COMP251: Elementary graph algorithms

Giulia Alberini & Jérôme Waldispühl
School of Computer Science
McGill University

Based on (Cormen *et al.*, 2002)

Based on slides from D. Plaisted (UNC)

Announcements

- Assignment 1: Due tonight (no extension)
- Assignment 2: Released next week.
- Midterm: November 1 at 10am on CrowdMark.
- Final: Dec. 12 at 2pm. **In Person!**

Outline

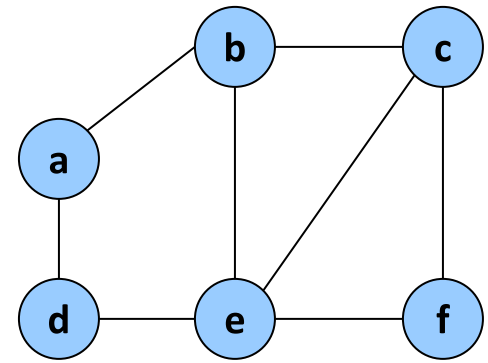
- Vocabulary, definition, and properties of graphs
- Exploring graphs:
  - Breadth First Search (BFS)
  - Depth First Search (DFS)
- Parenthesis theorem

Why DFS & BFS again?

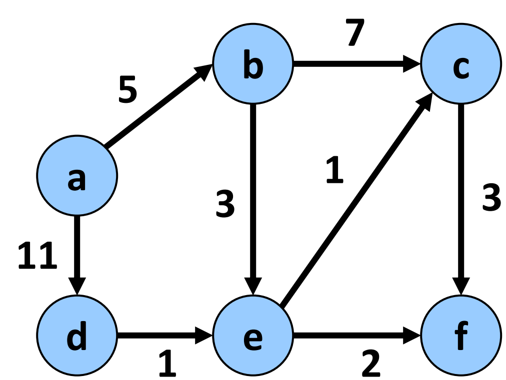
We will cover many algorithms on graphs based on these techniques.

Graphs

- *Graph* $G = (V, E)$
  - V = set of vertices
  - E = set of edges $\subseteq (V \times V)$



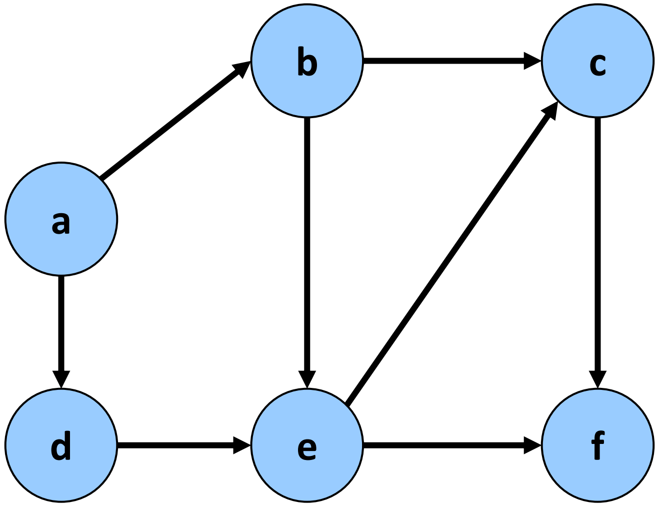
- Types of graphs
  - Undirected: edge $(u, v) = (v, u)$; for all v, $(v, v) \notin E$ (No self loops.)
  - Directed: (u, v) is edge from u to v, denoted as $u \rightarrow v$. Self loops are allowed.
  - Weighted: each edge has an associated weight, given by a weight function $w: E \rightarrow \boldsymbol{R}$.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



Properties

- 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.
  - $|E|$ ▯ $|V| - 1$.
  - Furthermore, if $|E| = |V| - 1$, then G is a tree.

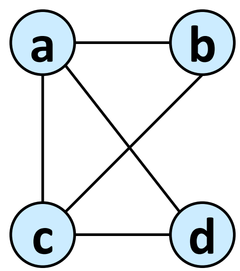
Vocabulary

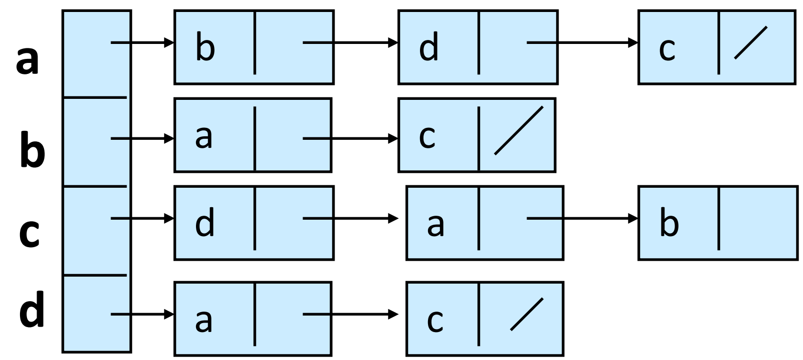


- Ingoing edges of u: $\{\ (v,u)\ \in\ E\ \}$
  e.g. $in(e)\ =\ \{\ (b,e),(d,e)\ \}$
- Outgoing edges of u: $\{\ (u,v) \in\ E\ \}$
  e.g. $out(d)\ =\ \{\ (d,e)\ \}$
- In-degree(u): $|\ in(u)|$
- Out-degree(u): $|\ out(u)\ |$

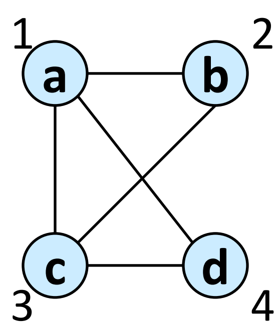
Representation of Graphs

- Two standard ways.
  - Adjacency Lists.





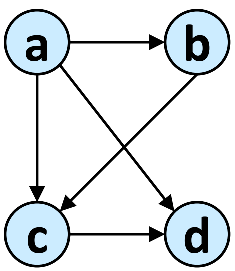
  - Adjacency Matrix.

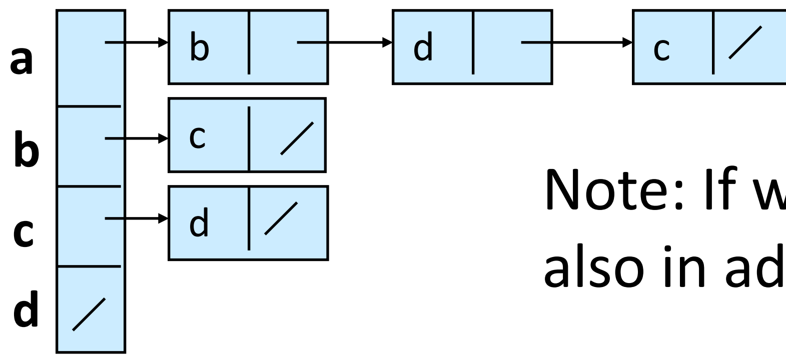


	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

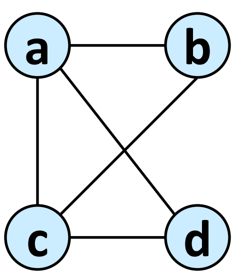
Adjacency Lists

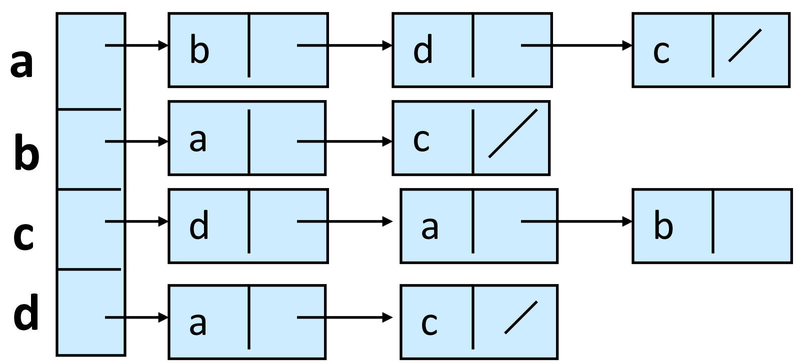
- Consists of an array Adj of $|V|$ lists.
- One list per vertex.
- For $u \in V$, $Adj[u]$ consists of all vertices adjacent to u.





