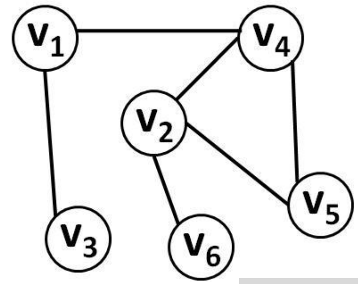
Graphs

Introduction

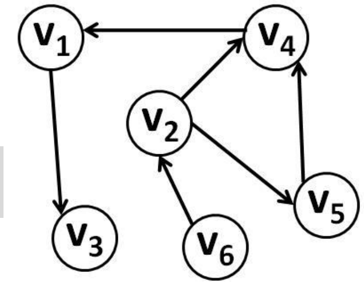
- Generalization of a tree.
- Collection of vertices (or nodes) and connections between them.

- No restriction on
  - The number of vertices.
  - The number of connections between the two vertices.
- Have several real life applications.

Definition



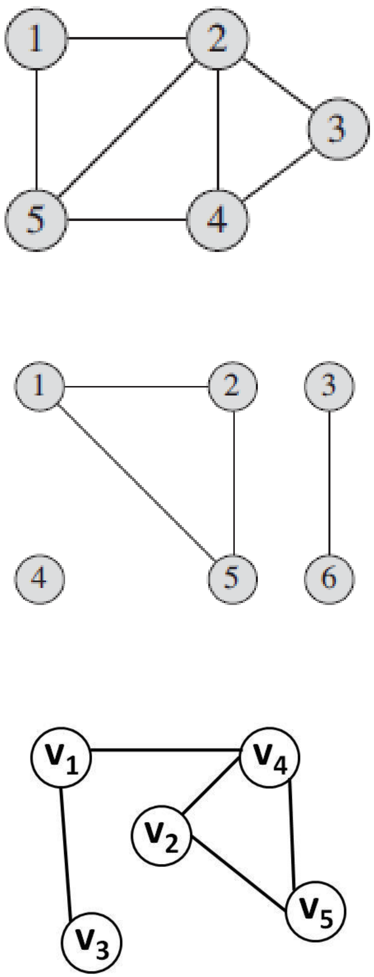
Undirected Graph



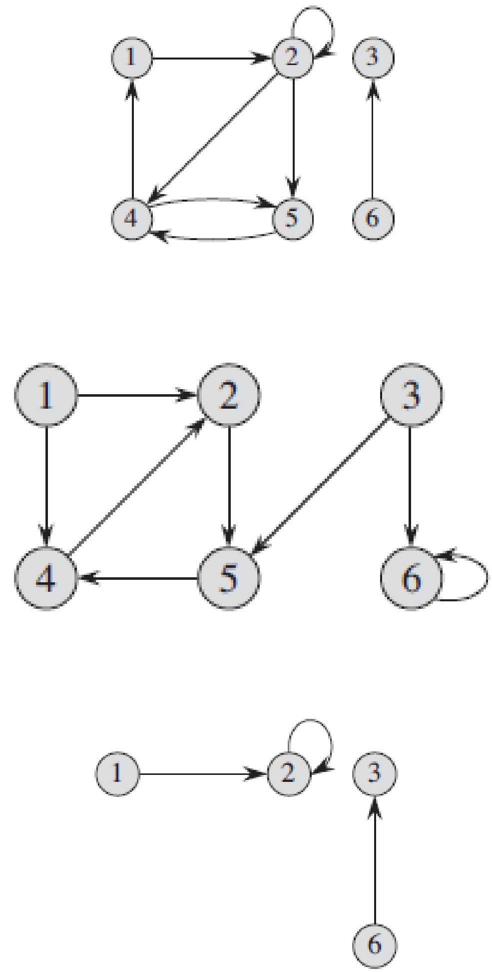
Directed Graph

- A graph ***G* = (*V*,*E*)** consists of a
  - Finite, non-empty set ***V*** of ***vertices*** and
  - Possibly empty set ***E*** of ***edges***. A binary relation on ***V***.
- **|*V*|** denotes number of vertices.
- **|*E*|** denotes number of edges.
- An edge (or arc) is a pair of vertices (v_i, v_j) from ***V***.
  - Simple or undirected graph $(v_i, v_j) = (v_j, v_i)$.
  - Digraph or directed graph $(v_i, v_j) \neq (v_j, v_i)$.
- An edge has an associated **weight** or **cost** as well.

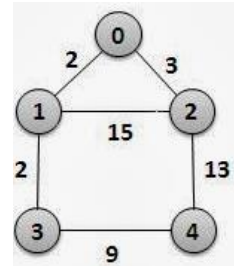
Contd…



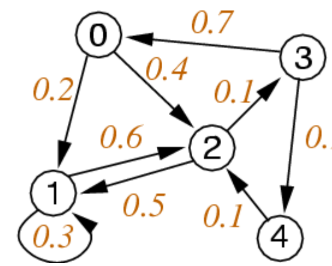
Undirected Graph



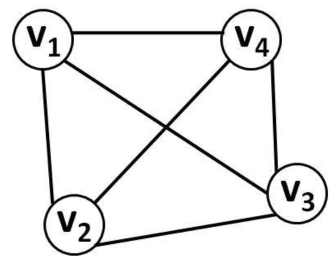
Directed Graph



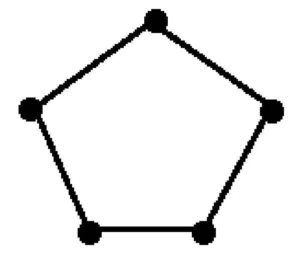
Weighted Undirected Graph



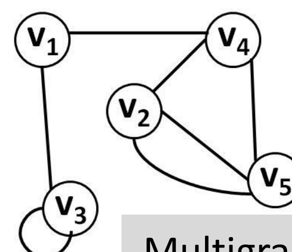
Weighted Directed Graph



Complete Graph



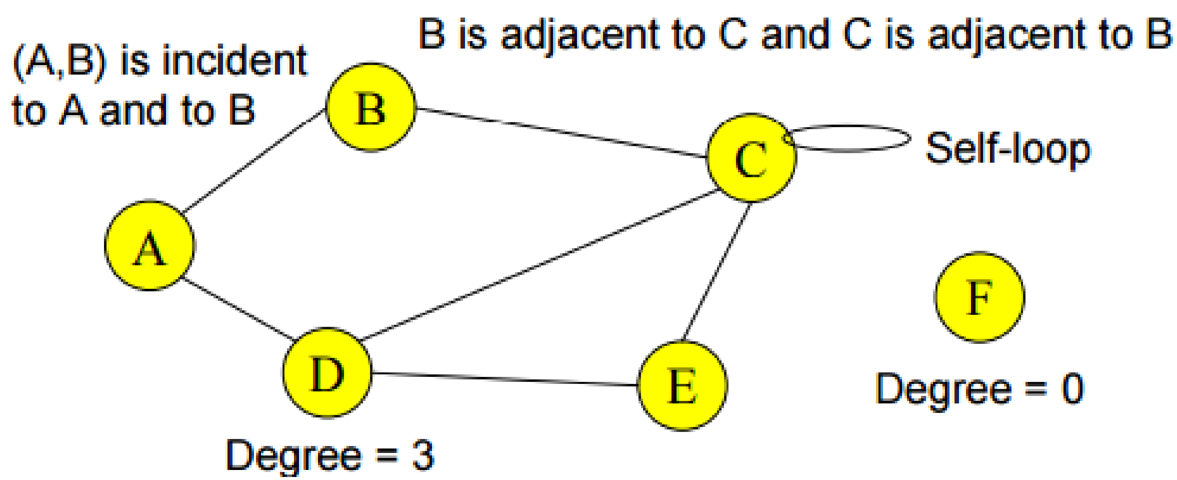
Cycle Graph



Multigraph

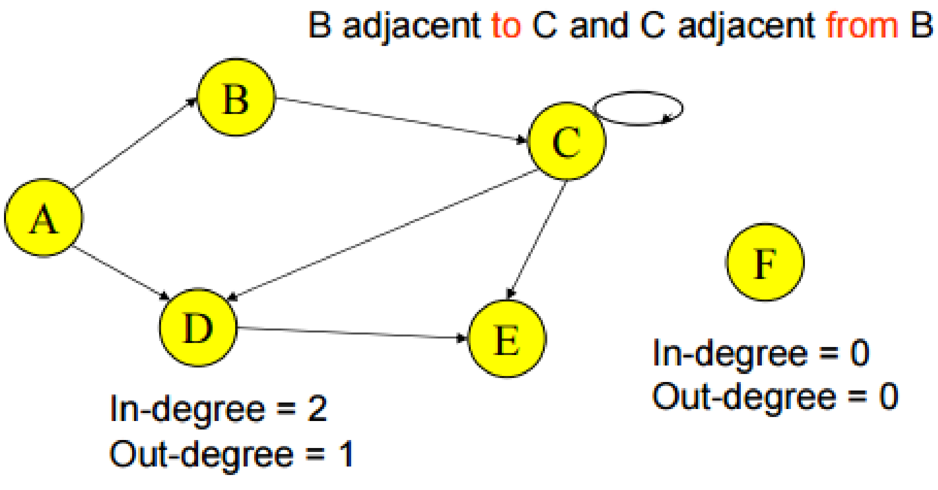
Terminology (Undirected)

- Two vertices u and v are adjacent if {u,v} is an edge in G.
  – Edge {u,v} is incident with vertex u and vertex v.
- Degree of a vertex is the number of edges incident with it.
  – A self-loop counts twice (both ends count).



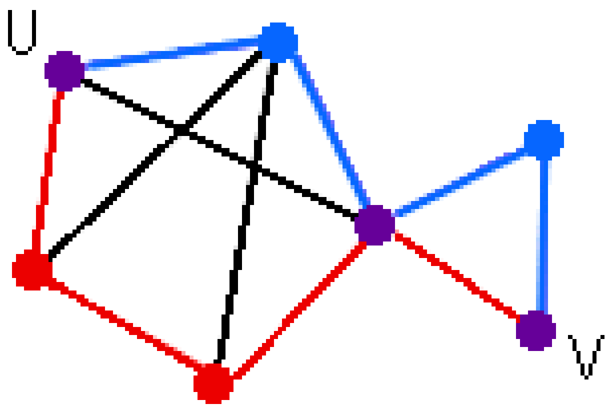
Terminology (Directed)

- Vertex u is adjacent to vertex v if (u,v) is an edge in G and vertex u is the initial vertex of (u,v).
- Vertex v is adjacent from vertex u, if vertex v is the terminal (or end) vertex of (u,v).
- A vertex has two types of degree.
  - in-degree: The number of edges with the vertex as the terminal vertex.
  - out-degree: The number of edges with the vertex as the initial vertex



Some Definitions

- Walk
  - An alternating sequence of vertices and connecting edges.
  - Can end on the same vertex on which it began or on a different vertex.
  - Can travel over any edge and any vertex any number of times.
- Path or Simple Path
  - A walk that does not include any vertex twice, except that its first and last vertices might be the same.



Representations of Graphs

Representations of Graphs

- Two standard ways are:
  - Collection of adjacency lists.
  - Adjacency matrix.
- Applies to both directed and undirected graphs.
- Adjacency-list representation provides a compact way to represent sparse graphs ($|E| << |V|^2$).
  - Usually the method of choice.
- Adjacency-matrix representation is preferred when the graph is dense ($|E| \approx |V|^2$).

Representation – I

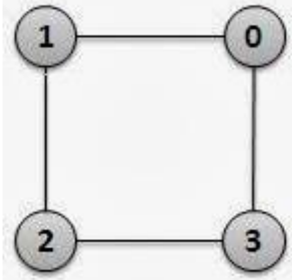
- Adjacency matrix
  - Adjacency matrix for a graph G = (V, E) is a two dimensional matrix of size |V| x |V| such that each entry of this matrix

$$a[i][j] = \begin{cases} 1 \text{ (or weight), if an edge } (v_i,v_j) \text{ exists.} \\ 0\text{, otherwise.} \end{cases}$$

  - For an undirected graph, it is always a symmetric matrix, as $(v_i, v_j) = (v_j, v_i)$.

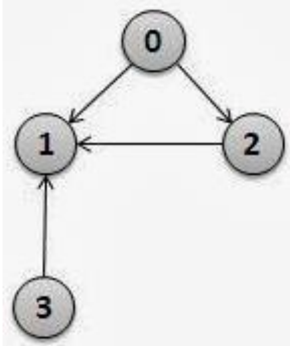
Adjacency matrix

- Undirected.
  - V = {0, 1, 2, 3}
  - E = {(0,1), (1,2), (2,3), (3,0)}



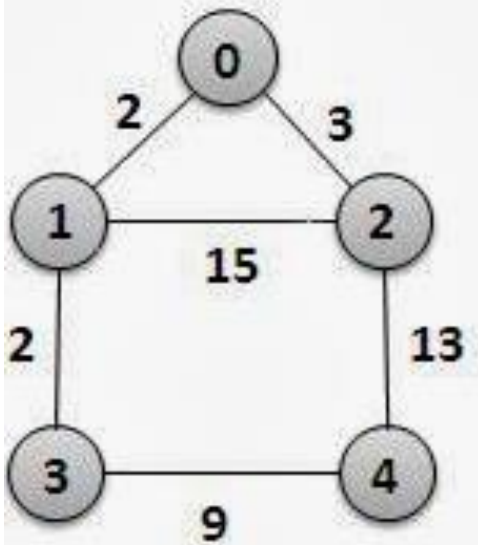
	0	1	2	3
0	0	1	0	1
1	1	0	1	0
2	0	1	0	1
3	1	0	1	0

- Directed.
  - V = {0, 1, 2, 3}
  - E = {(0,1), (0,2), (2,1), (3,1)}



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

Contd… (weighted)

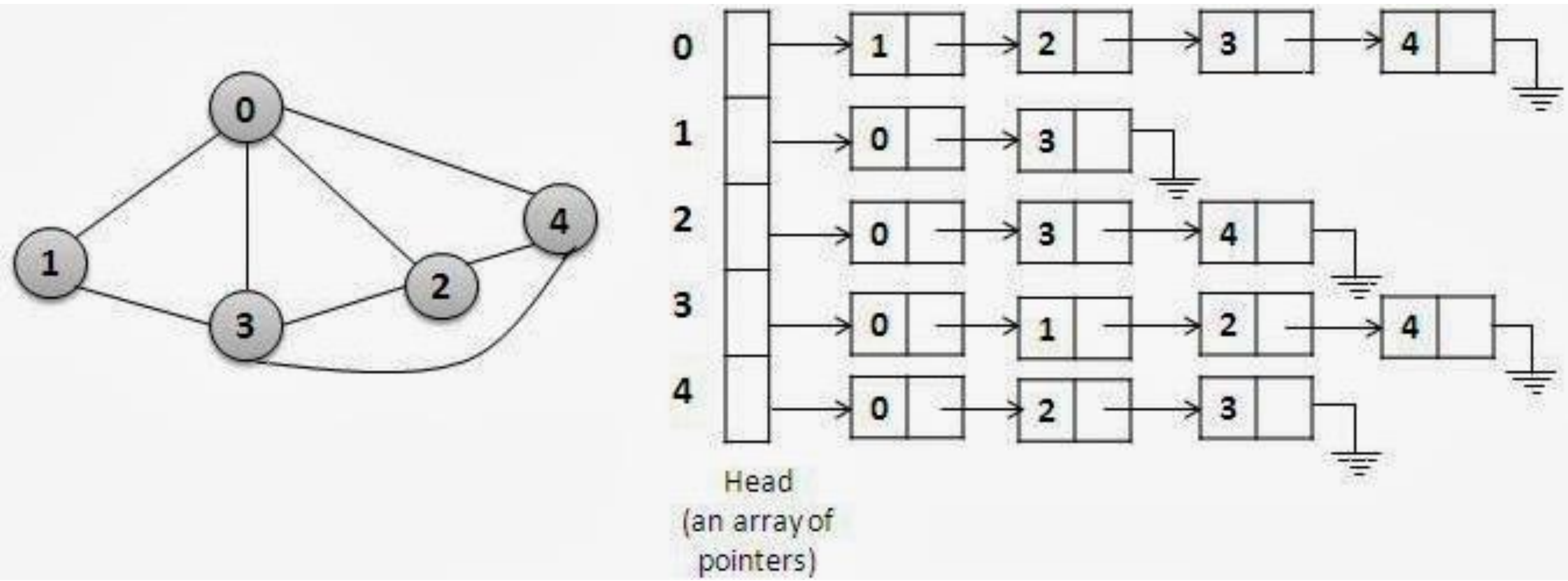


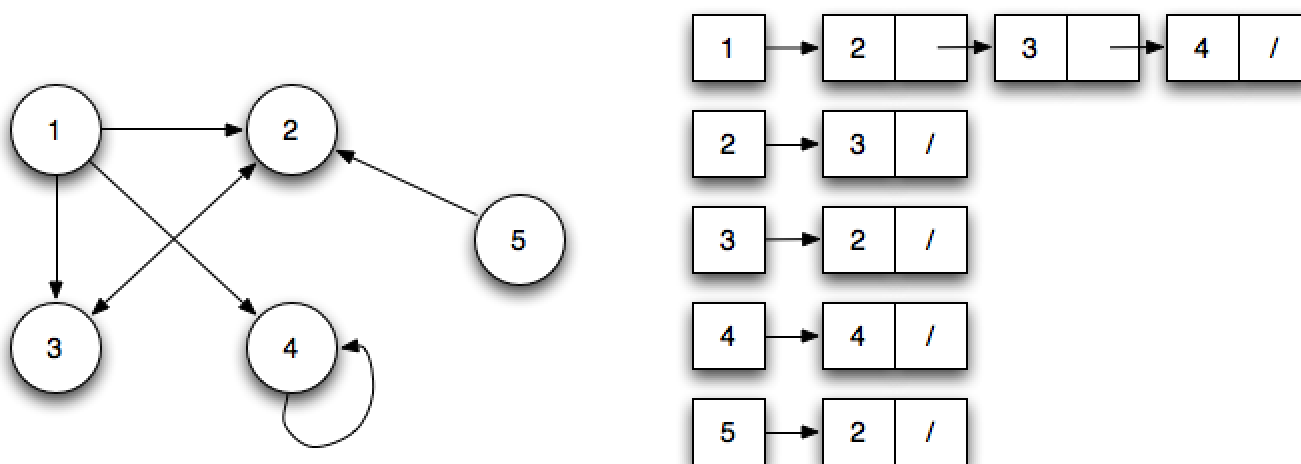
	0	1	2	3	4
0	0	2	3	0	0
1	2	0	15	2	0
2	3	15	0	0	13
3	0	2	0	0	9
4	0	0	13	9	0

Representation – II

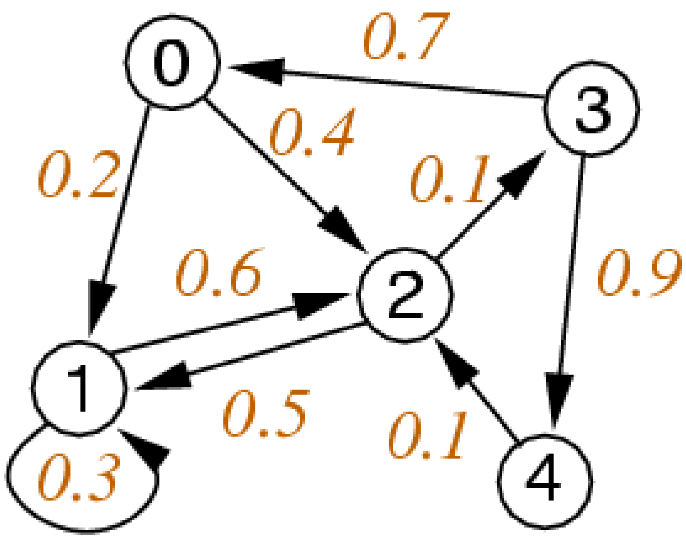
- Adjacency list
  - Uses an array of linked lists with size equals to **|V|**.
  - An i^{th} entry of an array points to a linked list of vertices adjacent to $\mathbf{v_i}$.
  - The weights of edges are stored in nodes of linked lists to represent a weighted graph.

Adjacency List

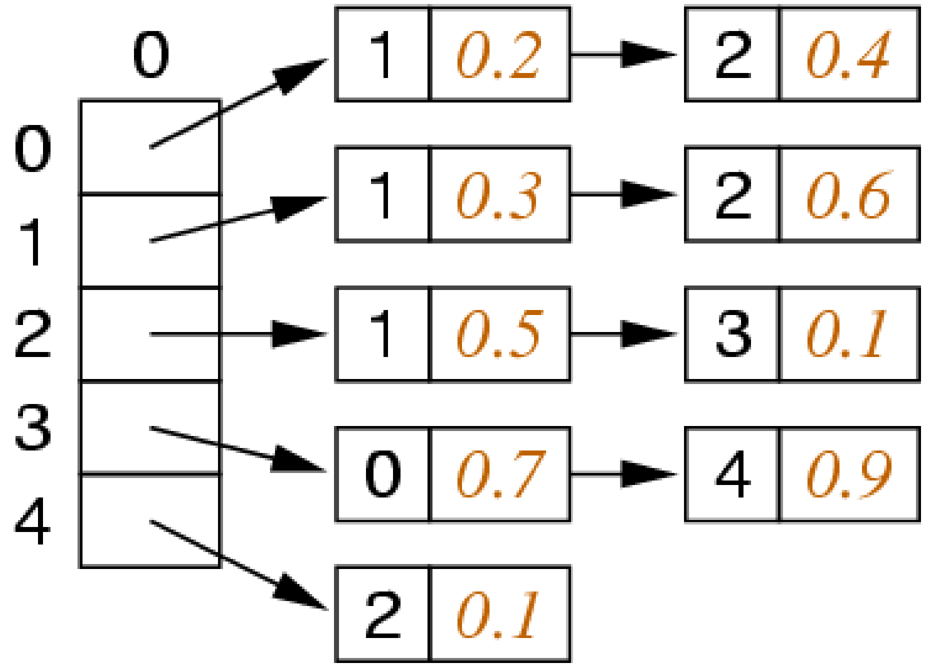




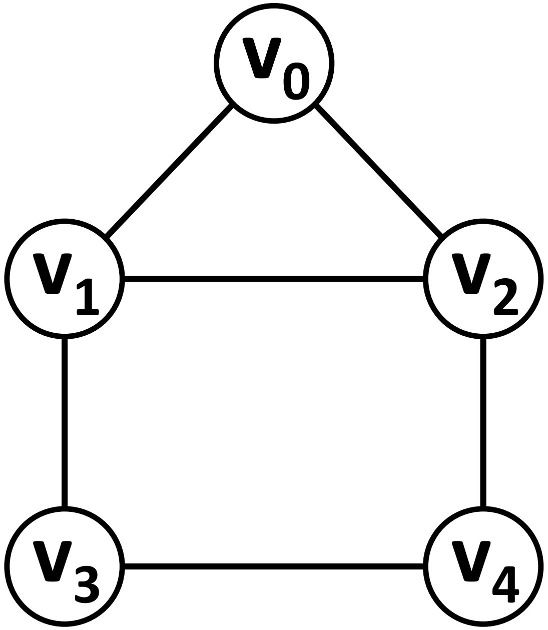
Contd…(weighted)



Weighted Digraph



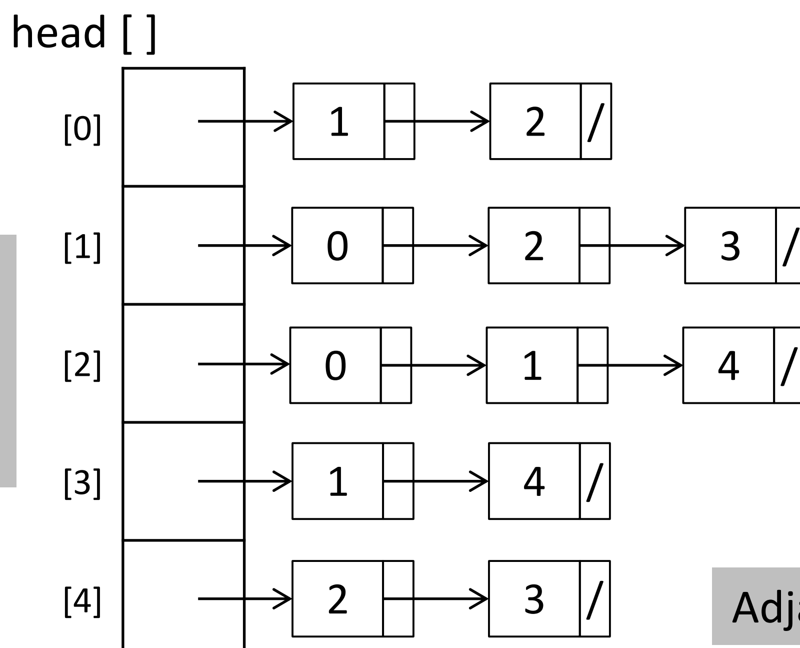
Adjacency Lists



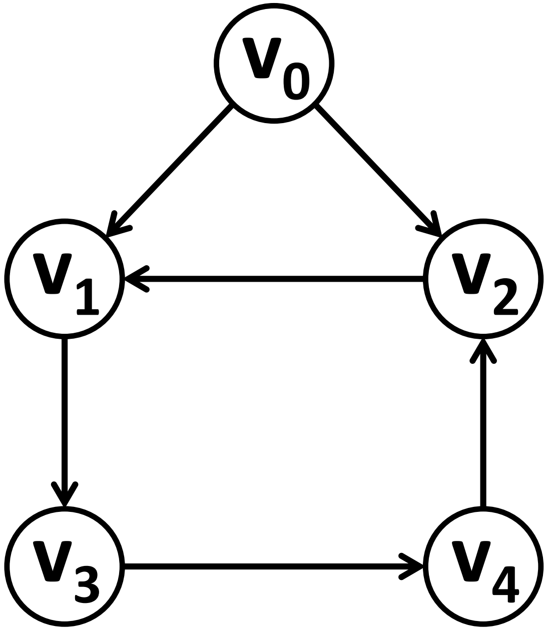
	V_0	V_1	V_2	V_3	V_4
V_0	0	1	1	0	0
V_1	1	0	1	1	0
V_2	1	1	0	0	1
V_3	0	1	0	0	1
V_4	0	0	1	1	0

Adjacency Matrix

```
struct node
{ int v;
  struct node *next;
} *head[5];
```



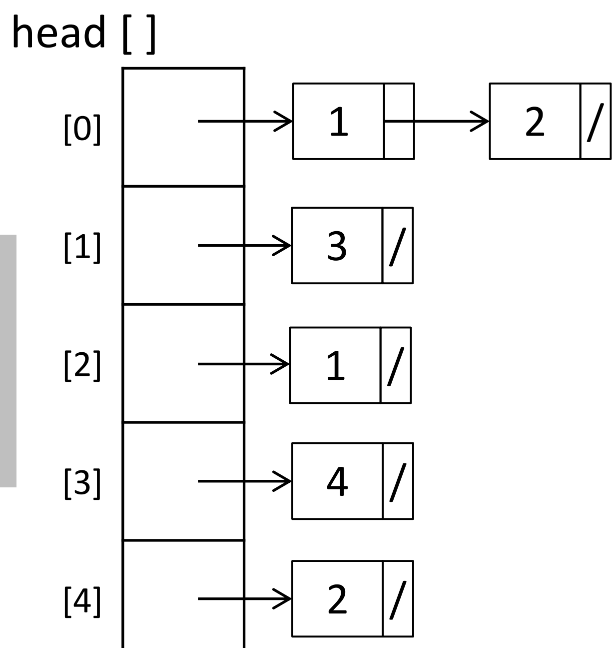
Adjacency List



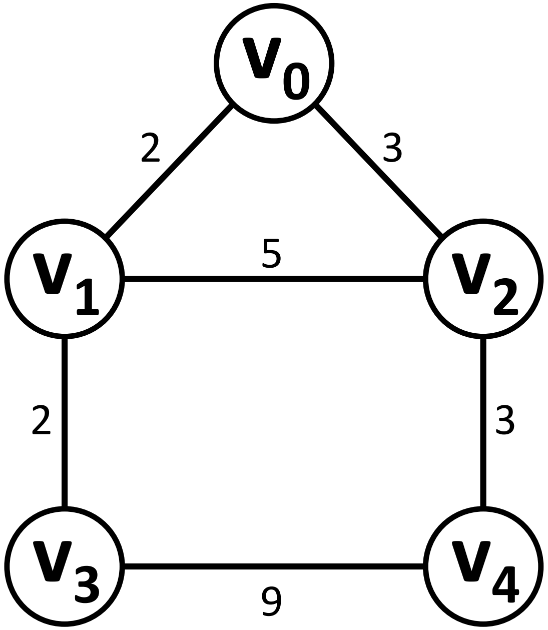
	V_0	V_1	V_2	V_3	V_4
V_0	0	1	1	0	0
V_1	0	0	0	1	0
V_2	0	1	0	0	0
V_3	0	0	0	0	1
V_4	0	0	1	0	0

