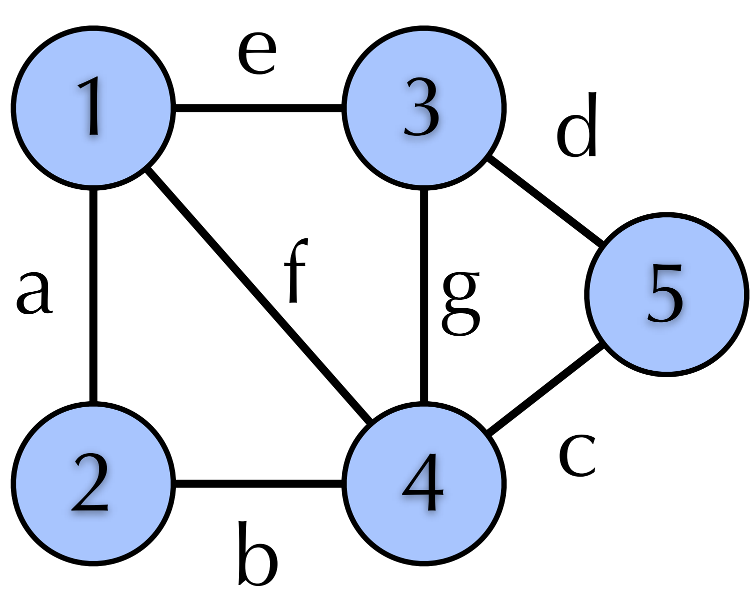
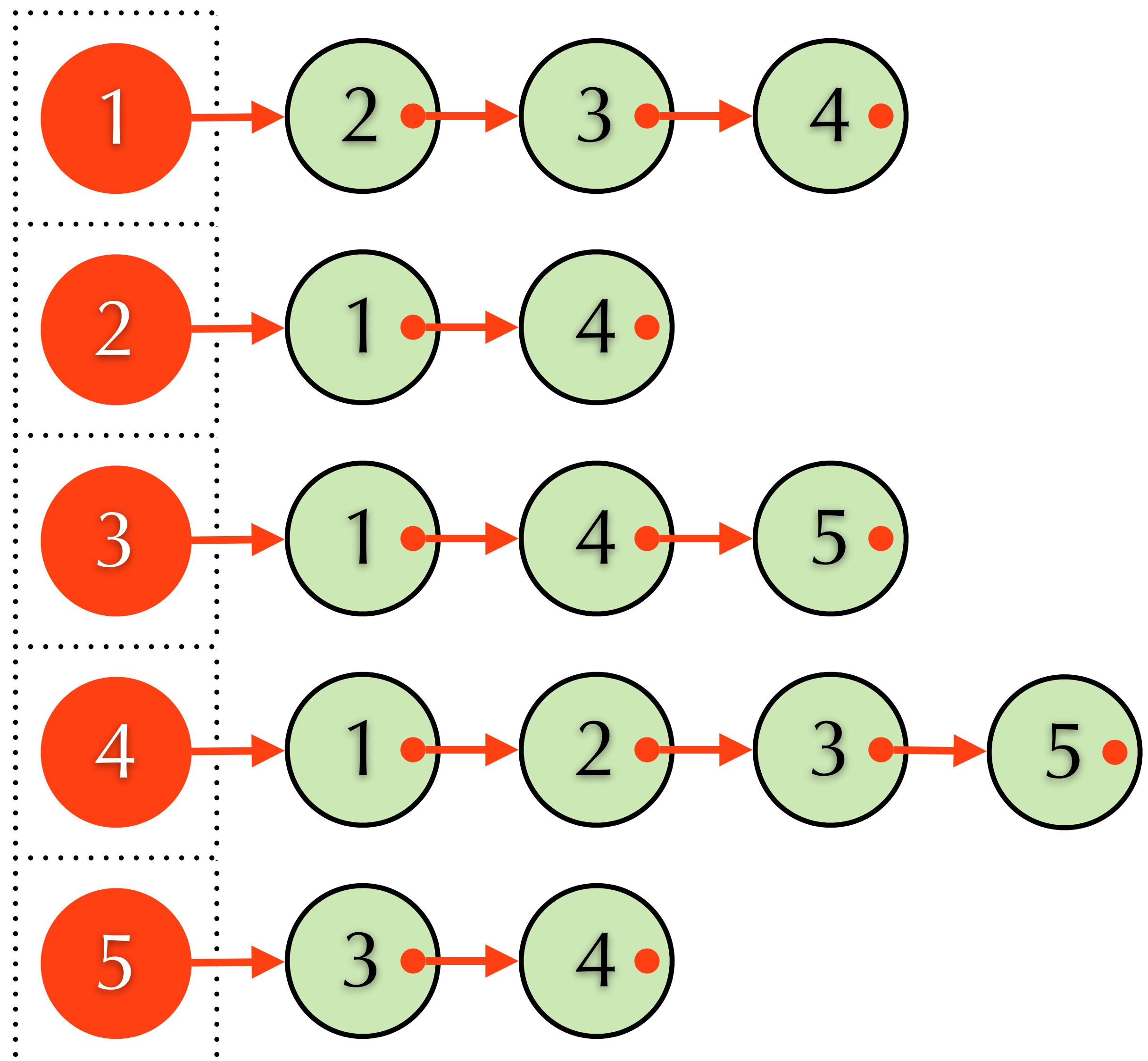
Elementary Graph Algorithms

Graph Representation

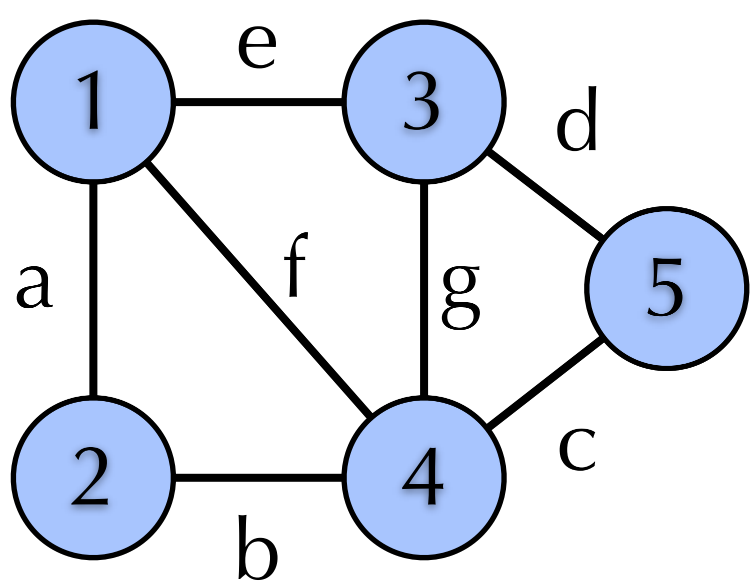
- G=(V,E)
  - V: set of vertices
  - E: set of edges
    - Directed edge: (u,v)
    - Undirected edge: {u,v}
- Graph representations affect the complexity of algorithms

Adjacency List: Undirected



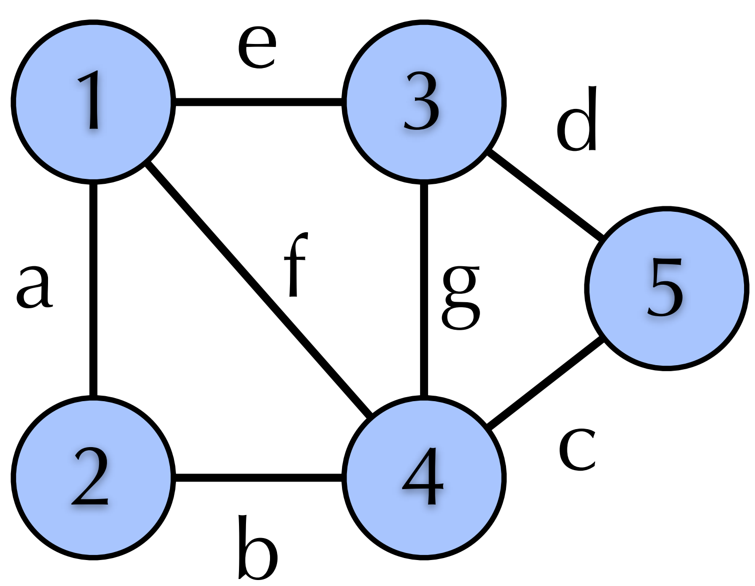


Adjacency Matrix: Undirected



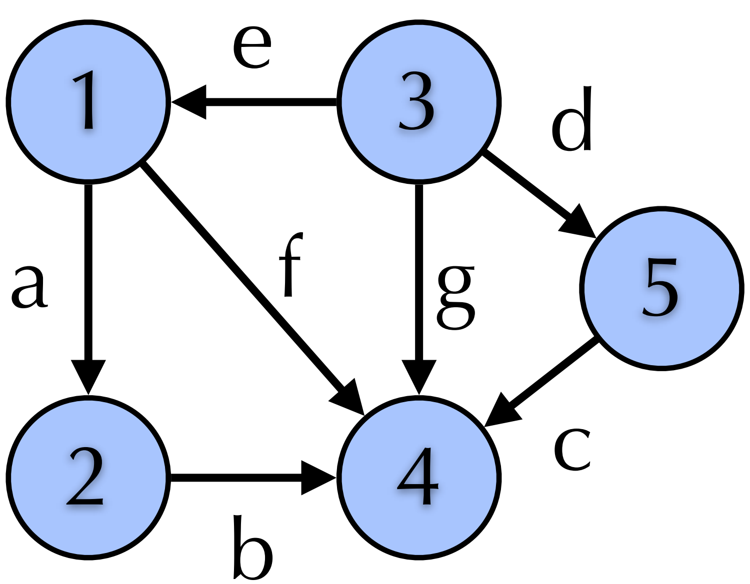
	1	2	3	4	5
1	0	1	1	1	0
2	1	0	0	1	0
3	1	0	0	1	1
4	1	1	1	0	1
5	0	0	1	1	0

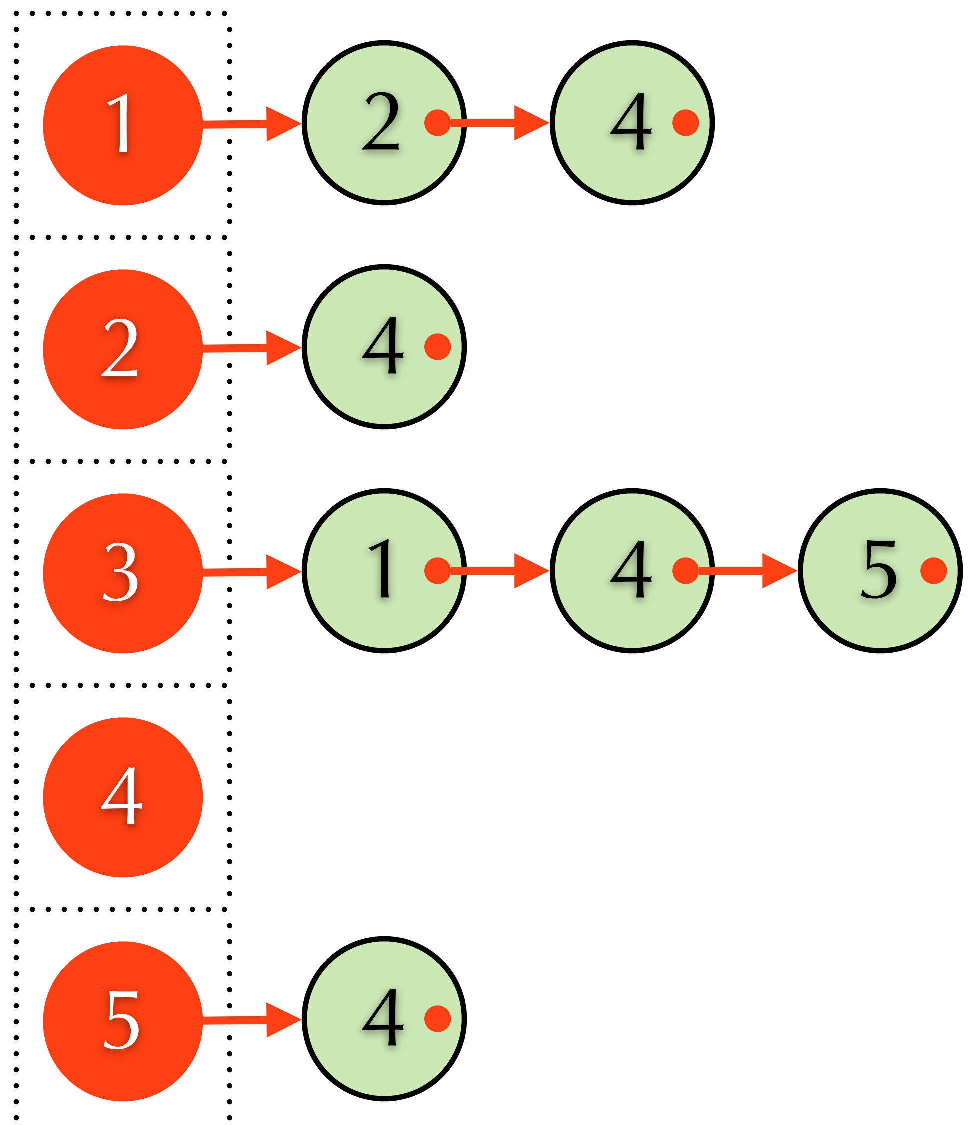
Incidence Matrix: Undirected



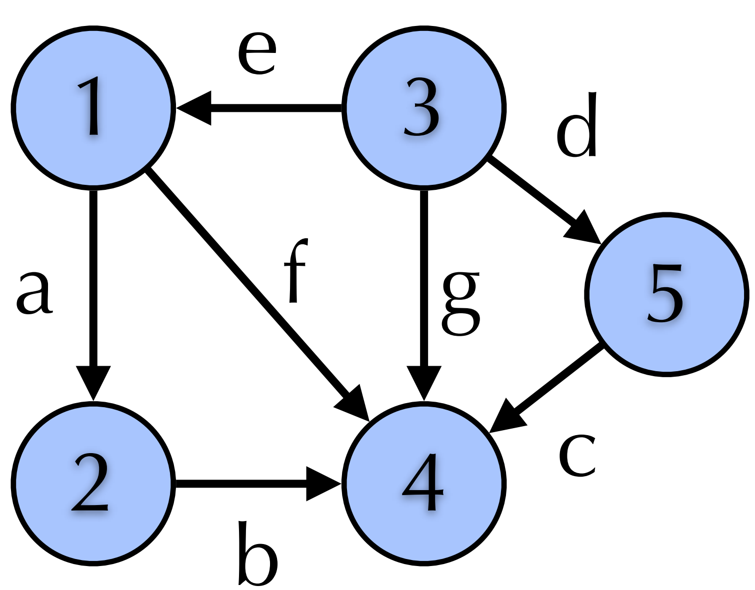
	a	b	c	d	e	f	g
1	1	0	0	0	1	1	0
2	1	1	0	0	0	0	0
3	0	0	0	1	1	0	1
4	0	1	1	0	0	1	1
5	0	0	1	1	0	0	0

Adjacency List: Directed



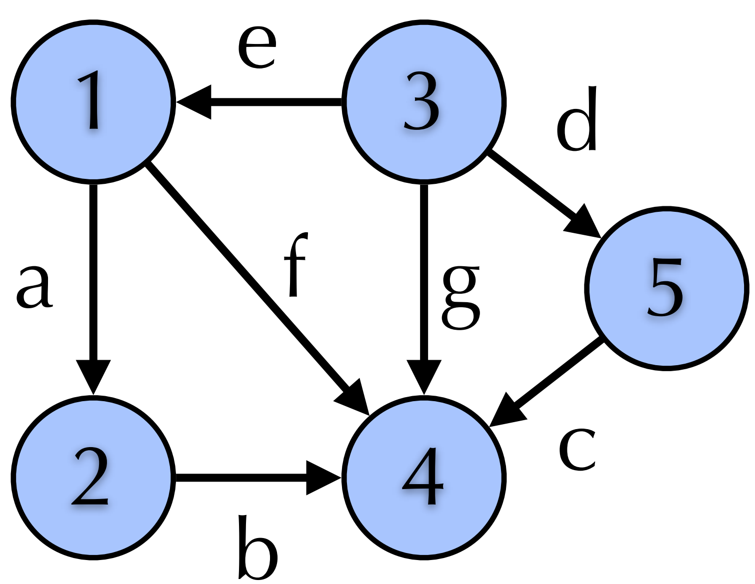


Adjacency Matrix: Directed



	1	2	3	4	5
1	0	1	0	1	0
2	0	0	0	1	0
3	1	0	0	1	1
4	0	0	0	0	0
5	0	0	0	1	0

Incidence Matrix: Directed



	a	b	c	d	e	f	g
1	–1	0	0	0	1	–1	0
2	1	–1	0	0	0	0	0
3	0	0	0	–1	–1	0	–1
4	0	1	1	0	0	1	1
5	0	0	–1	1	0	0	0

Graph Representation: Comparison

- Adjacency list
  - Space: $\Theta(|V|+|E|)$
  - Random access an edge: $O(d)$ d: degree
  - Enumerate edges incident to a vertex: $O(d)$
- Adjacency matrix
  - Space: $\Theta(|V|^2)$
  - Random access an edge: $\Theta(1)$
  - Enumerate edges incident to a vertex: $O(|V|)$

Breadth First Search

- Given a graph $G=(V,E)$ and a source vertex $s \in V$.
- BFS computes the set of vertices reachable from s.
- For each vertex v reachable set, BFS computes the shortest unweighted path from s to v.
- Use a queue

Breadth First Search

- Initialization: For each $v \in V \setminus \{s\}$
  - v.c=WHITE white color: not discovered
  - $v.d=\infty$ known min distance from s
  - $v.\pi$=NIL predecessor
- Initialization: vertex s
  - s.c=GRAY gray color: discovered & unvisited
  - s.d=0
  - $s.\pi$=NIL predecessor

Breadth First Search

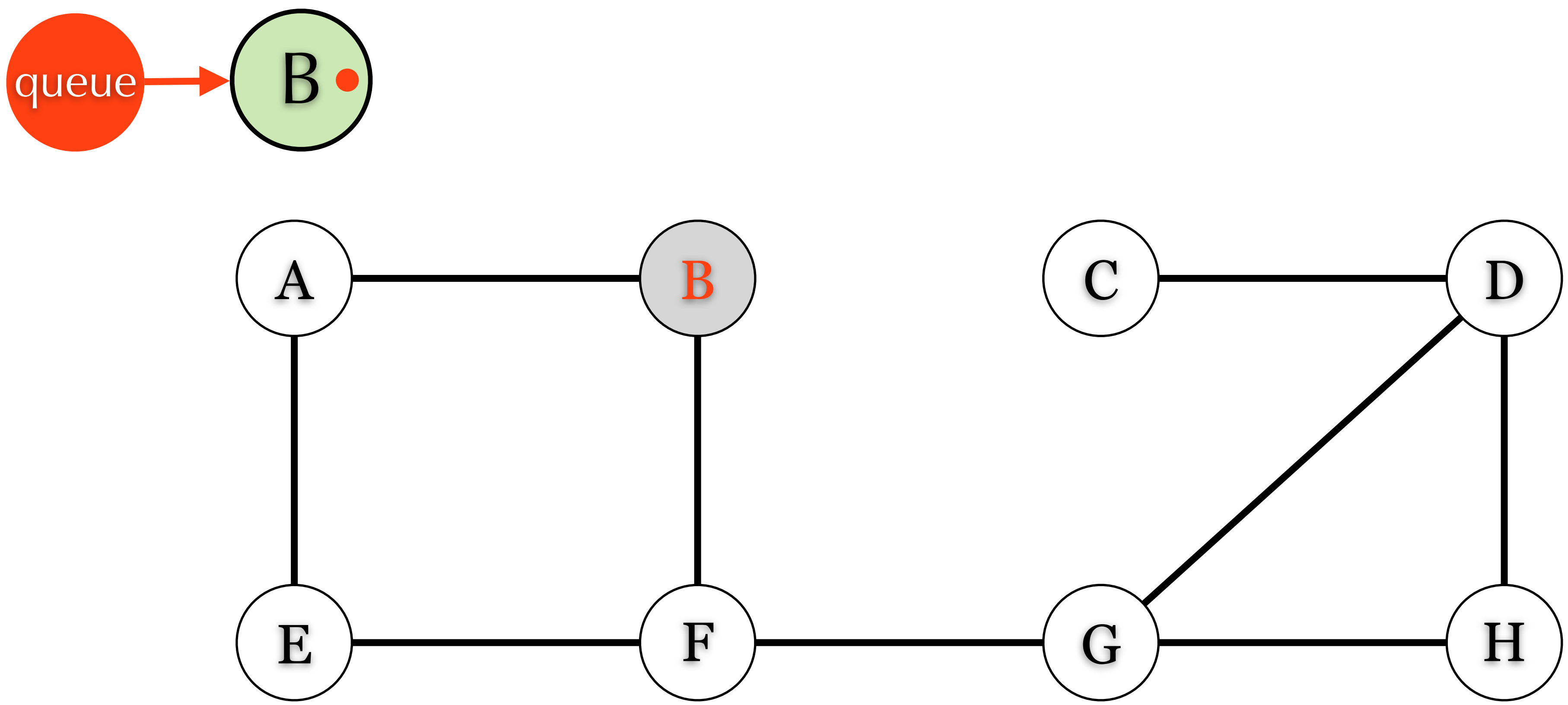
- Initialization: queue Q
  - Q=∅
  - Q.enqueue(s)
- Main loop: while(Q≠∅)
  - u=Q.dequeue()
  - For each v s.t. (u,v)∈E

    if v.c==WHITE

    v.c=GRAY, v.d=u.d+1, v.π=u, Q.enqueue(v)

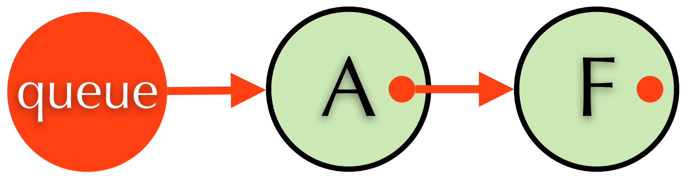
  u.c=BLACK black color: visited

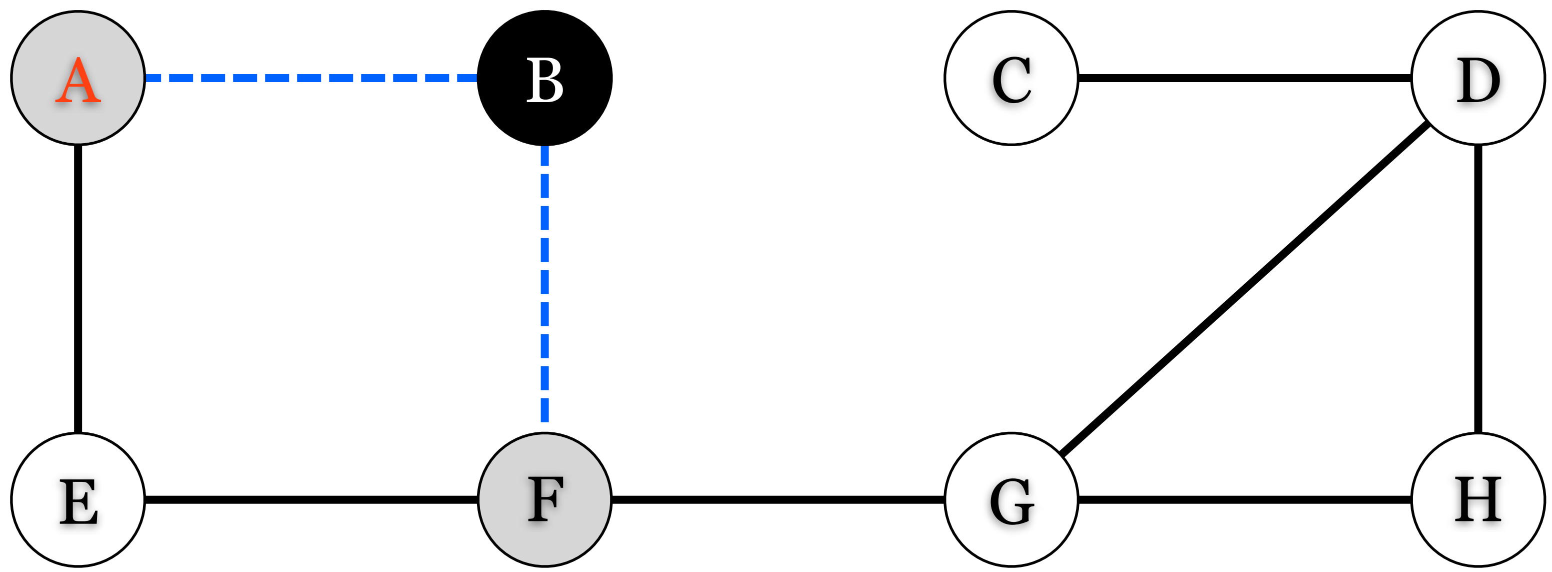
Breadth First Search



v	A	B	C	D	E	F	G	H
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL
d	∞	0	∞	∞	∞	∞	∞	∞