Note: If weighted, store weights also in adjacency lists.





Storage Requirement

- For directed graphs:
  - Sum of lengths of all adj. lists is

$$\sum_{v \in V} out - degree(v) = |E|$$

No. of edges leaving v

  - Total storage: $\Theta(V + E)$

- For undirected graphs:
  - Sum of lengths of all adj. lists is

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

No. of edges incident on v. Edge (u, v) is incident on vertices u and v.

  - Total storage: $\Theta(V + E)$

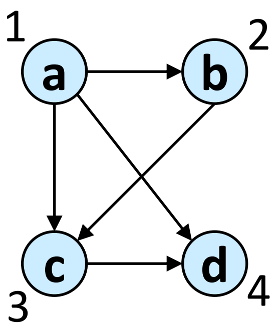
Pros and Cons: adj list

- Pros
  - Space-efficient, when a graph is sparse.
  - Can be modified to support many graph variants.
- Cons
  - Determining if an edge $(u, v) \in E$ is not efficient.
    - Have to search in u' s adjacency list. $\Theta(degree(u))$ time.
    - $\Theta(V)$ in the worst case.

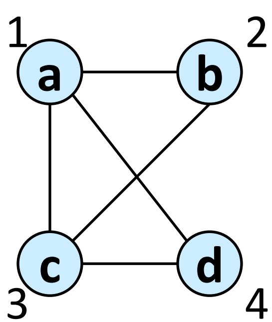
Adjacency Matrix

- $|V|$ ⍰ $|V|$ matrix A.
- Number vertices from 1 to $|V|$ in some arbitrary manner.
- A is then given by:

$$A[i,j] = a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$



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



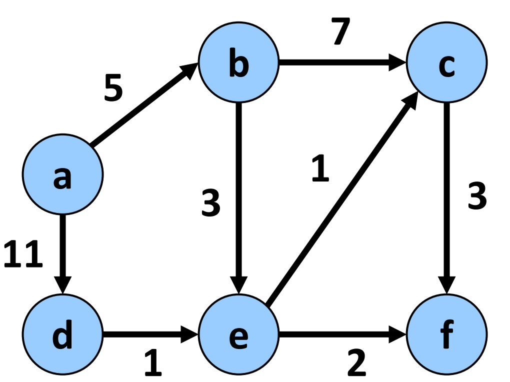
	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

$A = A^{\mathsf{T}}$ for undirected graphs.

Space and Time

- **Space:** $\Theta(V^2)$.
  - Not memory efficient for large sparse graphs.
- **Time:** to list all vertices adjacent to u: $\Theta(V)$.
- **Time:** to determine if $(u, v) \in E$: $\Theta(1)$.
- Can store weights instead of bits for weighted graph.

	a	b	c	d	e	f
a	0	5	0	11	0	0
b	0	0	7	0	3	0
c	0	0	0	0	0	3
d	0	0	0	0	1	0
e	0	0	1	0	0	2
f	0	0	0	0	0	0



Graph-searching Algorithms (COMP250)

- 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 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.

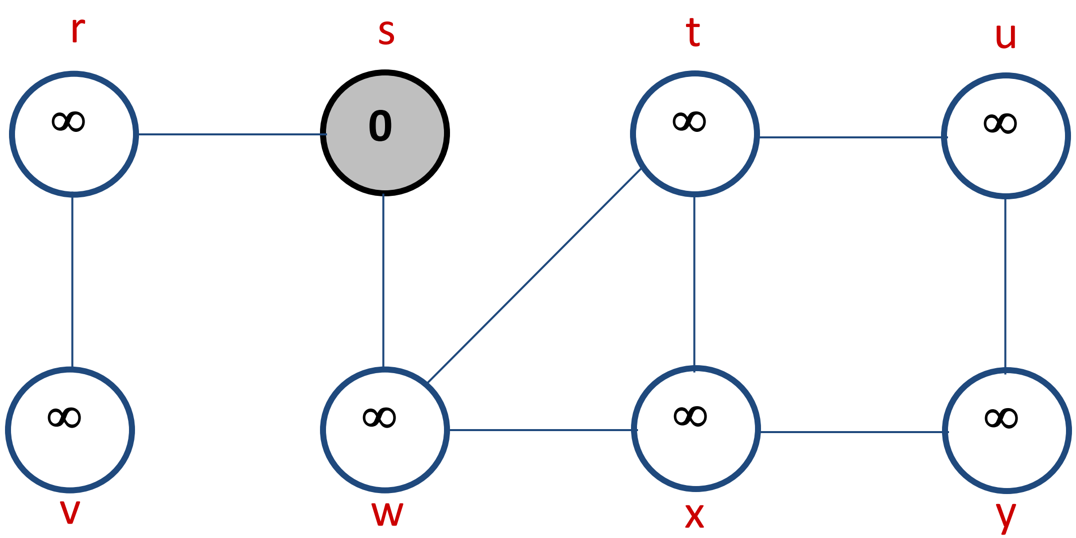
- Colors the vertices to keep track of progress.
  - White – Undiscovered.
  - Gray – Discovered but not finished.
  - Black – Finished.
    - Colors are required only to reason about the algorithm. Can be implemented without colors.

Breadth-first Search

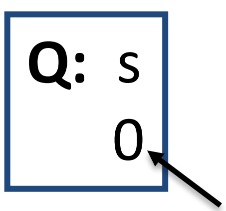
- **Input:** Graph $G = (V, E)$, either directed or undirected, and ***source vertex*** $s \in V$.

- **Output:**
  - $d[v]$ = distance (smallest # of edges, or shortest path) 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 $s \rightsquigarrow v$.
    - u is v's predecessor.
  - Builds breadth-first tree with root s that contains all reachable vertices.

Example (BFS)



We use a priority queue to determine the next vertices to visit.



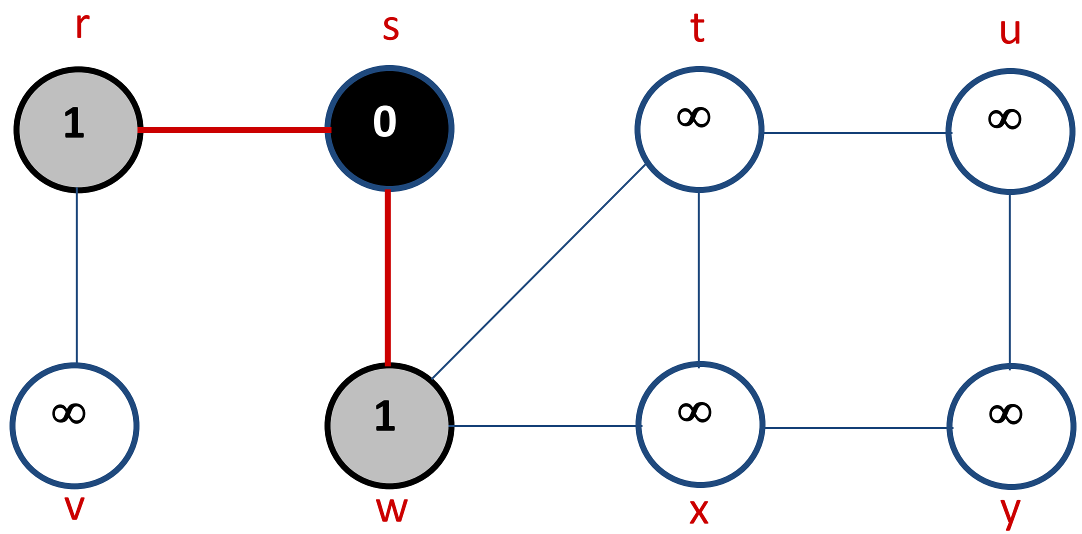
The first vertex we add in the queue is the source.

