UNIVERSIDAD NACIONAL DE COLOMBIA

Algoritmia Avanzada

Sesión 4

Graph Algorithms

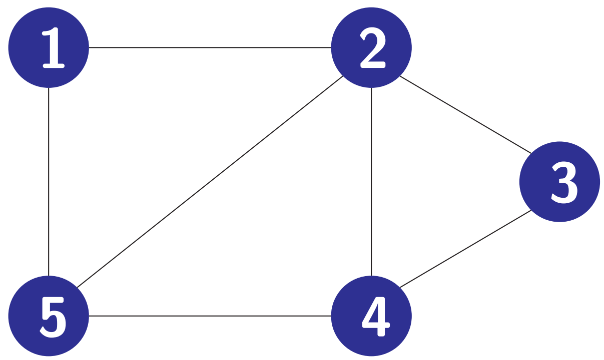
Yoan Pinzón, PhD
Universidad Nacional de Colombia
http://disi.unal.edu.co/~ypinzon/2019762/

Session 4

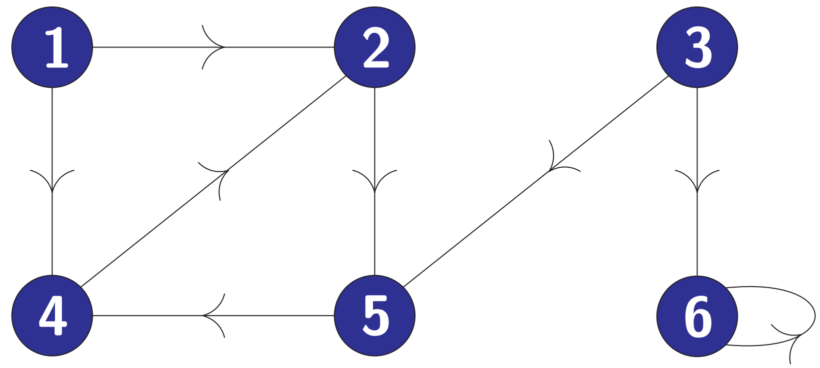
- **Graph Algorithms**
  - ▷ Definitions
  - ▷ Representation of a Graph
    - ◇ Adjacency-Matrix Representation
    - ◇ Adjacency-List Representation
  - ▷ Elementary Graph Algorithms
    - ◇ BFS (Breadth-First Search)
    - ◇ DFS (Depth-First Search)
    - ◇ SCC (Strongly Connected Components)
    - ◇ TS (Topological Sort)

Definitions

- A **graph** $G = (V, E)$ consist of a set V of vertices (*a.k.a* nodes) and a set E of edges (*a.k.a* arcs)
- Let $u, v \in V$, then (u, v) denotes the edge between vertex u and v. $n = |V|, m = |E|$



(a) An undirected graph



(b) A directed graph (digraph)

- An **undirected graph** is a graph with undirected edges ($(u, v) = (v, u)$)
- A **directed graph** is a graph with directed edges ($(u, v) \neq (v, u)$)
- Vertices u and v are **adjacent** vertices iff (u, v) is an edge in the graph
- The edge (u, v) is **incident** on the vertices u and v

Definitions (cont.)

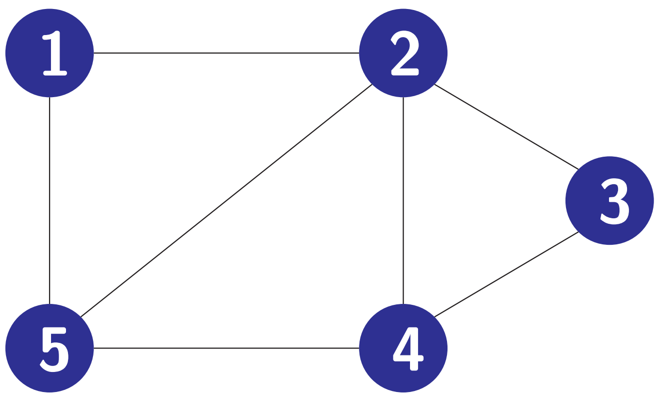
- A **path** P from v_1 to v_k is a sequence of vertices $P = < v_1, v_2, v_3, \ldots, v_k >$. P is said to be **simple** iff all vertices in P are unique
- A **cycle** in G is a path such that $v_1 = v_k$. A cycle is said to be **simple** iff all vertices in the path are unique except for the first one and the last one
- A **directed acyclic graph (DAG)** is a directed graph without cycles
- Let G be an *undirected graph*. The **degree** d_i of vertex i is the number of edges incident on vertex i
- Let G be a *digraph*. The **in-degree** d_i^{in} of vertex i is the number of edges incident to i (incoming edges). The **out-degree** d_i^{out} of vertex i is the number of edges incident from this vertex (outgoing edges)
- A **connected component** of an undirected graph G is a maximal subset of vertices such that for every pair of vertices u and v in this subset, there is a path from u to v and from v to u
- A **biconnected component** of a graph G is a maximal subset of *edges* such that any two edges in the subset lie on a common simple cycle

Adjacency-matrix Representation

n x n matrix A such that:

$$A_{i,j} = \begin{cases} 1 & \text{, if } (i,j) \in E \\ 0 & \text{, otherwise} \end{cases}$$

- **Undirected Graph:**



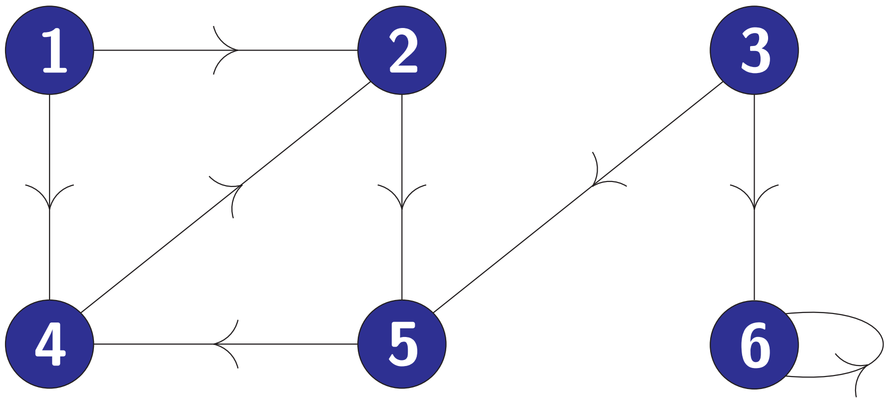
(a) An undirected graph G

$$\begin{array}{c|ccccc|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \\ \hline \end{array}$$

(b) The adjacency-matrix representation of G

Space Complexity: $O(n^2)$

▶ **Digraph:**



(a) A digraph G'

$$\begin{array}{c|cccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
2 & 0 & 0 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 0 & 0 & 1 & 1 \\
4 & 0 & 1 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
\end{array}$$

(b) The adjacency-matrix representation of G'

