Algorithms

Lecture Topic: Strongly Connected Components

Anxiao (Andrew) Jiang

Roadmap of this lecture:

1. Application of DFS: Finding “strongly connected components” in a graph.

    1.1 Define the "Strongly Connected Component Problem".

    1.2 Algorithm for finding strongly connected components.

Strongly Connected Components

Definition of Strongly Connected Component:

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 $u, v \in C$, both $u \rightsquigarrow v$ and $v \rightsquigarrow u$, that is, vertices u and v are reachable from each other.

Strongly Connected Components

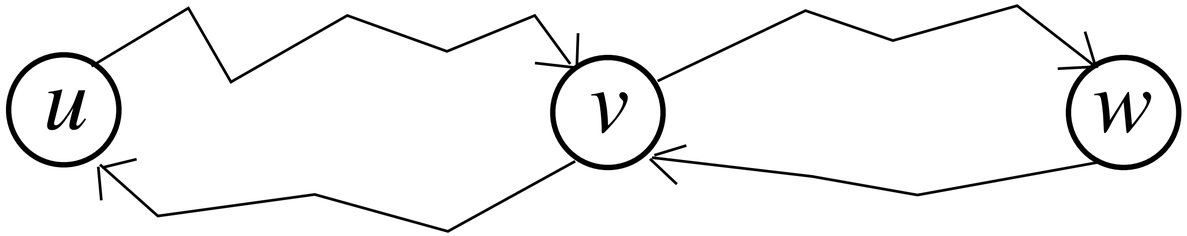
Definition of Strongly Connected Component:

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 $u, v \in C$, both $u \rightsquigarrow v$ and $v \rightsquigarrow u$, that is, vertices u and v are reachable from each other.

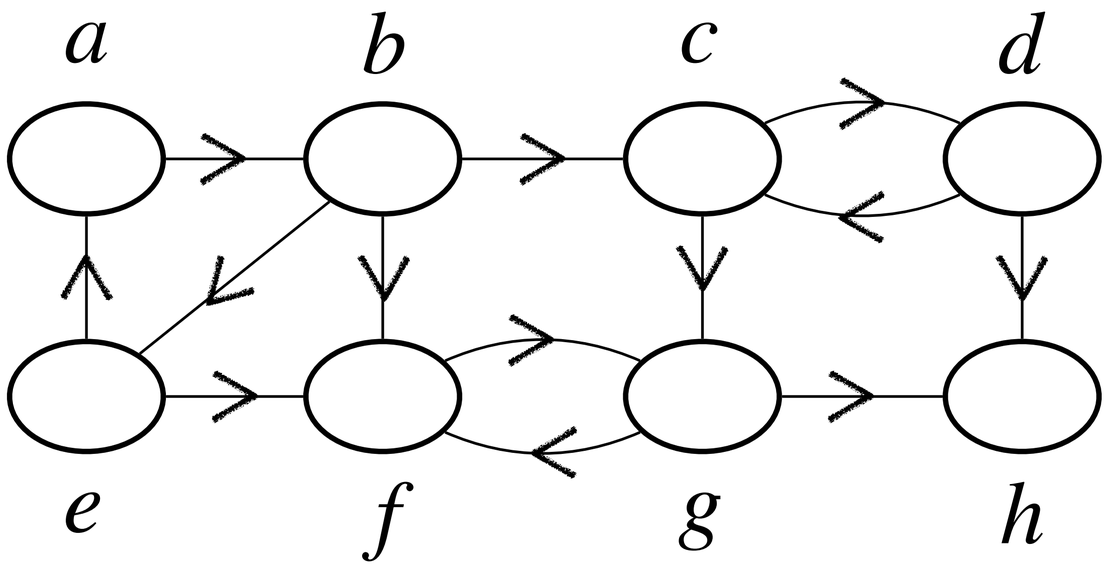
Strong connectivity for vertices is an “equivalence” condition:

If vertices u, v are strongly connected (namely, they can reach each other via paths),

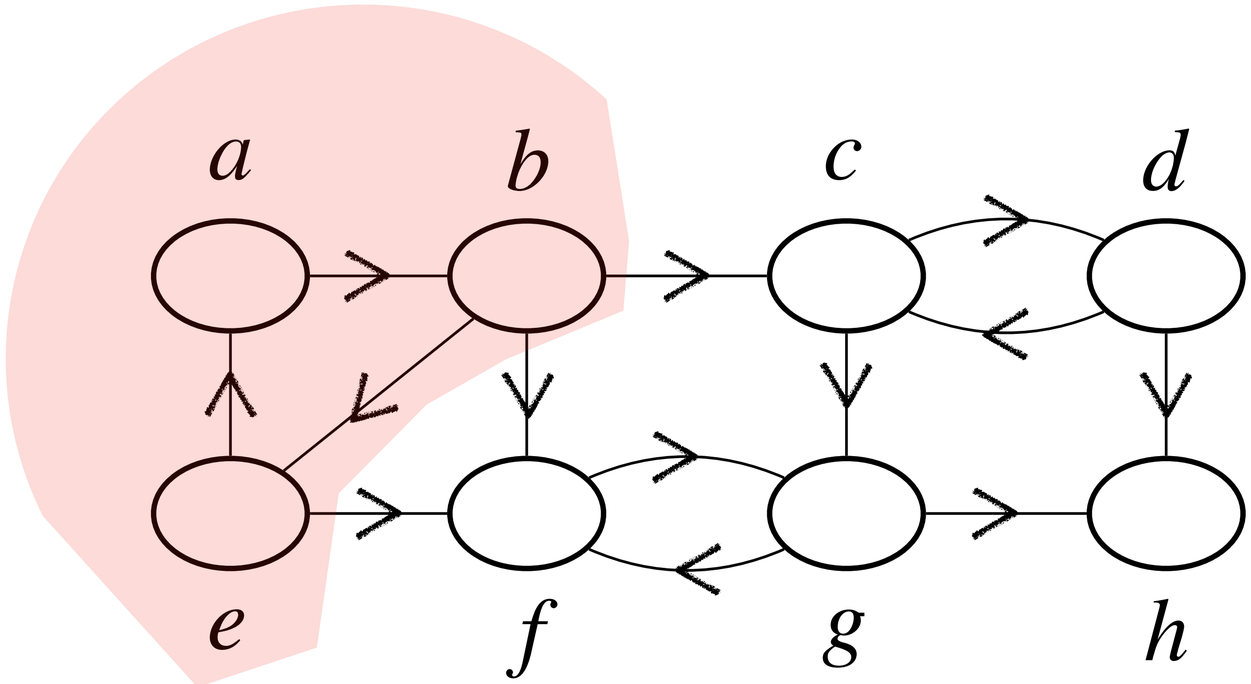
and vertices v, w are strongly connected, then u, w are also strongly connected.



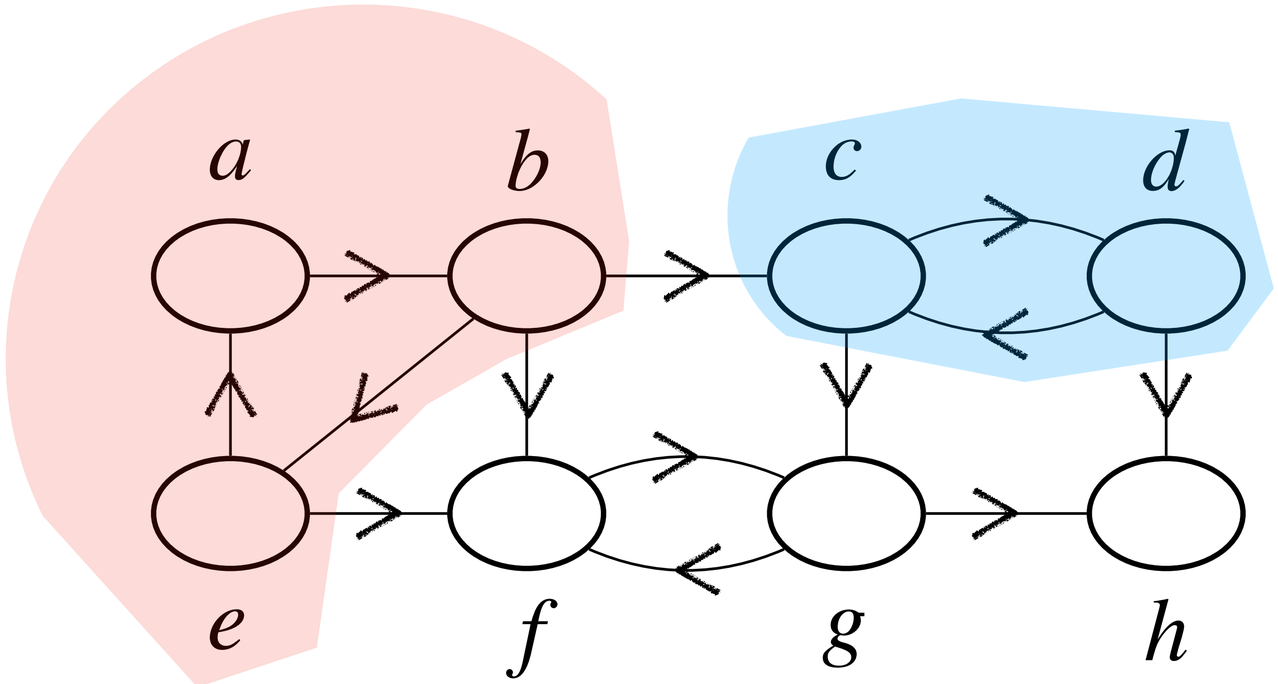
Strongly Connected Components



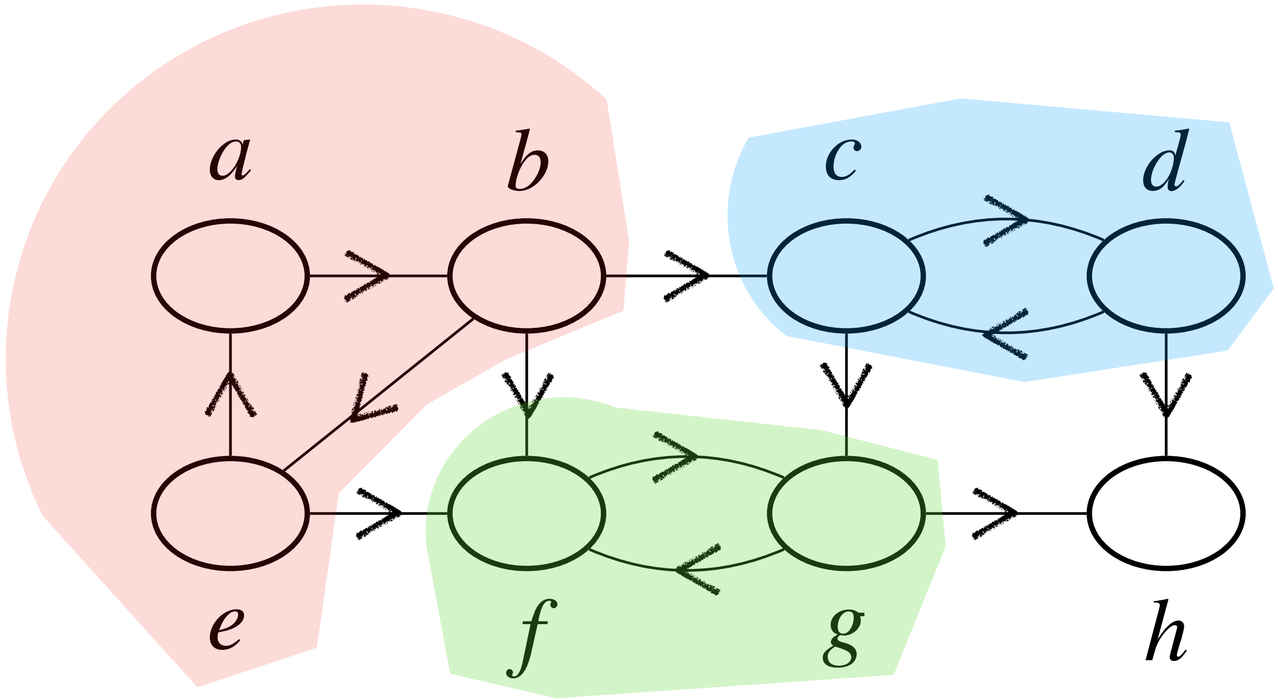
Strongly Connected Components



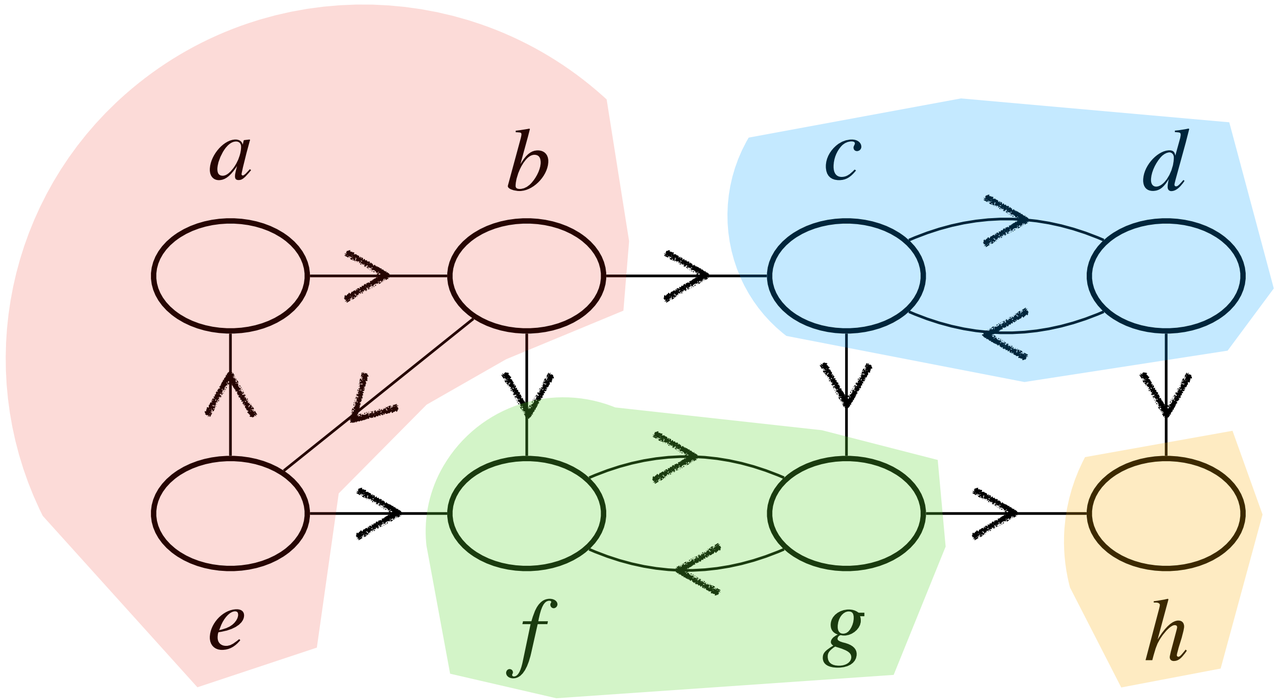
Strongly Connected Components



Strongly Connected Components



Strongly Connected Components



Strongly Connected Component Problem:

Input: A directed graph $G = (V, E)$.

Output: All the strongly connected components in G.

Quiz questions:

1. Given a graph, why are its strongly connected components uniquely determined?

2. What are the applications of finding strongly connected components in a graph?

Roadmap of this lecture:

1. Application of DFS: Finding “strongly connected components” in a graph.

    1.1 Define the "Strongly Connected Component Problem".

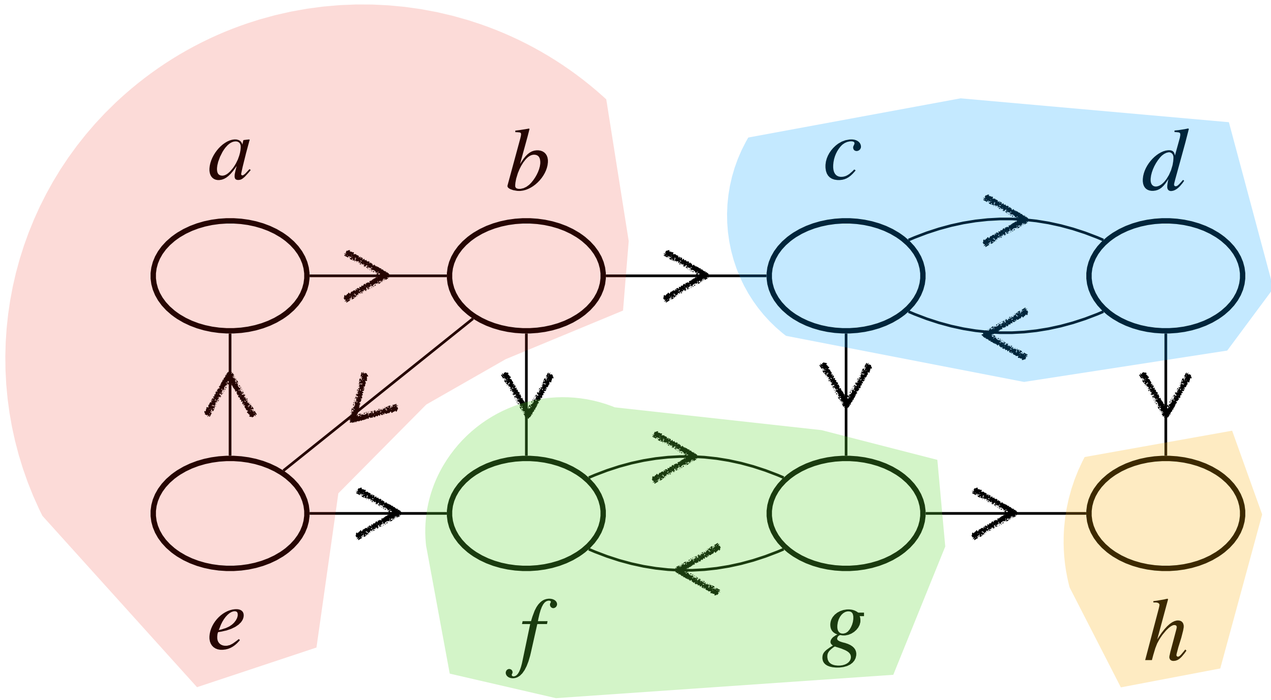
    1.2 Algorithm for finding strongly connected components.

Strongly Connected Component Problem:

Input: A directed graph $G = (V, E)$.

Output: All the strongly connected components in G.

Basic idea: use DFS.



Definition of Transpose of Graph:

Given a directed graph $G = (V, E)$, its transpose is a graph $G^T = (V, E^T)$, where $E^T = \{(u, v) : (v, u) \in E\}$.

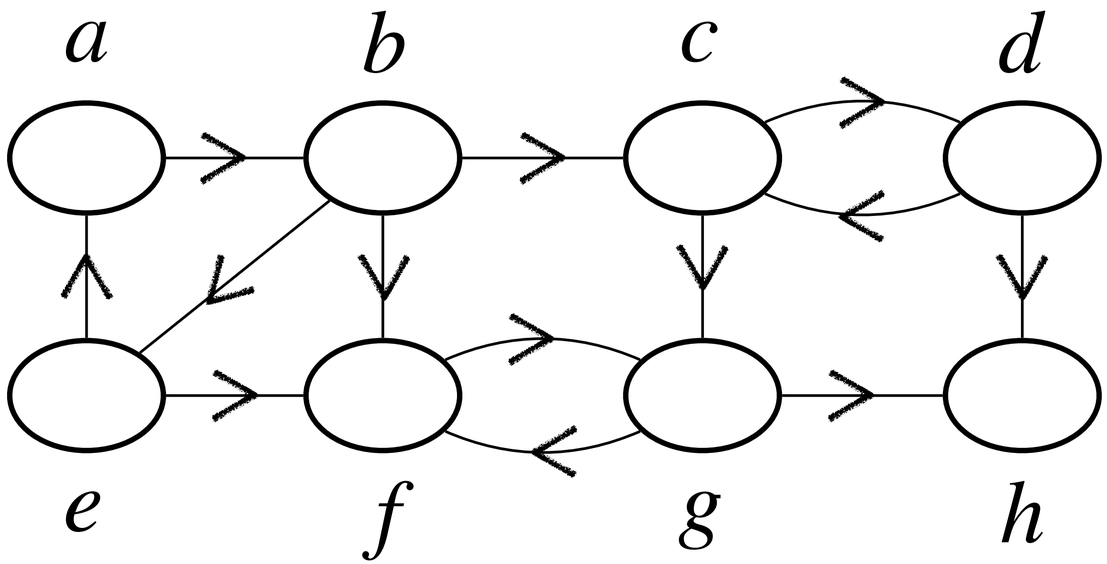
That is, E^T consists of the edges of G with their directions reversed.

Definition of Transpose of Graph:

Given a directed graph $G = (V, E)$, its transpose is a graph $G^T = (V, E^T)$, where $E^T = \{(u, v) : (v, u) \in E\}$.

That is, E^T consists of the edges of G with their directions reversed.

$G = (V, E)$

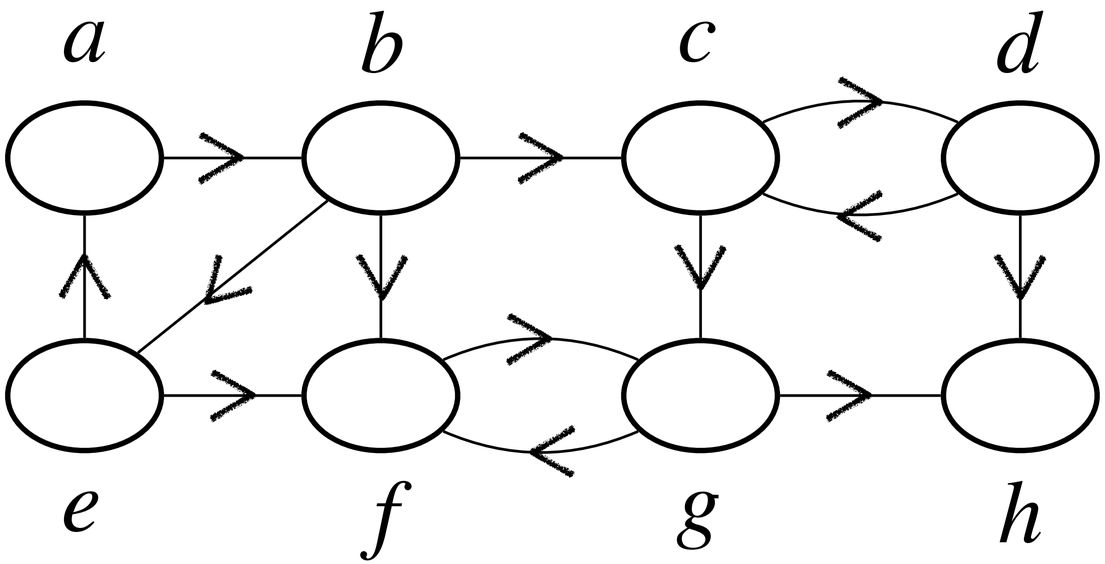


Definition of Transpose of Graph:

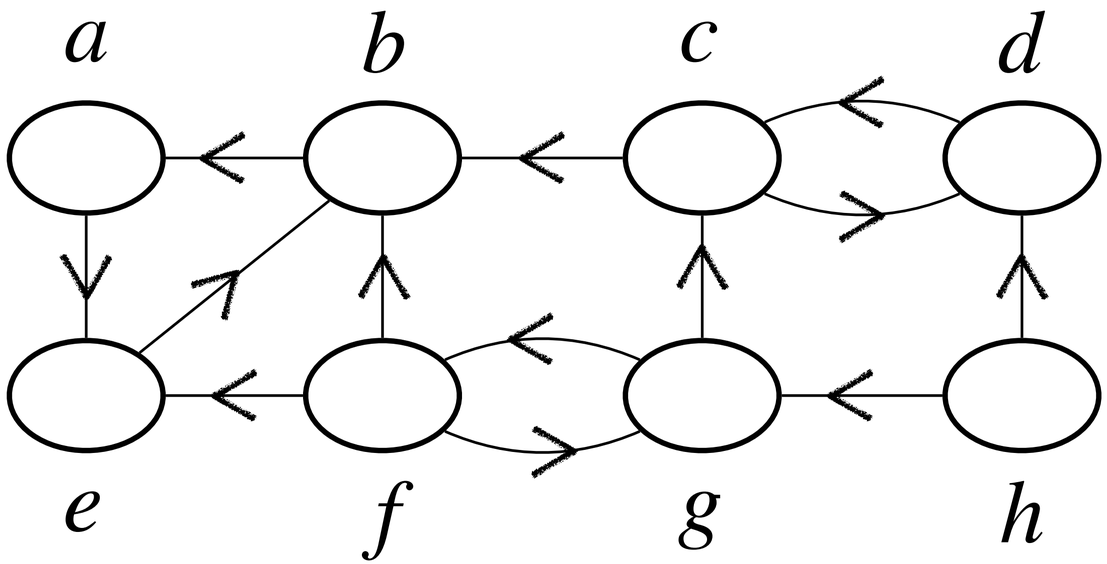
Given a directed graph $G = (V, E)$, its transpose is a graph $G^T = (V, E^T)$, where $E^T = \{(u, v) : (v, u) \in E\}$.

