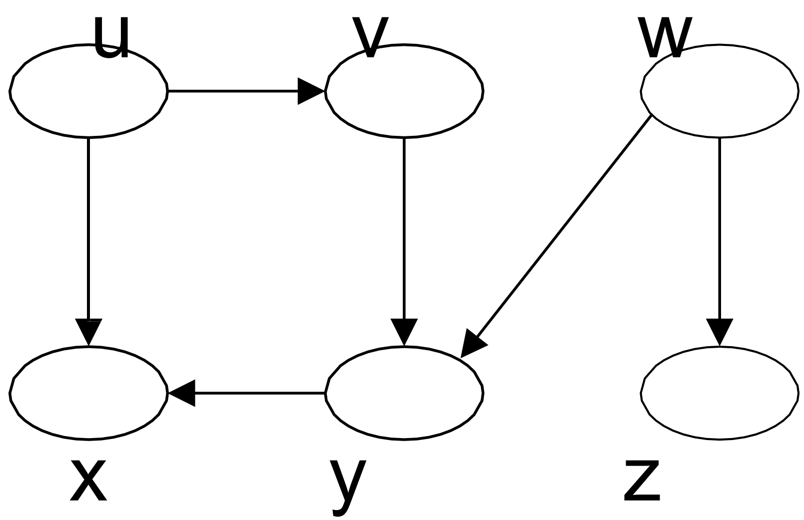
Biconnected components

Topological Sort

- *Topological sort* of a DAG (Directed Acyclic Graph):
  - Linear ordering of all vertices in a DAG G such that vertex *u* comes before vertex *v* if there is an edge $(u, v) \in G$
  - This property is important for a class of *scheduling* problems

Example – Topological Sorting



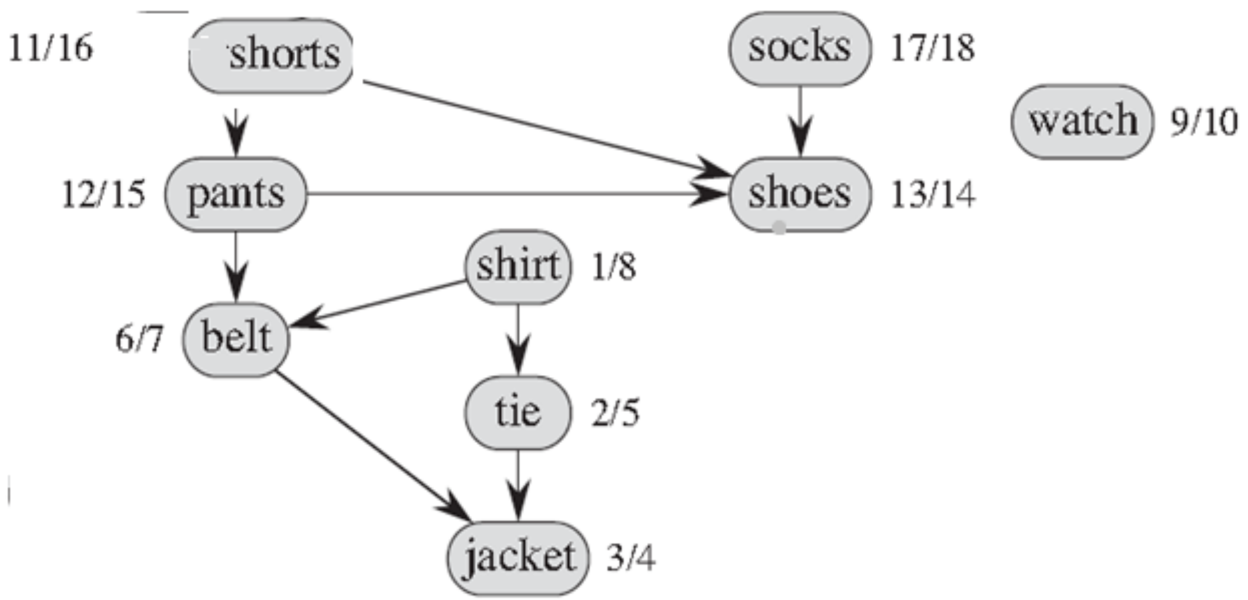
- There can be several orderings of the vertices that fulfill the topological sorting condition:
  - u, v, w, y, x, z
  - w, z, u, v, y, x
  - w, u, v, y, x, z
  - …

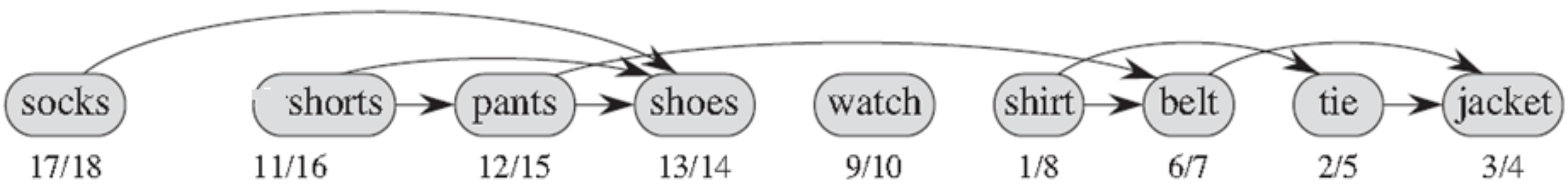
Topological Sorting

- *Algorithm principle:*
  1. *Call DFS to compute finishing time v.f for every vertex*
  2. *As every vertex is finished (BLACK) insert it onto the front of a linked list*
  3. *Return the list as the linear ordering of vertices*

- Time: O(V+E)

Using DFS for Topological Sorting





Strongly Connected Components

- A strongly connected component of a directed graph G=(V,E) is a maximal set of vertices C such that for every pair of vertices u and v in C, both vertices u and v are reachable from each other.