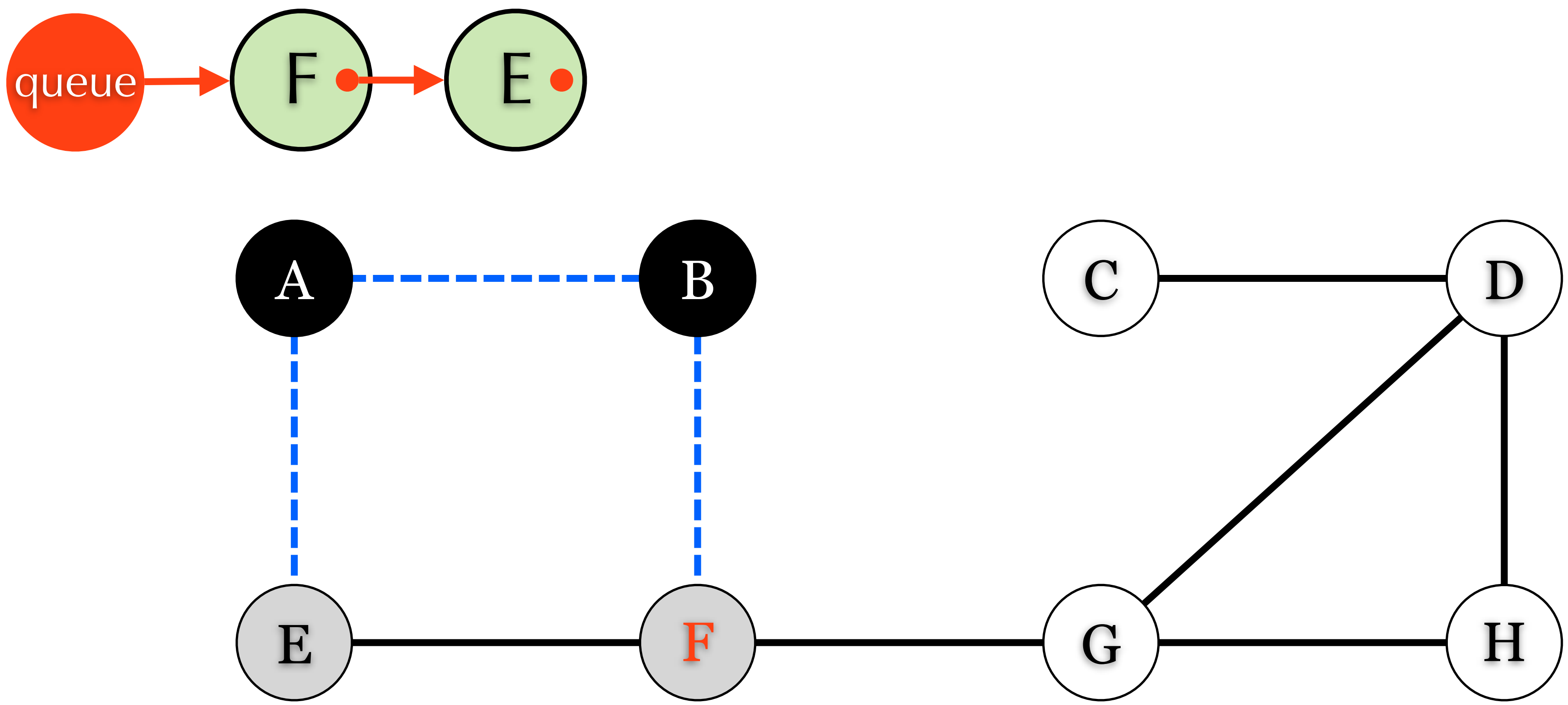
Breadth First Search





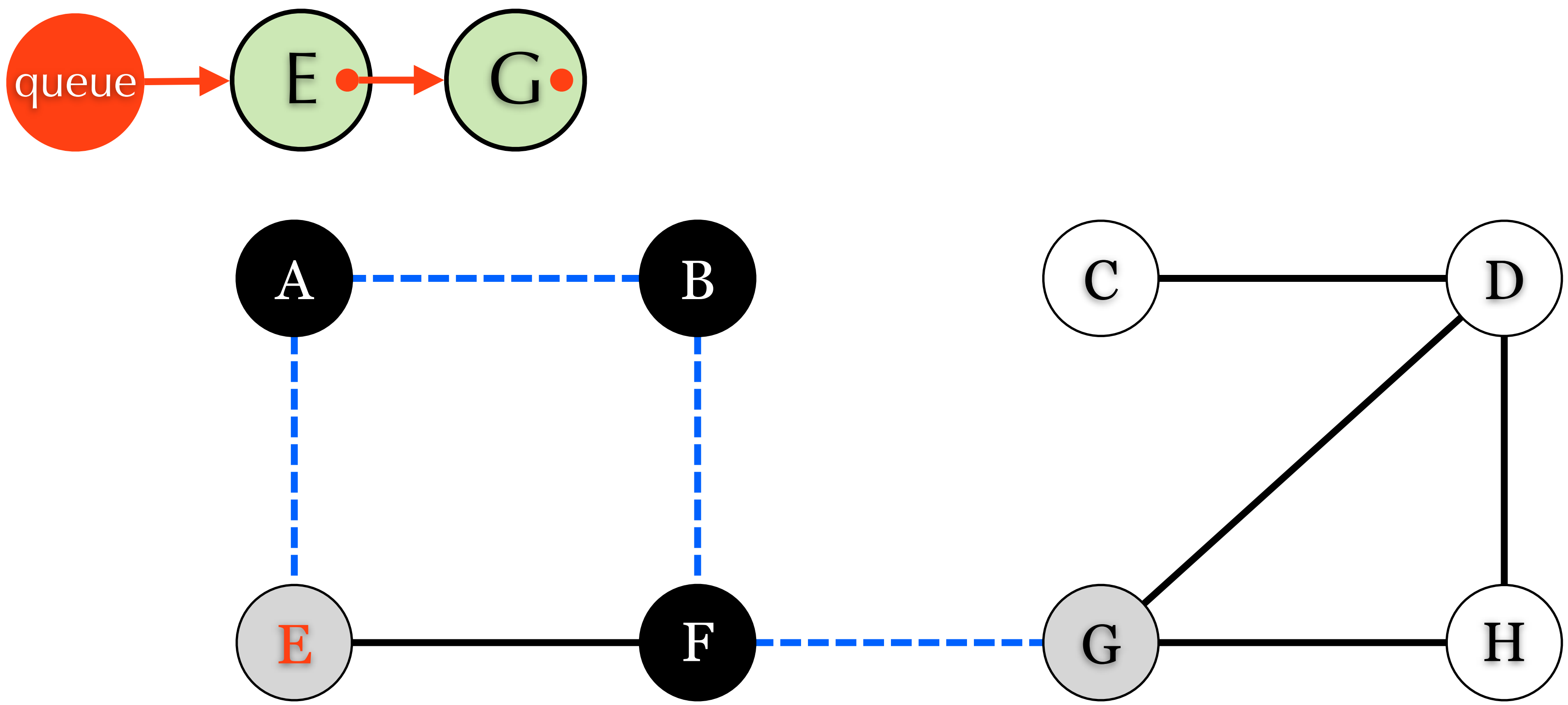
v	A	B	C	D	E	F	G	H
π	B	NIL	NIL	NIL	NIL	B	NIL	NIL
d	1	0	∞	∞	∞	1	∞	∞

Breadth First Search



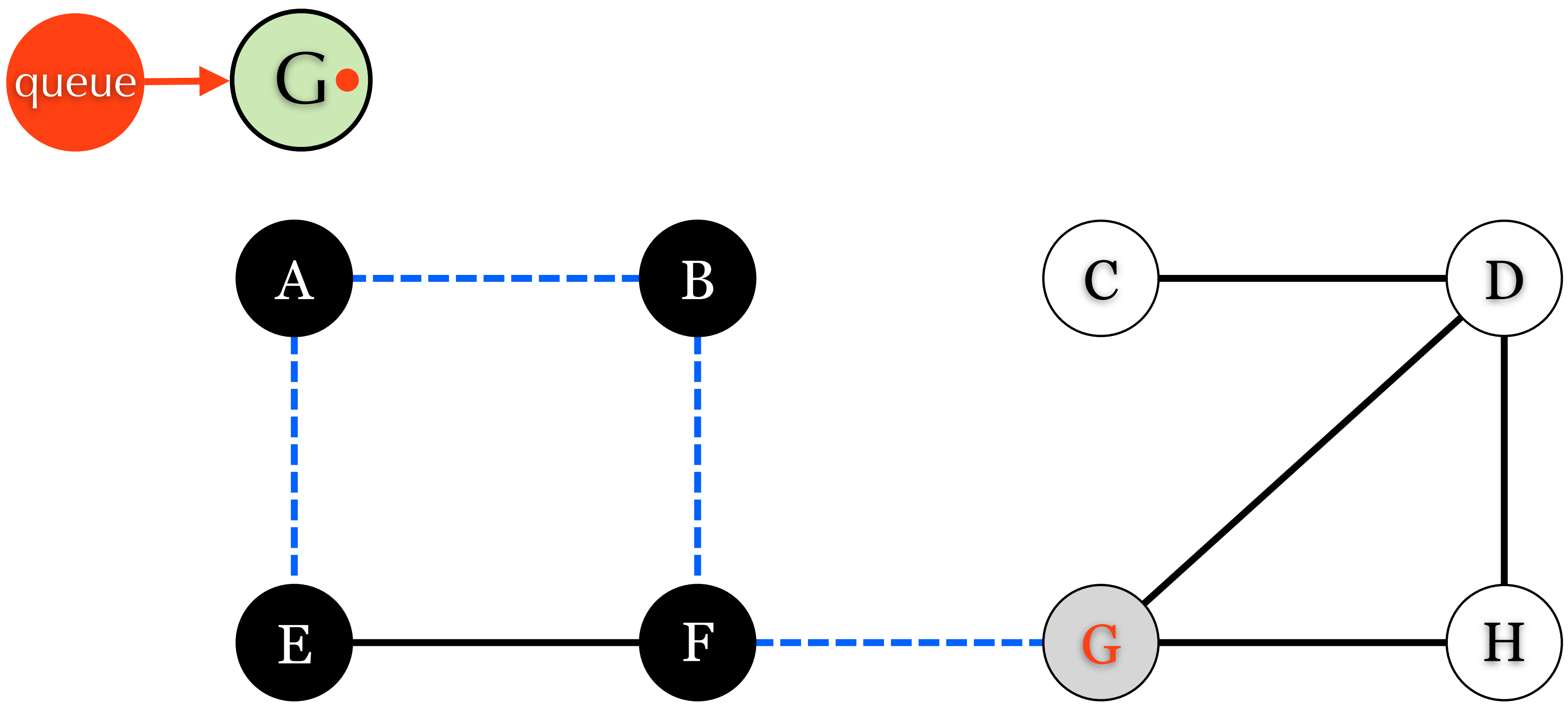
v	A	B	C	D	E	F	G	H
π	B	NIL	NIL	NIL	A	B	NIL	NIL
d	1	0	∞	∞	2	1	∞	∞

Breadth First Search



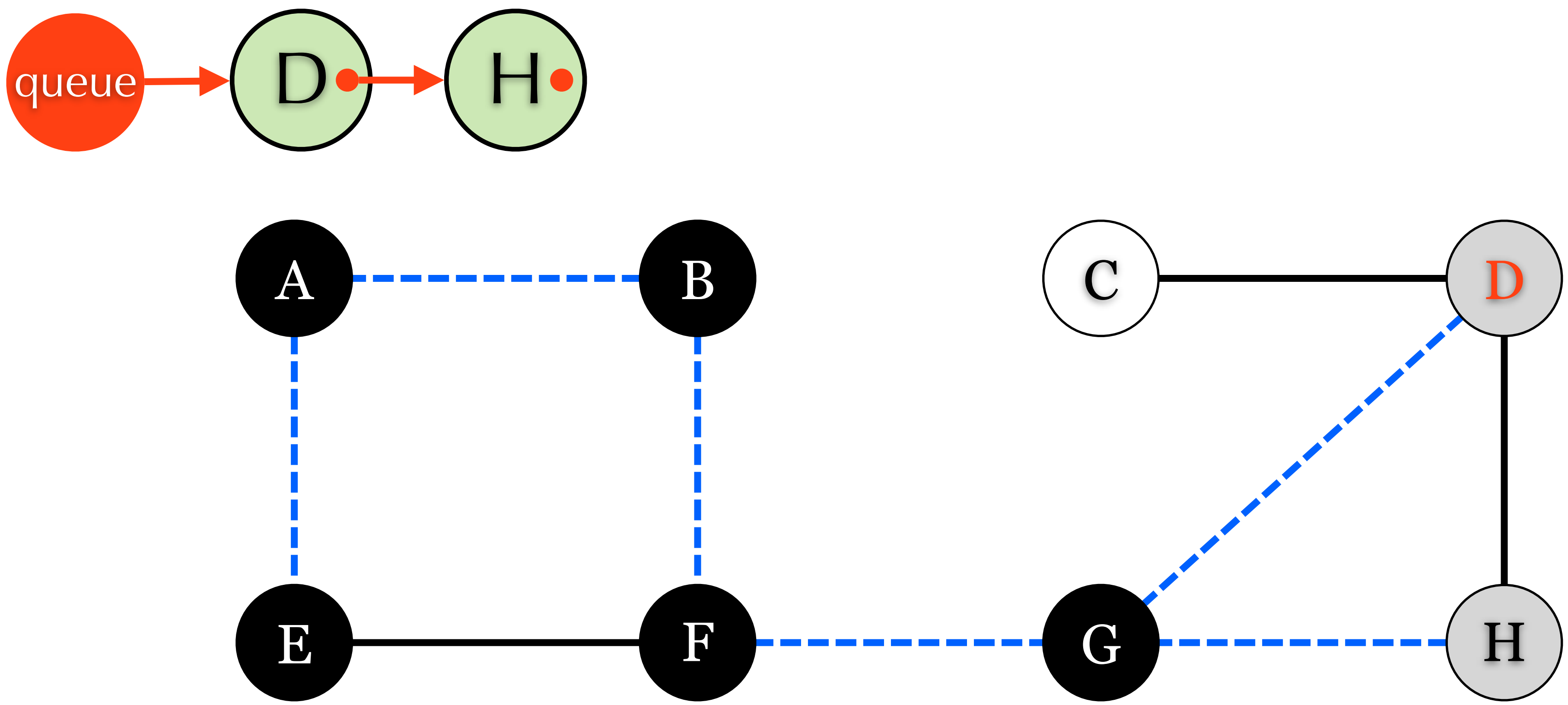
v	A	B	C	D	E	F	G	H
π	B	NIL	NIL	NIL	A	B	F	NIL
d	1	0	∞	∞	2	1	2	∞

Breadth First Search



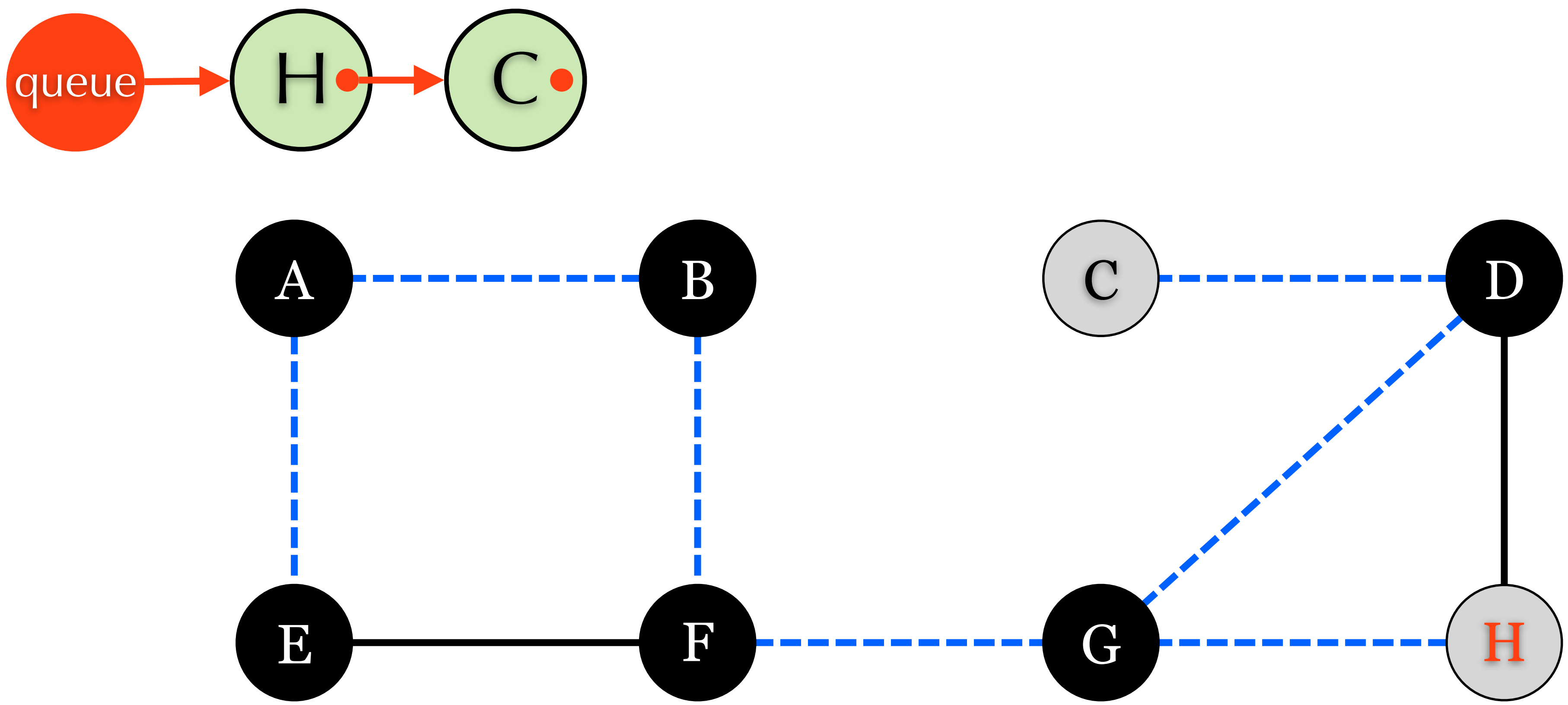
v	A	B	C	D	E	F	G	H
π	B	NIL	NIL	NIL	A	B	F	NIL
d	1	0	∞	∞	2	1	2	∞

Breadth First Search



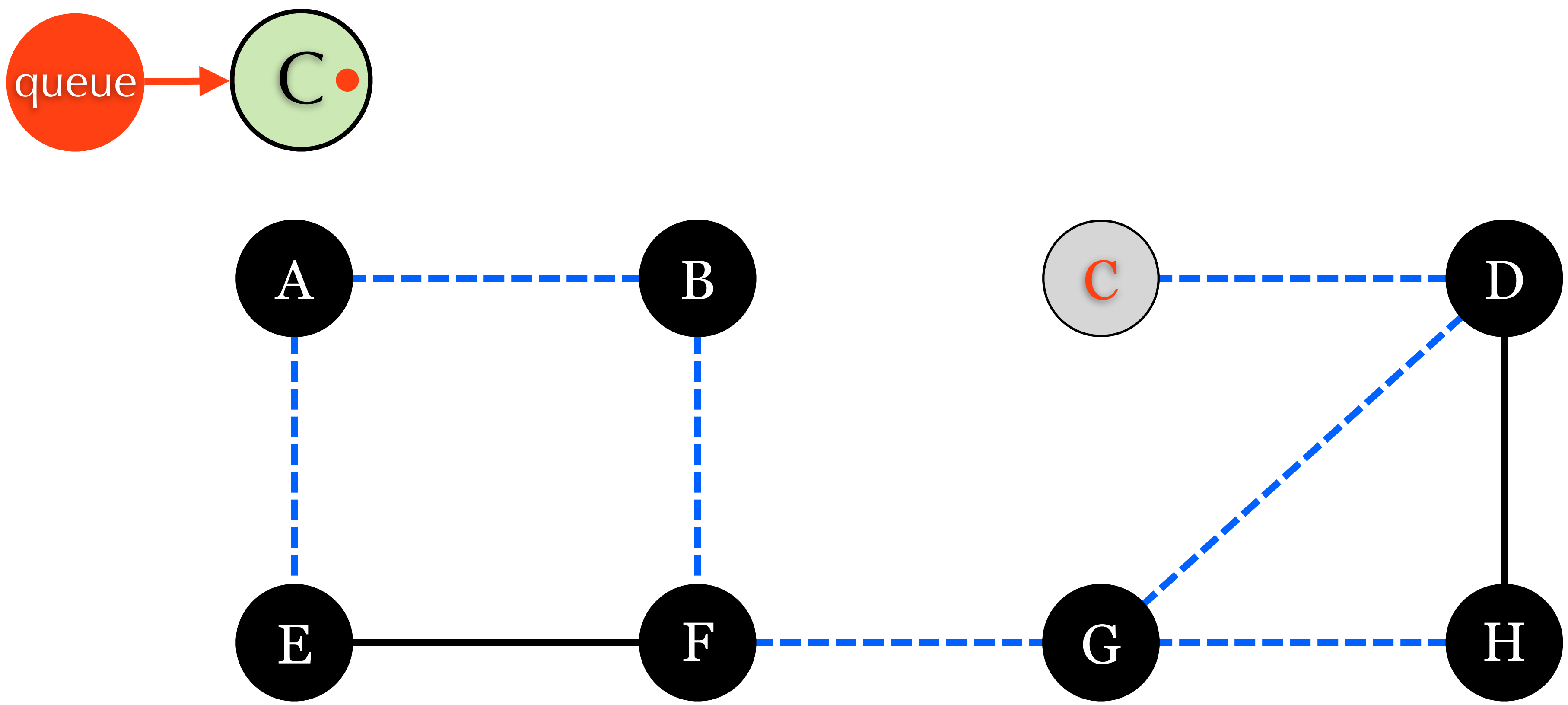
v	A	B	C	D	E	F	G	H
π	B	NIL	NIL	G	A	B	F	G
d	1	0	∞	3	2	1	2	3