That is, E^T consists of the edges of G with their directions reversed.

$G = (V, E)$



$G^T = (V, E^T)$: transpose of G



Strongly Connected Component (SCC) Problem:

Input: A directed graph $G = (V, E)$.

Output: All the strongly connected components (SCCs) in G.

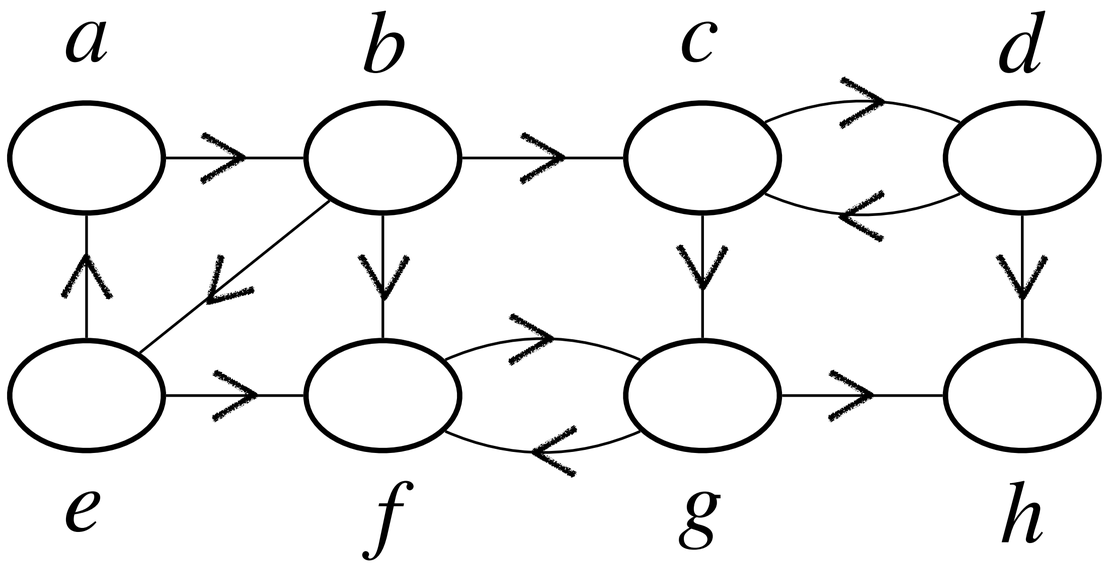
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