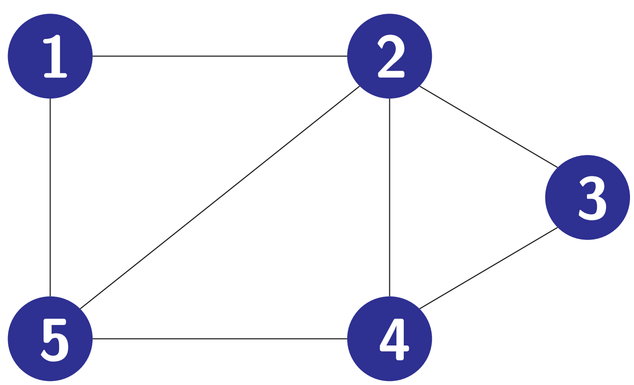
Space Complexity: $O(n^2)$

Adjacency-List Representation

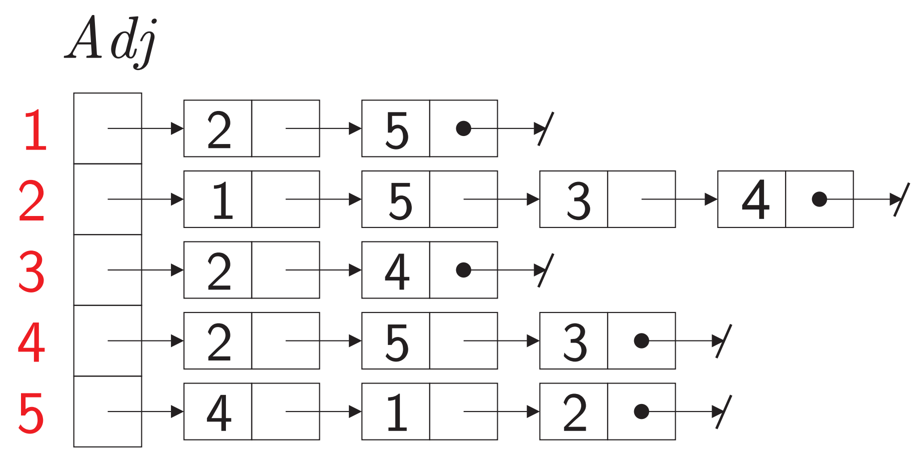
Array Adj of length n such that:

$$Adj[u] = \text{linked list of vertices adjacent to } u$$

▶ **Undirected Graph:**



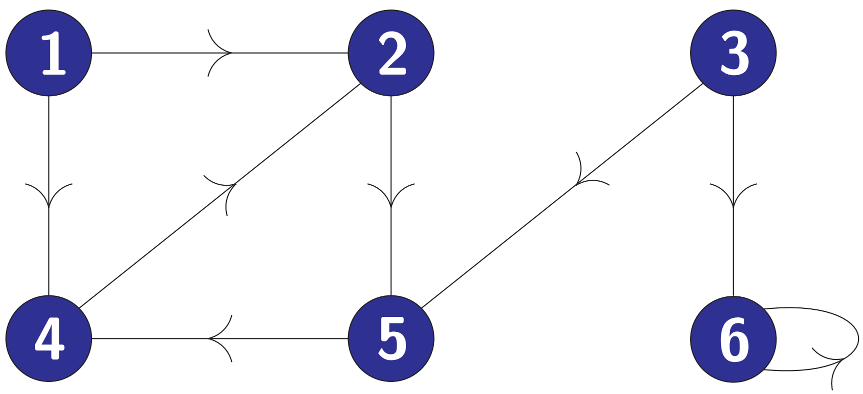
(a) An undirected graph G



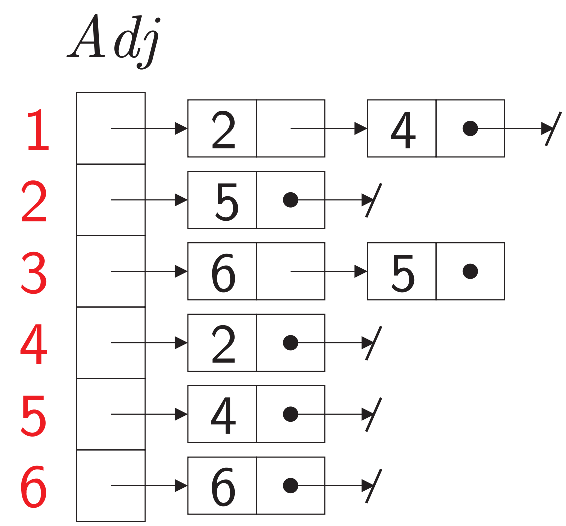
(b) The adjacency-list representation of G

Space Complexity: $O(n^2)$

▶ **Digraph:**



(a) A digraph G'



(b) The adjacency-list representation of G'