Priority = distance from the source.
Lower the distance, higher the priority.

Example (BFS)

Color code:

- White: not visited yet
- Gray: visited but neighborhood not fully explored
- Black: Complete

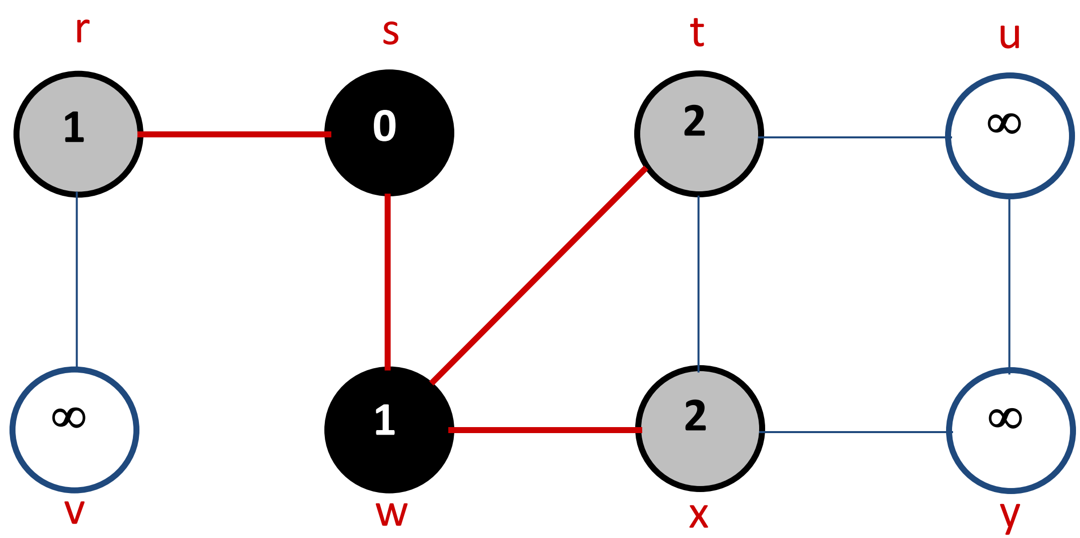


We store in the queue the vertices in the neighborhood of the current vertex.

Q:	w	r
	1	1

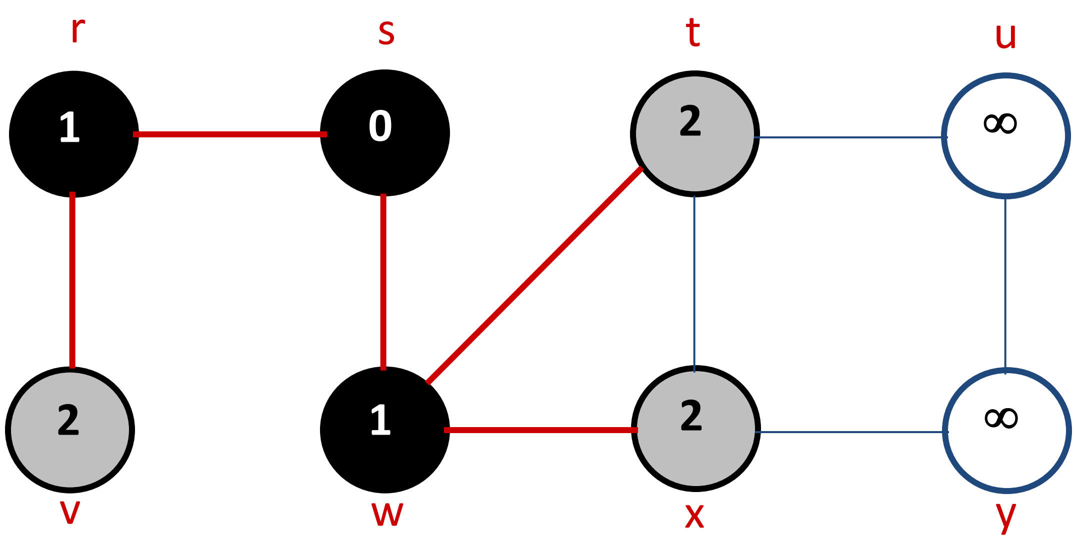
The vertices are indexed with the number of edges from the source.

Example (BFS)



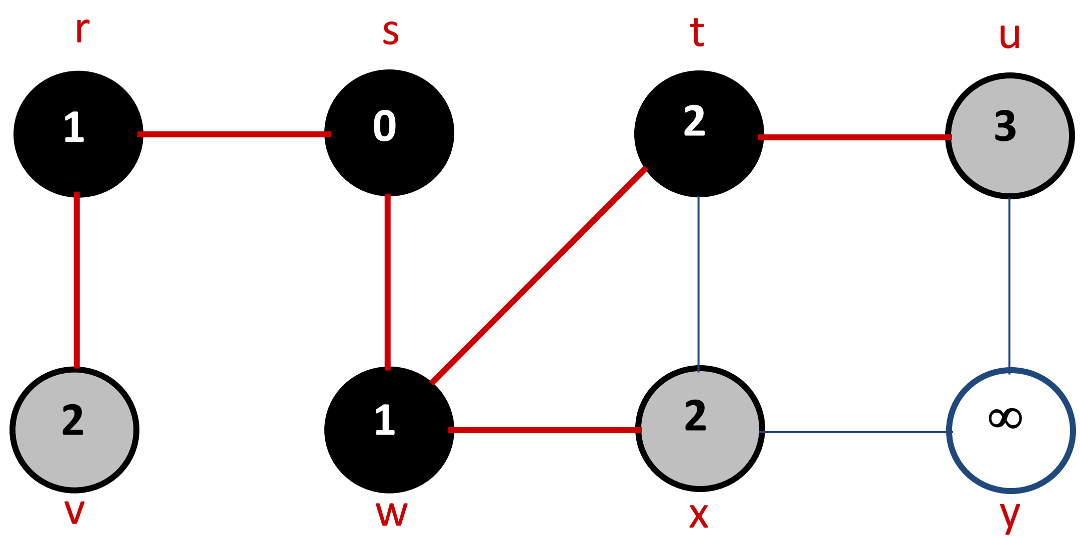
Q:	r	t	x
	1	2	2

Example (BFS)