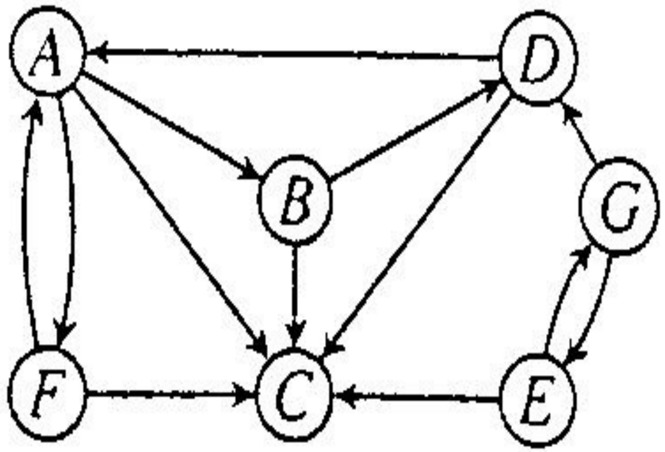
- KOSARAJU ALGORITHM

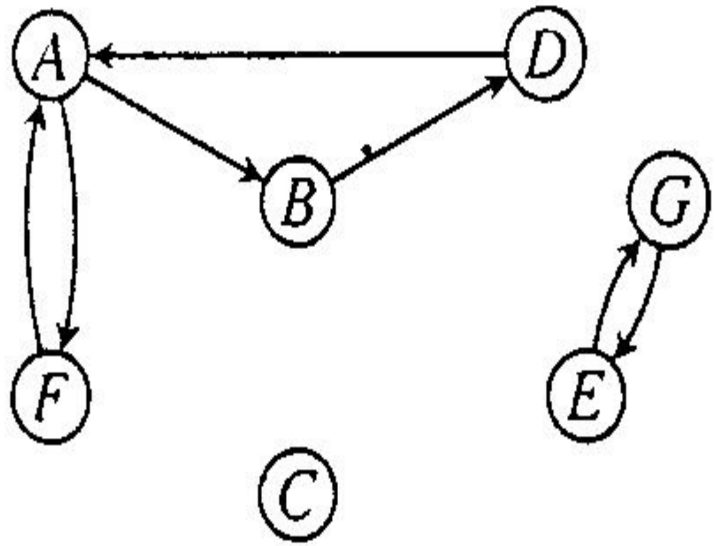
Strongly Connected Components of a Digraph

Strongly connected:

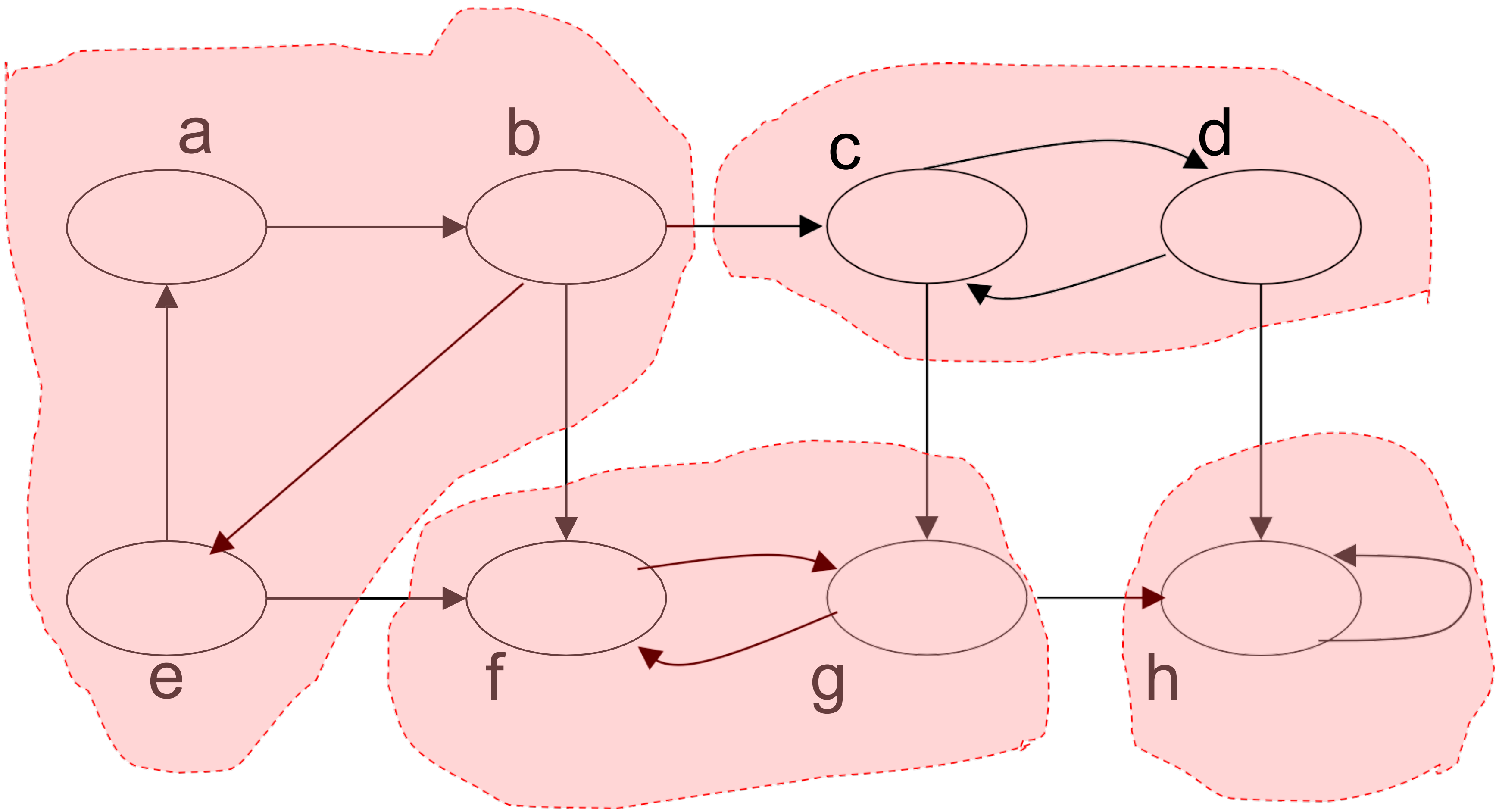
A directed graph is strongly connected if and only if, for each pair of vertices v and w, there is a path from v to w.