Breadth First Search



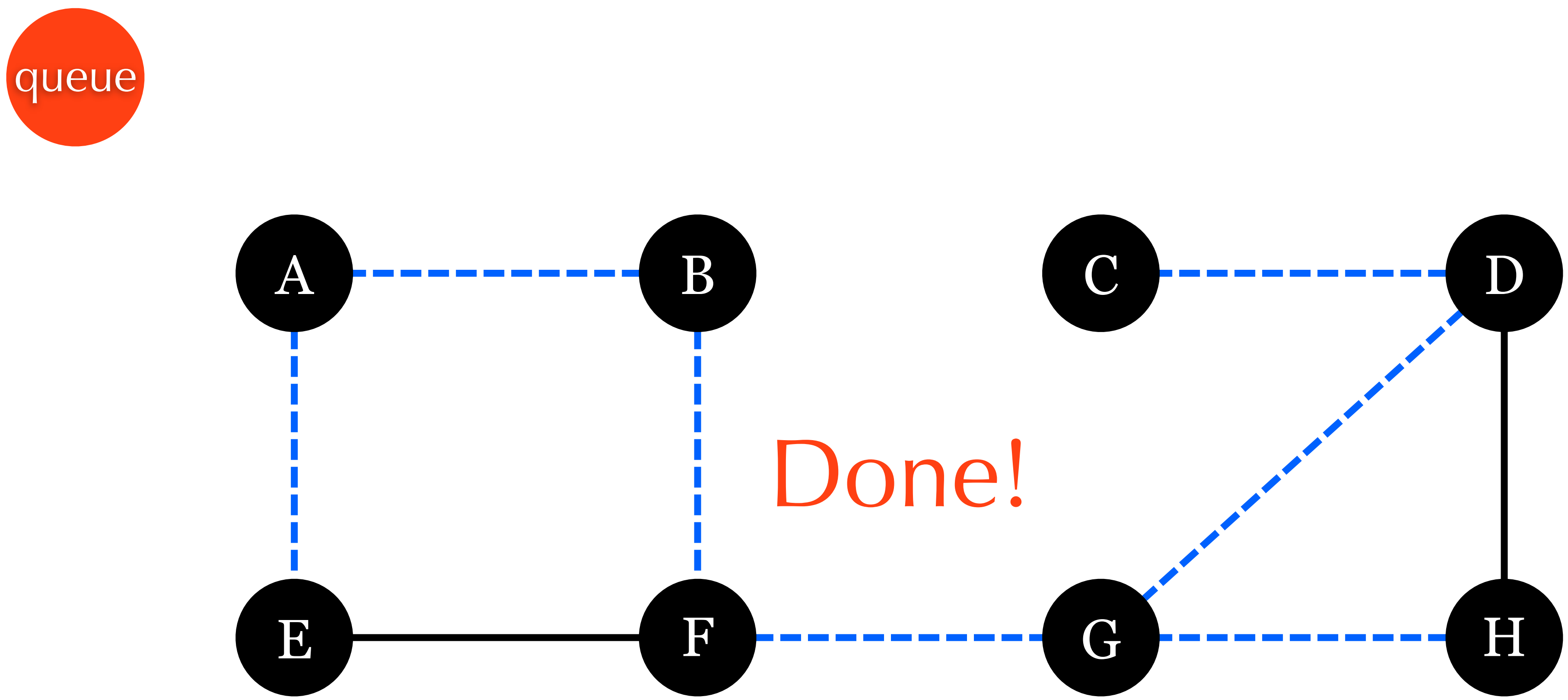
v	A	B	C	D	E	F	G	H
π	B	NIL	D	G	A	B	F	G
d	1	0	4	3	2	1	2	3

Breadth First Search



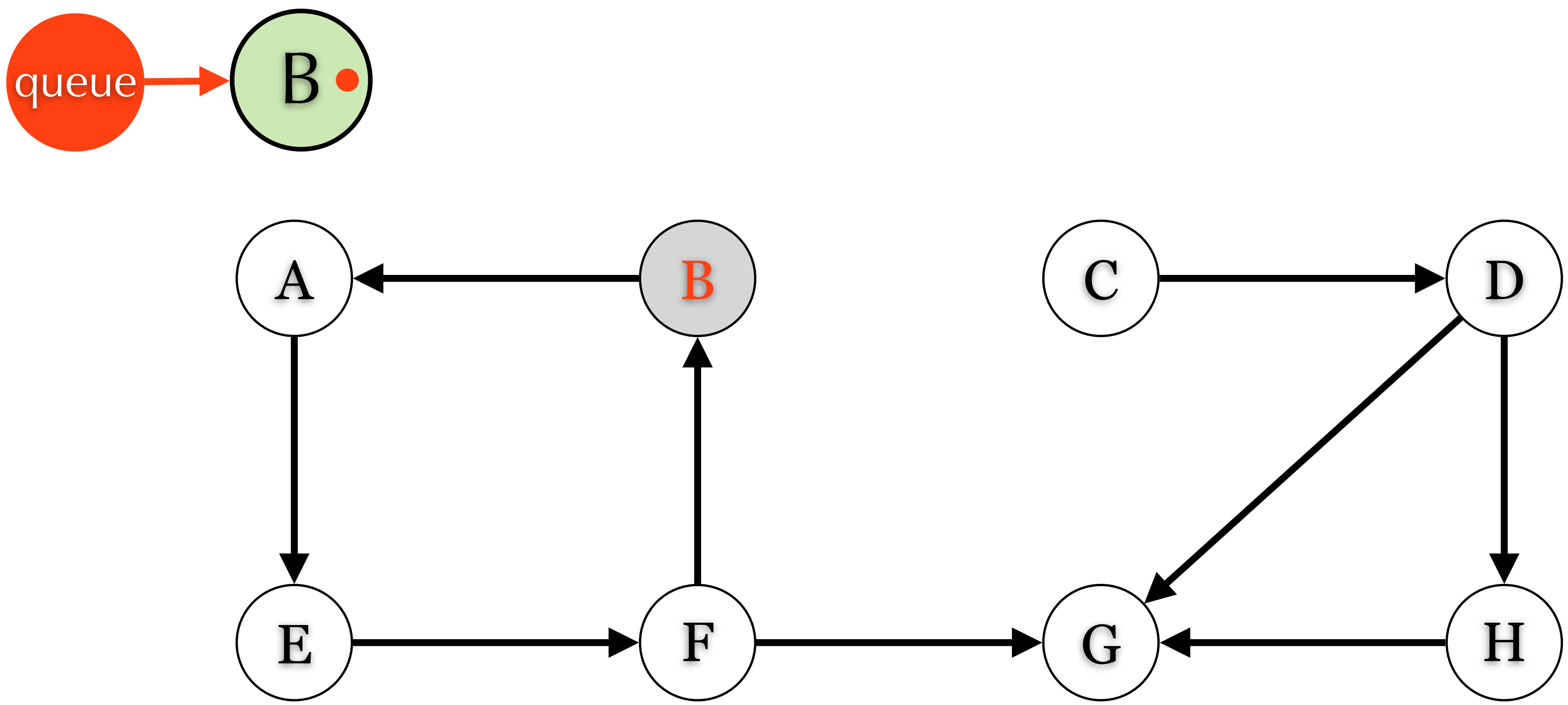
v	A	B	C	D	E	F	G	H
π	B	NIL	D	G	A	B	F	G
d	1	0	4	3	2	1	2	3

Breadth First Search



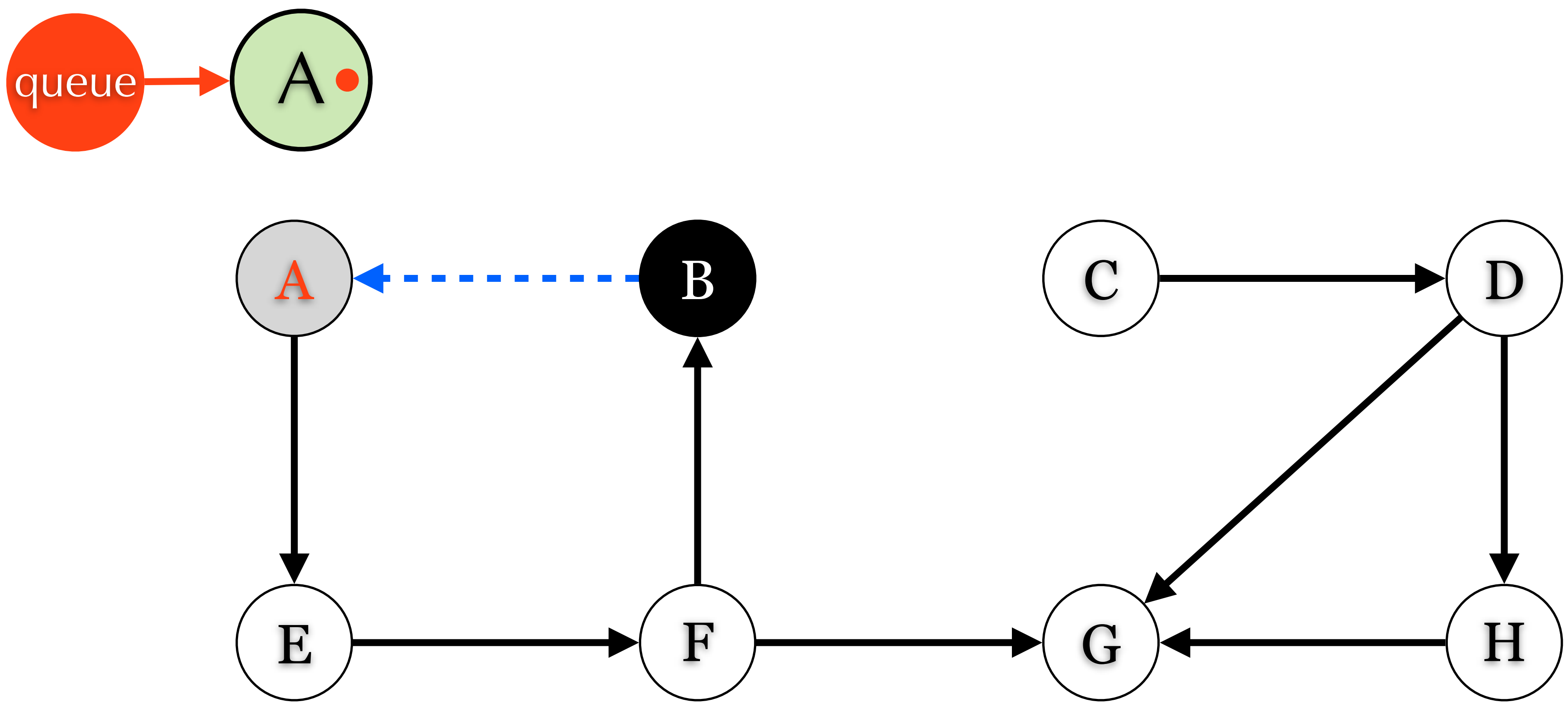
v	A	B	C	D	E	F	G	H
π	B	NIL	D	G	A	B	F	G
d	1	0	4	3	2	1	2	3

Breadth First Search



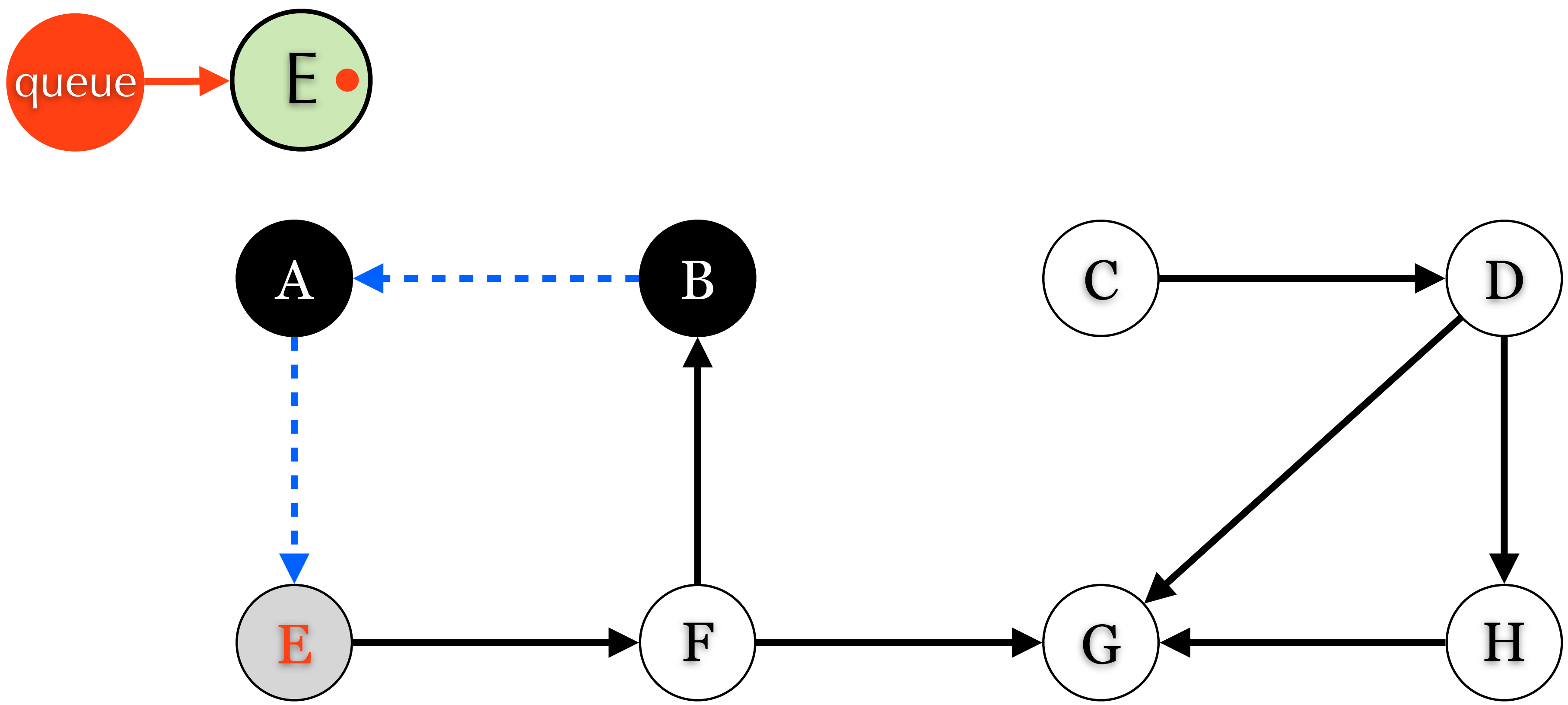
v	A	B	C	D	E	F	G	H
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL
d	∞	0	∞	∞	∞	∞	∞	∞

Breadth First Search



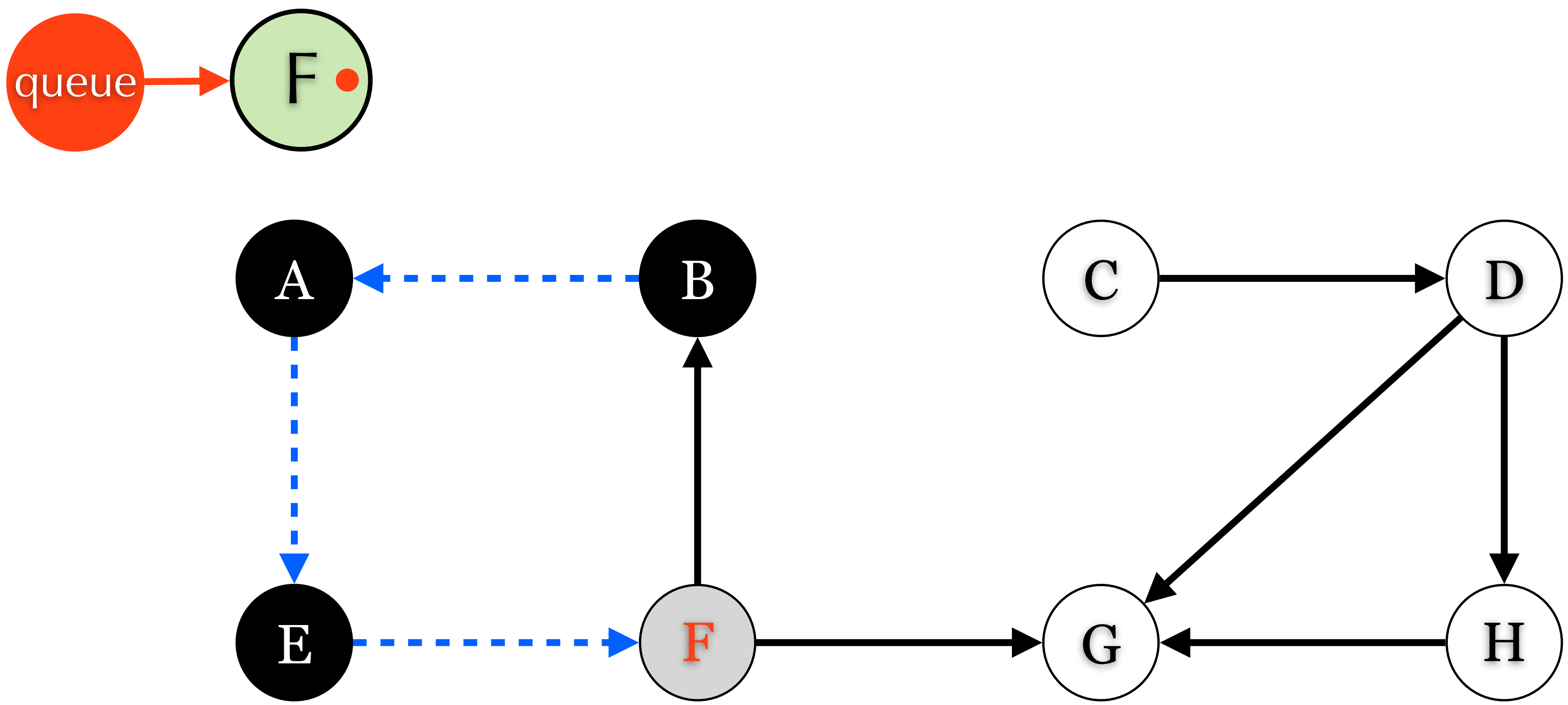
v	A	B	C	D	E	F	G	H
π	B	NIL	NIL	NIL	NIL	NIL	NIL	NIL
d	1	0	∞	∞	∞	∞	∞	∞

Breadth First Search



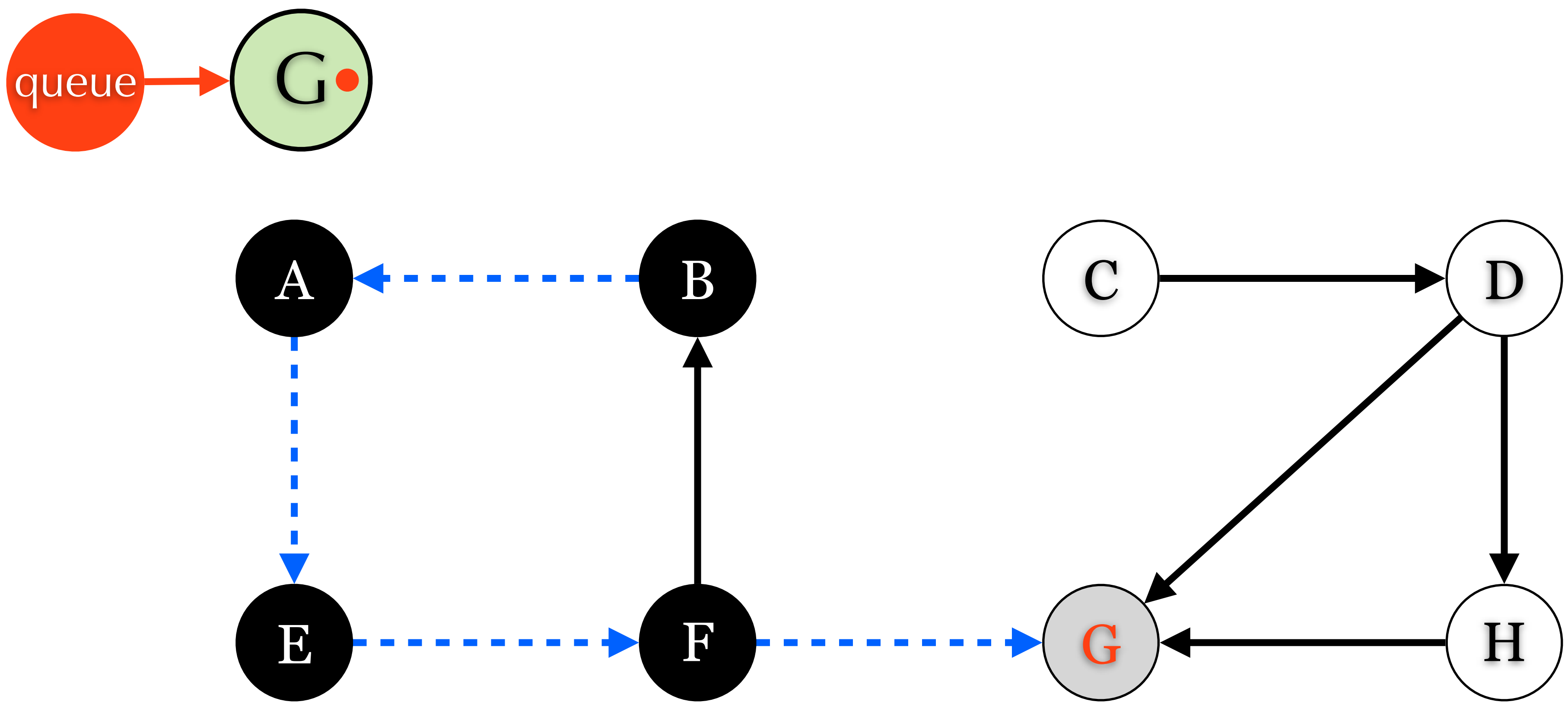
v	A	B	C	D	E	F	G	H
π	B	NIL	NIL	NIL	A	NIL	NIL	NIL
d	1	0	∞	∞	2	∞	∞	∞

Breadth First Search



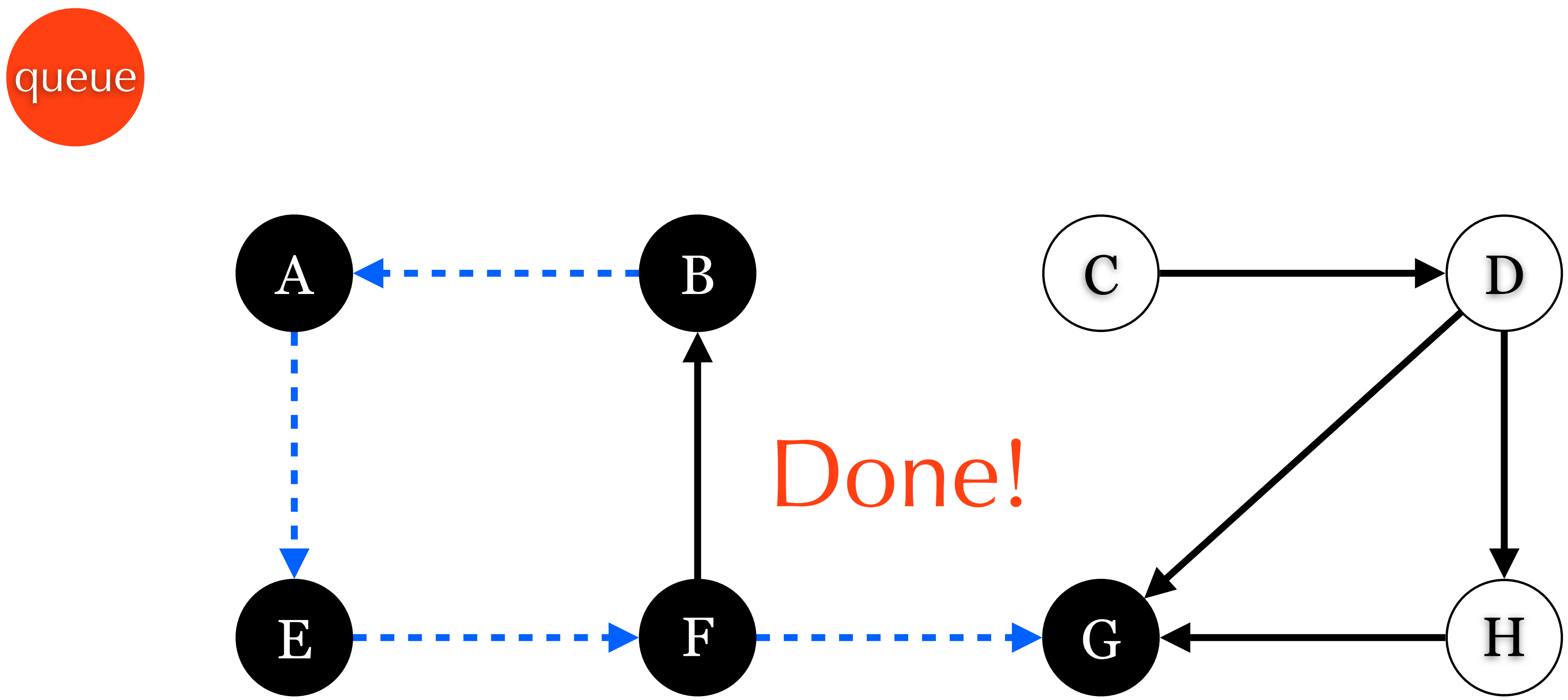
v	A	B	C	D	E	F	G	H
π	B	NIL	NIL	NIL	A	E	NIL	NIL
d	1	0	∞	∞	2	3	∞	∞

Breadth First Search



v	A	B	C	D	E	F	G	H
π	B	NIL	NIL	NIL	A	E	F	NIL
d	1	0	∞	∞	2	3	4	∞

Breadth First Search



v	A	B	C	D	E	F	G	H
π	B	NIL	NIL	NIL	A	E	F	NIL
d	1	0	∞	∞	2	3	4	∞

Time Complexity

- Adjacency list $O(|V|+|E|)$
  - Initialization: $O(|V|)$
  - Main loop: $O(|E|)$
    - Outer: $O(|V|)$
    - Inner: $O(d)$
- Adjacency array $O(|V|^2)$
  - Initialization: $O(|V|)$
  - Main loop: $O(|V|^2)$
    - Outer: $O(|V|)$
    - Inner: $O(|V|)$

BFS Property

- $\delta(u,v)$: the minimum #edges from u to v
- Lemma 22.1:
  - $\delta(s,v) \le \delta(s,u)+1$ for every $(u,v) \in E$.
- Proof
  - u is reachable from s: Assume the shortest path from s to v is $(s,u_1),\ldots,(u_{k-1},u_k),(u_k,u)$. Then $(s,u_1),\ldots,(u_{k-1},u_k),(u_k,u),(u,v)$ is a path from s to v and its length is $\delta(s,u)+1$.
  - If u is unreachable, then the inequality is always true, since $\delta(s,u)=\infty$.