Space Complexity: $O(n^2)$

BFS: Breadth-First Search

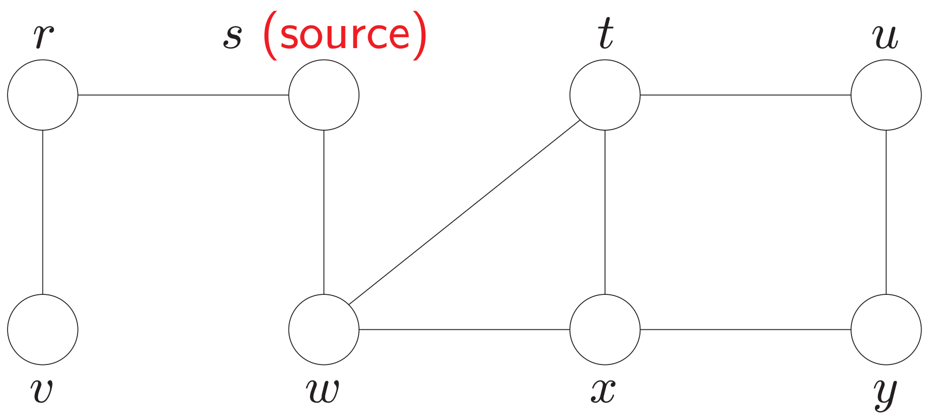
Pseudo-code

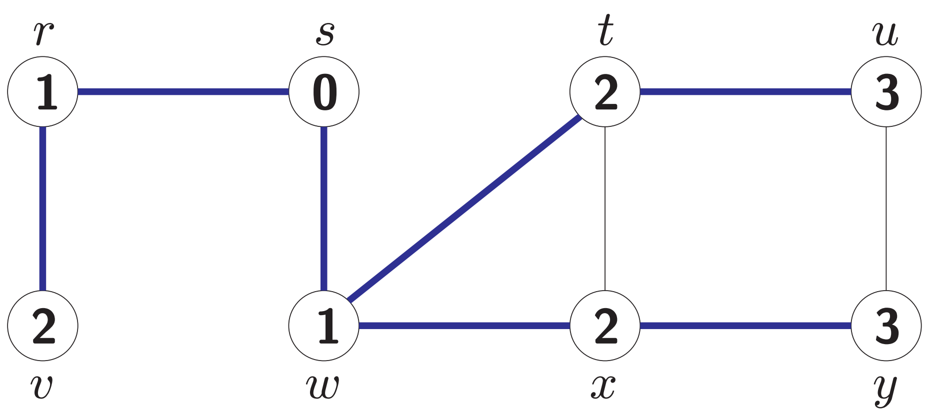
```
BFS(G, s) { n = |V|, m = |E|}
begin
    for each vertex v ∈ V − {s} do
        C_u ← "White"                    ▷ Color variable
        d_u ← ∞                          ▷ Distance variable
        π_u ← NIL                        ▷ Ancestor variable
    od
    C_s ← "Gray" , d_s ← 0, π ← NIL
    Q ← {s}                              ▷ FIFO queue
    while Q ≠ ∅ do
        u ← HEAD(Q)
        for each vertex v ∈ Adj_u do
            if C_v = "White" then do
                C_v ← "Gray" , d_v ← d_u + 1, π_v ← u
                ENQUEUE(Q, v)            ▷ Adds v to the queue
            od
        od
        DEQUEUE(Q)
        C_u ← "Black"
    od
end
```

- **Time Complexity:** $O(n + m)$

BFS: Breadth-First Search

Example

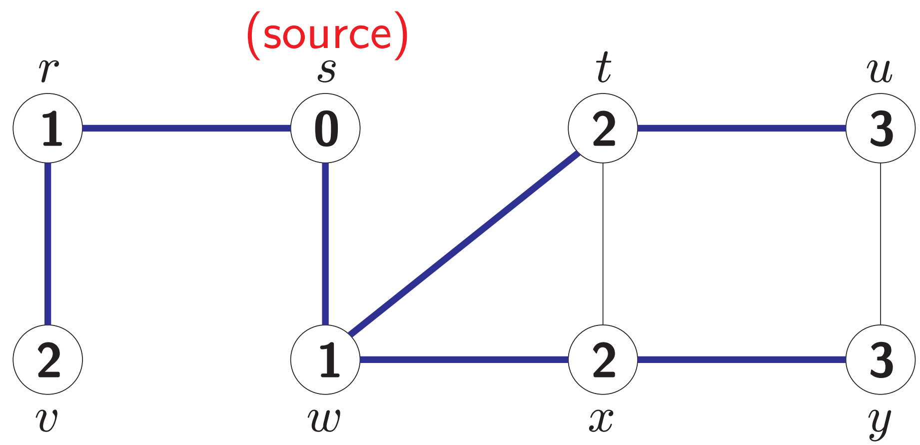




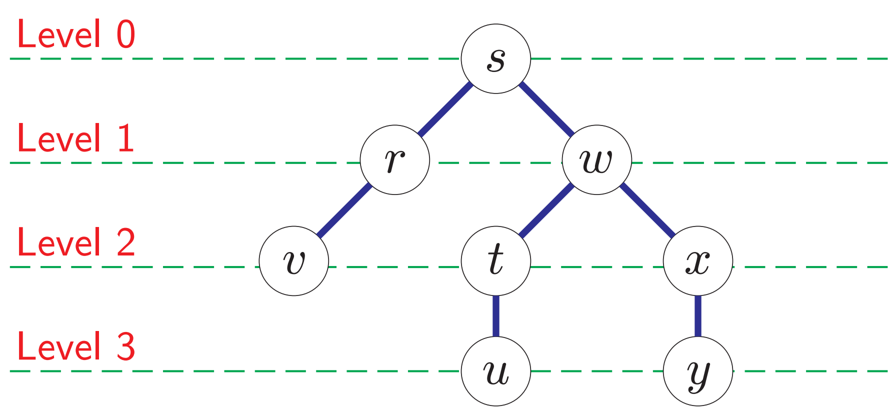
	r	s	t	u	v	w	x	y
	s	—	w	t	r	s	w	x
d	**1**	**0**	**2**	**3**	**2**	**1**	**2**	**3**

BFS: Breadth-First Search

Example (cont.)



	r	s	t	u	v	w	x	y
	s	—	w	t	r	s	w	x
d	**1**	**0**	**2**	**3**	**2**	**1**	**2**	**3**



DFS: Depth-First Search

Pseudocode

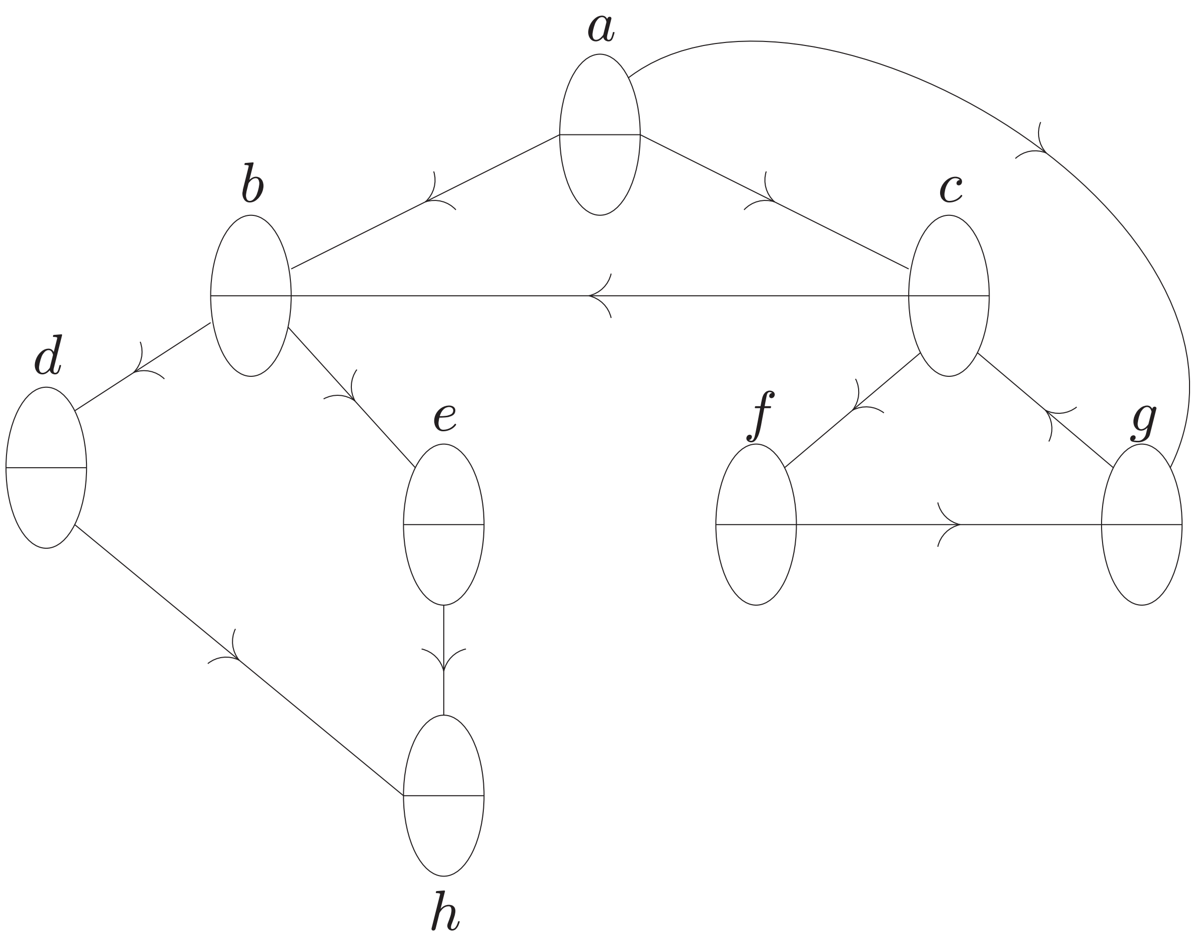
DFS(G) $\{ n = |V|, m = |E| \}$
begin
    $C_u \leftarrow$ "White", $\pi_u \leftarrow$ NIL $\forall u \in V$
    $t \leftarrow 0$
    for each vertex $u \in V$ **do**
        if $C_u =$ "White" **then** DFS-VISIT(u)
    od
end