Adjacency Matrix

```
struct node
{ int v;
  struct node *next;
} *head[5];
```

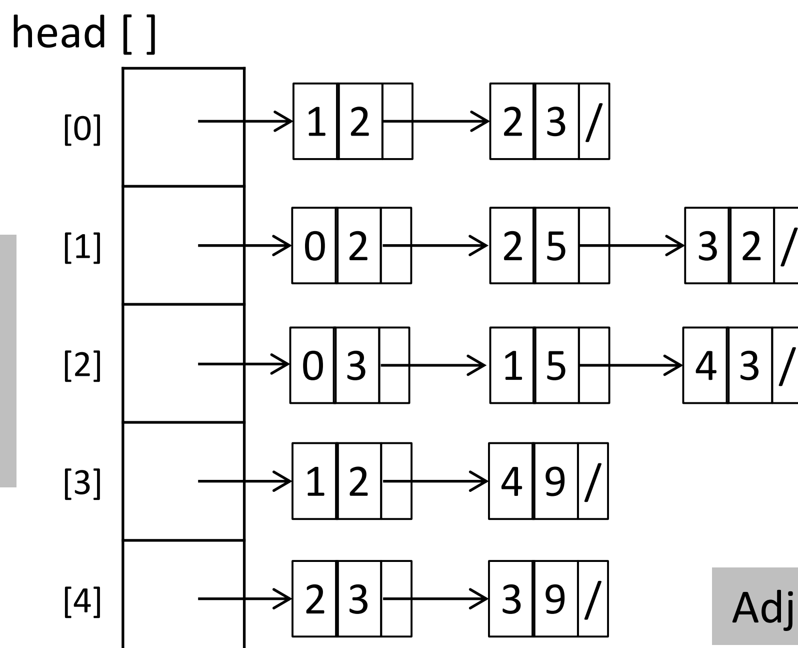


Adjacency List



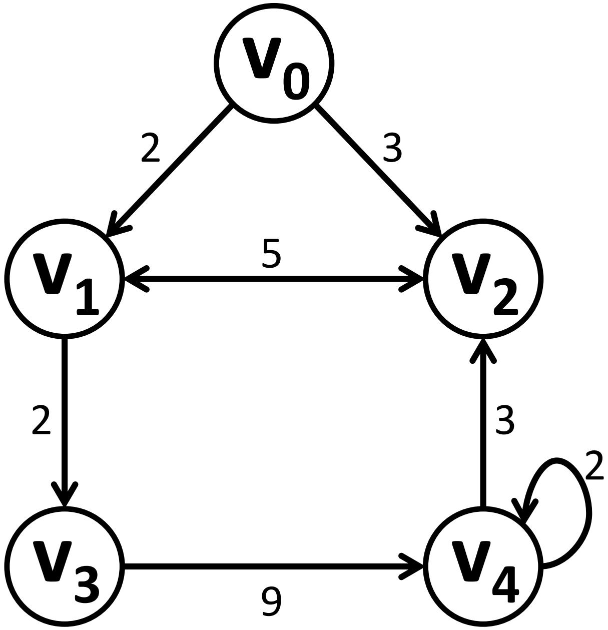
	V_0	V_1	V_2	V_3	V_4
V_0	0	2	3	0	0
V_1	2	0	5	2	0
V_2	3	5	0	0	3
V_3	0	2	0	0	9
V_4	0	0	3	9	0

Adjacency Matrix



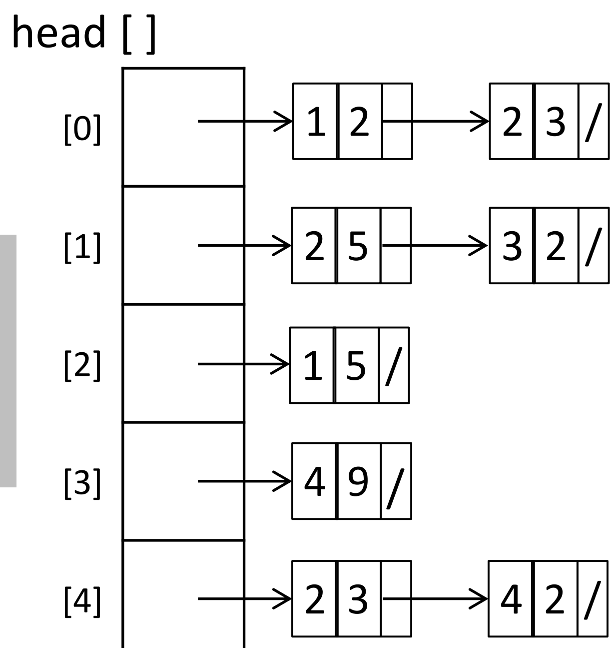

```
struct node
{ int v, w;
  struct node *next;
} *head[5];
```

Adjacency List



	V_0	V_1	V_2	V_3	V_4
V_0	0	2	3	0	0
V_1	0	0	5	2	0
V_2	0	5	0	0	0
V_3	0	0	0	0	9
V_4	0	0	3	0	2

Adjacency Matrix




```
struct node
{ int v, w;
  struct node *next;
} *head[5];
```

Adjacency List

Graph Searching

- Breadth-first search
- Depth-first search

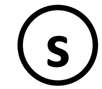
Breadth-first search (BFS)

- Given a graph G = (V,E) and a distinguished source vertex s, BFS systematically explores the edges of G to “discover” every vertex that is reachable from s.
- Discovers all vertices at distance k from a source vertex s before discovering any vertices at distance k + 1.
- It computes the distance (smallest number of edges) from s to each reachable vertex.
- It produces a “breadth-first tree” with root s that contains all reachable vertices.
- It works on both directed and undirected graphs.

Compute BFS - Undirected

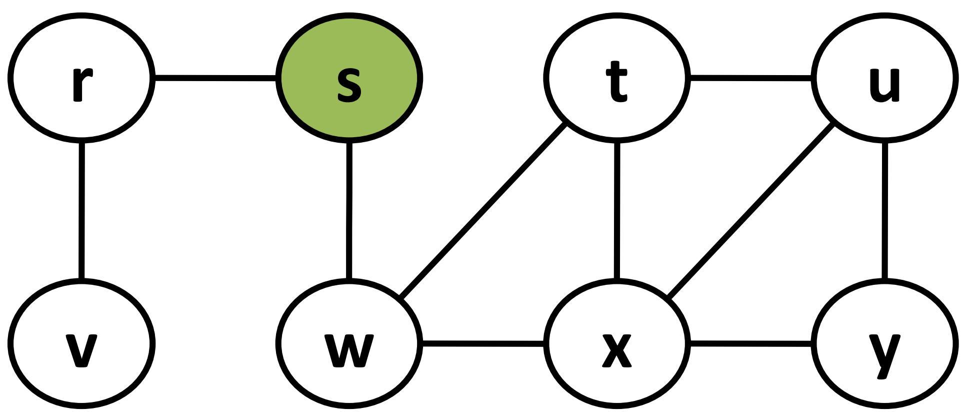
- BFS:

Predecessor sub-graph



Breadth-first tree

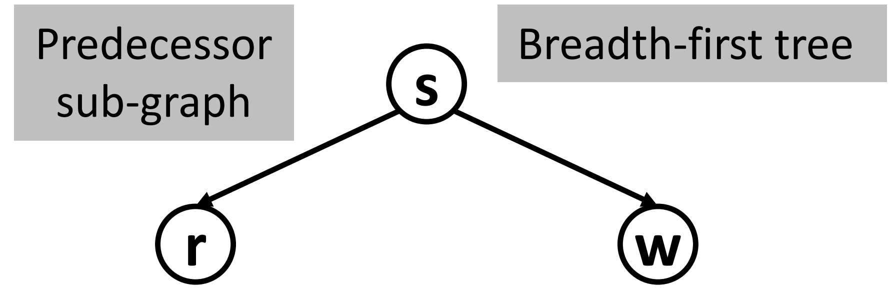
- Queue: s

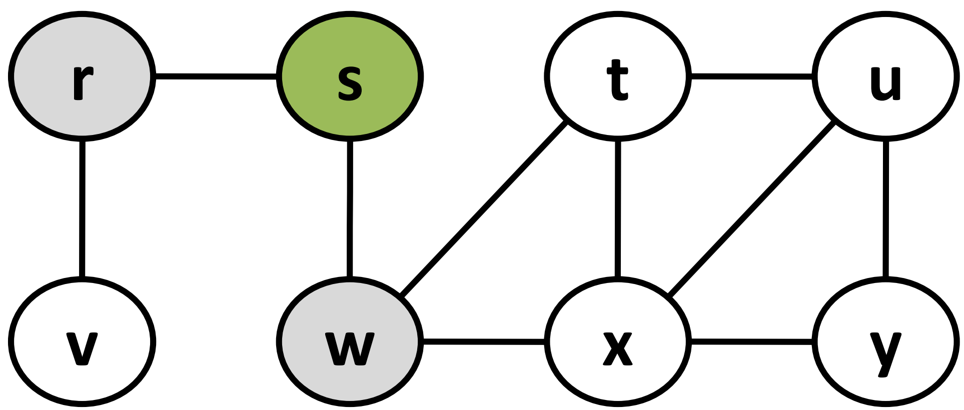


Compute BFS - Undirected

- BFS: s
- Queue:

r	**w**



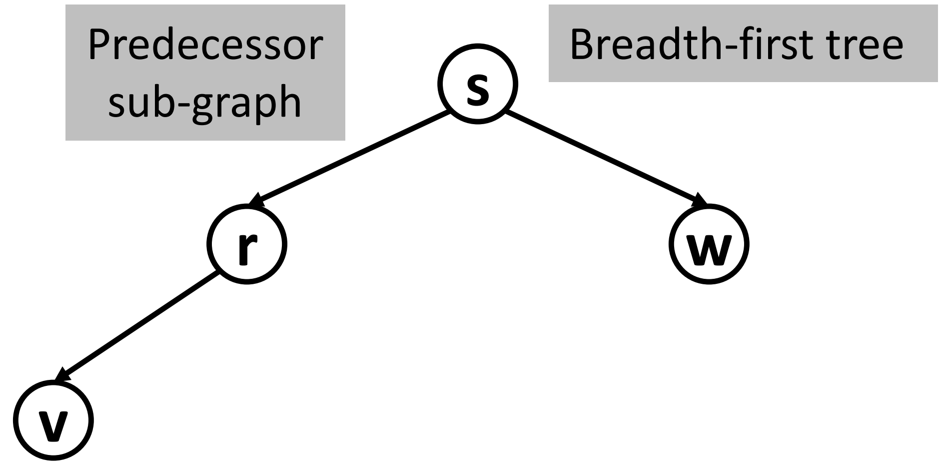


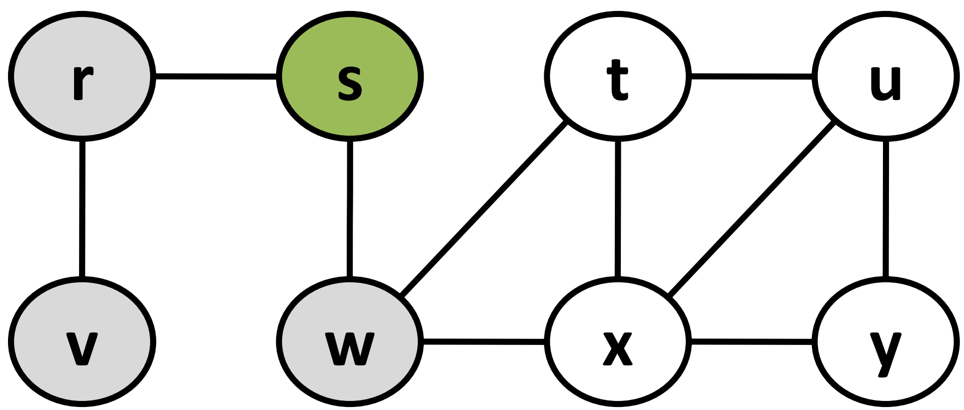
Compute BFS - Undirected

- BFS: s r

- Queue:

w	**v**



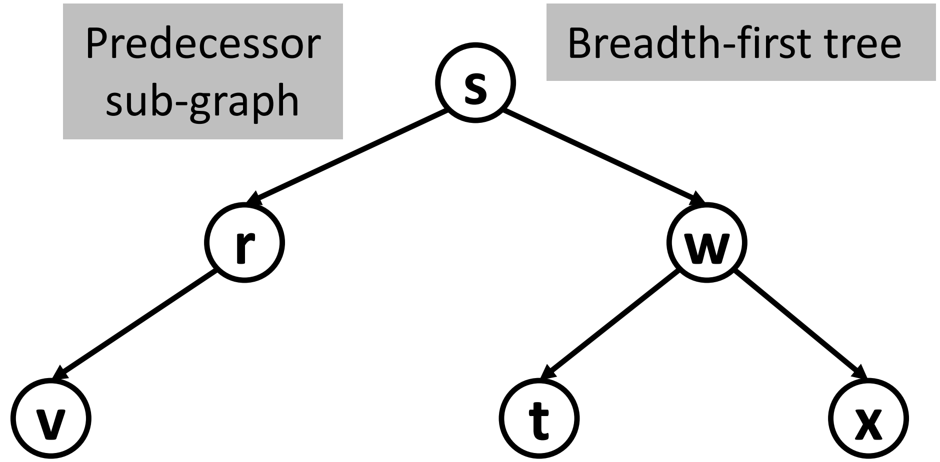


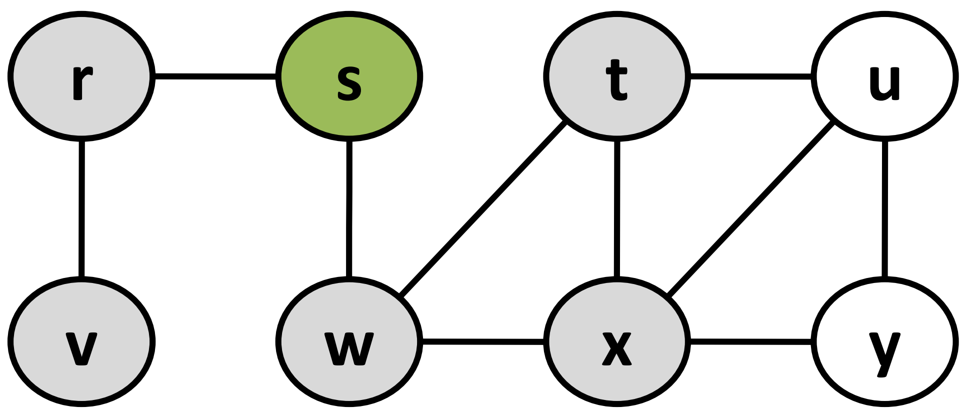
Compute BFS - Undirected

- BFS: s r w

- Queue:

v	**t**	**x**

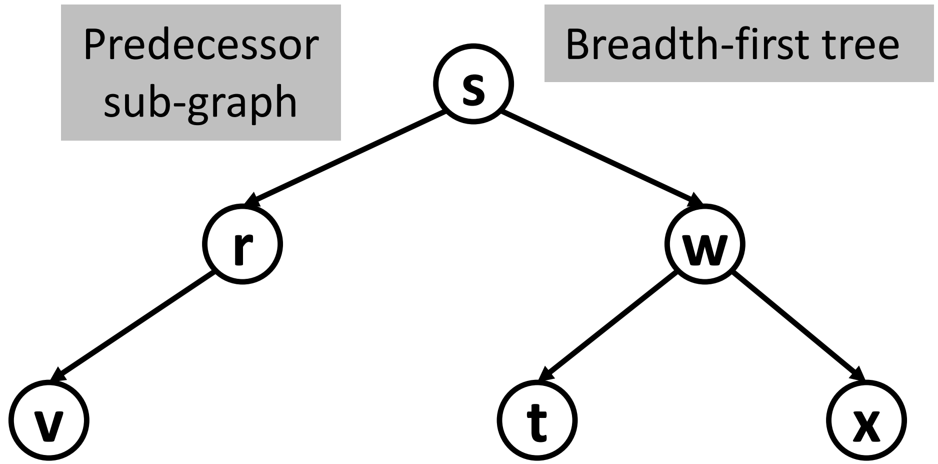


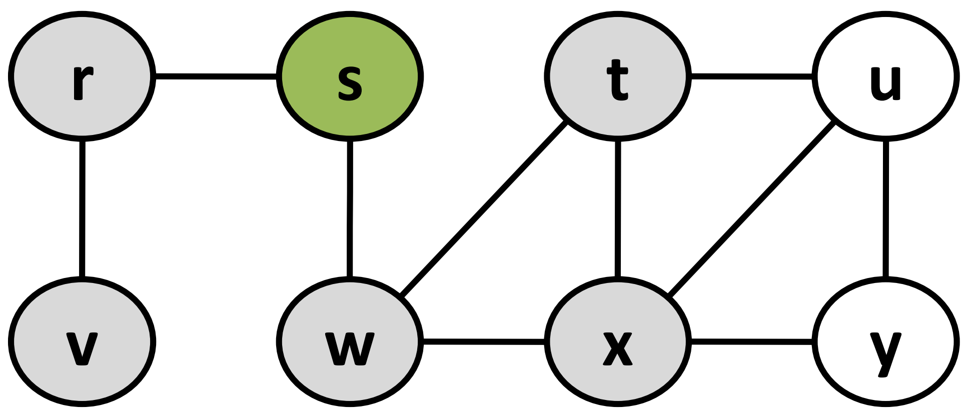


Compute BFS - Undirected

- BFS: s r w v
- Queue:

t	x

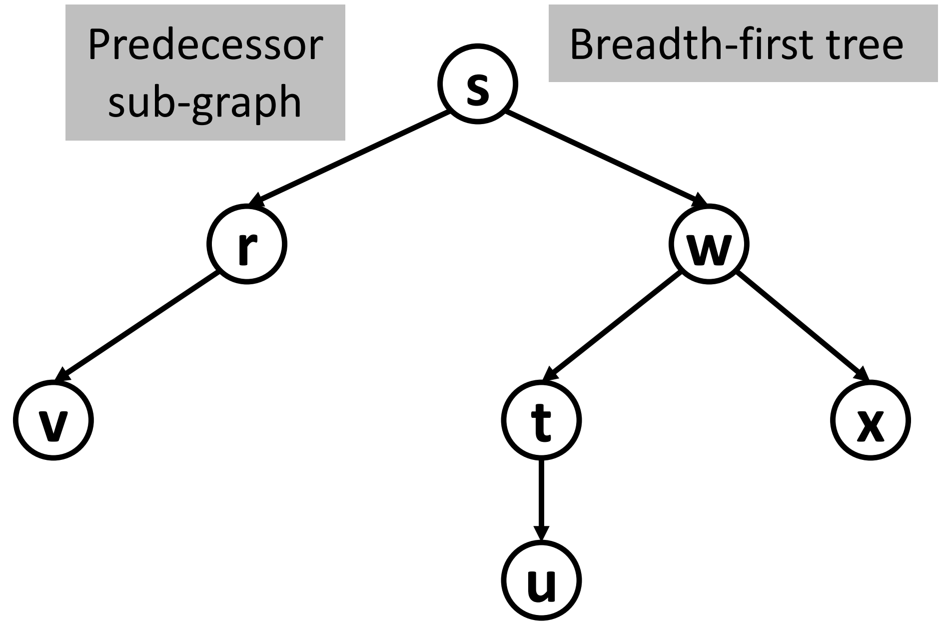


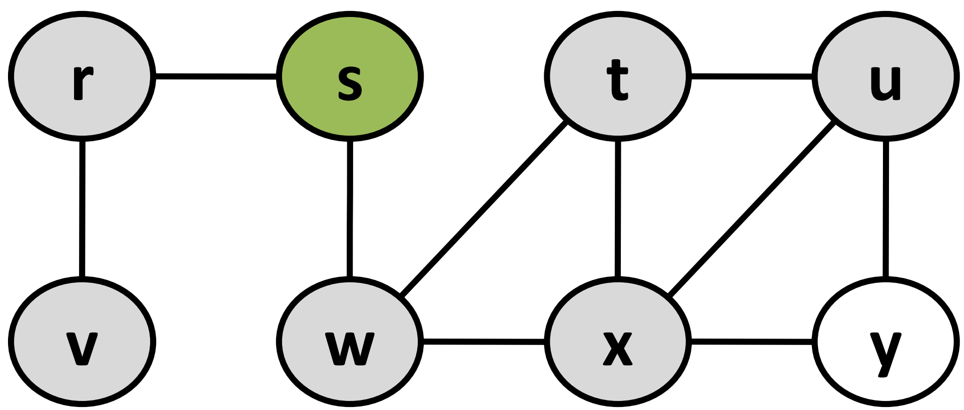


Compute BFS - Undirected

- BFS: s r w v t
- Queue:

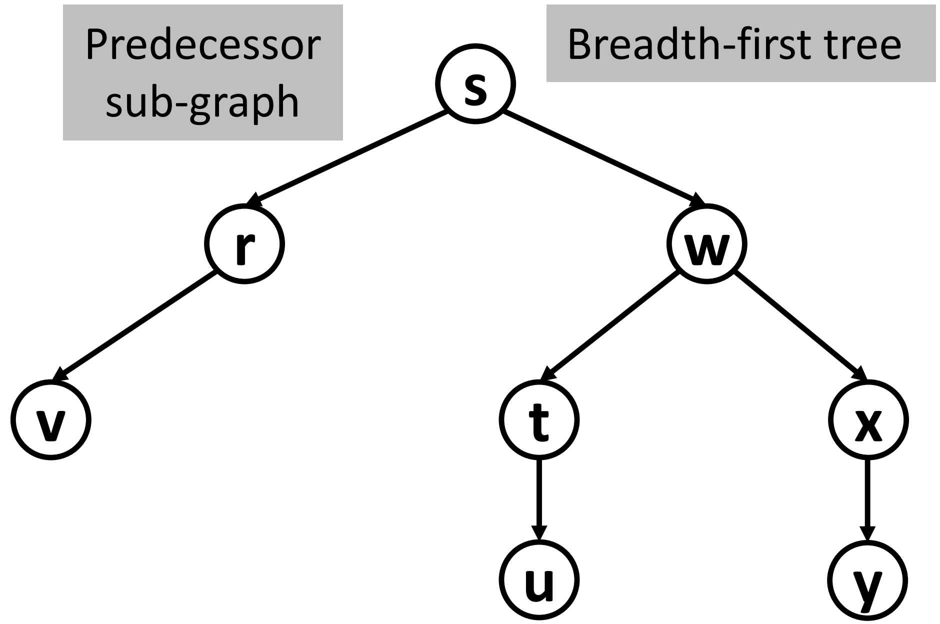
x	**u**

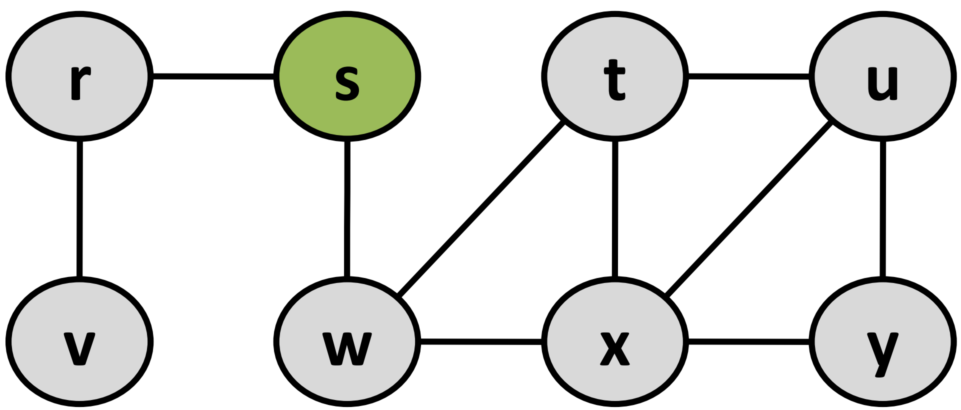




Compute BFS - Undirected

- BFS: s r w v t x
- Queue: u y



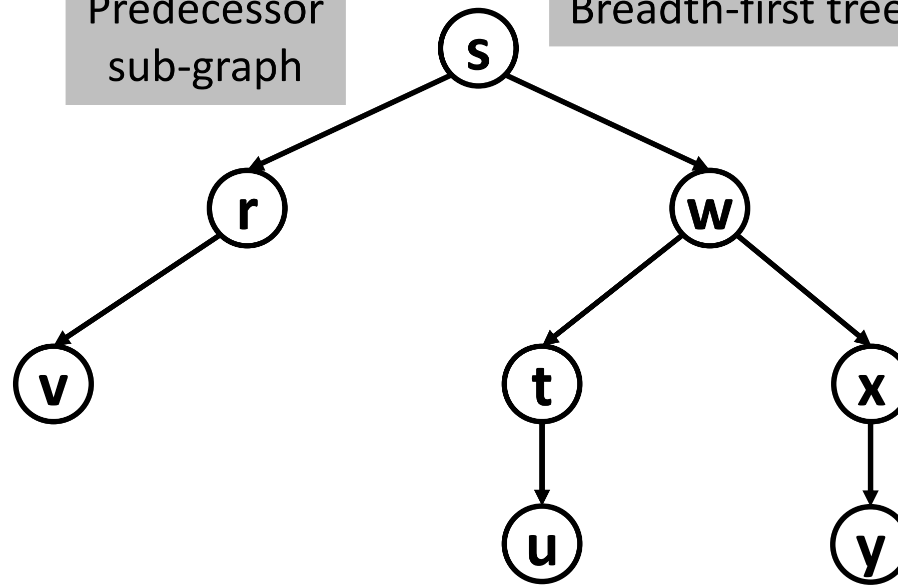


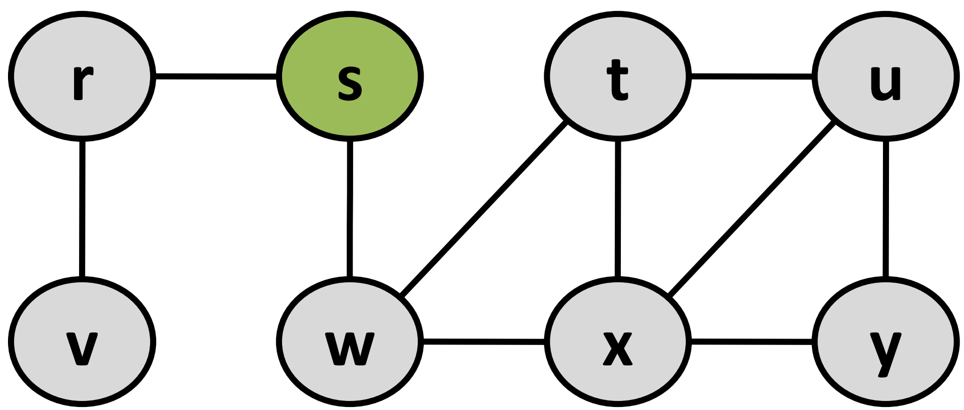
Compute BFS - Undirected

- BFS: s r w v t x u
- Queue: y

Predecessor sub-graph

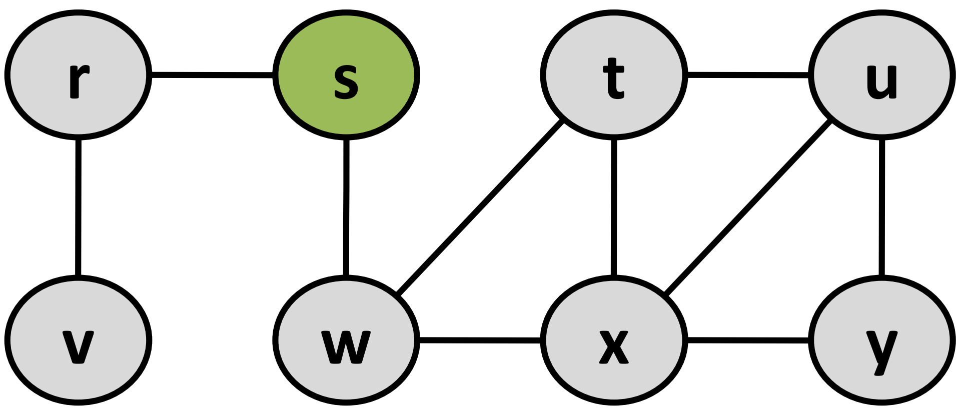
Breadth-first tree



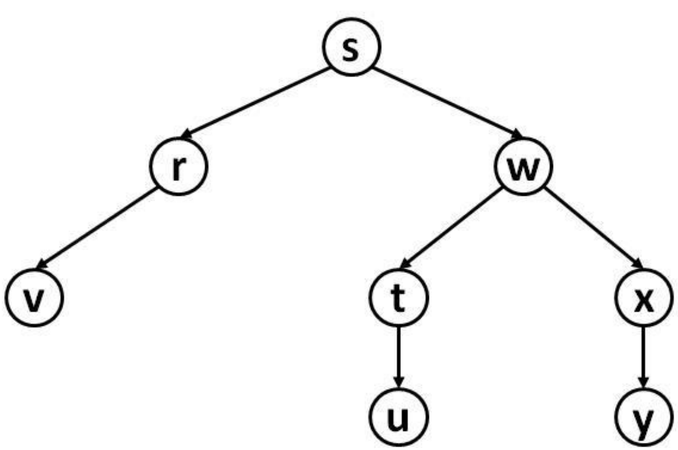


Compute BFS - Undirected

- BFS: s r w v t x u y
- Queue:

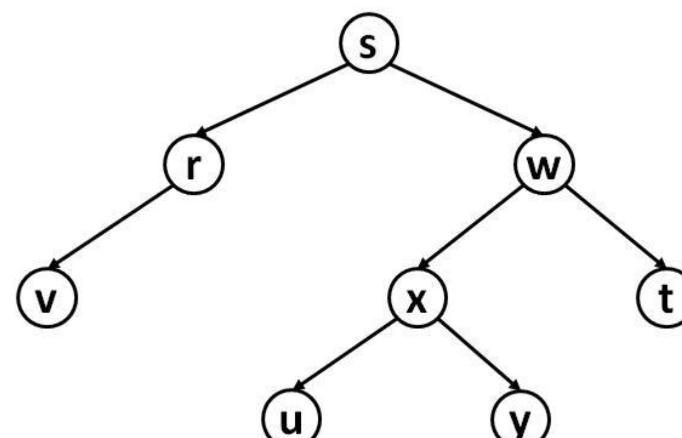


BFS	Queue
	s
s	w r
s w	r x t
s w r	x t v
s w r x	t v y u
s w r x t	v y u
s w r x t v	y u
s w r x t v y	u
s w r x t v y u	