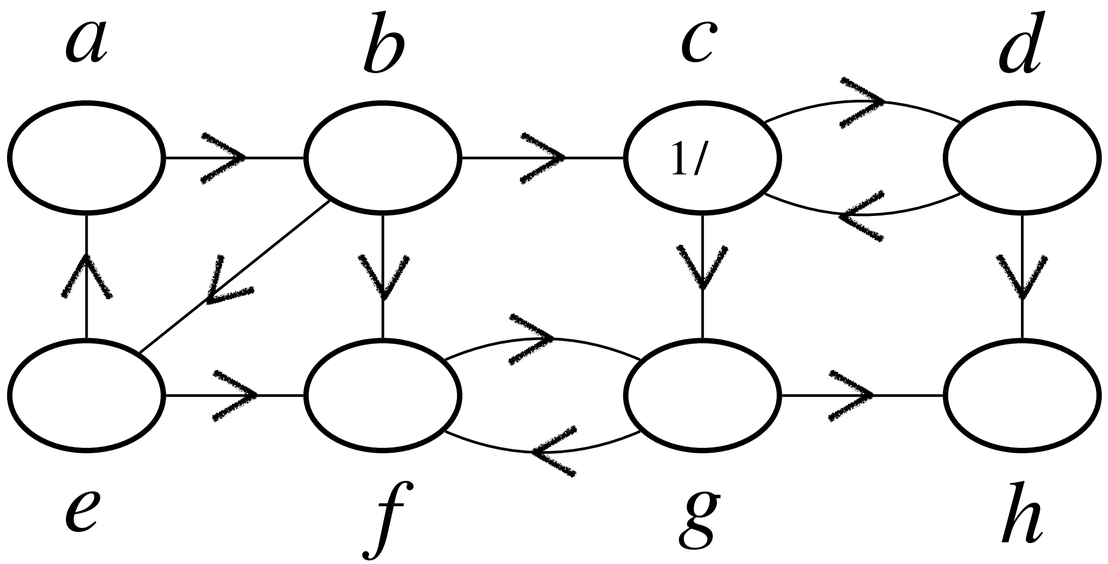
$G = (V, E)$



Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

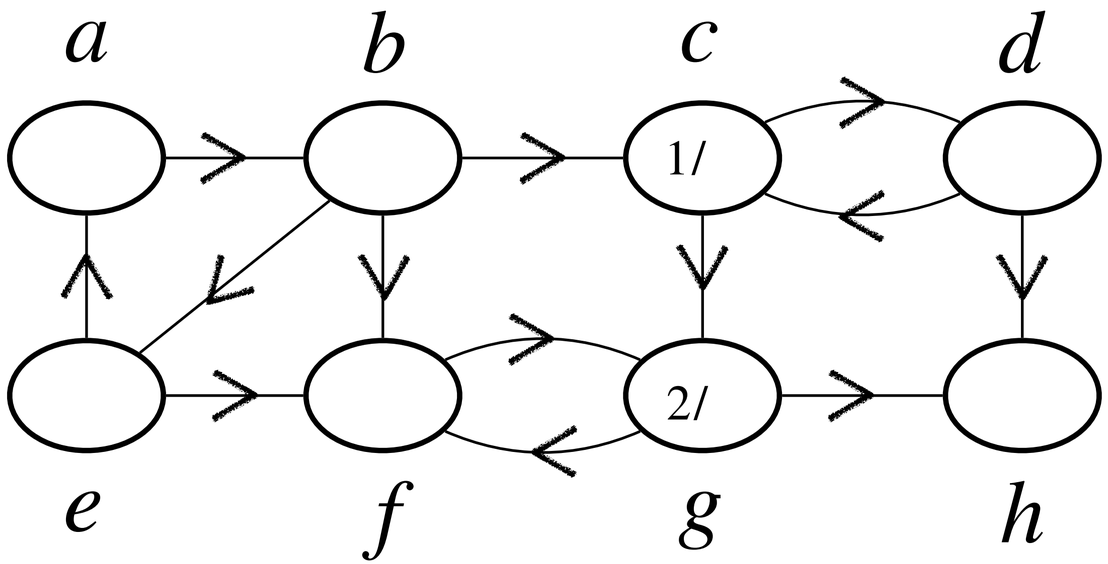
$G = (V, E)$



Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

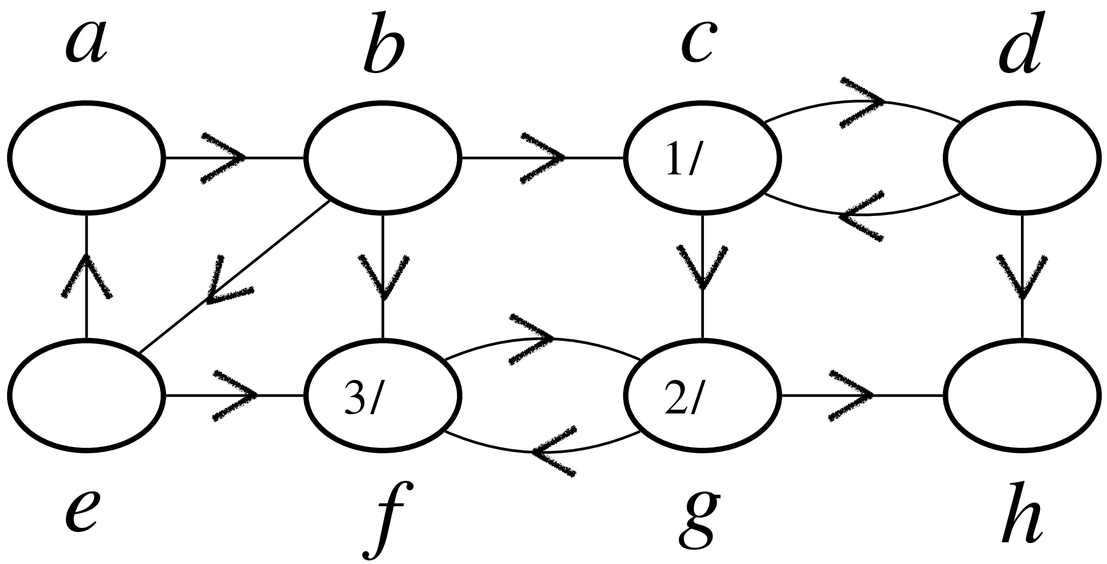
$G = (V, E)$



Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

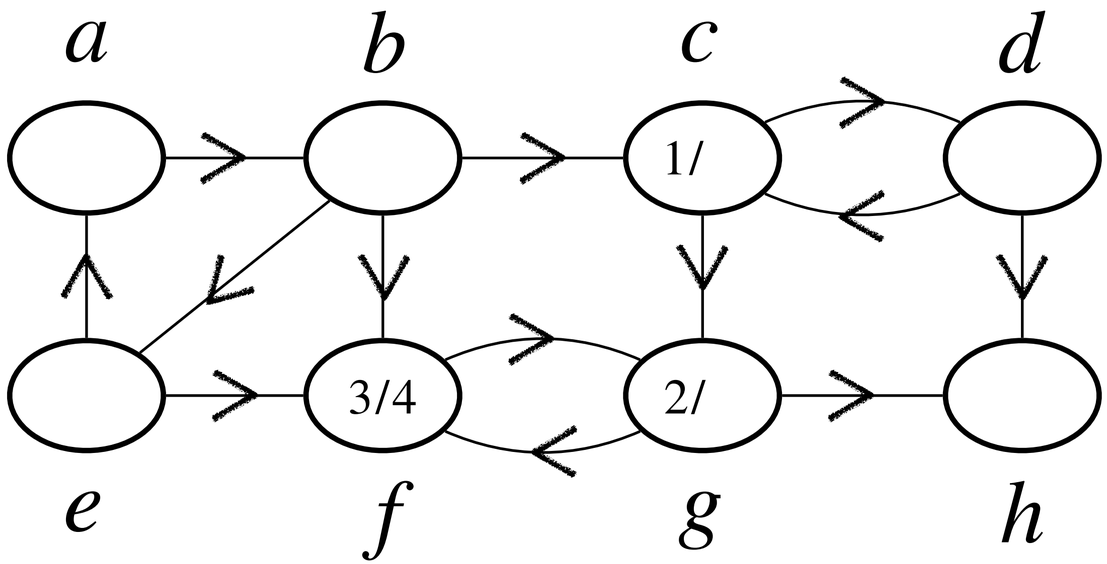
$G = (V, E)$



Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

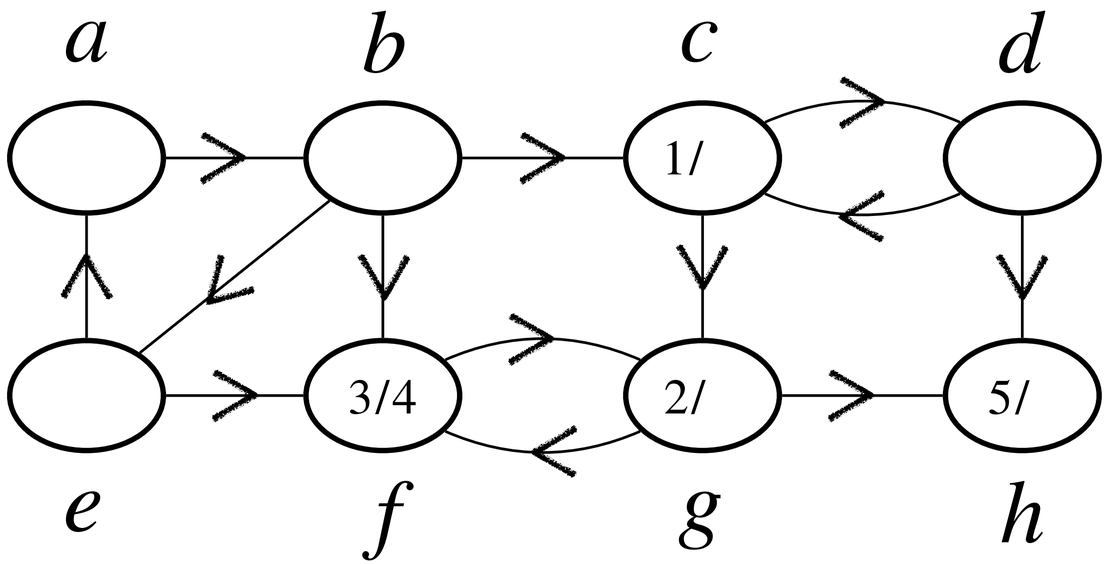
$G = (V, E)$



Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

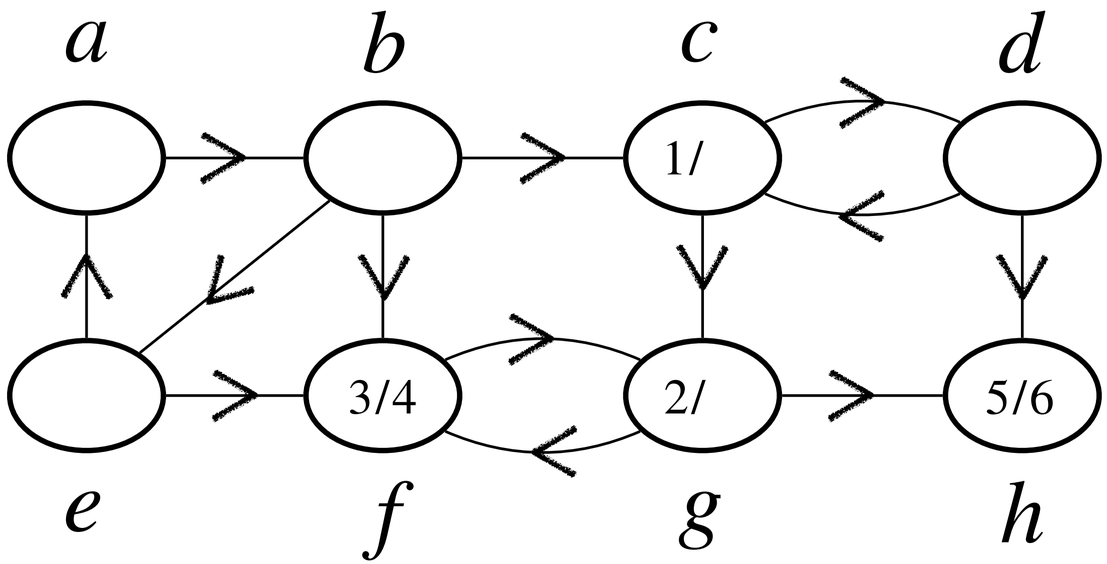
$G = (V, E)$



Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

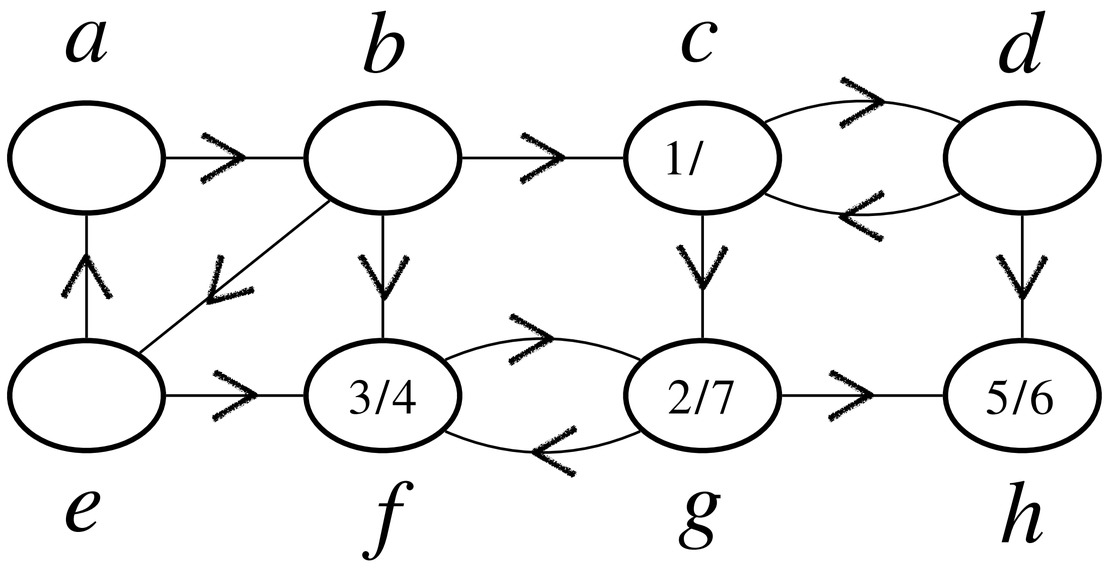
$G = (V, E)$



Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

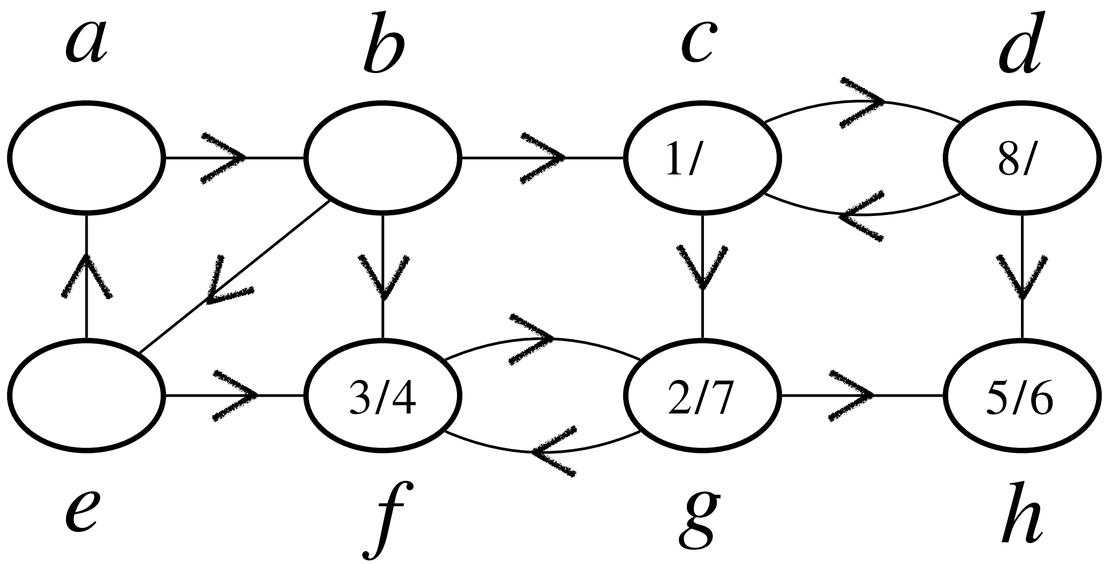
$G = (V, E)$



Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

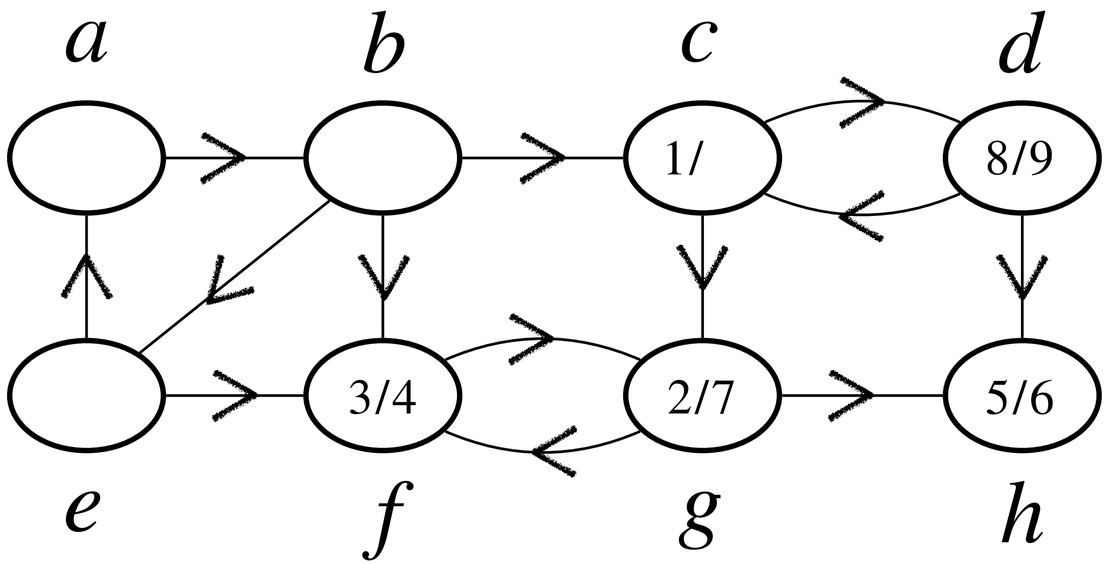
$G = (V, E)$



Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

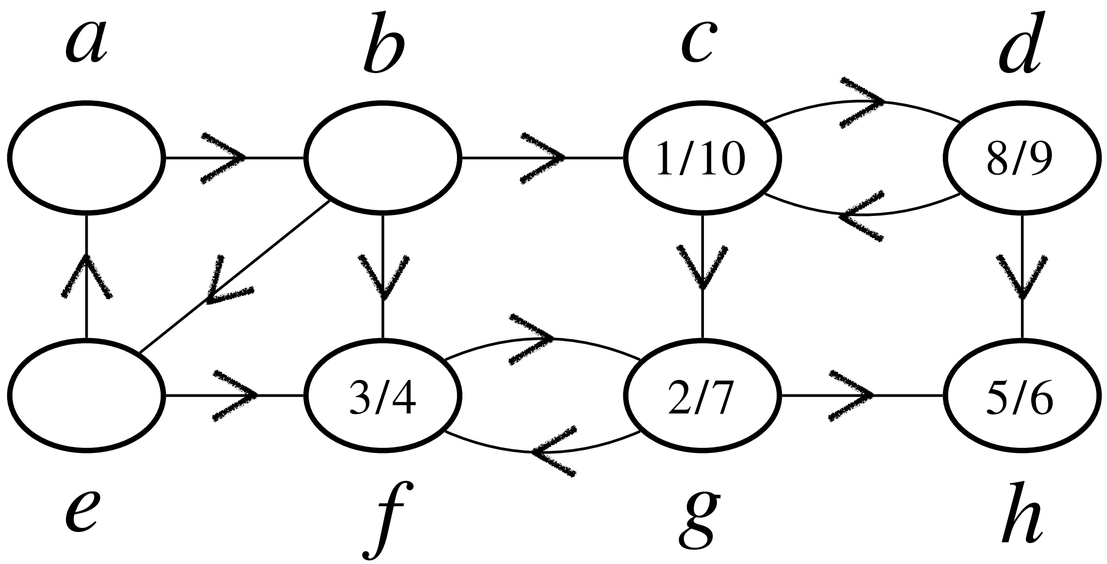
$G = (V, E)$



Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

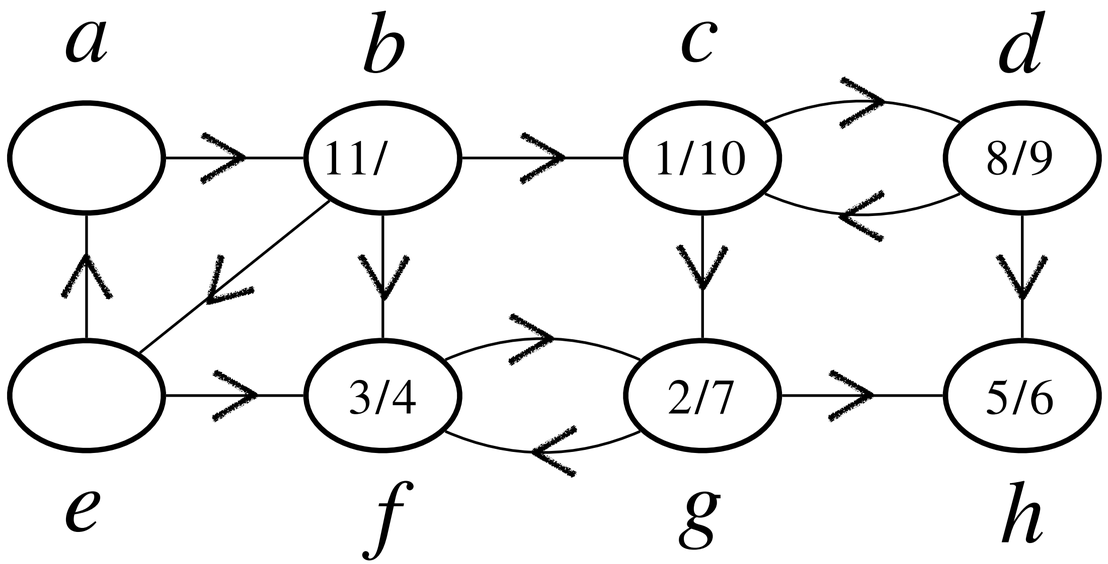
$G = (V, E)$



Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

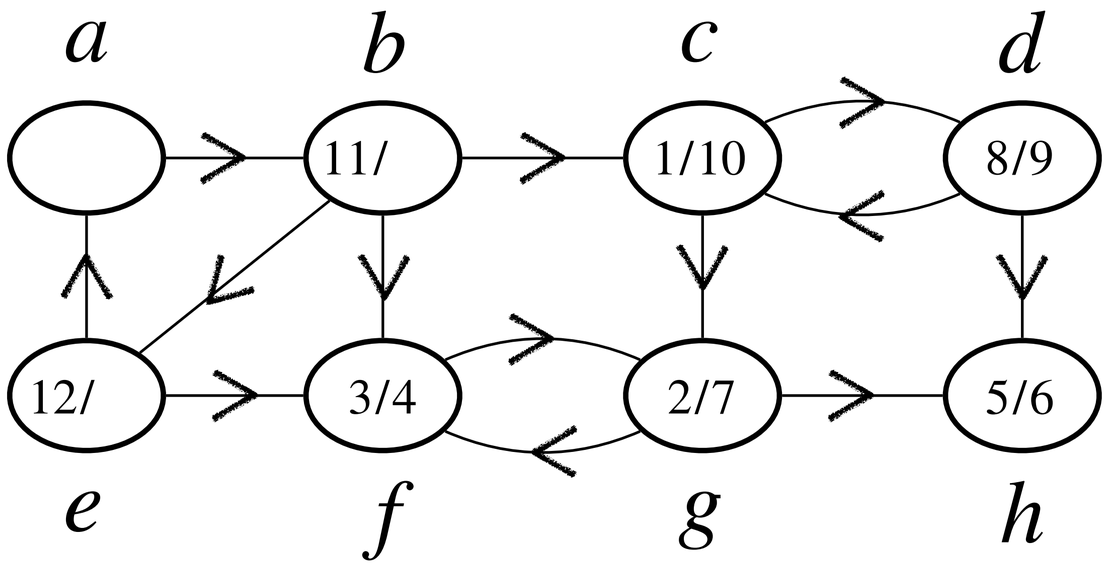
$G = (V, E)$



Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

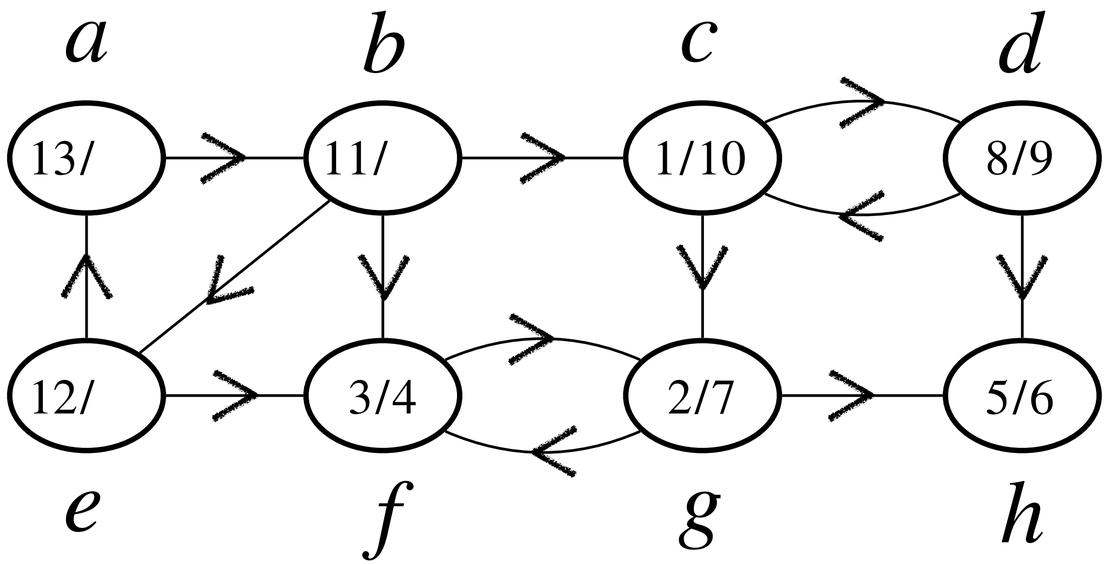
$G = (V, E)$



Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

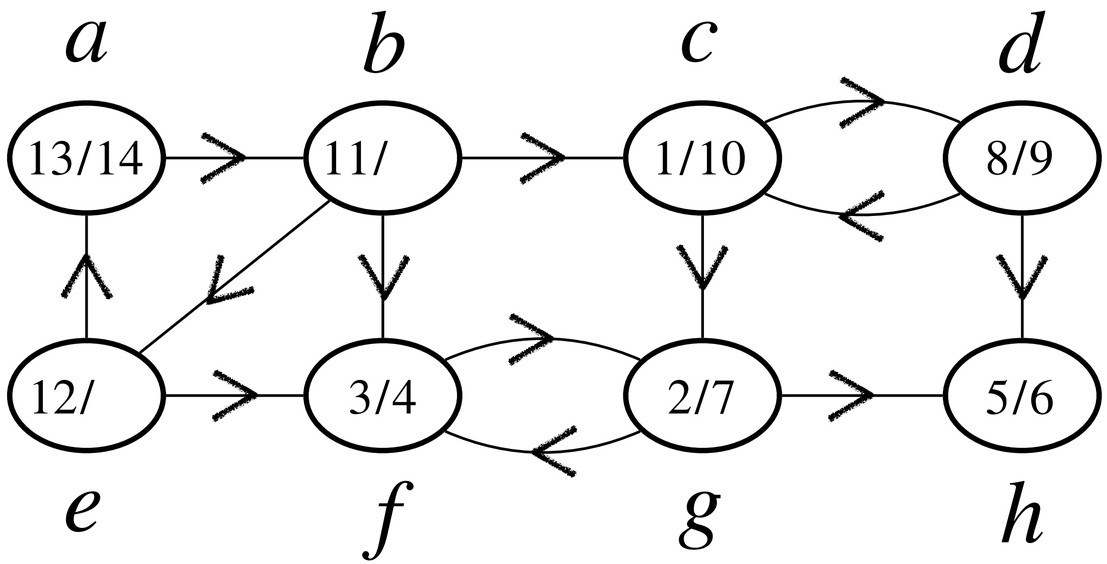
$G = (V, E)$



Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

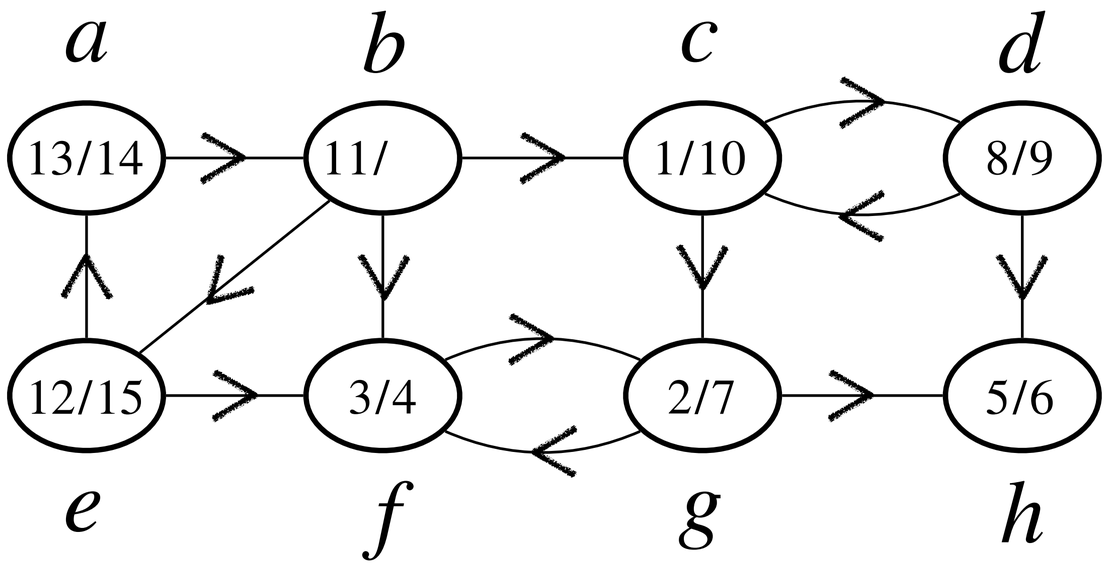
$G = (V, E)$



Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

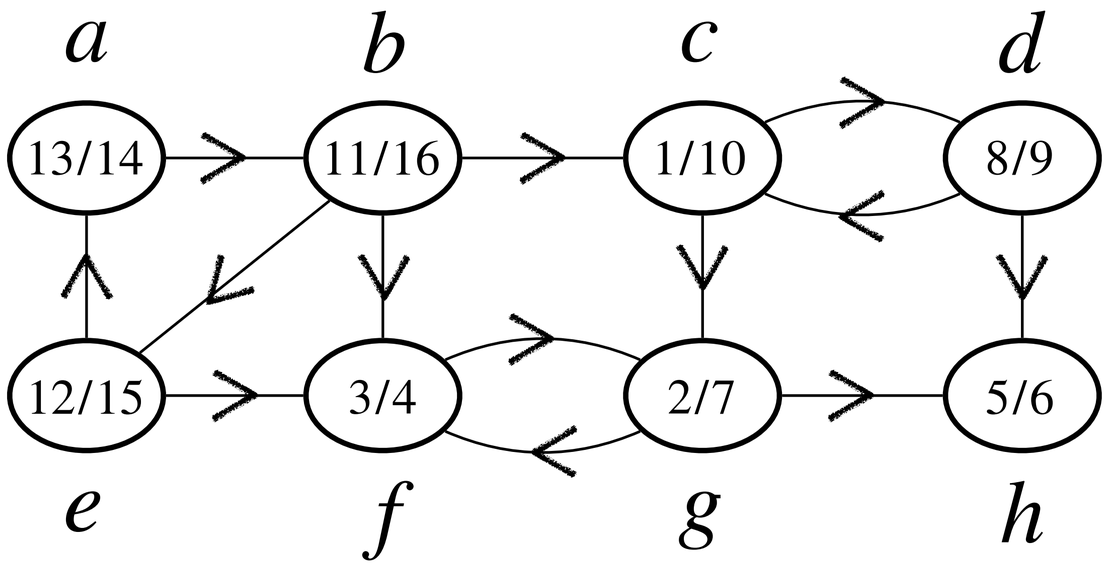
$G = (V, E)$



Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

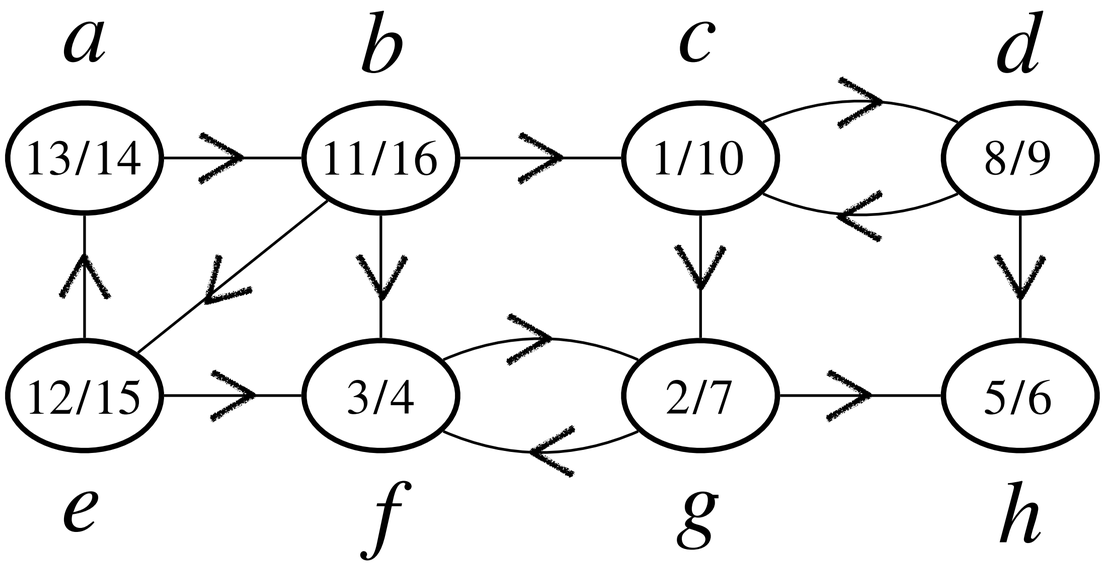
$G = (V, E)$



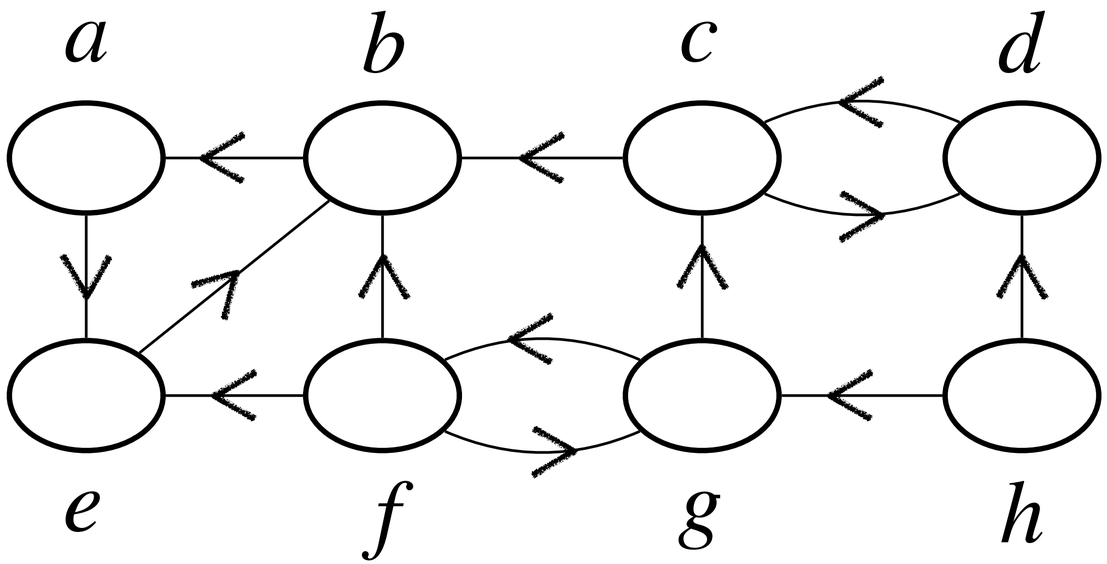
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



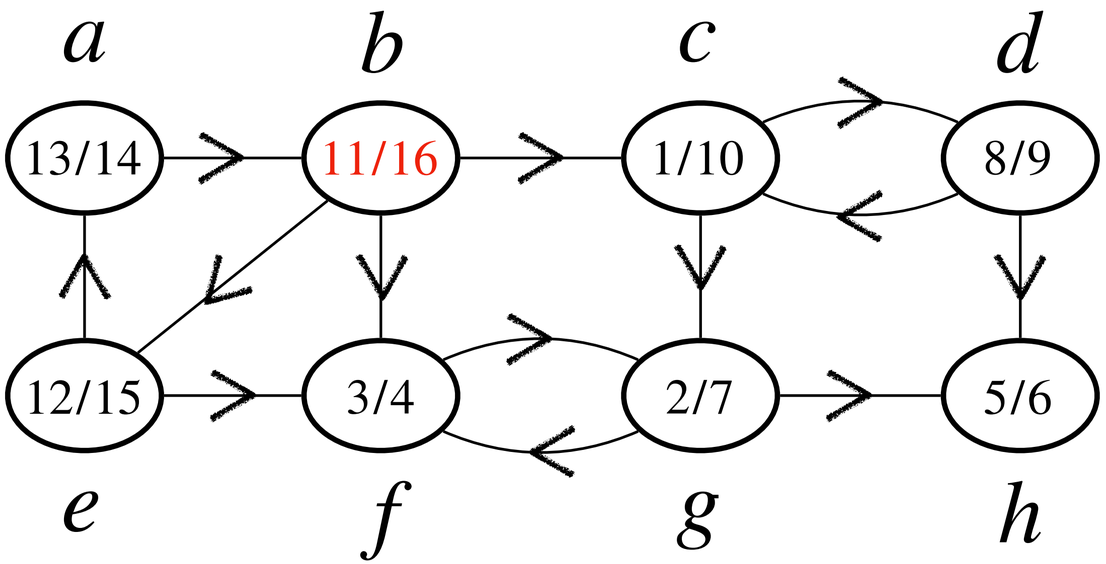
$G^T = (V, E^T)$: transpose of G



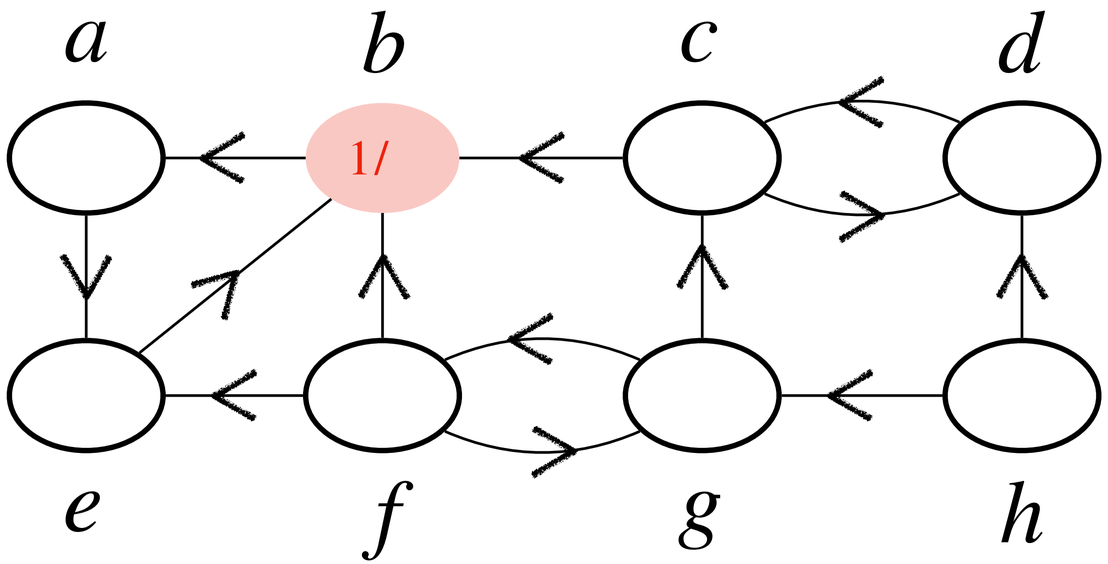
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



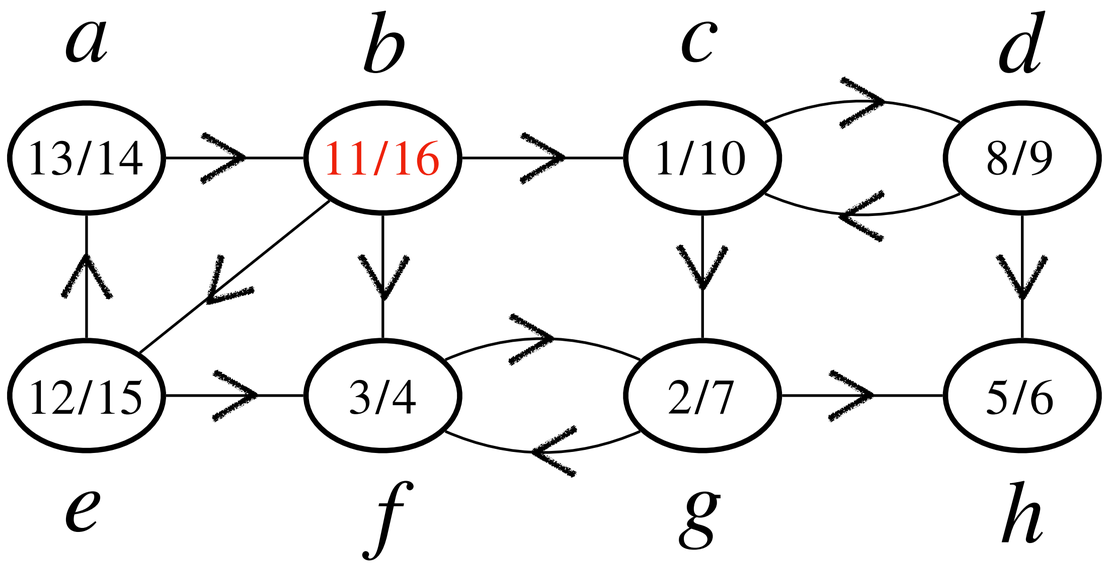
$G^T = (V, E^T)$: transpose of G



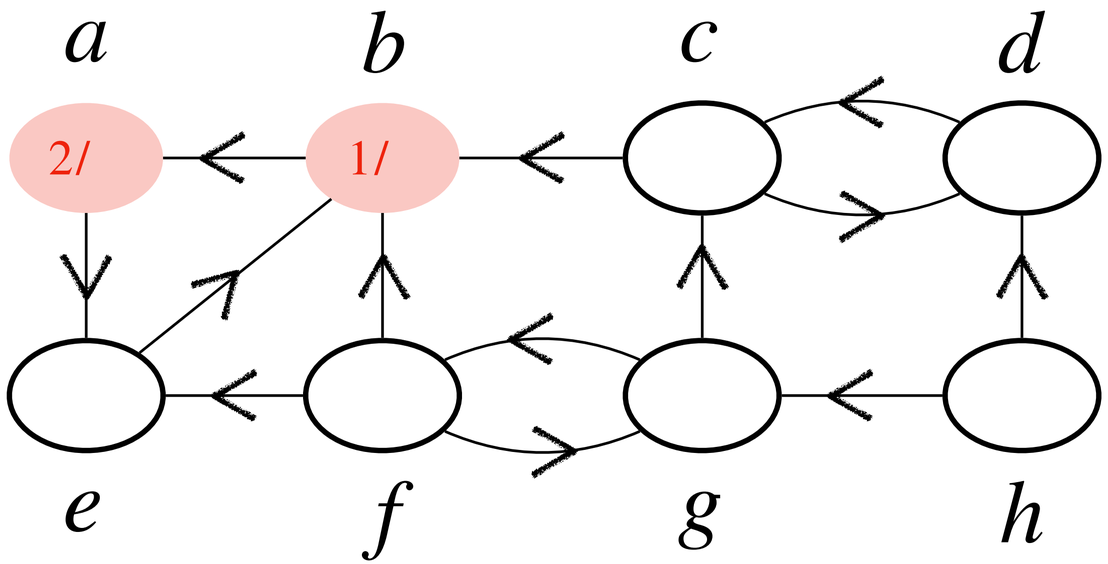
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