Strongly connected component:

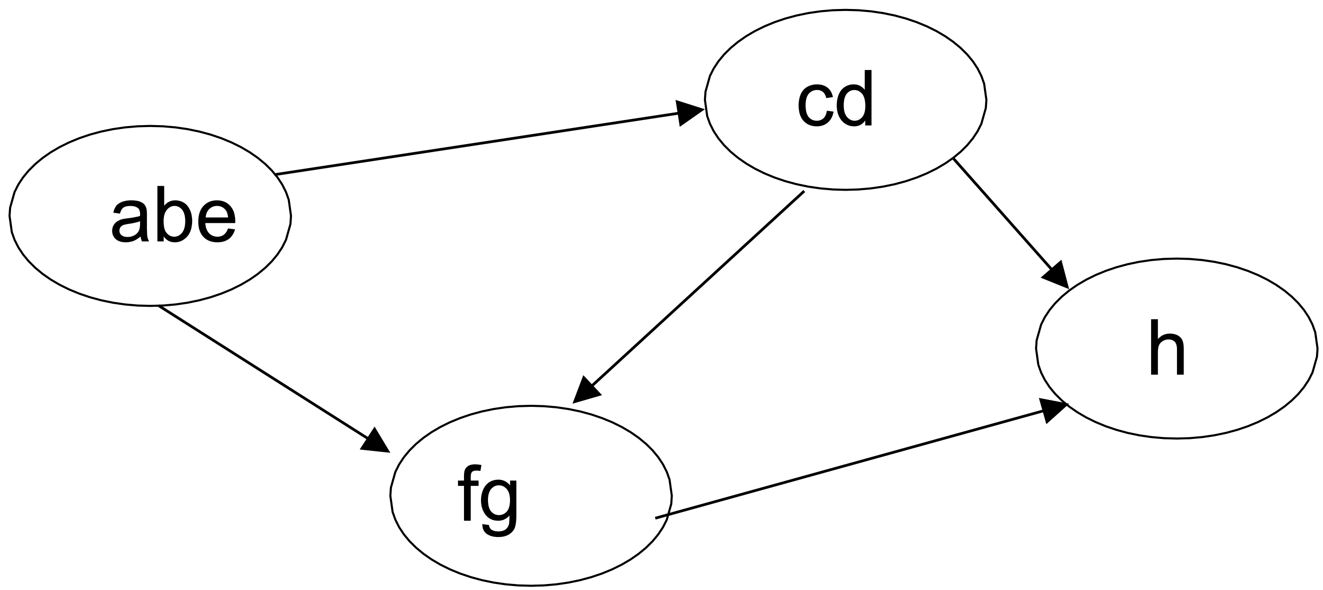




Strongly connected components - Example



Strongly connected components – Example – The Component Graph



The Component Graph results by collapsing each strong component into a single vertex