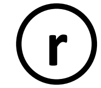
Breadth-first tree

Predecessor sub-graph



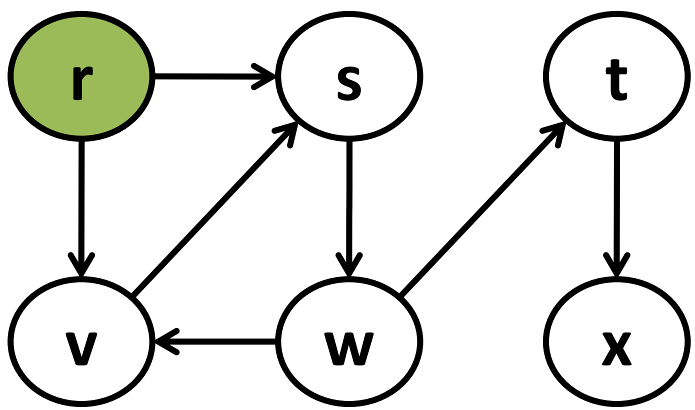
Compute BFS - Directed

Predecessor sub-graph



Breadth-first tree

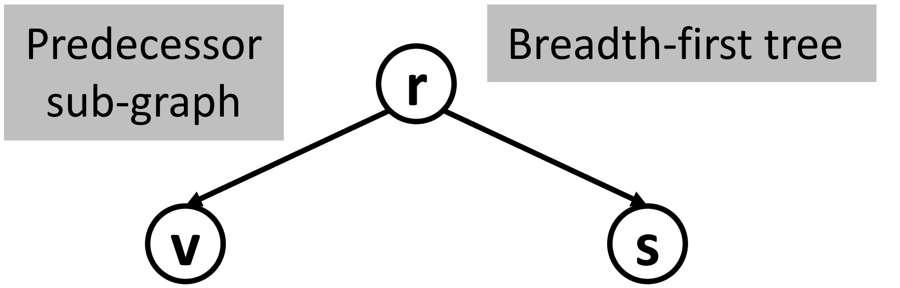
- BFS:
- Queue: r

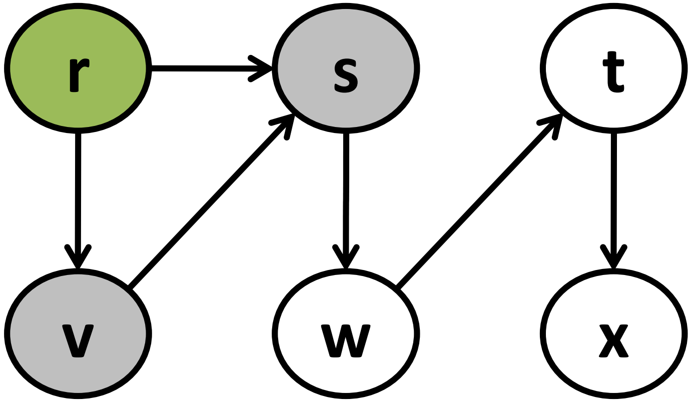


Compute BFS - Directed

- BFS: r
- Queue:

s	v





Compute BFS - Directed

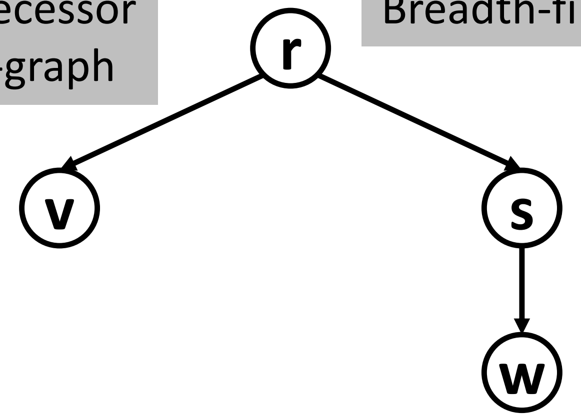
- BFS: r s

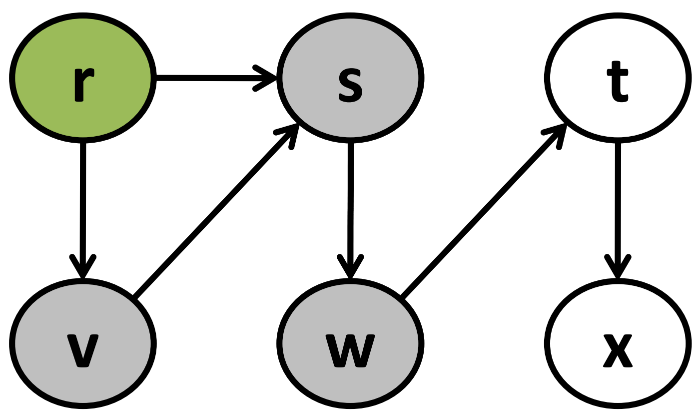
- Queue:

v	**w**

Predecessor sub-graph

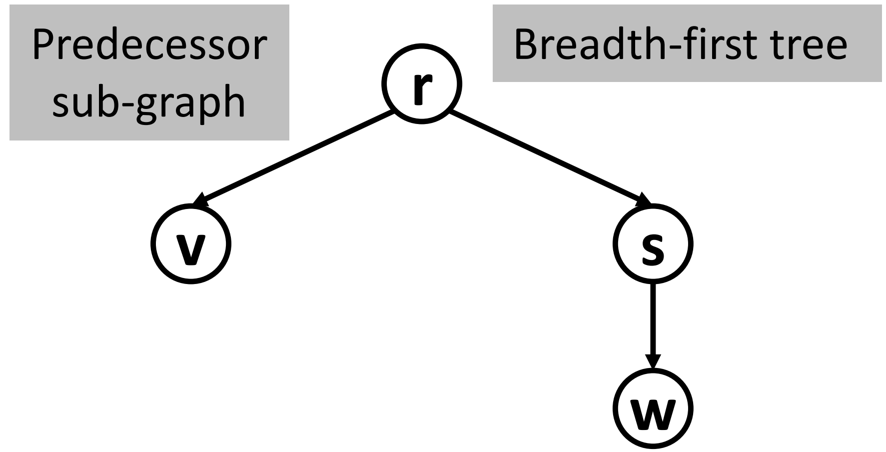
Breadth-first tree

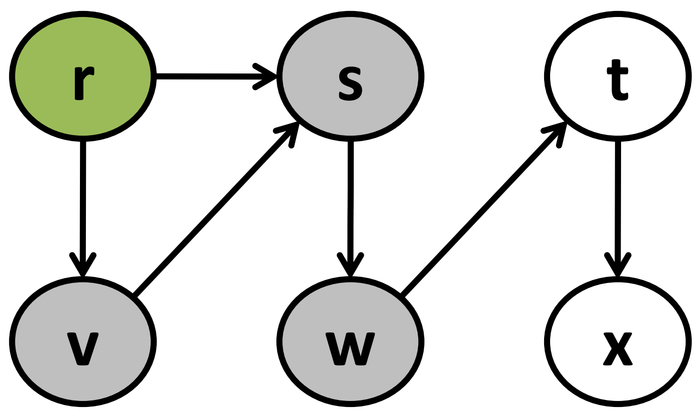




Compute BFS - Directed

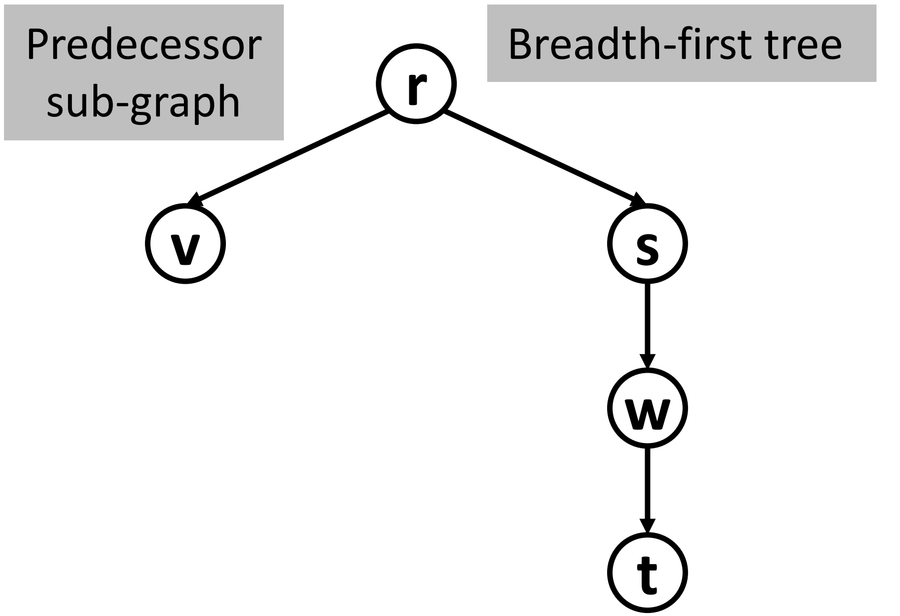
- BFS: r s v
- Queue: w

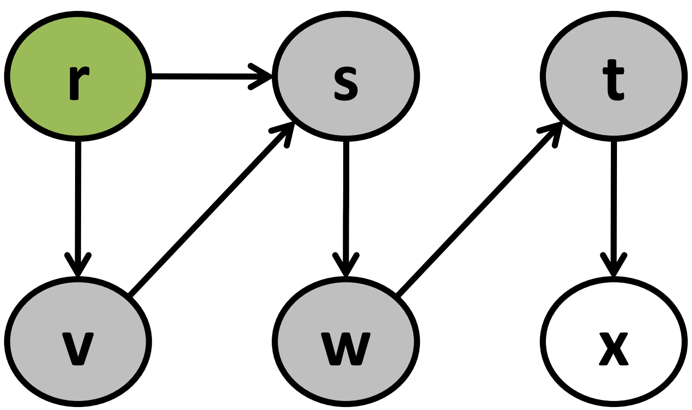




Compute BFS - Directed

- BFS: r s v w
- Queue: t

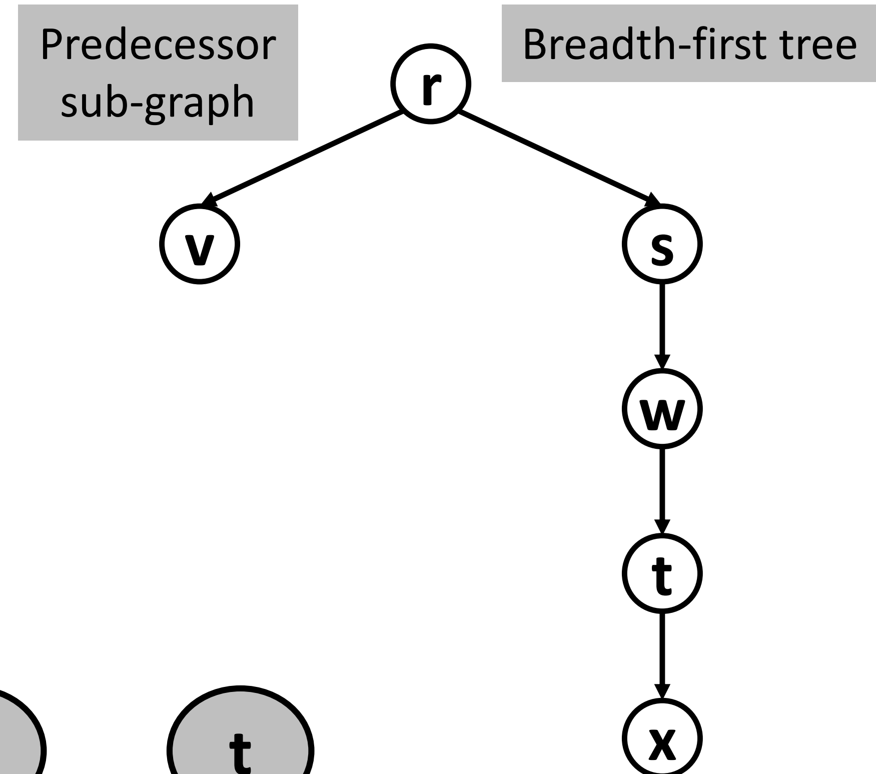


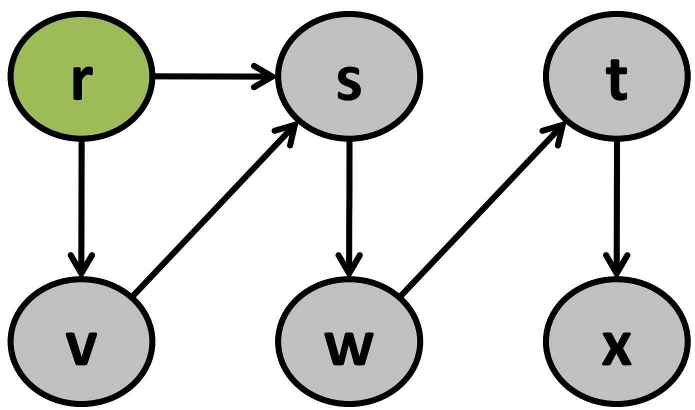


Compute BFS - Directed

- BFS: r s v w t

- Queue: x



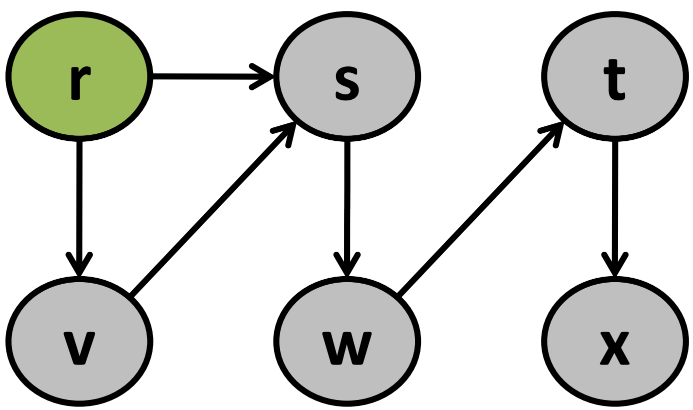


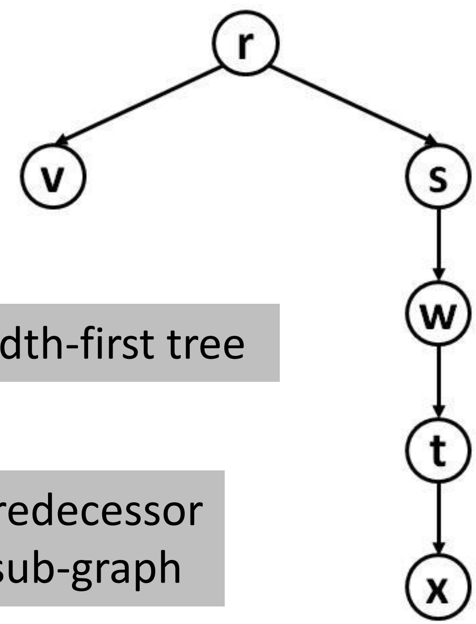
Compute BFS - Directed

- BFS: r s v w t x

- Queue:

BFS	Queue
	r
r	v s
r v	s
r v s	w
r v s w	t
r v s w t	x
r v s w t x	





Breadth-first tree

Predecessor sub-graph

Procedure BFS

- Assumptions:
  - The input graph G = (V,E) is represented using adjacency lists.
  - Each vertex in the graph has following additional attributes.
    - Color: Can be white (undiscovered), gray (may have some adjacent white vertices), or black (all adjacent vertices have been discovered).
    - π: predecessor of a vertex. Can be NIL.
    - d: The distance from the source vertex computed by the algorithm.
  - The queue Q is used to manage the set of gray vertices.

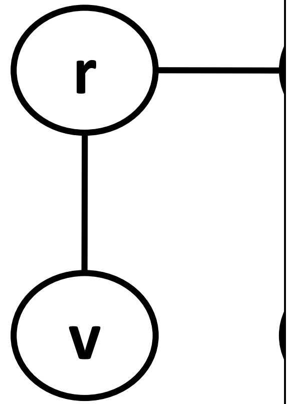
Contd...

$O(V+E)$

```
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 = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    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
```

Execution example

- s is the starting vertex.



for each vertex $u \in G.V - \{s\}$
  $u.color = \text{WHITE}$
  $u.d = \infty$
  $u.\pi = \text{NIL}$
$s.color = \text{GRAY}$
$s.d = 0$
$s.\pi = \text{NIL}$
$Q = \emptyset$
$\text{ENQUEUE}(Q, s)$

Vertex	Color	Distance (d)	Predecessor (π)
r	White	∞	NIL
s	**Gray**	**0**	**NIL**
t	White	∞	NIL
u	White	∞	NIL
v	White	∞	NIL
w	White	∞	NIL
x	White	∞	NIL
y	White	∞	NIL

r $\rightarrow$ s $\rightarrow$ v
s $\rightarrow$ r $\rightarrow$ w
t $\rightarrow$ u $\rightarrow$ w $\rightarrow$ x
u $\rightarrow$ t $\rightarrow$ x $\rightarrow$ y
v $\rightarrow$ r
w $\rightarrow$ s $\rightarrow$ t $\rightarrow$ x
x $\rightarrow$ t $\rightarrow$ u $\rightarrow$ w $\rightarrow$ y
y $\rightarrow$ u $\rightarrow$ x

Q: s

BFS:

Contd…

- s is the starting vertex.

Vertex	Color	Distance (d)	Predecessor (π)
r	White	∞	NIL
s	**Gray**	**0**	**NIL**
t	White	∞	NIL
u	White	∞	NIL
v	White	∞	NIL
w	White	∞	NIL
x	White	∞	NIL
y	White	∞	NIL

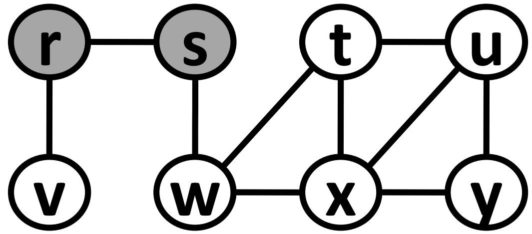
while $Q \neq \emptyset$
    $u = \textsc{Dequeue}(Q)$
    for each $v \in G.Adj[u]$
        if $v.color == \text{WHITE}$
            $v.color = \text{GRAY}$
            $v.d = u.d + 1$
            $v.\pi = u$
            $\textsc{Enqueue}(Q, v)$
    $u.color = \text{BLACK}$

$r \rightarrow s \rightarrow v$
$s \rightarrow r \rightarrow w$
$t \rightarrow u \rightarrow w \rightarrow x$
$u \rightarrow t \rightarrow x \rightarrow y$
$v \rightarrow r$
$w \rightarrow s \rightarrow t \rightarrow x$
$x \rightarrow t \rightarrow u \rightarrow w \rightarrow y$
$y \rightarrow u \rightarrow x$

Q: s

BFS:

Contd…




```
while Q ≠ ∅
    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
```

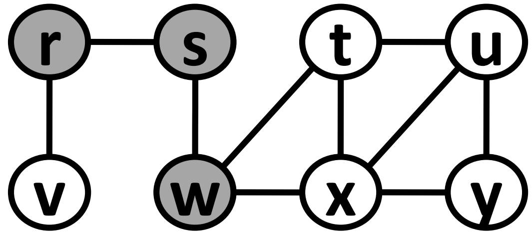
Vertex	Color	Distance (d)	Predecessor (π)
r	**Gray**	**1**	**s**
s	**Gray**	**0**	**NIL**
t	White	∞	NIL
u	White	∞	NIL
v	White	∞	NIL
w	White	∞	NIL
x	White	∞	NIL
y	White	∞	NIL

r → s → v
s → r → w
t → u → w → x
u → t → x → y
v → r
w → s → t → x
x → t → u → w → y
y → u → x

Q: r

BFS: s

Contd…



Vertex	Color	Distance (d)	Predecessor (π)
r	**Gray**	**1**	**s**
s	**Gray**	**0**	**NIL**
t	White	∞	NIL
u	White	∞	NIL
v	White	∞	NIL
w	**Gray**	**1**	**s**
x	White	∞	NIL
y	White	∞	NIL

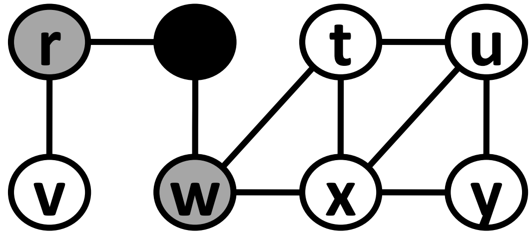
while $Q \neq \emptyset$
    $u = \text{DEQUEUE}(Q)$
    for each $v \in G.Adj[u]$
        if $v.color == \text{WHITE}$
            $v.color = \text{GRAY}$
            $v.d = u.d + 1$
            $v.\pi = u$
            $\text{ENQUEUE}(Q, v)$
    $u.color = \text{BLACK}$

r $\rightarrow$ s $\rightarrow$ v
s $\rightarrow$ r $\rightarrow$ w
t $\rightarrow$ u $\rightarrow$ w $\rightarrow$ x
u $\rightarrow$ t $\rightarrow$ x $\rightarrow$ y
v $\rightarrow$ r
w $\rightarrow$ s $\rightarrow$ t $\rightarrow$ x
x $\rightarrow$ t $\rightarrow$ u $\rightarrow$ w $\rightarrow$ y
y $\rightarrow$ u $\rightarrow$ x

Q: | r | w |

BFS: s

Contd…



Vertex	Color	Distance (d)	Predecessor (π)
r	**Gray**	**1**	**s**
s	**Black**	**0**	**NIL**
t	White	∞	NIL
u	White	∞	NIL
v	White	∞	NIL
w	**Gray**	**1**	**s**
x	White	∞	NIL
y	White	∞	NIL

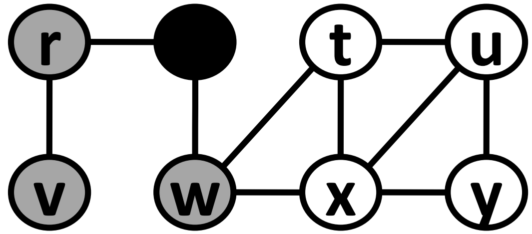
while $Q \neq \emptyset$
  $u = \text{DEQUEUE}(Q)$
  for each $v \in G.Adj[u]$
    if $v.color == \text{WHITE}$
      $v.color = \text{GRAY}$
      $v.d = u.d + 1$
      $v.\pi = u$
      $\text{ENQUEUE}(Q, v)$
  $u.color = \text{BLACK}$

r	→ s → v
s	→ r → w
t	→ u → w → x
u	→ t → x → y
v	→ r
w	→ s → t → x
x	→ t → u → w → y
y	→ u → x

Q: | r | w |

BFS: s

Contd…



Vertex	Color	Distance (d)	Predecessor (π)
r	**Gray**	**1**	**s**
s	**Black**	**0**	**NIL**
t	White	∞	NIL
u	White	∞	NIL
v	**Gray**	**2**	**r**
w	**Gray**	**1**	**s**
x	White	∞	NIL
y	White	∞	NIL

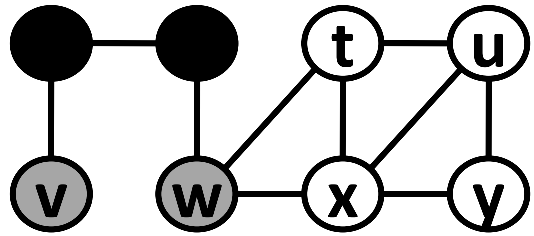
```
while Q ≠ Ø
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

r → s → v
s → r → w
t → u → w → x
u → t → x → y
v → r
w → s → t → x
x → t → u → w → y
y → u → x

Q: | w | v |

BFS: s r

Contd…



while $Q \neq \emptyset$
- $u = \text{DEQUEUE}(Q)$
- **for each** $v \in G.Adj[u]$
  - **if** $v.color == \text{WHITE}$
    - $v.color = \text{GRAY}$
    - $v.d = u.d + 1$
    - $v.\pi = u$
    - $\text{ENQUEUE}(Q, v)$
- $u.color = \text{BLACK}$

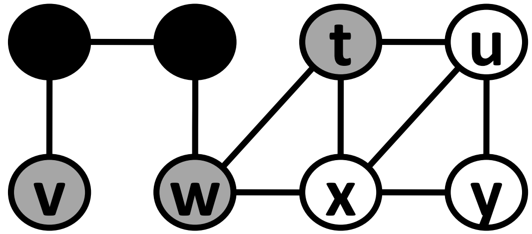
Vertex	Color	Distance (d)	Predecessor (π)
r	**Black**	**1**	**s**
s	**Black**	**0**	**NIL**
t	White	∞	NIL
u	White	∞	NIL
v	**Gray**	**2**	**r**
w	**Gray**	**1**	**s**
x	White	∞	NIL
y	White	∞	NIL

r	$\rightarrow$ s $\rightarrow$ v
s	$\rightarrow$ r $\rightarrow$ w
t	$\rightarrow$ u $\rightarrow$ w $\rightarrow$ x
u	$\rightarrow$ t $\rightarrow$ x $\rightarrow$ y
v	$\rightarrow$ r
w	$\rightarrow$ s $\rightarrow$ t $\rightarrow$ x
x	$\rightarrow$ t $\rightarrow$ u $\rightarrow$ w $\rightarrow$ y
y	$\rightarrow$ u $\rightarrow$ x

Q: | w | v |

BFS: s r

Contd…




```
while Q ≠ ∅
    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
```

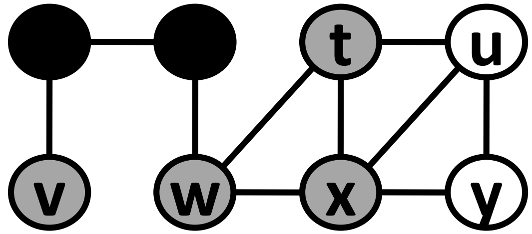
Vertex	Color	Distance (d)	Predecessor (π)
r	**Black**	**1**	**s**
s	**Black**	**0**	**NIL**
t	**Gray**	**2**	**w**
u	White	∞	NIL
v	**Gray**	**2**	**r**
w	**Gray**	**1**	**s**
x	White	∞	NIL
y	White	∞	NIL

r → s → v
s → r → w
t → u → w → x
u → t → x → y
v → r
w → s → t → x
x → t → u → w → y
y → u → x

Q: | v | t |

BFS: s r w

Contd…




```
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
```

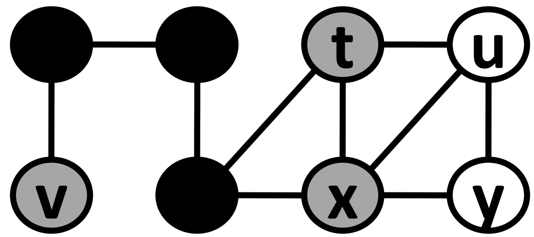
Vertex	Color	Distance (d)	Predecessor (π)
r	**Black**	1	**s**
s	**Black**	**0**	**NIL**
t	**Gray**	2	**w**
u	White	∞	NIL
v	**Gray**	2	**r**
w	**Gray**	1	**s**
x	**Gray**	2	**w**
y	White	∞	NIL

r → s → v
s → r → w
t → u → w → x
u → t → x → y
v → r
w → s → t → x
x → t → u → w → y
y → u → x

Q: | v | t | x |

BFS: s r w

Contd…



Vertex	Color	Distance (d)	Predecessor (π)
r	**Black**	**1**	**s**
s	**Black**	**0**	**NIL**
t	**Gray**	**2**	**w**
u	White	∞	NIL
v	**Gray**	**2**	**r**
w	**Black**	**1**	**s**
x	**Gray**	**2**	**w**
y	White	∞	NIL

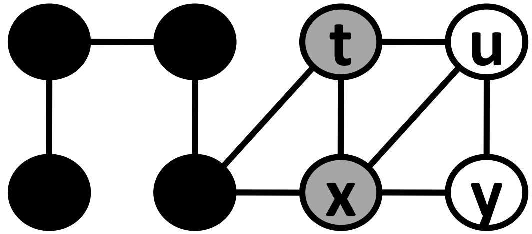
while $Q \neq \emptyset$
    $u = \text{DEQUEUE}(Q)$
    for each $v \in G.Adj[u]$
        if $v.color == \text{WHITE}$
            $v.color = \text{GRAY}$
            $v.d = u.d + 1$
            $v.\pi = u$
            $\text{ENQUEUE}(Q, v)$
    $u.color = \text{BLACK}$

r → s → v
s → r → w
t → u → w → x
u → t → x → y
v → r
w → s → t → x
x → t → u → w → y
y → u → x

Q: | v | t | x |

BFS: s r w

Contd…



while $Q \neq \emptyset$
  $u = \text{DEQUEUE}(Q)$
  for each $v \in G.Adj[u]$
    if $v.color == \text{WHITE}$
      $v.color = \text{GRAY}$
      $v.d = u.d + 1$
      $v.\pi = u$
      $\text{ENQUEUE}(Q, v)$
  $u.color = \text{BLACK}$

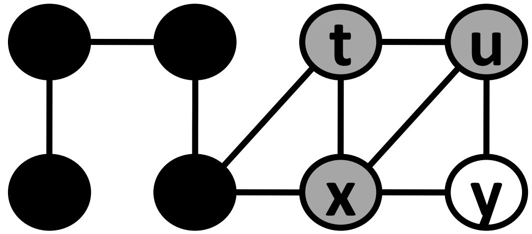
Vertex	Color	Distance (d)	Predecessor (π)
r	**Black**	**1**	**s**
s	**Black**	**0**	**NIL**
t	**Gray**	**2**	**w**
u	White	∞	NIL
v	**Black**	**2**	**r**
w	**Black**	**1**	**s**
x	**Gray**	**2**	**w**
y	White	∞	NIL

r → s → v
s → r → w
t → u → w → x
u → t → x → y
v → r
w → s → t → x
x → t → u → w → y
y → u → x

Q: | t | x |

BFS: s r w v

Contd...




