Graphs, Part 7

Ira A. Fulton Schools of Engineering
Arizona State University

Classification of Edges (by DFS)

- **Tree edges**: An edge (u, v) is in a tree edge if v is discovered by exploring edge (u, v).

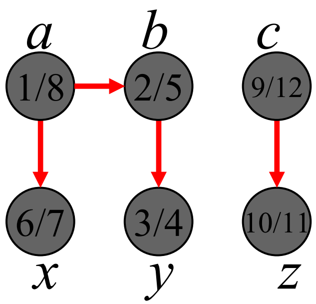
- **Back edges** are those edges (u, v) connecting a vertex u to an ancestor v in a depth-first tree. A self-loop edge is considered as a back edge.

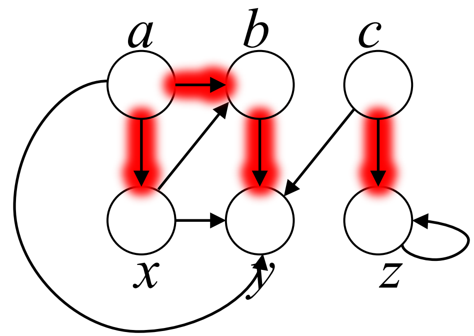
Classification of Edges (by DFS)

- **Forward edges** are non-tree edges (u, v) connecting a vertex u to a descendant v in a depth-first tree.

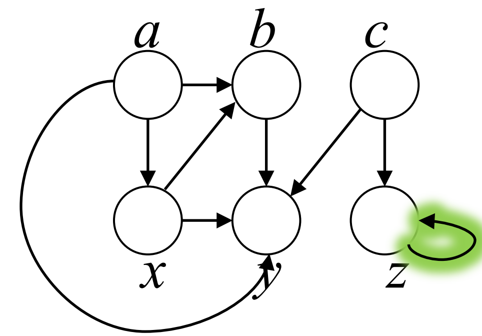
- **Cross edges** are all other edges.

Classification of Edges (directed graph)

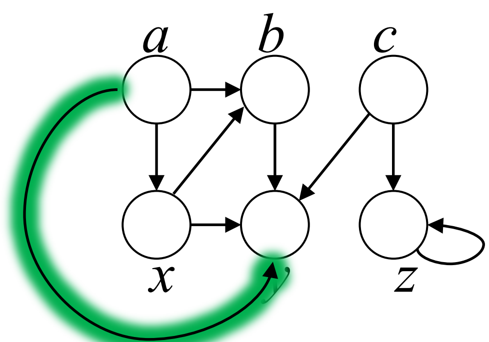




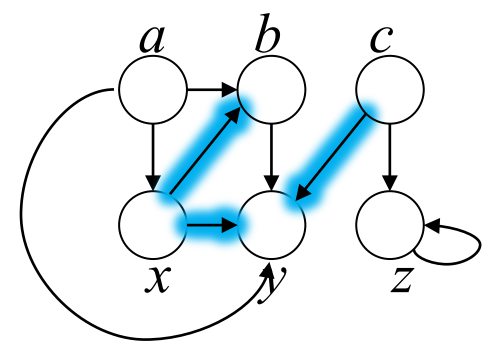
Tree Edges



Back Edges

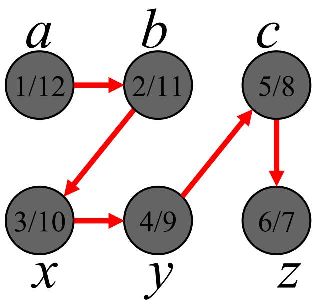


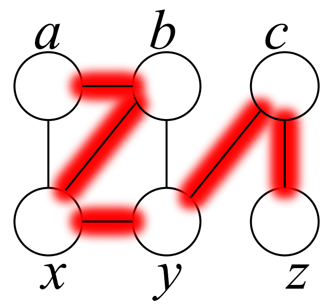
Forward Edges



Cross Edges

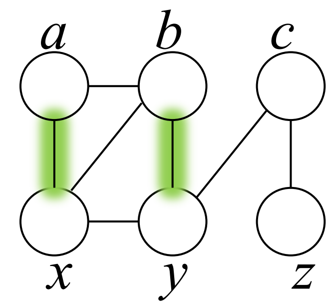
Classification of Edges (undirected graph)





Tree Edges

Traversed (a, b)
before (b, a)