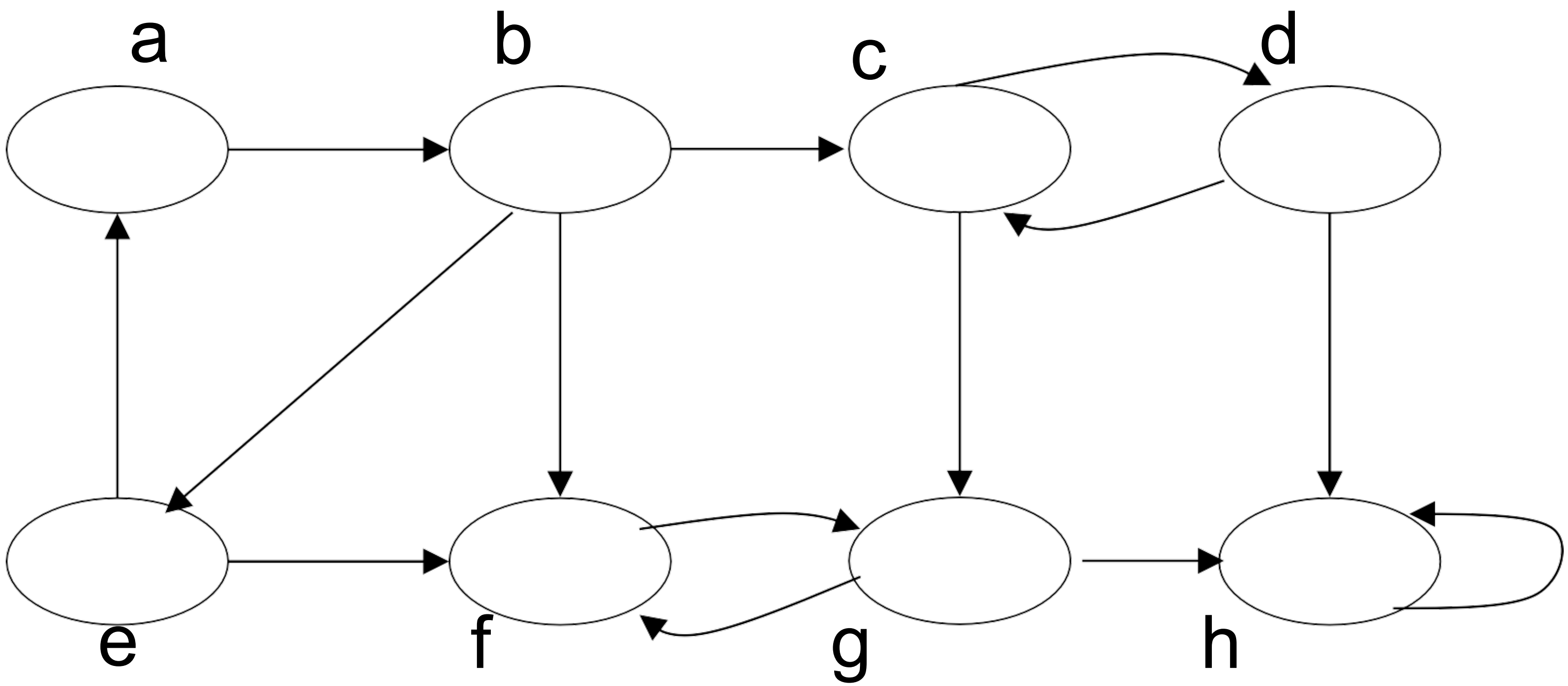
Strongly connected components

- *Strongly connected components of a directed graph G*
- ***Algorithm principle:***

1. ***Call DFS(G)*** *to compute finishing times u.f for every vertex u*
2. *Compute* ***Graph Transpose (GT)***
3. ***Call DFS(GT),*** *but in the main loop of DFS, consider the vertices* ***in order of decreasing u.f*** *as computed in step 1*
4. *Output the vertices of each DFS-tree formed in step 3 as the vertices of a strongly connected component. (Note: When there is no reachability, we make a manual transition. Each manual transition tell us that a new component is starting.)*

Strongly connected components - Example

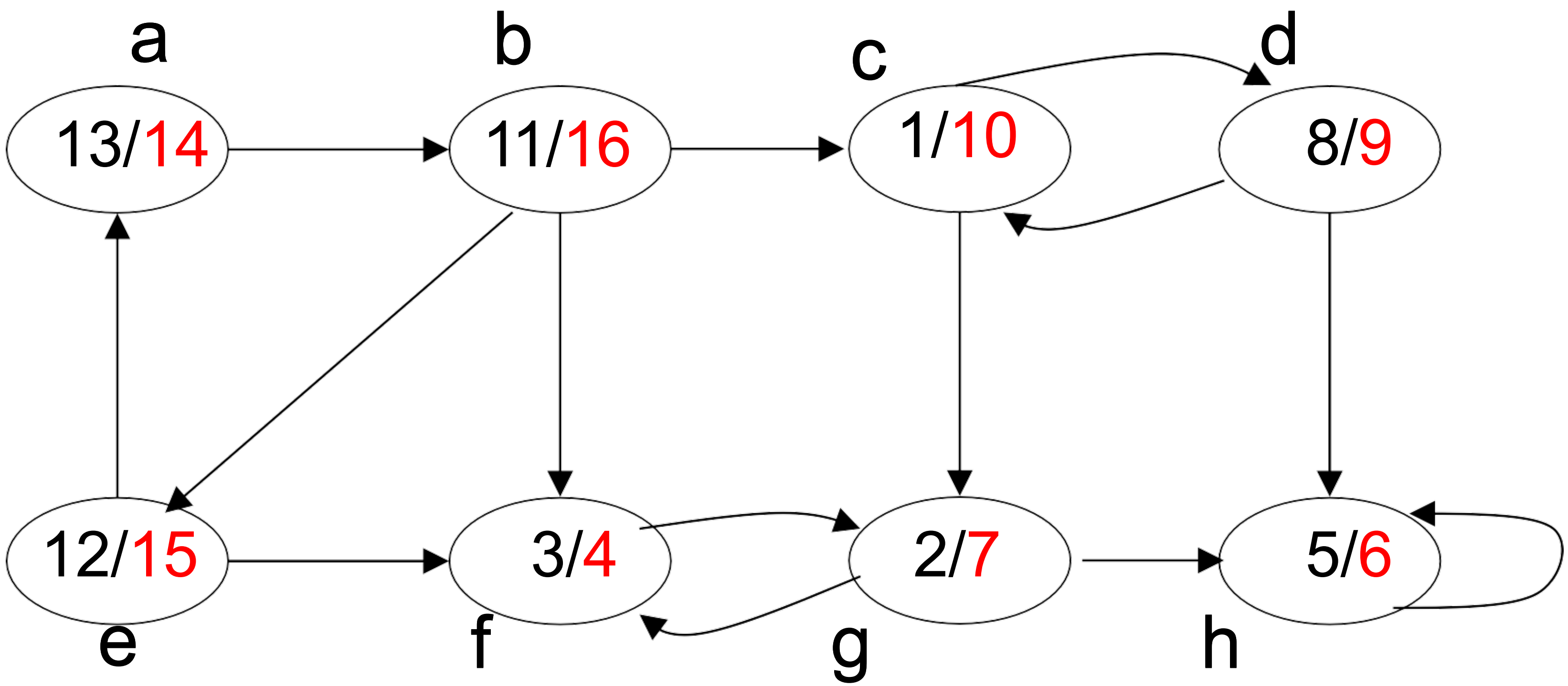
Step1: call DFS(G), compute u.f for all u. Say I start with ‘c’