```
while Q ≠ ∅
    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
```

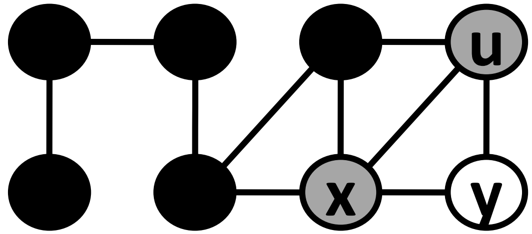
Vertex	Color	Distance (d)	Predecessor (π)
r	Black	1	s
s	Black	0	NIL
t	Gray	2	w
u	Gray	3	t
v	Black	2	r
w	Black	1	s
x	Gray	2	w
y	White	∞	NIL

r → s → v
s → r → w
t → u → w → x
u → t → x → y
v → r
w → s → t → x
x → t → u → w → y
y → u → x

Q: | x | u |

BFS: s r w v t

Contd…



Vertex	Color	Distance (d)	Predecessor (π)
r	Black	1	s
s	Black	0	NIL
t	Black	2	w
u	Gray	3	t
v	Black	2	r
w	Black	1	s
x	Gray	2	w
y	White	∞	NIL

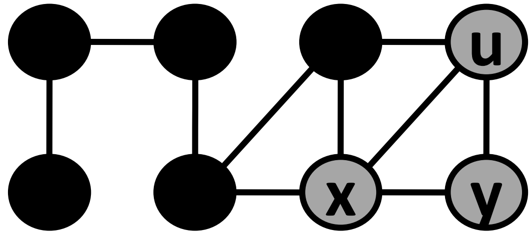
```
while Q ≠ Ø
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

r	→ s → v
s	→ r → w
t	→ u → w → x
u	→ t → x → y
v	→ r
w	→ s → t → x
x	→ t → u → w → y
y	→ u → x

Q: | x | u |

BFS: s r w v t

Contd…



while $Q \neq \emptyset$
$\quad u = \text{DEQUEUE}(Q)$
$\quad$ for each $v \in G.Adj[u]$
$\quad\quad$ if $v.color == \text{WHITE}$
$\quad\quad\quad v.color = \text{GRAY}$
$\quad\quad\quad v.d = u.d + 1$
$\quad\quad\quad v.\pi = u$
$\quad\quad\quad \text{ENQUEUE}(Q, v)$
$\quad u.color = \text{BLACK}$

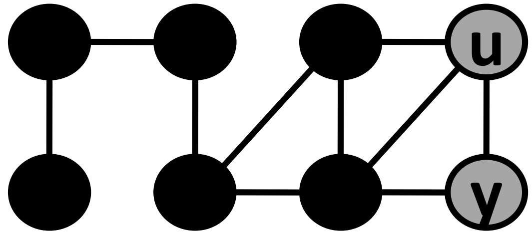
Vertex	Color	Distance (d)	Predecessor (π)
r	Black	1	s
s	Black	0	NIL
t	Black	2	w
u	Gray	3	t
v	Black	2	r
w	Black	1	s
x	Gray	2	w
y	Gray	3	x

r → s → v
s → r → w
t → u → w → x
u → t → x → y
v → r
w → s → t → x
x → t → u → w → y
y → u → x

Q: | u | y |

BFS: s r w v t x

Contd…




```
while Q ≠ ∅
    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
```

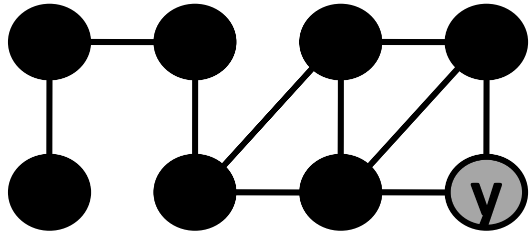
Vertex	Color	Distance (d)	Predecessor (π)
r	Black	1	s
s	Black	0	NIL
t	Black	2	w
u	Gray	3	t
v	Black	2	r
w	Black	1	s
x	Black	2	w
y	Gray	3	x

r → s → v
s → r → w
t → u → w → x
u → t → x → y
v → r
w → s → t → x
x → t → u → w → y
y → u → x

Q: | u | y |

BFS: s r w v t x

Contd...



Vertex	Color	Distance (d)	Predecessor (π)
r	Black	1	s
s	Black	0	NIL
t	Black	2	w
u	Black	3	t
v	Black	2	r
w	Black	1	s
x	Black	2	w
y	Gray	3	x

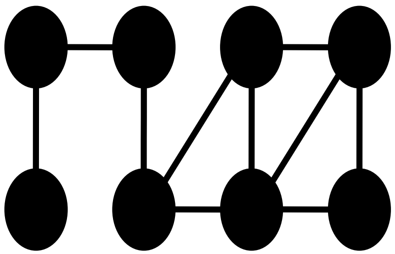
while $Q \neq \emptyset$
  $u = \text{DEQUEUE}(Q)$
  for each $v \in G.Adj[u]$
    if $v.color == \text{WHITE}$
      $v.color = \text{GRAY}$
      $v.d = u.d + 1$
      $v.\pi = u$
      $\text{ENQUEUE}(Q, v)$
  $u.color = \text{BLACK}$

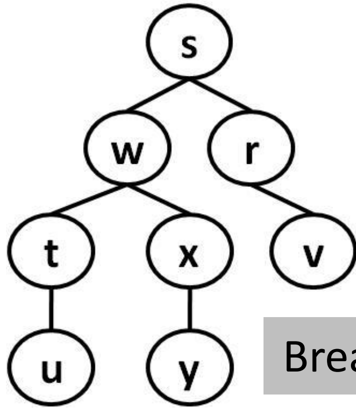
r → s → v
s → r → w
t → u → w → x
u → t → x → y
v → r
w → s → t → x
x → t → u → w → y
y → u → x

Q: y

BFS: s r w v t x u

Contd...





Breadth-first tree

```
while Q ≠ ∅
    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
```

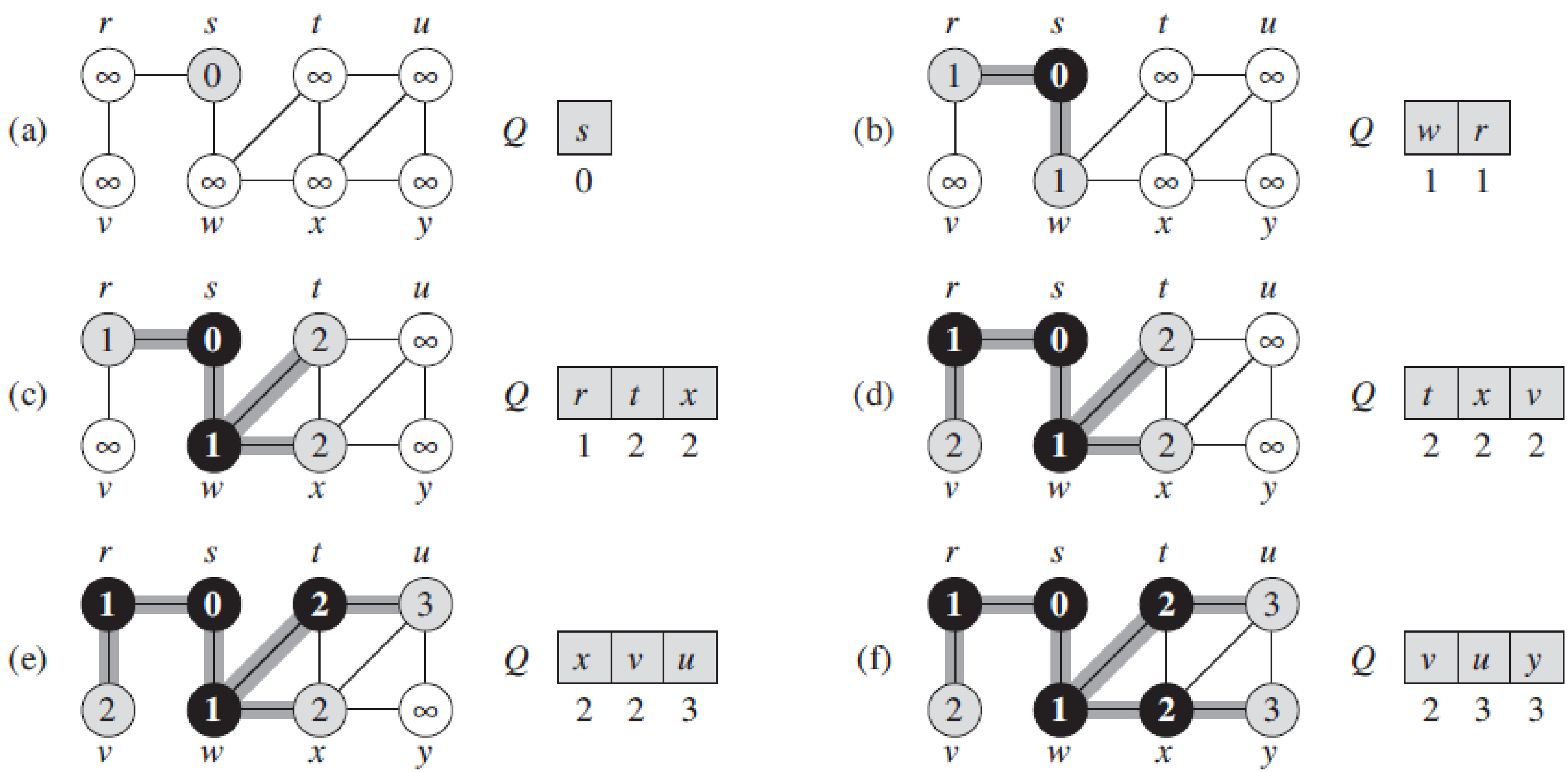
Vertex	Color	Distance (d)	Predecessor (π)
r	Black	1	s
s	Black	0	NIL
t	Black	2	w
u	Black	3	t
v	Black	2	r
w	Black	1	s
x	Black	2	w
y	Black	3	x

r → s → v
s → r → w
t → u → w → x
u → t → x → y
v → r
w → s → t → x
x → t → u → w → y
y → u → x

Q: ϕ

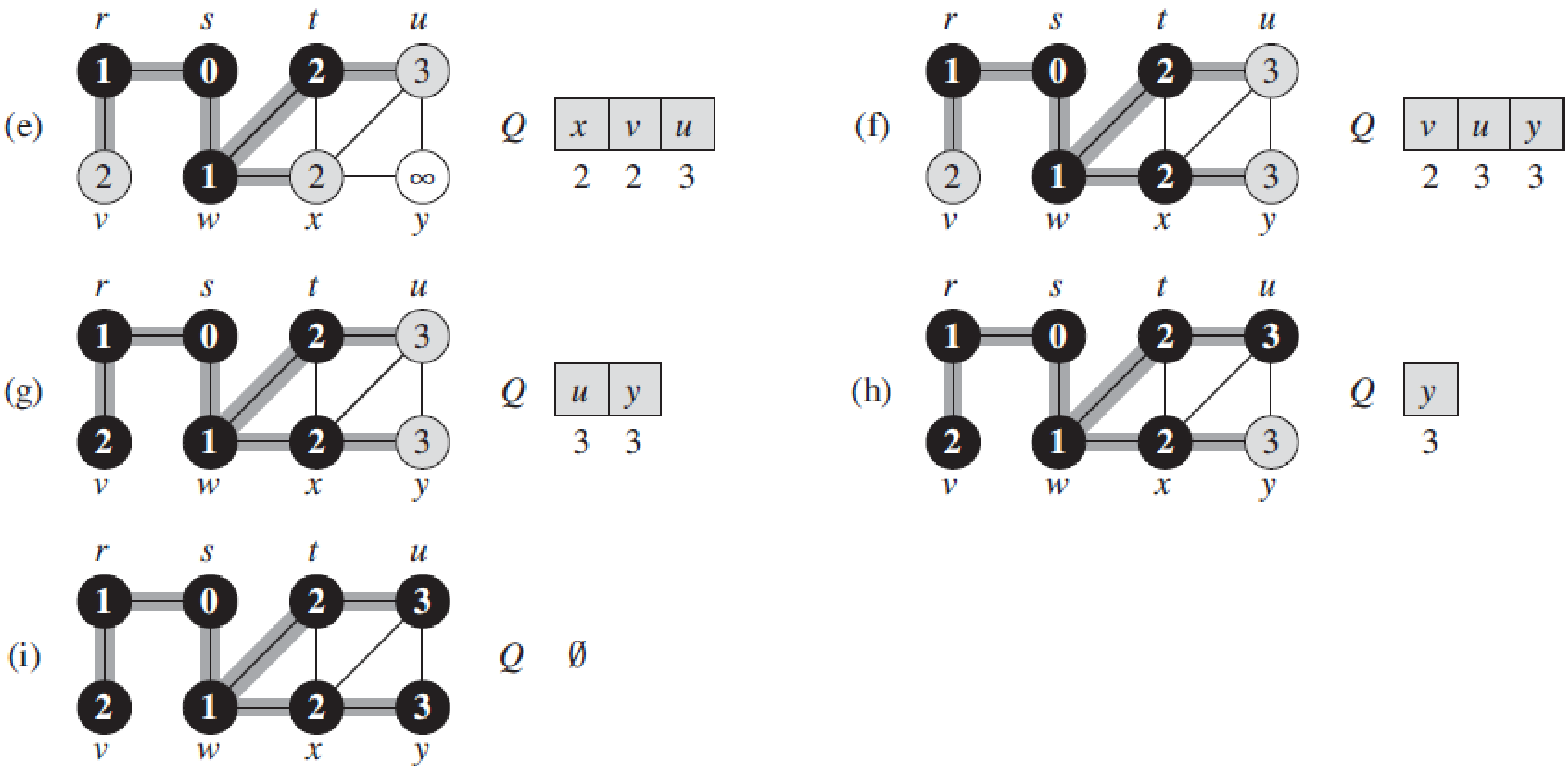
BFS: s r w v t x u y

Contd…



(a)
r
s
t
u
∞
0
∞
∞
∞
∞
∞
∞
v
w
x
y
Q
s
0
(b)
r
s
t
u
1
0
∞
∞
∞
1
∞
∞
v
w
x
y
Q
w
r
1 1
(c)
r
s
t
u
1
0
2
∞
∞
1
2
∞
v
w
x
y
Q
r
t
x
1 2 2
(d)
r
s
t
u
1
0
2
∞
2
1
2
∞
v
w
x
y
Q
t
x
v
2 2 2
(e)
r
s
t
u
1
0
2
3
2
1
2
∞
v
w
x
y
Q
x
v
u
2 2 3
(f)
r
s
t
u
1
0
2
3
2
1
2
3
v
w
x
y
Q
v
u
y
2 3 3

Contd...


(e)
r s t u
1 0 2 3
2 1 2 ∞
v w x y
Q x v u
2 2 3
(f)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q v u y
2 3 3
(g)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q u y
3 3
(h)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q y
3
(i)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q Ø

Depth-first search (DFS)

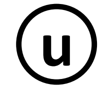
- Search "deeper" in the graph whenever possible.
- If any undiscovered vertices remain, then DFS selects one of them as a new-source, and it repeats the search from that source.
- The algorithm continues until it has discovered every vertex.
- It produces a "depth-first forest" comprising several "depth-first trees".
- It works on both directed and undirected graphs.

Procedure DFS

- Assumptions:
  - The input graph G = (V,E) is represented using adjacency lists.
  - Each vertex in the graph has following additional attributes.
    - Color: Can be white (undiscovered), gray (when discovered), or black (all adjacent vertices have been examined completely).
    - π: predecessor of a vertex. Can be NIL.
    - d: Timestamp to record when the vertex is first discovered.
    - f: Timestamp to record when the vertex is examined completely.