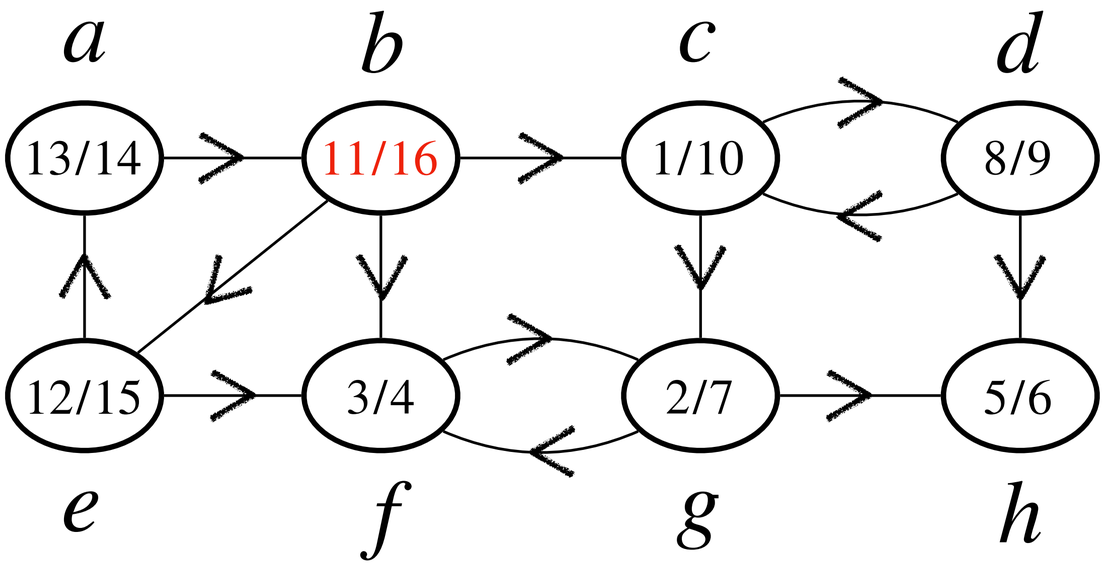
$G^T = (V, E^T)$: transpose of G



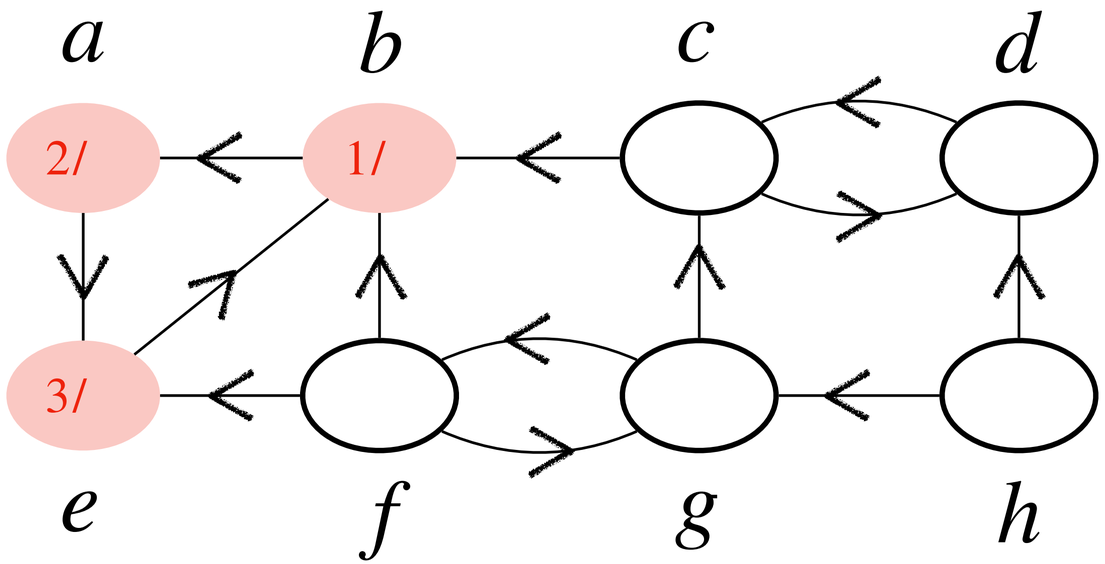
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



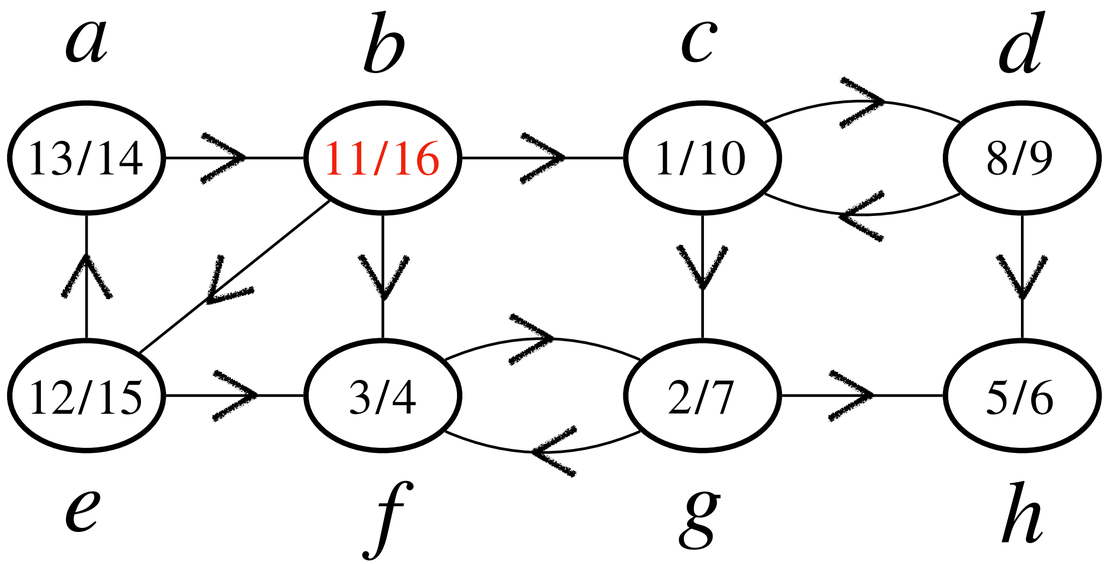
$G^T = (V, E^T)$: transpose of G



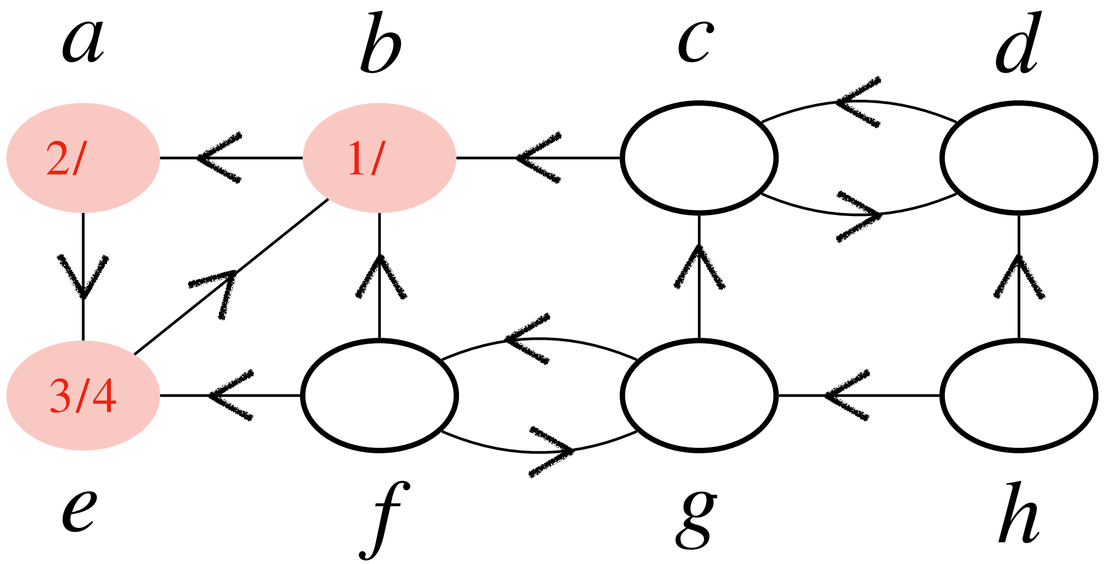
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



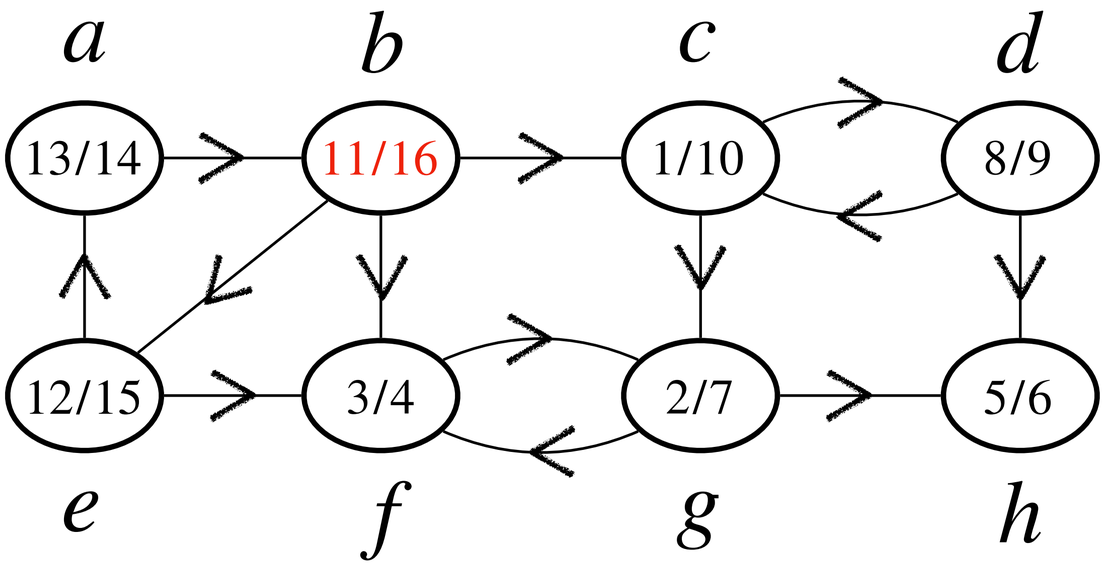
$G^T = (V, E^T)$: transpose of G



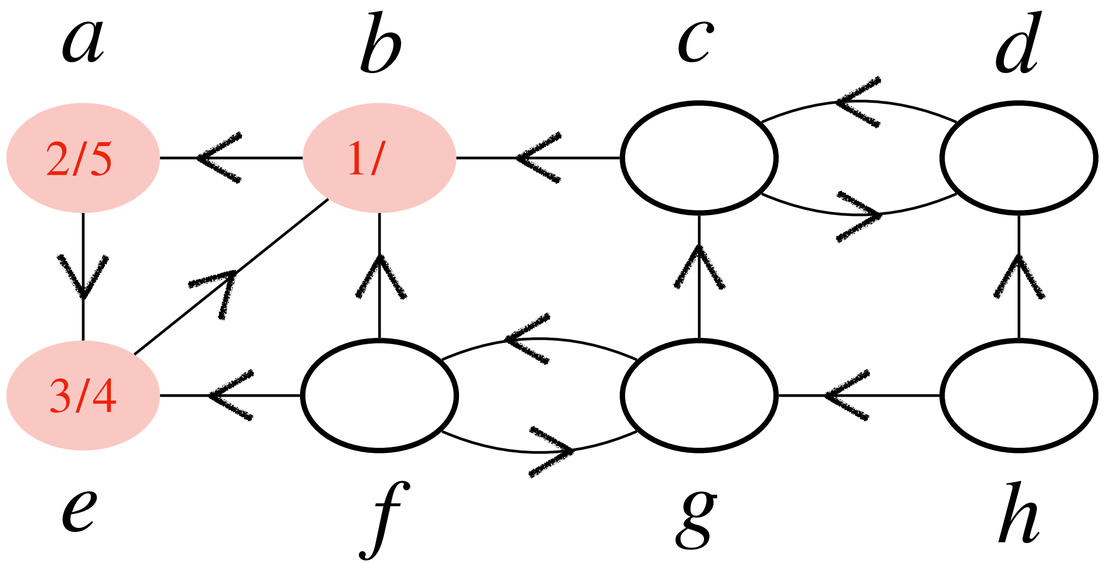
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



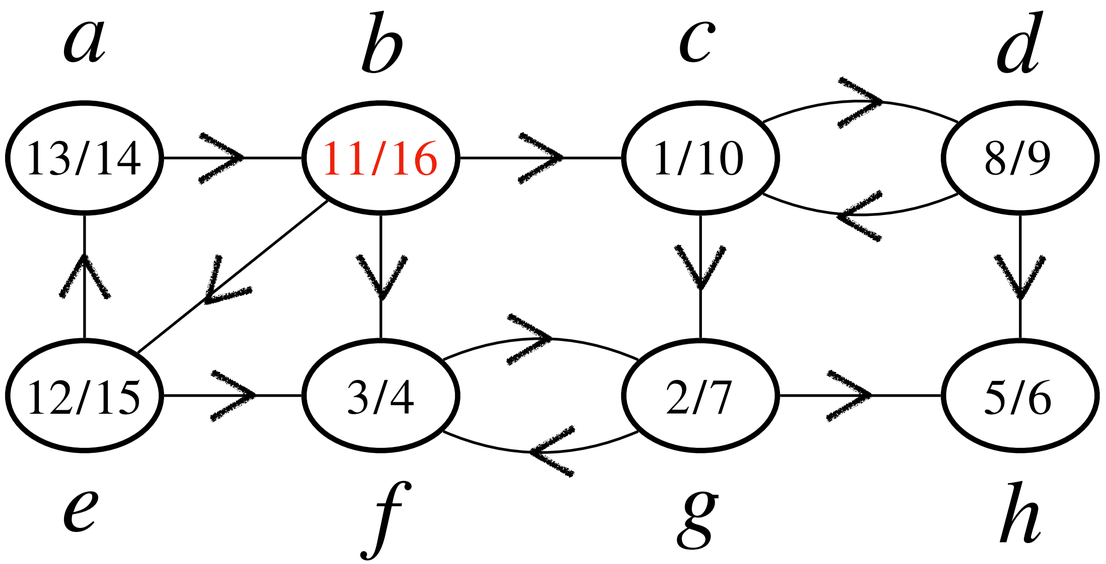
$G^T = (V, E^T)$: transpose of G



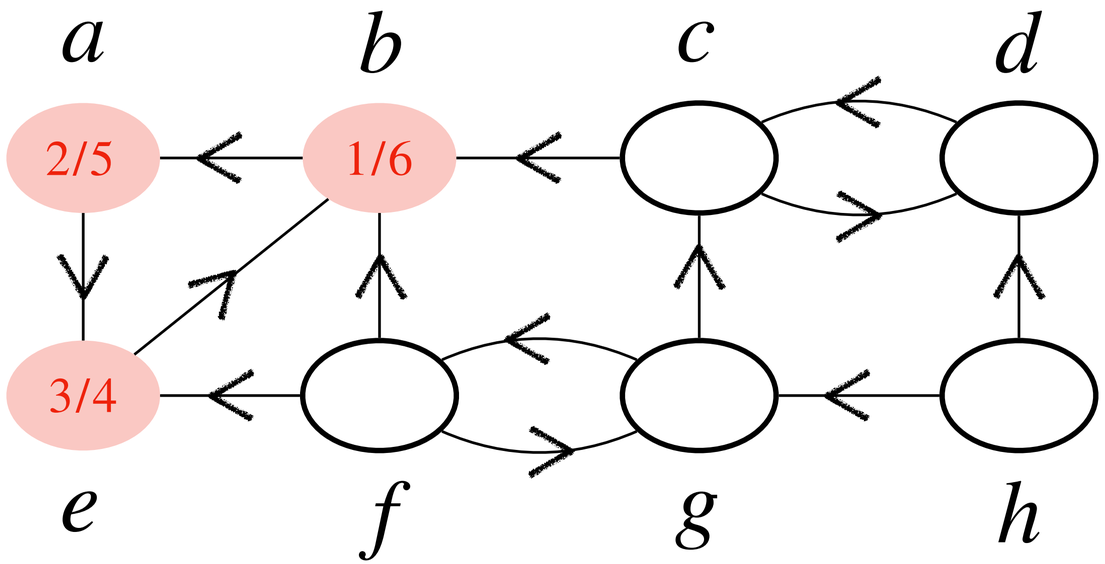
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



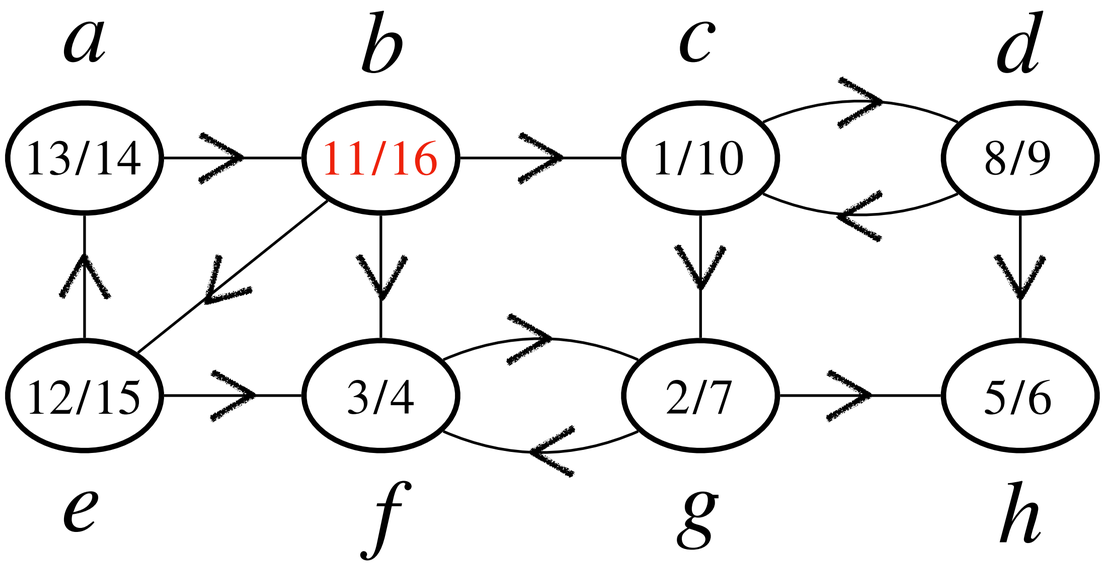
$G^T = (V, E^T)$: transpose of G



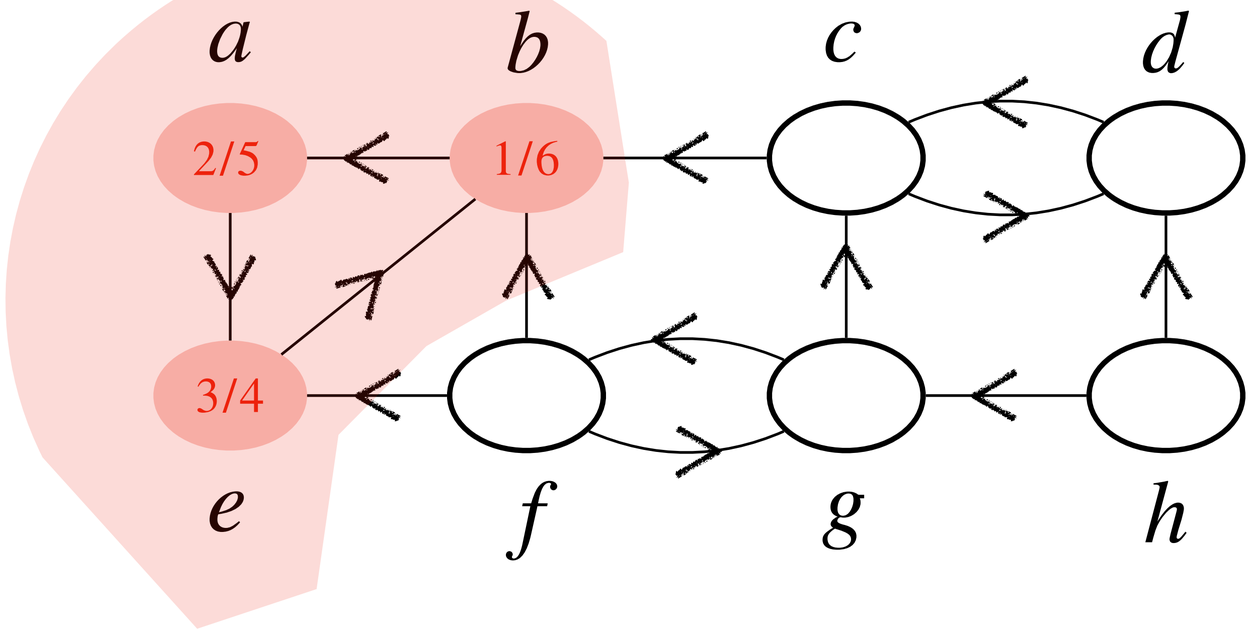
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



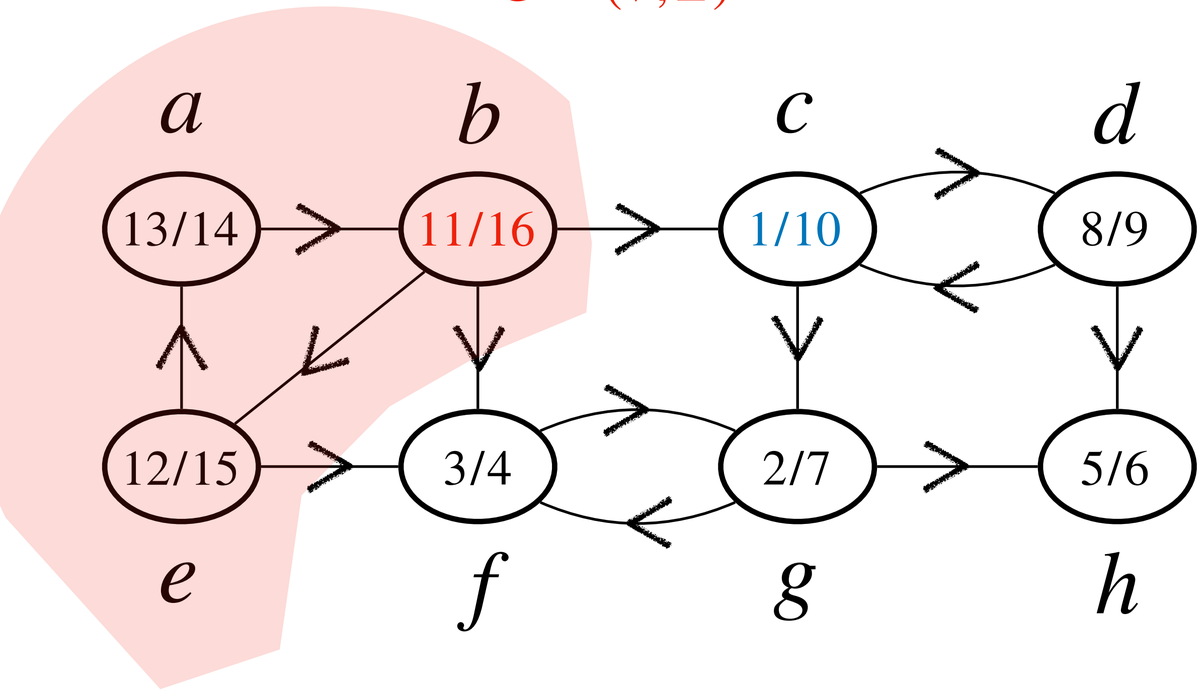
$G^T = (V, E^T)$: transpose of G



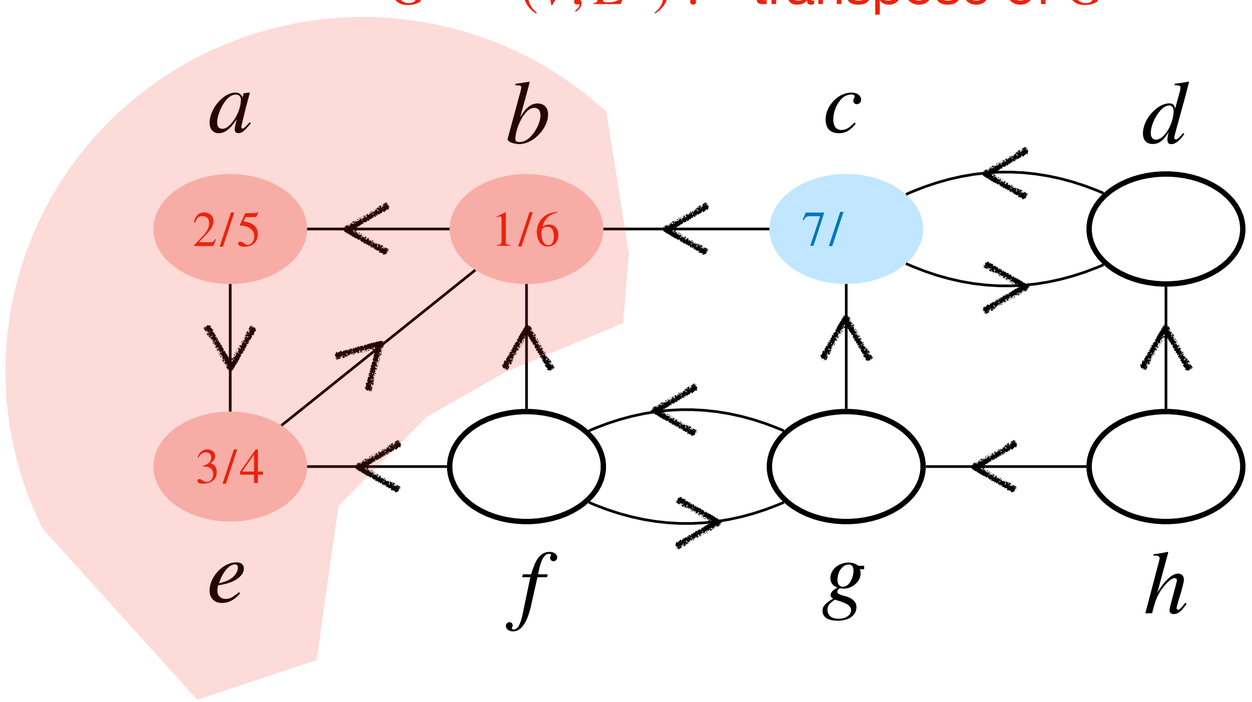
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