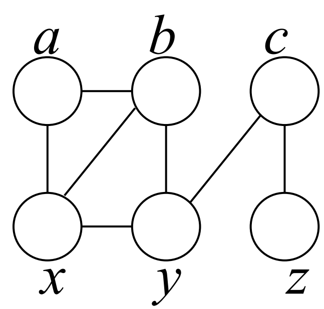
Back Edges

Traversed (x, a)
before (a, x)

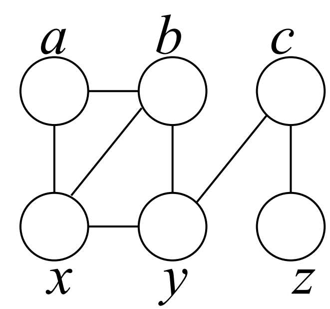
Theorem: In a DFS of an undirected graph G, for each undirected edge (u, v), we have either

(1) (u, v) or (v, u) is first traversed as a tree edge, or

(2) (u, v) or (v, u) is first traversed as a back edge.



Forward Edges



Cross Edges

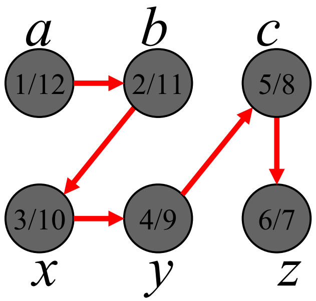
The Case of Undirected Graphs

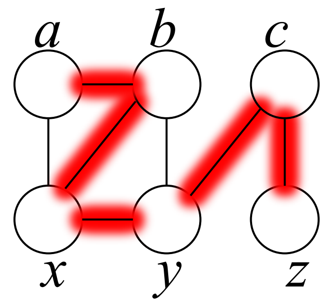
- In a DFS of an undirected graph, every edge is either a tree edge or a back edge.
- Why?
- Let (u, v) be an arbitrary edge in the graph. WLOG, assume that u.d < v.d. This means that when u turns GRAY, v is still WHITE. Since (u, v) is an edge, v will turn GRAY while u is still GRAY.
- Therefore, the search must discover and finish v before it finishes u.

The Case of Undirected Graphs

- Therefore, the search must discover and finish v before it finishes u.
- If (u, v) is first explored from u, (u, v) is a tree edge. (a, b) in the example is such an edge.
- If (u, v) is first explored from v, (u, v) is a back edge. (a, x) in the example is such an edge.

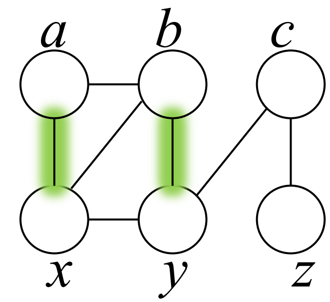
Illustration





Tree Edges

Traversed (a, b) before (b, a)



Back Edges

Traversed (x, a) before (a, x)

Nesting of Descendants' Intervals

Theorem: Vertex v is a proper descendant of vertex u in the DFS forest if and only if $u.d < v.d < v.f < u.f$.

Proof. It follows from the DFS algorithm, the set $\{v.d, v.f | v \in V\}$ are 2n distinct positive integers $\{1, 2, \ldots, 2n-1, 2n\}$.

Assume that v is a proper descendent of u. Then v is WHITE when u turns GRAY, and v turns GRAY (then BLACK) when u is still GRAY. Hence, we have $u.d < v.d < v.f < u.f$.

Assume that $u.d < v.d < v.f < u.f$. Then v turns GRAY when u is GRAY, and v turns BLACK while u is still GRAY. Hence v is a proper descendent of u.

White-path Theorem

Theorem: In a DFS forest of a (directed or undirected) graph G=(V, E), vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.

Proof. This follows directly from the DFS algorithm.

Parenthesis Theorem

Theorem: In any depth-first search of a (directed or undirected) graph G=(V, E), for any two vertices u and v, exactly one of the following three conditions holds:

- the intervals [u.d, u.f] and [v.d, v.f] are entirely disjoint, and neither u nor v is a descendant of the other in the depth-first forest.
- the interval [u.d, u.f] is contained entirely within the interval [v.d, v.f], and u is a descendant of v in a depth-first tree, or
- the interval [v.d, v.f] is contained entirely within the interval [u.d, u.f], and v is a descendant of u in a depth-first tree.

