Advanced Algorithms Analysis and Design

By

Nazir Ahmad Zafar

Lecture No 30

Proof (White Path Theorem)

Applications of Depth First Search

Algorithm: Depth First Search

DFS(G)

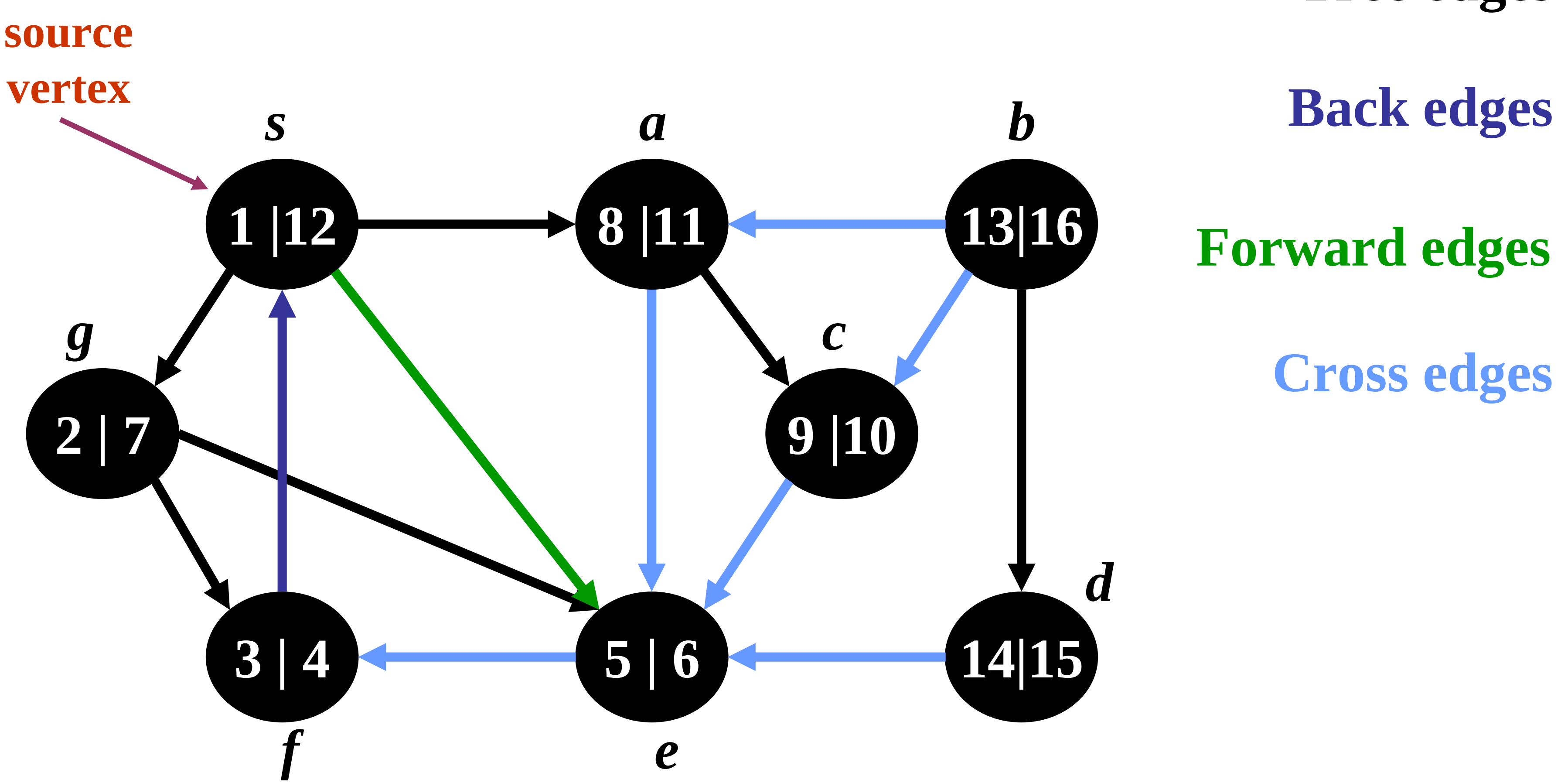
for each vertex $u \in V[G]$
do $color[u] \leftarrow$ WHITE
    $\pi[u] \leftarrow$ NIL
$time \leftarrow 0$
for each vertex $u \in V[G]$
    do if $color[u] =$ WHITE
        then DFS-Visit (u)