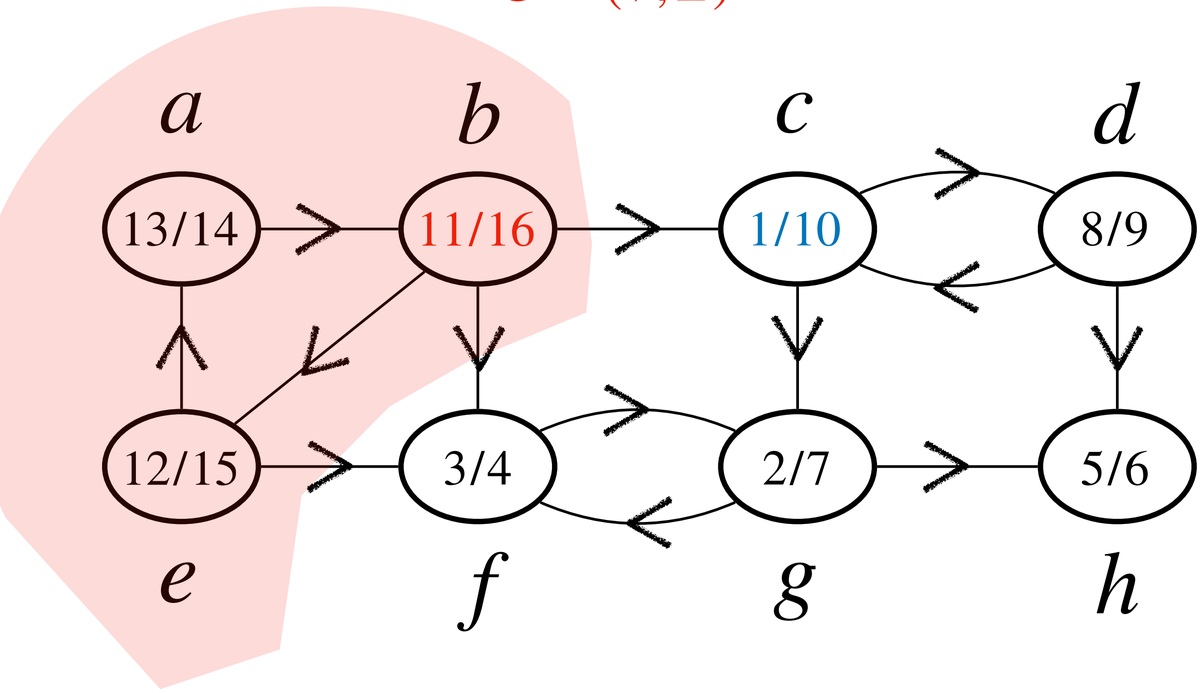
$G^T = (V, E^T)$: transpose of G



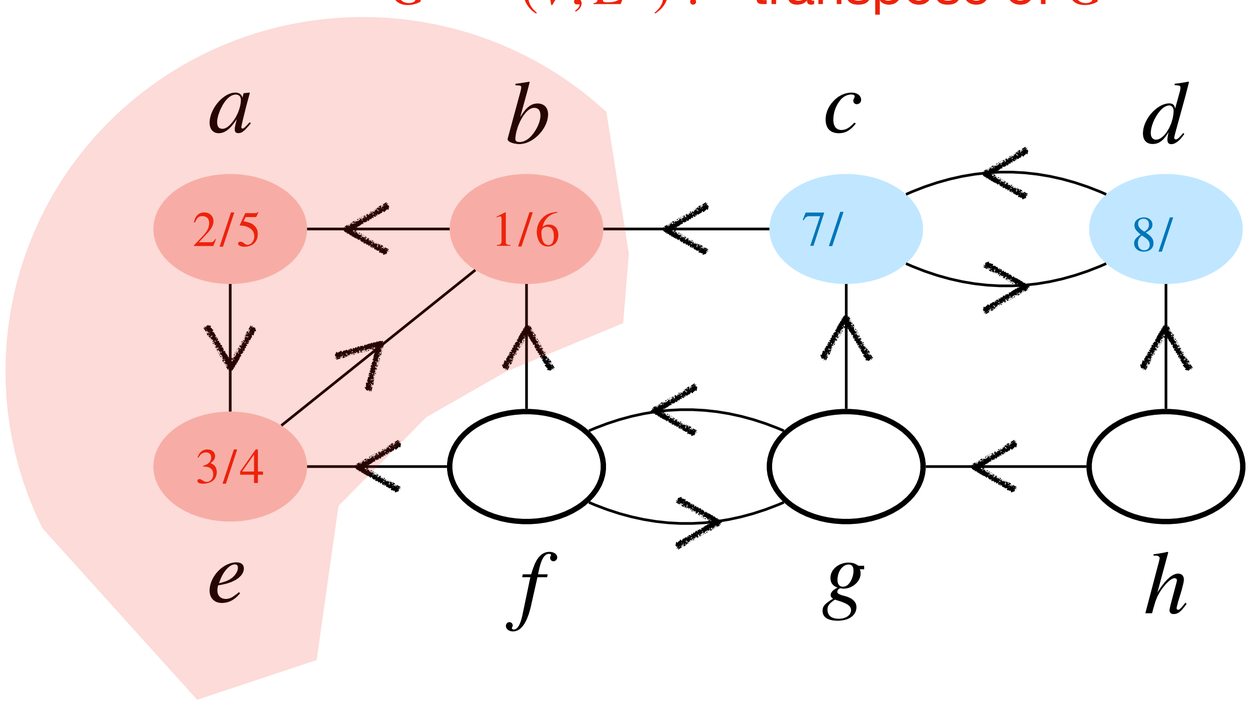
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



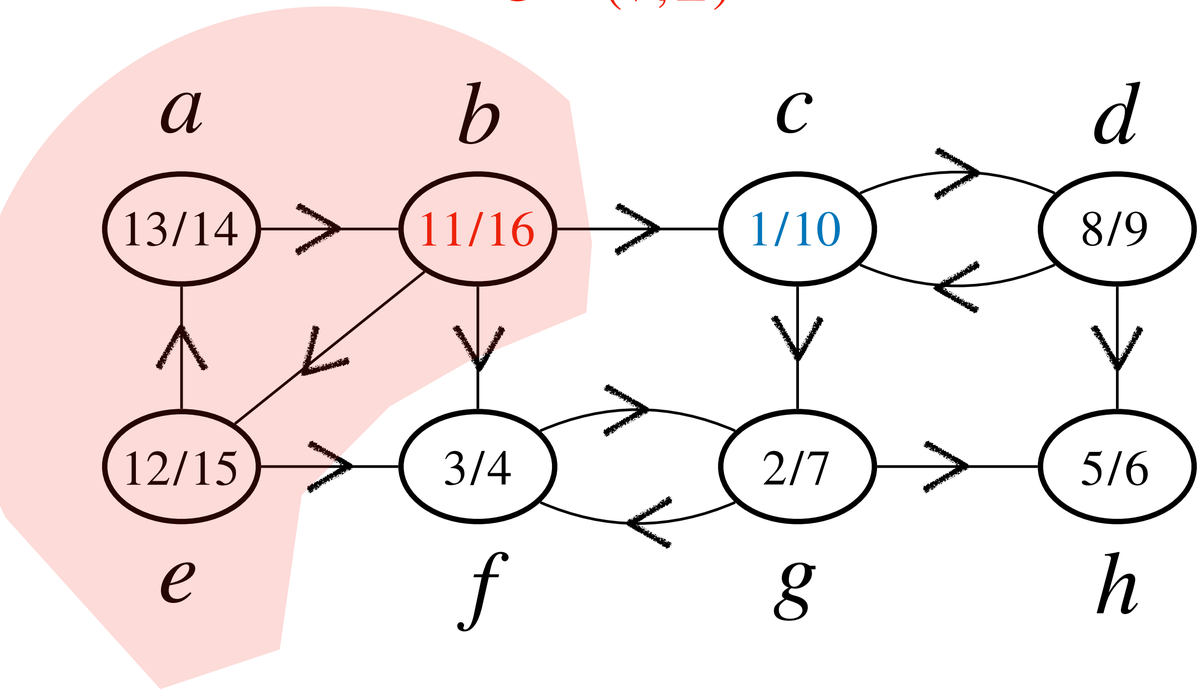
$G^T = (V, E^T)$: transpose of G



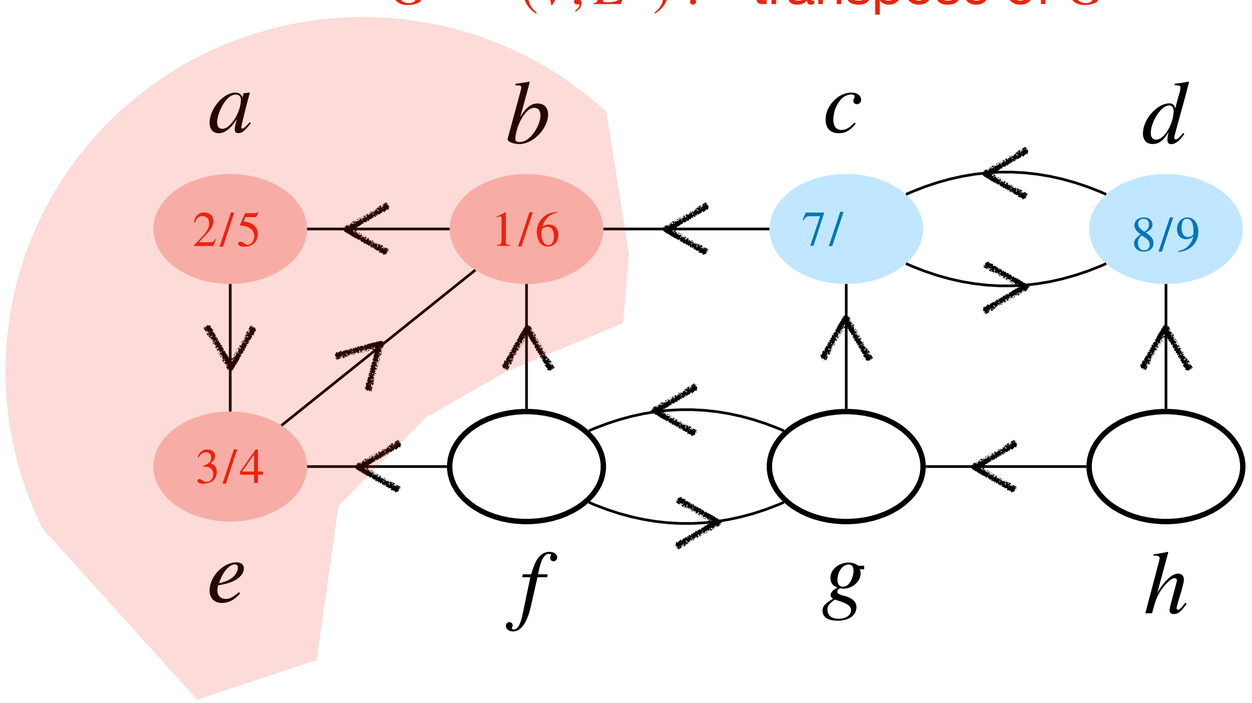
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



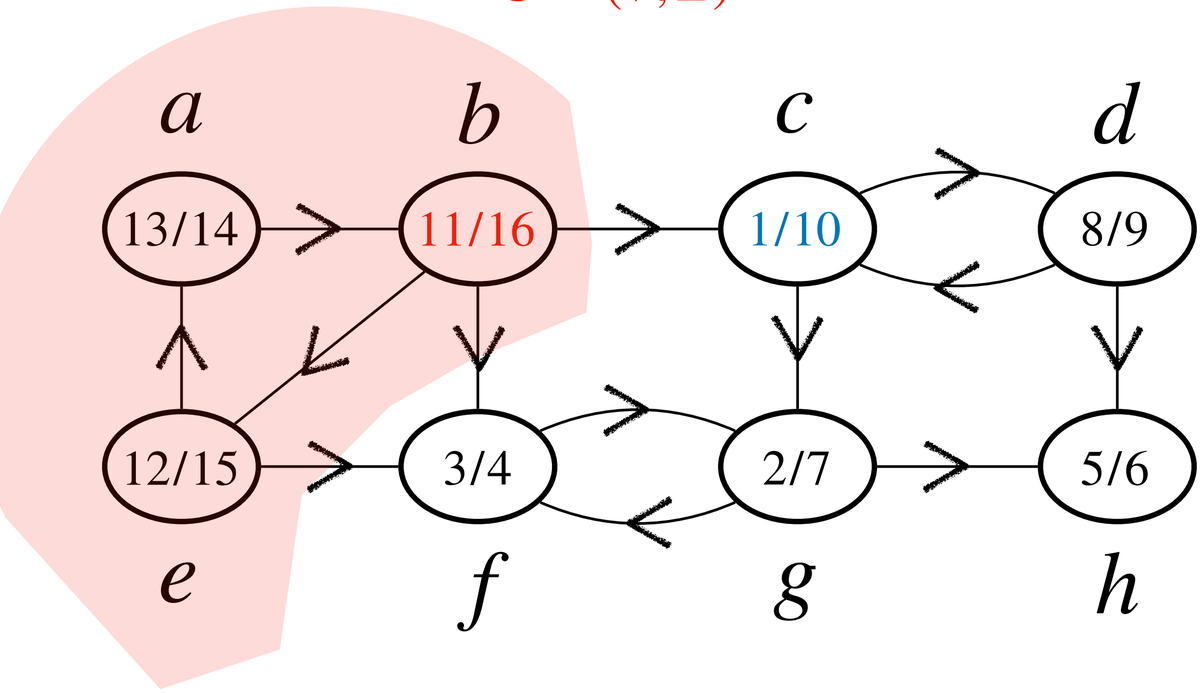
$G^T = (V, E^T)$: transpose of G



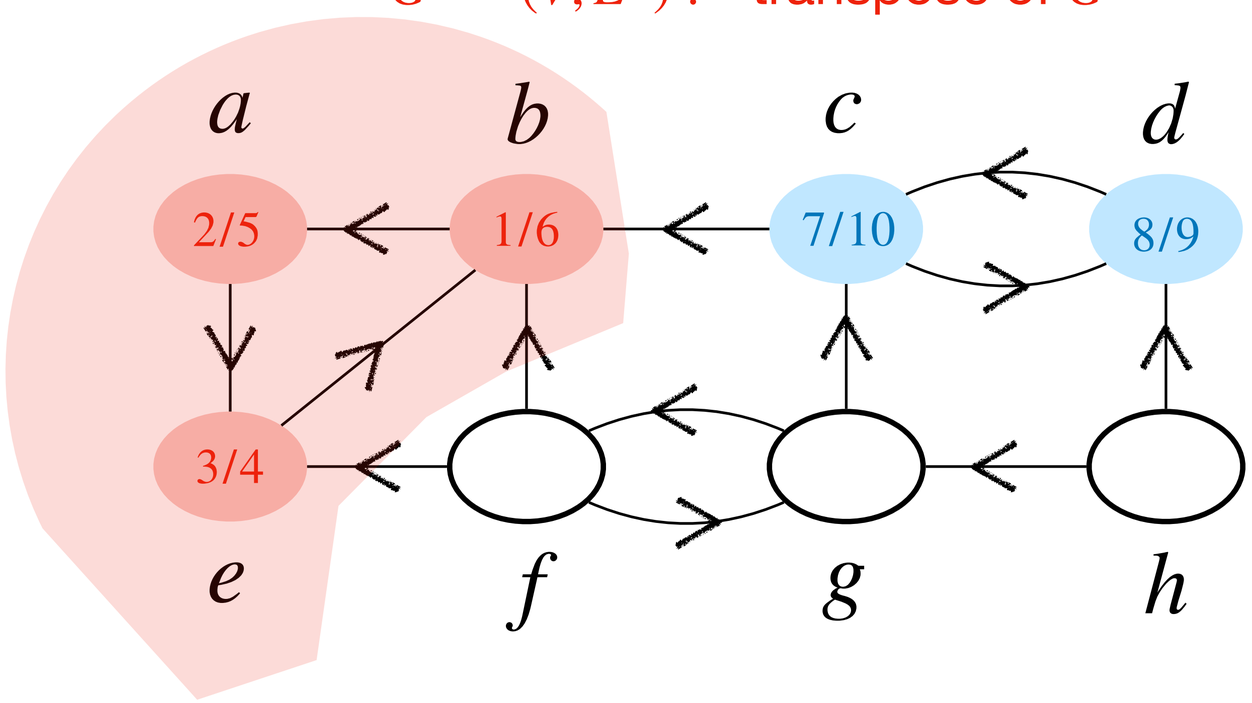
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



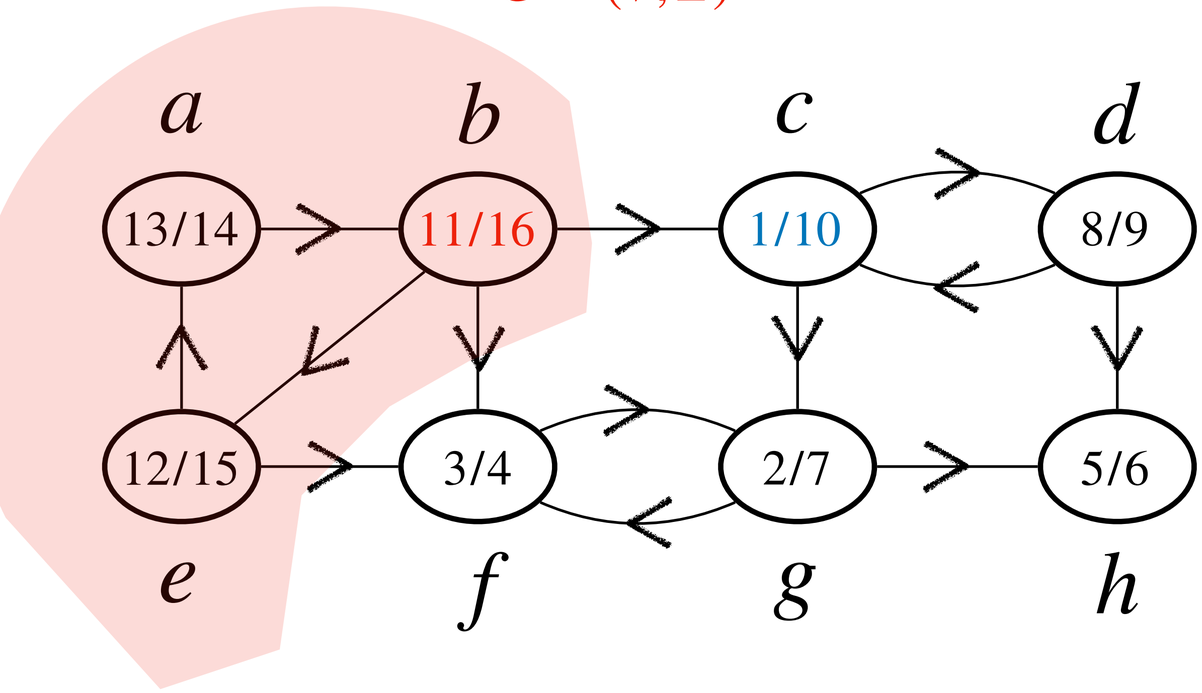
$G^T = (V, E^T)$: transpose of G



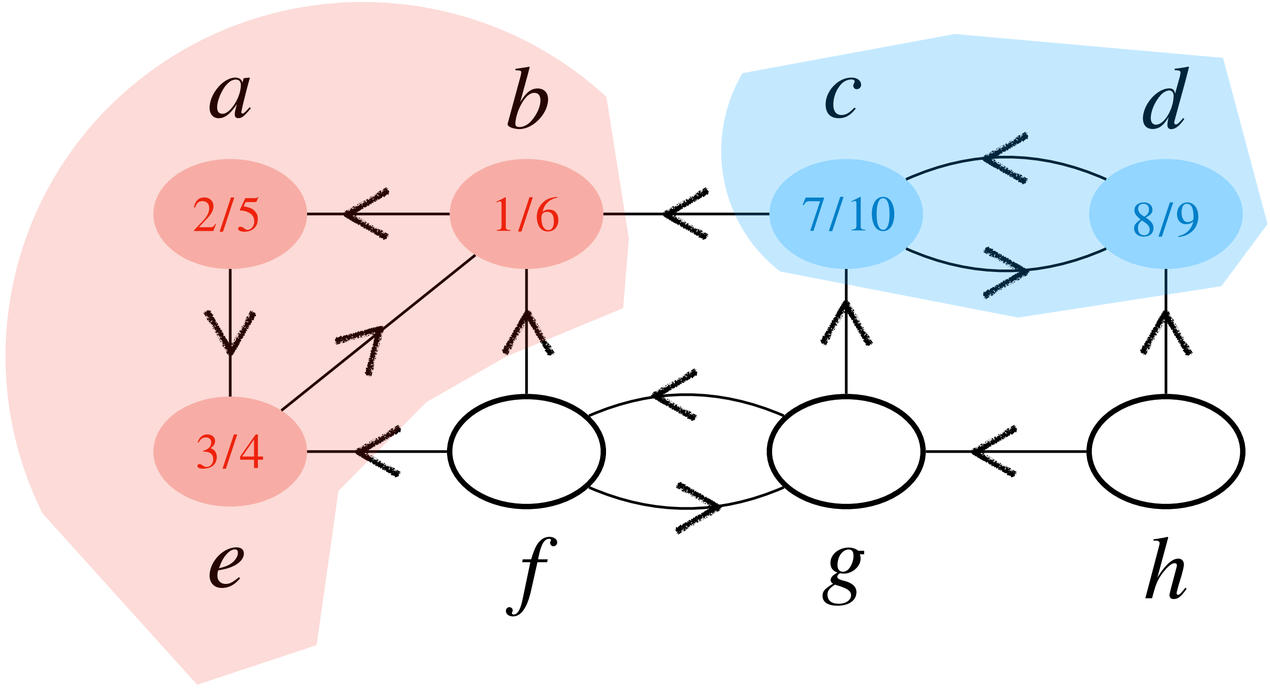
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