Parenthesis Theorem (Proof)

Without loss of generality, assume that $u.d < v.d$. Hence when u turns GRAY, v is still WHITE. We consider two disjoint cases: (1) there is a WHITE path from u to v; (2) there is no WHITE path from u to v.

In case (1), v will turn GRAY before u turns BLACK. Hence, we have $u.d < v.d < v.f < u.f$. v is a descendent of u.

In case (2), v remains WHITE when u turns BLACK. Hence, we have $u.d < u.f < v.d < v.f$. Neither is a descendent of the other.

Summary

- Classification of Edges
  - Tree, Back, Forward, Cross
- White-Path Theorem
- Parenthesis Theorem

ASU Ira A. Fulton Schools of Engineering

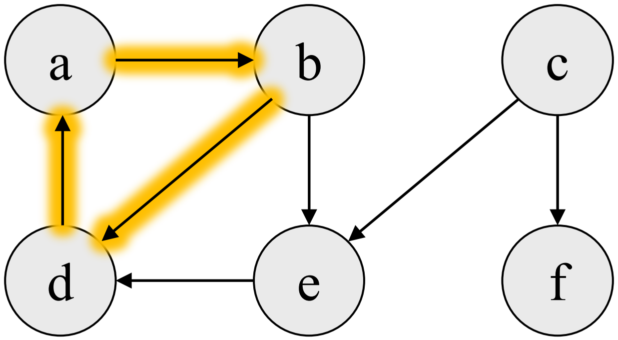
Arizona State University

Graphs, Part 8

Ira A. Fulton Schools of Engineering
Arizona State University

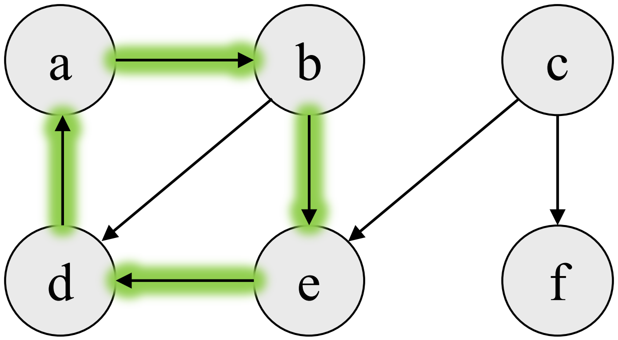
Cycle in a Directed Graph

- A cycle in directed graph (without edges from and to the same vertex) $G=(V, E)$ is a sequence of two or more vertices $<v_1, v_2, \ldots, v_k>$ such that $(v_i, v_{i+1}) \in E$ for $i=1, 2, \ldots, k-1$, and $(v_k, v_1) \in E$.
- A self-loop is considered a cycle.
- An example graph is given on this page
- <a, b, d> is a cycle