Lemma 22.2

- Suppose v.d is computed by BFS running on G from s. We have $v.d \geq \delta(s,v)$ for $v \in V$.
- Proof: By induction on #Enqueue
  - Basis: $s.d=0=\delta(s,s)$ and $v.d=\infty \geq \delta(s,v)$ for $v \neq s$.
  - Inductive hypothesis $v.d \geq \delta(s,v)$ after the k-th Enqueue.
  - Inductive step: (k+1)-th enqueue triggered by edge (u,v). $v.d=u.d+1 \geq \delta(s,u)+1 \geq \delta(s,v)$

Lemma 22.3

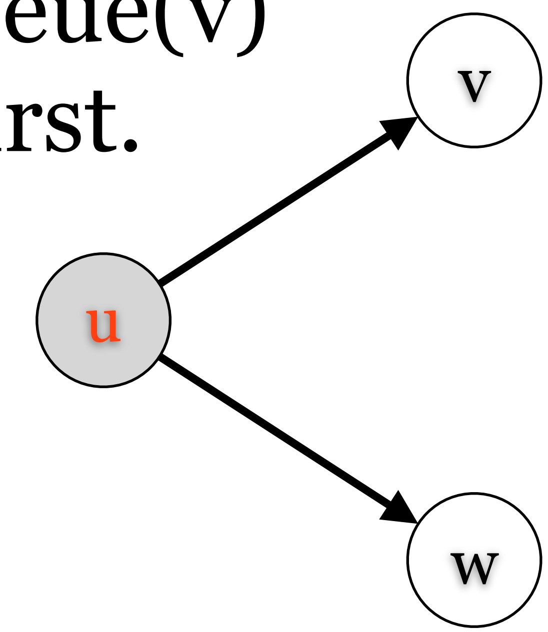
- Suppose that during the execution of BFS on a graph $G=(V,E)$, the queue Q contains the vertices $\langle v_1,\ldots,v_r\rangle$ where v_1 is the head of Q and v_r is the tail. Then, $v_r.d \le v_1.d+1$ and $v_i.d \le v_{i+1}.d$ for $1 \le i < r$.
- Proof: by induction on #operation
- Basis: Initially $Q=\langle s\rangle$, $s.d \le s.d+1$.
- Induction hypothesis: Lemma 22.2 is true for <k operations.

Proof: Lemma 22.3

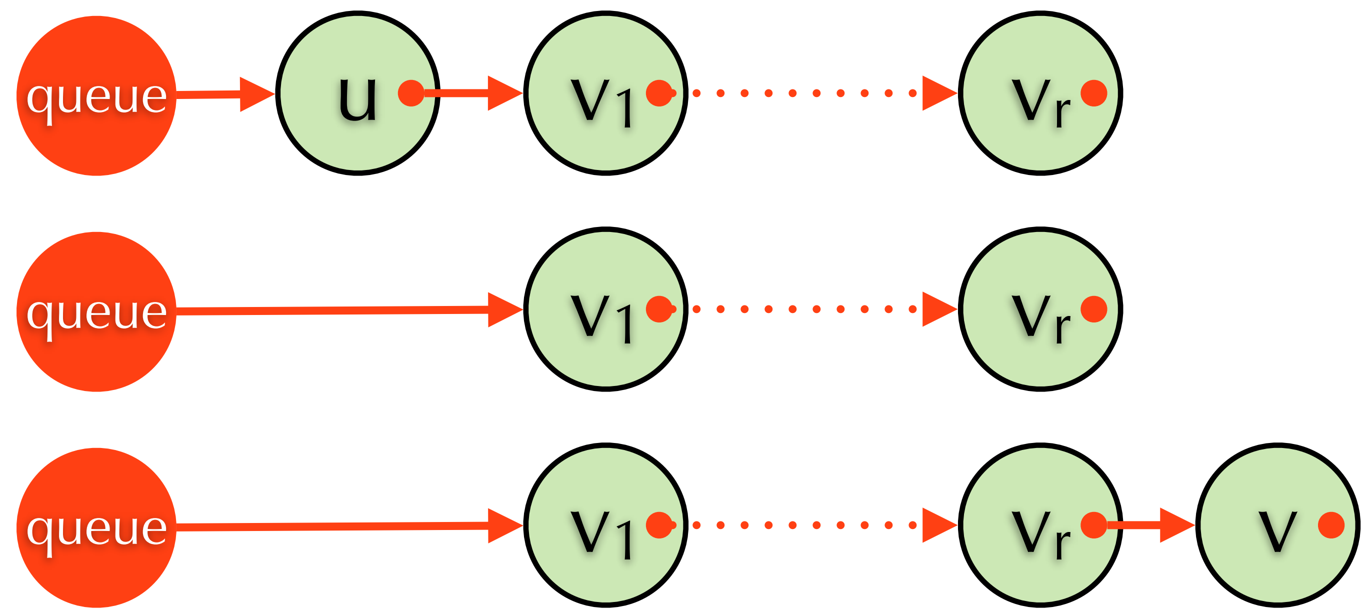
- Inductive step: the k-th operation
  - Queue $\langle v_1,\ldots,v_r\rangle$
  - Dequeue: Since $v_1.d \le v_2.d$ and $v_1.d+1 \ge v_r.d$, we have $v_2.d+1 \ge v_1.d+1 \ge v_r.d$. v_2: new head
  - Enqueue triggered by (u,v):
    - $v.d=u.d+1 \le v_1.d+1$ tail.d≤head.d+1
    - $v_i.d$ is increasing:
      - Q.enqueue(v) is the first enqueue after u=Q.dequeue: $v_r.d \le u.d+1=v.d$
      - Otherwise: $v_r.d=u.d+1=v.d$

Proof: Lemma 22.3

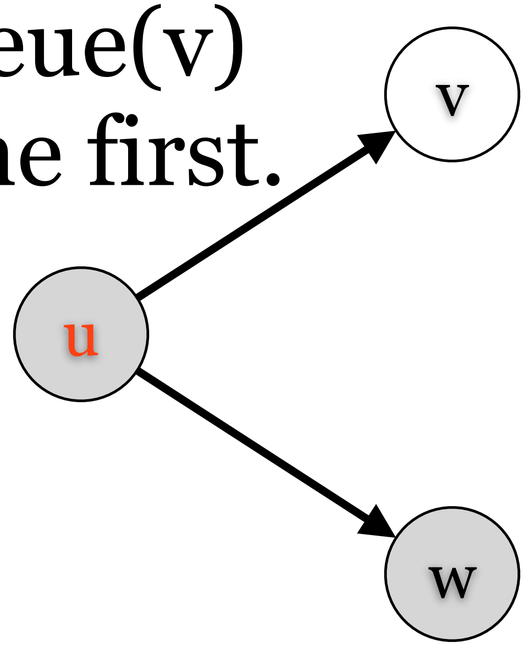
Q.enqueue(v) is the first.

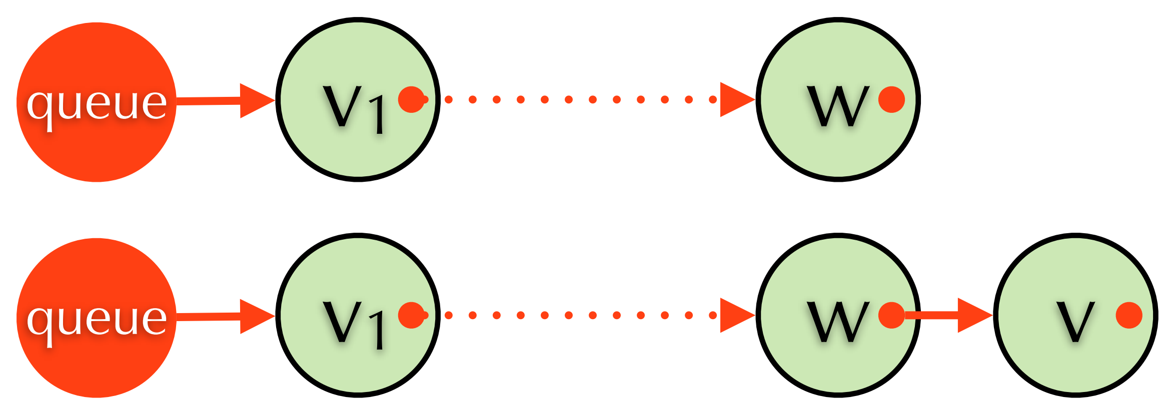


$v_r.d \le u.d+1 = v.d$



Q.enqueue(v) is not the first.





$v_r.d = w.d = u.d+1 = v.d$

Corollary 22.4

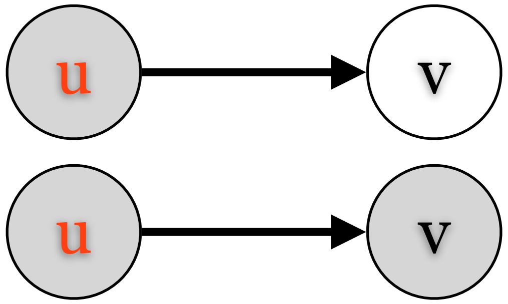
- Suppose that vertices v_i and v_j are enqueued during the execution of BFS, and that v_i is enqueued before v_j. Then $v_i.d \leq v_j.d$ at the time that v_j is enqueued.
- Proof:
  - By lemma 22.3, and $v_j.d$ is set to a finite value only once.

Theorem 22.5

- Suppose v.d is computed by BFS running on G from s. We have $v.d=\delta(s,v)$ for $v \in V$.
- Proof: BWOC, assume $v.d>\delta(s,v)$ and $\delta(s,v)$ is minimum. Lemma 22.2
- $v \neq s$ and v is reachable, so $\delta(s,v) \geq 1$.
- There exists (u,v) s.t. $\delta(s,u)+1=\delta(s,v)$.
- $\delta(s,u)<\delta(s,v)$ implies $u.d=\delta(s,u)<\delta(s,v)<v.d$.
- u is enqueued before v. Corollary 22.4

Proof: Theorem 22.5

- When u is dequeued, there are two cases
  - v is not in Q, BFS set v.d=u.d+1.
  - v is in Q, BFS set v.d≤u.d+1
    Lemma 22.3

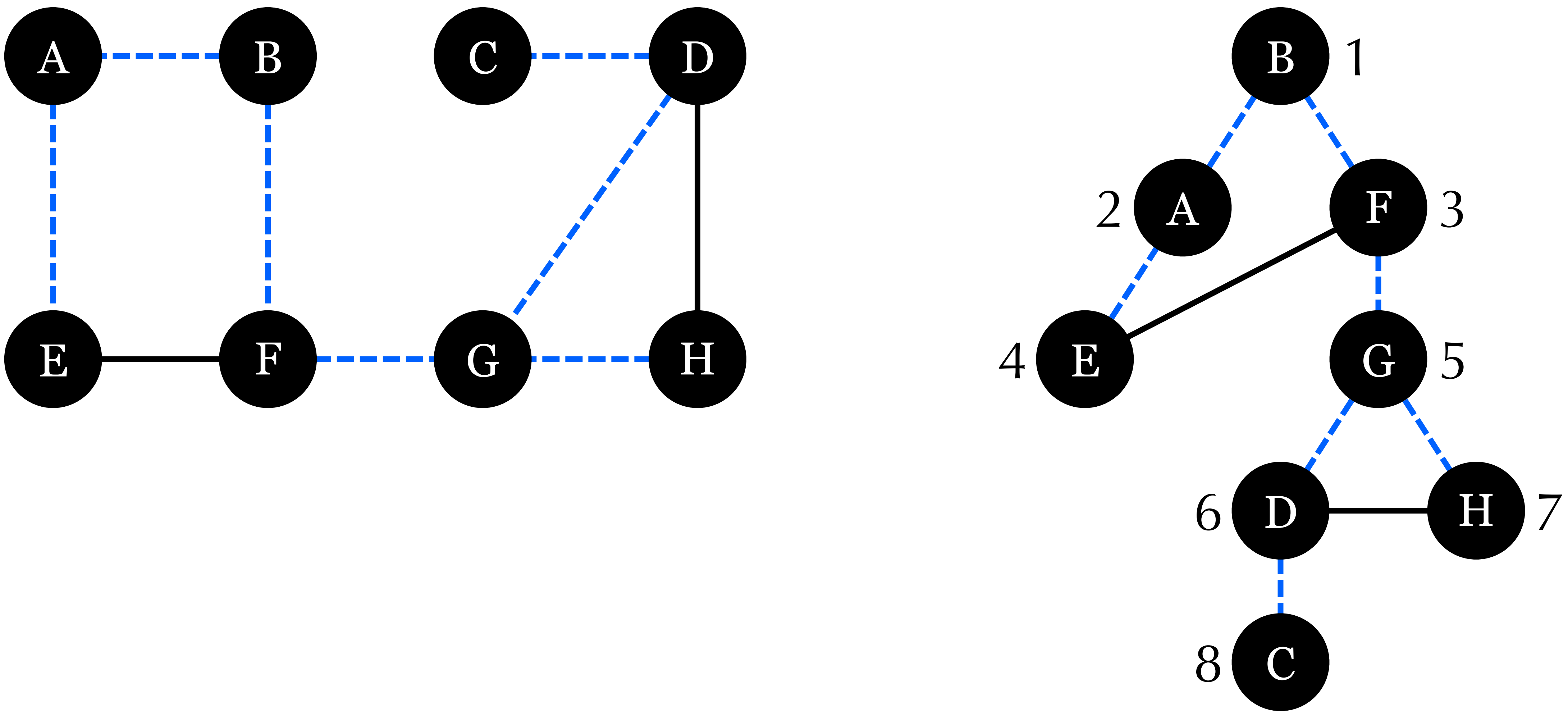


- v.d≤u.d+1=δ(s,u)+1=δ(s,v), a contradiction.

BFS Tree

- Predecessor subgraph $G_\pi=(V_\pi,E_\pi)$
  - $V_\pi=\{s\}\cup\{v:v.\pi\neq NIL\}$
  - $E_\pi=\{(v.\pi,v):v\in V_\pi\setminus\{s\}\}$
- If $v.\pi$ are obtained by running BFS on vertex s, then G_π is called BFS tree.
- The shortest path from s to v in G_π is also the shortest path from s to v in G.

BFS Tree



Depth First Search

- Given a graph G=(V,E) and an optional source vertex $s \in V$.
- DFS computes the set of vertices reachable from s.
- DFS can detect the existence of cycles.
- DFS cannot directly compute the shortest distance!
- DFS needs a stack, however, you may use recursive function to implement DFS.

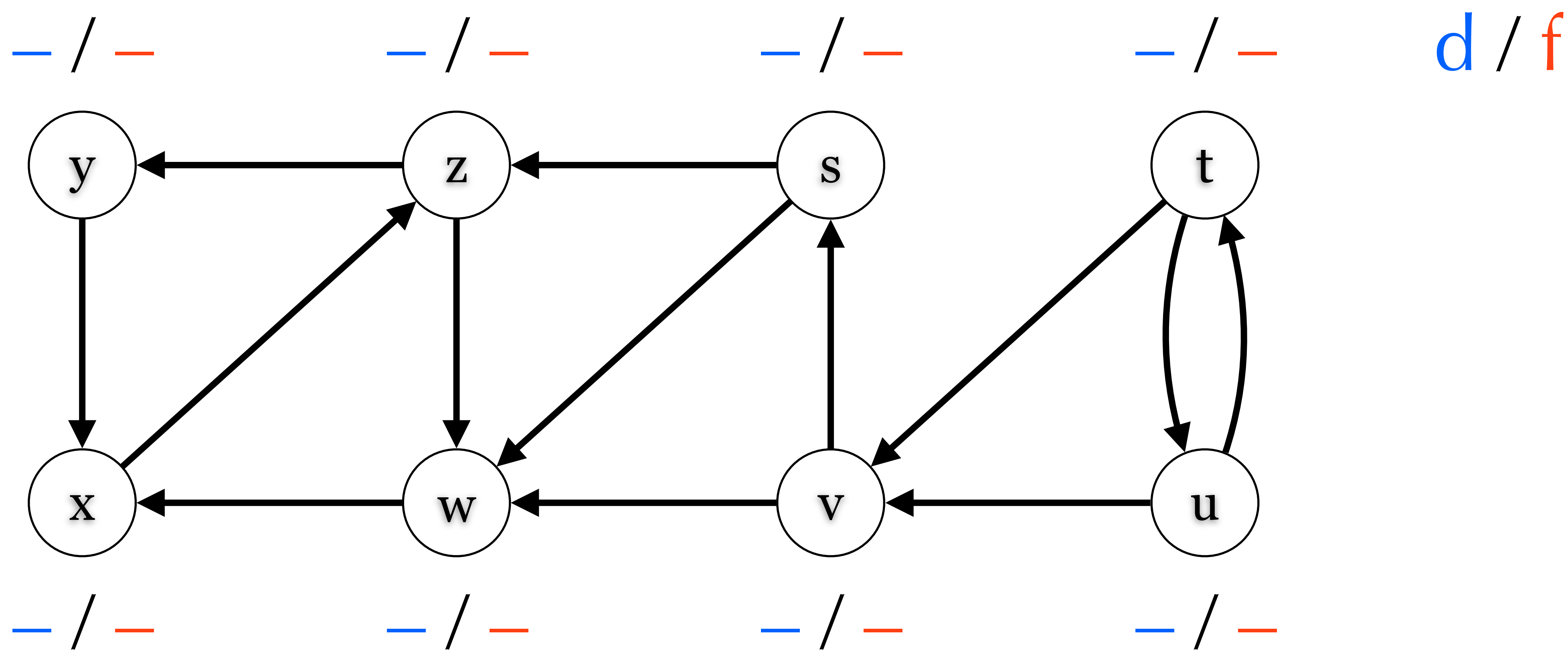
Depth First Search

- Initialization: For each $v \in V$
  - v.c=WHITE white color: not discovered
  - $v.\pi$=NIL predecessor
- Initialization: time=0
- Main Loop: For each $v \in V$
  - if v.c==WHITE
    DFS-Visit(v)

DFS-Visit(u)

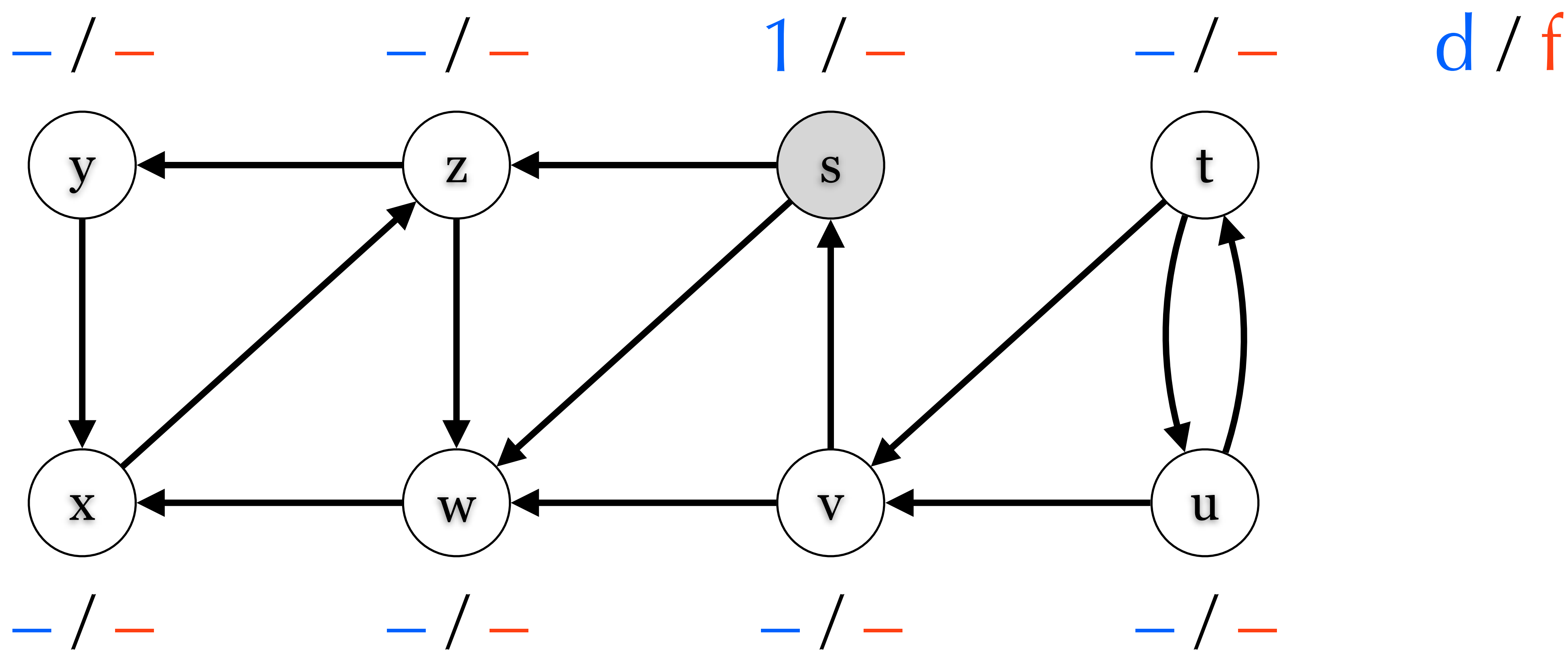
```
time = time +1
u.d = time              discover time
u.c = GRAY              mark u discovered
for each v s.t. (u,v)∈E
    if v.c==WHITE
        v.π=u
        DFS-Visit(v)
time = time+1
u.f = time              finish time
u.c = BLACK             mark u visited
```