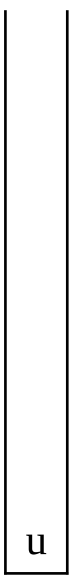
Compute DFS - Undirected

Predecessor sub-graph

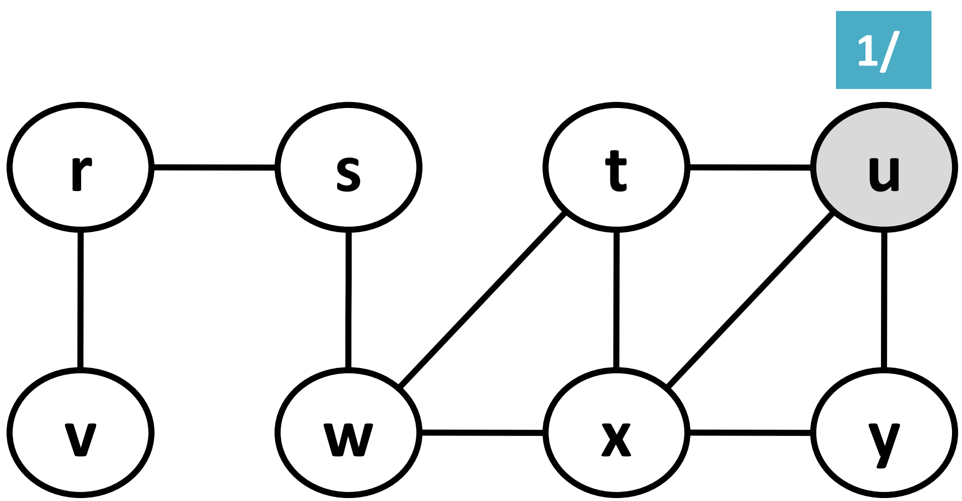


Depth-first forest

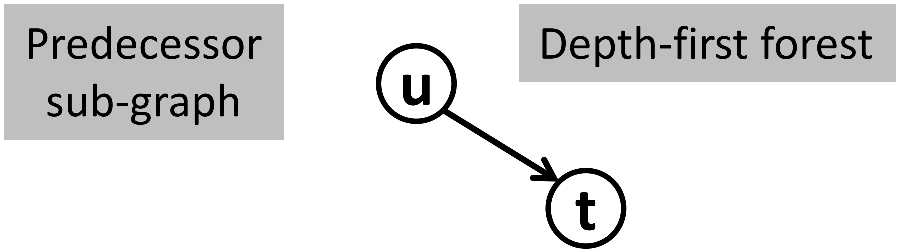
- DFS: u



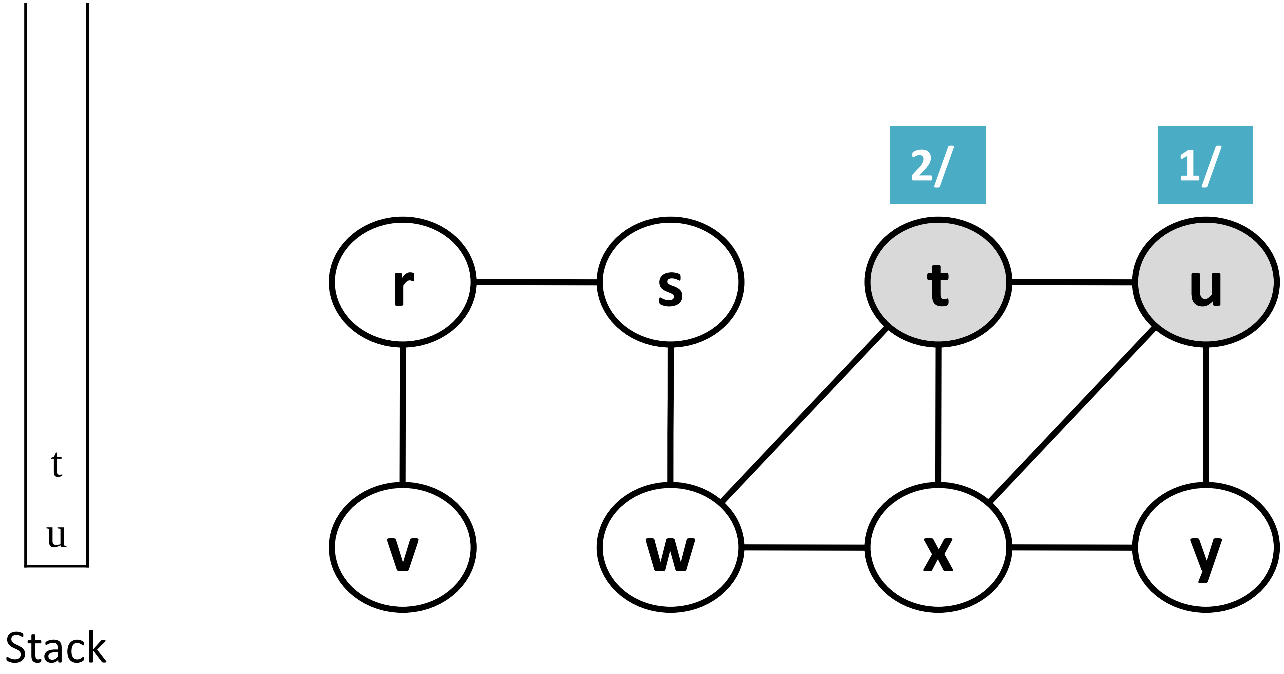
Stack



Compute DFS - Undirected

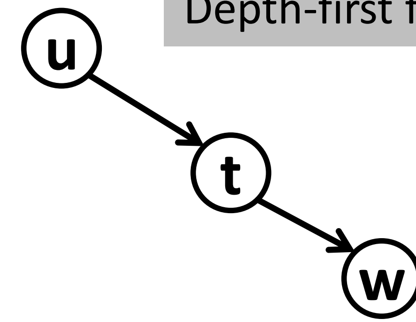


- DFS: u t



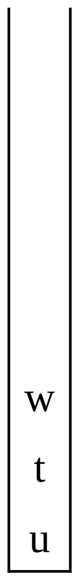
Compute DFS - Undirected

Predecessor sub-graph

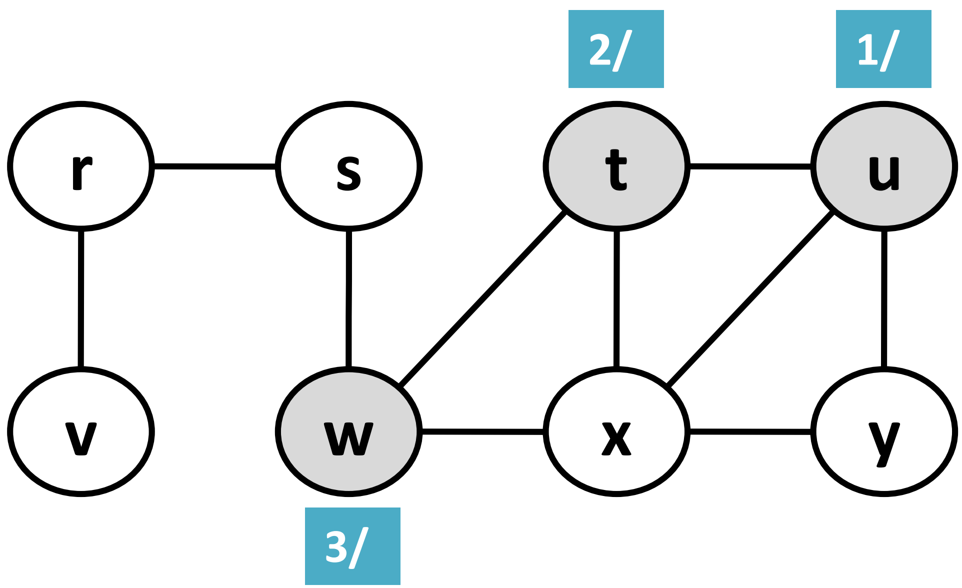


Depth-first forest

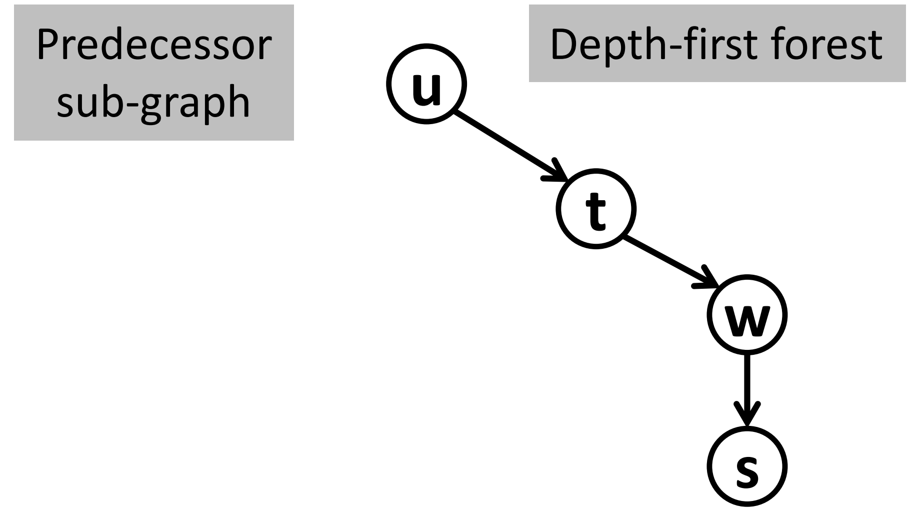
- DFS: u t w



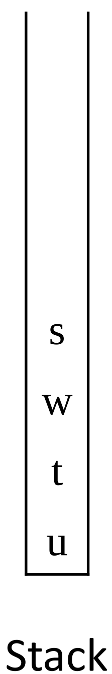
Stack

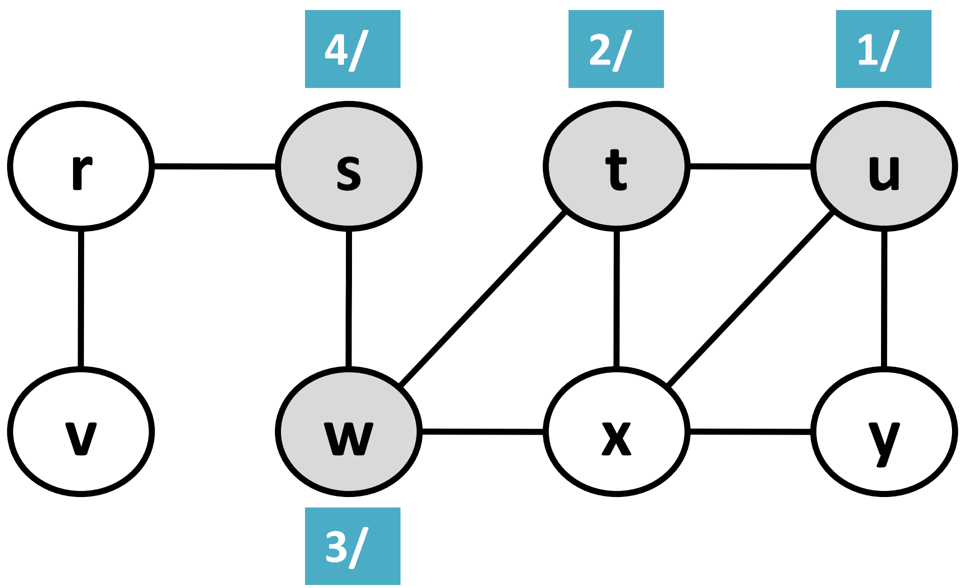


Compute DFS - Undirected



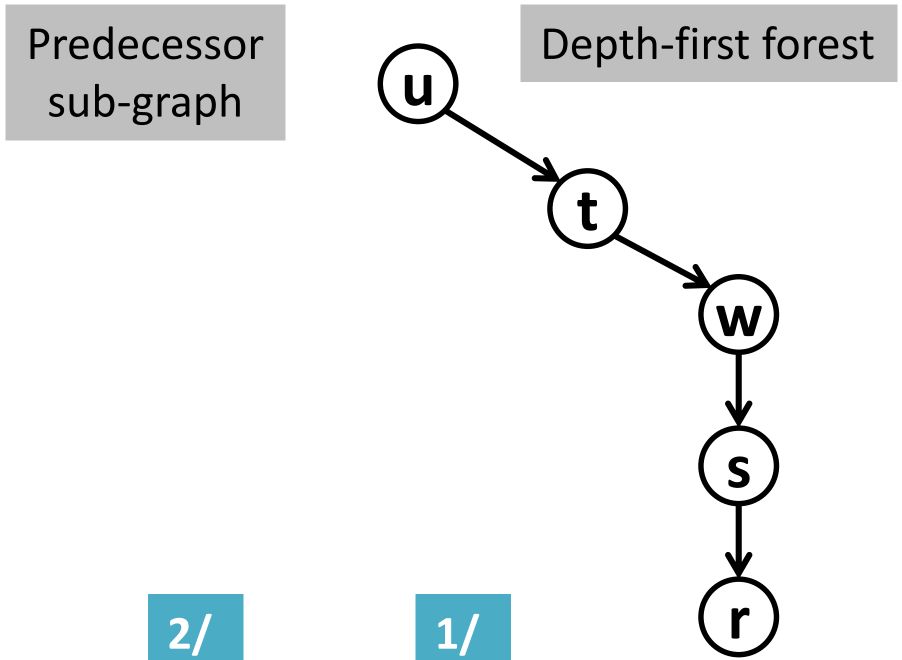
- DFS: u t w s

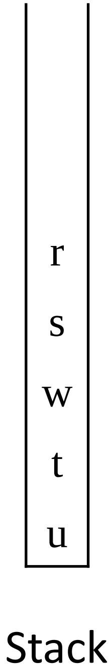


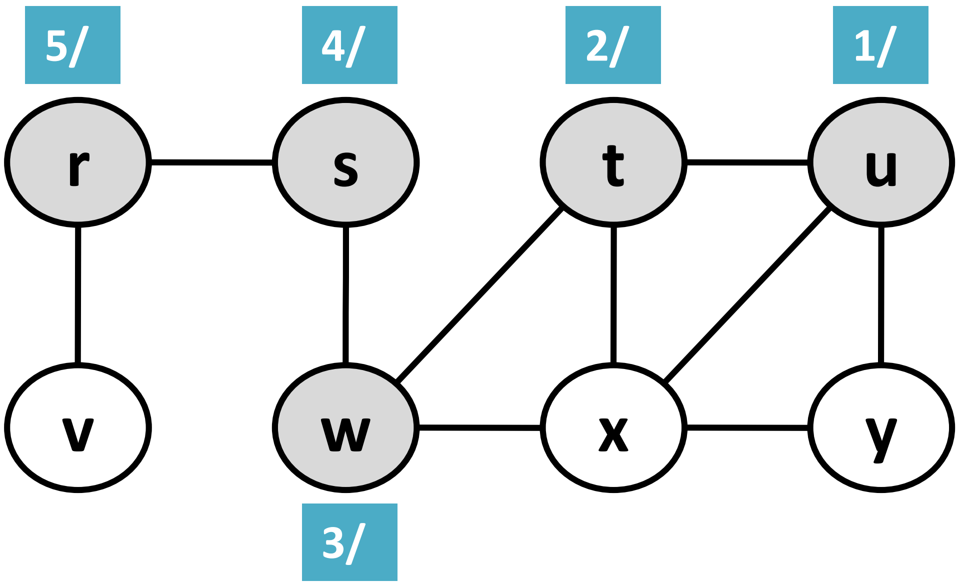


Compute DFS - Undirected

- DFS: u t w s r

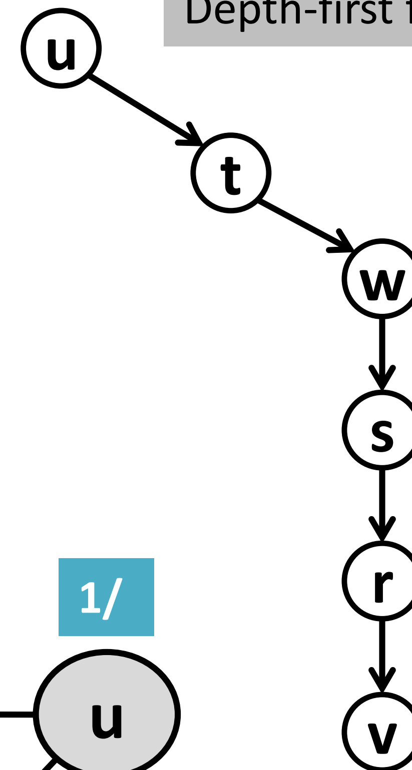






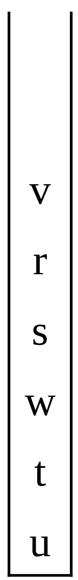
Compute DFS - Undirected

Predecessor sub-graph

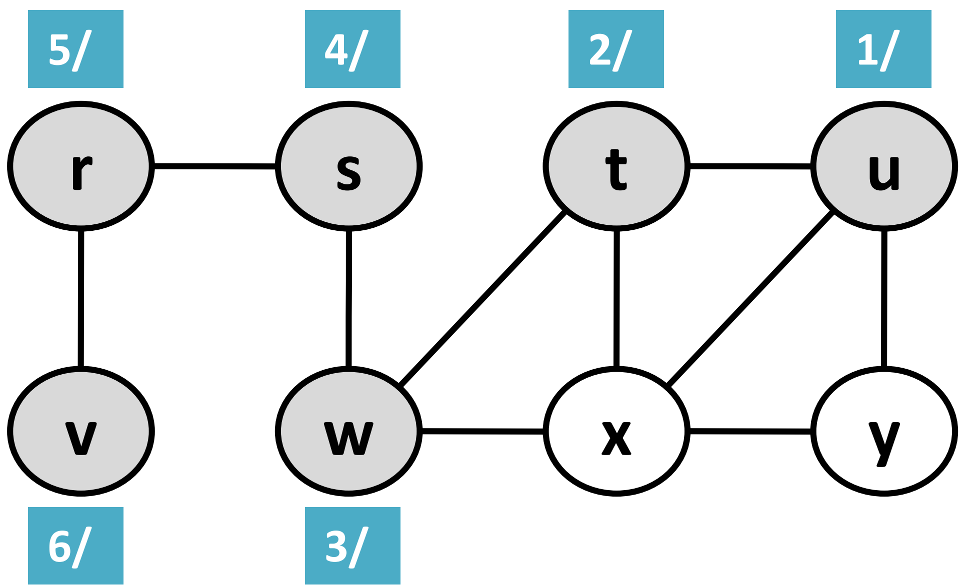


Depth-first forest

- DFS: u t w s r v

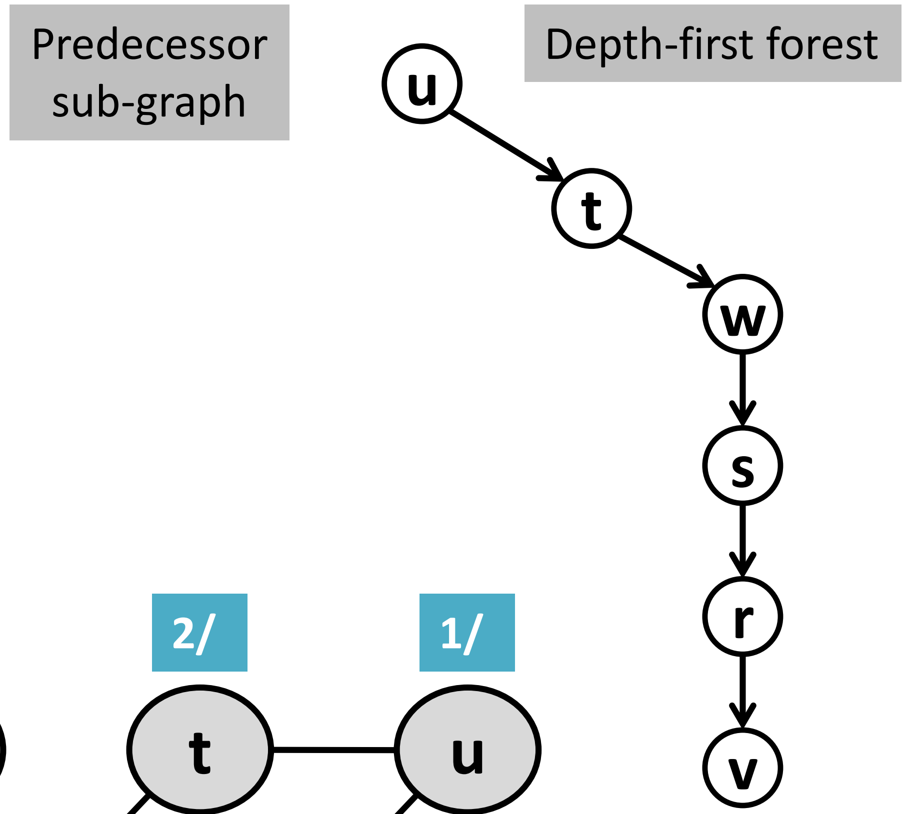


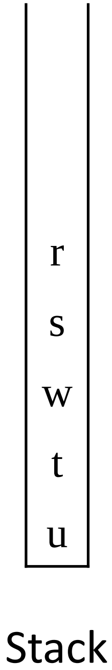
Stack

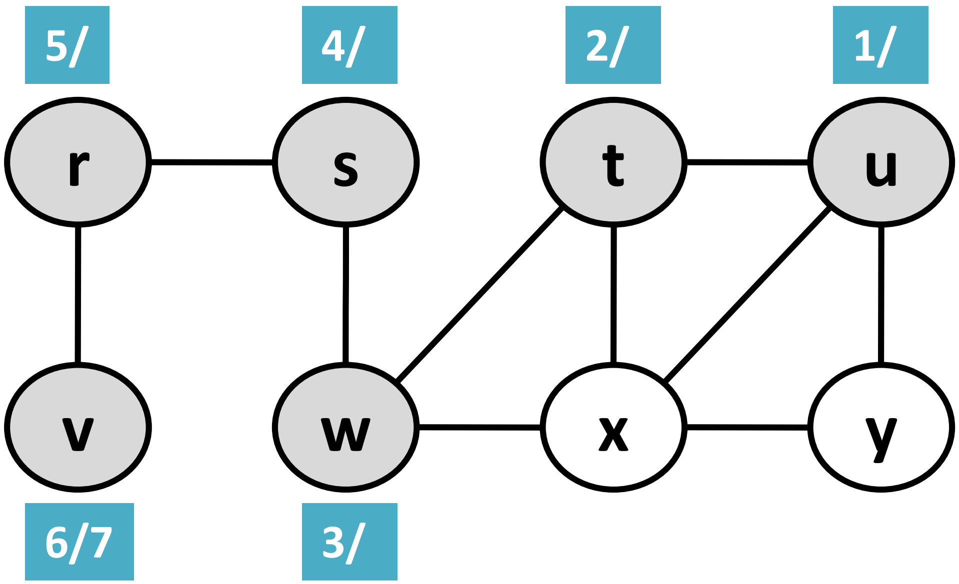


Compute DFS - Undirected

- DFS: u t w s r v



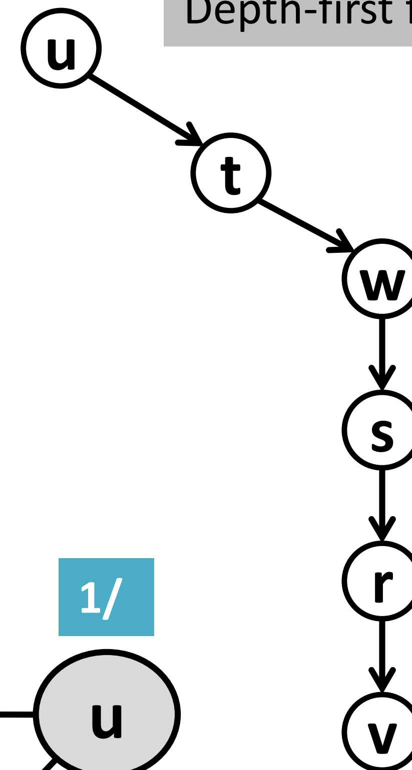




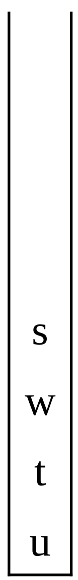
Compute DFS - Undirected

- DFS: u t w s r v

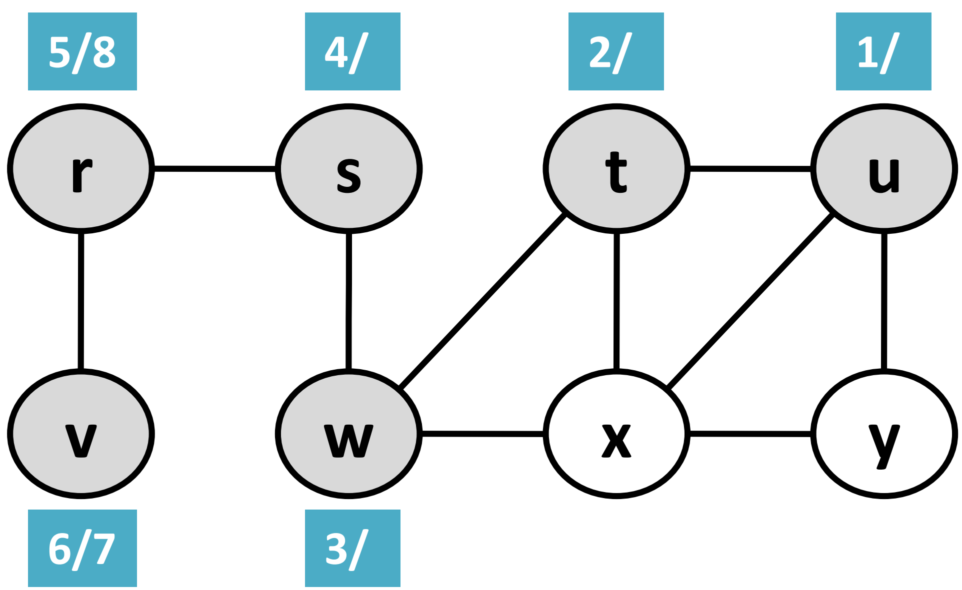
Predecessor sub-graph



Depth-first forest



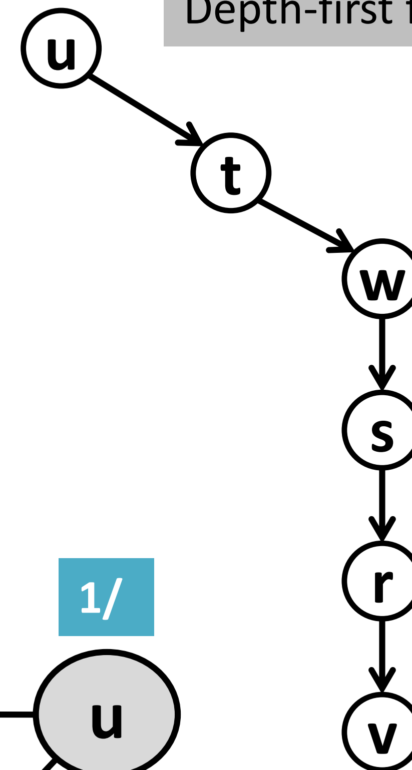
Stack



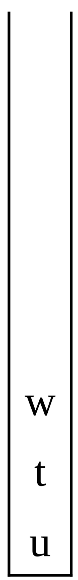
Compute DFS - Undirected

Predecessor sub-graph

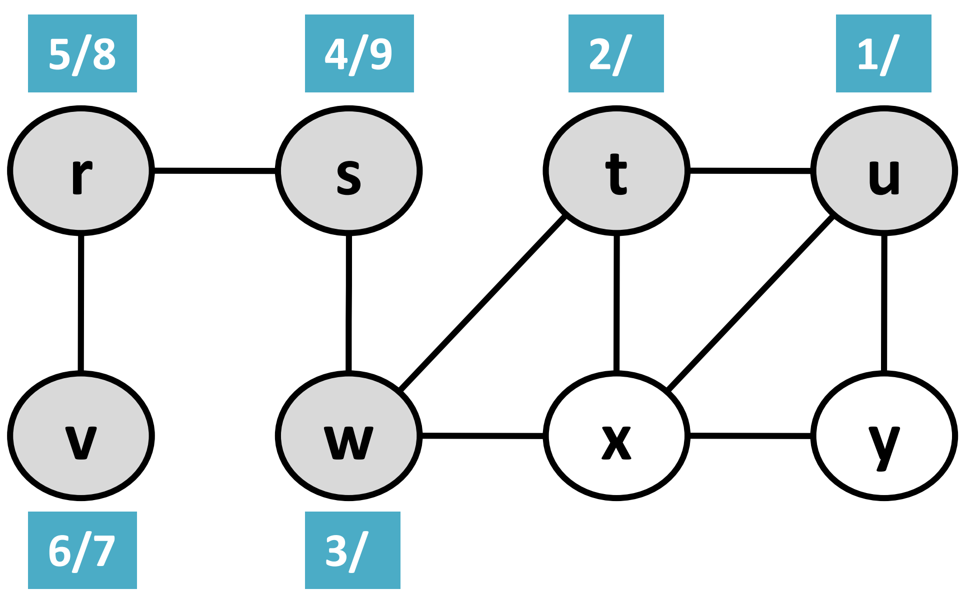
Depth-first forest



- DFS: u t w s r v



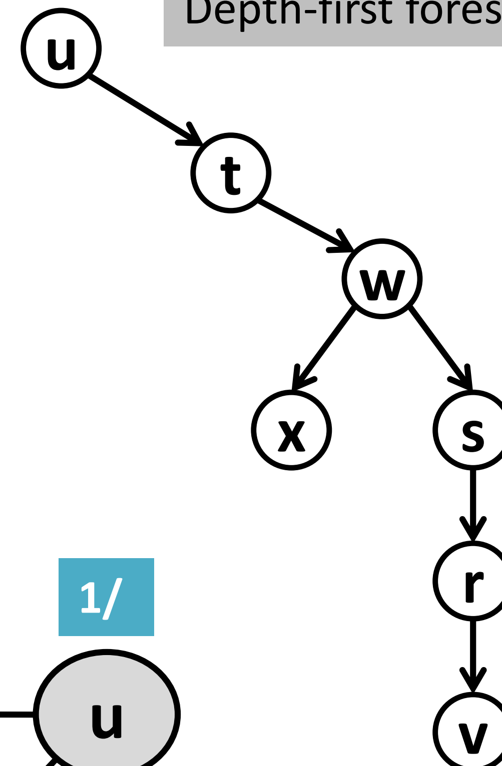
Stack



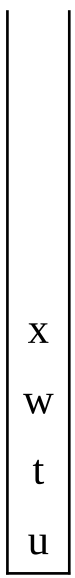
Compute DFS - Undirected

Predecessor sub-graph

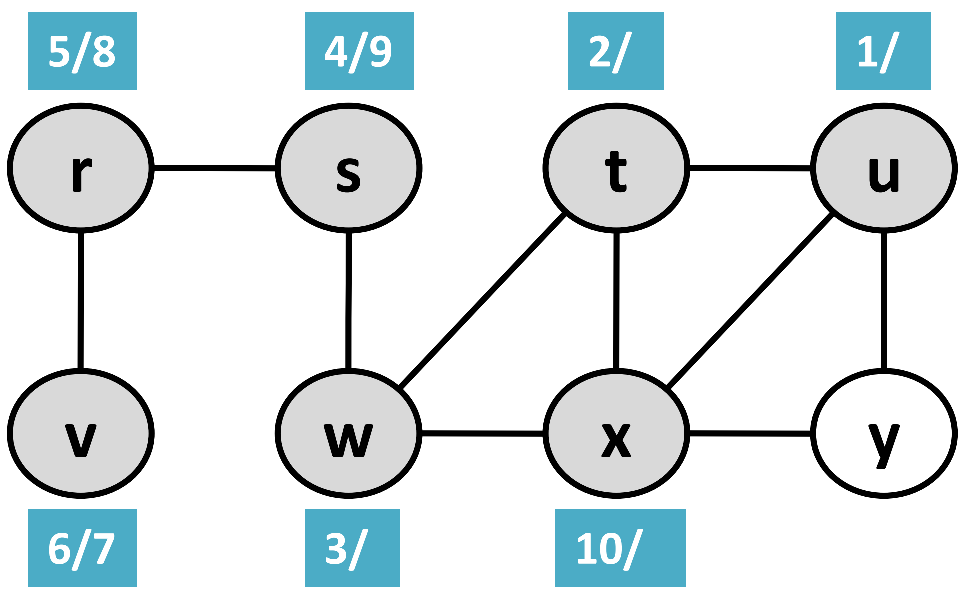
Depth-first forest



- DFS: u t w s r v x

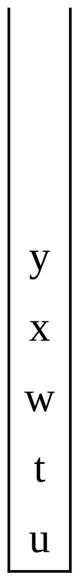


Stack

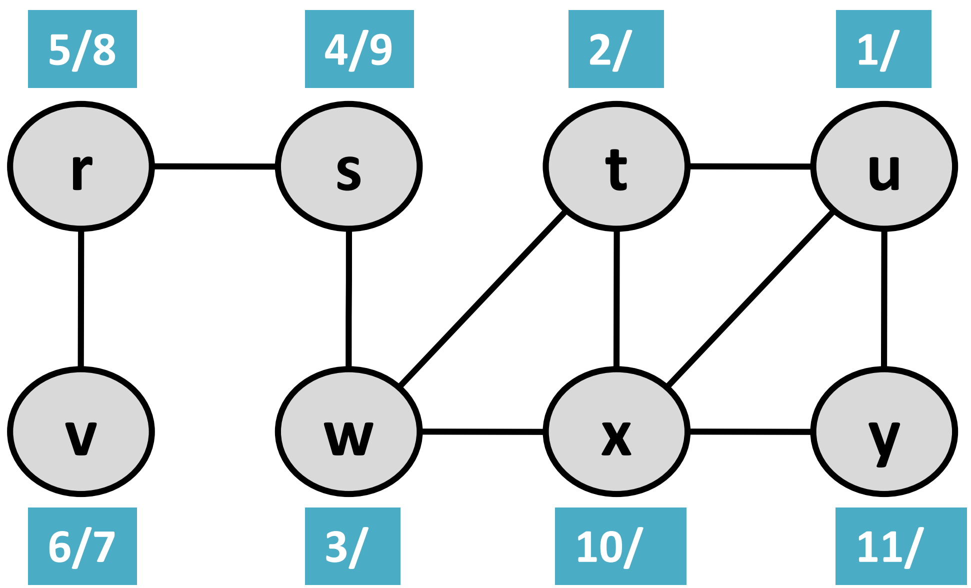


Compute DFS - Undirected

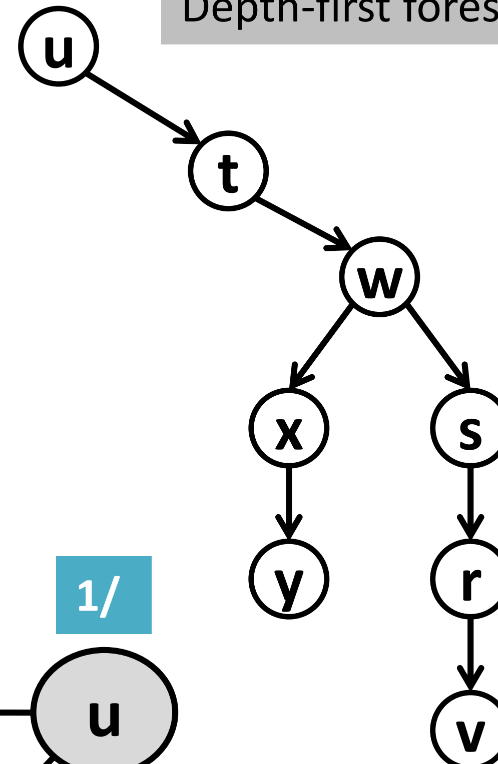
- DFS: u t w s r v x y



Stack



Predecessor sub-graph

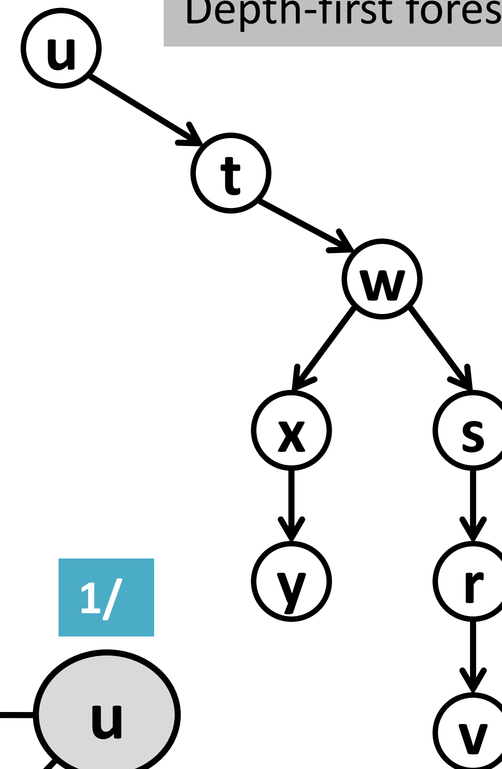


Depth-first forest

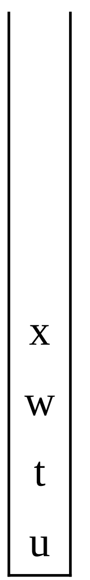
Compute DFS - Undirected

Predecessor sub-graph

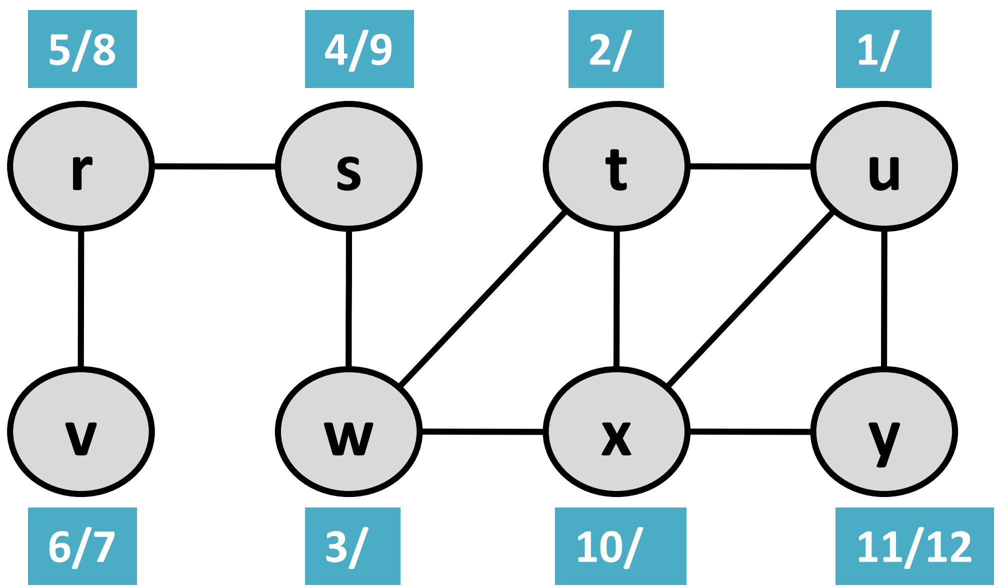
Depth-first forest



- DFS: u t w s r v x y



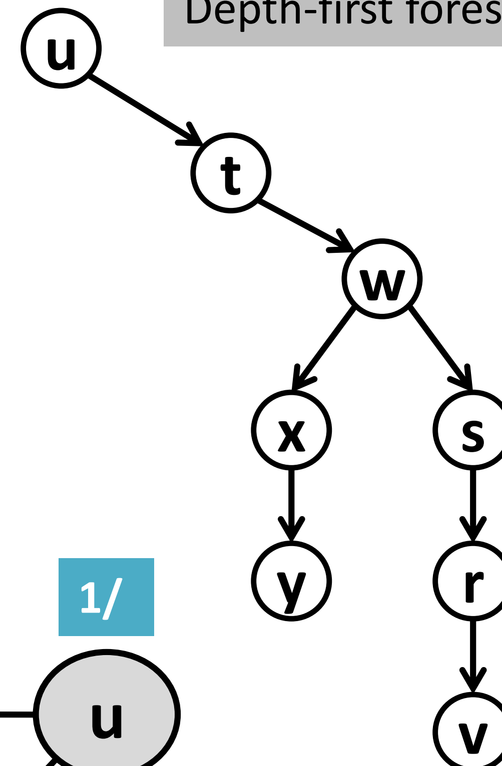
Stack



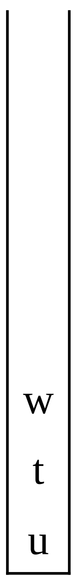
Compute DFS - Undirected

Predecessor sub-graph

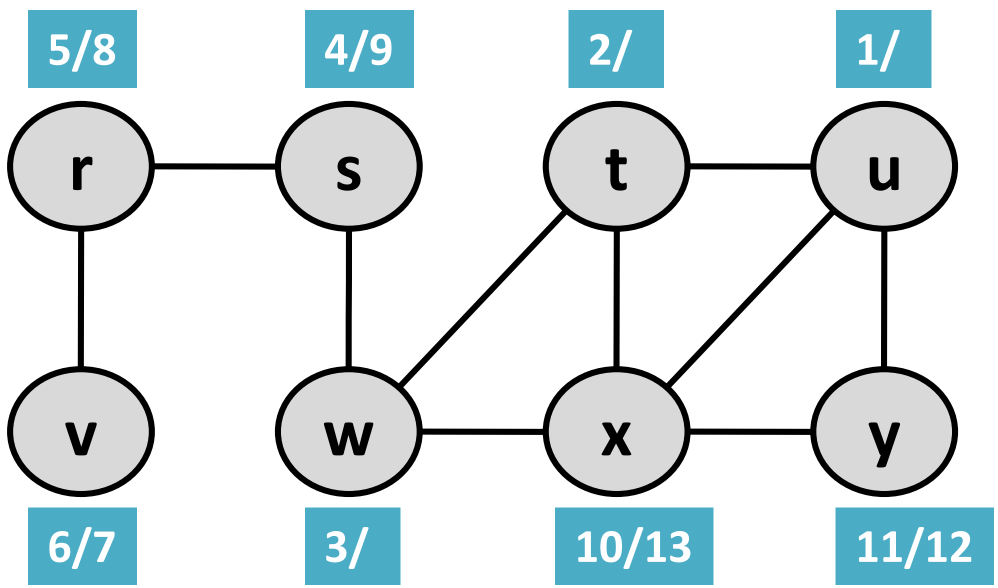
Depth-first forest



- DFS: u t w s r v x y



Stack

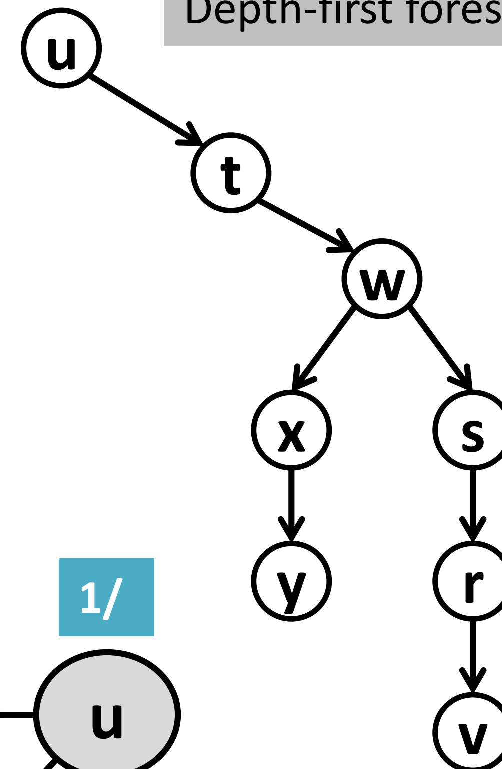


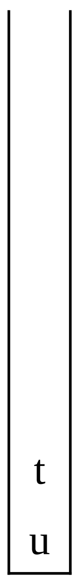
Compute DFS - Undirected

- DFS: u t w s r v x y

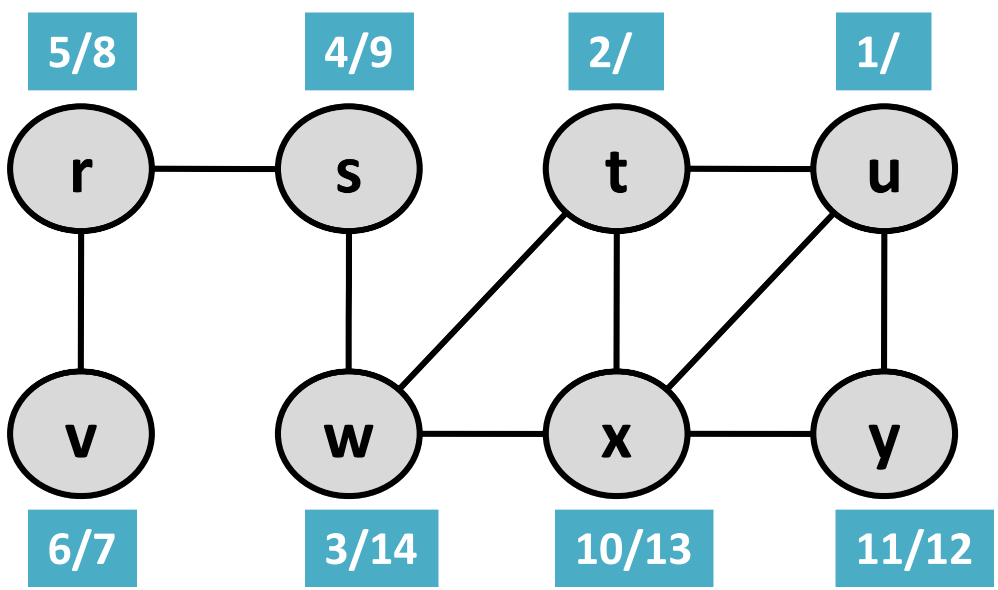
Predecessor sub-graph

Depth-first forest



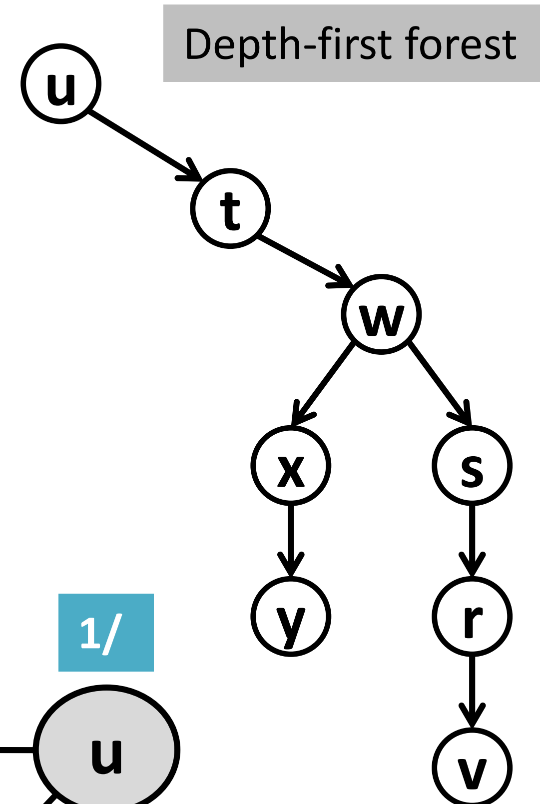


Stack

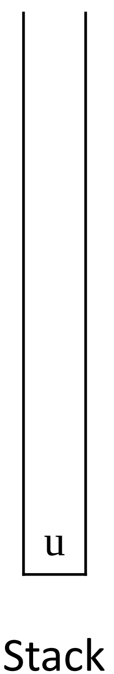


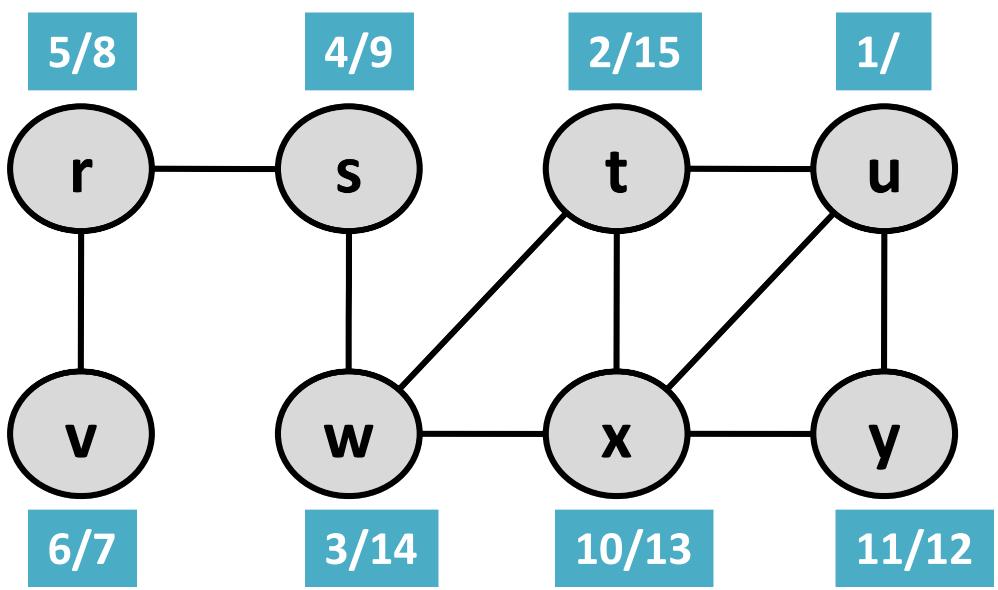
Compute DFS - Undirected

Predecessor sub-graph



- DFS: u t w s r v x y

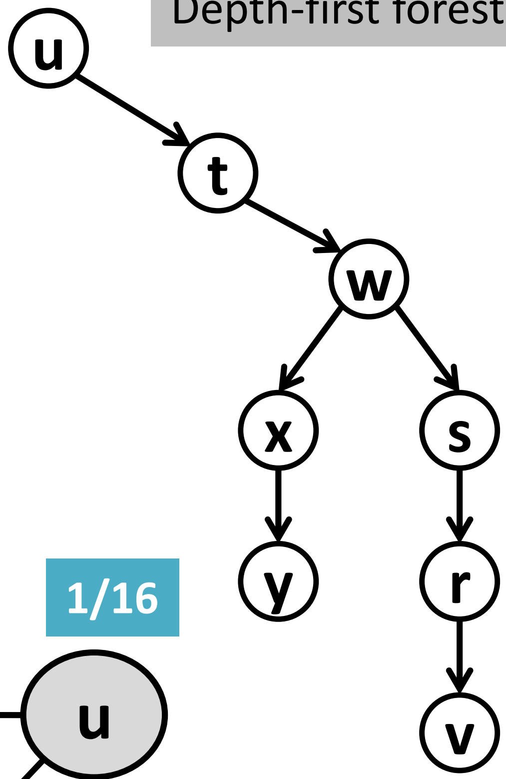




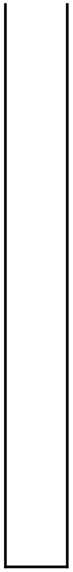
Compute DFS - Undirected

- DFS: u t w s r v x y

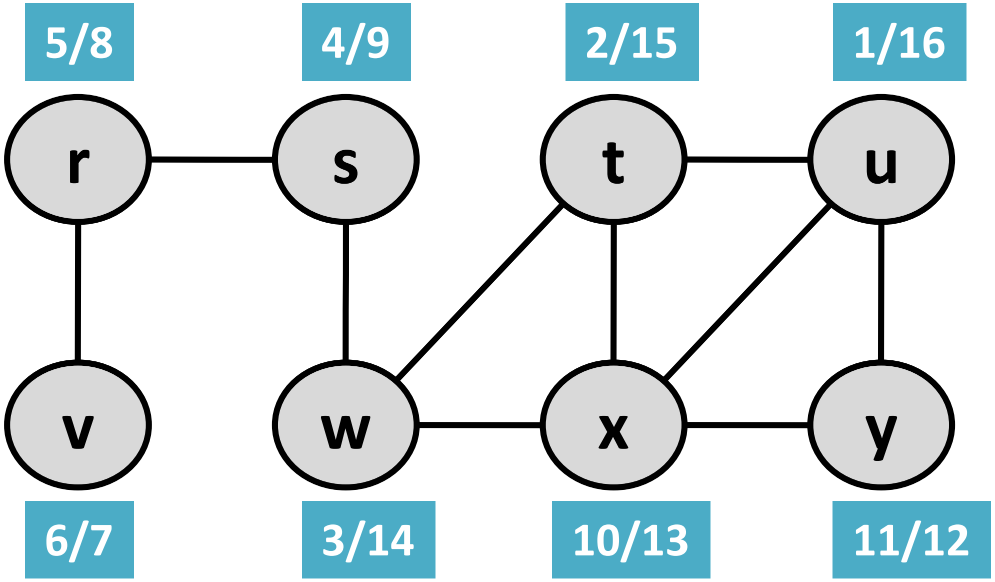
Predecessor sub-graph



Depth-first forest



Stack



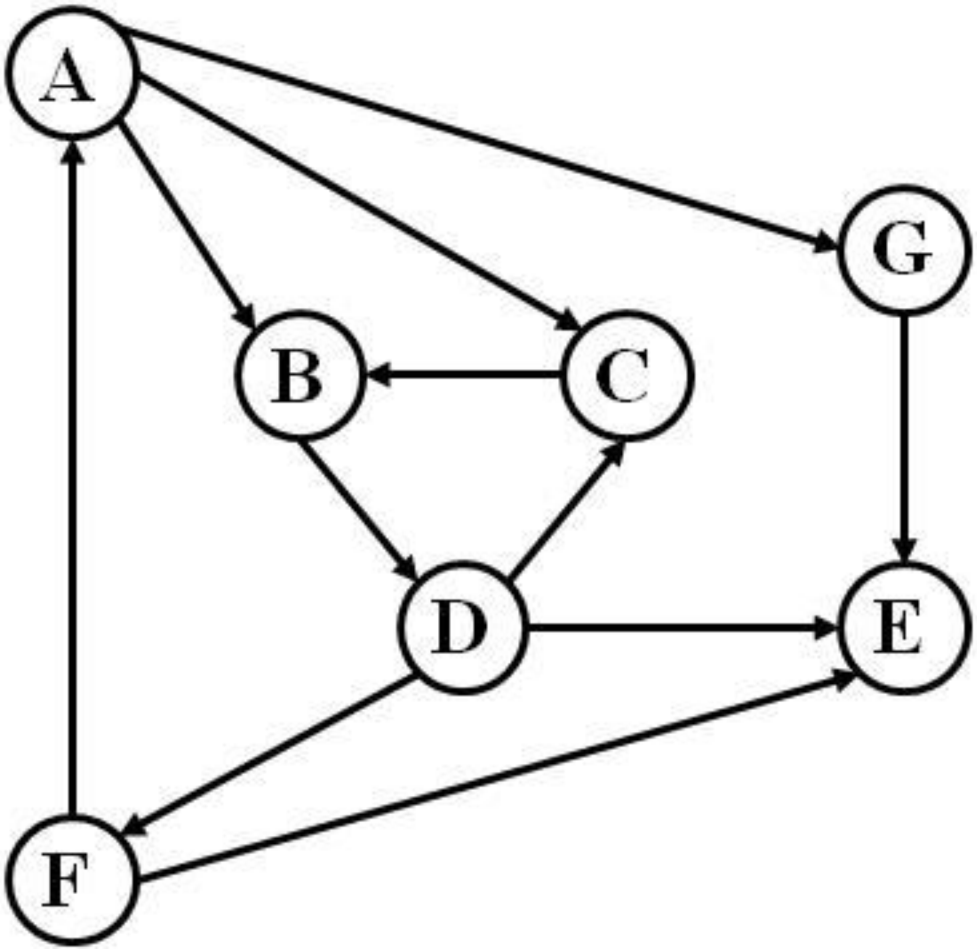
Compute DFS - Directed

Predecessor sub-graph

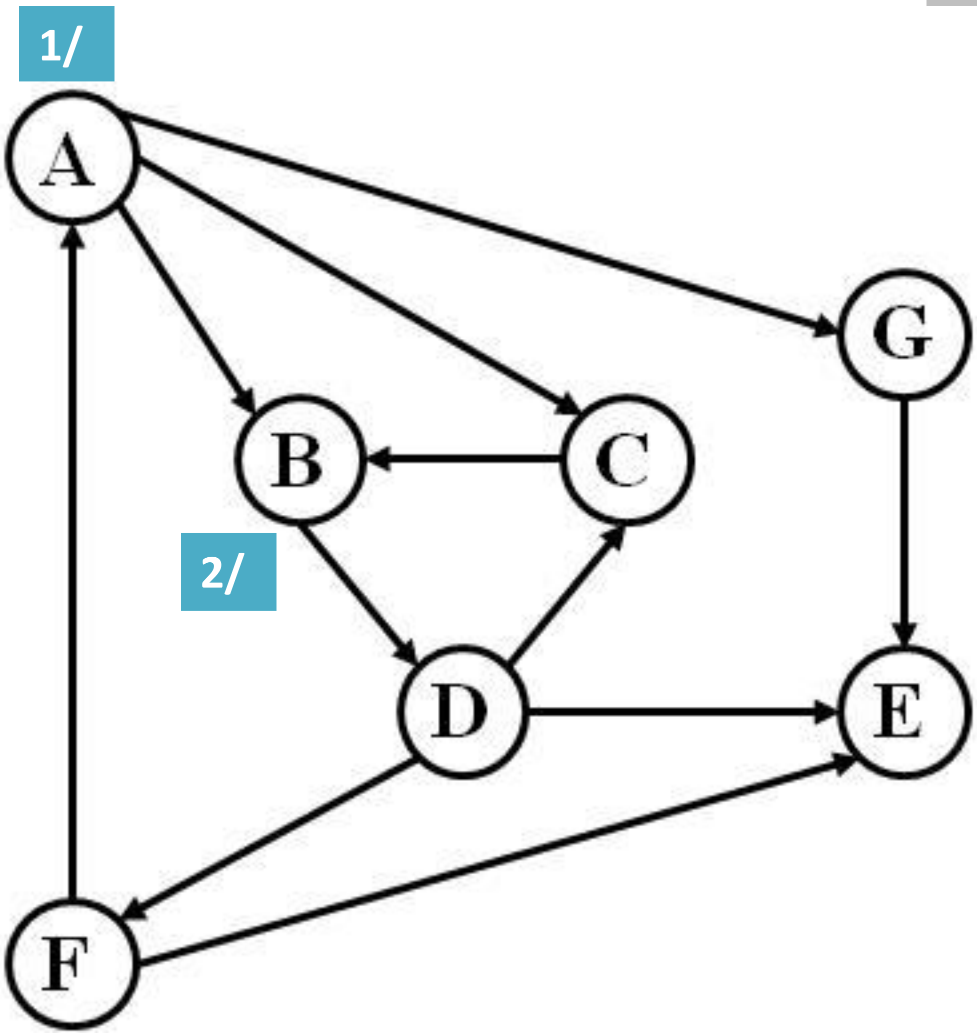
A

Depth-first forest

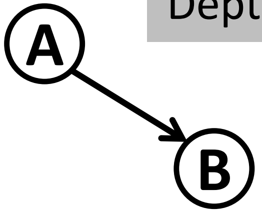
1/



Compute DFS - Directed

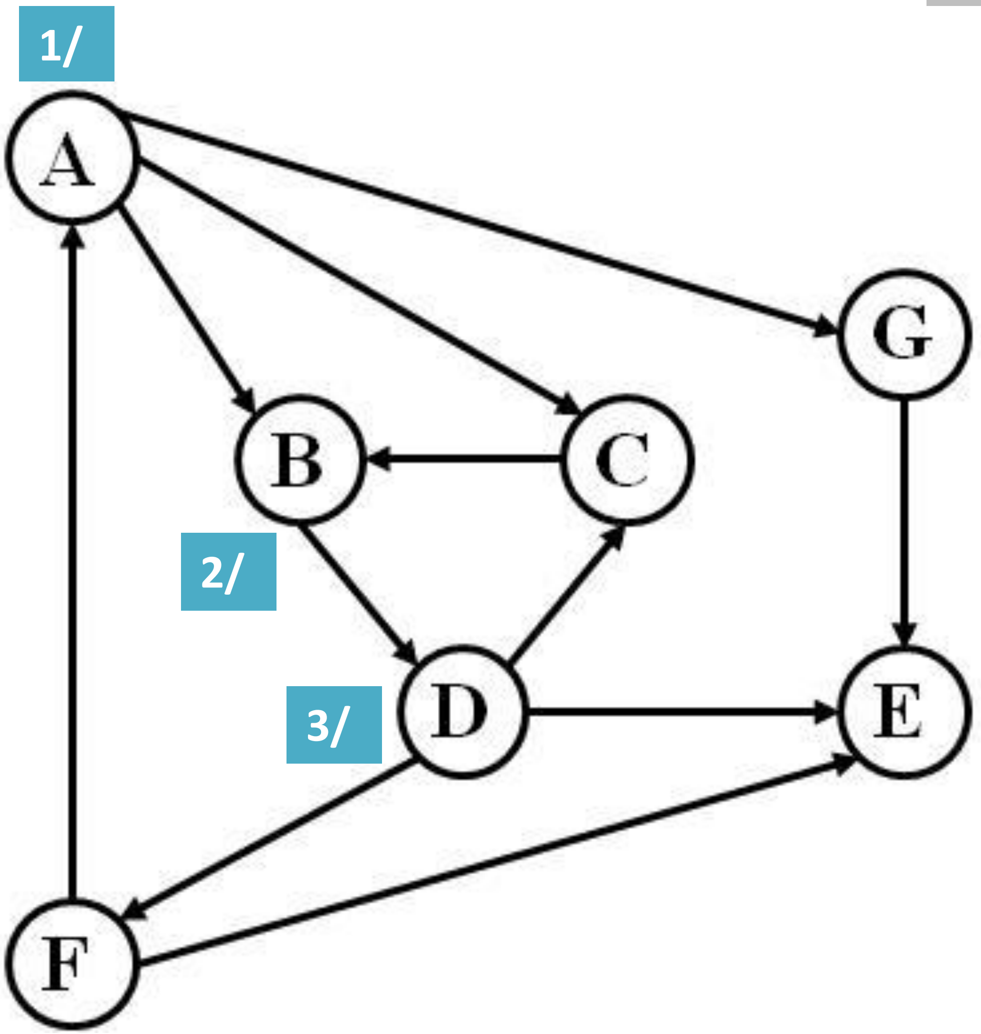


Predecessor sub-graph

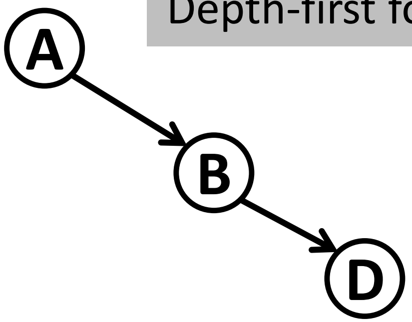


Depth-first forest

Compute DFS - Directed

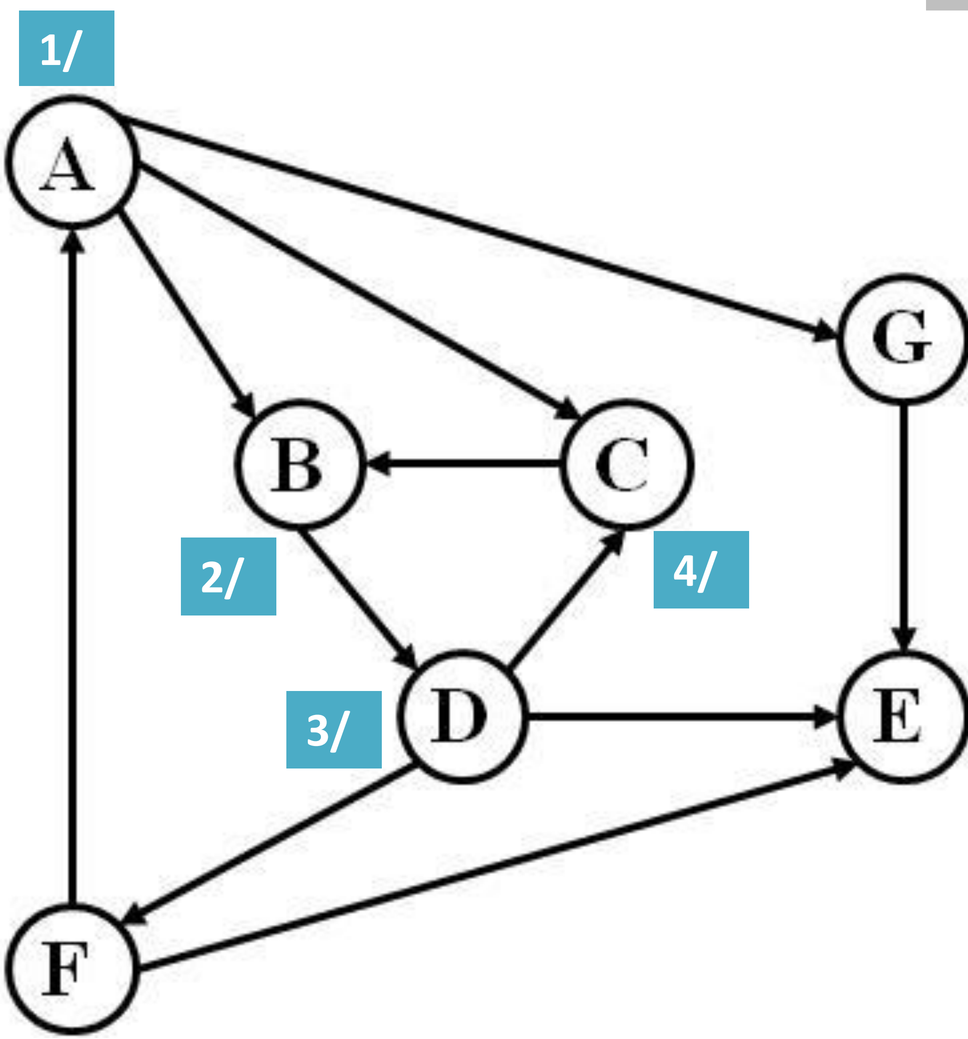


Predecessor sub-graph