Node	a	b	c	d	e	f	g	h
Vis	F	F	F	F	F	F	F	F

Strongly connected components - Example

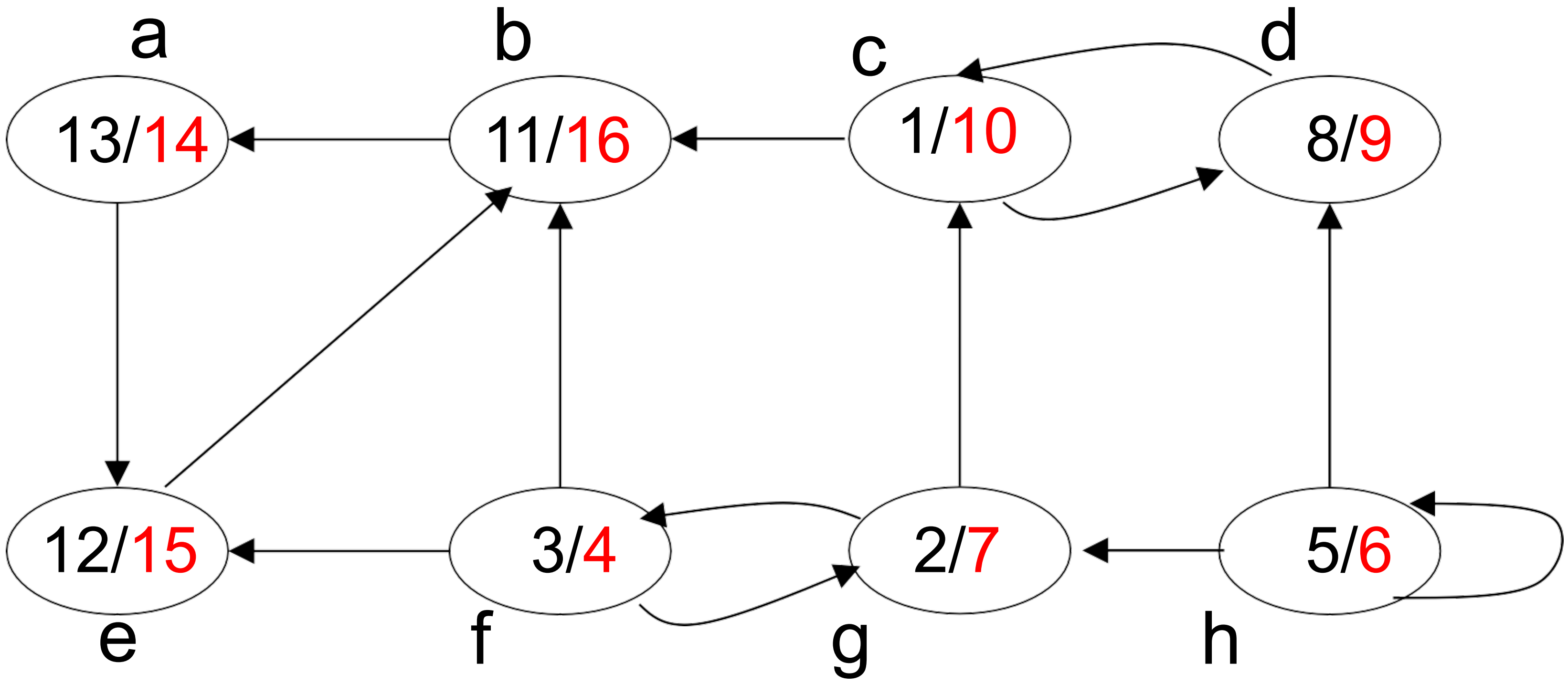
Step1: call DFS(G), compute u.f for all u. Say I start with ‘c’



Node	a	b	c	d	e	f	g	h
Vis	F	F	F	F	F	F	F	F

Strongly connected components - Example

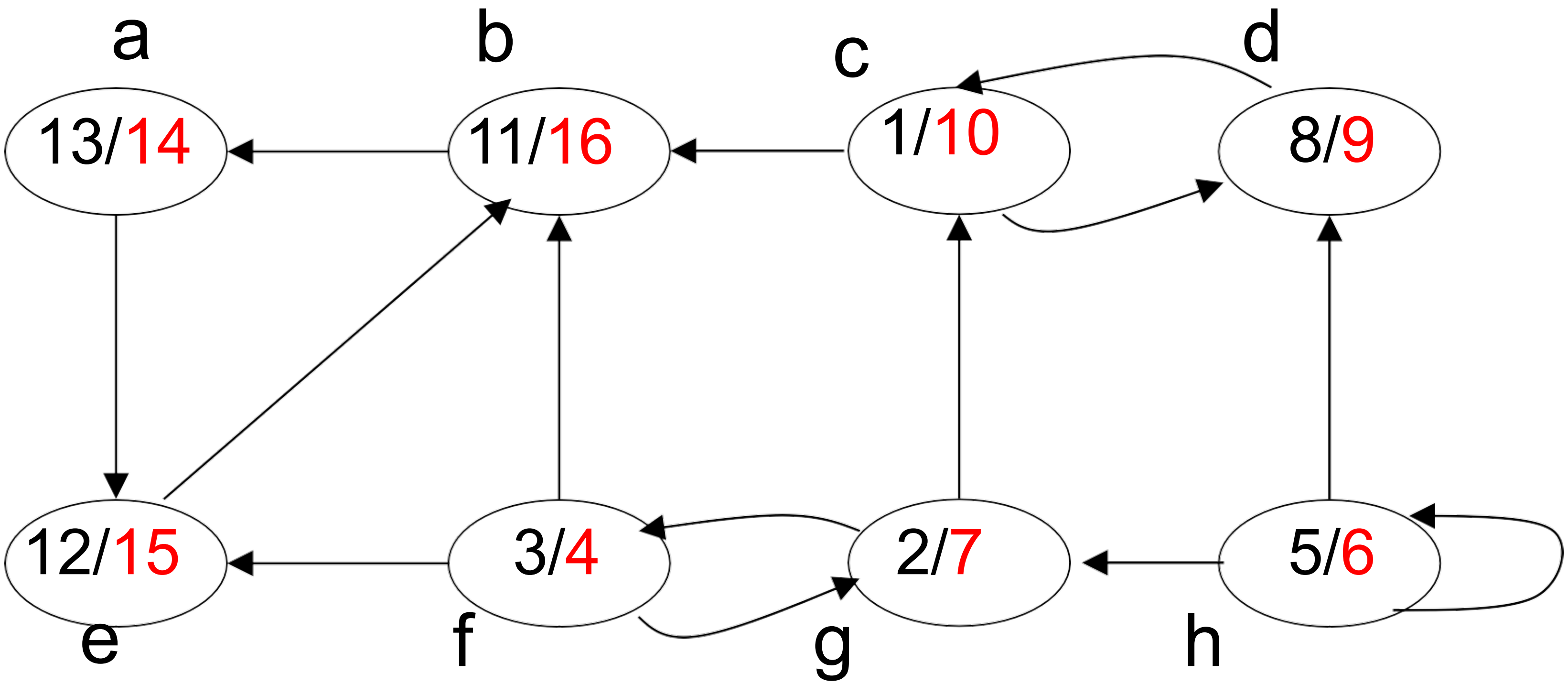
Step2: compute GT (Reverse directions of the edges