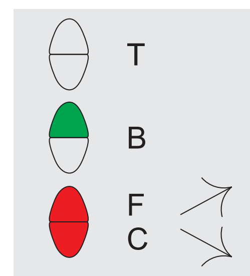
DFS-VISIT(u)
begin
    $C_u \leftarrow$ "Gray"
    $d_u \leftarrow t \leftarrow t + 1$
    for each vertex $v \in Adj_u$ **do**
        if $C_v =$ "White" **then do**
            $\pi_v \leftarrow u$
            DFS-VISIT(v)
        od
    od
    $C_u \leftarrow$ "Black"
    $f_u \leftarrow t \leftarrow t + 1$
end

- **Time Complexity:** $O(n + m)$

DFS: Depth-First Search

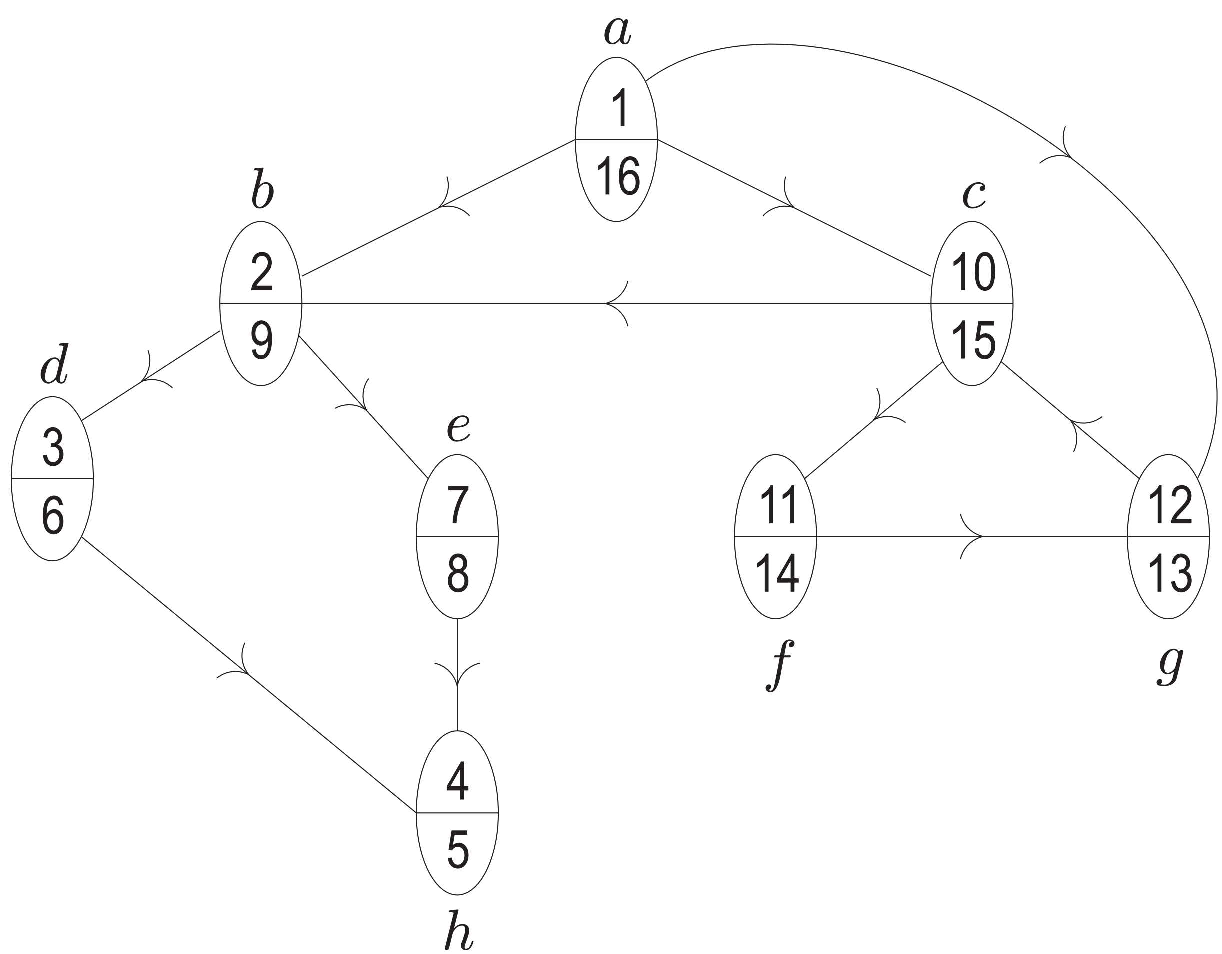
Example





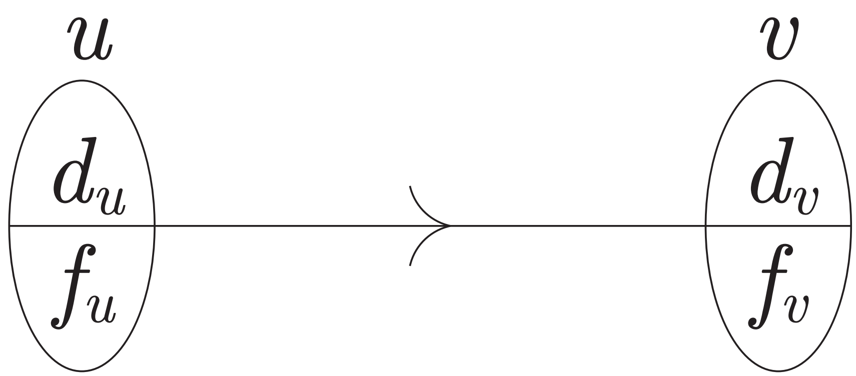
DFS: Depth-First Search

Example (cont.)



