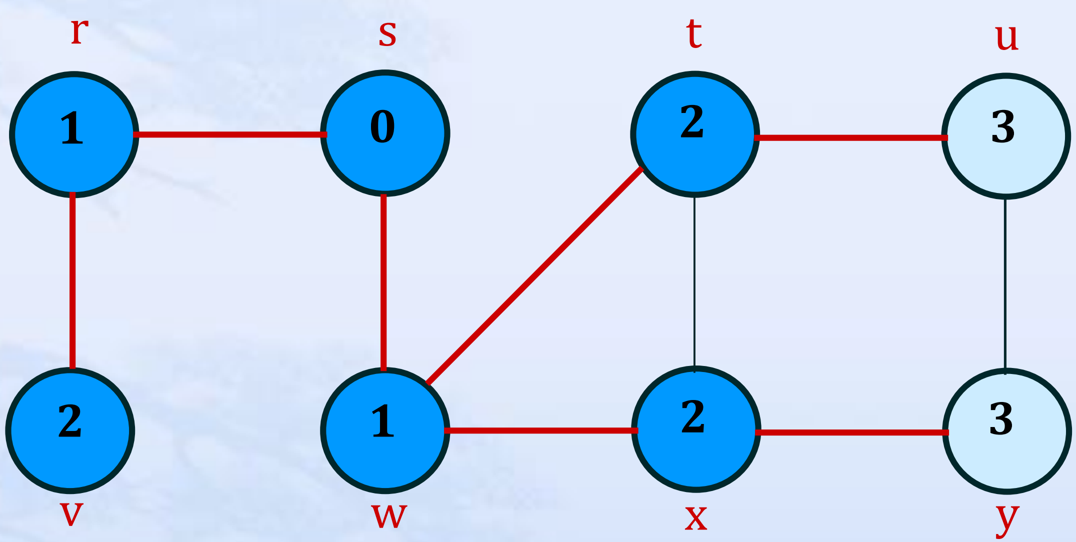
Q: x v u
2 2 3

BFS Example



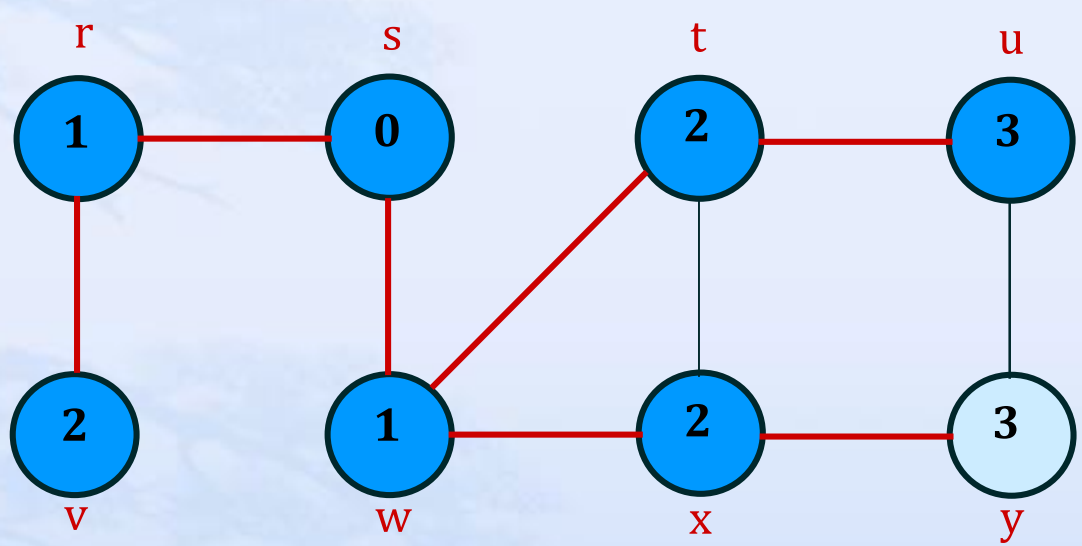
Q:	v u y
	2 3 3

BFS Example



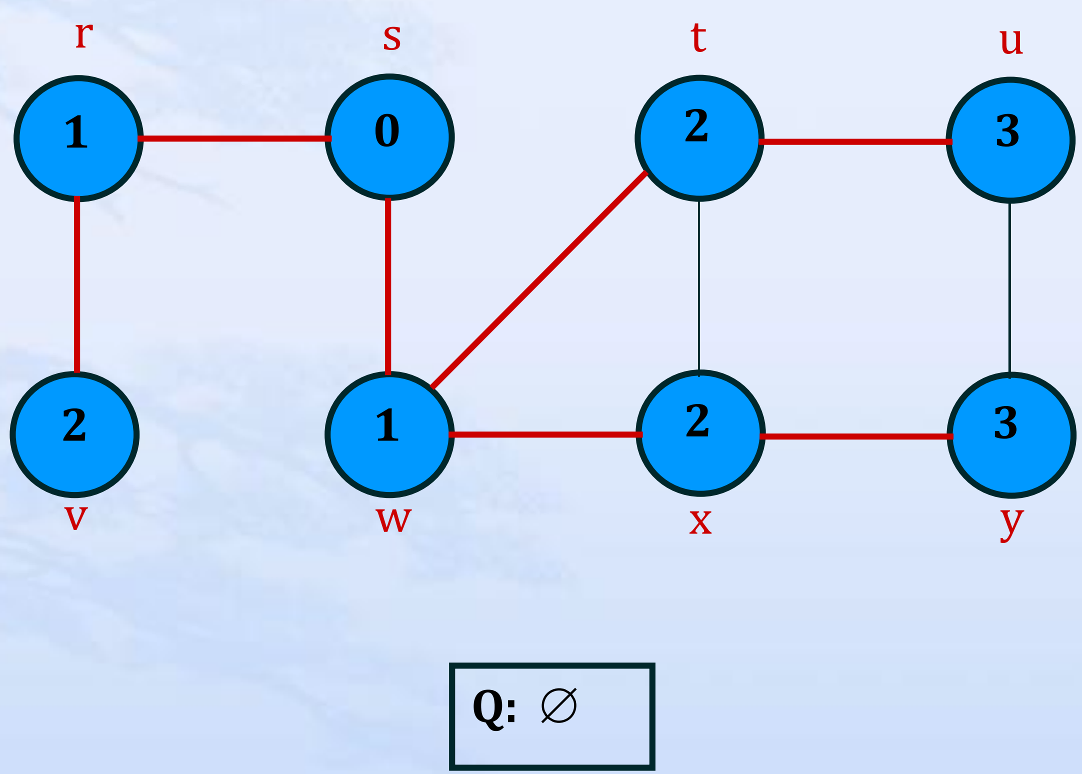
Q: u y
3 3

BFS Example



Q: y
3

BFS Example



Analysis of BFS

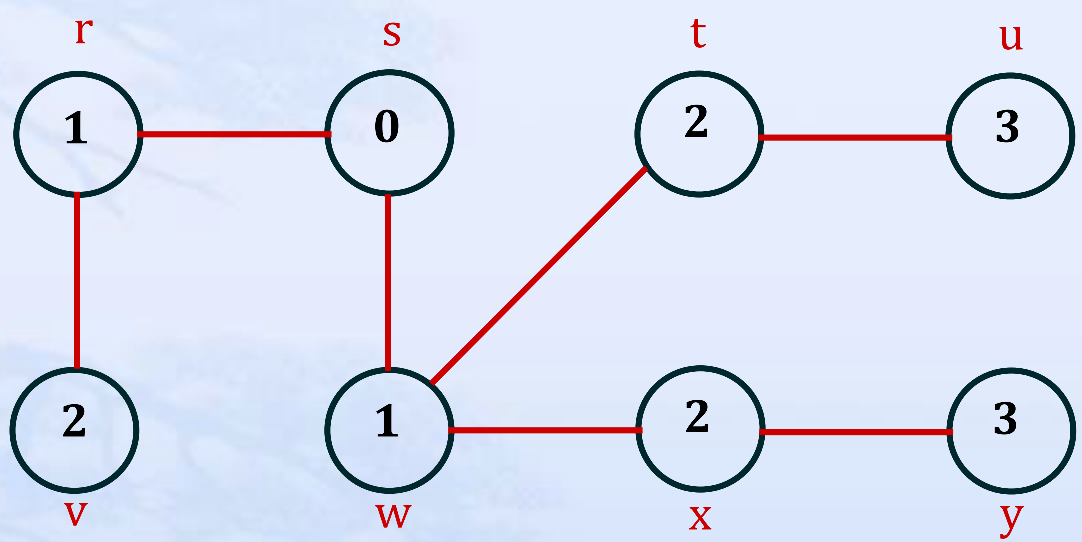
- Initialization takes O(V)
- Traversal Loop
  - After initialization, each vertex is enqueued and dequeued at most once, and each operation takes O(1)
    - So, total time for queuing is O(V)
  - The adjacency list of each vertex is scanned at most once
    - The sum of lengths of all adjacency lists 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

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
  - $V_\pi = \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}$
  - $E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$
- 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**

  $|E_\pi| = |V_\pi| - 1$

BF Tree Example



BF Tree

Depth-first Search

- Idea
  - Search as deep as possible first
- Mechanism
  - Explore edges out of the most recently discovered vertex **v**
  - When all edges of **v** have been explored, backtrack to explore other edges leaving the vertex from which **v** was discovered (its predecessor)

Depth-first Search

- Mechanism Cont…
  - Continue until all vertices reachable from the original source are discovered
  - If any undiscovered vertices remain, then one of them is chosen as a new source and search is repeated from that source

Depth-first Search

- **Input:** G = (V, E), 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 tuns from gray to black)
  - π[v] : predecessor of v = u, such that v was discovered during the scan of u's adjacency list
- Uses the Same Coloring Scheme for Vertices as BFS

Depth-first Search

DFS(*G*)