Node	a	b	c	d	e	f	g	h
u.f	14	16	10	9	15	4	7	6

Strongly connected components - Example

Step3: call DFS(GT), consider vertices in order of decreasing u.f

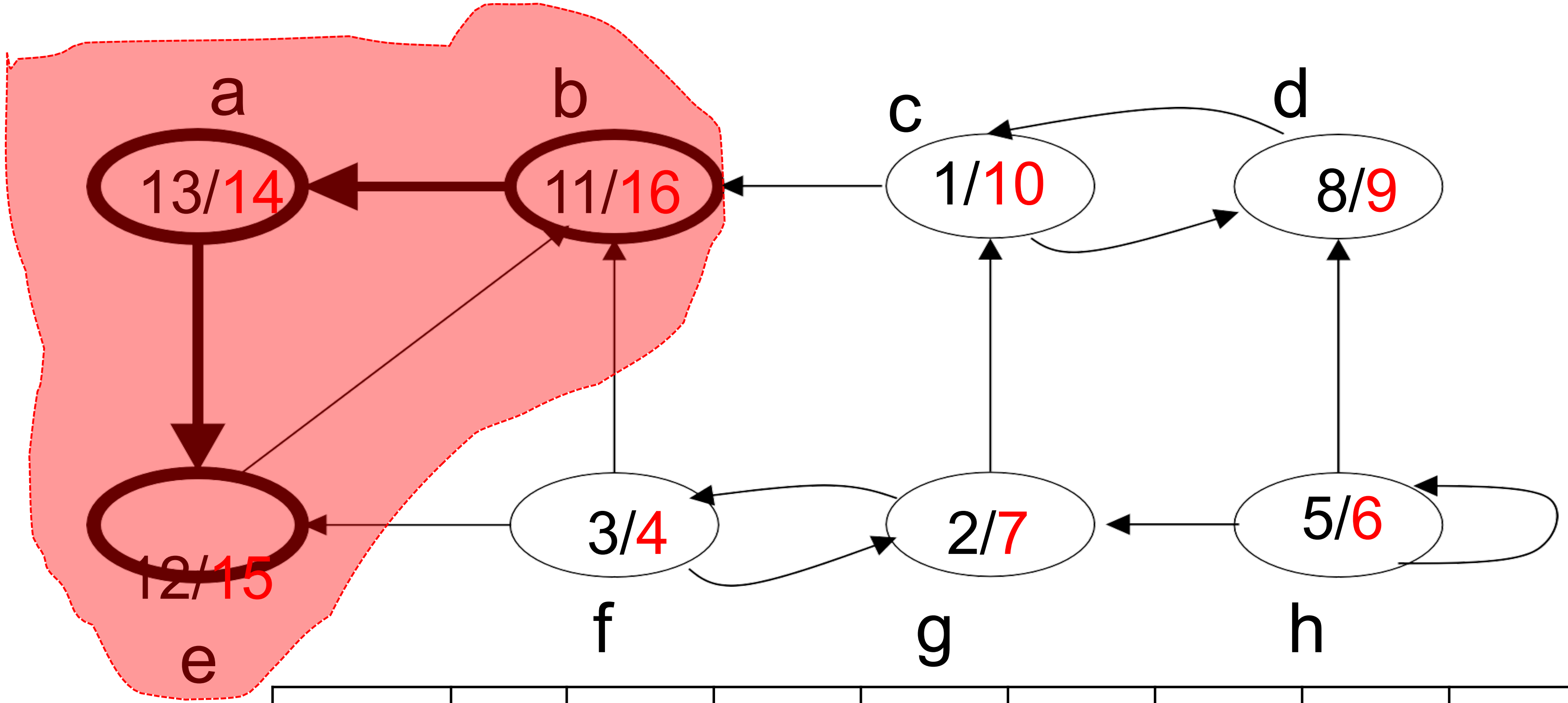


Node	a	b	c	d	e	f	g	h
u.f	14	16	10	9	15	4	7	6

Strongly connected components - Example

Step3: call DFS(GT), consider vertices in order of decreasing u.f

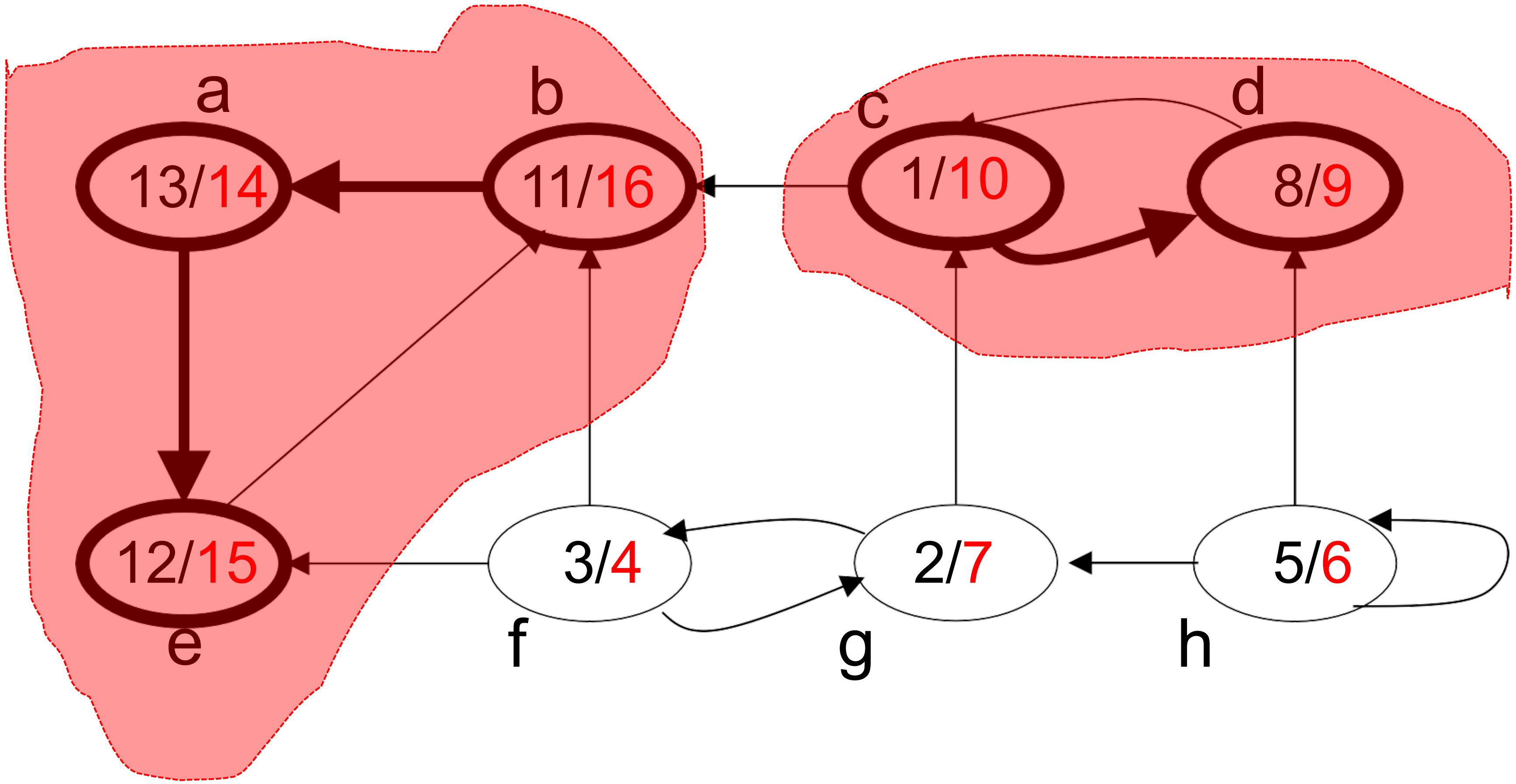