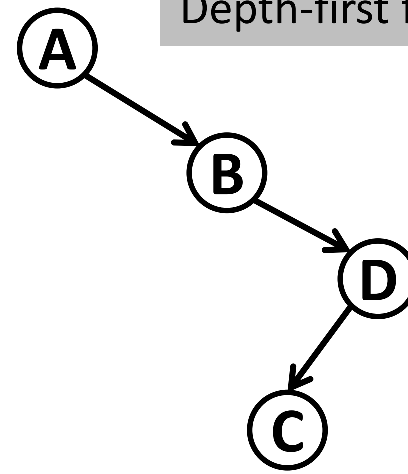


Depth-first forest

Compute DFS - Directed



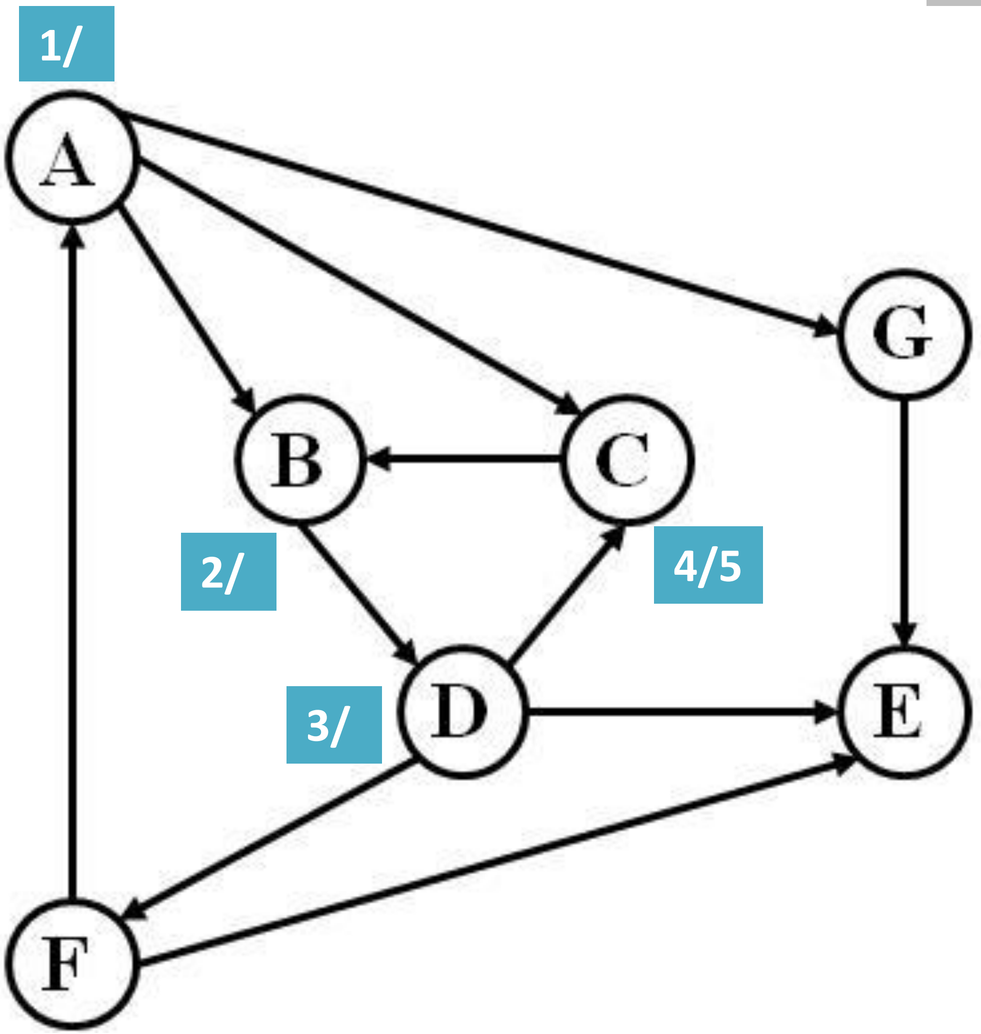
Predecessor sub-graph



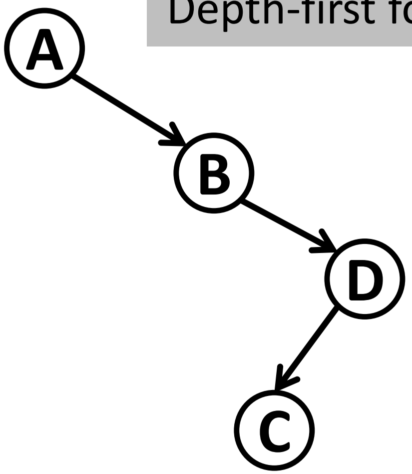
Depth-first forest

Compute DFS - Directed

Predecessor sub-graph

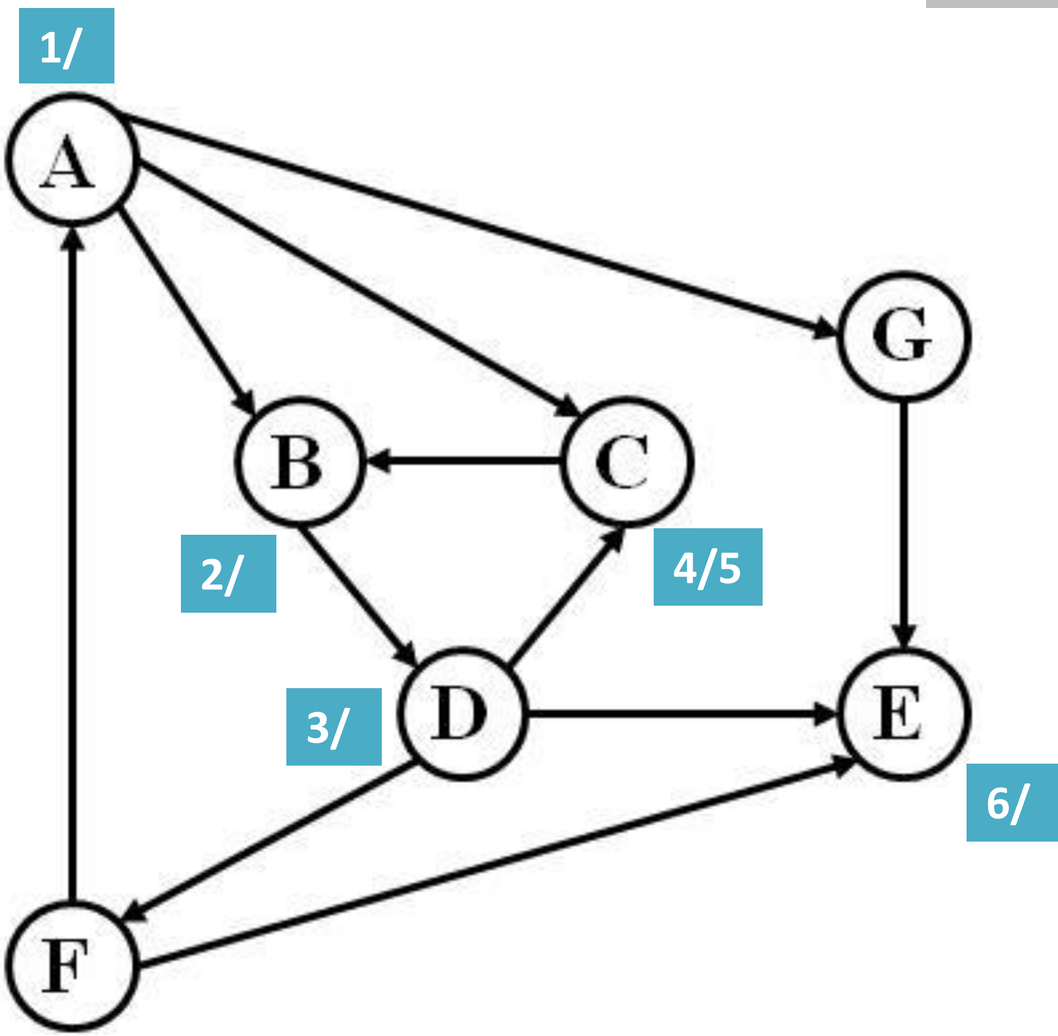


Depth-first forest

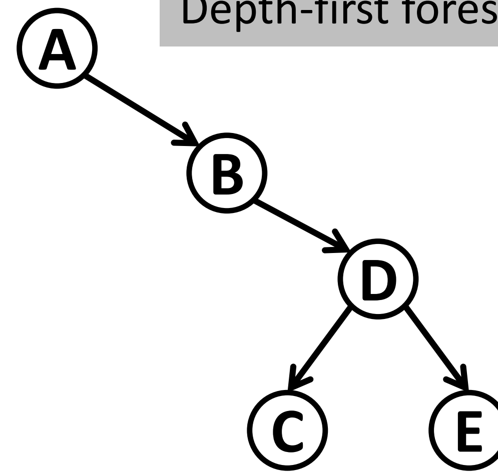


Compute DFS - Directed

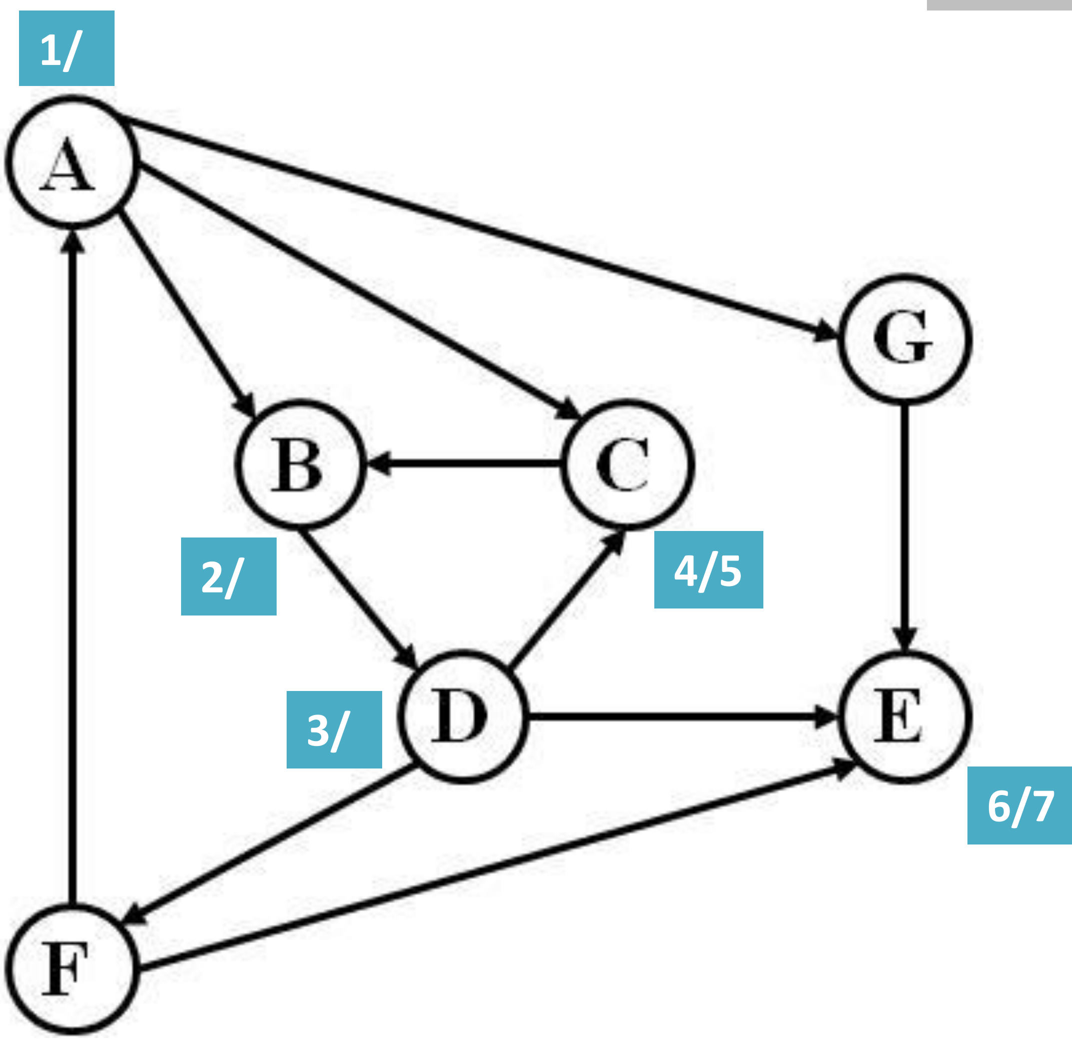
Predecessor sub-graph



Depth-first forest

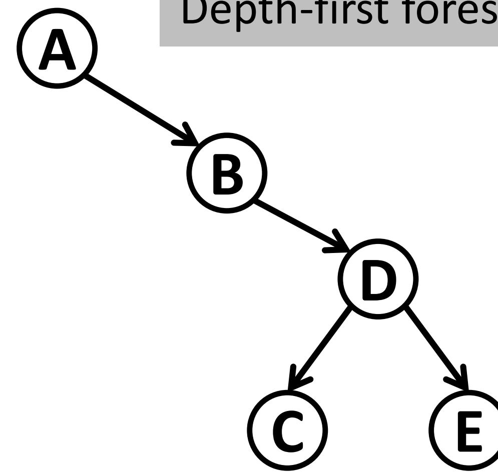


Compute DFS - Directed

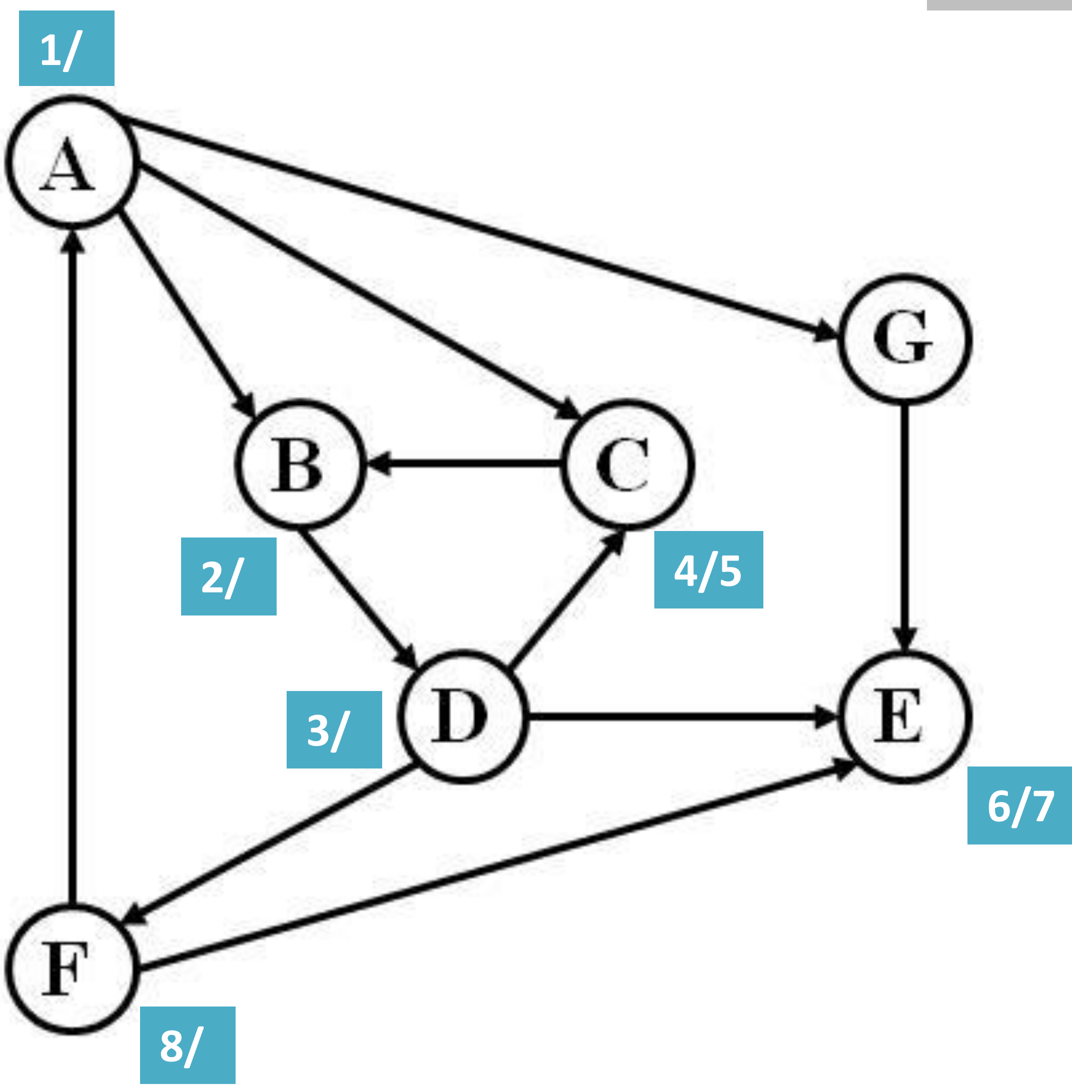


Predecessor sub-graph

Depth-first forest

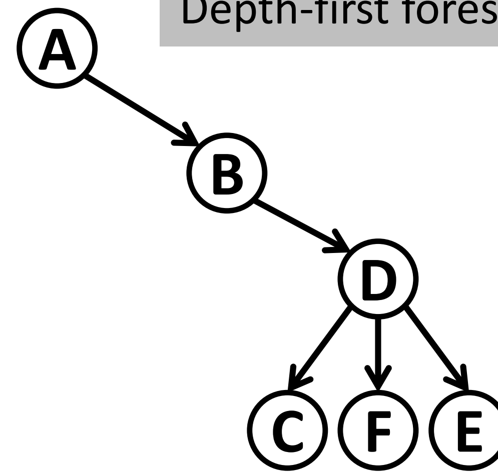


Compute DFS - Directed

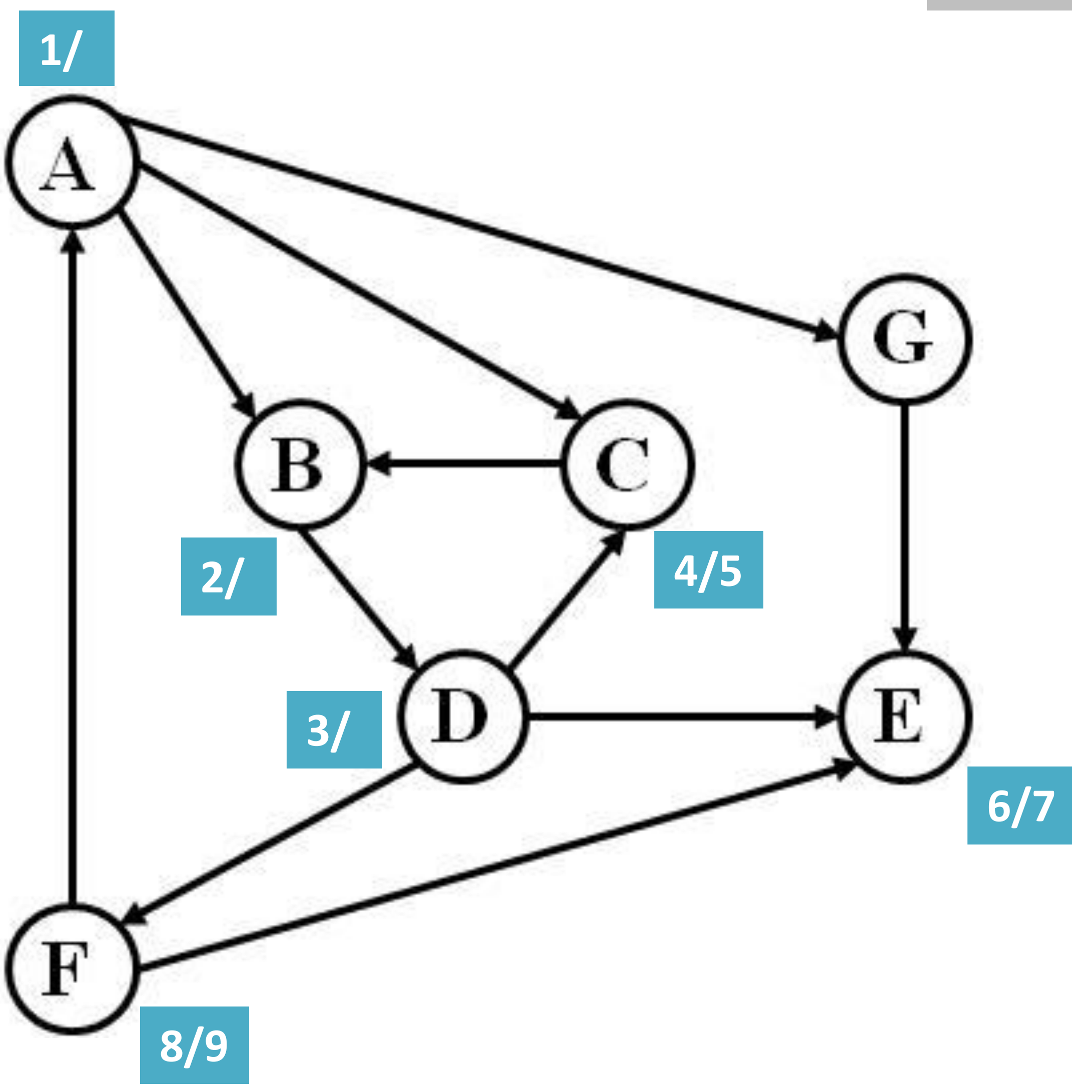


Predecessor sub-graph

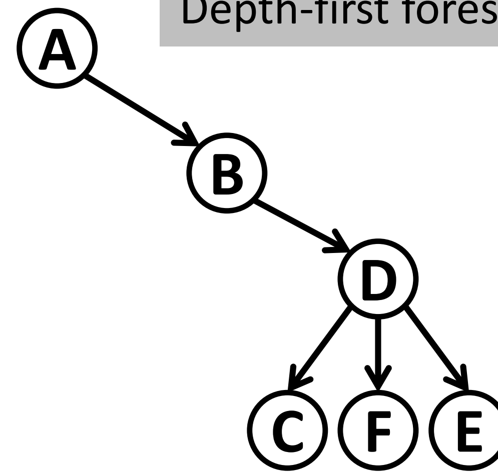
Depth-first forest



Compute DFS - Directed



Predecessor sub-graph

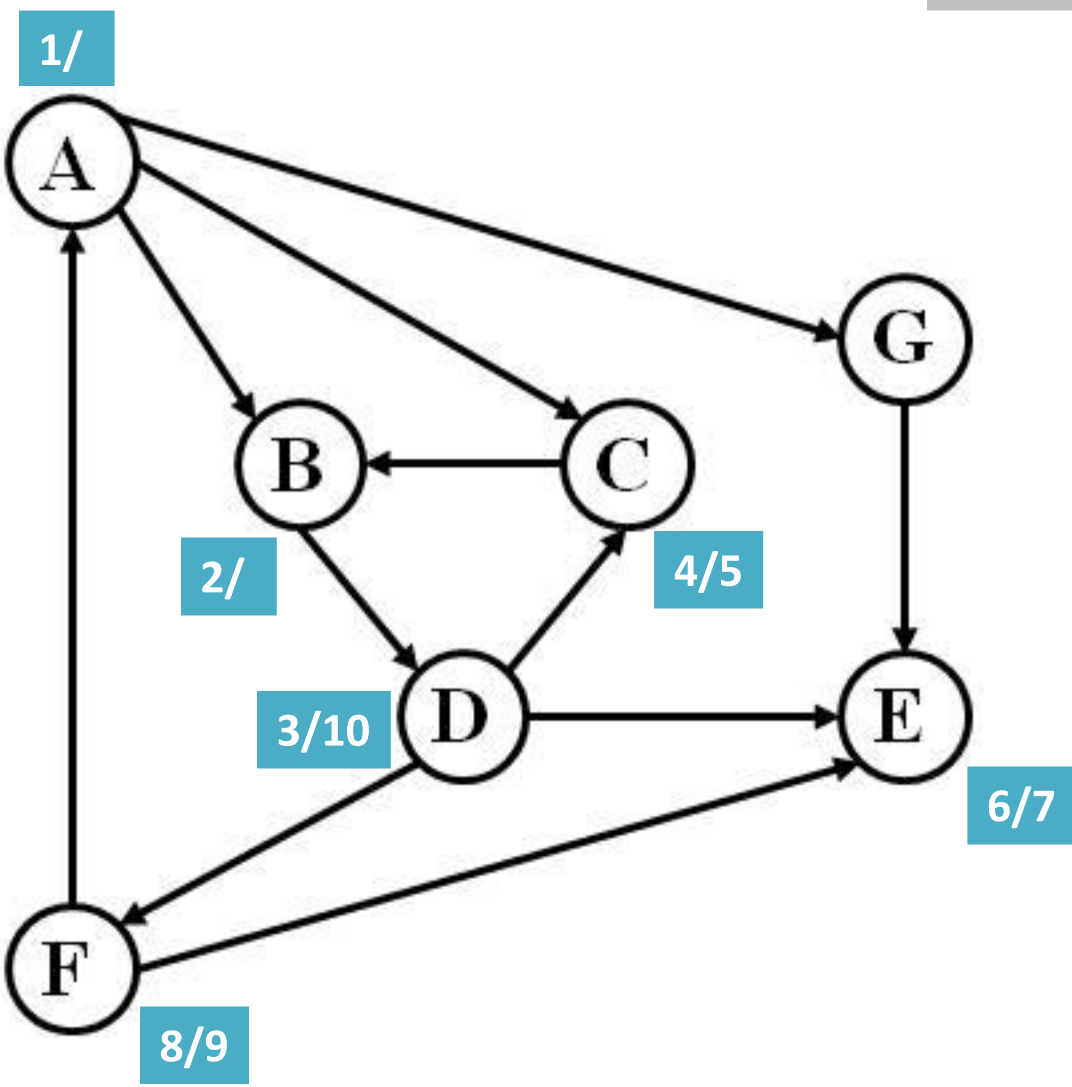


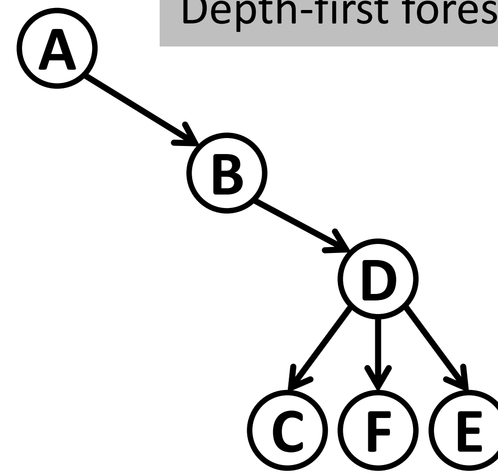
Depth-first forest

Compute DFS - Directed

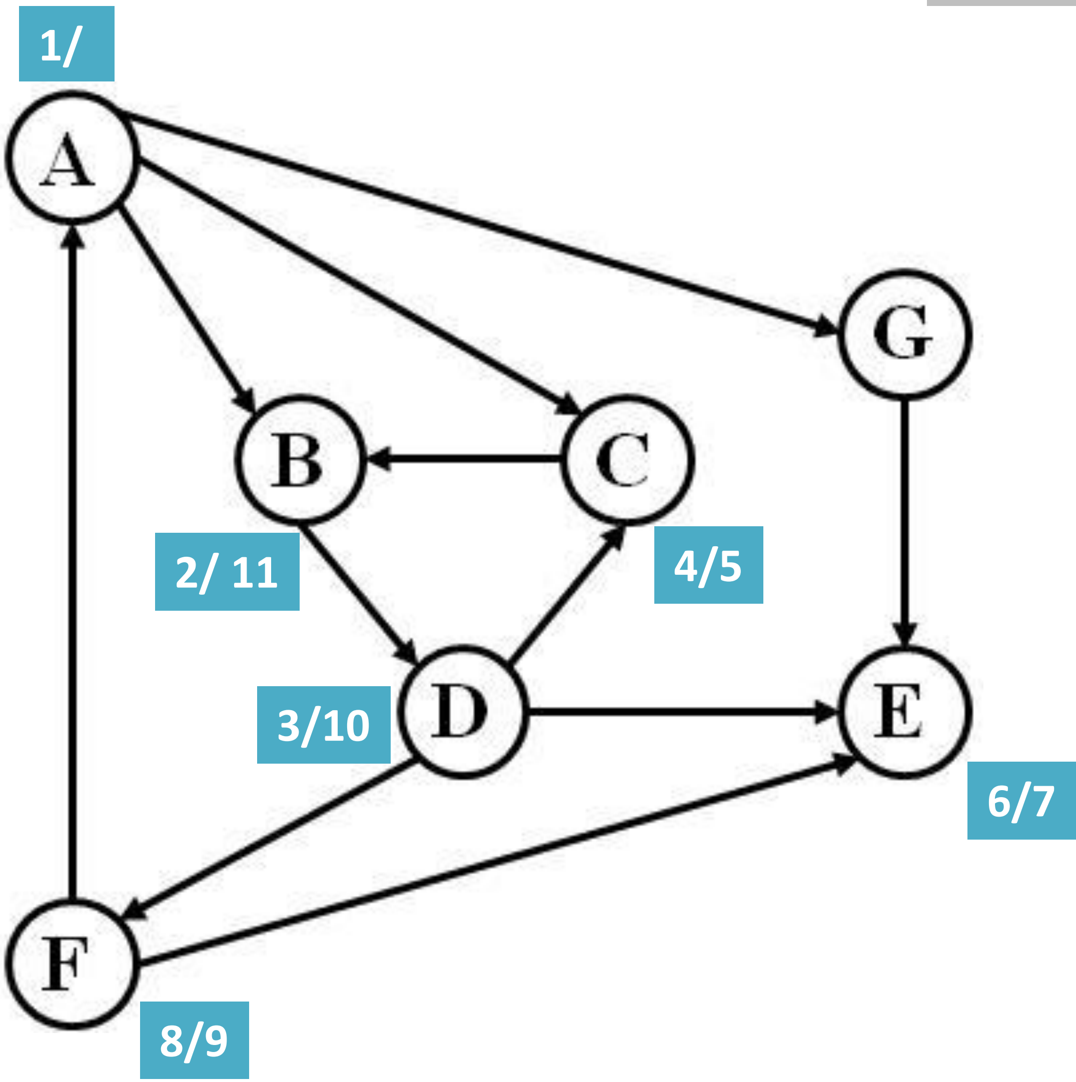
Predecessor sub-graph

Depth-first forest



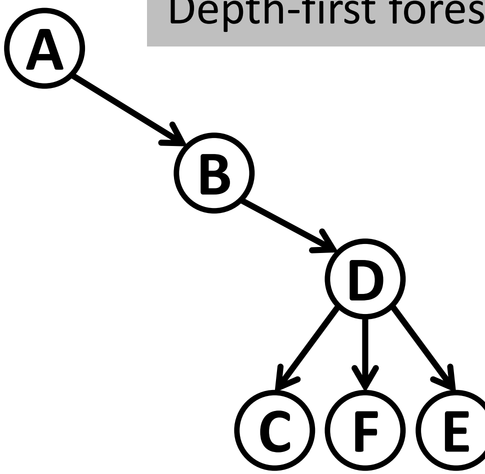


Compute DFS - Directed

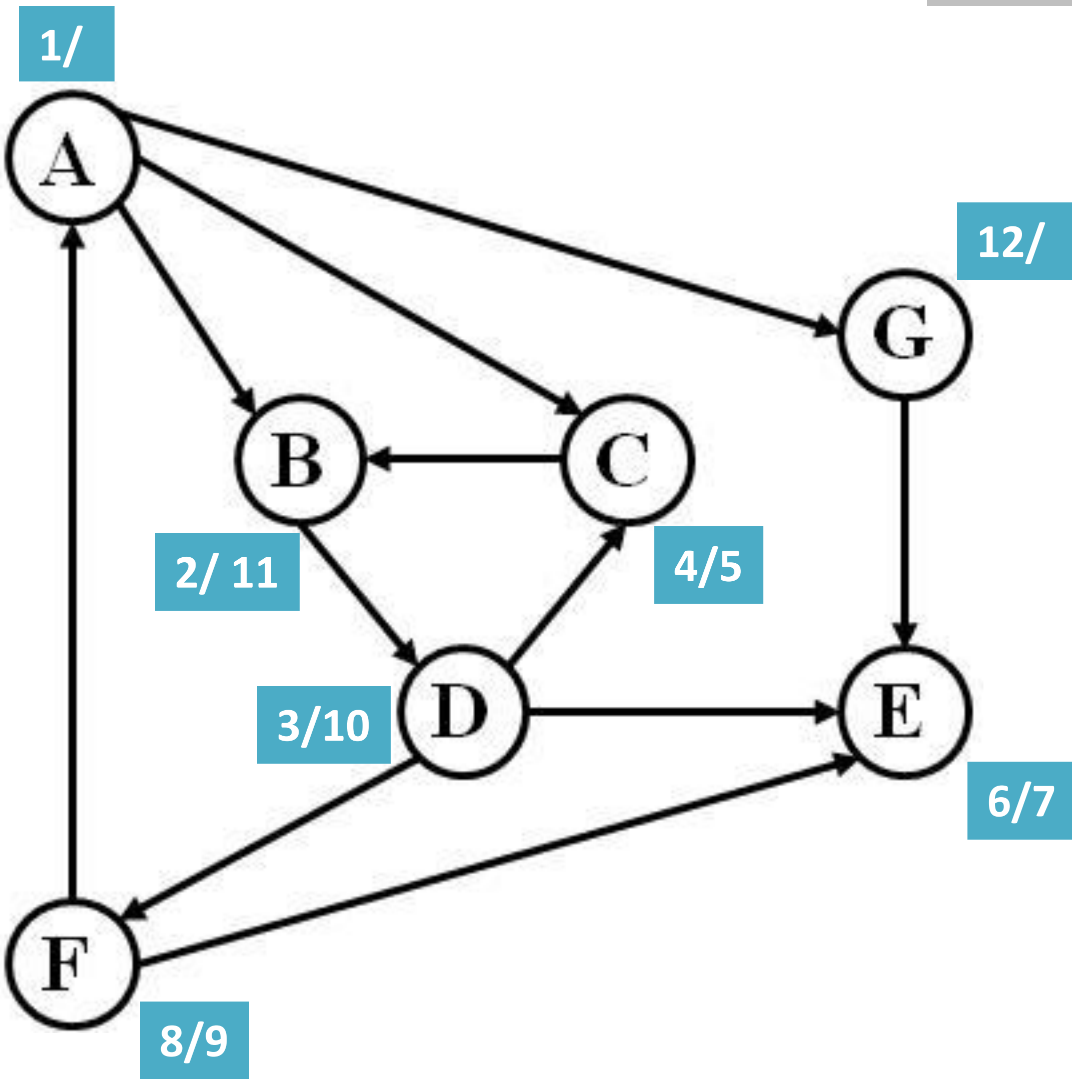


Predecessor sub-graph

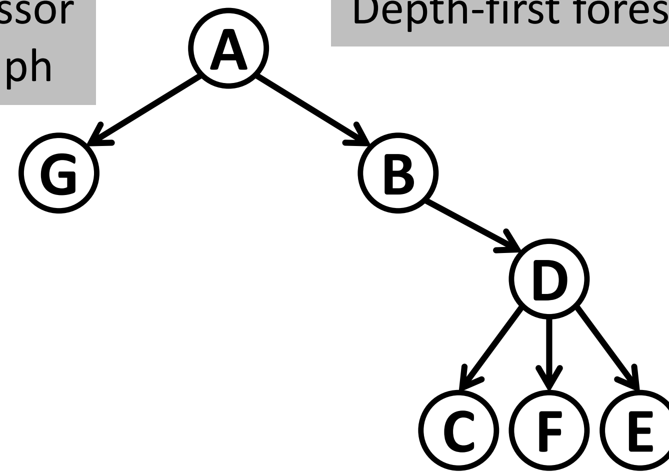
Depth-first forest



Compute DFS - Directed

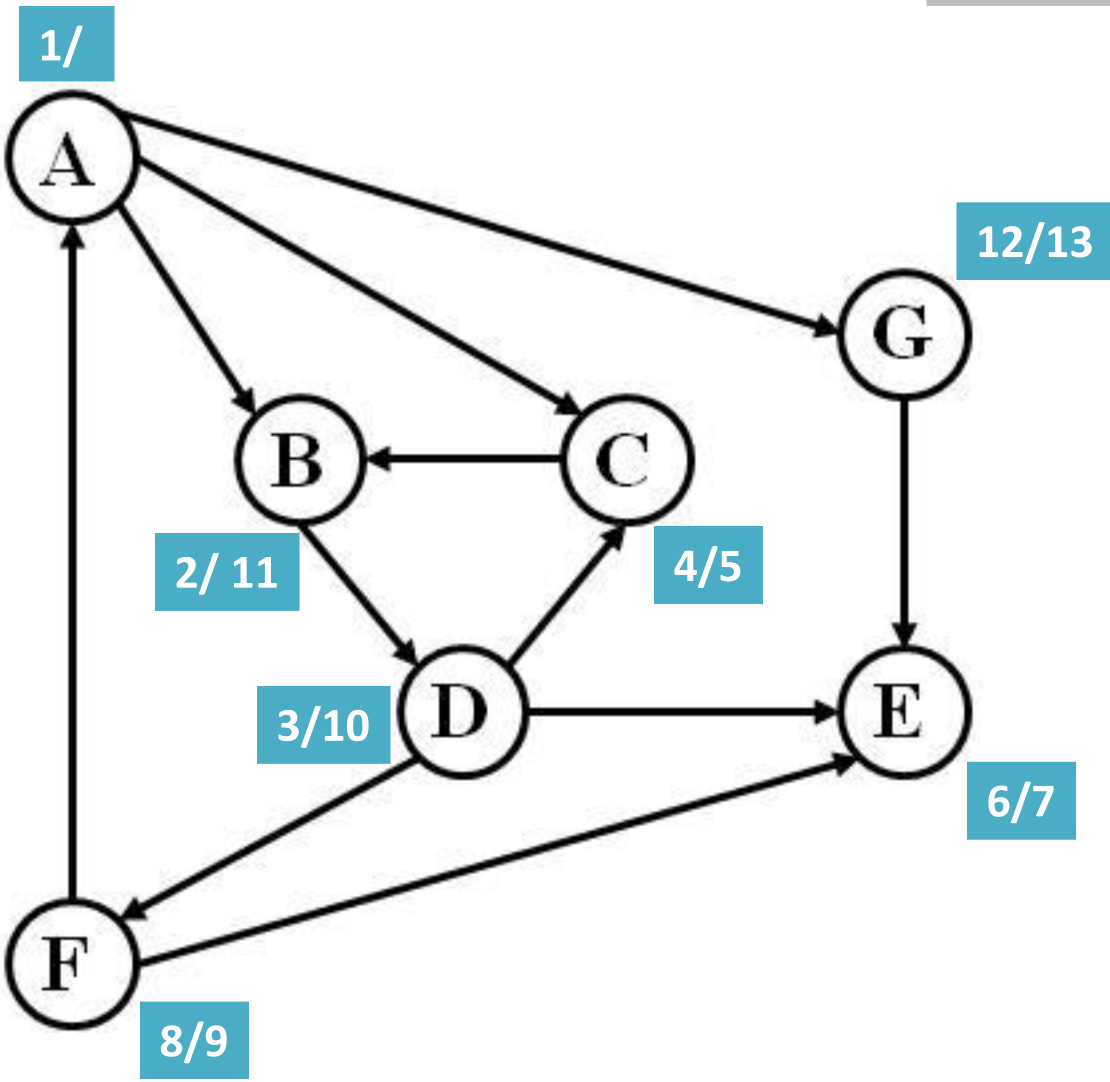


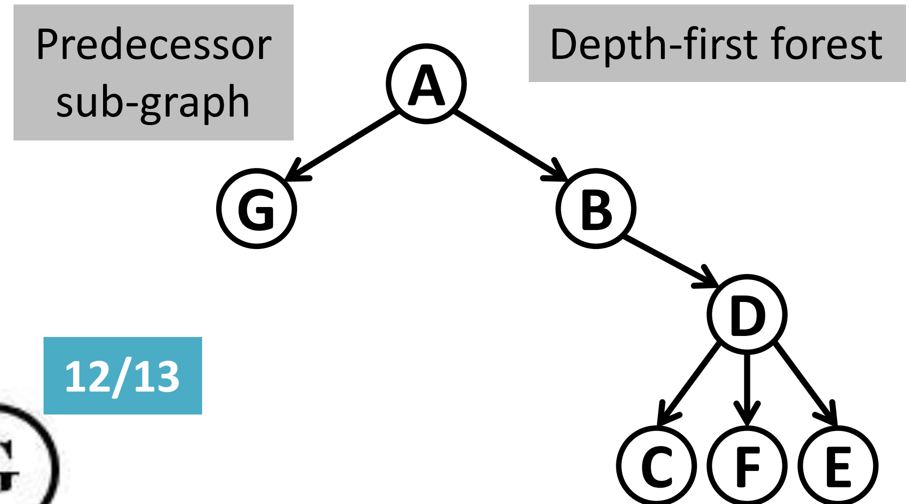
Predecessor sub-graph



Depth-first forest

Compute DFS - Directed

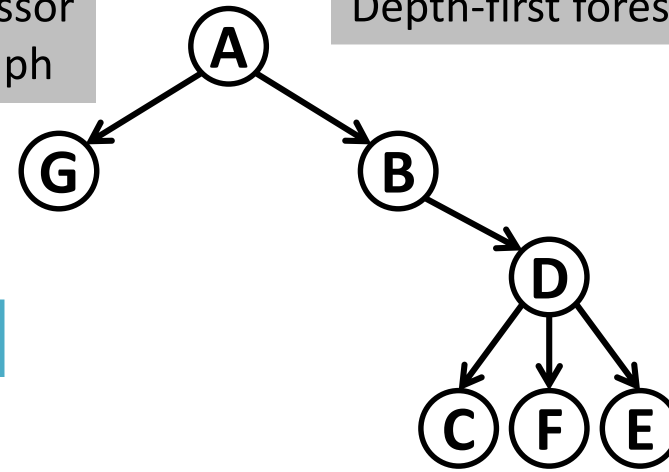


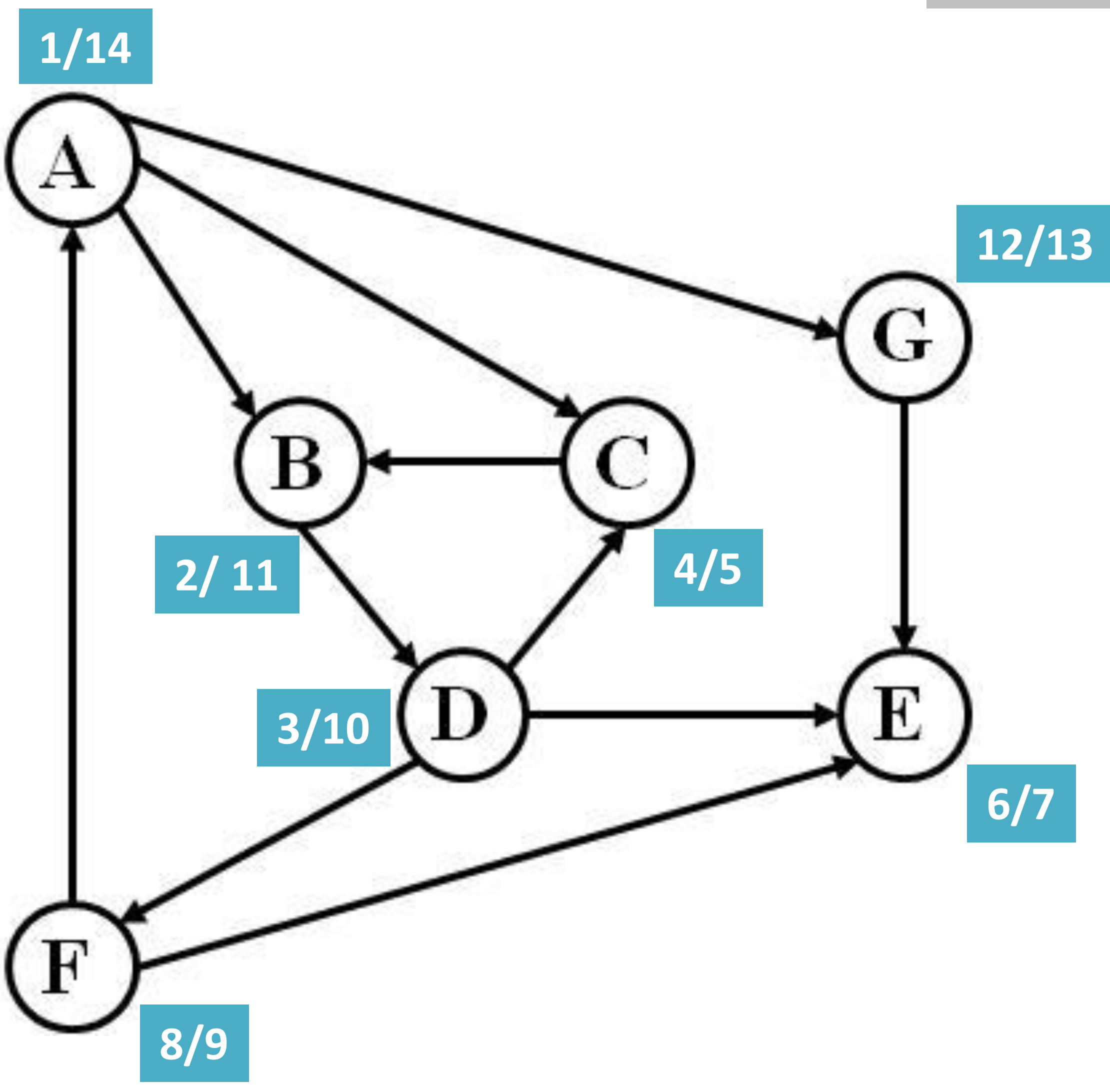


Compute DFS - Directed

Predecessor sub-graph

Depth-first forest





DFS: A B D C E F G

Procedure DFS

```
DFS(G)
for each vertex u ∈ G.V
        u.color = WHITE
        u.π = NIL
time = 0
for each vertex u ∈ G.V
        if u.color == WHITE
                DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
        if v.color == WHITE
                v.π = u
                DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

Execution example

- Let's start with vertex u.

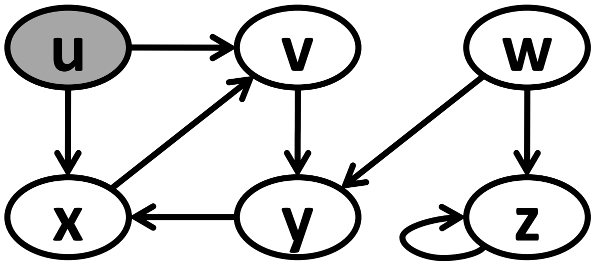
Vertex	Color	Timestamp		Predecessor (π)
		d	f	
u	White			NIL
v	White			NIL
w	White			NIL
x	White			NIL
y	White			NIL
z	White			NIL

time = 0

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u → v → x
v → y
w → y → z
x → v
y → x
z → z

Execution example



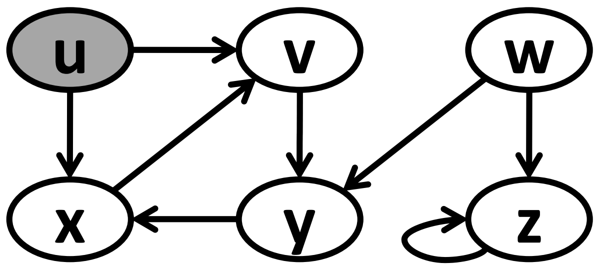
Vertex	Color	Timestamp d	Timestamp f	Predecessor (π)
u	**Gray**	**1**		**NIL**
v	White			NIL
w	White			NIL
x	White			NIL
y	White			NIL
z	White			NIL

```
DFS-VISIT(G, u)
 time = time + 1
 u.d = time
 u.color = GRAY
 for each v ∈ G.Adj[u]
     if v.color == WHITE
         v.π = u
         DFS-VISIT(G, v)
 u.color = BLACK
 time = time + 1
 u.f = time
```

u	→	v	→	x
v	→	y		
w	→	y	→	z
x	→	v		
y	→	x		
z	→	z		

u = u   time = 1

Execution example



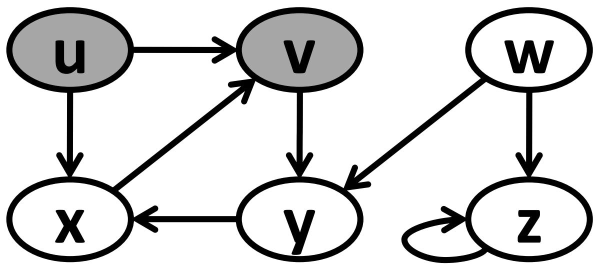
Vertex	Color	Timestamp d	Timestamp f	Predecessor (π)
u	**Gray**	**1**		**NIL**
v	**White**			**u**
w	White			NIL
x	White			NIL
y	White			NIL
z	White			NIL

u = u time = 1

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u → v → x
v → y
w → y → z
x → v
y → x
z → z

Execution example



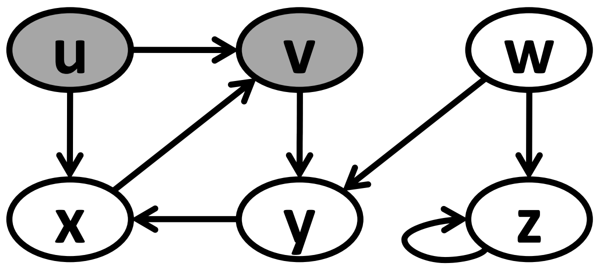
Vertex	Color	Timestamp		Predecessor (π)
		d	f	
u	**Gray**	**1**		**NIL**
v	**Gray**	**2**		**u**
w	White			NIL
x	White			NIL
y	White			NIL
z	White			NIL

u = v time = 2

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u	→ v → x
v	→ y
w	→ y → z
x	→ v
y	→ x
z	→ z

Execution example



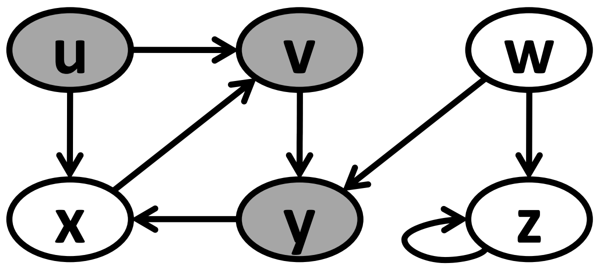
Vertex	Color	Timestamp		Predecessor
		d	f	(π)
u	**Gray**	**1**		**NIL**
v	**Gray**	**2**		**u**
w	White			NIL
x	White			NIL
y	**White**			**v**
z	White			NIL

u = v time = 2

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