Example



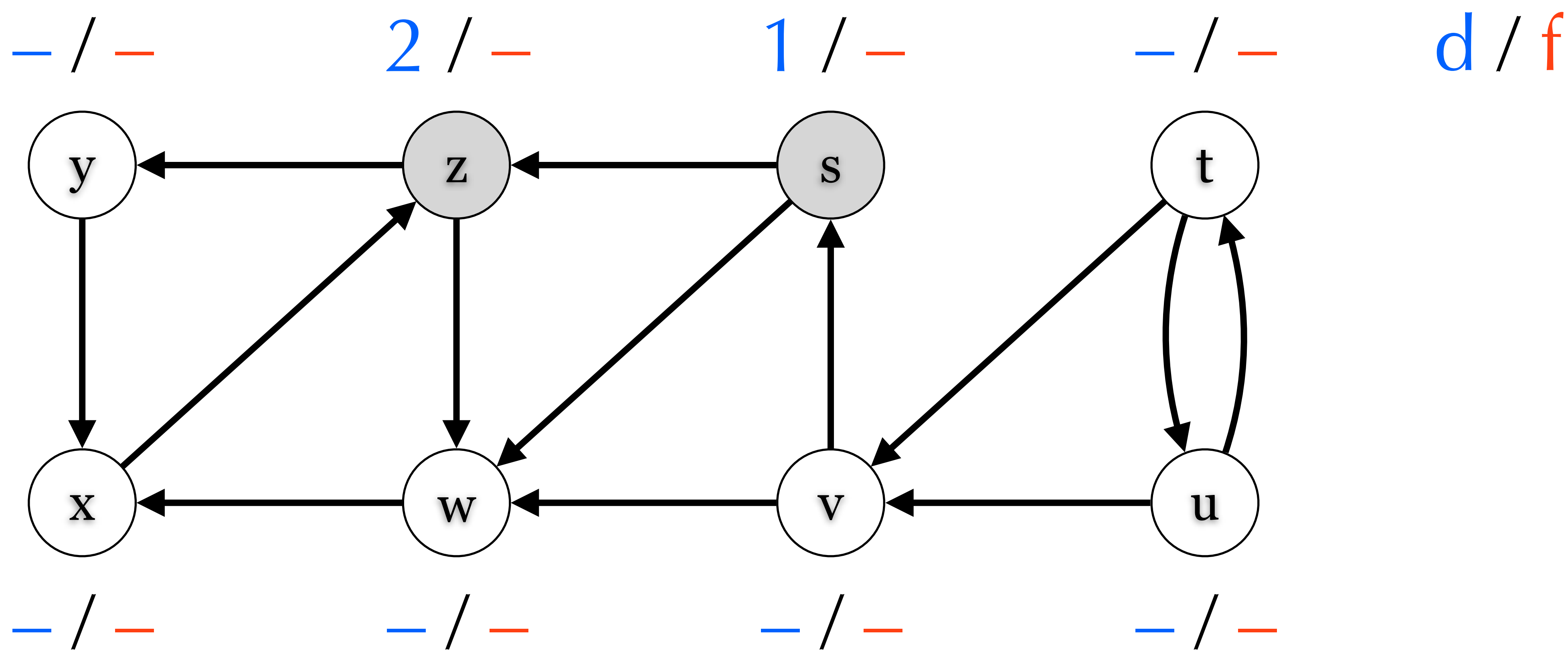
	s	t	u	v	w	x	y	z
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL

Example



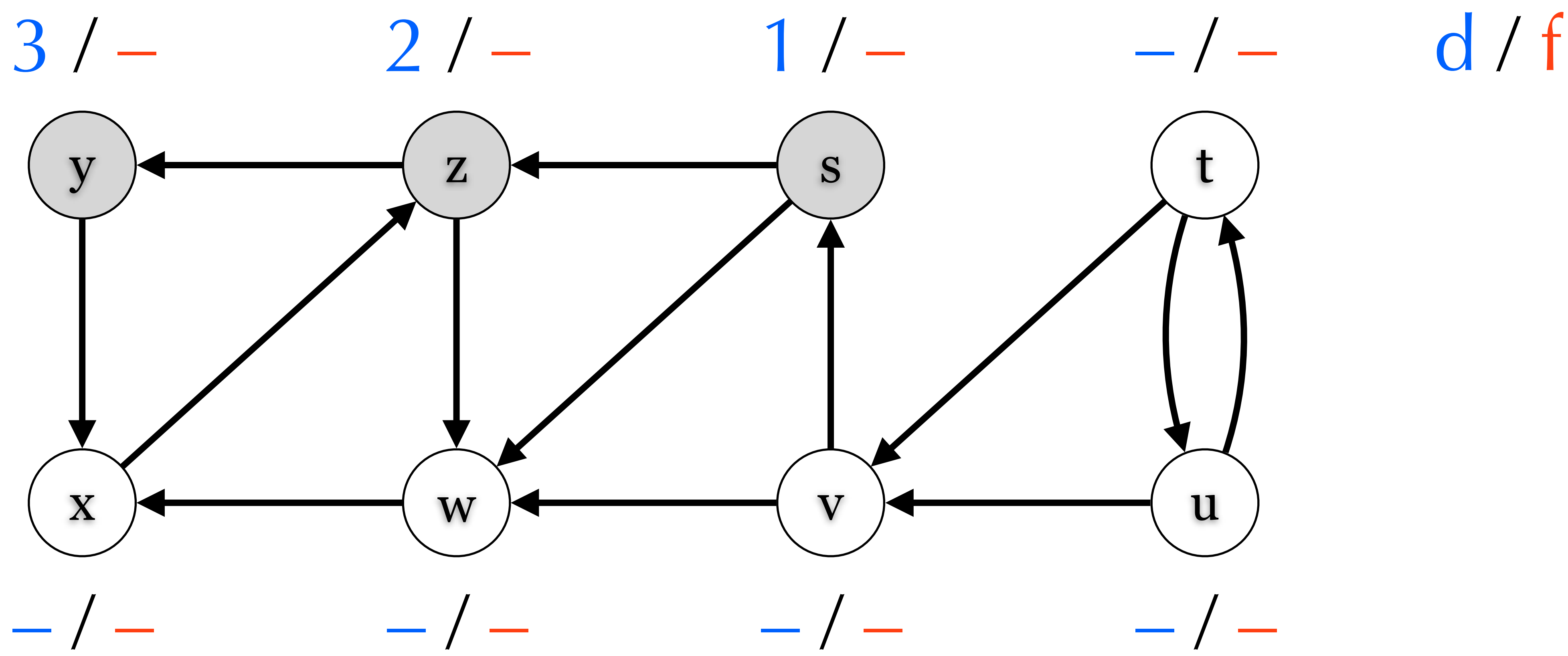
	s	t	u	v	w	x	y	z
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL

Example



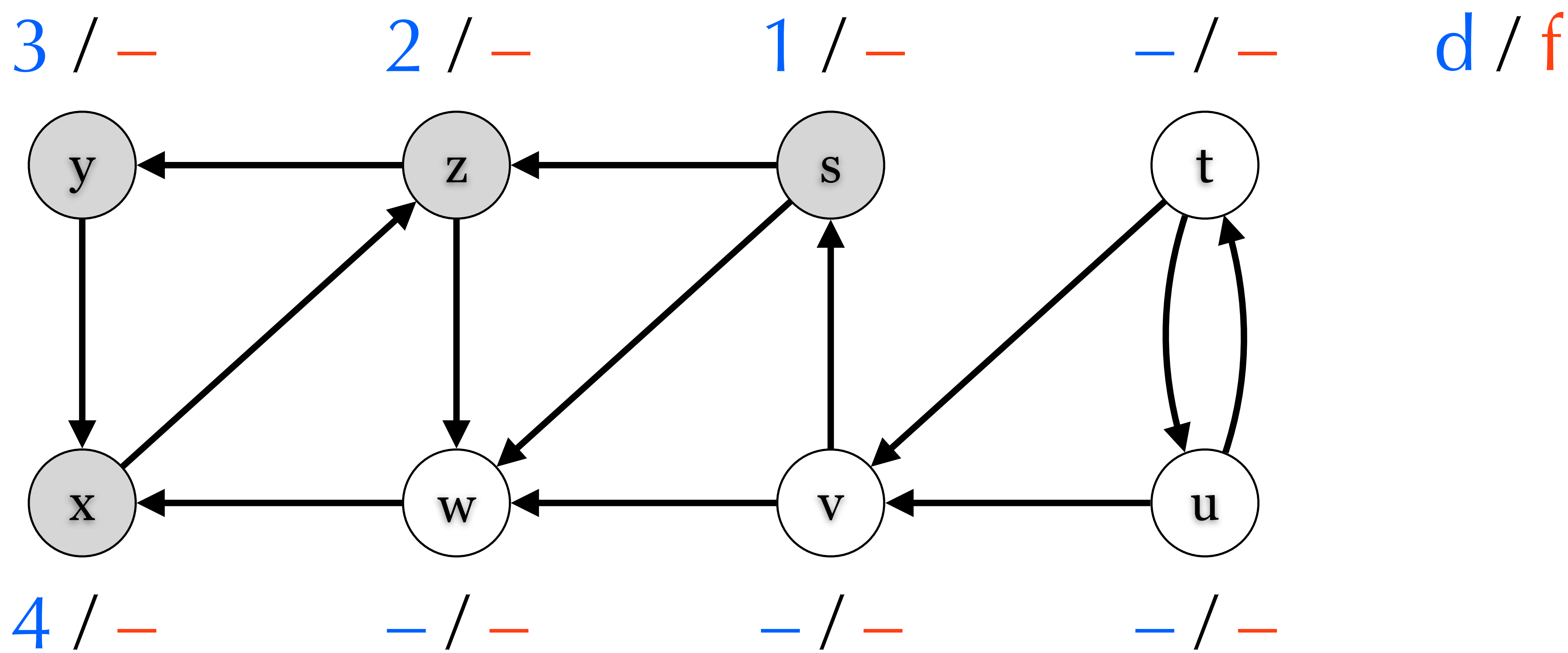
	s	t	u	v	w	x	y	z
π	NIL	NIL	NIL	NIL	NIL	NIL	NIL	s

Example



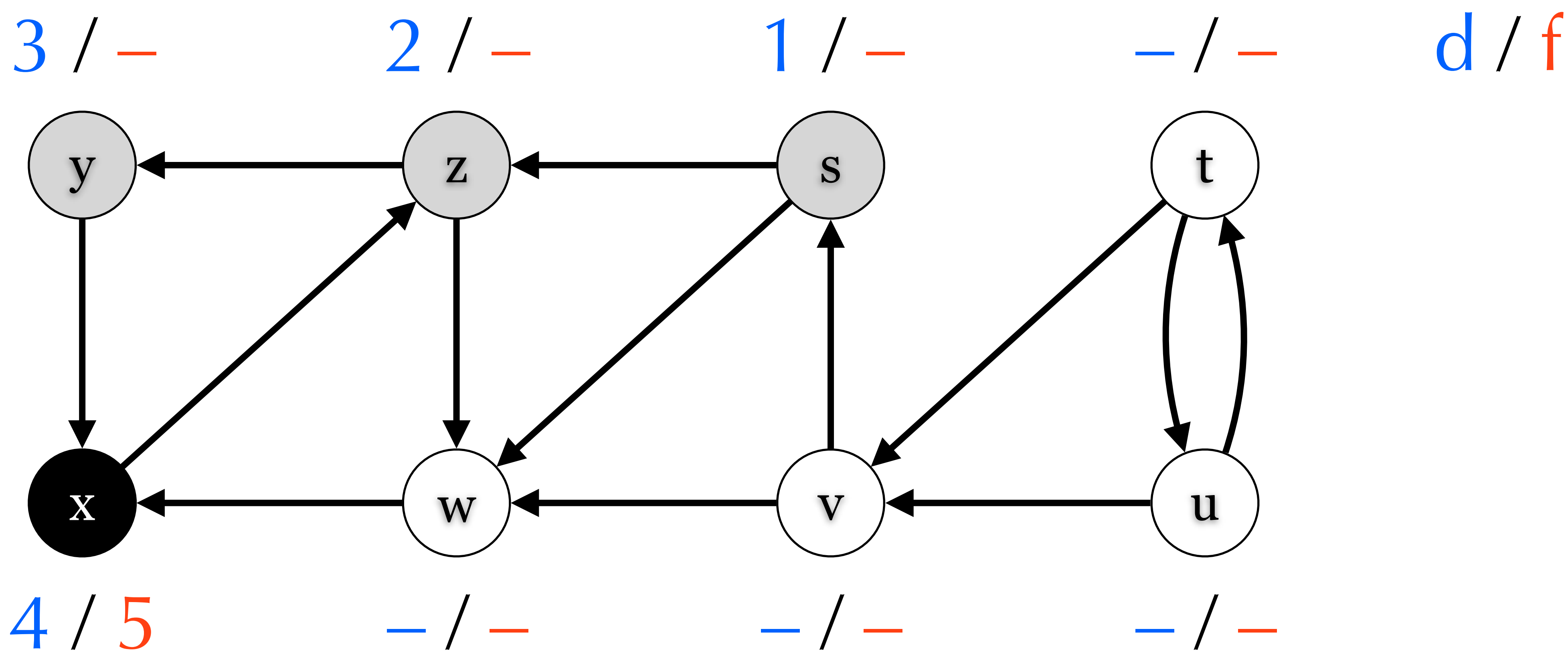
	s	t	u	v	w	x	y	z
π	NIL	NIL	NIL	NIL	NIL	NIL	z	s

Example



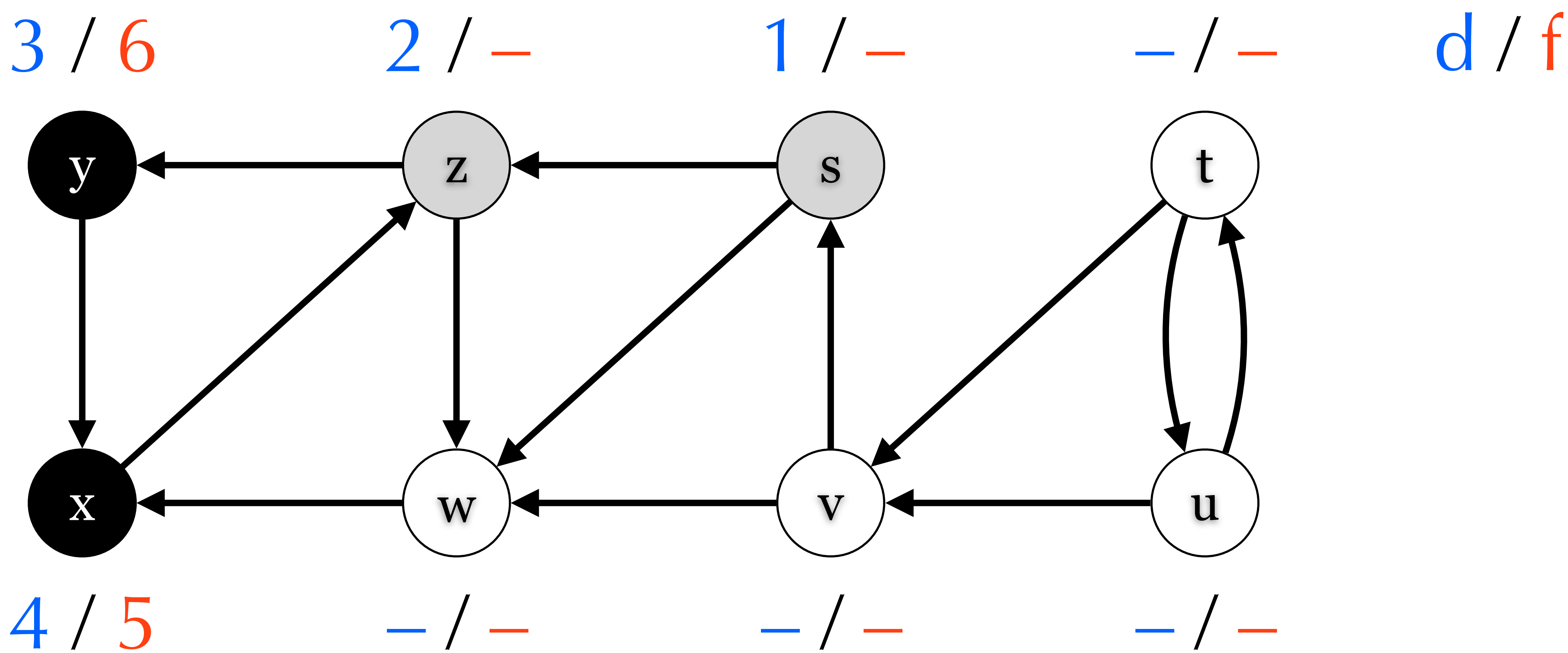
	s	t	u	v	w	x	y	z
π	NIL	NIL	NIL	NIL	NIL	y	z	s

Example



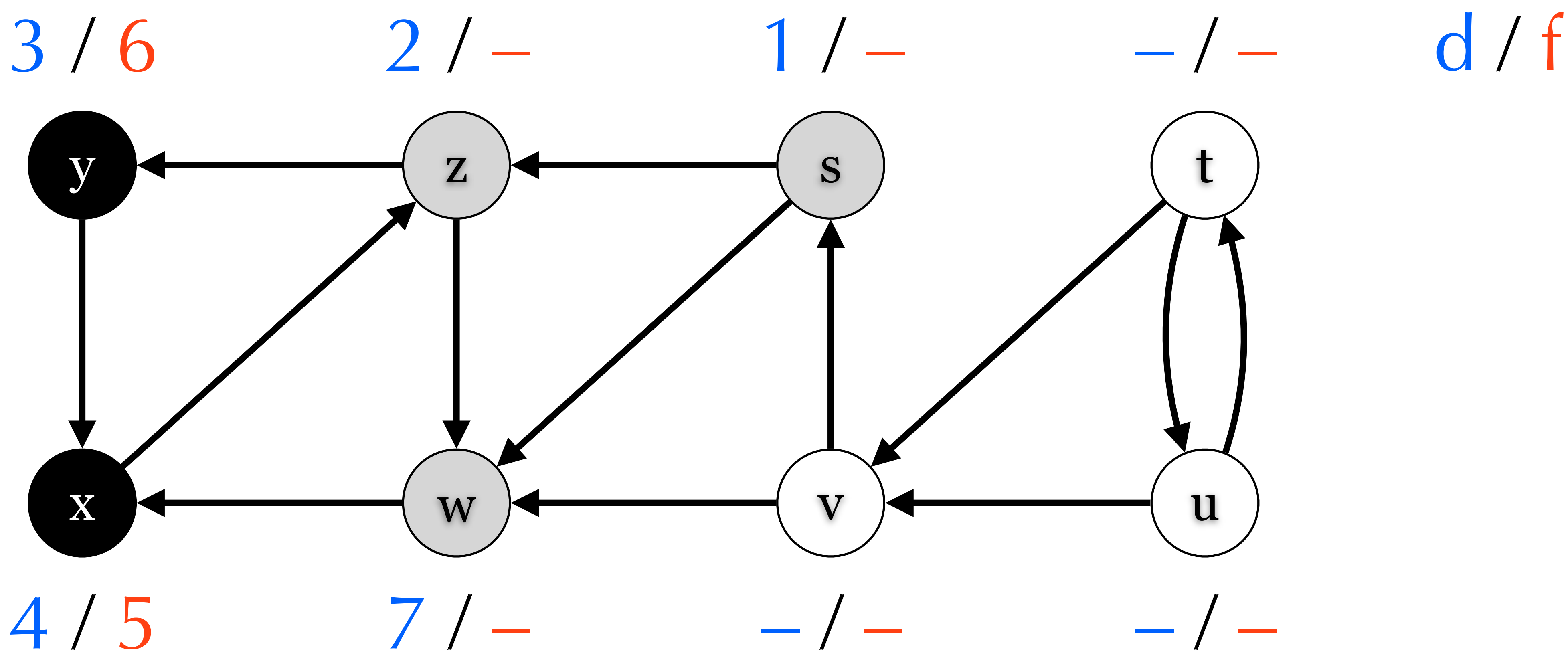
	s	t	u	v	w	x	y	z
π	NIL	NIL	NIL	NIL	NIL	y	z	s

Example



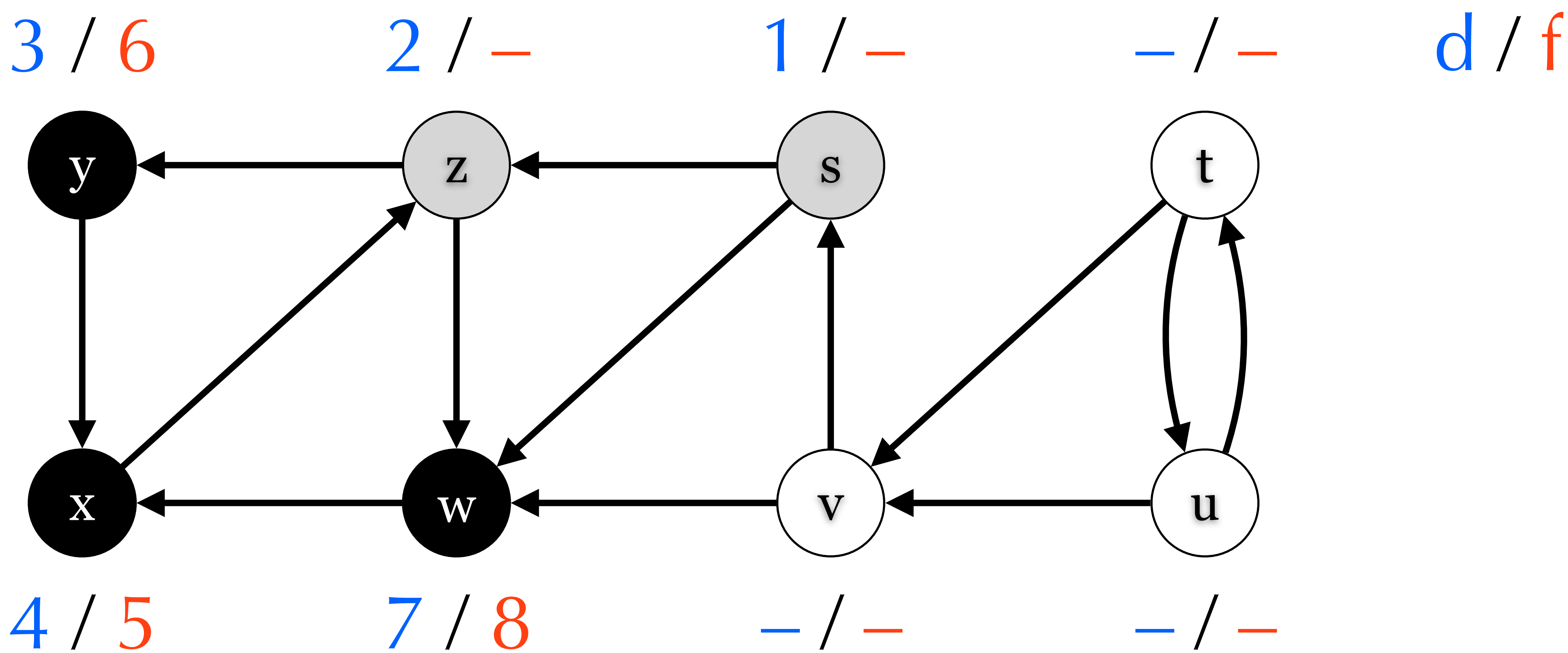
	s	t	u	v	w	x	y	z
π	NIL	NIL	NIL	NIL	NIL	y	z	s