(Highest u.f=16. Start DFS(GT) from b. We can go only to b,a,e.
Then manual transition to next highest u.f id made i.e. 10.
Hence, one component (a,b,e) is obtained.



Node	a	b	c	d	e	f	g	h
u.f	14	16	10	9	15	4	7	6

Strongly connected components - Example

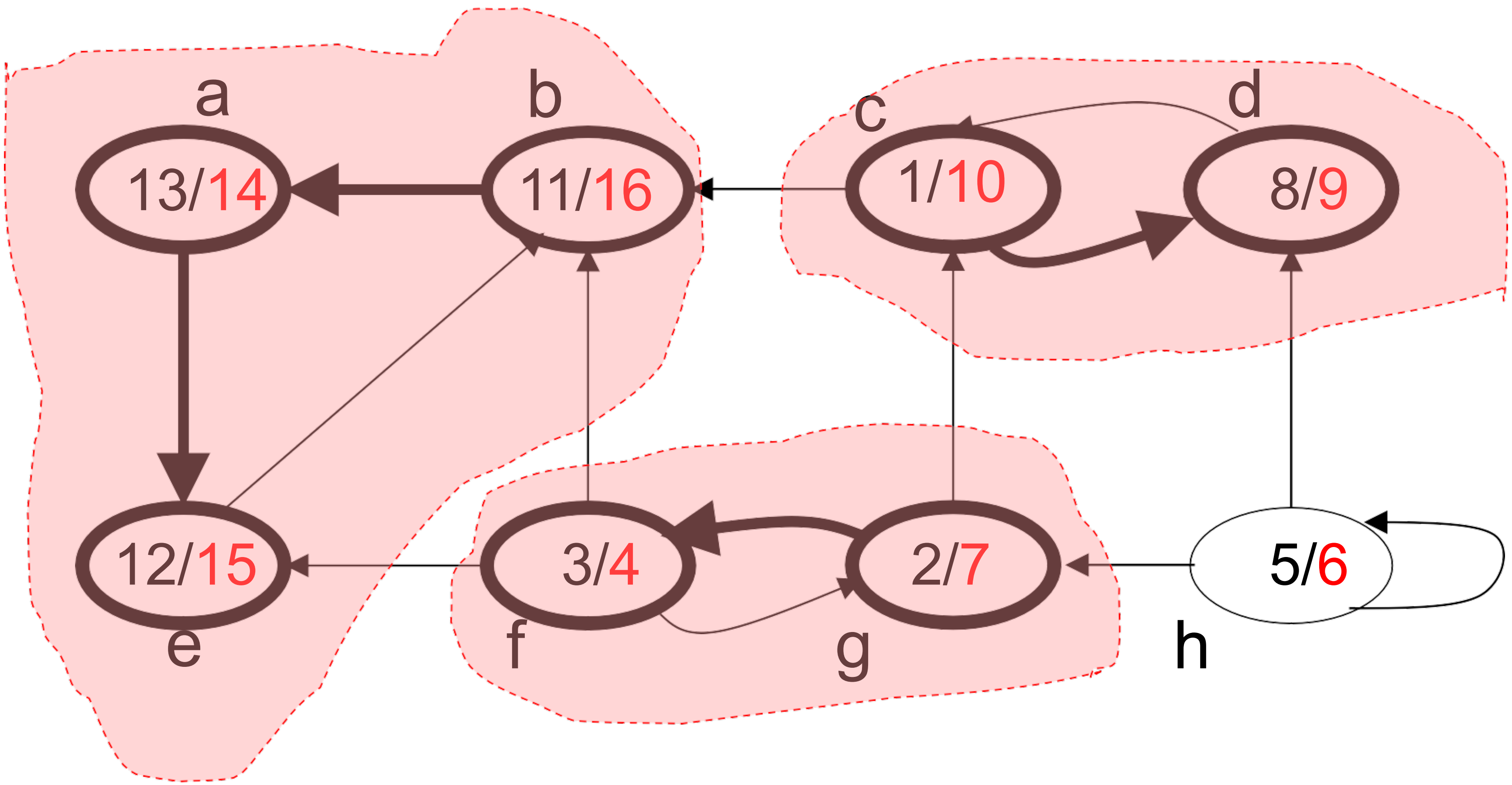
Step3: call DFS(GT), consider vertices in order of decreasing u.f