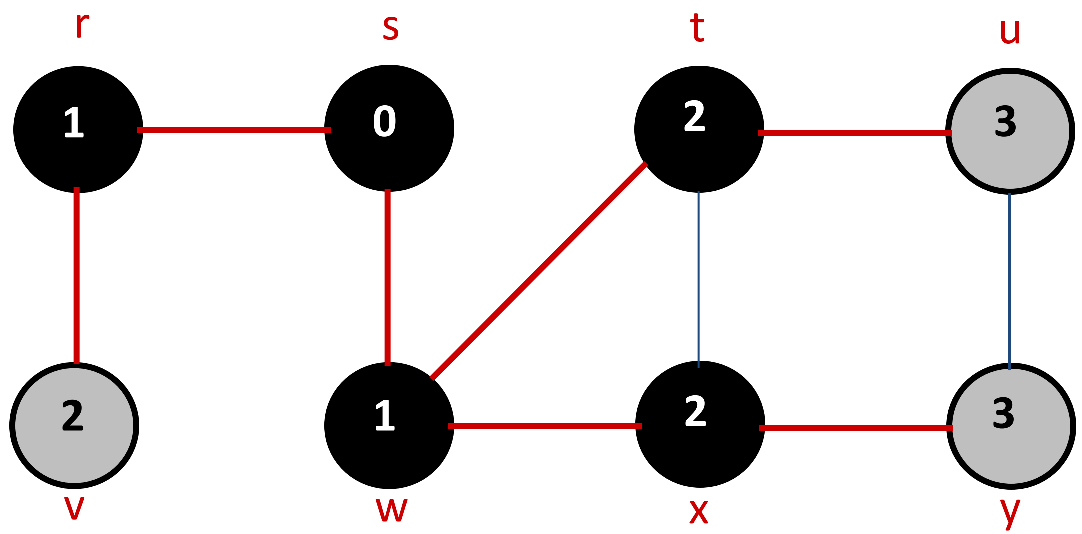
Q:	t	x	v
	2	2	2

Example (BFS)



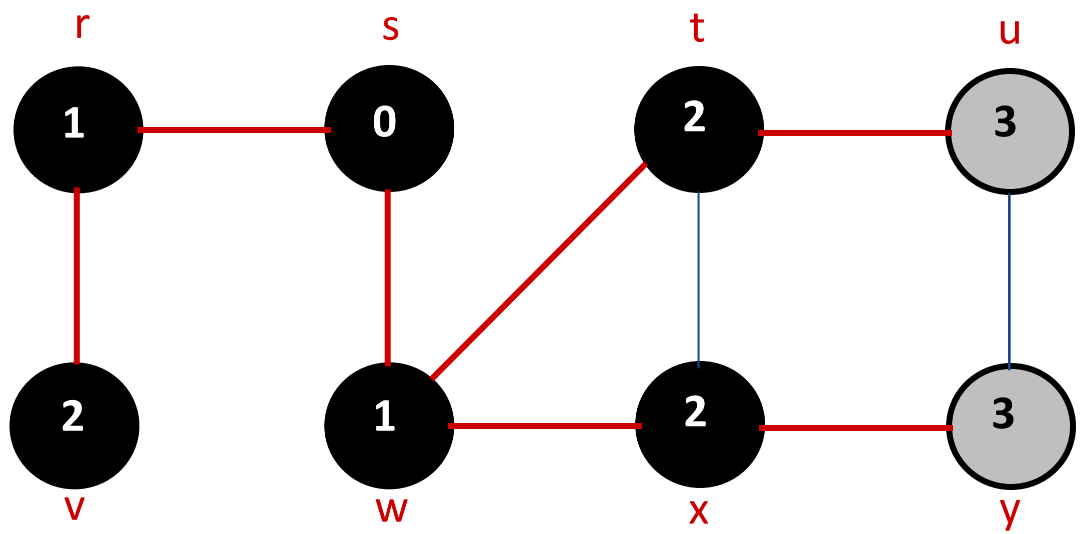
Q:	x	v	u
	2	2	3

Example (BFS)



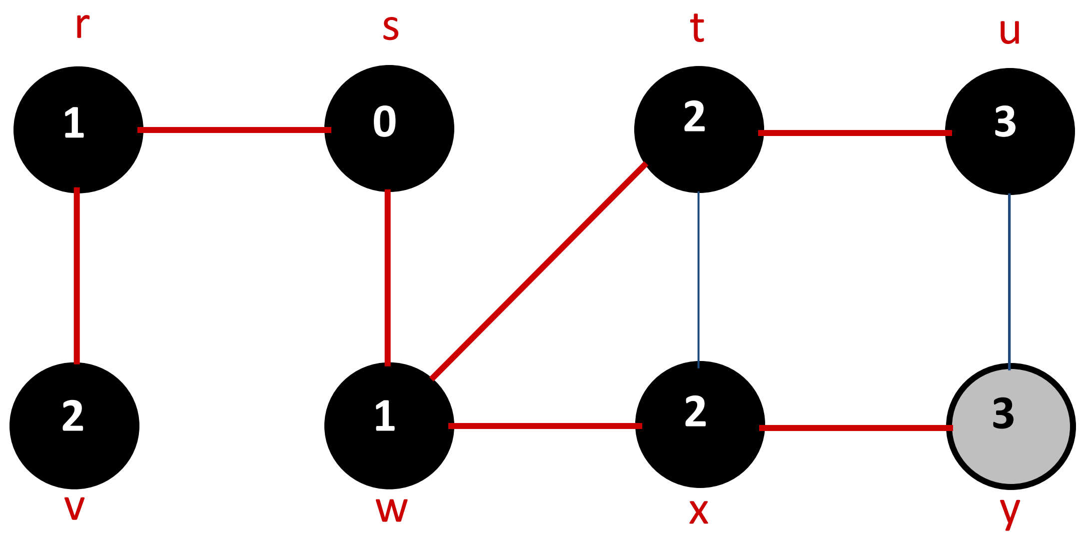
Q: v u y
2 3 3

Example (BFS)



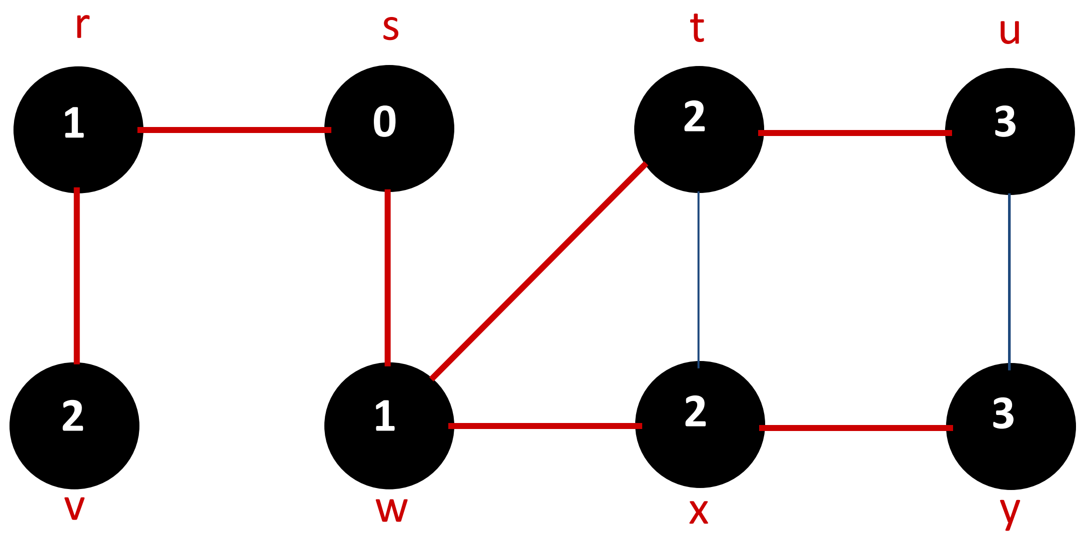
Q: u y
3 3

Example (BFS)