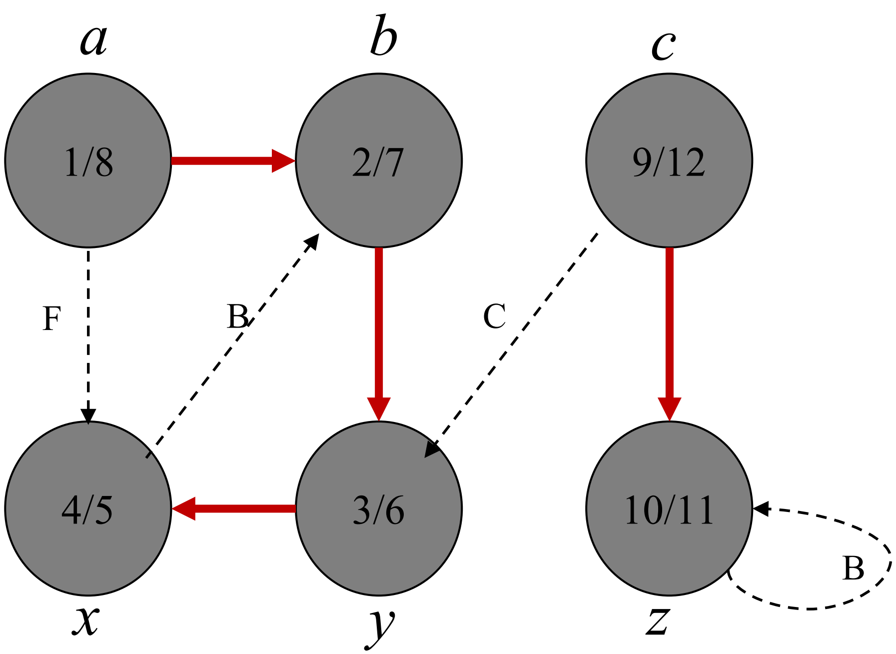
Cycle in a Directed Graph

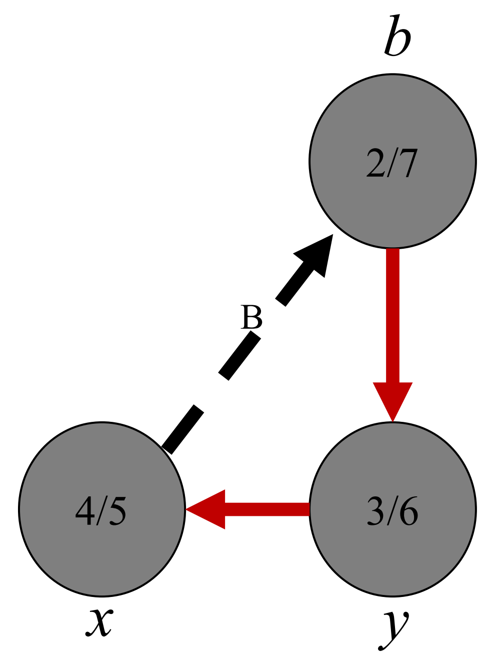
- A cycle in directed graph (without edges from and to the same vertex) G=(V, E) is a sequence of two or more vertices $<v_1, v_2, \ldots, v_k>$ such that $(v_i, v_{i+1}) \in E$ for $i=1, 2, \ldots, k-1$, and $(v_k, v_1) \in E$.
- A self-loop is considered a cycle.
- An example graph is given on this page
- <a, b, d> is a cycle
- <a, b, e, d> is a cycle



Finding a Cycle in a Directed Graph

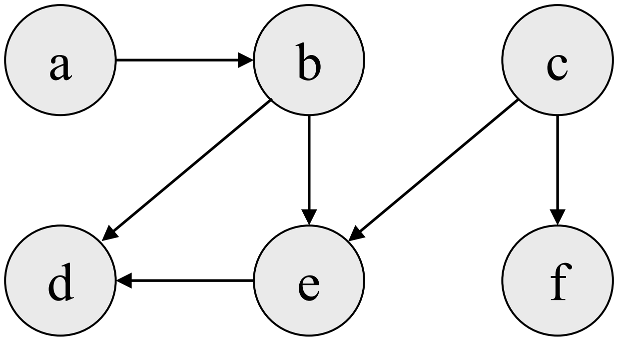
- **If there is a back edge, we have a cycle.**
- **If there is a cycle, there is a back edge.**





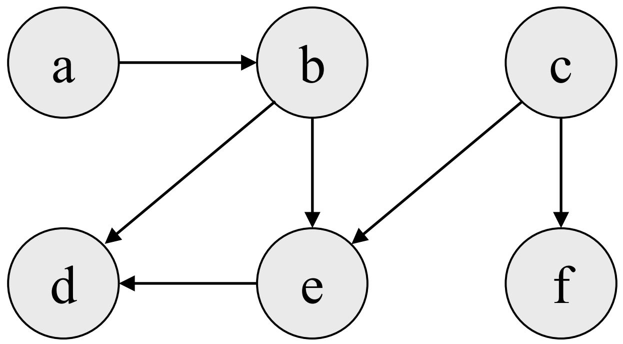
Directed Acyclic Graph (DAG)

- A directed graph is called a DAG (directed acyclic graph) if it does not contain a cycle.
- A DAG is given on this page



Topological Sort of a DAG

- A topological sort of a DAG $G=(V, E)$ is an ordering of the vertices $v_1, v_2, \ldots, v_n$ such that $(v_i, v_j) \in E$ implies $i<j$.
- Examples:
  - a, b, c, e, d, f
  - a, c, f, b, e, d
- Topological sort is not unique



Computing a Topological Sort of a DAG

Topological-Sort(G)

1 call DFS(G) to compute finishing times v.f for each vertex v

2 as each vertex is finished, insert it onto the front of a linked list

3 Return the linked list of vertices

NOTE: We can replace the linked list with a stack…

Computing a Topological Sort of a DAG

If G is a DAG, Topological-Sort(G) produces a topological sort of G.