Node	a	b	c	d	e	f	g	h
u.f	14	16	10	9	15	4	7	6

Strongly connected components - Example

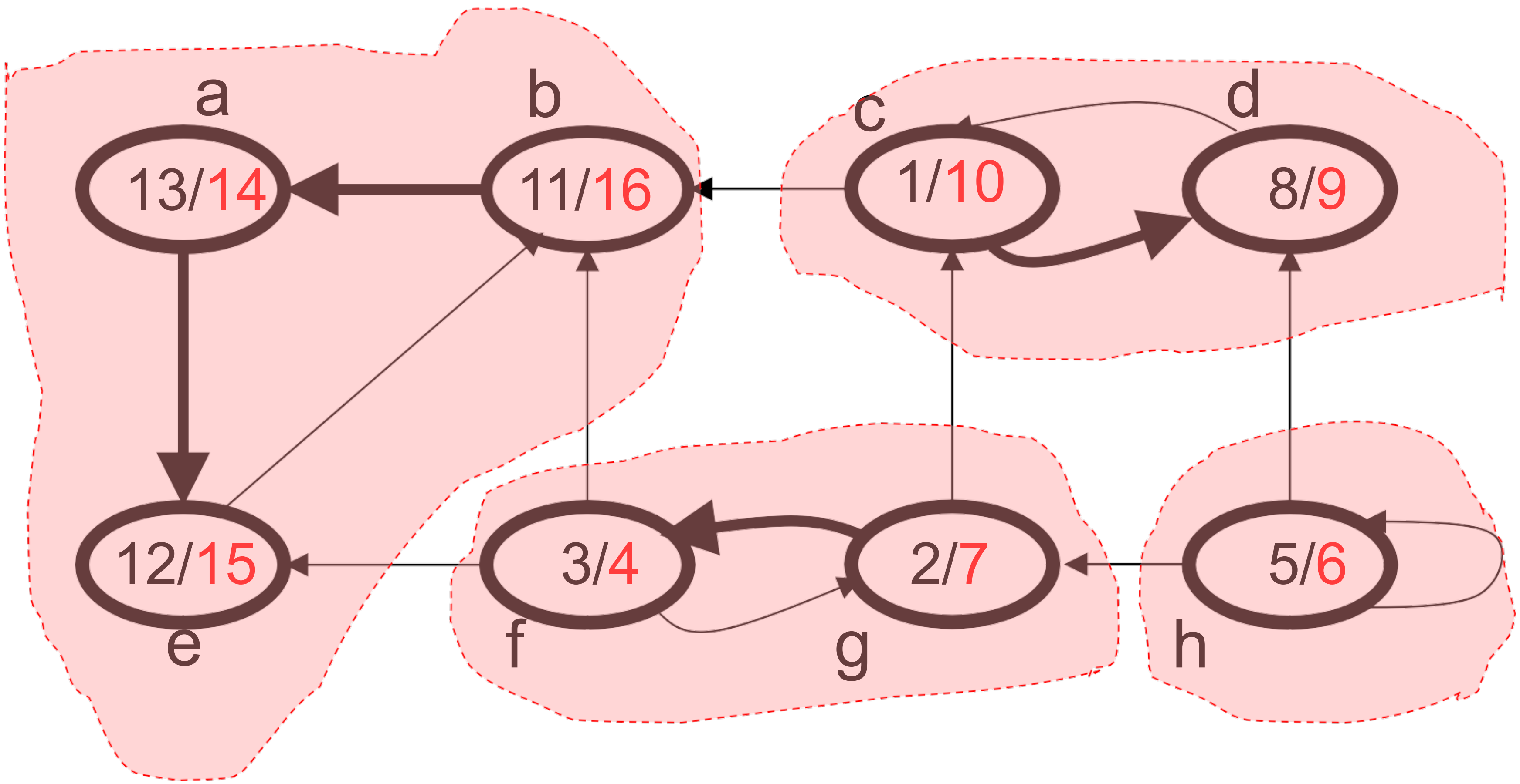
Step3: call DFS(GT), consider vertices in order of decreasing u.f



Node	a	b	c	d	e	f	g	h
u.f	14	16	10	9	15	4	7	6

Strongly connected components - Example

Step3: call DFS(GT), consider vertices in order of decreasing u.f



Node	a	b	c	d	e	f	g	h
u.f	14	16	10	9	15	4	7	6