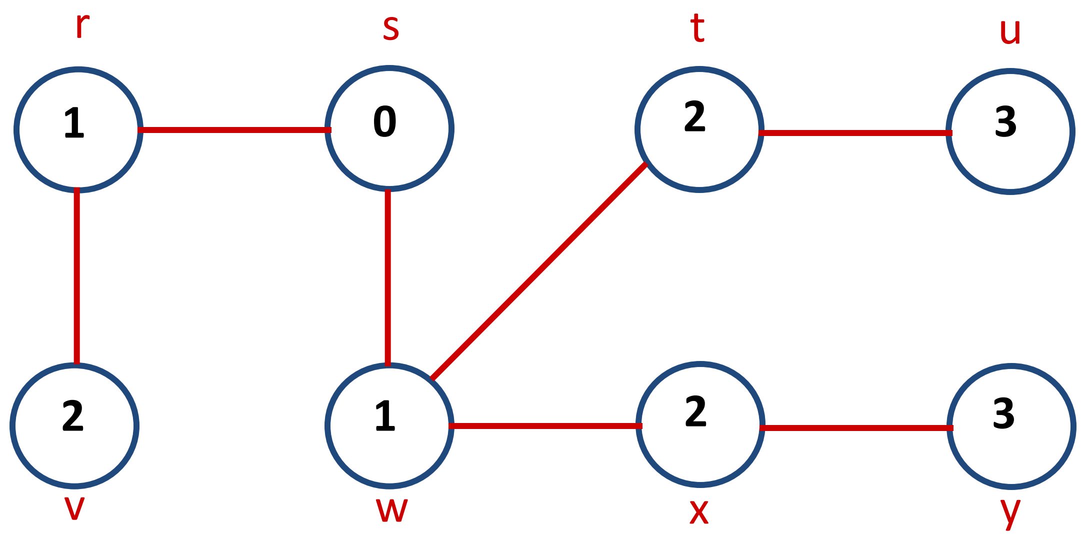
Q:	y
	3

Example (BFS)



Q: $\varnothing$

Example (BFS)



The index of each vertex stores the length of the shortest path to reach them (Note: unweighted graph!).

BF 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 $\Rightarrow$ total running time of BFS is $O(V + E)$, linear in the size of the adjacency list representation of graph.

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 other edges leaving the vertex from which v was discovered (its *predecessor*).
- "Search as deep as possible first."
- 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:** $G = (V, E)$, 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 $v = u$, such that v was discovered during the scan of u' s adjacency list.
- Uses the same coloring scheme for vertices as BFS.