	a	b	c	d	e	f	g	h
d	1	2	10	3	7	11	12	4
f	16	9	15	6	8	14	13	5

Classification of edges

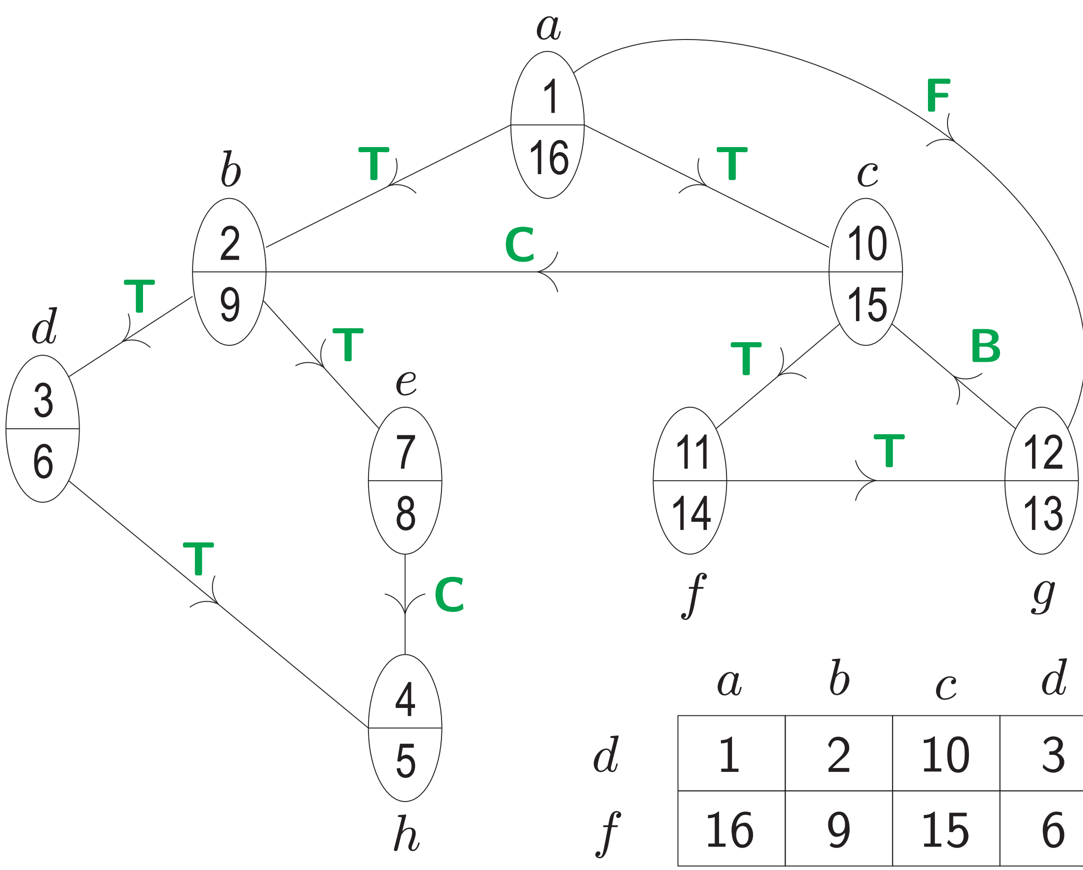


Edge (u, v) is:

- A **tree** edge or a **forward** edge iff $d_u < d_v < f_v < f_u$
- A **back** edge iff $d_v < d_u < f_u < f_v$
- A **cross** edge iff $d_v < f_v < d_u < f_u$

Classification of edges

Example



	a	b	c	d	e	f	g	h
d	1	2	10	3	7	11	12	4
f	16	9	15	6	8	14	13	5

Tree edges	(a,b), (b,d), (b,e), (d,h), (a,c), (c,f), (f,g)
Back edges	(g,c)
Forward edges	(a,g)
Cross edges	(e,h), (c,b)

SCC: Strongly Connected Components

- **Definition:** SCC of a digraph $G = (V, E)$ is a maximal subset of vertices $U \subseteq V$ such that for every pair of vertices u and v in U there is a path from u to v and from v to u.

- **Pseudo-code:**

$\mathrm{SCC}(G)$ $\{$ $n = |V|, m = |E|$ $\}$

1 **begin**

2 Call $\mathrm{DFS}(G)$ to compute f_u $\forall u \in V$

3 Compute G^T $\triangleright$ $G^T = (V, E^T), E^T = \{(u,v) : (v,u) \in E\}$

4 Call $\mathrm{DFS}(G^T)$ but considering the vertices in decreasing order using the f_u computed in step 2

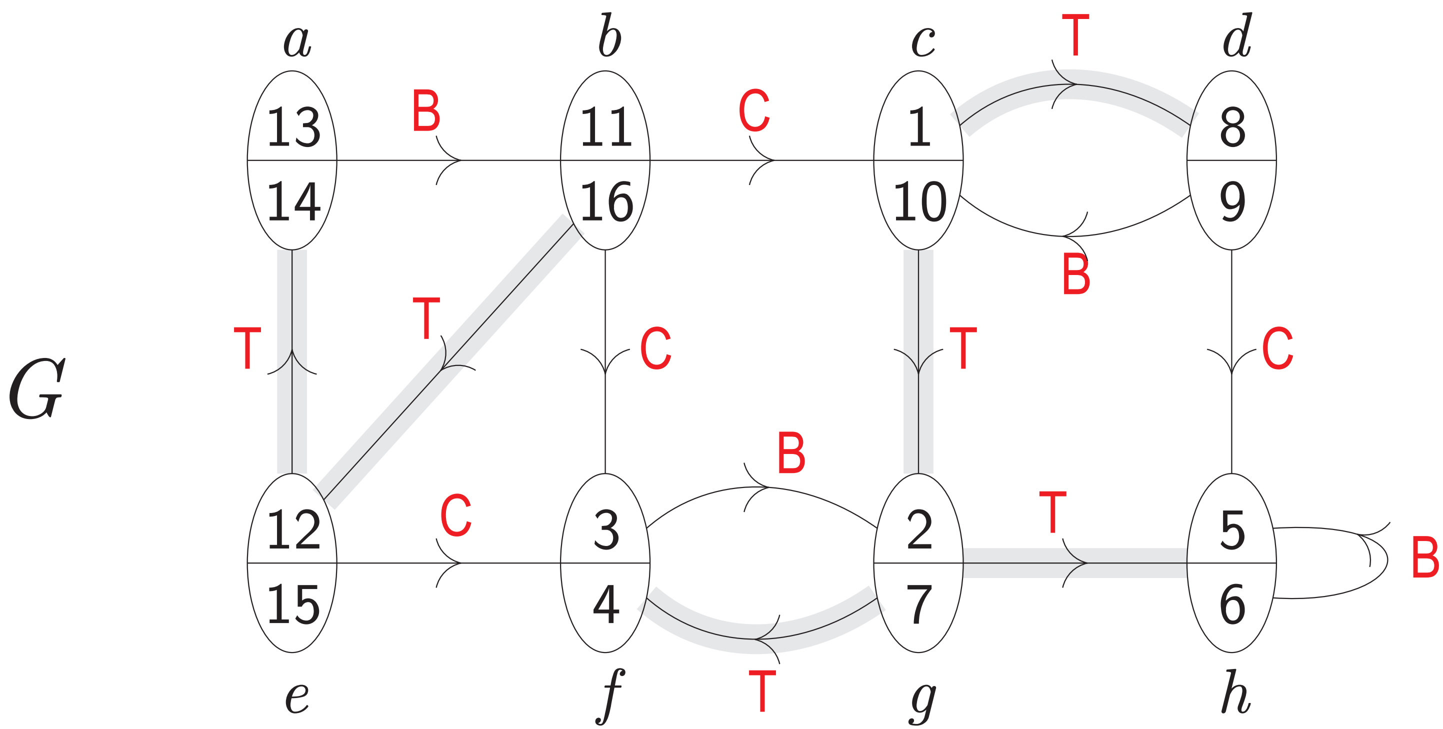
5 Output the vertices of each tree in step 4 as a separate SCC