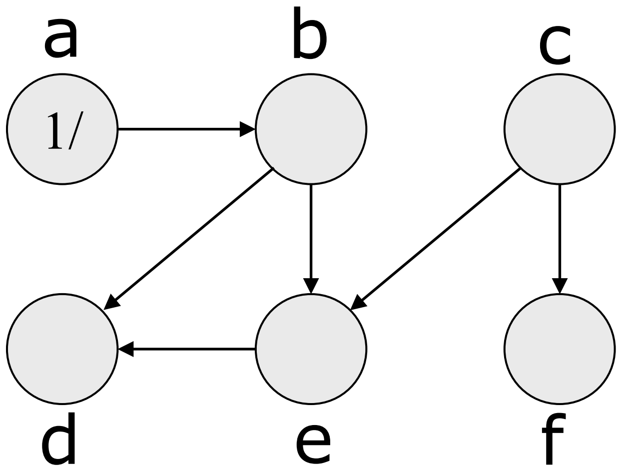
Proof. If (u, v) is an edge, then $v.f < u.f$.

We consider two cases when the edge (u, v) is explored.

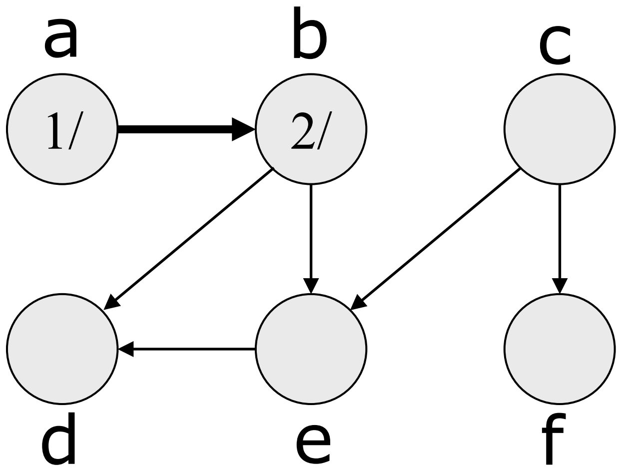
(1) v is WHITE: $u.d < v.d < v.f < u.f$

(2) v is BLACK: $u.f > v.f$

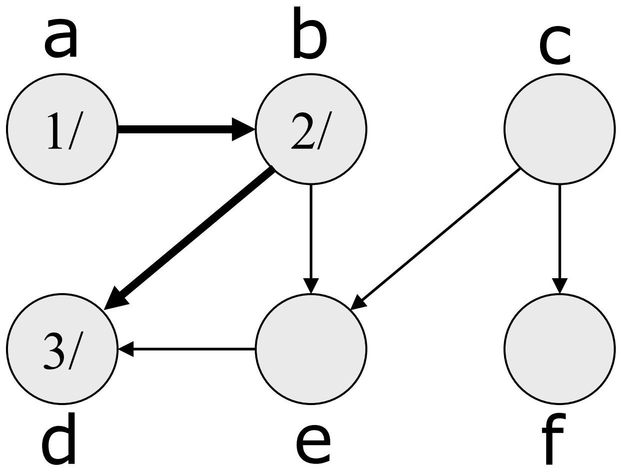
Computing a Topological Sort of a DAG



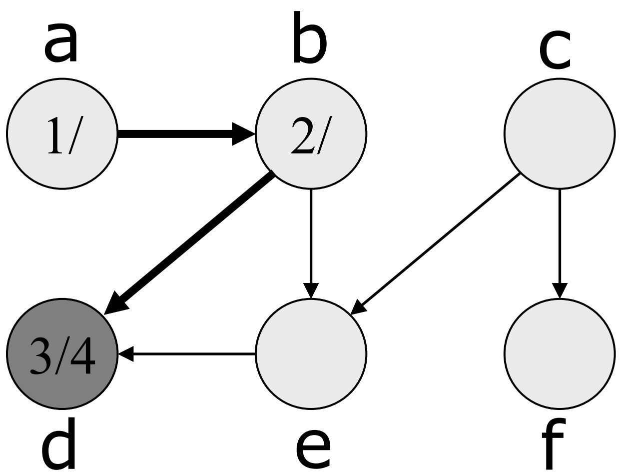
Computing a Topological Sort of a DAG



Computing a Topological Sort of a DAG

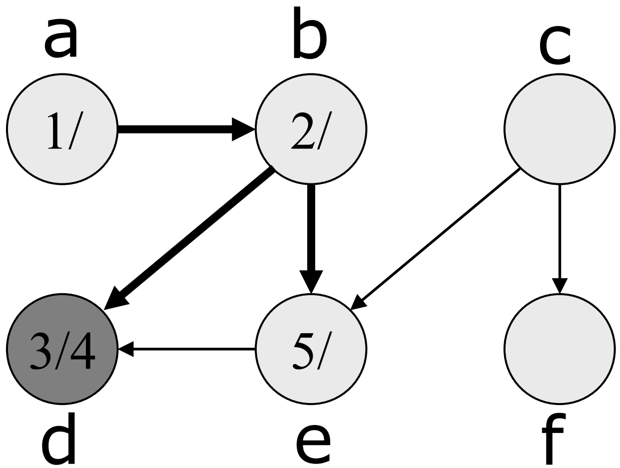