Example



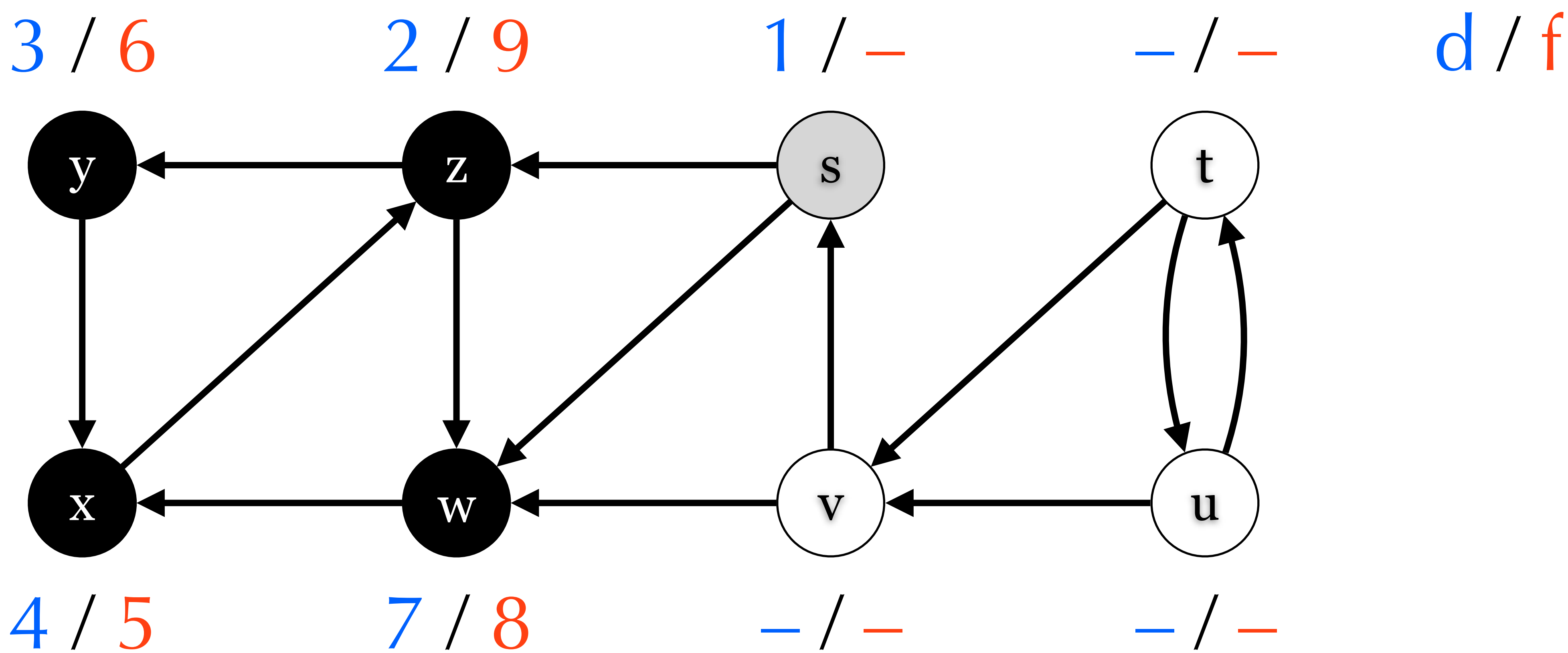
	s	t	u	v	w	x	y	z
π	NIL	NIL	NIL	NIL	z	y	z	s

Example



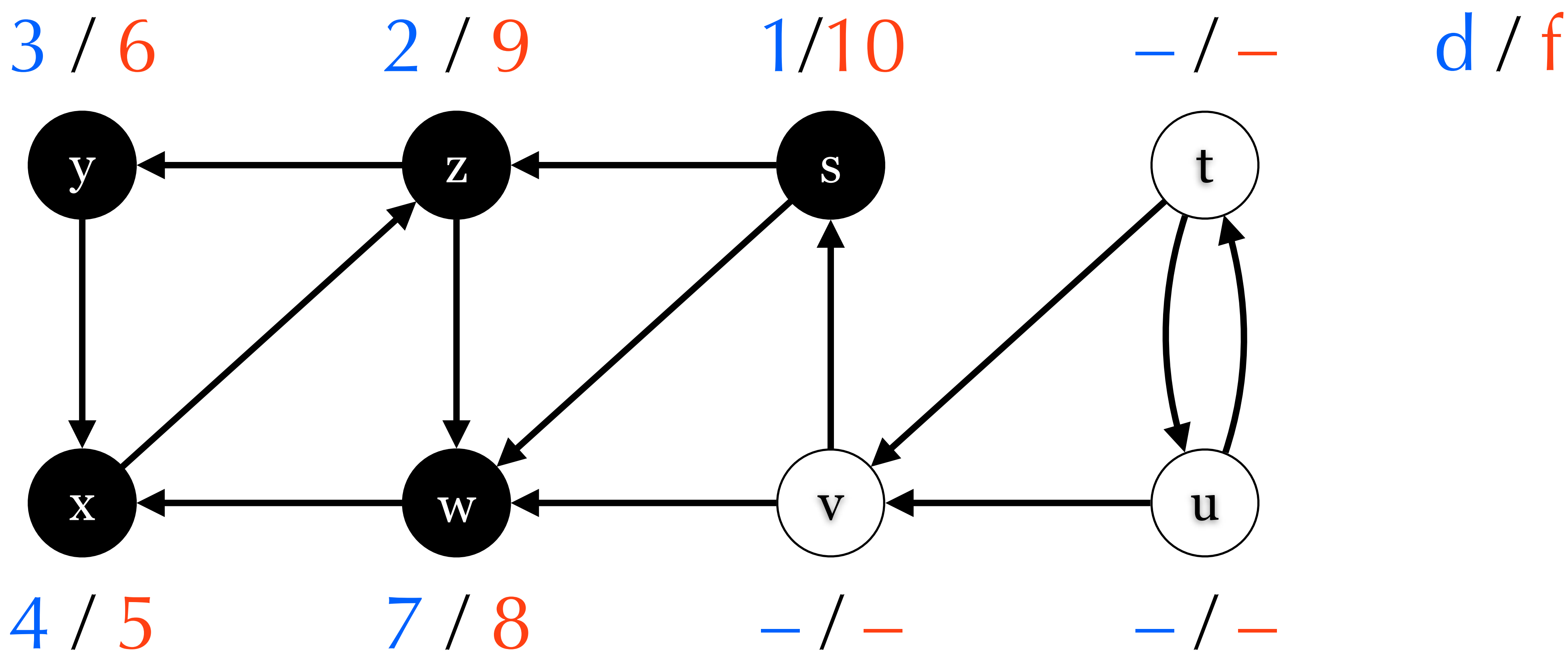
	s	t	u	v	w	x	y	z
π	NIL	NIL	NIL	NIL	z	y	z	s

Example



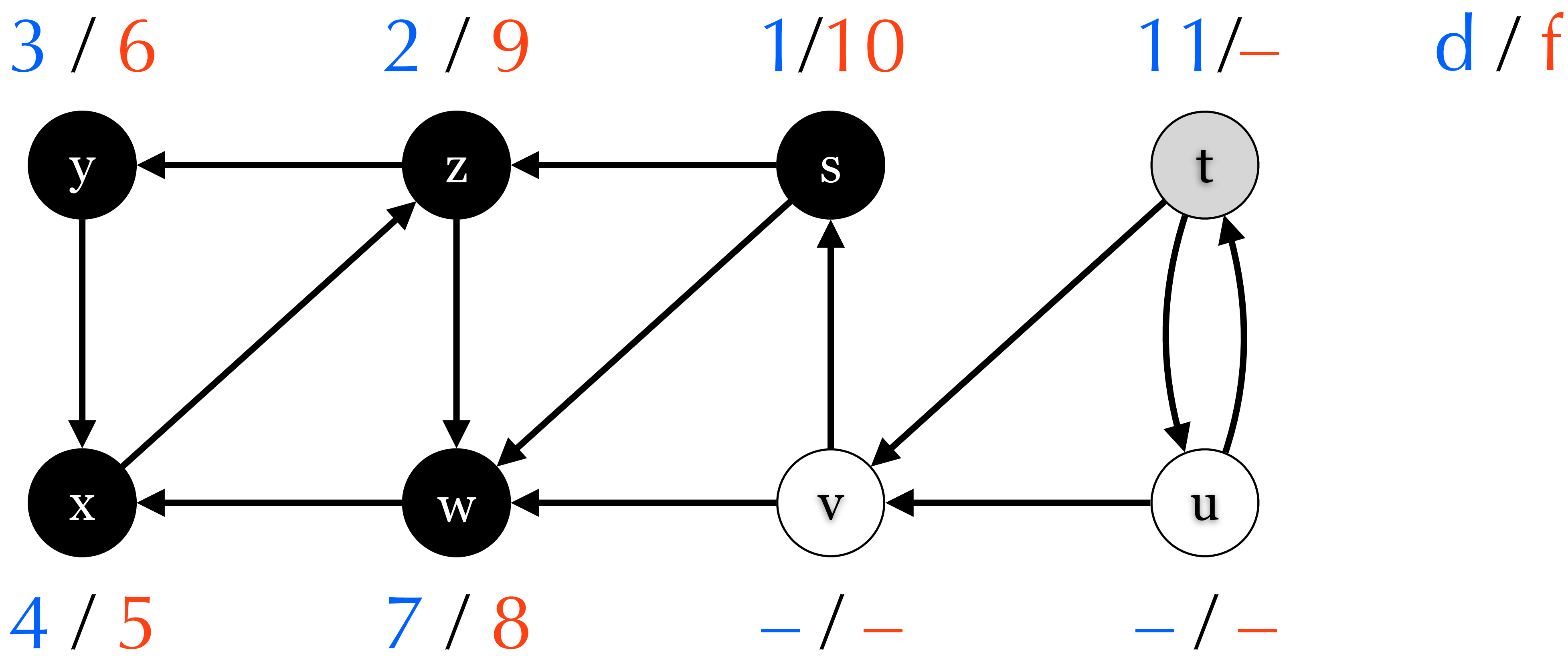
	s	t	u	v	w	x	y	z
π	NIL	NIL	NIL	NIL	z	y	z	s

Example



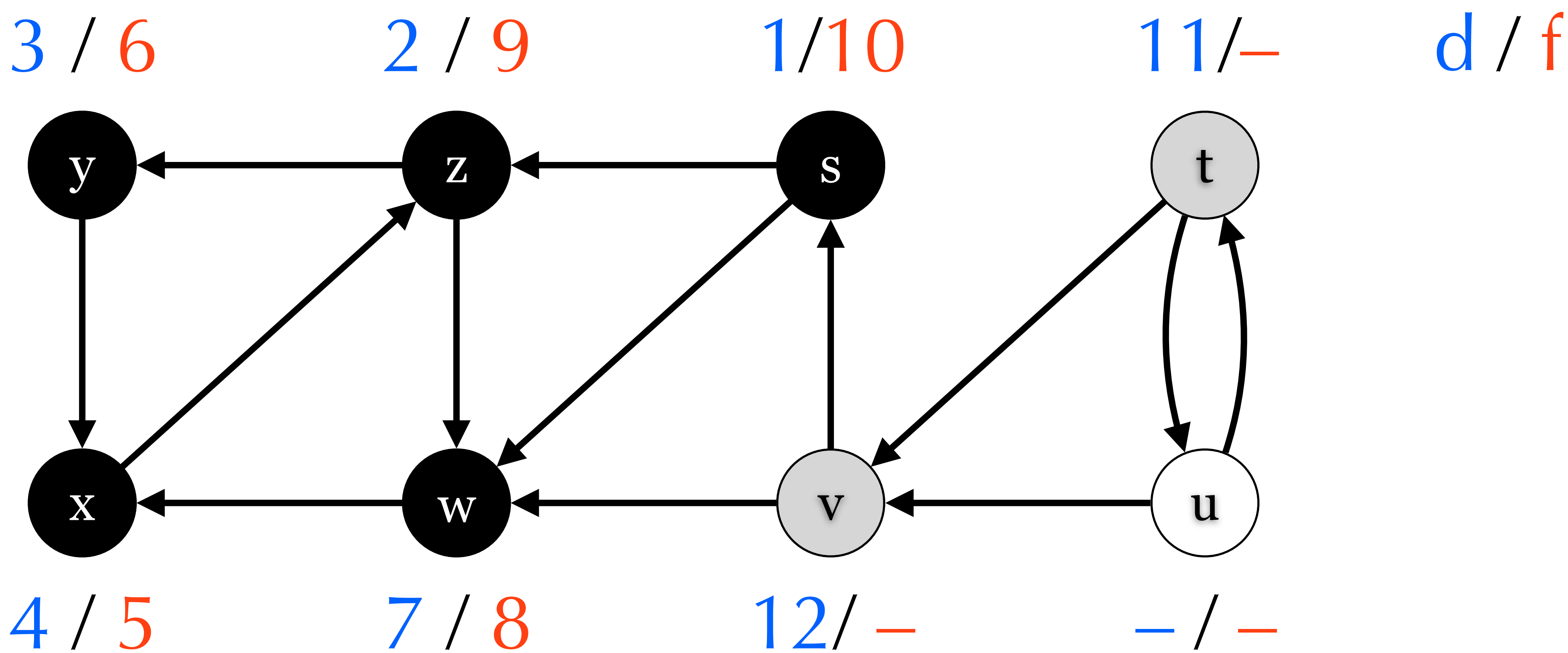
	s	t	u	v	w	x	y	z
π	NIL	NIL	NIL	NIL	z	y	z	s

Example



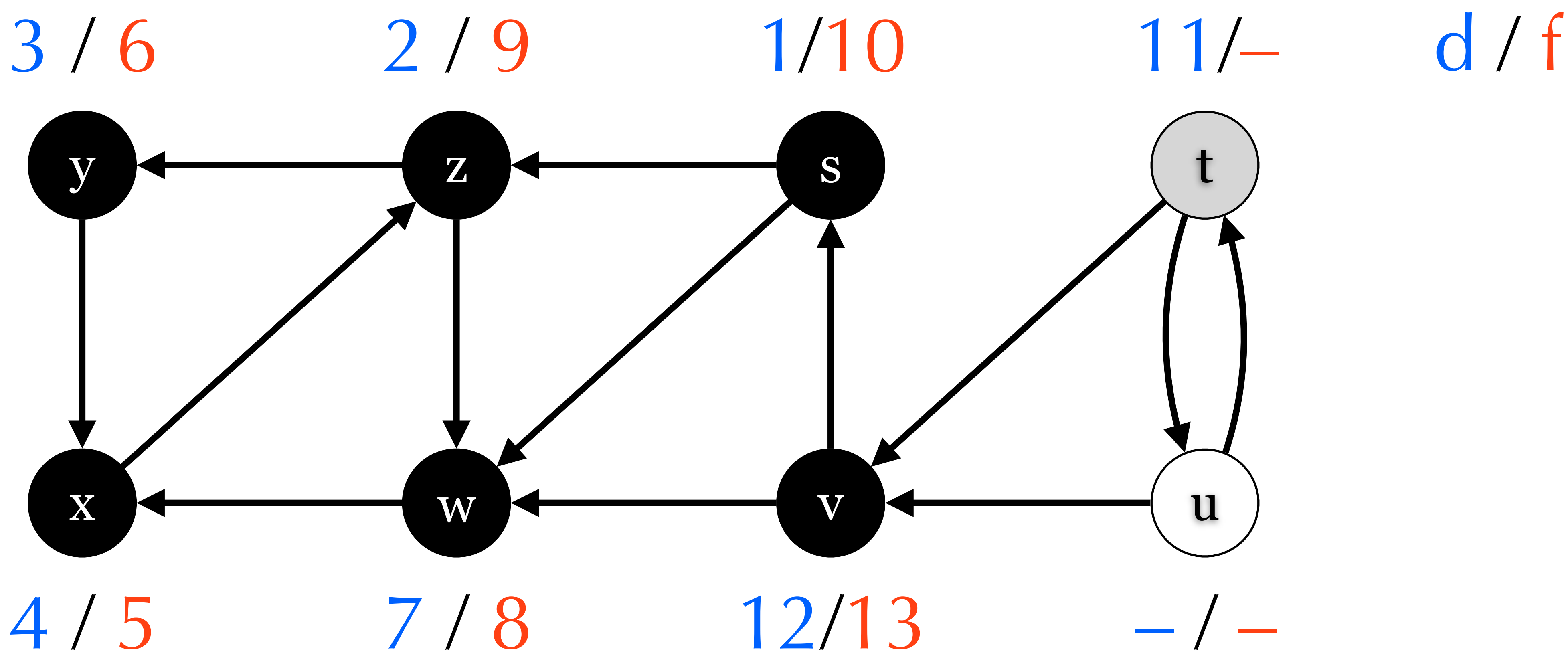
	s	t	u	v	w	x	y	z
π	NIL	NIL	NIL	NIL	z	y	z	s

Example



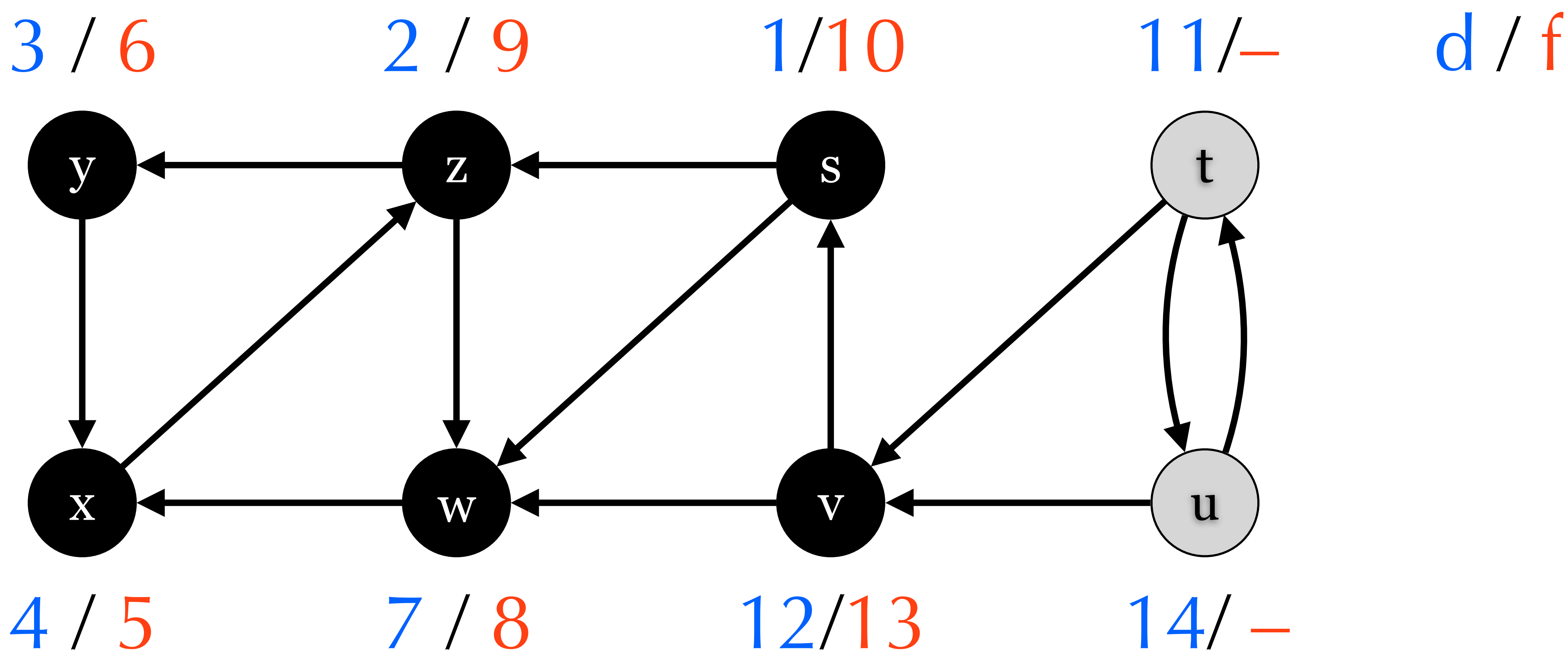
	s	t	u	v	w	x	y	z
π	NIL	NIL	NIL	t	z	y	z	s

Example



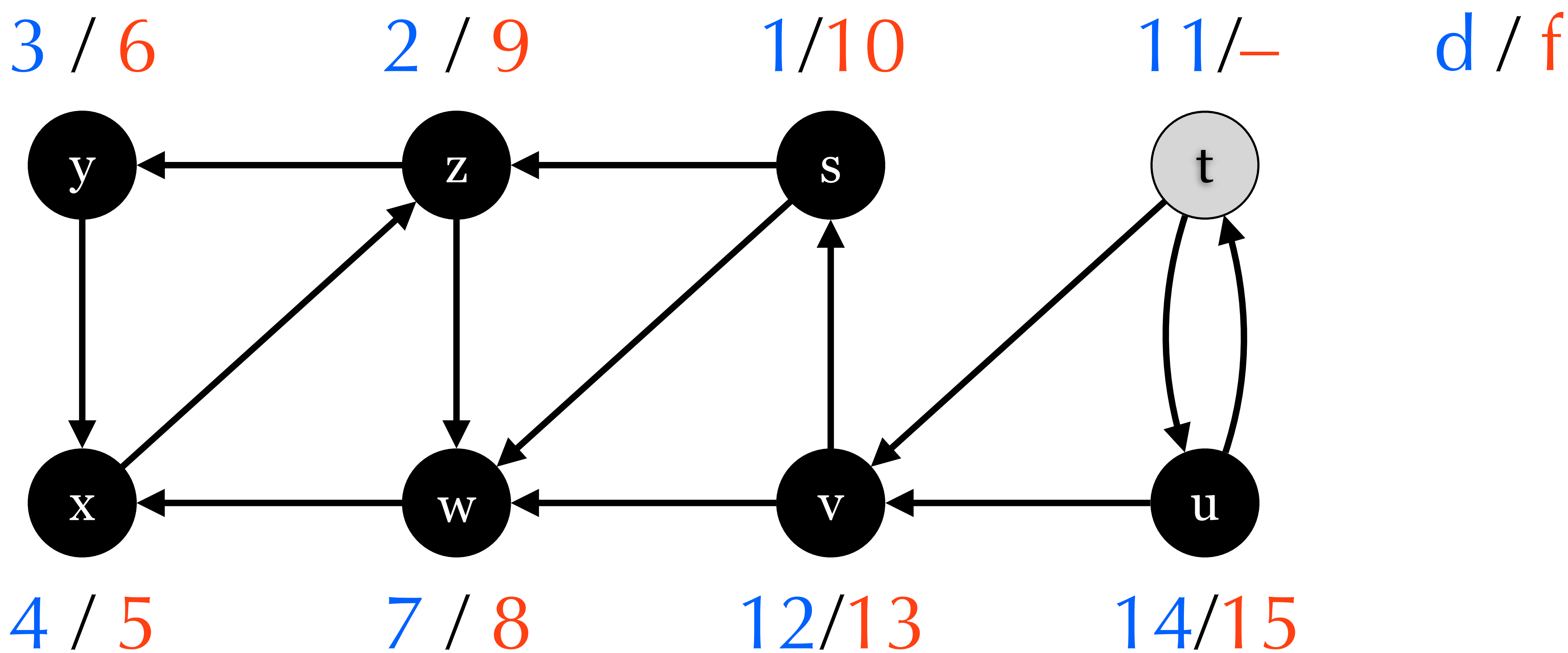
	s	t	u	v	w	x	y	z
π	NIL	NIL	NIL	t	z	y	z	s