6 **end**

- **Time Complexity:** $O(n+m)$

SCC: Strongly Connected Components

Example



	a	b	c	d	e	f	g	h
d	13	11	1	3	12	3	2	5
f	14	16	10	9	15	4	7	6

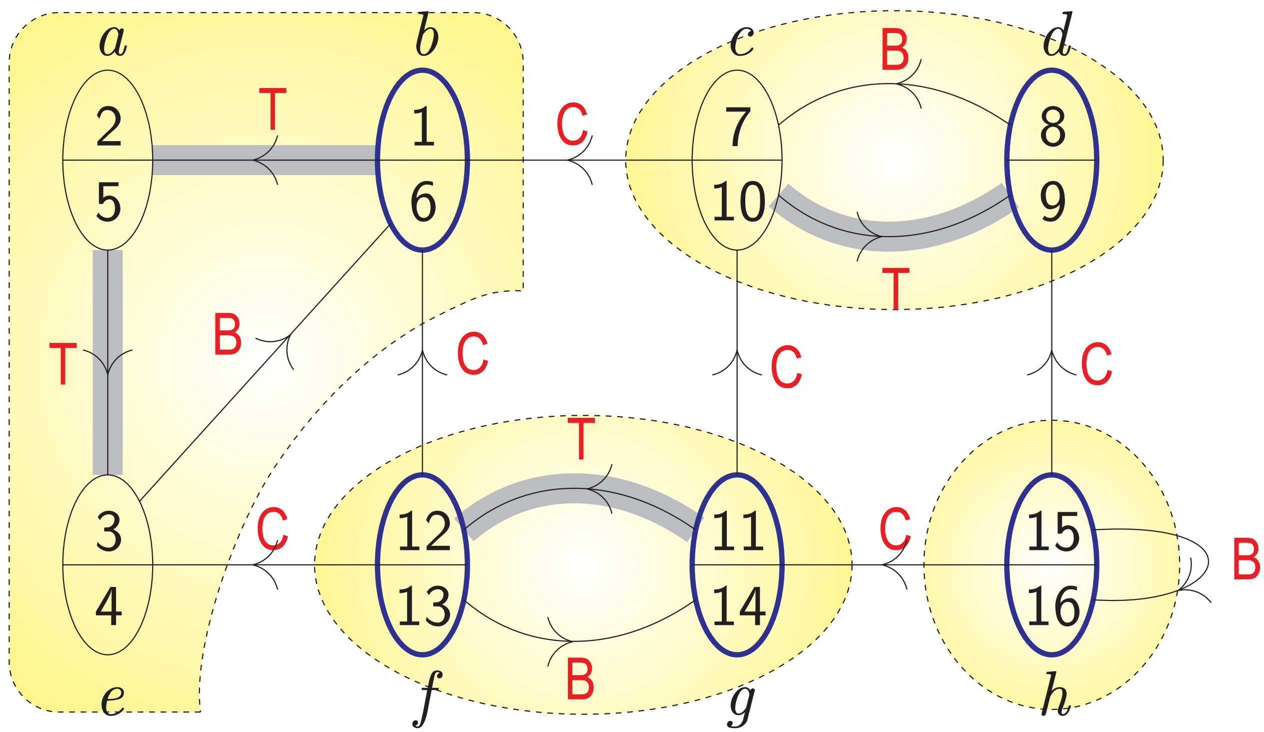
Vertices in order of decreasing f_u : b,e,a,c,d,g,h,f

SCC: Strongly Connected Components

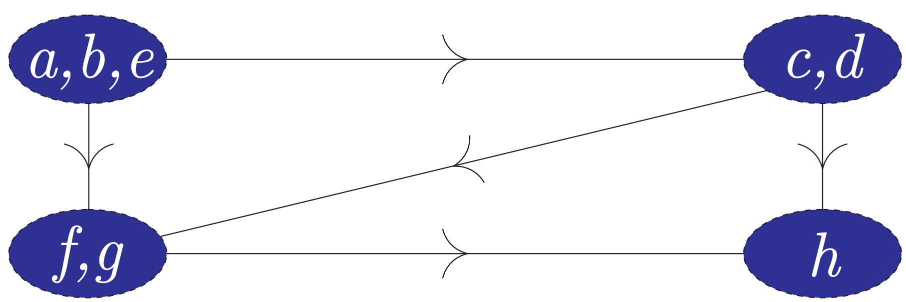
Example (cont.)

Vertices in order of decreasing f_u : b, e, a, c, d, g, h, f

G^T



G^{scc}



TS: Topological Sort

▶ **definition:** A *topological sort* of a DAG $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v), then u appears *before* v in the ordering.