Computing a Topological Sort of a DAG



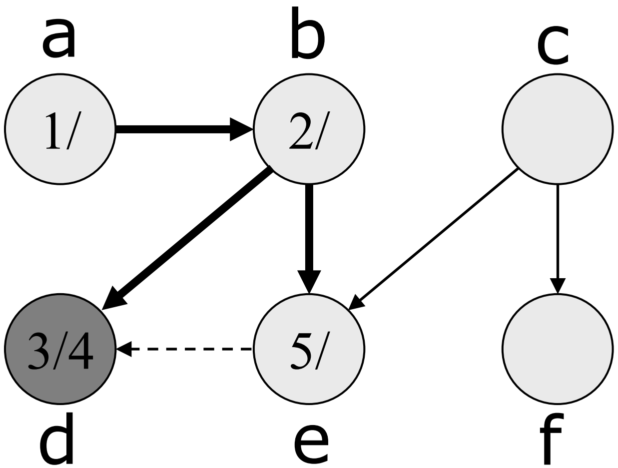
d	4

Computing a Topological Sort of a DAG



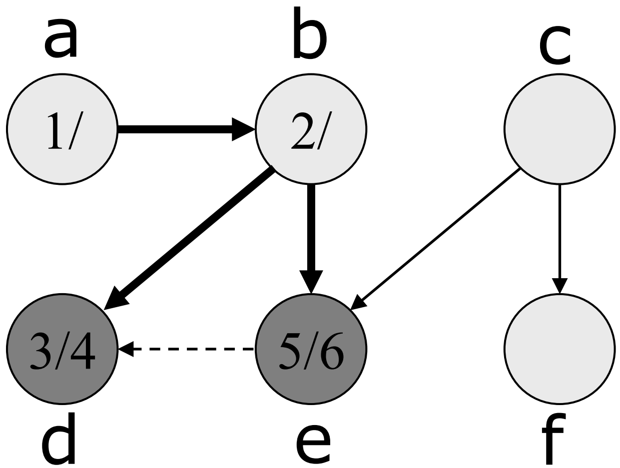
d	4

Computing a Topological Sort of a DAG



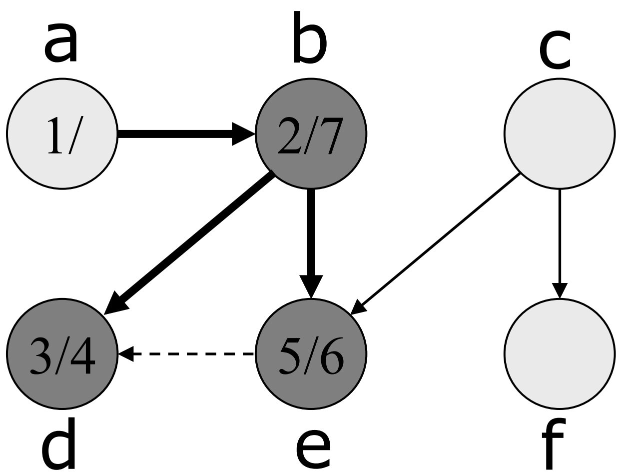
d	4

Computing a Topological Sort of a DAG



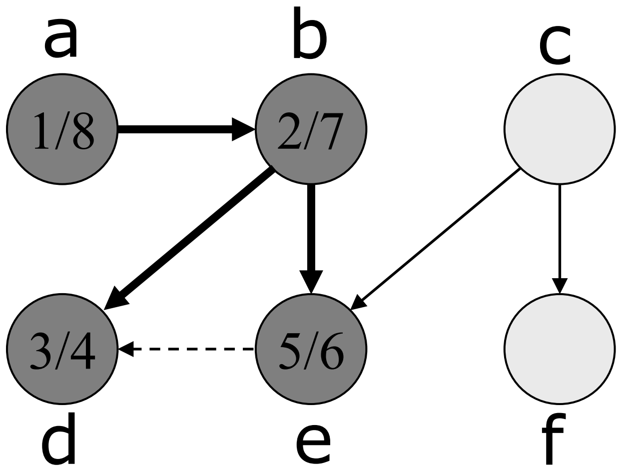
e	6
d	4

Computing a Topological Sort of a DAG



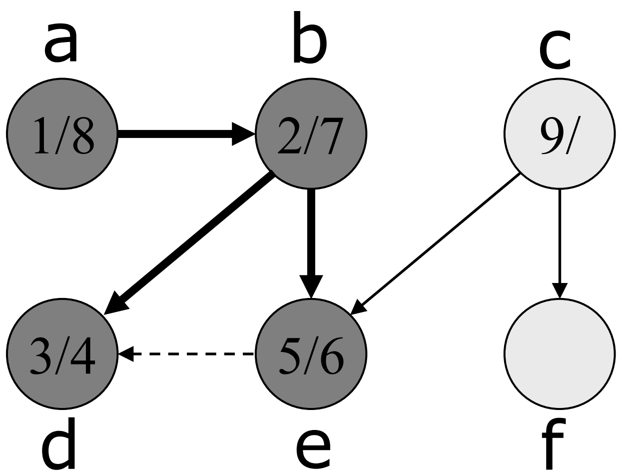
b	7
e	6
d	4

Computing a Topological Sort of a DAG