Pseudo-code

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)
```

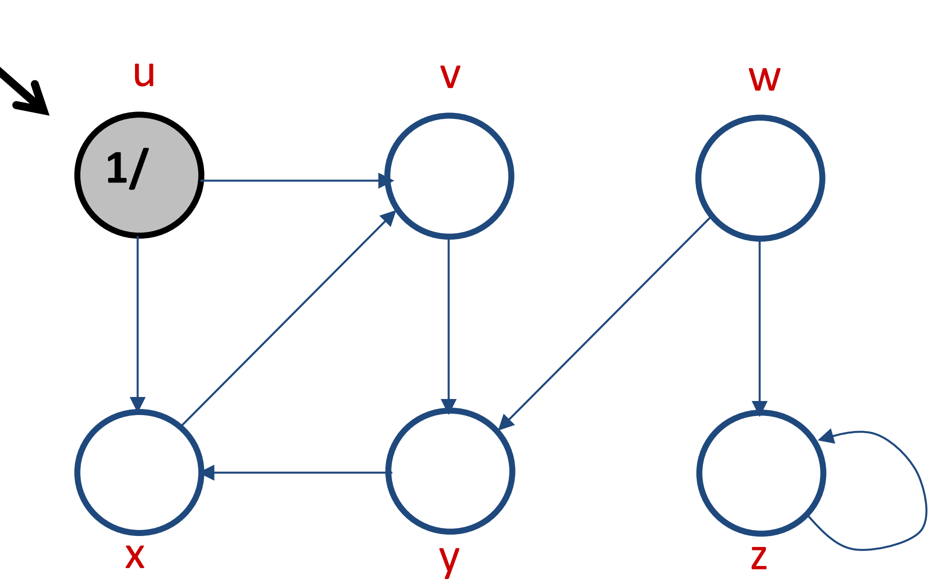
Uses a global timestamp ***time***.

DFS-Visit(*u*)

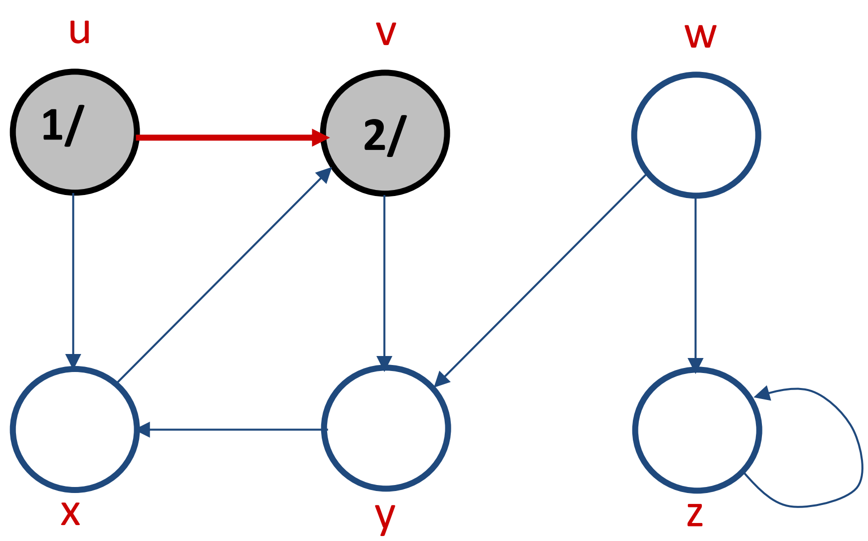
```
color[u] ← GRAY  # White vertex u has been discovered
time ← time + 1
d[u] ← time
for each v ∈ Adj[u]
    do if color[v] = WHITE
        then π[v] ← u
            DFS-Visit(v)
color[u] ← BLACK    # Blacken u;  it is finished.
f[u] ← time ← time + 1
```

Example (DFS)

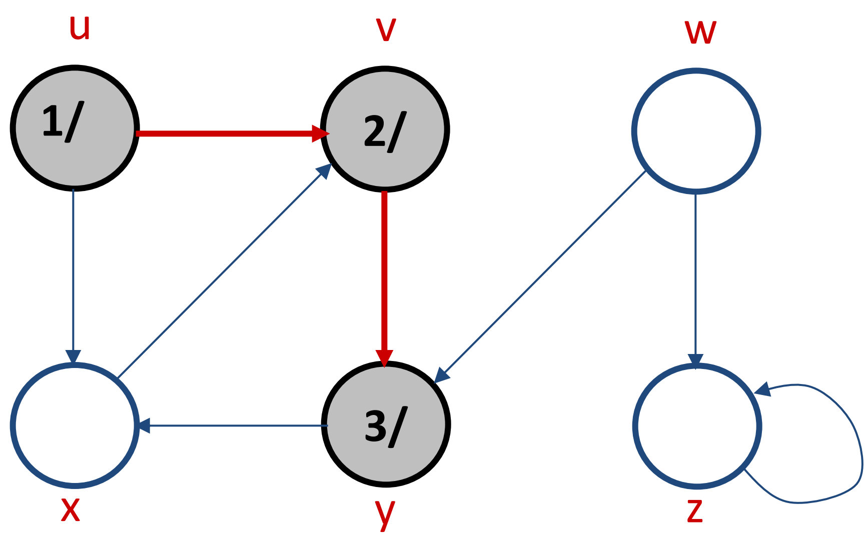
Starting time
$d[u]$



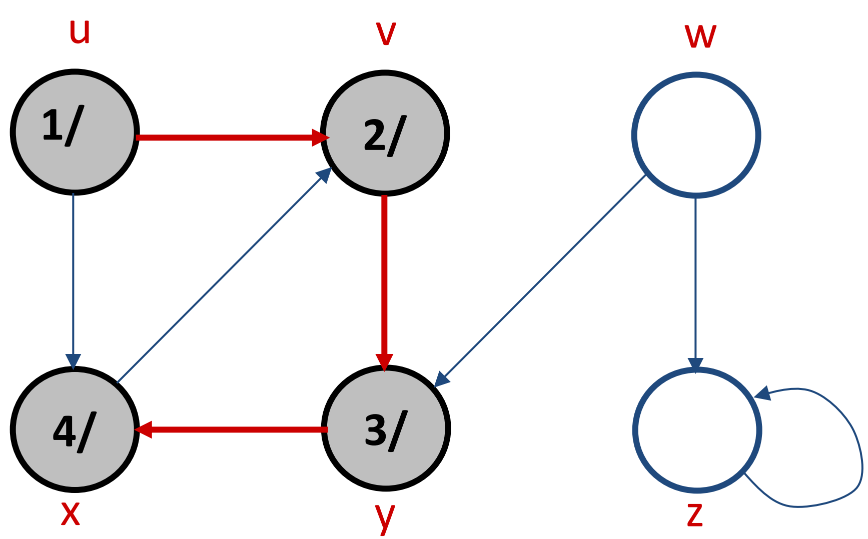
Example (DFS)



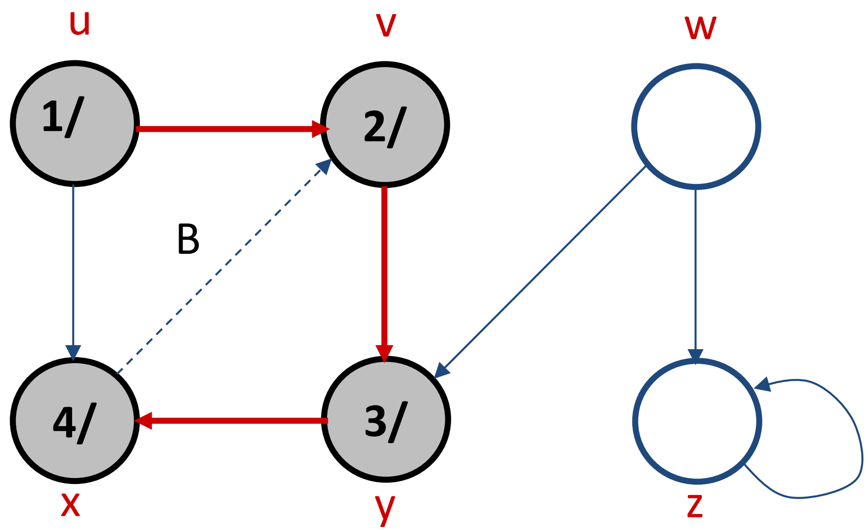
Example (DFS)