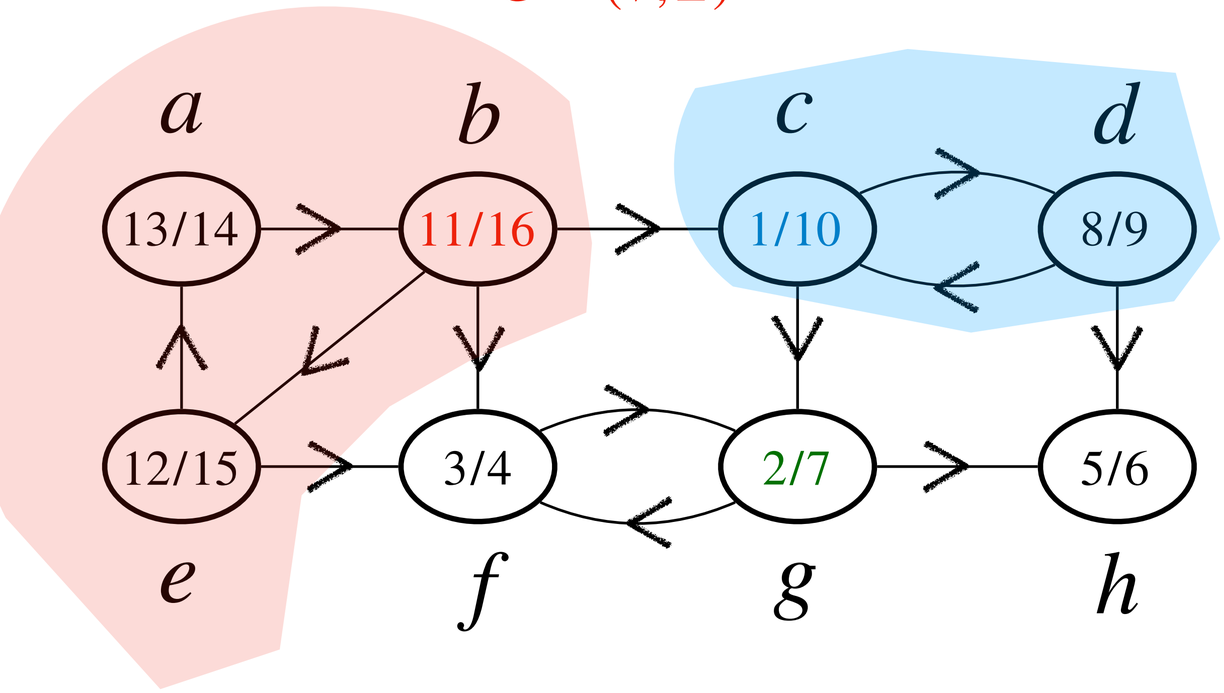
$G^T = (V, E^T)$: transpose of G



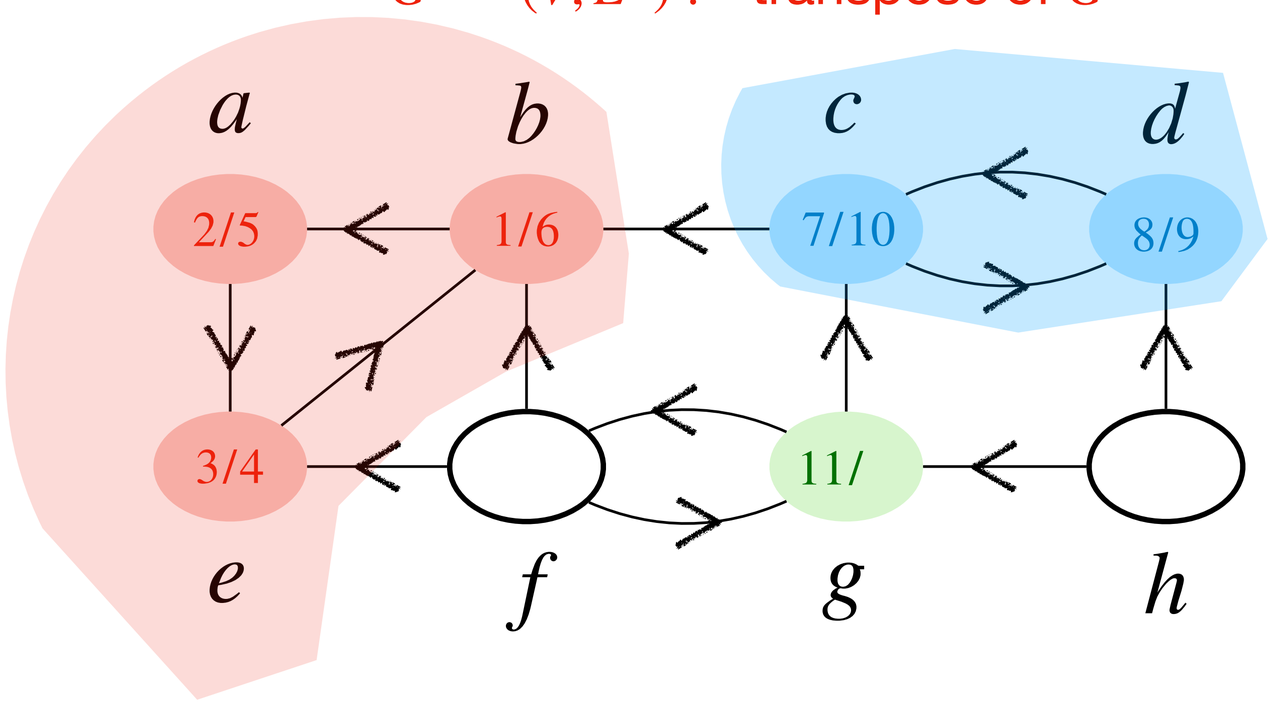
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



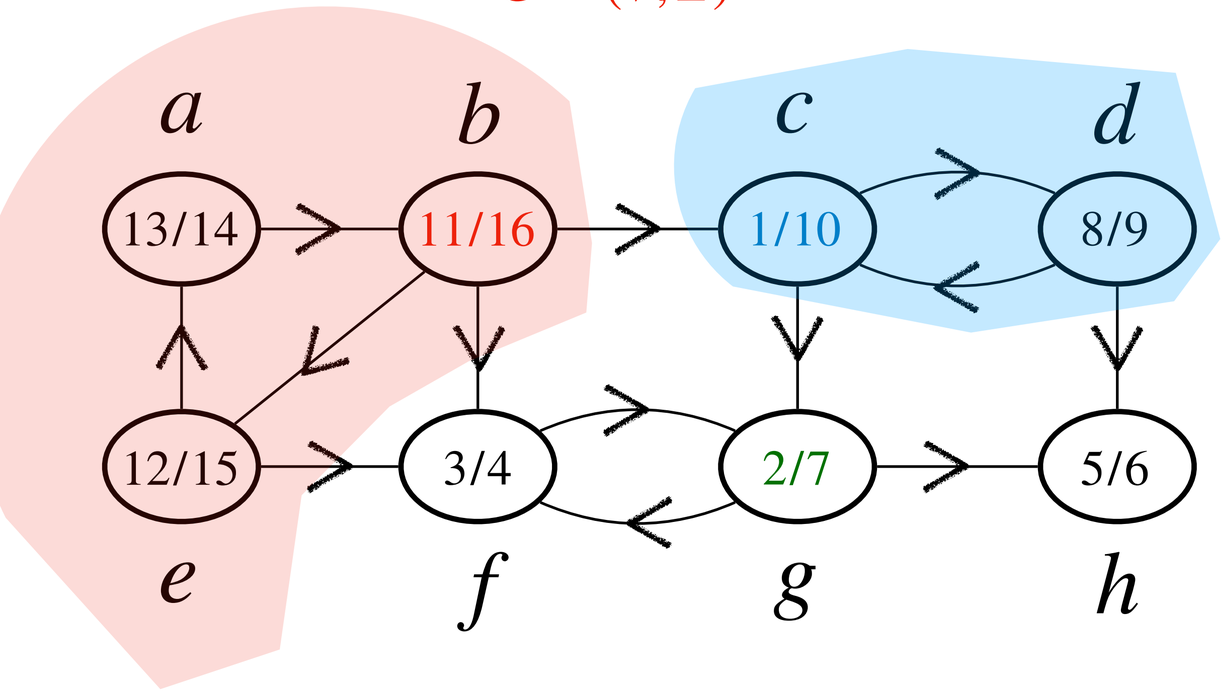
$G^T = (V, E^T)$: transpose of G



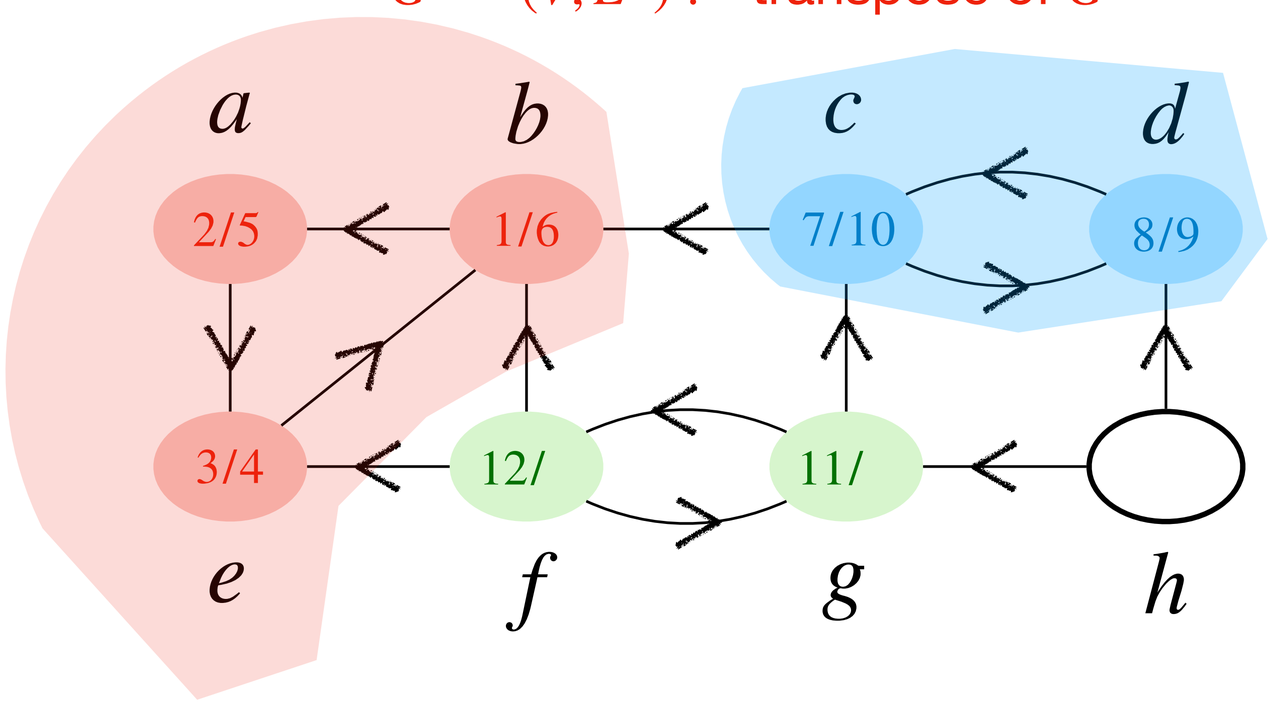
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



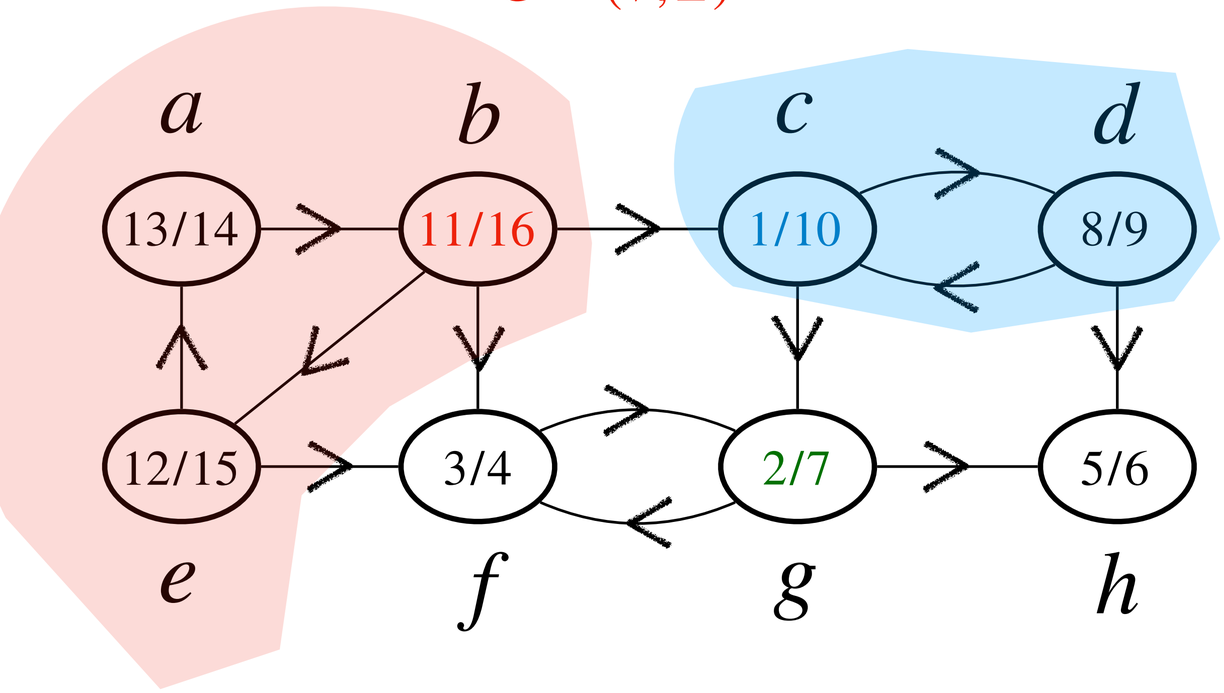
$G^T = (V, E^T)$: transpose of G



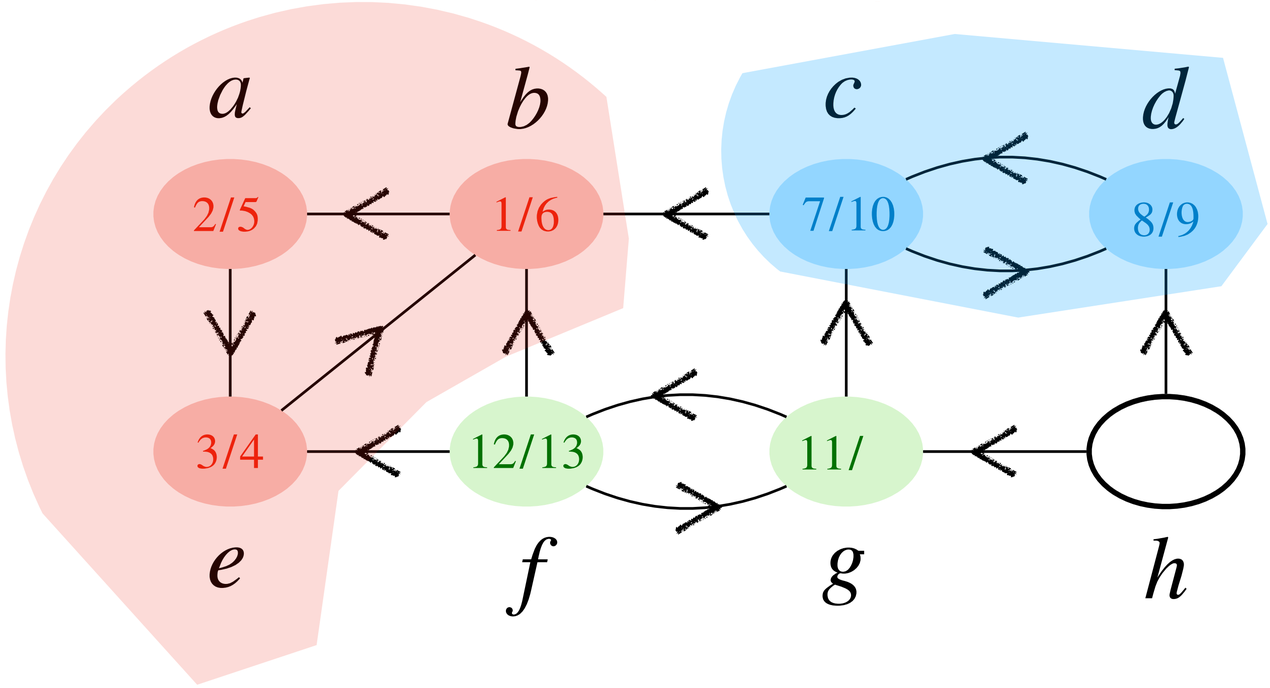
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



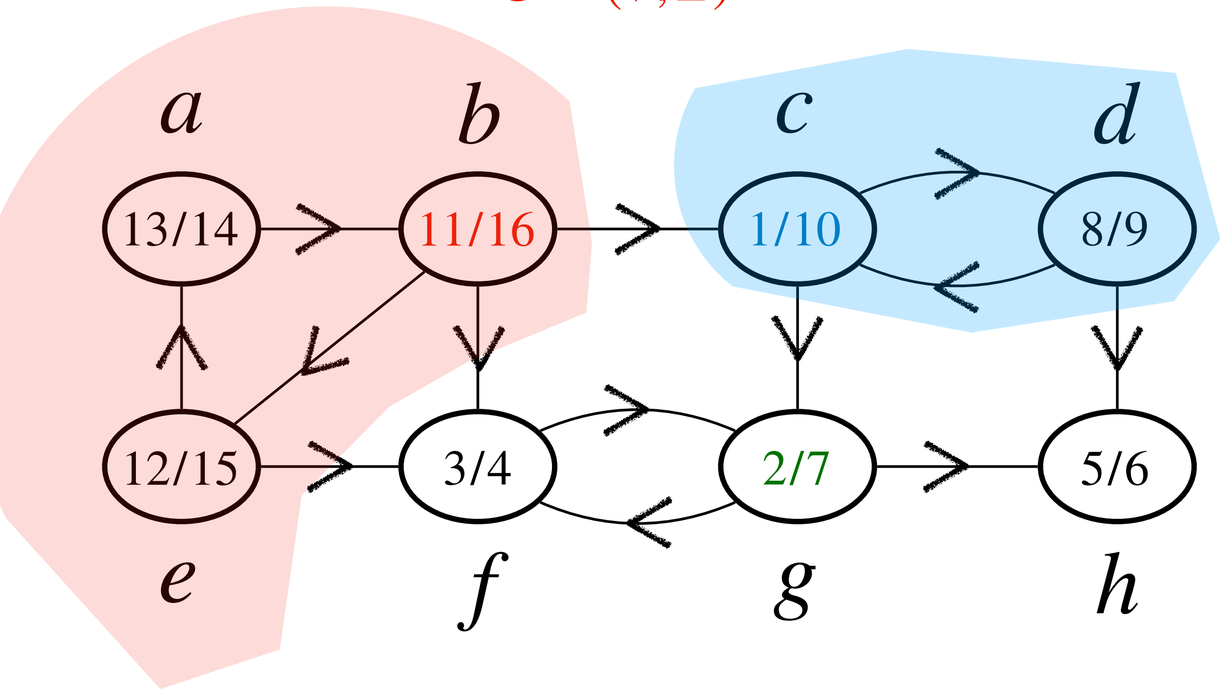
$G^T = (V, E^T)$: transpose of G



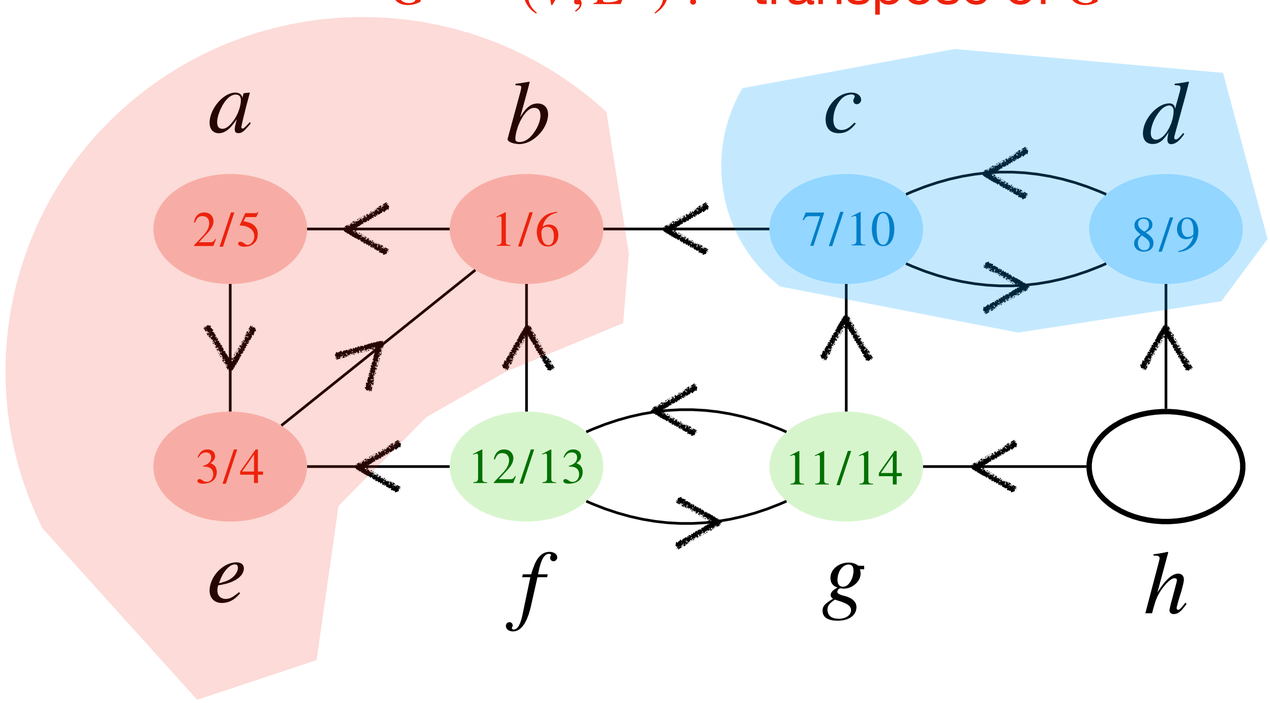
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



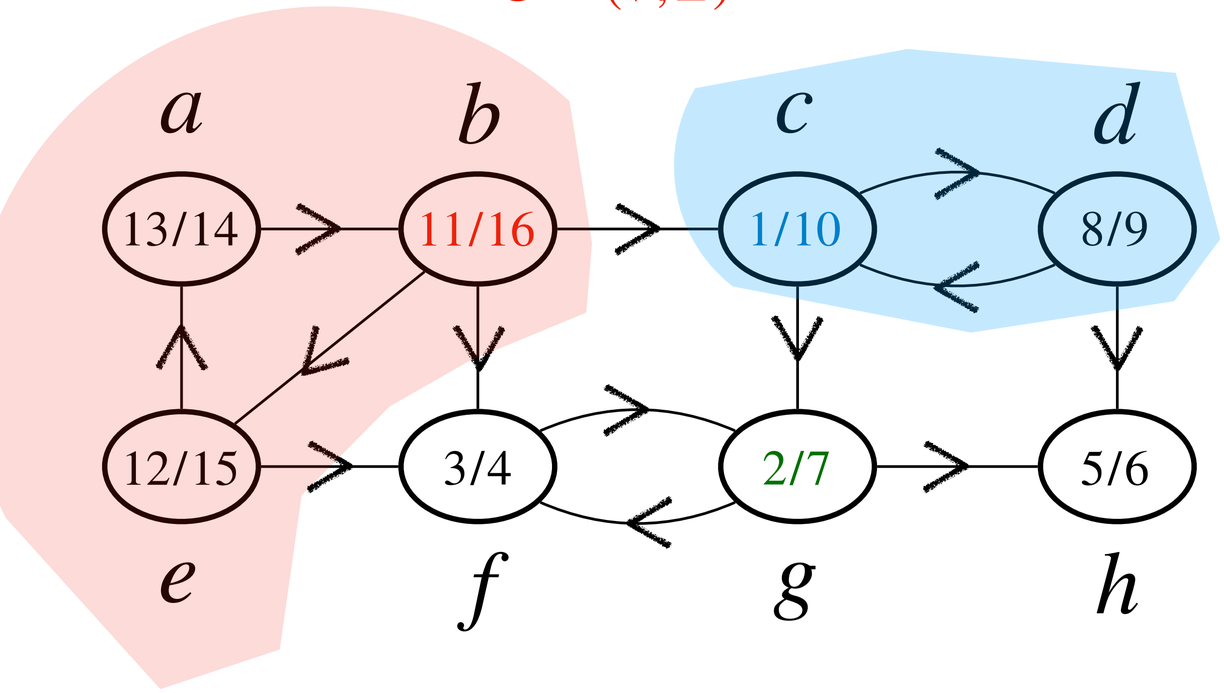
$G^T = (V, E^T)$: transpose of G



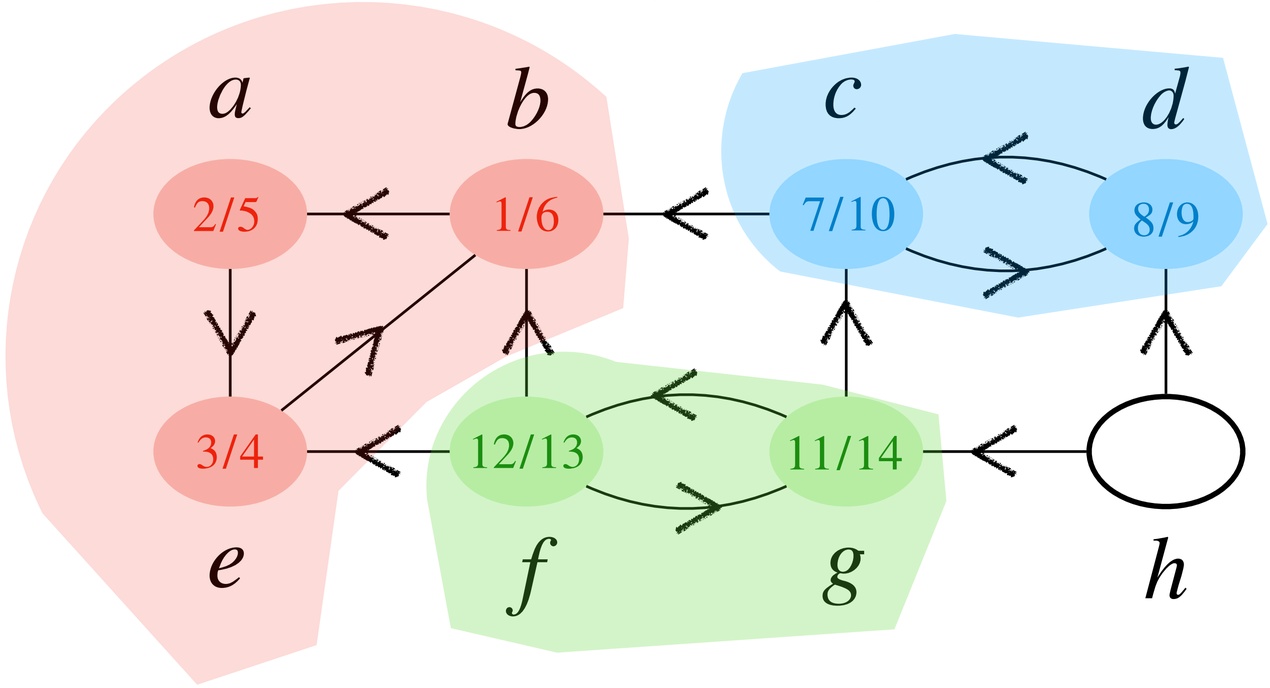
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