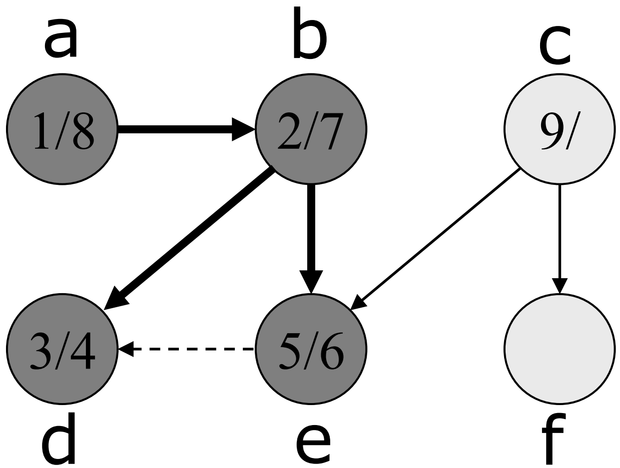
a	8
b	7
e	6
d	4

Computing a Topological Sort of a DAG



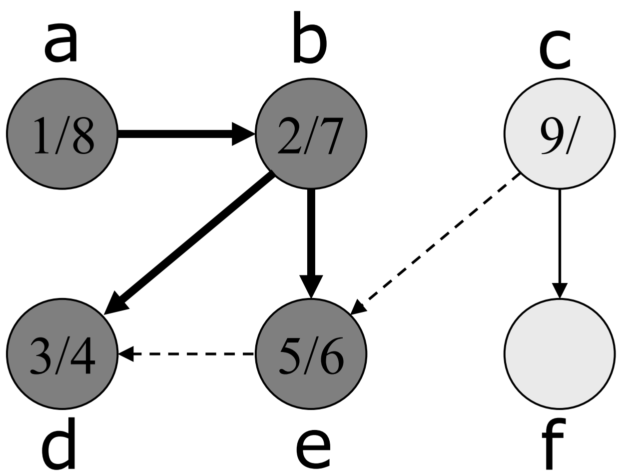
a	8
b	7
e	6
d	4

Computing a Topological Sort of a DAG



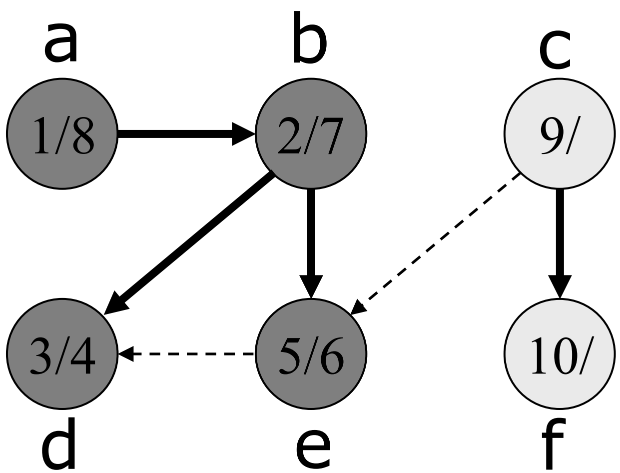
a	8
b	7
e	6
d	4

Computing a Topological Sort of a DAG



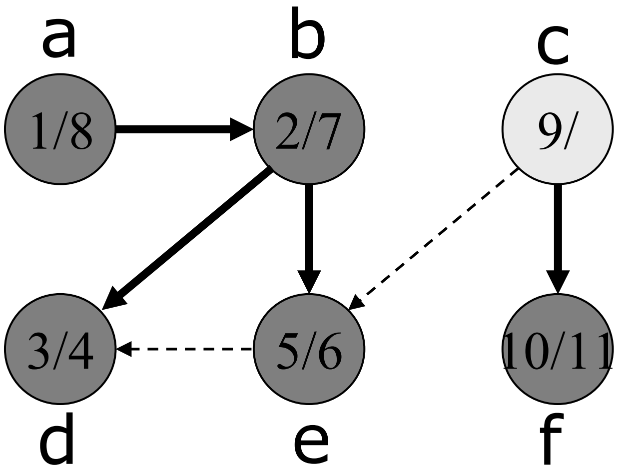
a	8
b	7
e	6
d	4

Computing a Topological Sort of a DAG



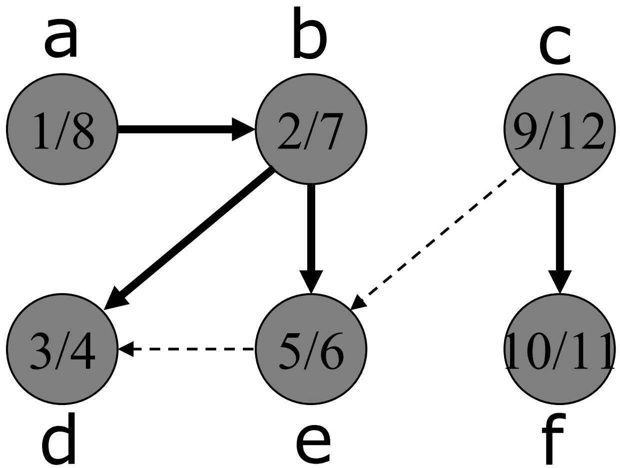
a	8
b	7
e	6
d	4

Computing a Topological Sort of a DAG