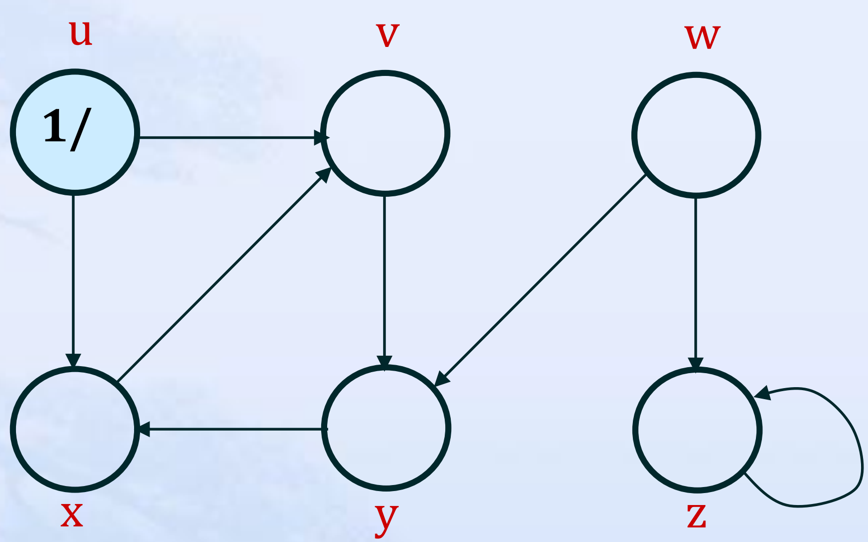
```
for each vertex u ∈ V[G]
    do color[u] ← white
       π[u] ← NIL
time ← 0
for each vertex u ∈ V[G]
    do if color[u] = white
       then DFS-Visit(u)
```

Uses a global timestamp ***time***.

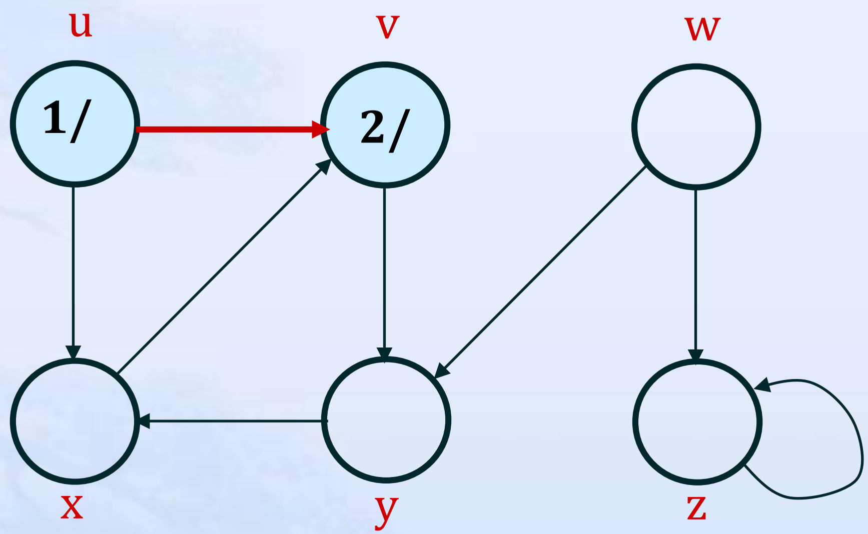
DFS-Visit(*u*)

```
color[u] ← GRAY   ∇ White vertex u has been discovered
time ← time + 1
d[u] ← time
for each v ∈ Adj[u]
    do if color[v] = WHITE
          then π[v] ← u
               DFS-Visit(v)
color[u] ← BLACK      ∇ Blacken u; it is finished.
time ← time + 1
f[u] ← time
```