DFS-Visit(u)

$color[u] \leftarrow$ GRAY
$time \leftarrow time + 1$
$d[u] \leftarrow time$
for each $v \in Adj[u]$
    do if $color[v] =$ WHITE
        then $\pi[v] \leftarrow u$
            DFS-Visit (v)
$color[u] \leftarrow$ BLACK
$f[u] \leftarrow time \leftarrow time + 1$

Total Running Time = Θ (V + E)

Classification of Edges



Theorem : White-Path Theorem

In a depth-first 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 $d[u]$ that the search discovers u, vertex v can be reached from u along a path consisting entirely of white vertices.

Proof

- Assume that v is a descendant of u.
- Let w be any vertex on the path between u and v in depth-first tree, so that w is a descendant of u.

Theorem:White-Path Theorem (Cont.)

– As $d[u] < d[w]$, and so w is white at time $d[u]$.

⇐: Second part is proved by contradiction

– Suppose that vertex v is reachable from u along a path of white vertices at time d[u], but v does not become a descendant of u in the depth-first tree.

– Without loss of generality, assume that every other vertex along the path becomes a descendant of *u*.

– (Otherwise, let *v* be the closest vertex to *u* along the path that doesn't become a descendant of *u*.)

Theorem : White-Path Theorem (Cont..)

- Let w be predecessor of v in the path, so that w is a descendant of u (w, u may be same) by Corollary above

  $f[w] \leq f[u]$. (1)
- Note v must be discovered after u is discovered,

  $d[u] < d[v]$ (2)
- but v must be discovered before w is finished.

  $d[v] < f[w]$ (3)
- Therefore, by (1), (2) and (3)

  $d[u] < d[v] < f[w] \leq f[u]$.
- Above Theorem implies that interval $[d[v], f[v]]$ is contained entirely within interval $[d[u], f[u]]$.
- By Corollary above, v must be a descendant of u.

Topological Sort

Topological Sort

- A Topological Sort of a directed acyclic graph, or 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.

- It is ordering of its vertices along a horizontal line so that all directed edges go from left to right

- The depth-first search can be used to perform a topological sort of a dag.