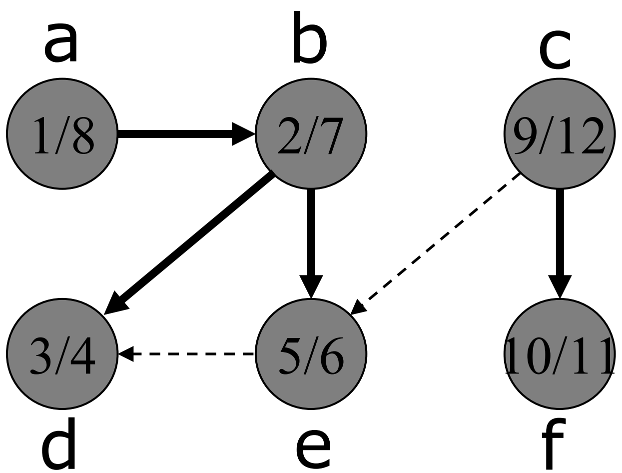
f	11
a	8
b	7
e	6
d	4

Computing a Topological Sort of a DAG



c	12
f	11
a	8
b	7
e	6
d	4

Computing a Topological Sort of a DAG



c, f, a, b, e, d

c	12
f	11
a	8
b	7
e	6
d	4

Summary

- Cycles in a Directed Graph
  - Can be detected in $\Theta(n+m)$ time
- Directed Acyclic Graph (DAG)
  - Can be detected in $\Theta(n+m)$ time
- Topological Sort of a Directed Acyclic Graph
  - Can be computed in $\Theta(n+m)$ time

Ira A. Fulton Schools of
Engineering
Arizona State University