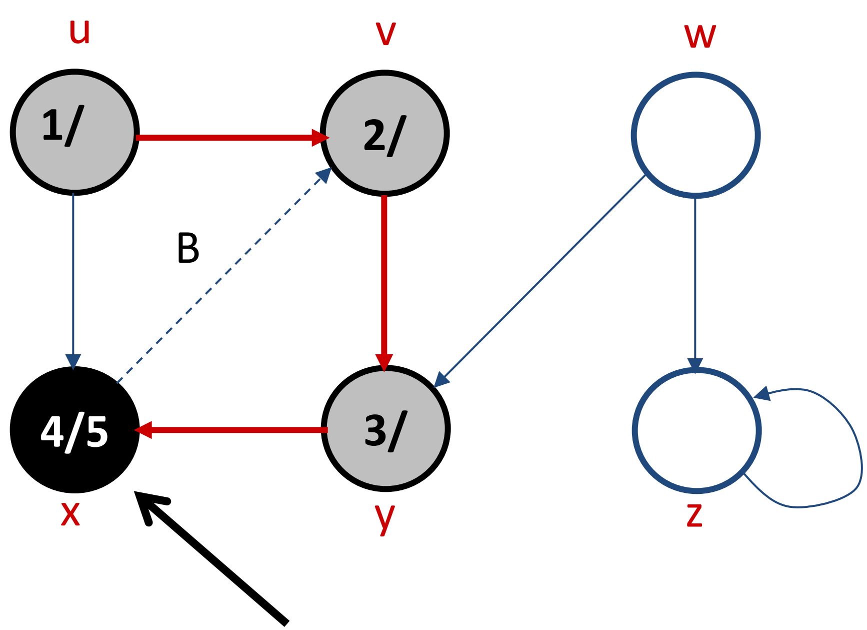
Example (DFS)



Example (DFS)



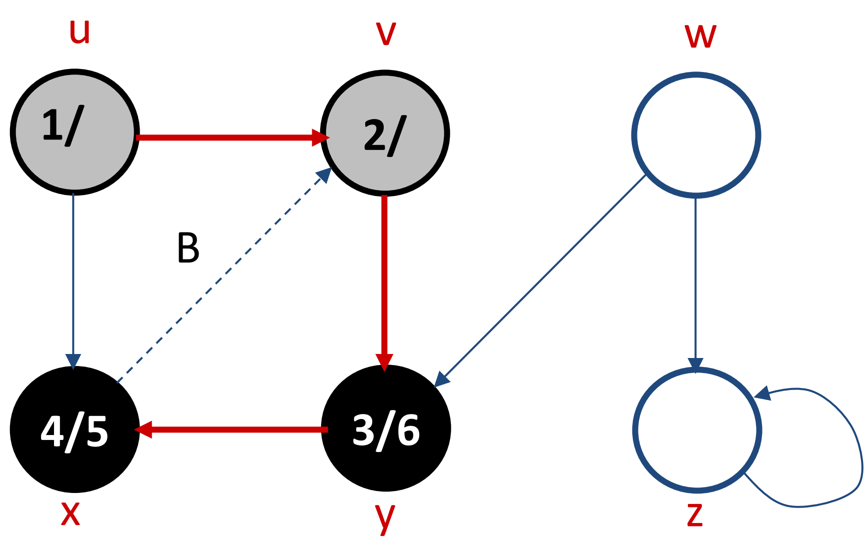
Example (DFS)



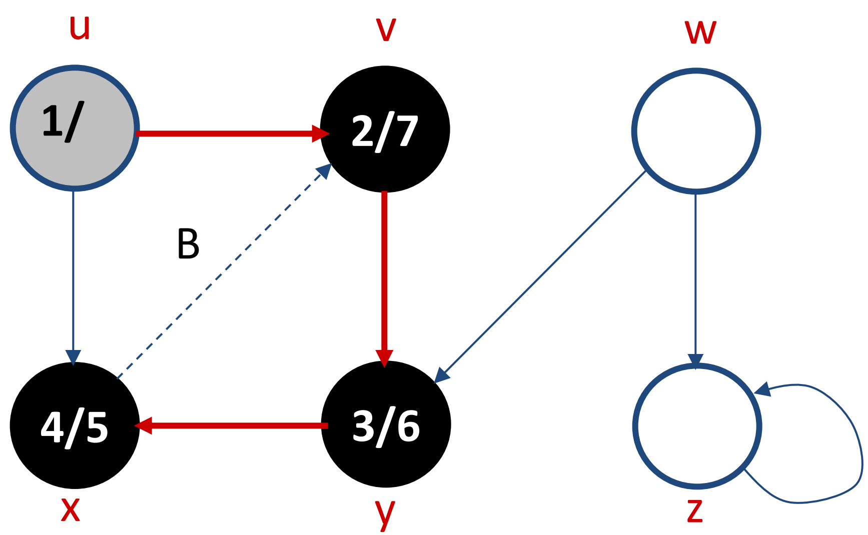
Starting time
$d[x]$

Finishing time
$f[x]$

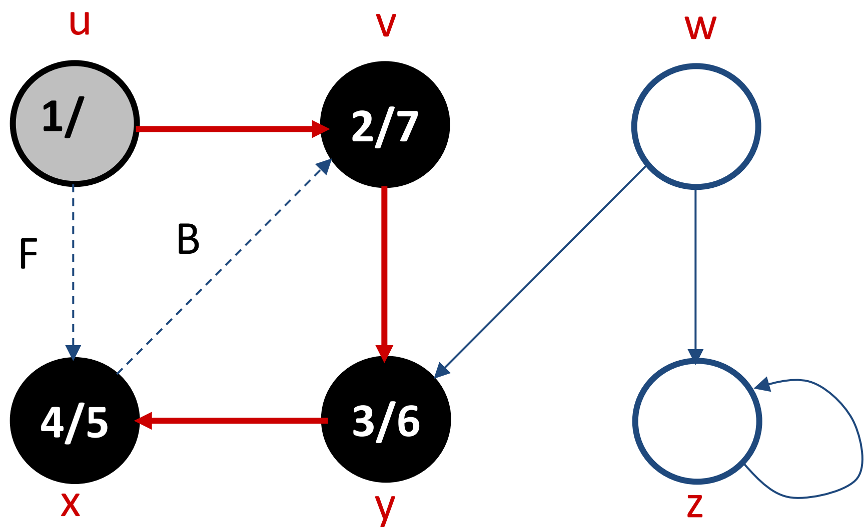
Example (DFS)



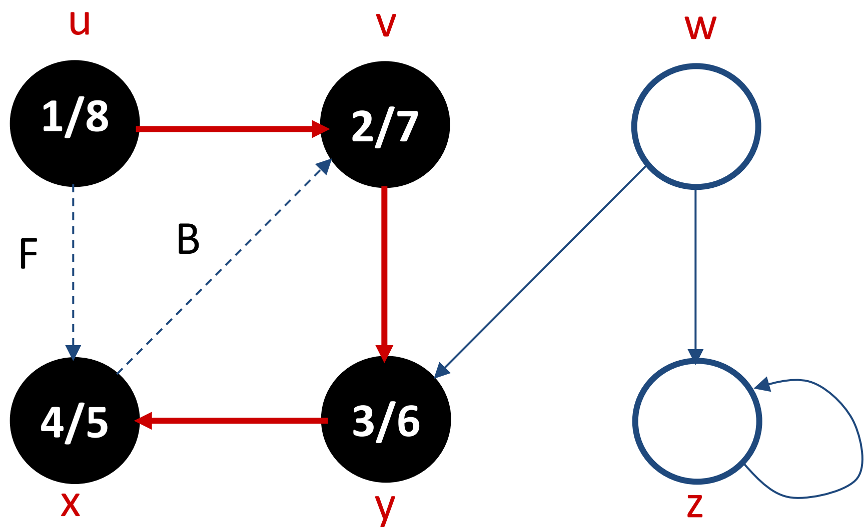
Example (DFS)



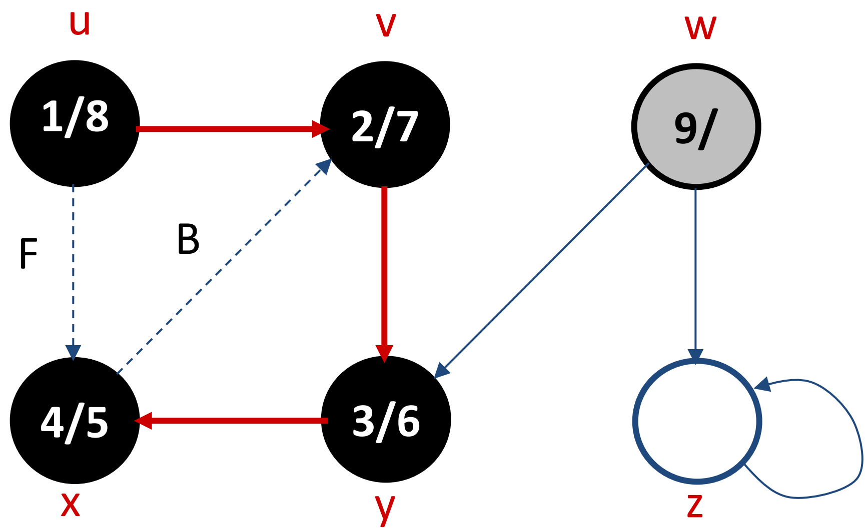
Example (DFS)



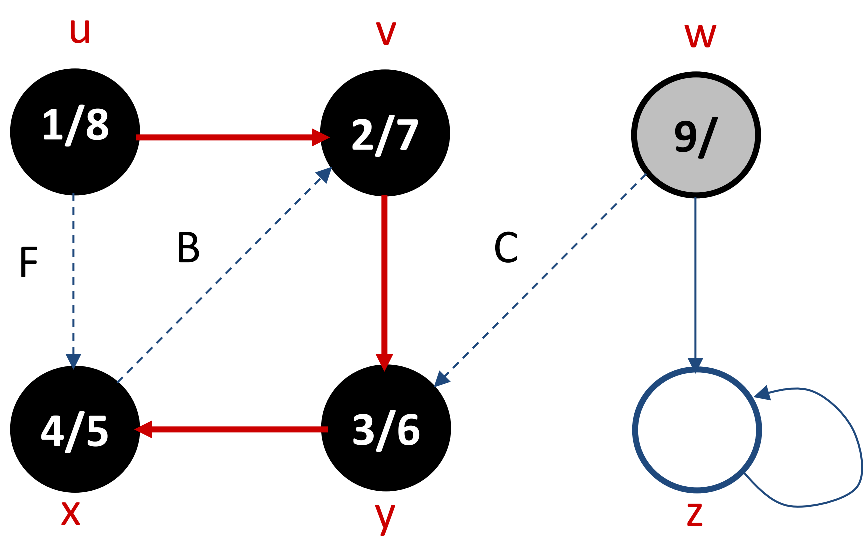
Example (DFS)



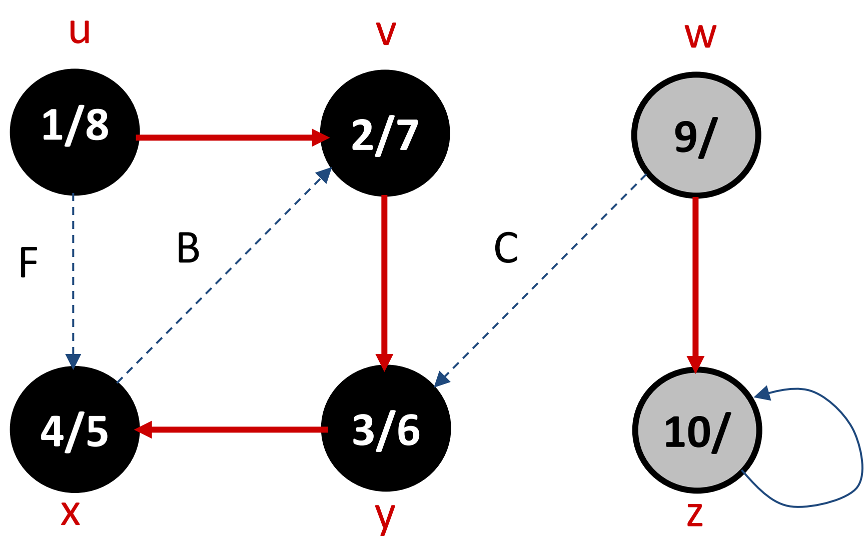
Example (DFS)



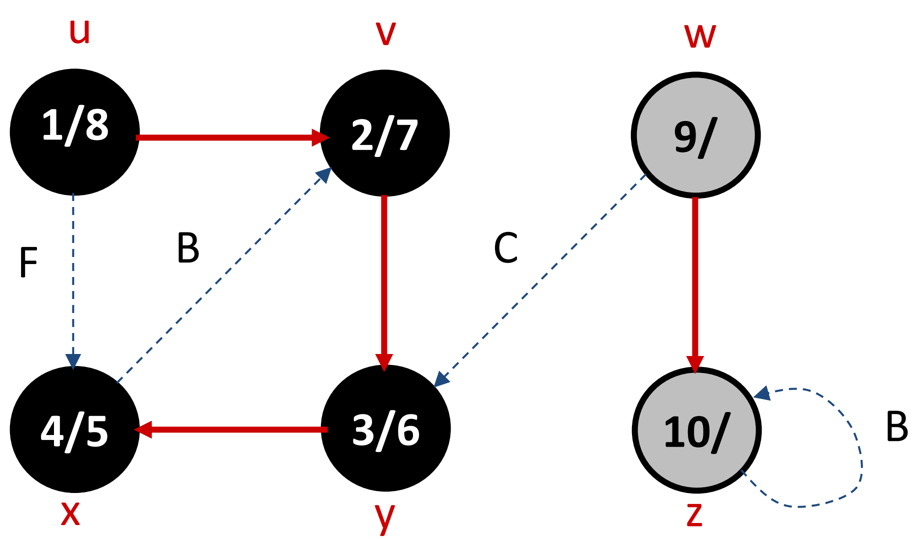
Example (DFS)



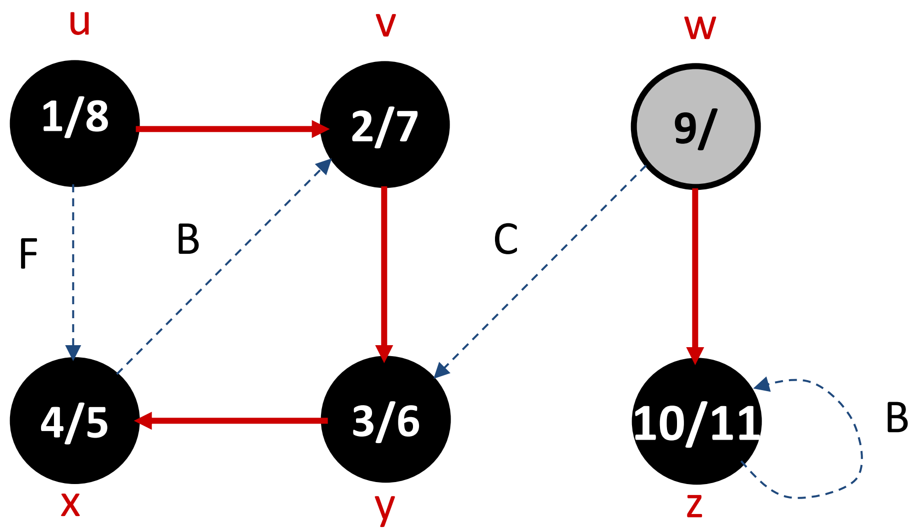
Example (DFS)



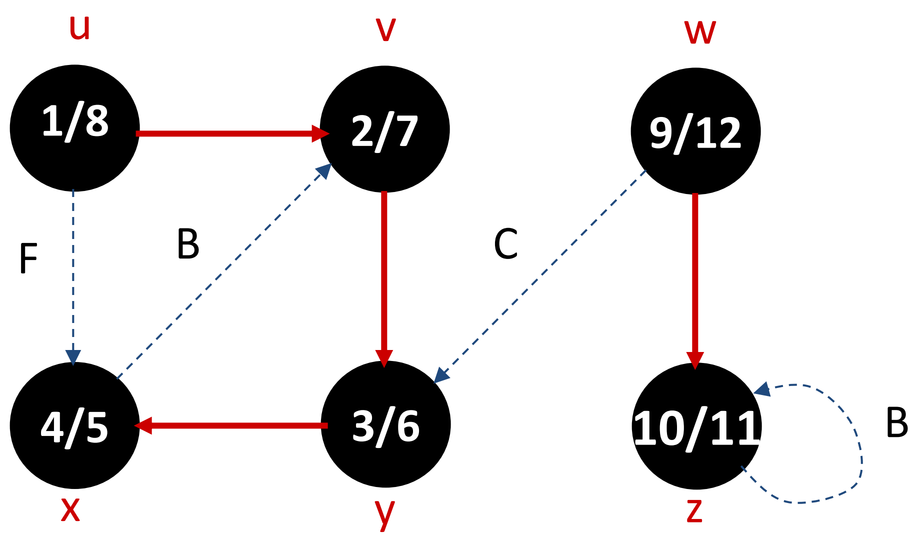