Algorithm

Algorithm : Topological Sort

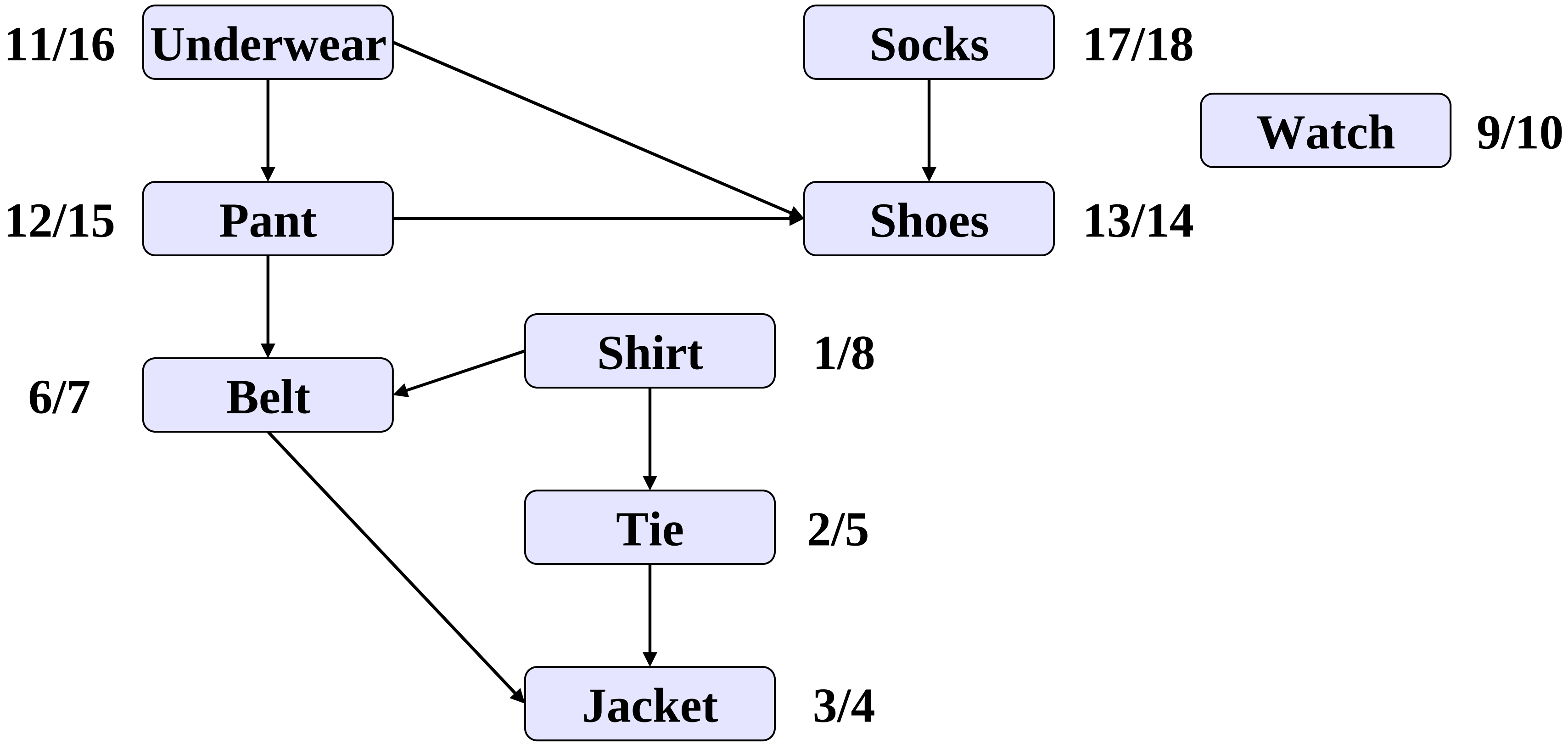
TOPOLOGICAL-SORT (*G*)

1. Call DFS(G) to compute $f\,[v]$ of each vertex $v \in V$.
2. Set an empty linked list $L = \emptyset$.
3. When a vertex v is colored black, assign it $f\,(v)$.
4. Insert v onto the front of the linked list, $L = \{v\}.L$.
5. **return** the linked list.
6. The rank of each node is its position in the linked list started from the head of the list.