Example



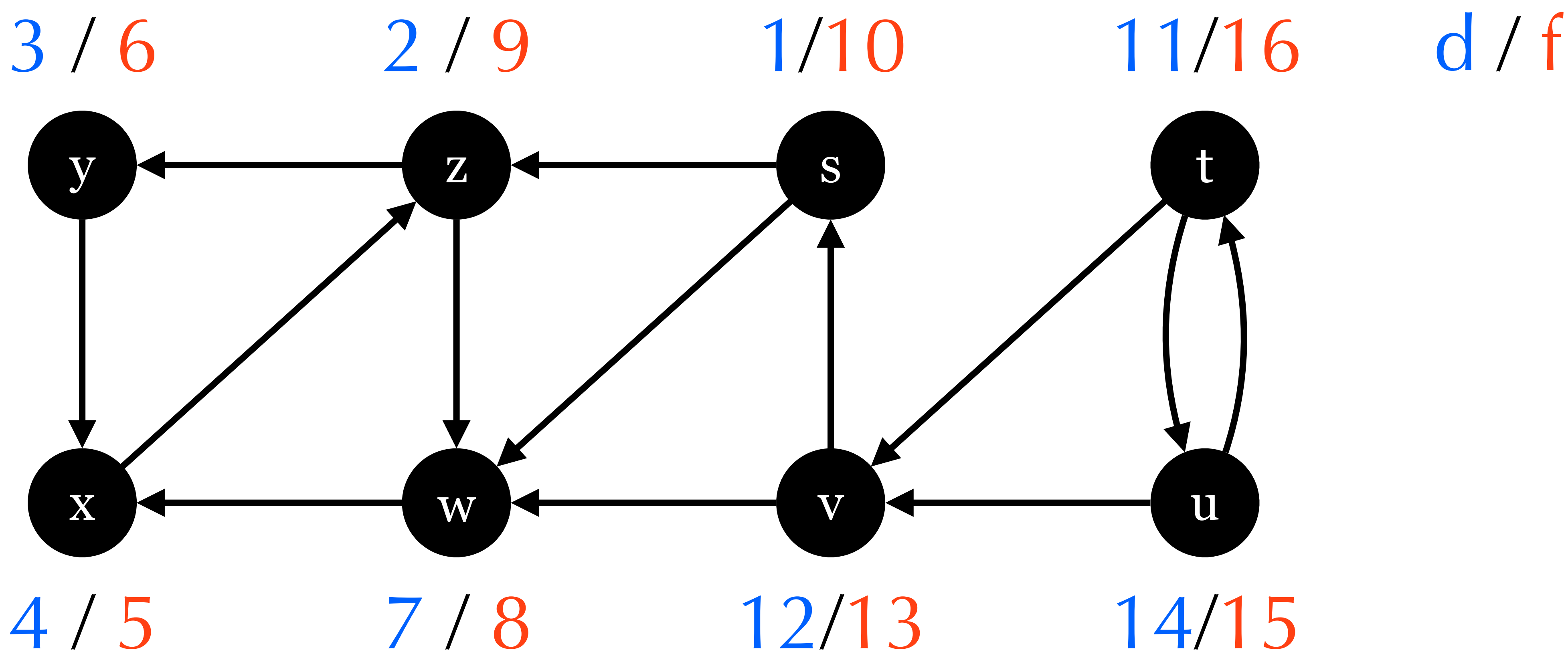
	s	t	u	v	w	x	y	z
π	NIL	NIL	t	t	z	y	z	s

Example



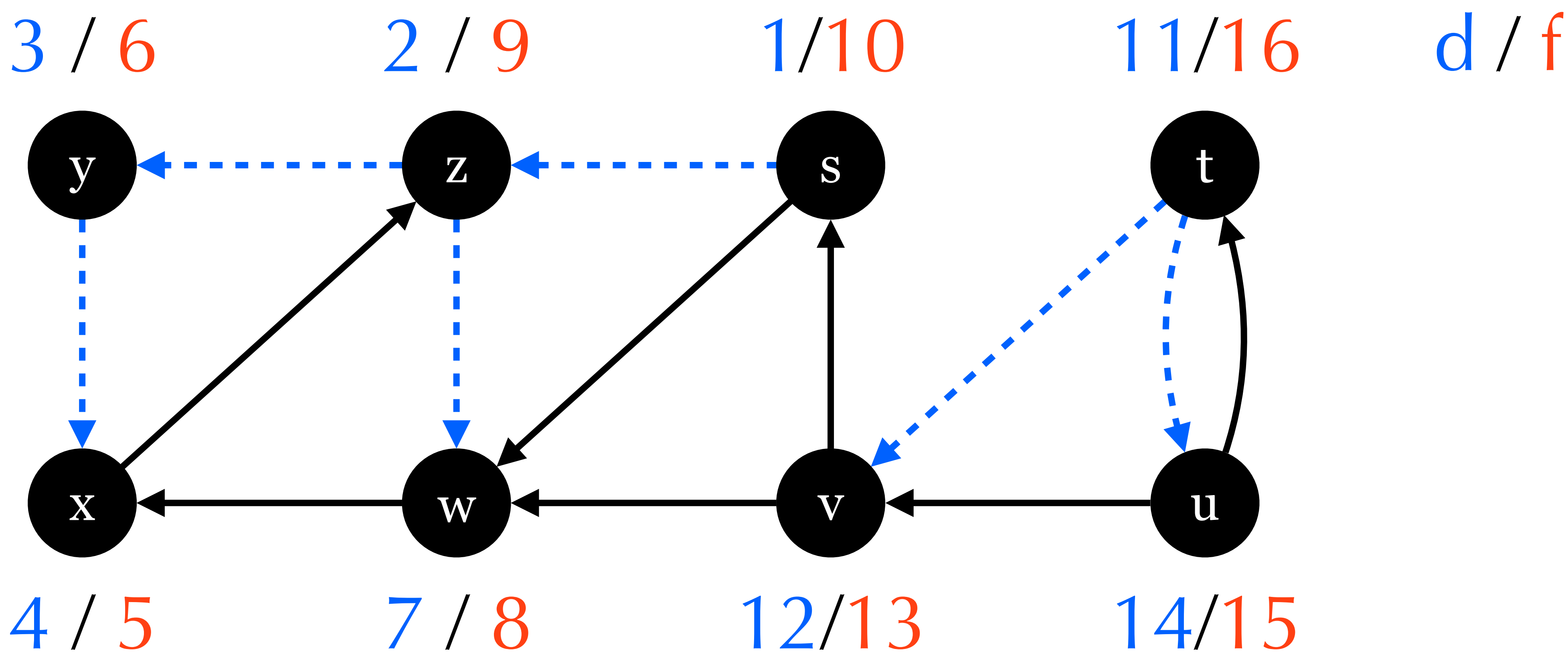
	s	t	u	v	w	x	y	z
π	NIL	NIL	t	t	z	y	z	s

Example



	s	t	u	v	w	x	y	z
π	NIL	NIL	t	t	z	y	z	s

DFS Forest

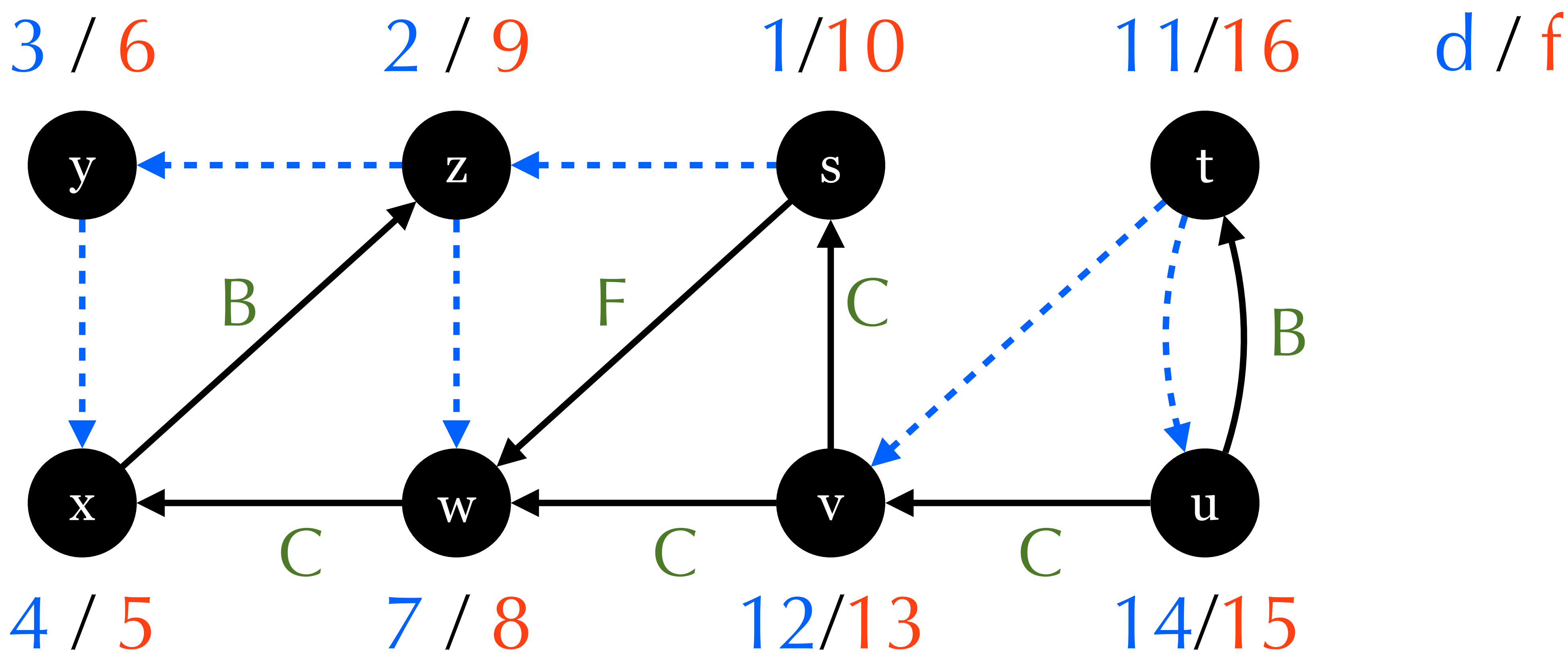


	s	t	u	v	w	x	y	z
π	NIL	NIL	t	t	z	y	z	s

DFS: Edge Classification

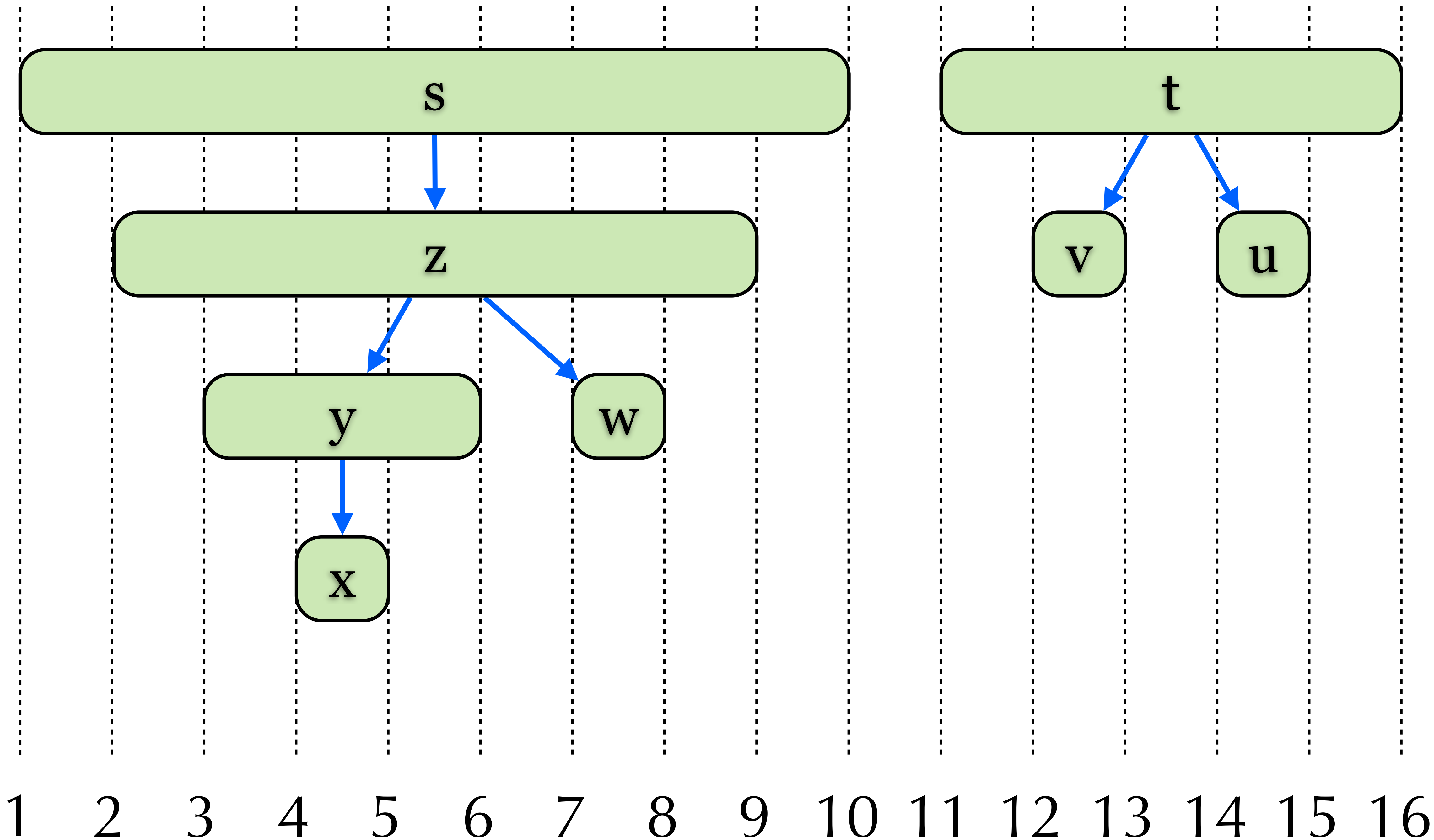
- Tree edge (u_T, v_T): the edge first discovers v_T. I.e., $v_T.\pi = u_T$.
- Back edge (u_B, v_B): v_B is an ancestor of u_B.
- Forward edge (u_F, v_F): u_F is already an ancestor of v_F.
- Cross edge (u_C, v_C): All the other edges.
  - u_C and v_C are not in the same tree
  - u_C and v_C are in the same tree but there is no path from one to another in the tree.

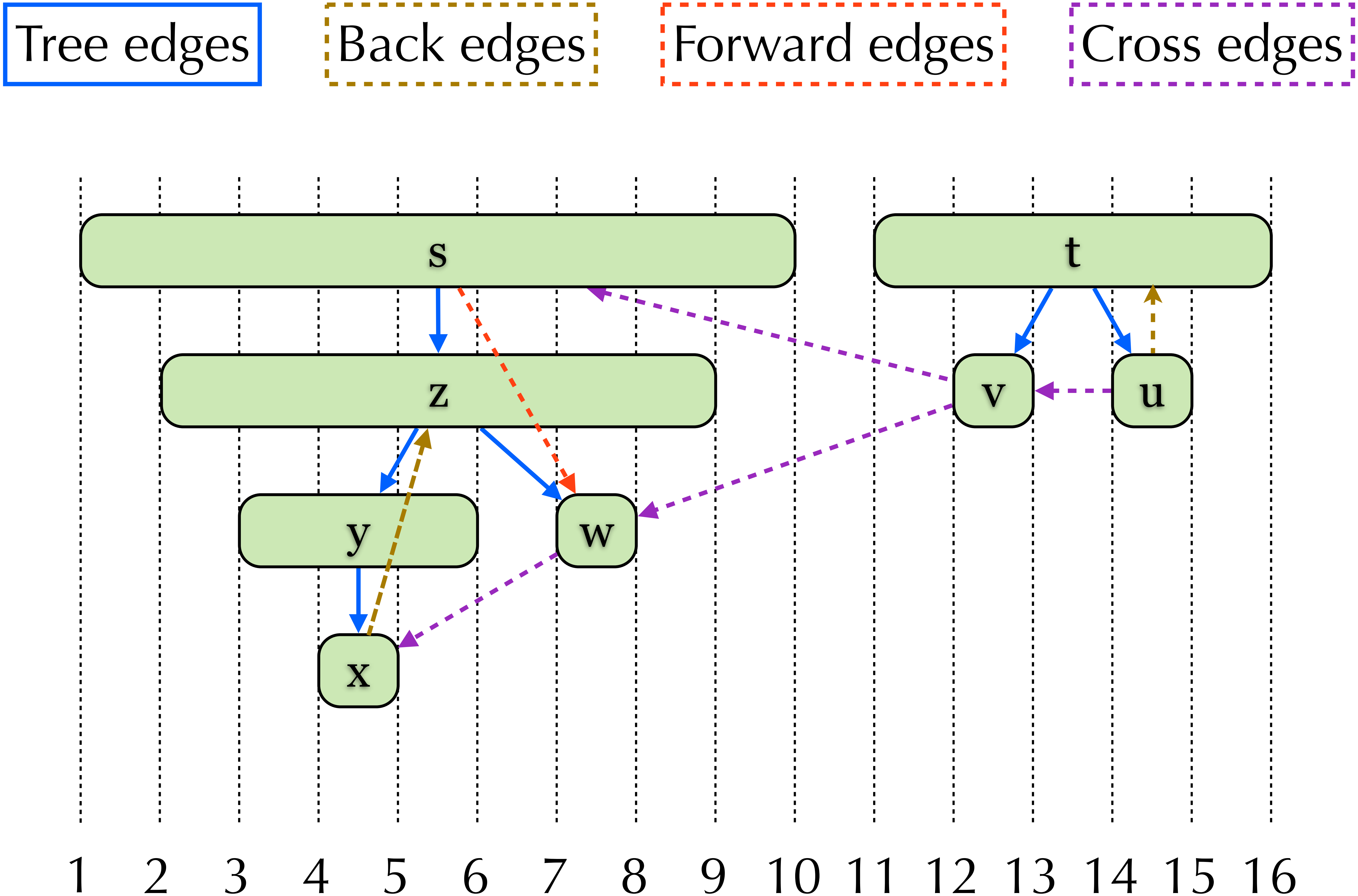
DFS Forest



	s	t	u	v	w	x	y	z
π	NIL	NIL	t	t	z	y	z	s

DFS: Edge Classification

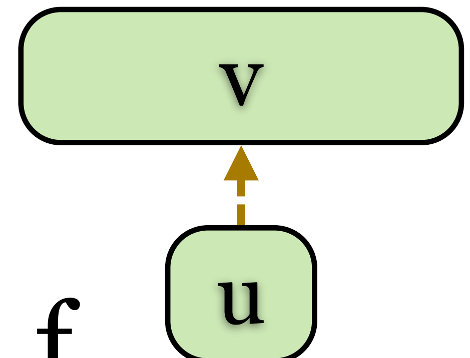




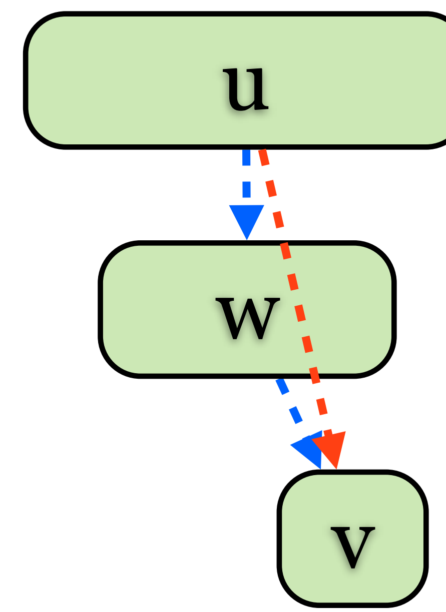
Tree edges
Back edges
Forward edges
Cross edges
s
t
z
v
u
y
w
x
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

DFS: Edge Classification

Back edges



- Tree edge (u_T, v_T): $v_T.\pi = u_T$
- Back edge (u_B, v_B):
  - $v_B = u_B$ or $v_B.d < u_B.d < u_B.f < v_B.f$
- Forward edge (u_F, v_F):
  - $v_F.\pi \neq u_F$ and $u_F.d < v_F.d < v_F.f < u_F.f$
- Cross edge (u_C, v_C): Otherwise.



Forward edges

Type	Tree	Back	Forward	Cross
v.c	WHITE	GRAY	BLACK	BLACK

DFS Property

- Parenthesis theorem (Thm 22.7)
- In any DFS of a graph G=(V,E), for any two vertices u and v, exactly one of the following 3 conditions holds:
  - $[u.d, u.f] \cap [v.d, v.f] = \emptyset$ and neither u nor v is a descendant of the other in the DFS forest.
  - $[u.d, u.f] \subseteq [v.d, v.f]$ and u is v's descendent.
  - $[v.d, v.f] \subseteq [u.d, u.f]$ and v is u's descendent.

Proof

- Discuss all cases
- Focus on the case u.d<v.d
- u.d>v.d: Symmetric to the case u.d<v.d
- u.d<v.d & v.d<u.f
  - v is GRAY when v is discovered.
  - v is a descendant of u.
  - v.f<u.f: DFS-Visit(v) is called by DFS-Visit(u).
  - We have u.d<v.d<v.f<u.f

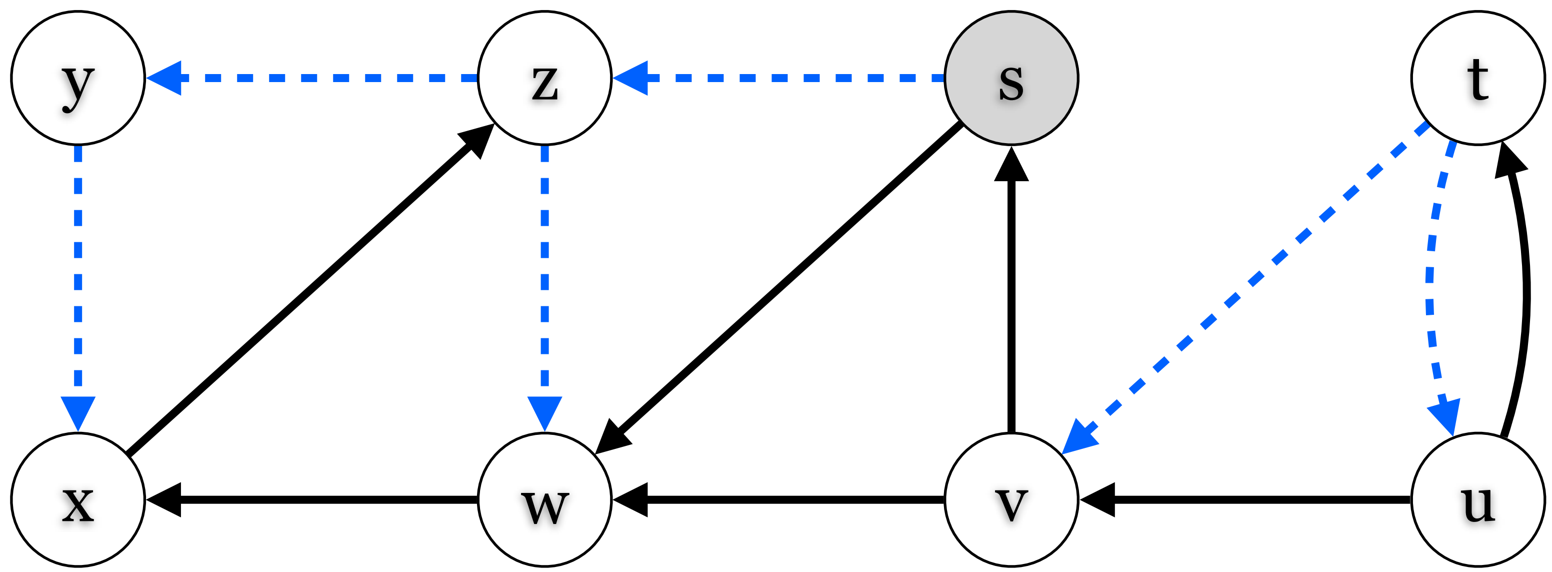
Proof

- $u.d<v.d$ & $v.d>u.f$
  - $u.d<u.f<v.d<v.f$
  - When v is discovered, u is BLACK.
  - u is not an ancestor of v.
  - When u is discovered, v is WHITE.
  - v is not an ancestor of u.
- We completely discussed the case $u.d<v.d$
- $u.d>v.d$: Symmetric to the case $u.d<v.d$

DFS Property

- White path theorem (Thm 22.9)
- In a DFS forest of a graph G=(V,E), v is a descendant of u if and only if at the time u.d that DFS discovers u, there is a path from u to v consisting entirely of white vertices.

Theorem 22.9



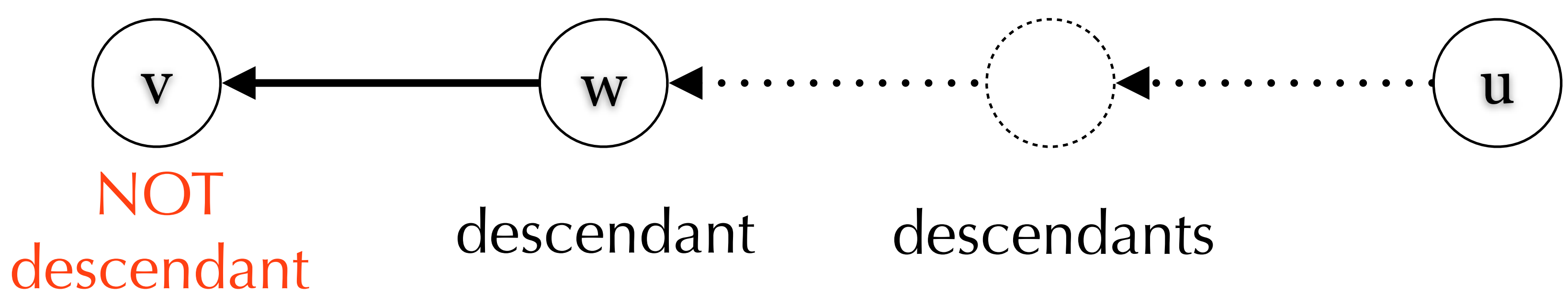
When z is discovered, there is a all-white path from z to x.