▶ **Pseudo-code:**

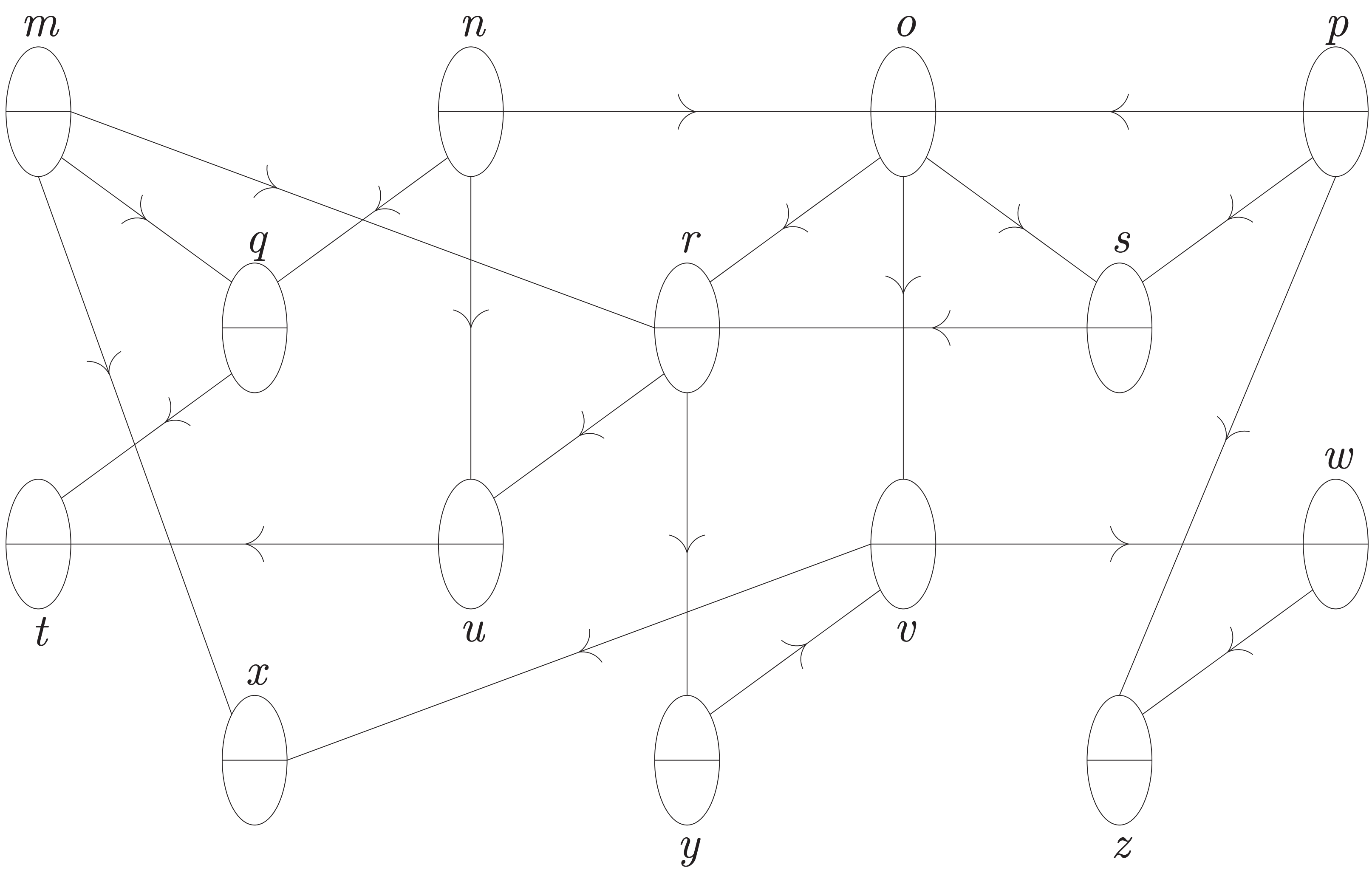
$\text{TS}(G)$ $\{ \ n = |V|, m = |E| \ \}$

begin
    Call $\text{DFS}(G)$ to compute $f_u \ \forall u \in V$
    as each vertex is finished, insert it onto the front of a linked list
    return the linked list of vertices
end

▶ **Time Complexity:** $O(n + m)$

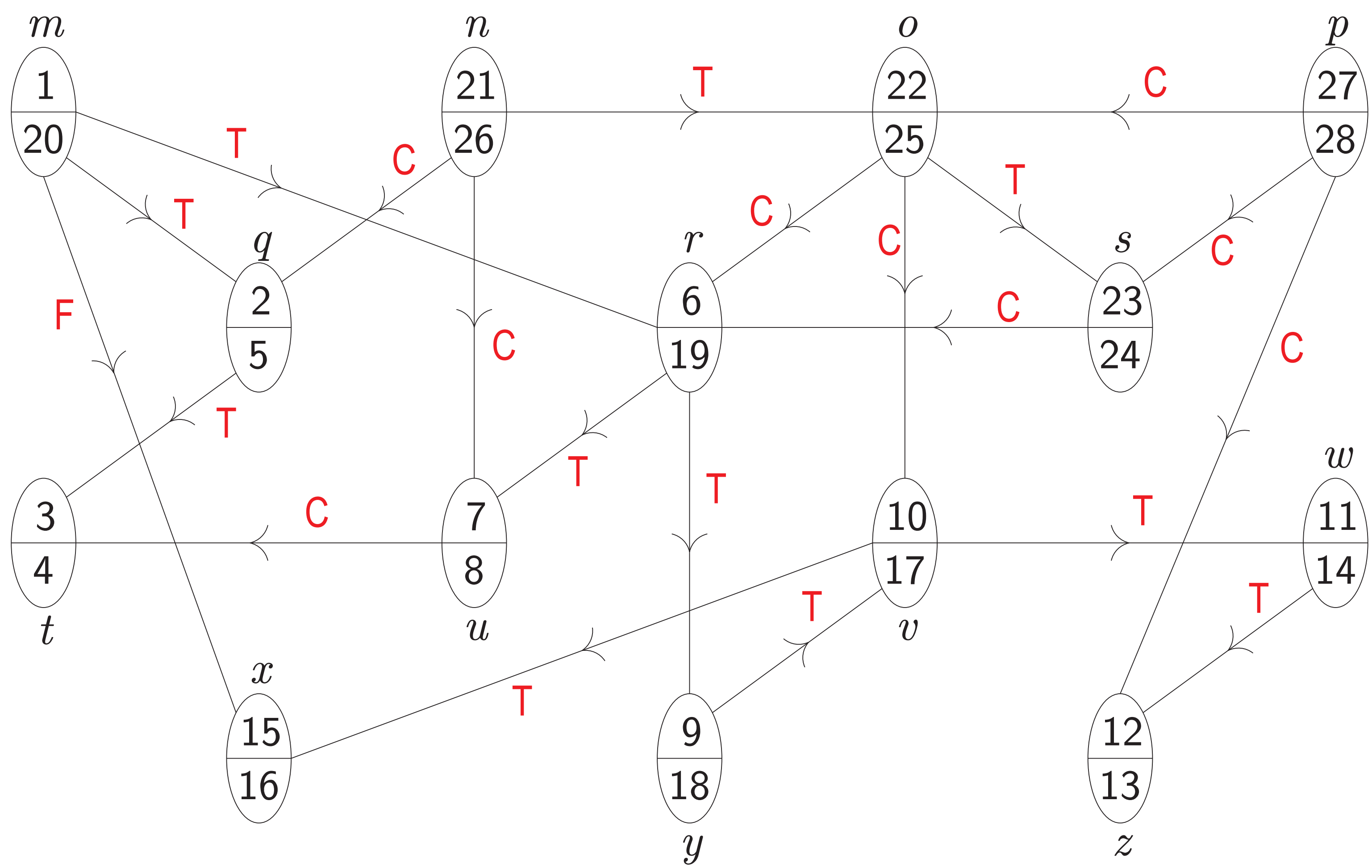
TS: Topological Sort

Example



TS: Topological Sort

Example (cont.)