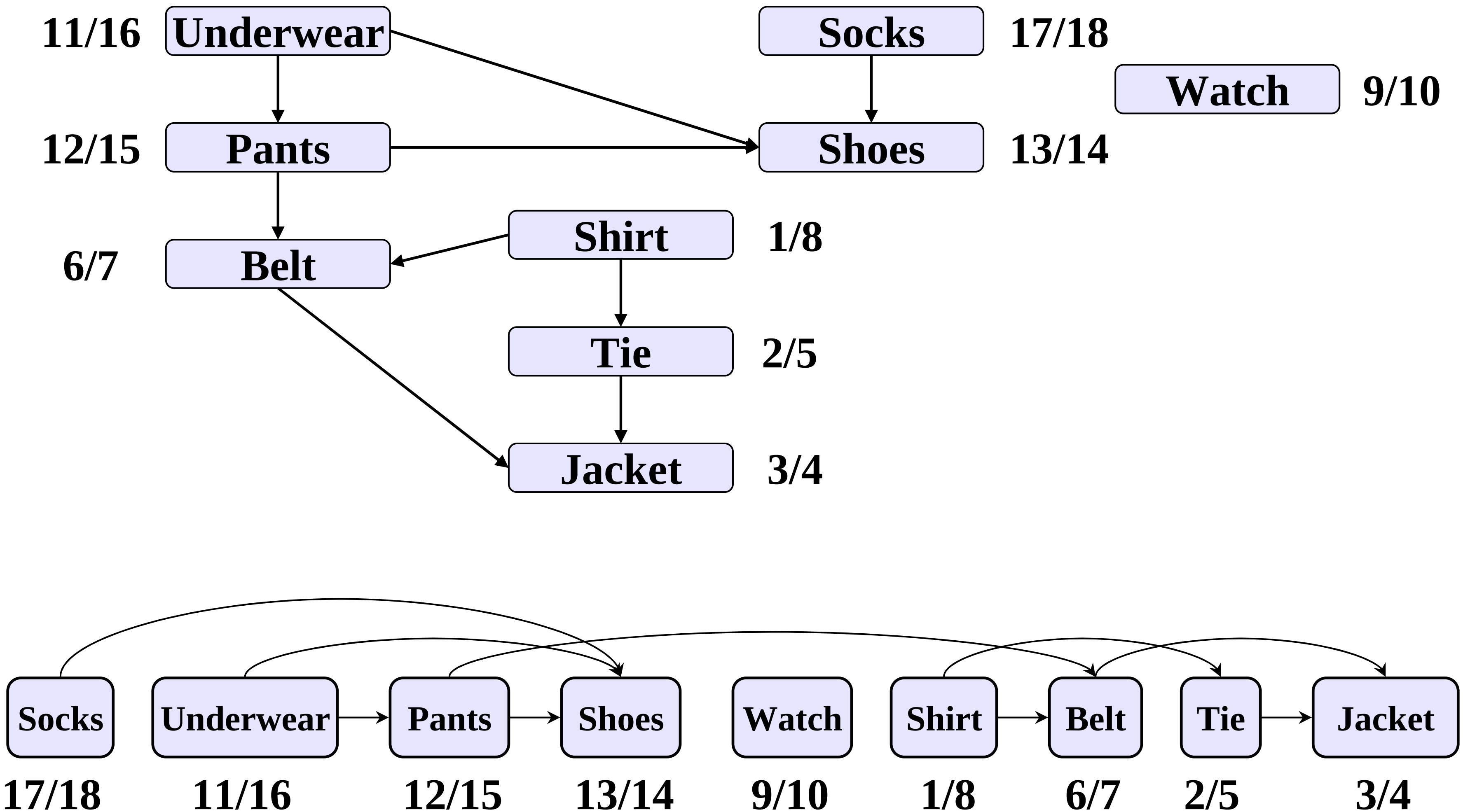
Total Running Time = Θ (V + E)

Example

Example : Topological Sort



Ordering w. r. t. Finishing Time: Topological Sort



Lemma

Lemma

A directed graph *G* is acyclic if and only if a depth-first search of *G* yields no back edges.

Proof

$\Rightarrow$: G is acyclic.

- Suppose that there is a back edge (*u*, *v*).
- Then, vertex *v* is an ancestor of *u* in DF forest.
- There is thus a path from *v* to *u* in *G*, and the back edge (*u*, *v*) completes a cycle.
- G is cyclic and hence a contradiction,
- Our supposition is wrong and
- Hence G has no back edge

Lemma (Cont.)

$\Leftarrow$: If DFS yields no back edges G has no cycle

We prove it by contra positive

- We prove that if *G* contains a cycle *c the* DFS of *G* yields a back edge.
- Let G has a cycle c.
- Let *v* be the first vertex to be discovered in *c*, and let (*u*, *v*) be the preceding edge in *c*.
- At time *d*[*v*], the vertices of *c* form a path of white vertices from *v* to *u*.
- By the white-path theorem, vertex *u* becomes a descendant of *v* in the depth-first forest. Therefore, (*u*, *v*) is a back edge.

Theorem

Theorem

TOPOLOGICAL-SORT (G) produces a topological sort of a directed acyclic graph G .