Proof

- Part 1: v is a descendant of u, then there exists a white path from u to v.
- u=v: we color u after setting u.d.
- Consider the path from u to v in DFS tree. Any vertex w on the path is a descendant of u, so we have $u.d \le w.d$ by Thm 22.7. This means w is WHITE when u is discovered.

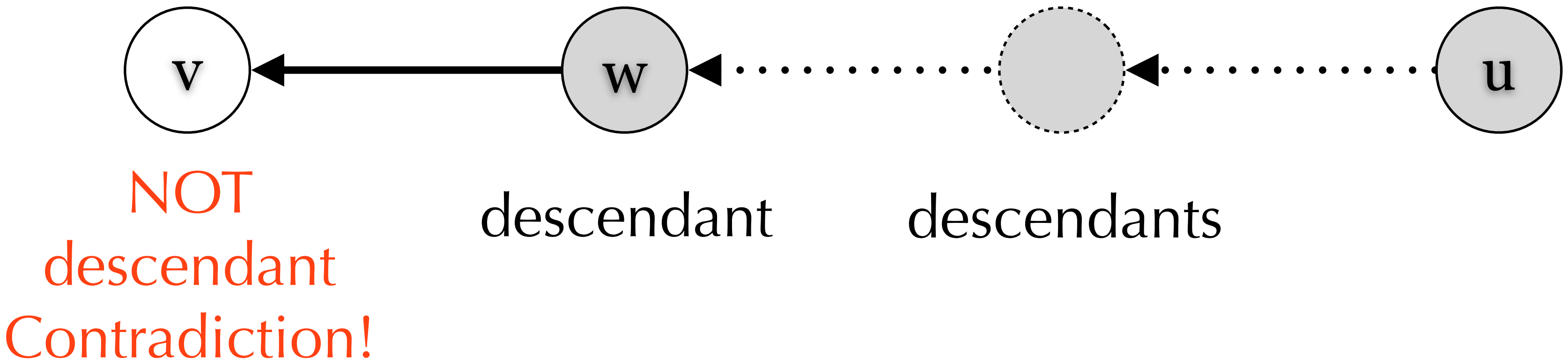
Proof

- Part 2: When u is discovered, there exists a white path from u to v implies v is a descendant of u.
- BWOC, assume v is not a descendant of u. WLOG, assume v is the closest one.



Proof

- By Thm 22.7: $u.d \le w.d < w.f \le u.f$
- After w is discovered, v is still WHITE.
- (w,v) triggers DFS-Visit(v), so $w.d < v.d < w.f$.
- By Thm 22.7: $u.d \le w.d < v.d < v.f < w.f \le u.f$
  - v is a descendant of u.



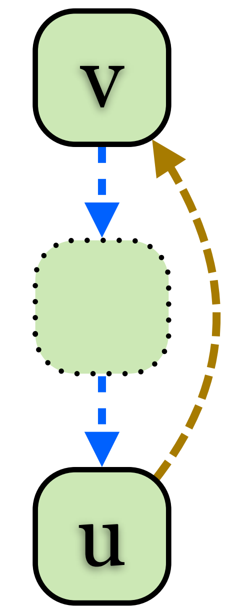
DFS Property

- DFS of an undirected graph (Thm 22.10)
- In a depth-first search of an undirected graph G, every edge of G is either a tree edge or a back edge.

Proof

- Treat an undirected edge {u,v} as two directed edges (u,v) and (v,u).
  - When one is marked, we ignore another.
- Assume u is discovered first and (u,v) is not marked as a tree edge.
  - v is a descendent of u. white path thm
  - When v is discovered, (v,u) will be marked as a back edge, and (u,v) will be ignored.
  - {u,v} is a back edge.

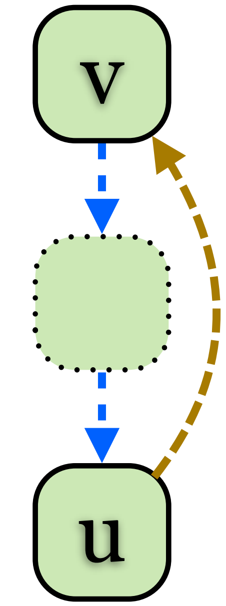
DFS Property

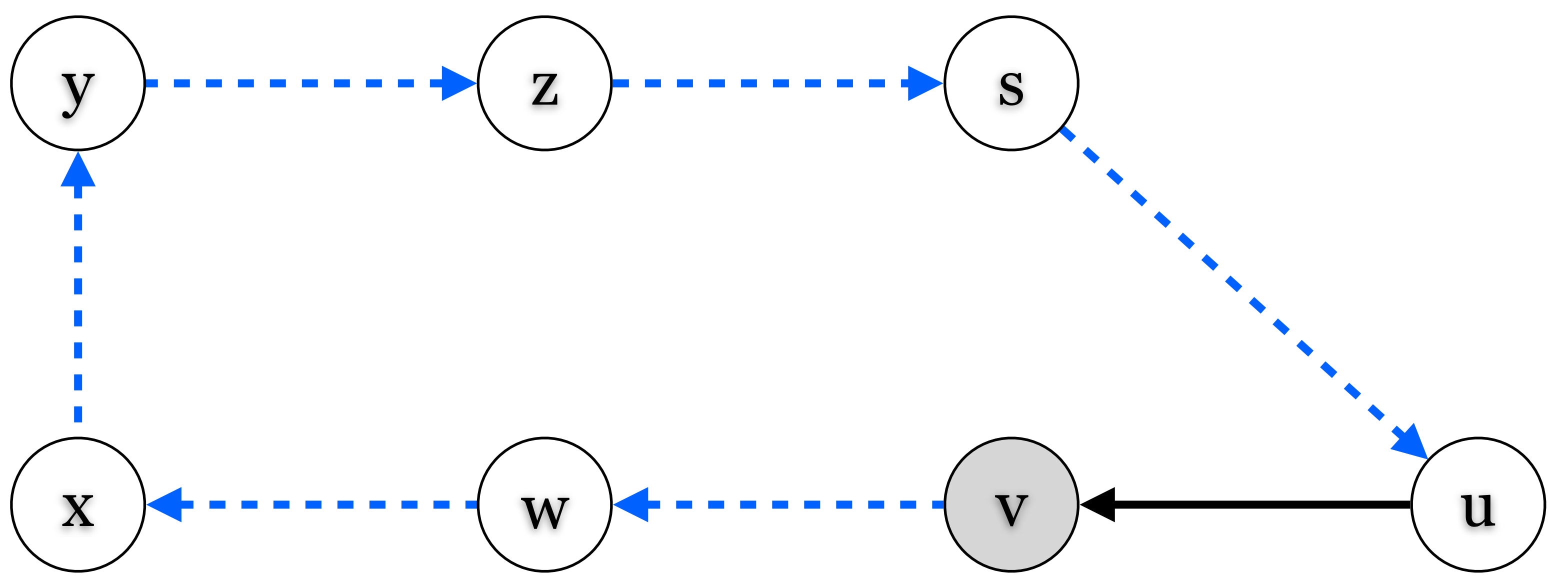


- Back edges imply cycles (Lemma 22.11)
- A directed graph G is acyclic if and only if a DFS of G yields no back edges.
- Proof:
  - If there is a back edge (u,v), then we know u is a descendant of v in DFS forest. So append (u,v) on the path from v to u, we have a cycle.

Proof

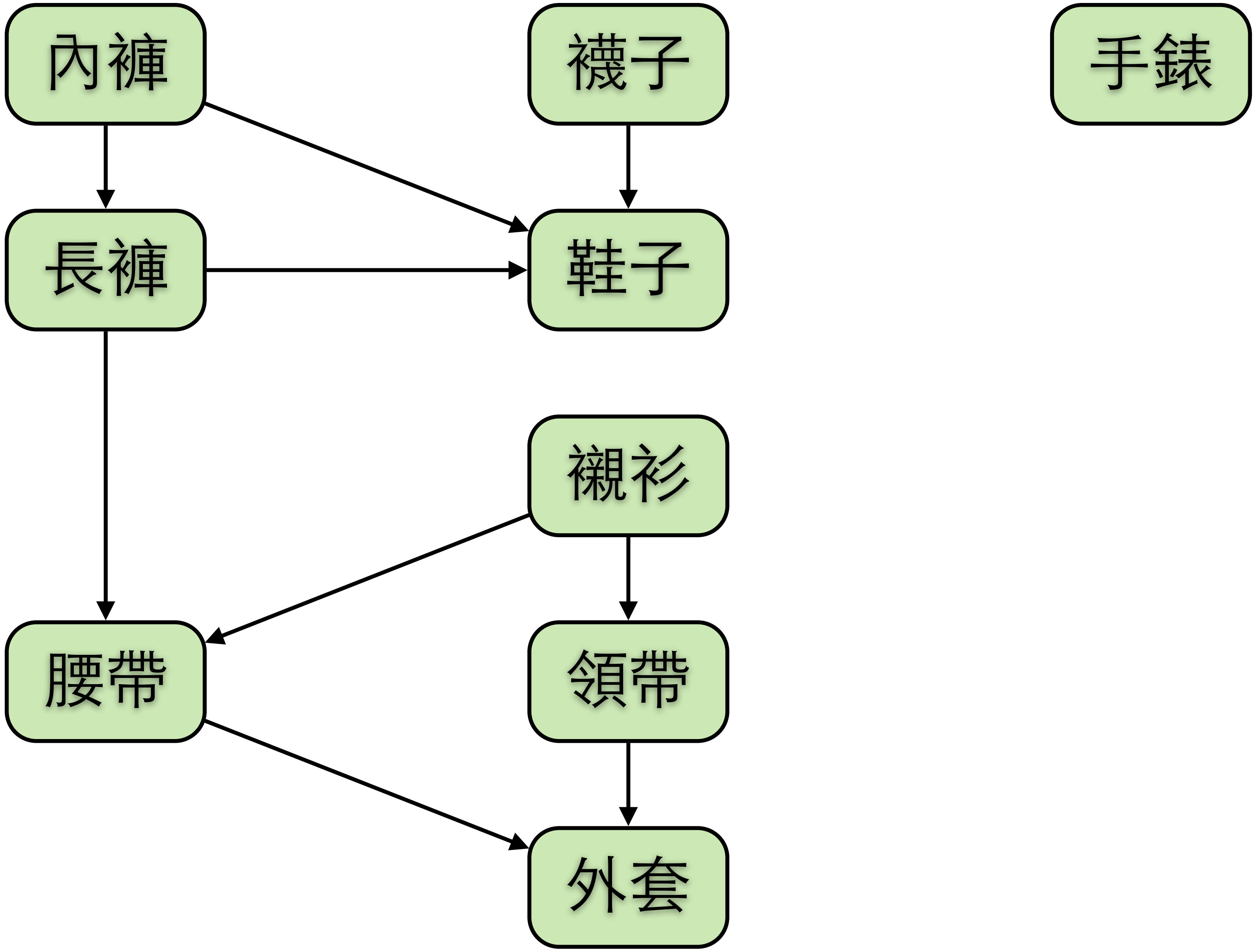
- Suppose there is a cycle containing (u,v), and v is the first discovered vertex. When v is discovered, all other vertices in the cycle are white. There is a white path from v to u, so u is a descendent of v in DFS forest. We can conclude (u,v) is a back edge.

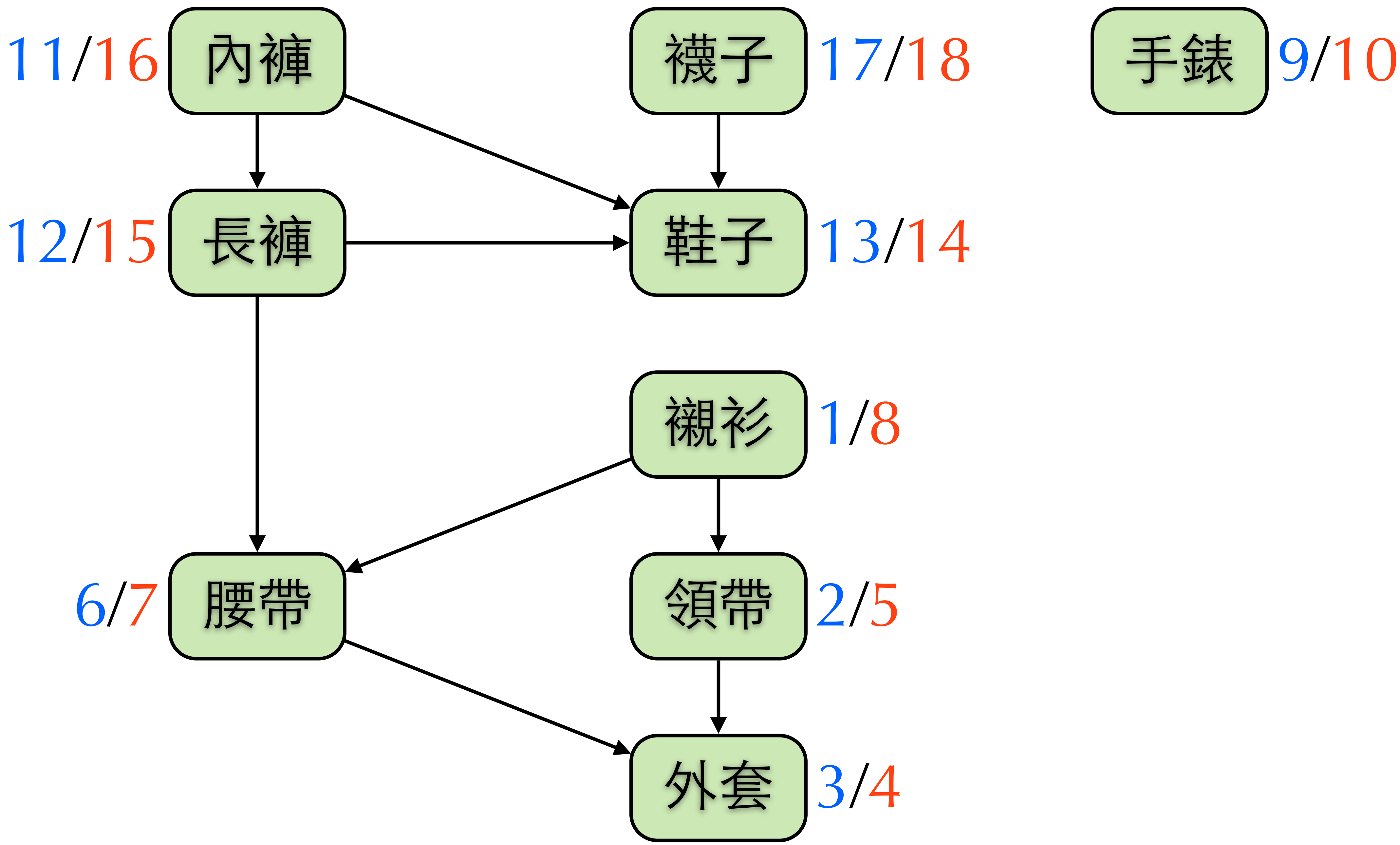


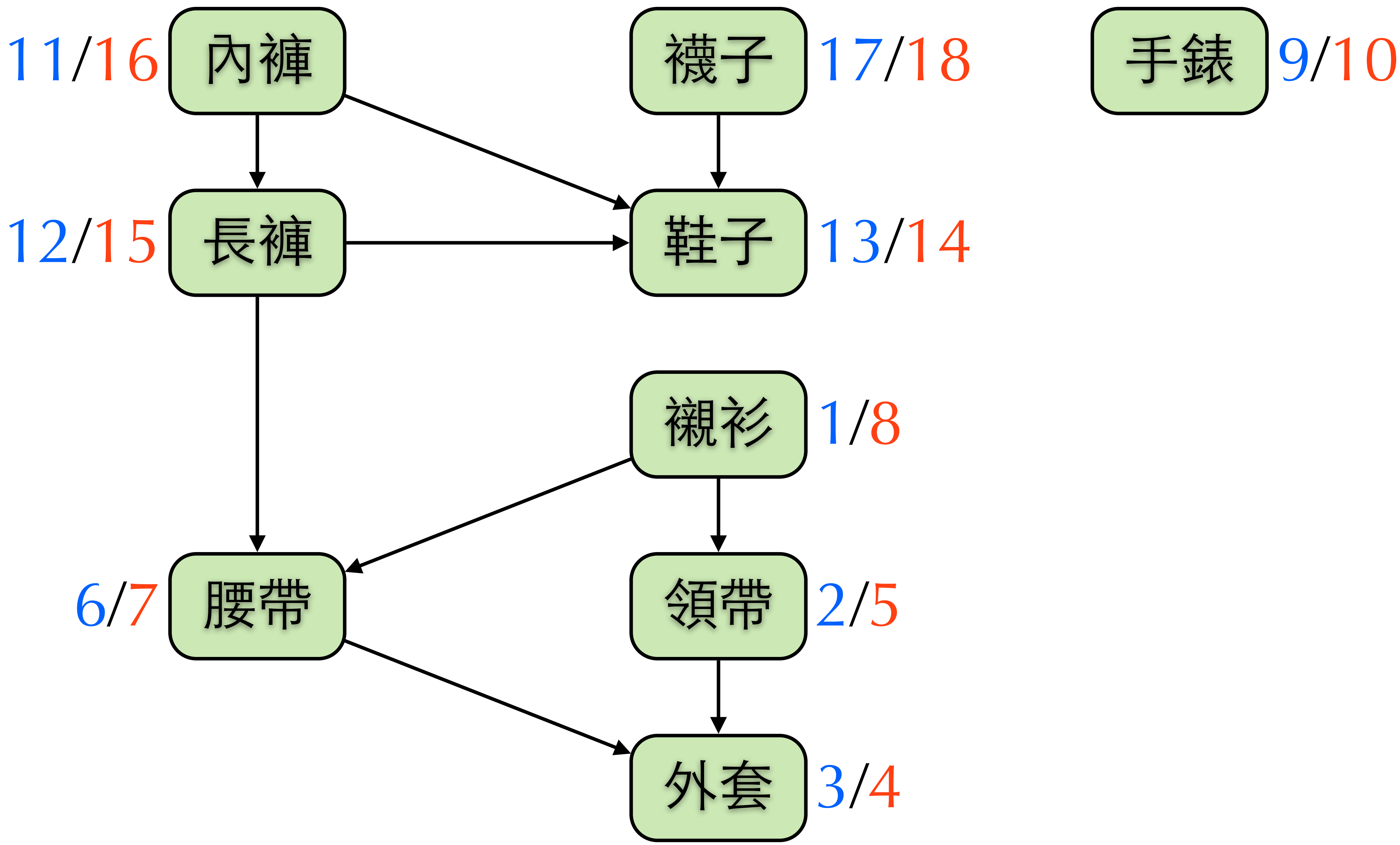


Topological Sort

- A topological sort of a directed acyclic graph (DAG) G=(V,E) is a linear ordering of V such that if G contains an edge (u,v), then u appears before v in the ordering.
- Algorithm: $O(|V|+|E|)$
  - Run DFS on G. abort if any back edge exists
  - Sort the vertices by their finish time in descending order.
    - Counting sort: $O(|V|)$
    - Modify DFS: Use a linked list/stack $O(|V|)$


內褲
襪子
手錶
長褲
鞋子
襯衫
腰帶
領帶
外套


11/16
內褲
襪子
17/18
手錶
9/10
12/15
長褲
鞋子
13/14
襯衫
1/8
6/7
腰帶
領帶
2/5
外套
3/4

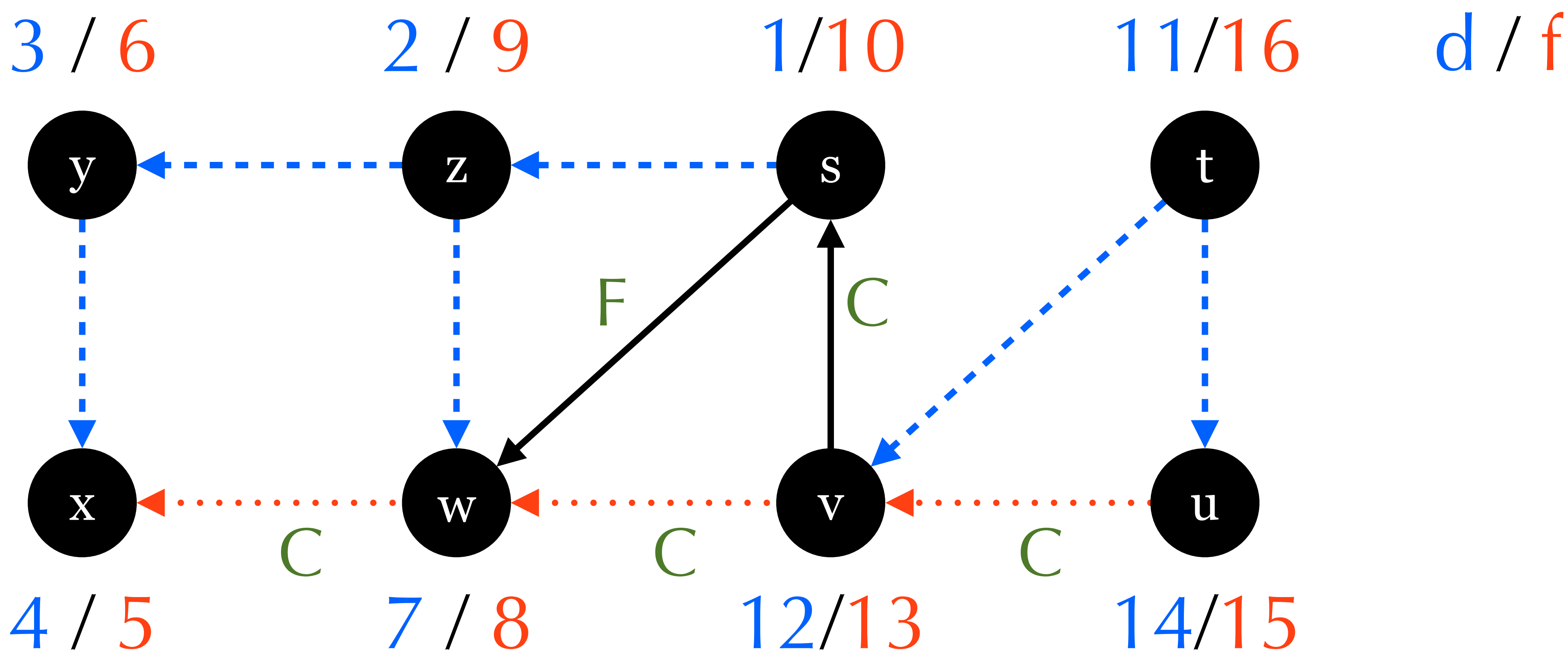


襪子 內褲 長褲 鞋子 手錶 襯衫 腰帶 領帶 外套

Correctness Proof

- Goal: if $(u,v) \in E$ then $v.f < u.f$.
  - This is sufficient (why?)
- G is acyclic: no back edges.
- (u,v) is a tree edge or a forward edge:
  - $u.d < v.d < v.f < u.f$ (Thm 22.7)
- (u,v) is a cross edge:
  - v.c=BLACK when u is discovered.
  - $v.f < u.d < u.f$

Sufficient



Homework & Bonus

- Homework: Give a non-recursive algorithm perform topological sort in O(|V|+|E|)-time.
- Bonus: strongly connected components
  - Present algorithms
    - Kosaraju's (Ch22.5)
    - Tarjan's
  - Create a programming assignment for future students.