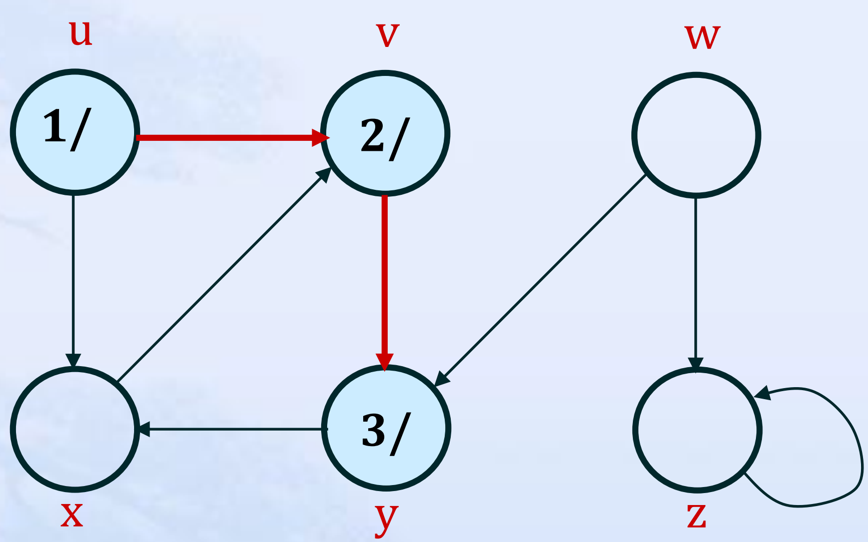
DFS Example



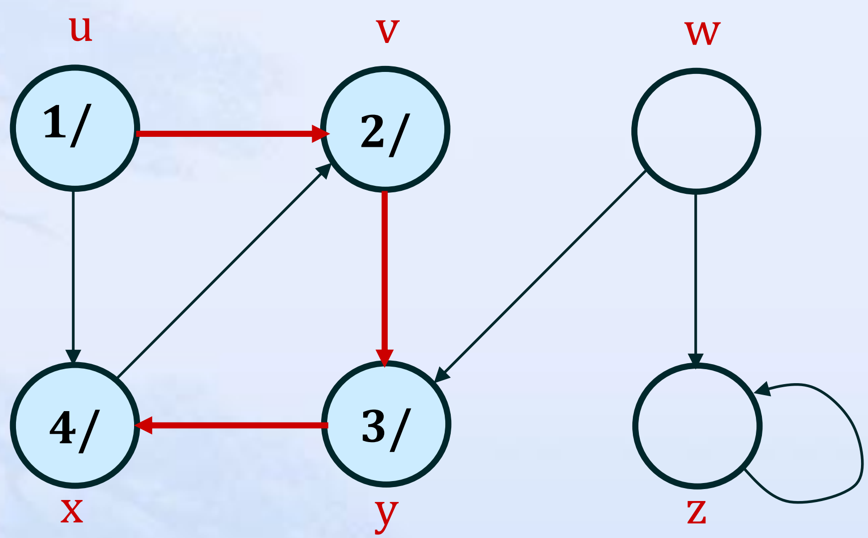
DFS Example



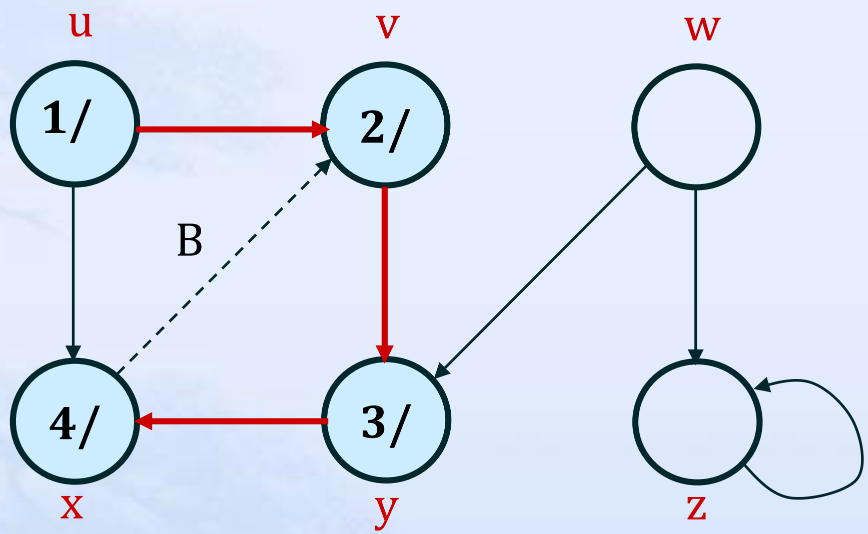
DFS Example



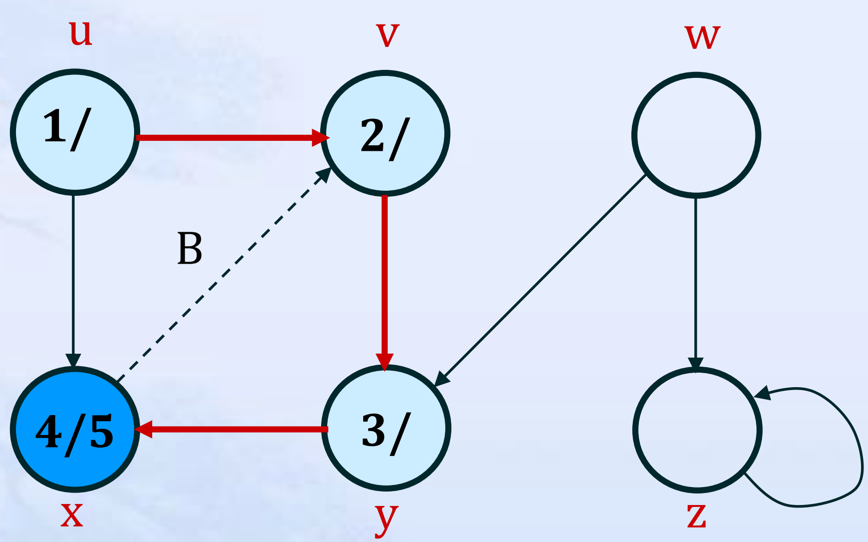
DFS Example



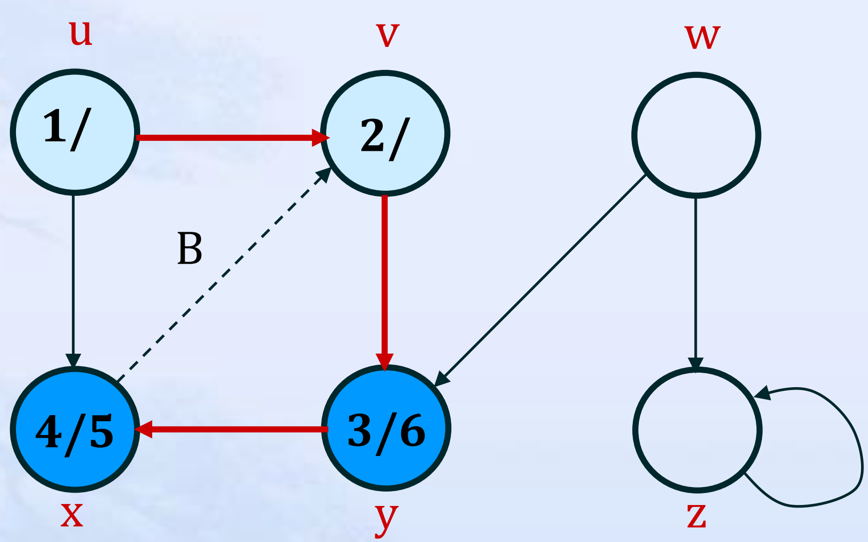
DFS Example



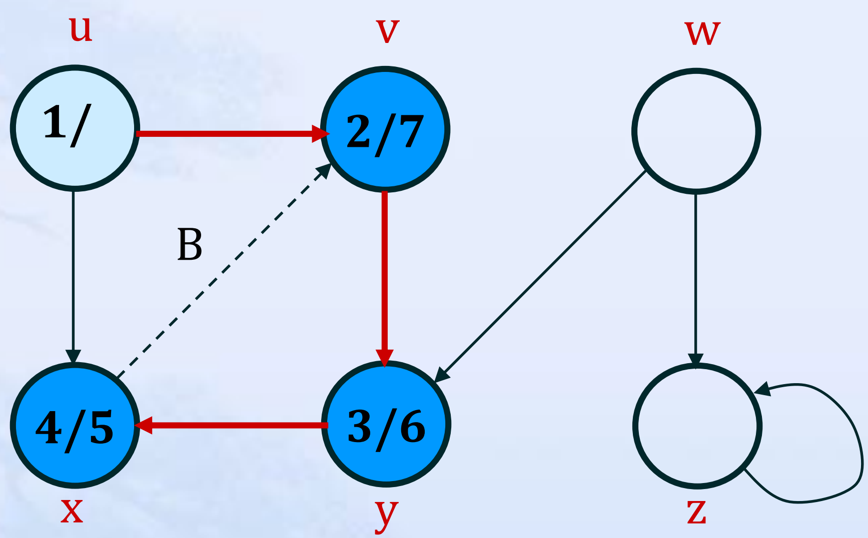
DFS Example



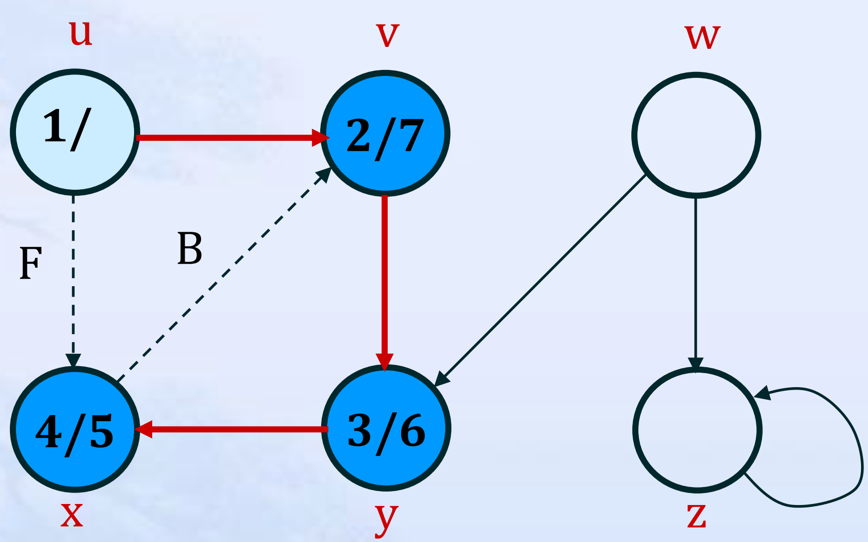
DFS Example



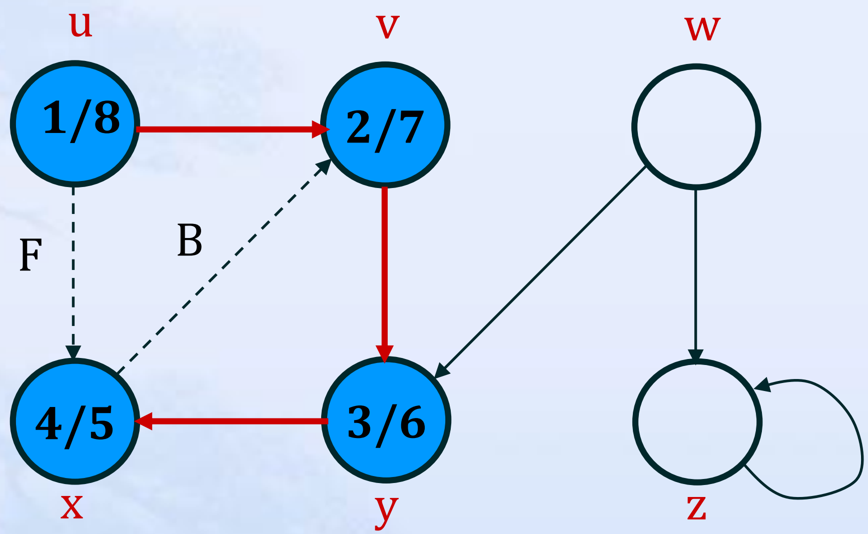
DFS Example



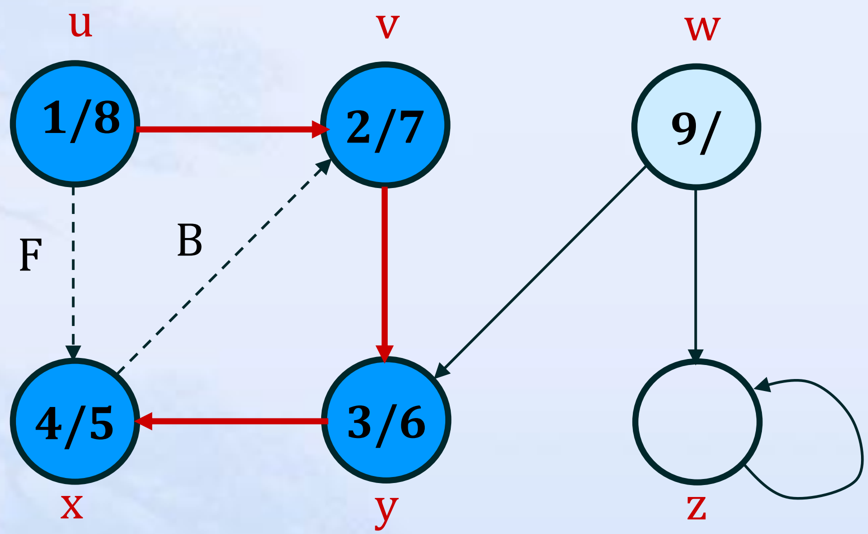
DFS Example