Proof

- Let DFS is run on G to determine finishing times.
- It sufficient to show that for any two distinct u, $v \in V$, if there is an edge in G from u to v, then $f[v] < f[u]$
- Consider any edge (u, v) explored by DFS(G).
- When (u, v) is explored, *v is* gray, white or black

Case 1

- *v* is gray. v is ancestor of *u.* (u, v) would be a back edge. It contradicts the above Lemma.

Theorem (Cont.)

Case 2

- If v is white, it becomes a descendant of u, and hence $f[v] < f[u]$.

Case 3

- If v is black, it has already been finished, so that $f[v]$ has already been set.
- Because we are still exploring from u, we have yet to assign a timestamp to $f[u]$ to u, and so once we do, we will have $f[v] < f[u]$ as well.

Thus, for any edge (u, v) in the dag, we have $f[v] < f[u]$. It proves the theorem.

SCC

Strongly Connected Components

Strongly Connected Components

- A **strongly connected component** of a directed graph $G = (V, E)$ is a maximal set of vertices $C \subseteq V$ such that for every pair of vertices u and v in C, we have
  - $u \leadsto v$, v *is* reachable from u.
  - $v \leadsto u$; u *is* reachable from v.

- The depth-first search can be used in decomposing a directed graph into its strongly connected components.

Notations

Transpose of a Graph

- The **strongly connected components** of a graph $G = (V, E)$ uses the transpose of G, which is defined as

$$G^T = (V, E^T), \text{ where}$$

$$E^T = \{(u, v) : (v, u) \in E\}$$

E^T consists of the edges of G with reversed directions.

- G and G^T have exactly the same strongly connected components
  - u and v are reachable from each other in G if and only if they are reachable from each other in G^T.

Algorithm

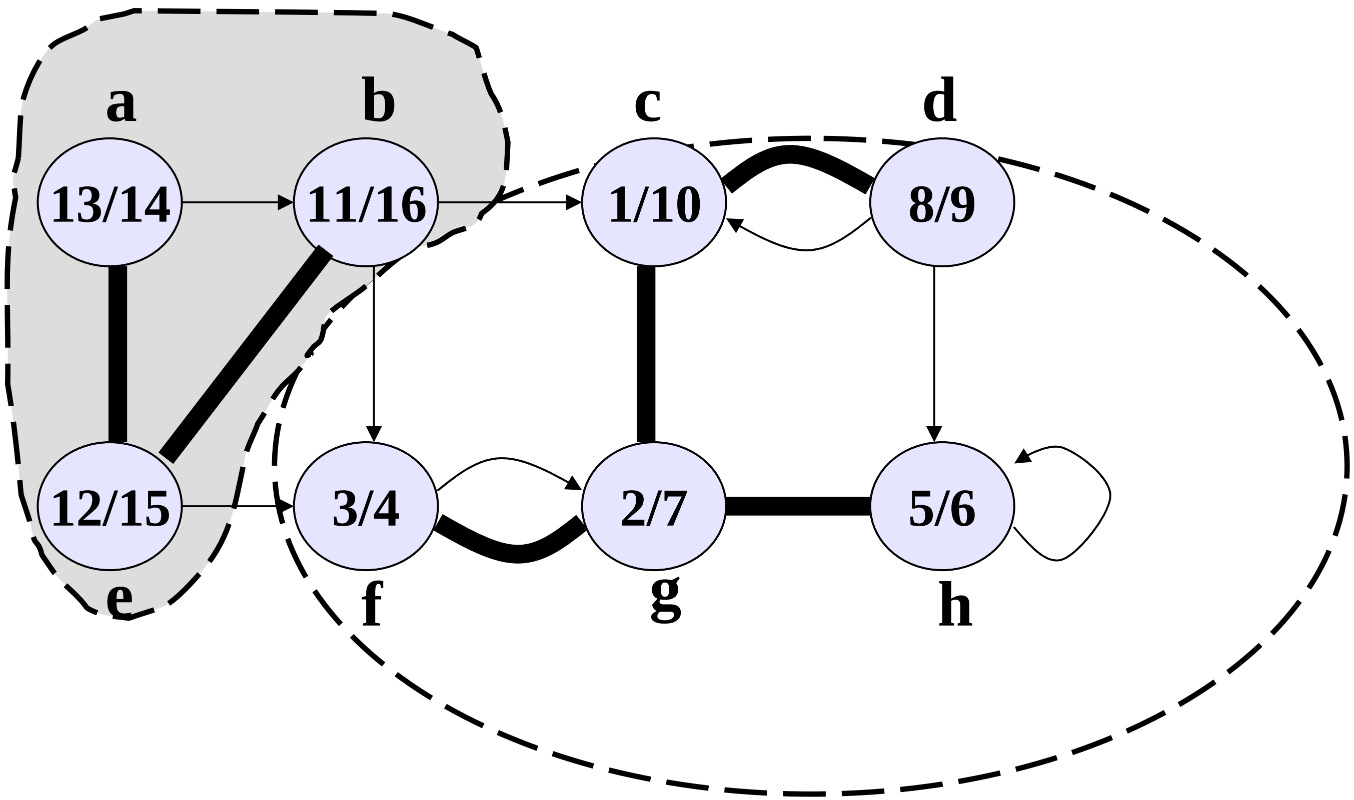
STRONGLY-CONNECTED-COMPONENTS (*G*)