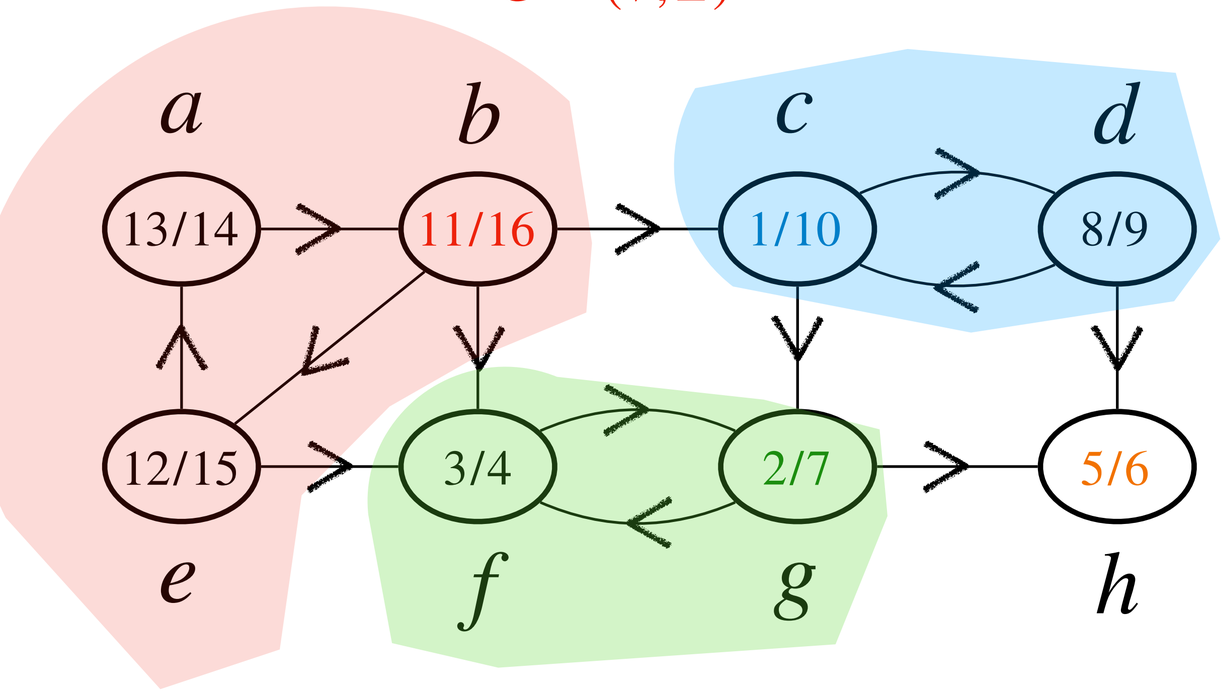
$G^T = (V, E^T)$: transpose of G



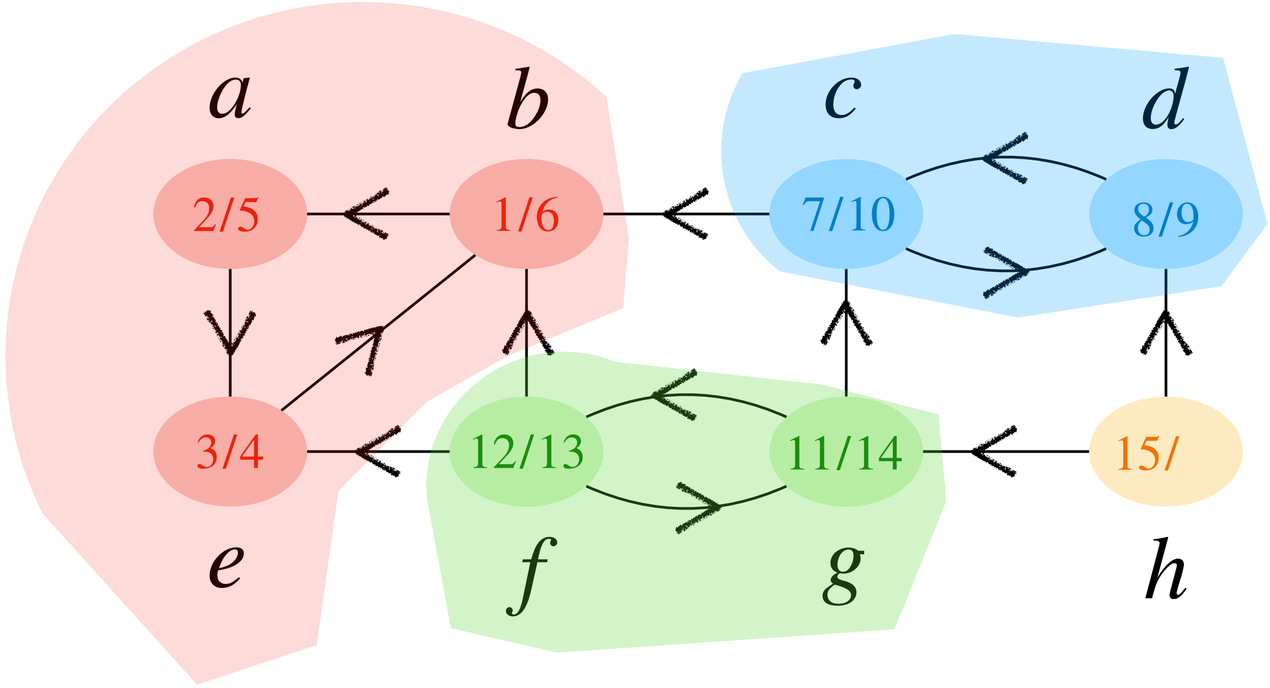
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



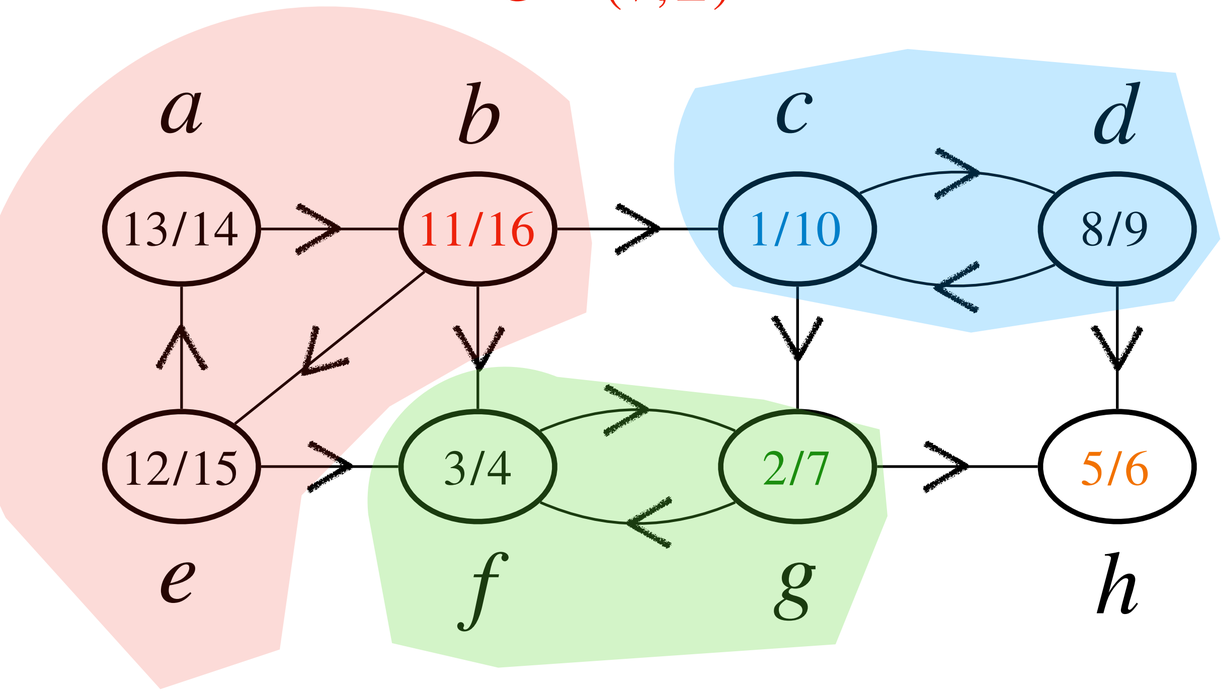
$G^T = (V, E^T)$: transpose of G



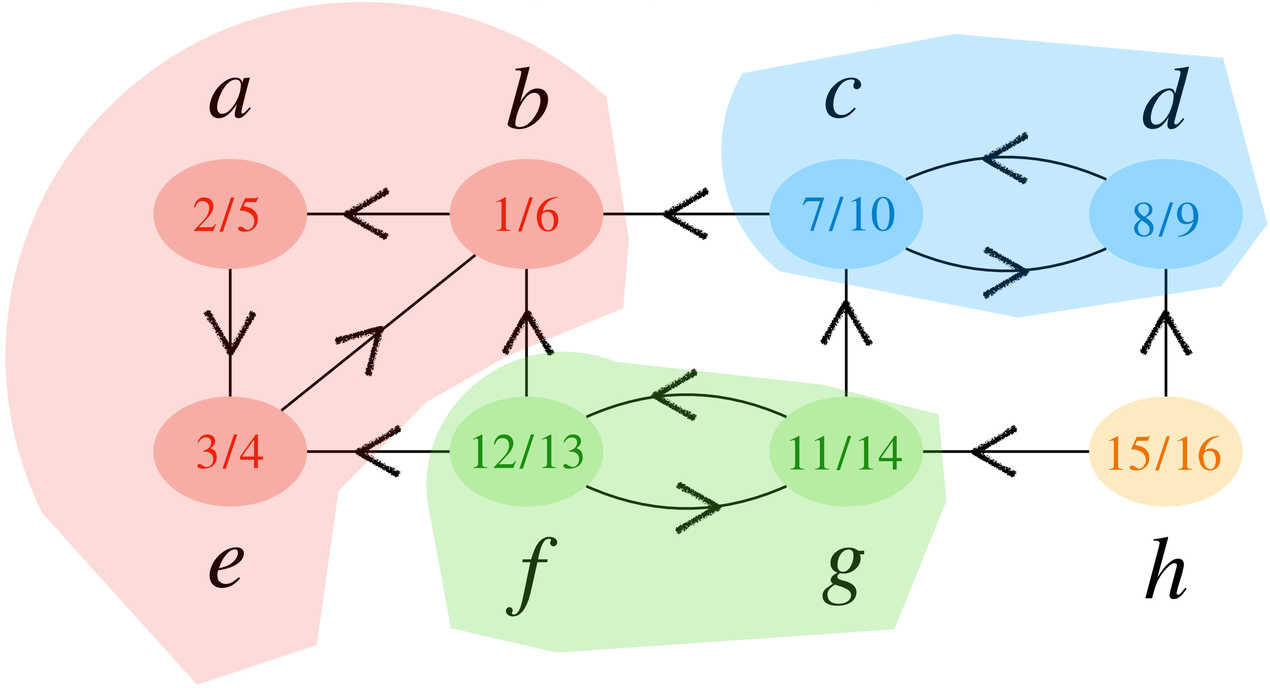
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



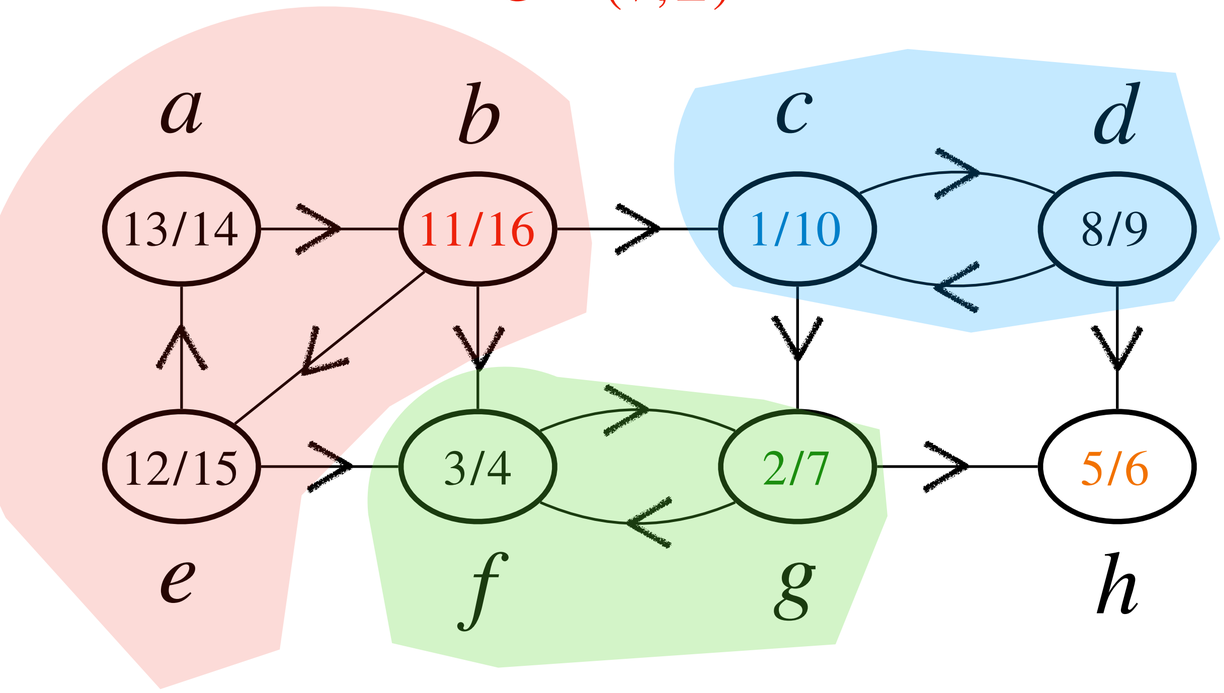
$G^T = (V, E^T)$: transpose of G



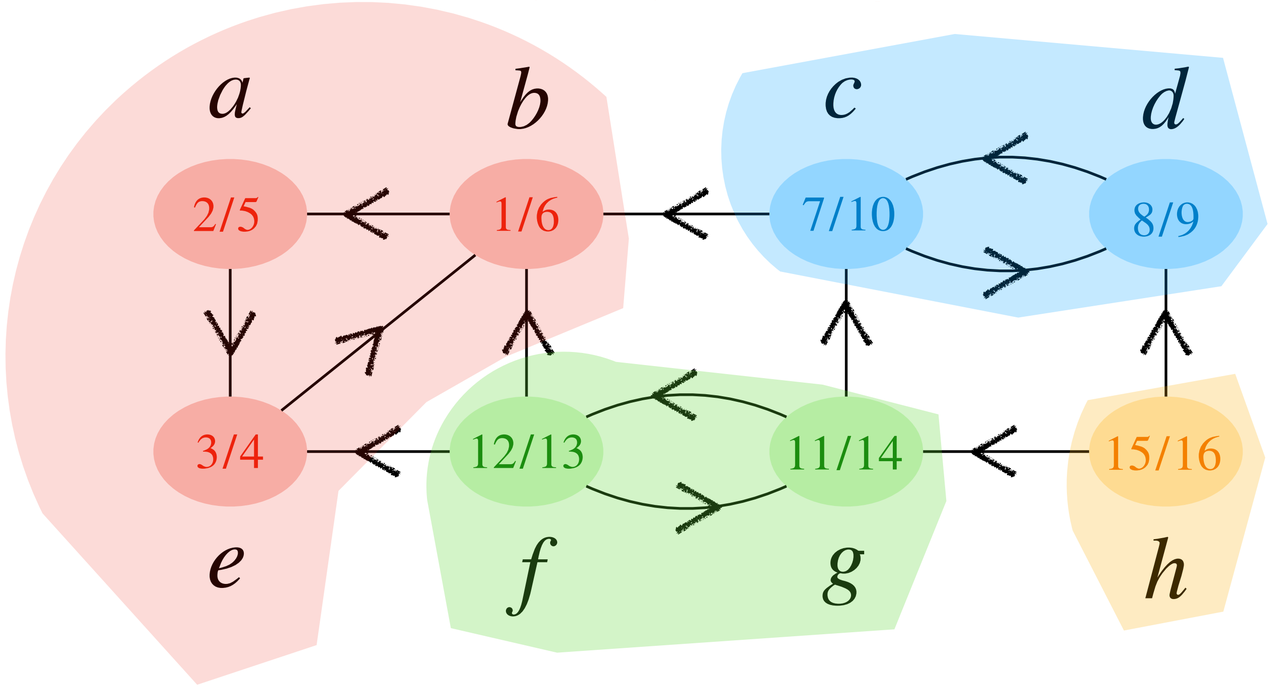
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



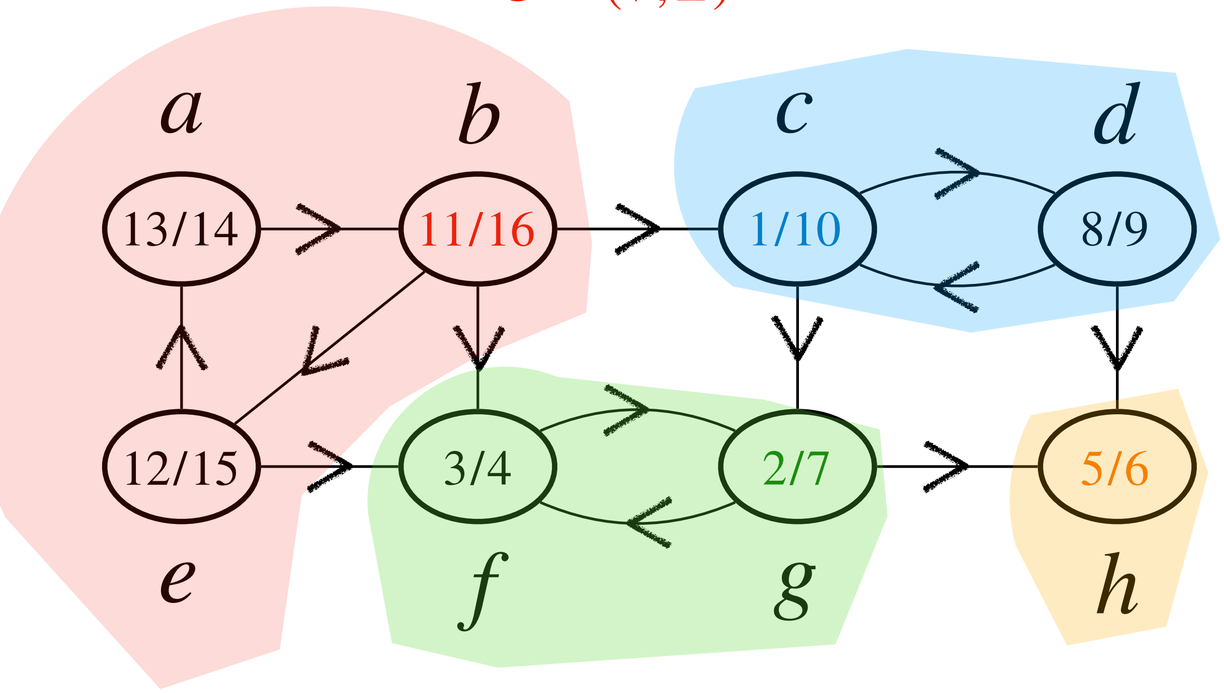
$G^T = (V, E^T)$: transpose of G



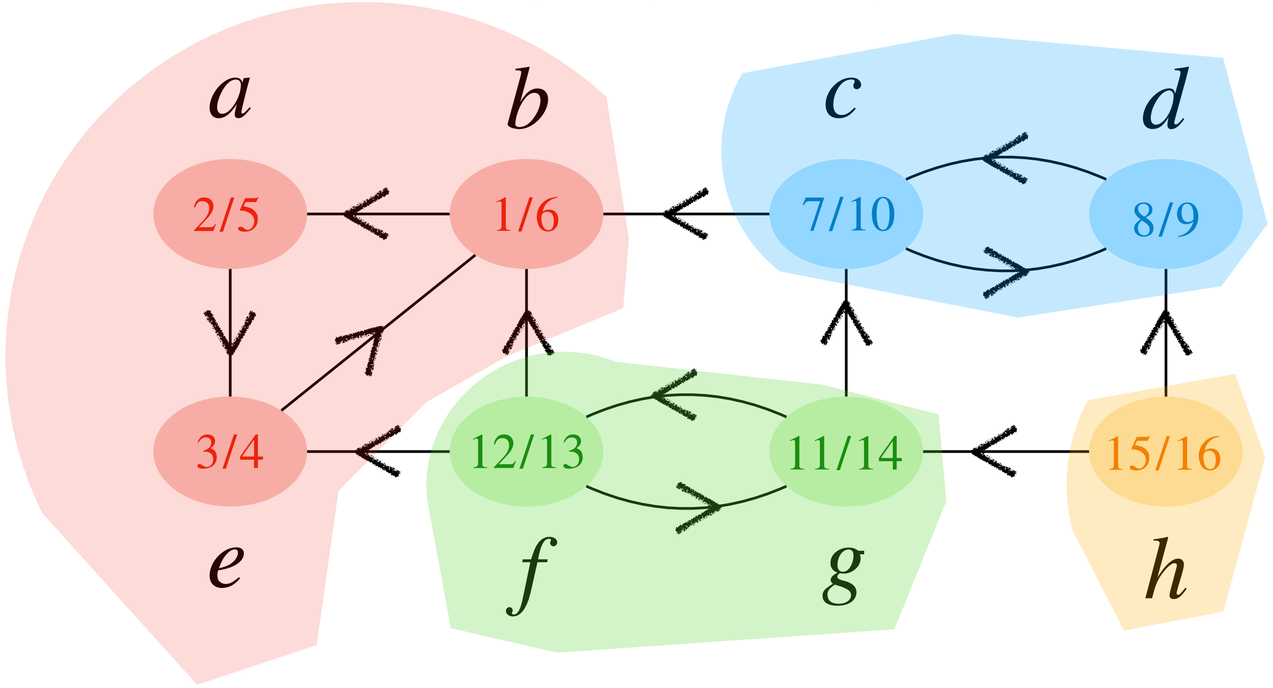
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