        if v.color == WHITE
                v.π = u
                DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u	→ v → x
v	→ y
w	→ y → z
x	→ v
y	→ x
z	→ z

Execution example



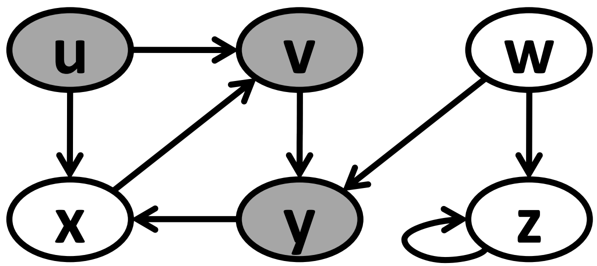
Vertex	Color	Timestamp d	f	Predecessor (π)
u	**Gray**	**1**		**NIL**
v	**Gray**	**2**		**u**
w	White			NIL
x	White			NIL
y	**Gray**	**3**		**v**
z	White			NIL

u = y   time = 3

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u	→ v → x
v	→ y
w	→ y → z
x	→ v
y	→ x
z	→ z

Execution example



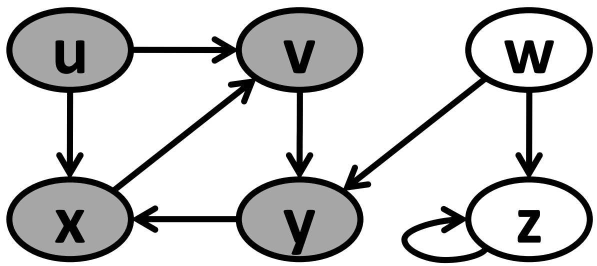
Vertex	Color	Timestamp d	Timestamp f	Predecessor (π)
u	**Gray**	**1**		**NIL**
v	**Gray**	**2**		**u**
w	White			NIL
x	**White**			**y**
y	**Gray**	**3**		**v**
z	White			NIL

$u = y$ time = 3

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
      if v.color == WHITE
            v.π = u
            DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u	→ v → x
v	→ y
w	→ y → z
x	→ v
y	→ x
z	→ z

Execution example



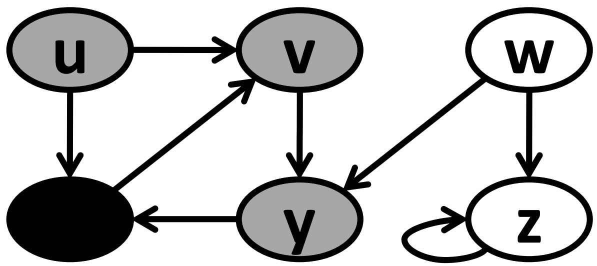
Vertex	Color	Timestamp d	f	Predecessor (π)
u	**Gray**	**1**		**NIL**
v	**Gray**	**2**		**u**
w	White			NIL
x	**Gray**	**4**		**y**
y	**Gray**	**3**		**v**
z	White			NIL

u = x   time = 4

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u	→	v	→	x
v	→	y		
w	→	y	→	z
x	→	v		
y	→	x		
z	→	z		

Execution example



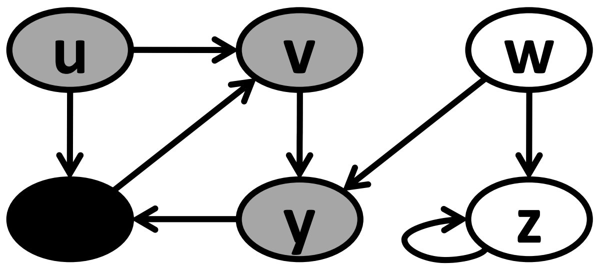
Vertex	Color	Timestamp d	Timestamp f	Predecessor (π)
u	**Gray**	**1**		**NIL**
v	**Gray**	**2**		**u**
w	White			NIL
x	**Black**	**4**	**5**	**y**
y	**Gray**	**3**		**v**
z	White			NIL

u = x time = 5

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u	→ v → x
v	→ y
w	→ y → z
x	→ v
y	→ x
z	→ z

Execution example



Vertex	Color	Timestamp d	Timestamp f	Predecessor (π)
u	**Gray**	**1**		**NIL**
v	**Gray**	**2**		**u**
w	White			NIL
x	**Black**	**4**	**5**	**y**
y	**Gray**	**3**		**v**
z	White			NIL

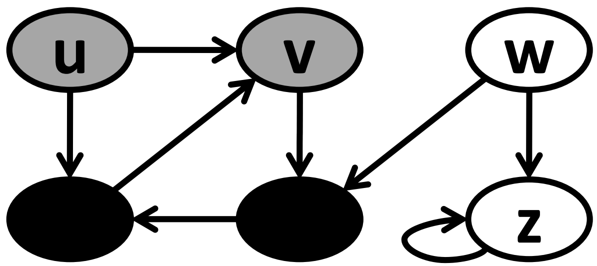
u = y　　　　　time = 5

DFS-VISIT(G, u)

```
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
        if v.color == WHITE
                v.π = u
                DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u	→ v → x
v	→ y
w	→ y → z
x	→ v
y	→ x
z	→ z

Execution example



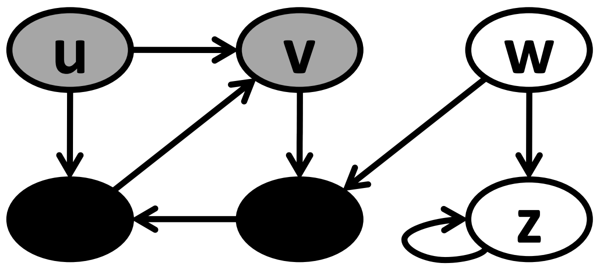
Vertex	Color	Timestamp d	Timestamp f	Predecessor (π)
u	**Gray**	**1**		**NIL**
v	**Gray**	**2**		**u**
w	White			NIL
x	**Black**	**4**	**5**	**y**
y	**Black**	**3**	**6**	**v**
z	White			NIL

u = y time = 6

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u	→ v → x
v	→ y
w	→ y → z
x	→ v
y	→ x
z	→ z

Execution example



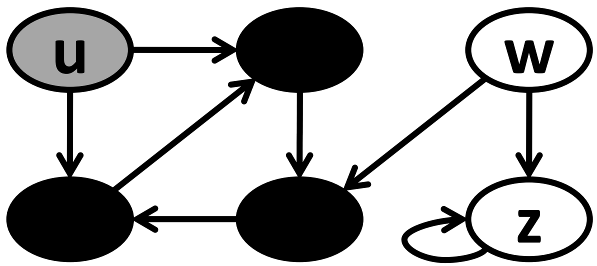
Vertex	Color	Timestamp d	Timestamp f	Predecessor (π)
u	**Gray**	**1**		**NIL**
v	**Gray**	**2**		**u**
w	White			NIL
x	**Black**	**4**	**5**	**y**
y	**Black**	**3**	**6**	**v**
z	White			NIL

u = v time = 6

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
        if v.color == WHITE
                v.π = u
                DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u → v → x
v → y
w → y → z
x → v
y → x
z → z

Execution example



Vertex	Color	Timestamp		Predecessor (π)
		d	f	
u	**Gray**	**1**		**NIL**
v	**Black**	**2**	**7**	**u**
w	White			NIL
x	**Black**	**4**	**5**	**y**
y	**Black**	**3**	**6**	**v**
z	White			NIL

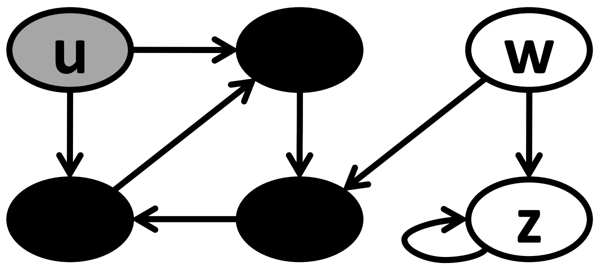
u = v   time = 7

DFS-VISIT(G, u)

```
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u	$\rightarrow$ v $\rightarrow$ x
v	$\rightarrow$ y
w	$\rightarrow$ y $\rightarrow$ z
x	$\rightarrow$ v
y	$\rightarrow$ x
z	$\rightarrow$ z

Execution example



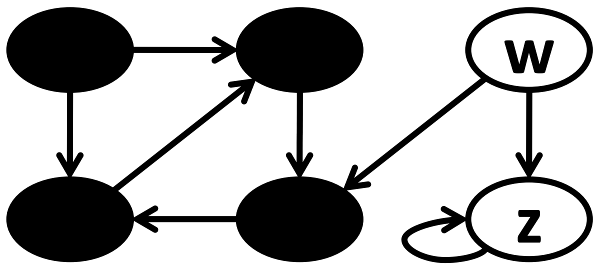
Vertex	Color	Timestamp d	Timestamp f	Predecessor (π)
u	**Gray**	**1**		**NIL**
v	**Black**	**2**	**7**	**u**
w	White			NIL
x	**Black**	**4**	**5**	**y**