1 call DFS(*G*), to compute the finish time $f[u]$ of each vertex u
2 compute G^T.
3 call DFS (G^T), but in the main loop of DFS, consider the vertices in order of decreasing $f[u]$. (as computed in line 1)
4 Output of the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component.

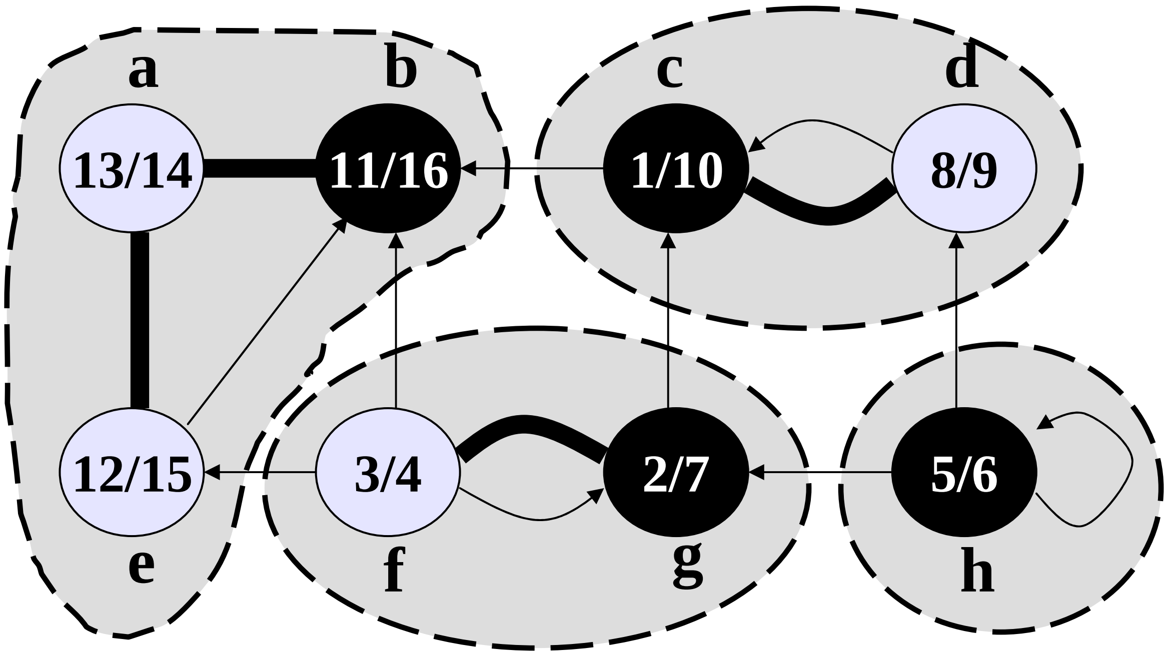
Total Running Time = Θ (V + E)

Example

Strongly Connected Components



Strongly Connected Components



Component Graph