Example (DFS)



Example (DFS)



Example (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 $|Adj[v]|$ 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 1:

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

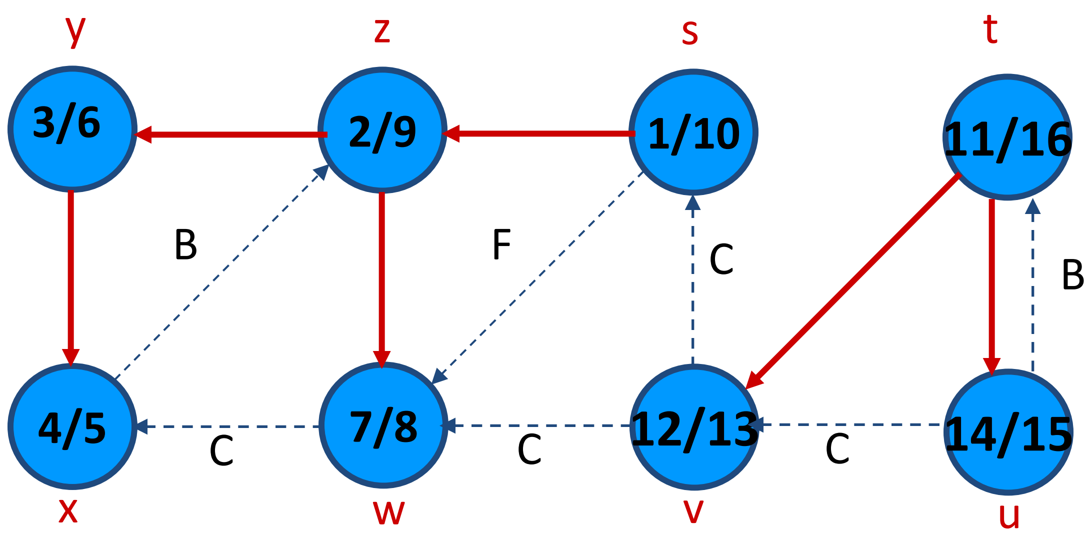
1. $d[u] < f[u] < d[v] < f[v]$ or $d[v] < f[v] < d[u] < f[u]$ and neither u nor v is a descendant of the other.
2. $d[u] < d[v] < f[v] < f[u]$ and v is a descendant of u.
3. $d[v] < d[u] < f[u] < f[v]$ and u is a descendant of v.

- 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* $d[u] < d[v] < f[v] < f[u]$.

Example (Parenthesis Theorem)



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