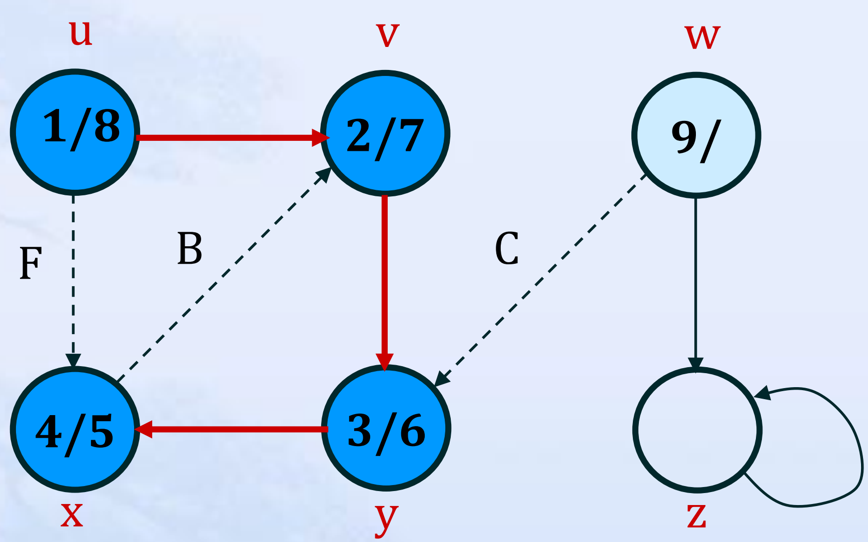
DFS Example



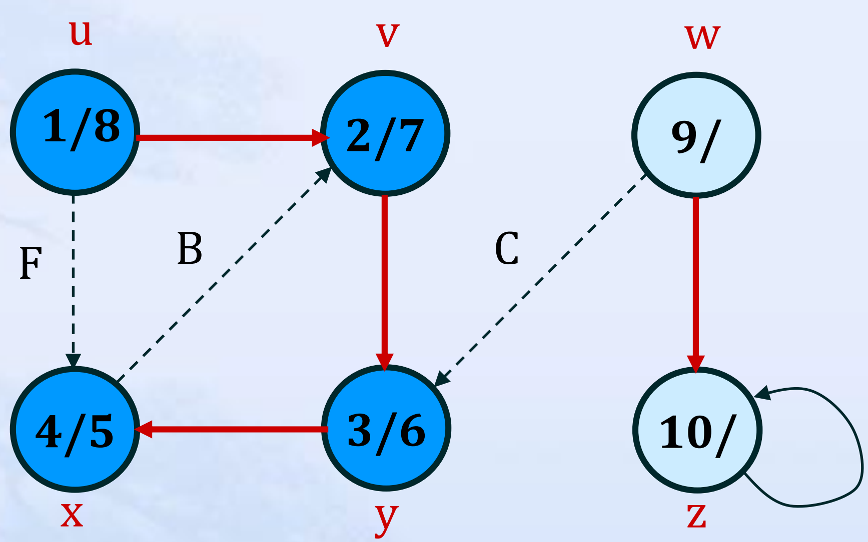
DFS Example



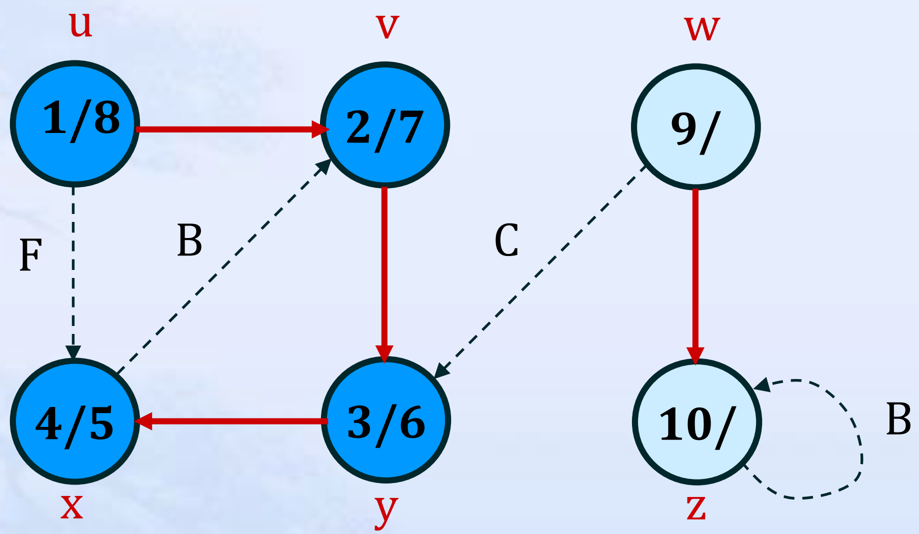
DFS Example



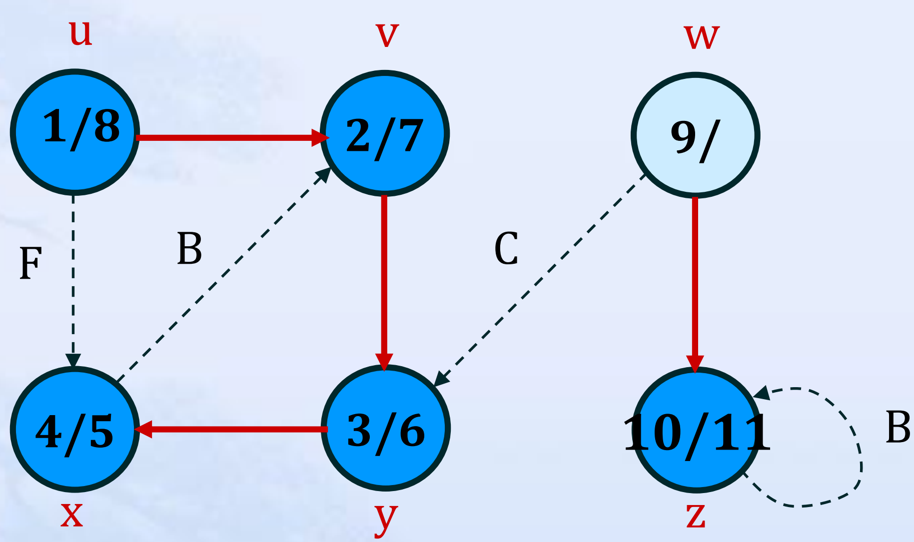
DFS Example



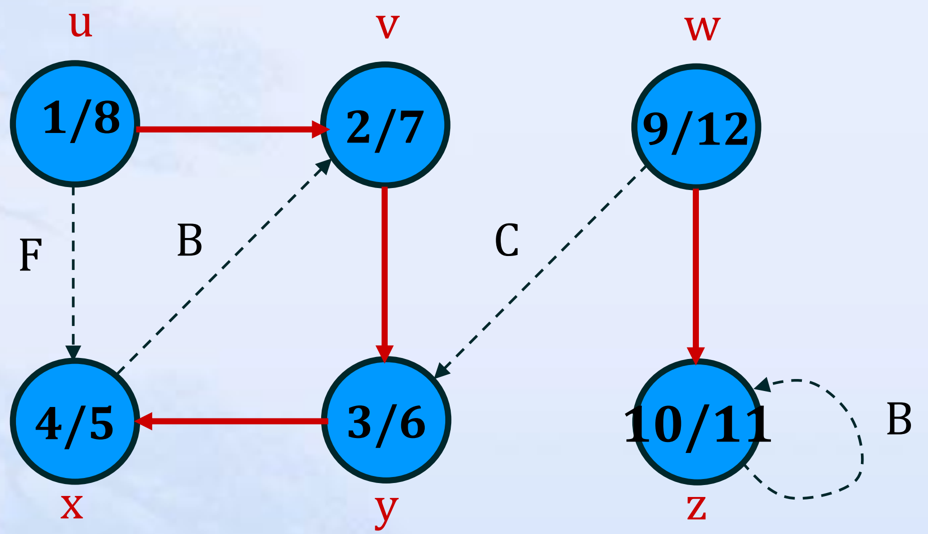
DFS Example



DFS Example



DFS Example



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 $\Theta(V+E)$

Parenthesis Theorem

Theorem 22.7

For all u, v, exactly one of the following holds:

1. $d[u] < f[u] < d[v] < f[v]$ or $d[v] < f[v] < d[u] < f[u]$ and neither u nor v is a descendant of the other.
2. $d[u] < d[v] < f[v] < f[u]$ and v is a descendant of u.
3. $d[v] < d[u] < f[u] < f[v]$ and u is a descendant of v.

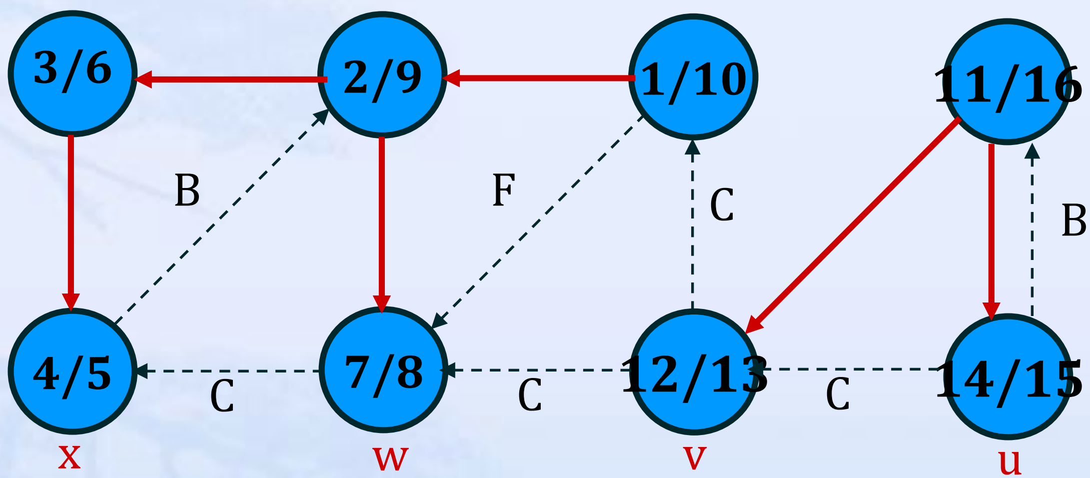
So $d[u] < d[v] < f[u] < f[v]$ cannot happen.

- Like parentheses:
  - OK: ( ) [ ] ([ ]) [( )]
  - Not OK: ([ )] [( ])

Corollary

v is a proper descendant of u if and only if $d[u] < d[v] < f[v] < f[u]$

Parenthesis Theorem Example



(s (z (y (x x) y) (w w) z) s) (t (v v) (u u) t)

Depth-First Forest (Trees)

- The Predecessor Subgraph of DFS is $G_\pi = (V, E_\pi)$ where $E_\pi = \{(\pi[v], v) : v \in V \text{ and } \pi[v] \neq \text{NIL}\}$
  - How does it differ from that of BFS?
    - *Depth-first forest* composed of several *depth-first trees*
  - The edges in E_π are called *tree edges*
  - Forest
    - An acyclic graph G that may be disconnected

WHITE-PATH THEOREM

Theorem 22.9

v is a descendant of u if and only if at time $d[u]$, there is a path $u \rightsquigarrow v$ consisting of only white vertices. (Except for u, which was *just* colored gray.)

SELF STUDYING

Self Studying

- Reading Assignment
  - Chapter 22
    - 22.1: Representation of Graphs
    - 22.2: Breadth First Search
    - 22.3: Depth First Search

References

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, *Introduction to Algorithms*, 3rd Ed. Cambridge, MA, MIT Press, 2009.

[2] Lecture slides available at http://www.cs.unc.edu/~plaisted/comp550/19-graph1.ppt

Note: Slides a modified version of [2]