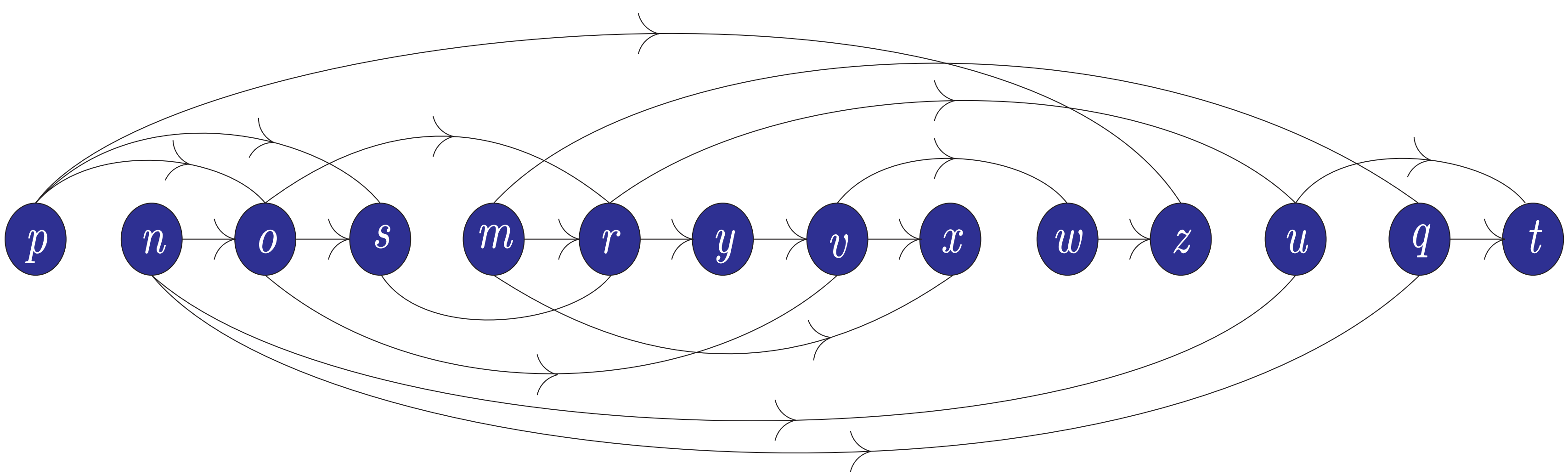


	m	n	o	p	q	r	s	t	u	v	w	x	y	z
d	1	21	22	27	2	6	23	3	7	10	11	15	9	12
f	20	26	25	28	5	19	24	4	8	17	14	16	18	13

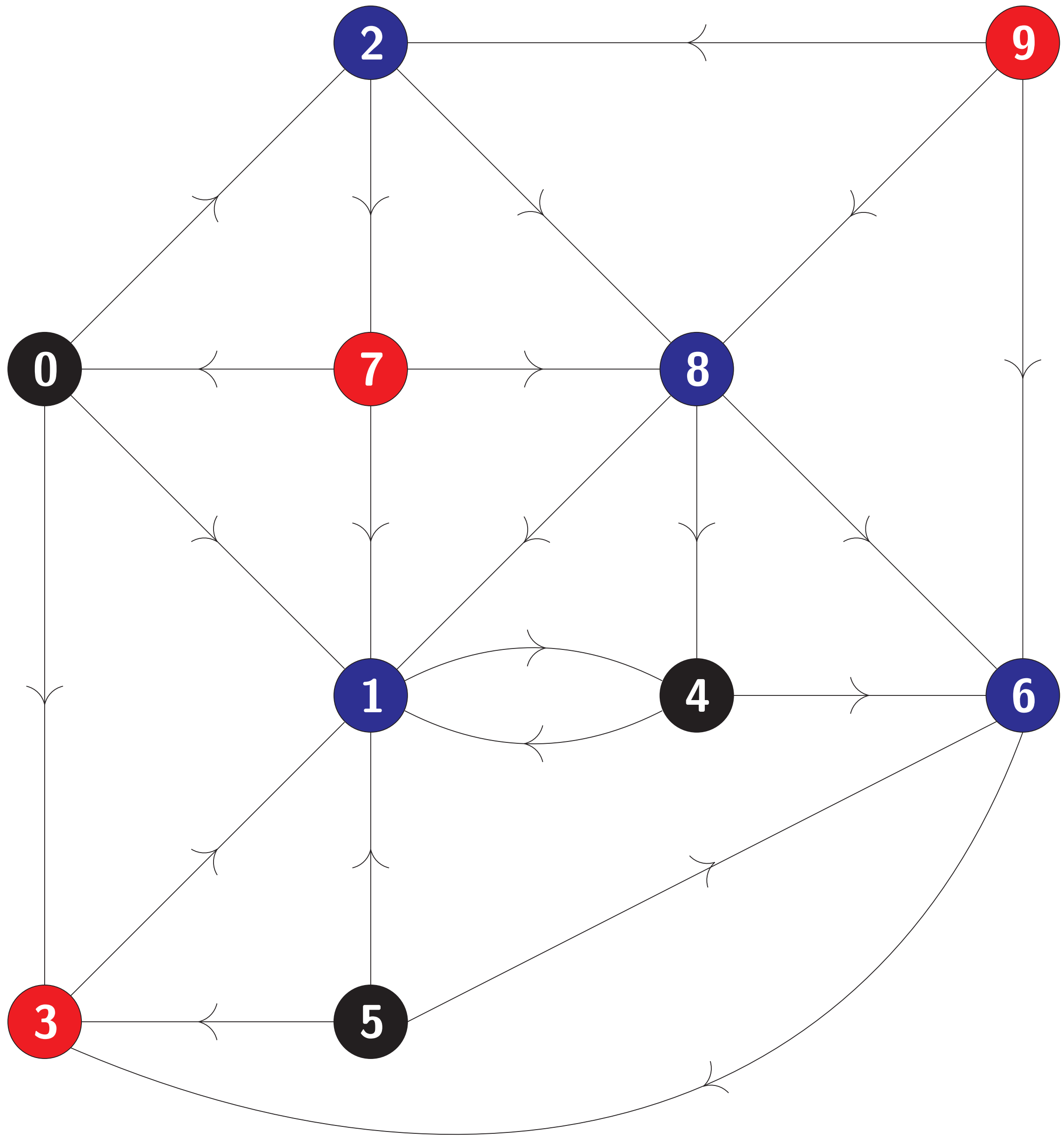
TS: Topological Sort

Example (cont.)

	m	n	o	p	q	r	s	t	u	v	w	x	y	z
d	1	21	22	27	2	6	23	3	7	10	11	15	9	12
f	20	26	25	28	5	19	24	4	8	17	14	16	18	13



SCC – Exercise



TS – Exercise