y	**Black**	**3**	**6**	**v**
z	White			NIL

u = u   time = 7

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u → v → x
v → y
w → y → z
x → v
y → x
z → z

Execution example



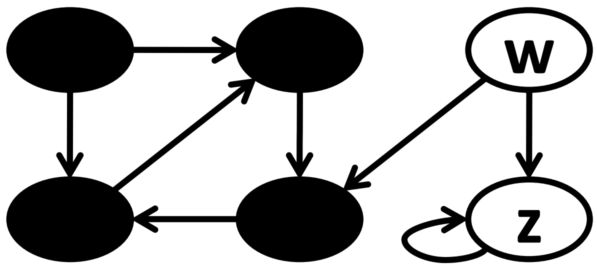
Vertex	Color	Timestamp d	Timestamp f	Predecessor (π)
u	**Black**	**1**	**8**	**NIL**
v	**Black**	**2**	**7**	**u**
w	White			NIL
x	**Black**	**4**	**5**	**y**
y	**Black**	**3**	**6**	**v**
z	White			NIL

u = u time = 8

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u	→ v → x
v	→ y
w	→ y → z
x	→ v
y	→ x
z	→ z

Execution example

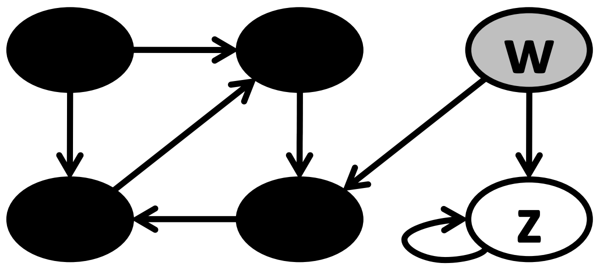


Vertex	Color	Timestamp		Predecessor (π)
		d	f	
u	**Black**	**1**	**8**	**NIL**
v	**Black**	**2**	**7**	**u**
w	White			NIL
x	**Black**	**4**	**5**	**y**
y	**Black**	**3**	**6**	**v**
z	White			NIL

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

u	→	v	→	x
v	→	y		
w	→	y	→	z
x	→	v		
y	→	x		
z	→	z		

Execution example



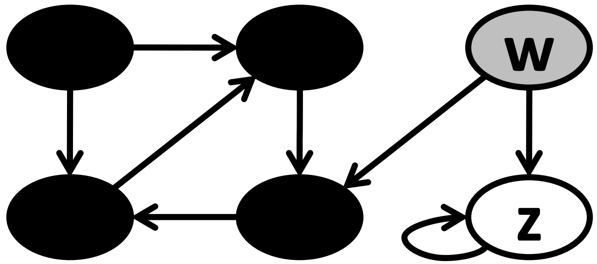
Vertex	Color	Timestamp		Predecessor (π)
		d	f	
u	**Black**	**1**	**8**	**NIL**
v	**Black**	**2**	**7**	**u**
w	**Gray**	**9**		**NIL**
x	**Black**	**4**	**5**	**y**
y	**Black**	**3**	**6**	**v**
z	White			NIL

u = w time = 9

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u → v → x
v → y
w → y → z
x → v
y → x
z → z

Execution example



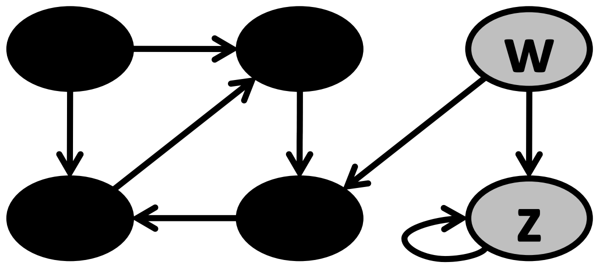
Vertex	Color	Timestamp d	Timestamp f	Predecessor (π)
u	Black	1	8	NIL
v	Black	2	7	u
w	Gray	9		NIL
x	Black	4	5	y
y	Black	3	6	v
z	White			w

u = w   time = 9

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
      if v.color == WHITE
            v.π = u
            DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u	→	v	→	x
v	→	y		
w	→	y	→	z
x	→	v		
y	→	x		
z	→	z		

Execution example



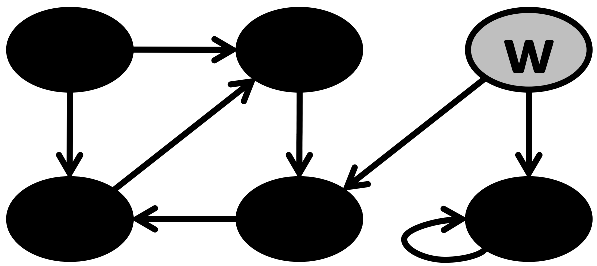
Vertex	Color	Timestamp d	Timestamp f	Predecessor (π)
u	Black	1	8	NIL
v	Black	2	7	u
w	Gray	9		NIL
x	Black	4	5	y
y	Black	3	6	v
z	Gray	10		w

u = z time = 10

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u → v → x
v → y
w → y → z
x → v
y → x
z → z

Execution example



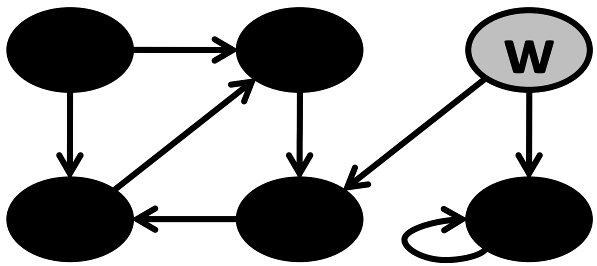
Vertex	Color	Timestamp d	Timestamp f	Predecessor (π)
u	Black	1	8	NIL
v	Black	2	7	u
w	Gray	9		NIL
x	Black	4	5	y
y	Black	3	6	v
z	Black	10	11	w

u = z time = 11

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u	→	v	→	x
v	→	y		
w	→	y	→	z
x	→	v		
y	→	x		
z	→	z		

Execution example



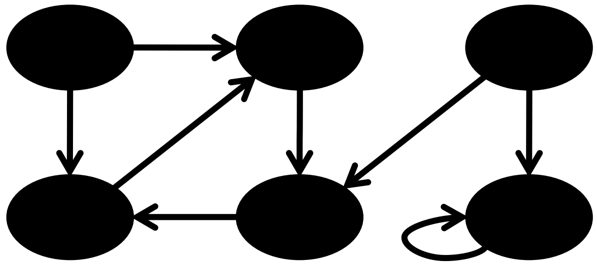
Vertex	Color	Timestamp d	Timestamp f	Predecessor (π)
u	Black	1	8	NIL
v	Black	2	7	u
w	Gray	9		NIL
x	Black	4	5	y
y	Black	3	6	v
z	Black	10	11	w

u = w   time = 11

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
        if v.color == WHITE
                v.π = u
                DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u	→ v → x
v	→ y
w	→ y → z
x	→ v
y	→ x
z	→ z

Execution example



Vertex	Color	Timestamp d	Timestamp f	Predecessor (π)
u	Black	1	8	NIL
v	Black	2	7	u
w	Black	9	12	NIL
x	Black	4	5	y
y	Black	3	6	v
z	Black	10	11	w

u = w time = 12

DFS-VISIT(G, u)

```
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
        if v.color == WHITE
                v.π = u
                DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

u	→ v → x
v	→ y
w	→ y → z
x	→ v
y	→ x
z	→ z

Execution example

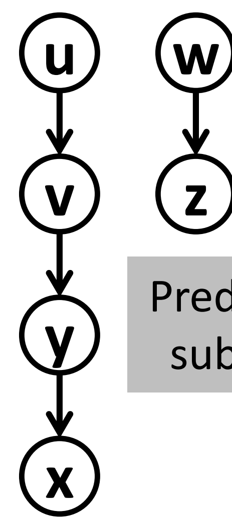
Vertex	Color	Timestamp		Predecessor (π)
		d	f	
u	Black	1	8	NIL
v	Black	2	7	u
w	Black	9	12	NIL
x	Black	4	5	y
y	Black	3	6	v
z	Black	10	11	w

```
DFS(G)
  for each vertex u ∈ G.V
      u.color = WHITE
      u.π = NIL
  time = 0
  for each vertex u ∈ G.V
      if u.color == WHITE
          DFS-VISIT(G, u)
```

u	→ v → x
v	→ y
w	→ y → z
x	→ v
y	→ x
z	→ z

DFS: u v y x w z

Depth-first forest



Predecessor sub-graph

Thank You