$G^T = (V, E^T)$: transpose of G

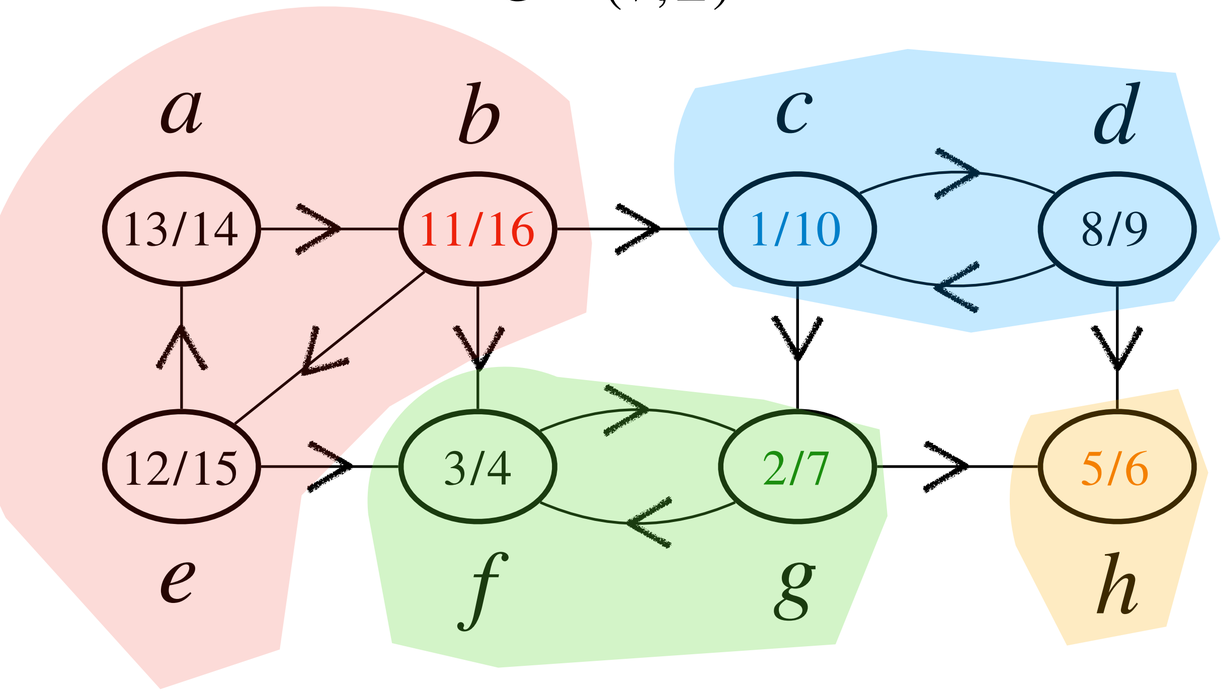


Time Complexity

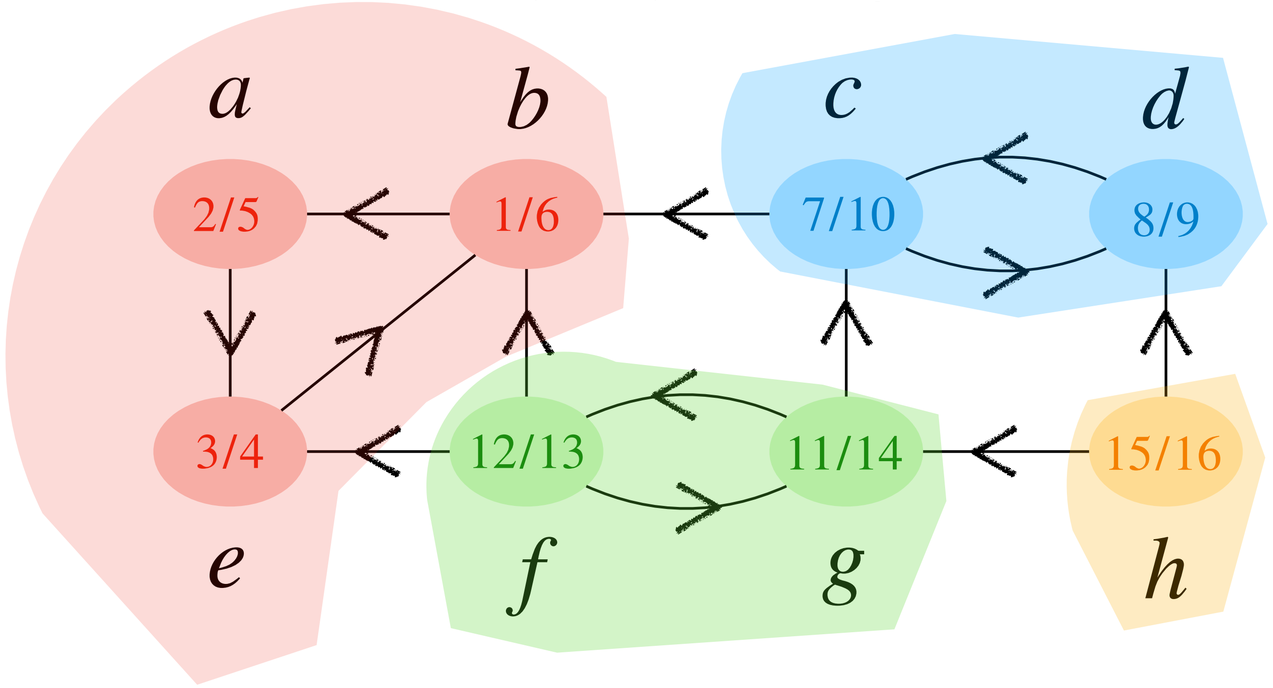
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u $O(V+E)$
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



$G^T = (V, E^T)$: transpose of G

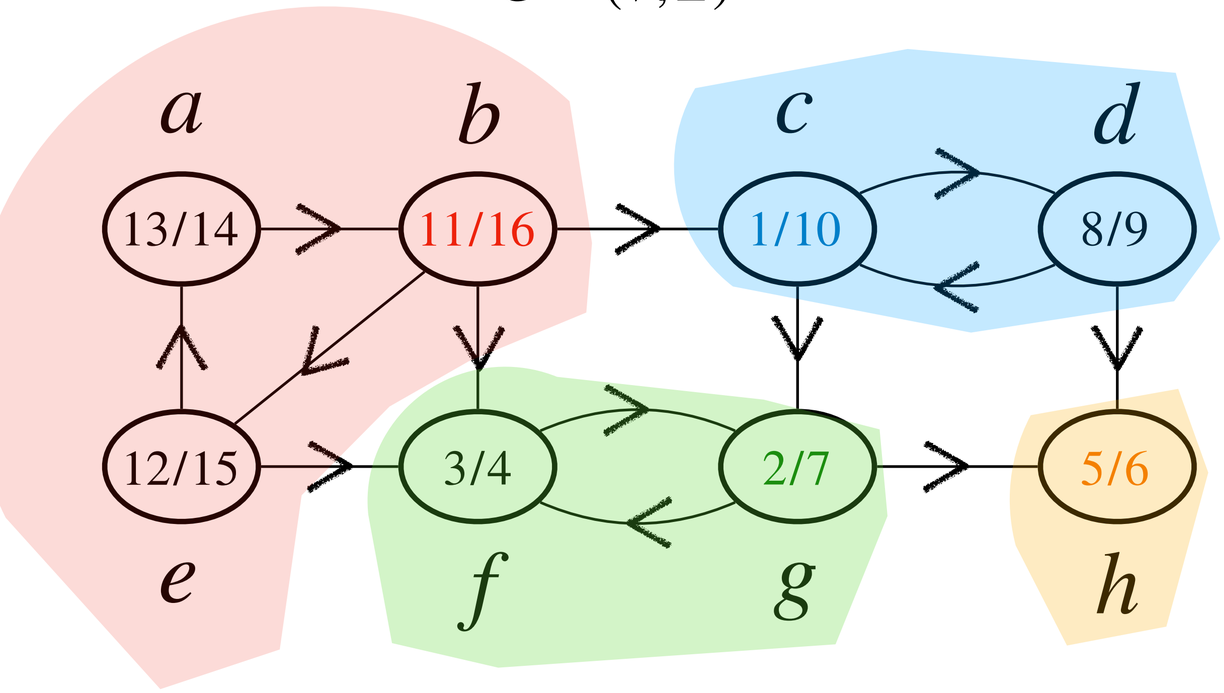


Time Complexity

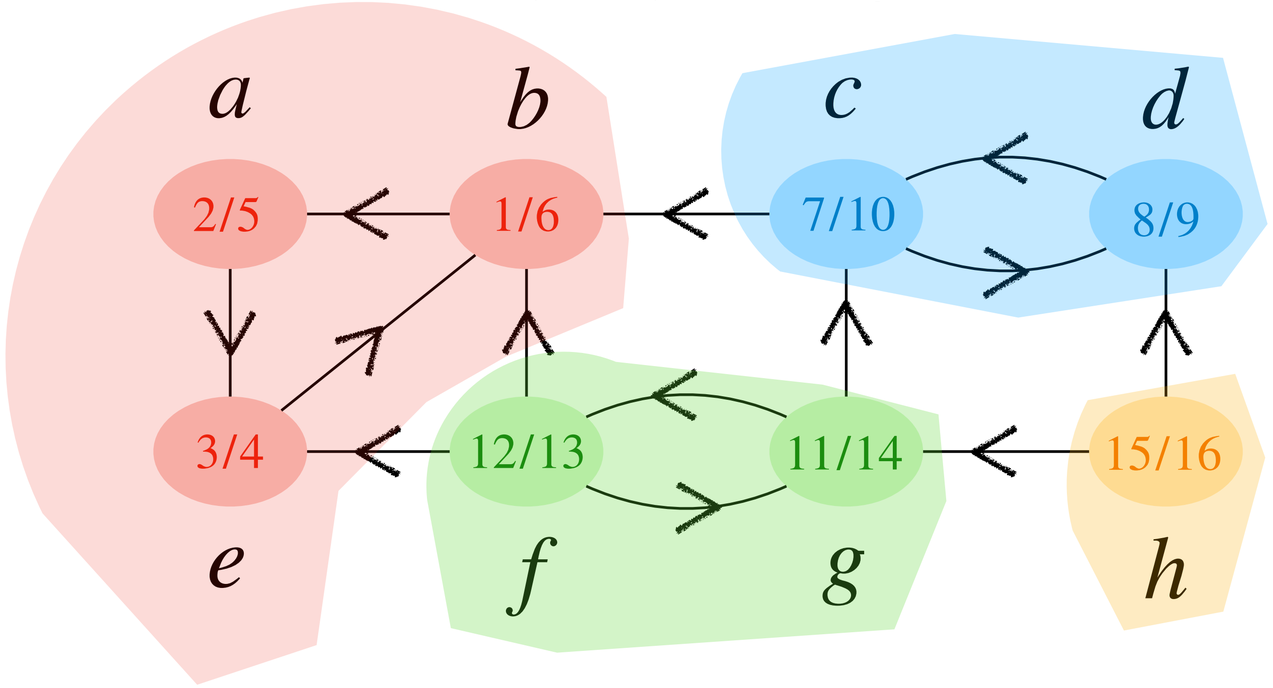
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u $O(V+E)$
2. create the transpose graph G^T $O(V+E)$
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



$G^T = (V, E^T)$: transpose of G

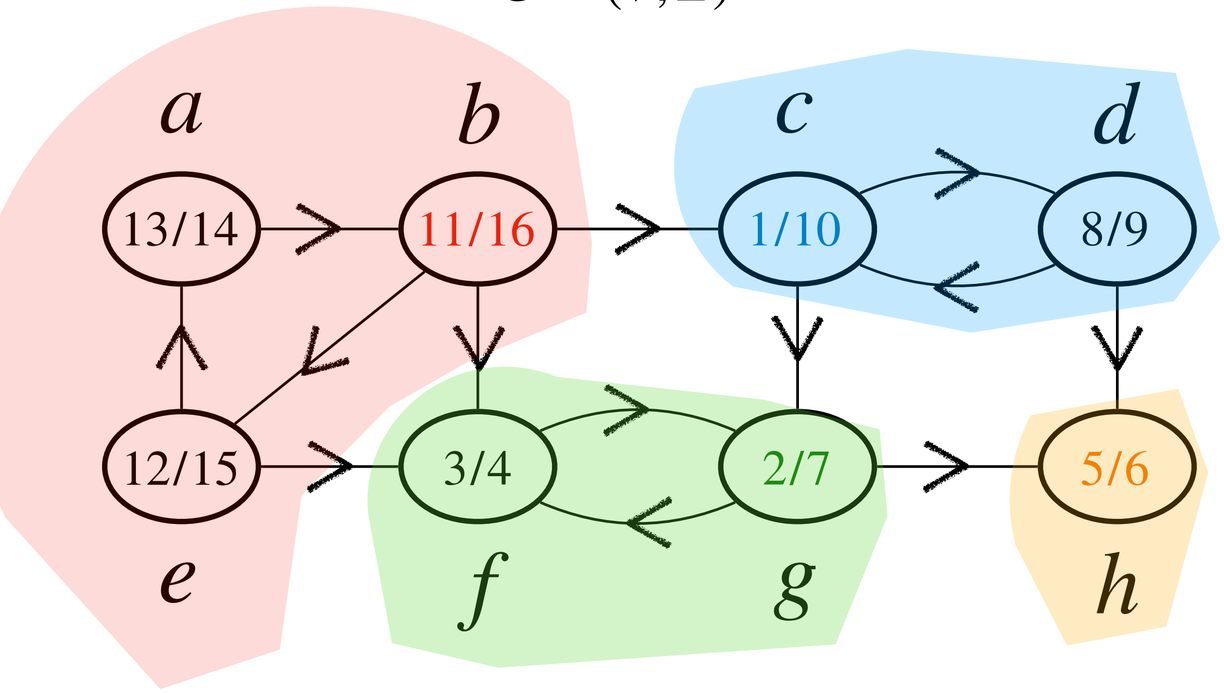


Time Complexity

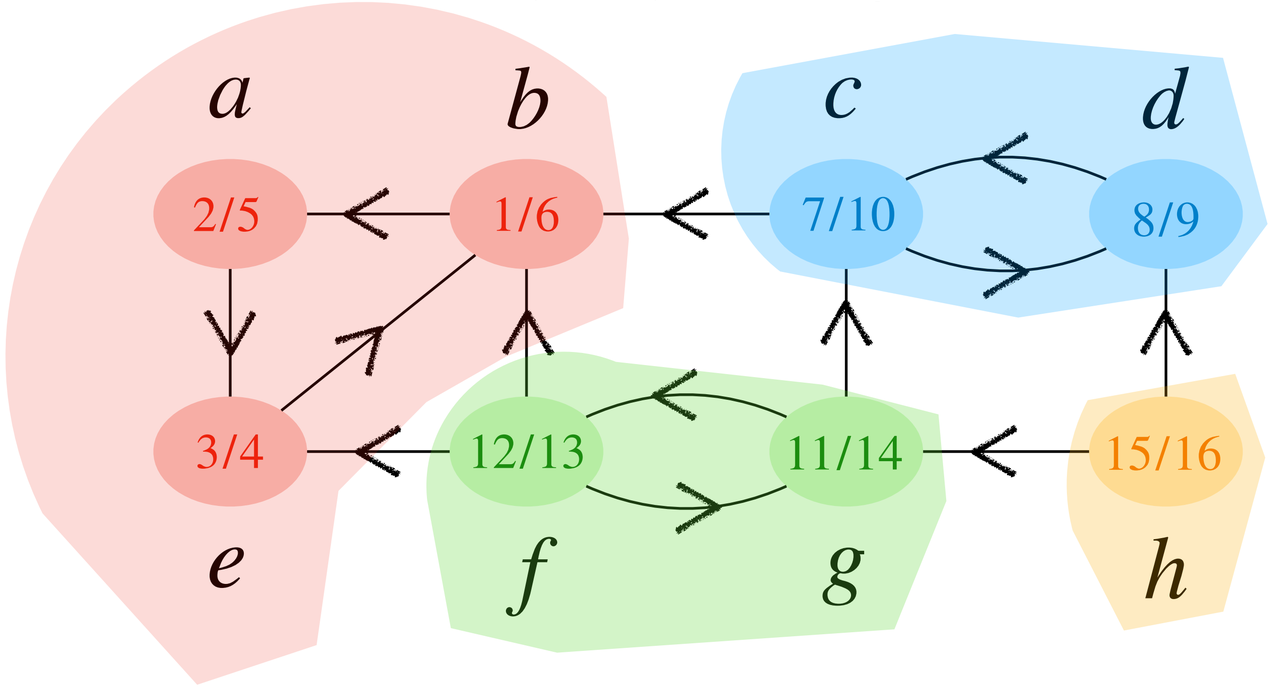
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u $O(V+E)$
2. create the transpose graph G^T $O(V+E)$
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$ $O(V+E)$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



$G^T = (V, E^T)$: transpose of G

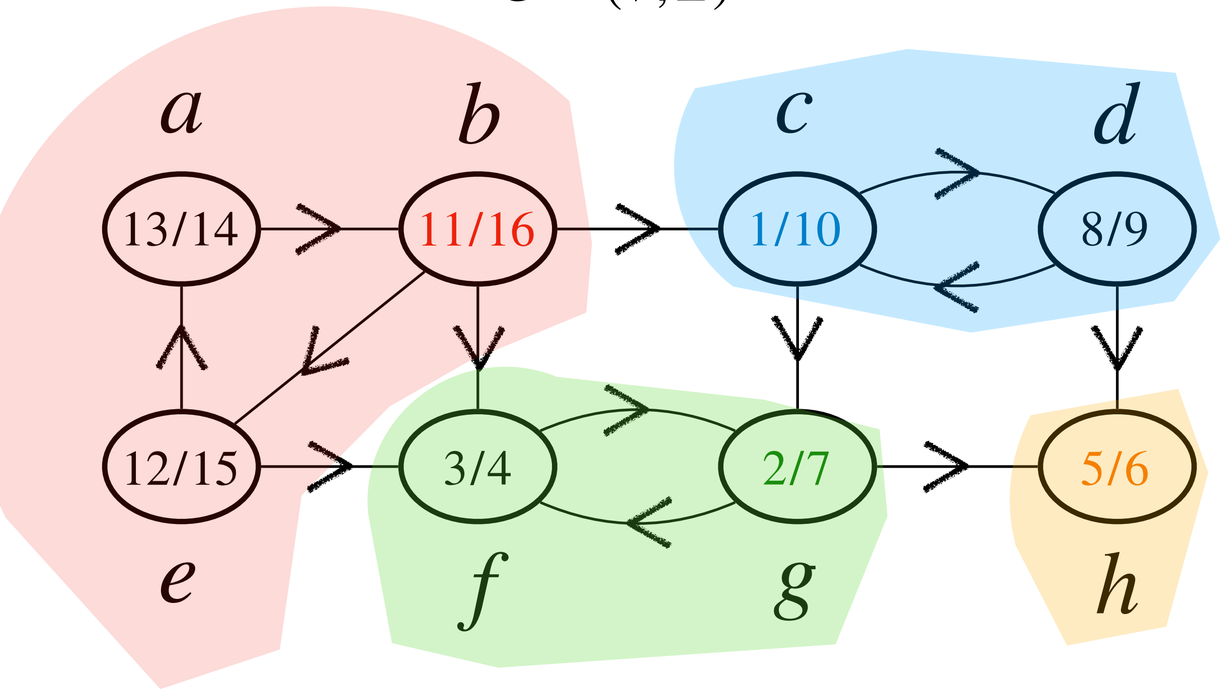


Time Complexity

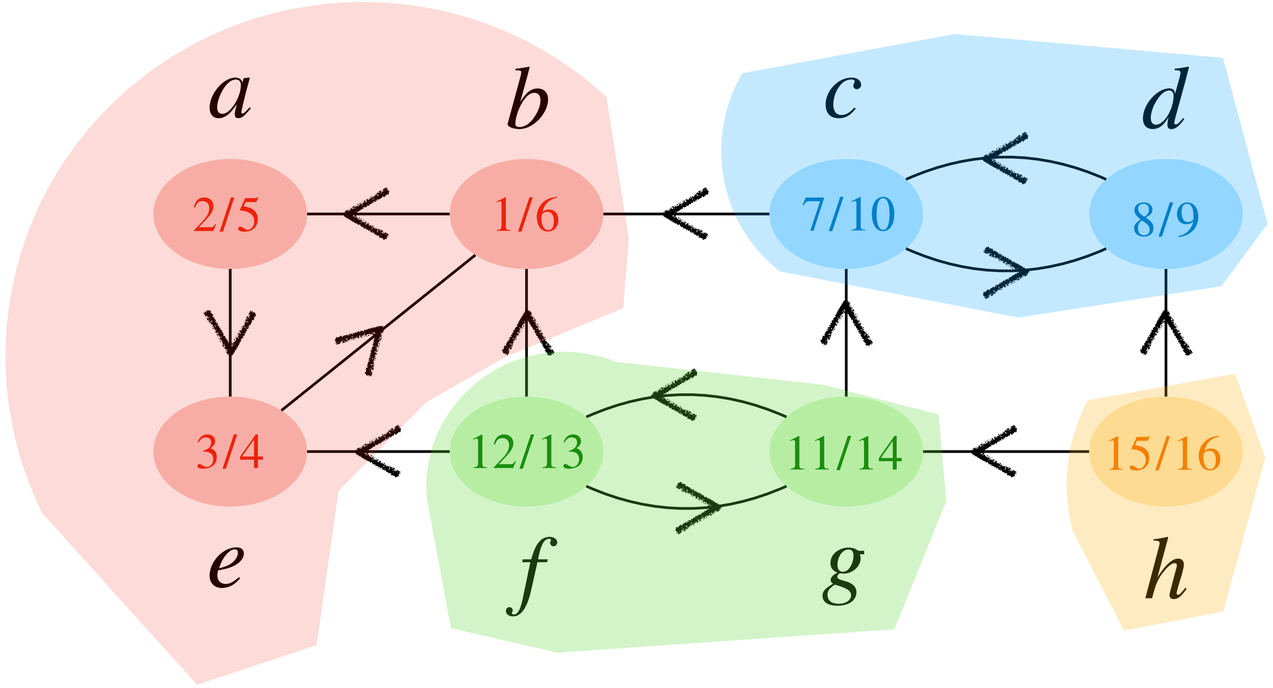
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u $O(V+E)$
2. create the transpose graph G^T $O(V+E)$
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$ $O(V+E)$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC. $O(V+E)$

$G = (V, E)$



$G^T = (V, E^T)$: transpose of G



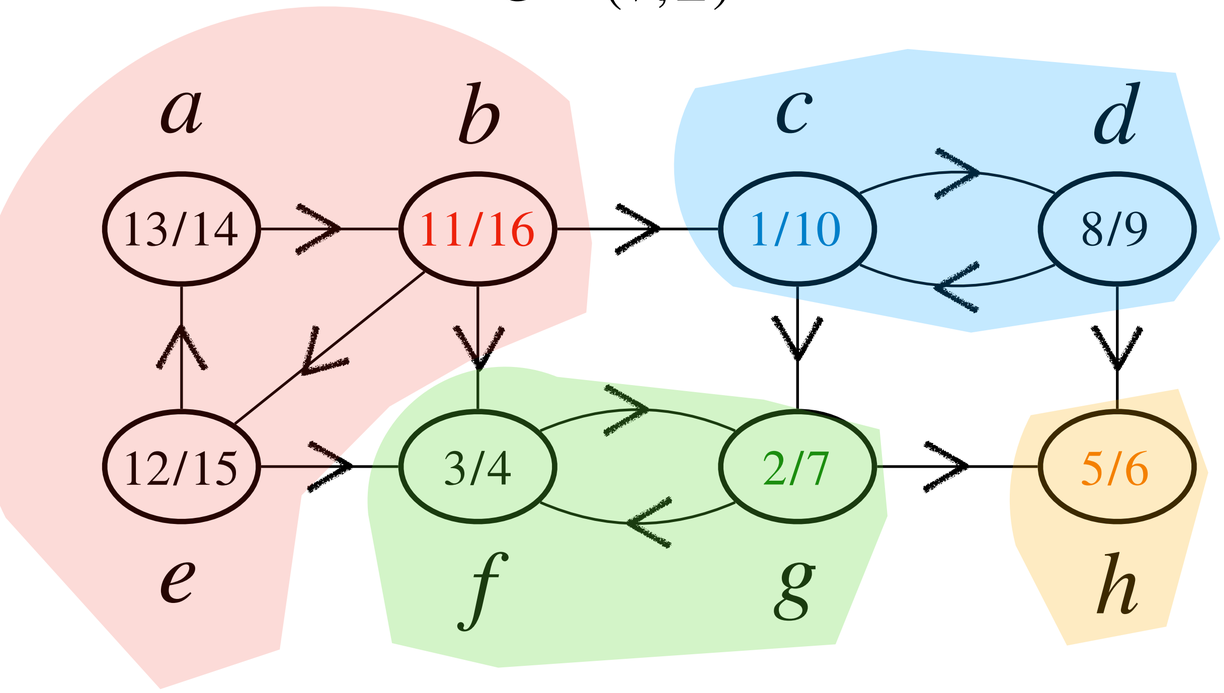
Time Complexity

$O(V+E)$

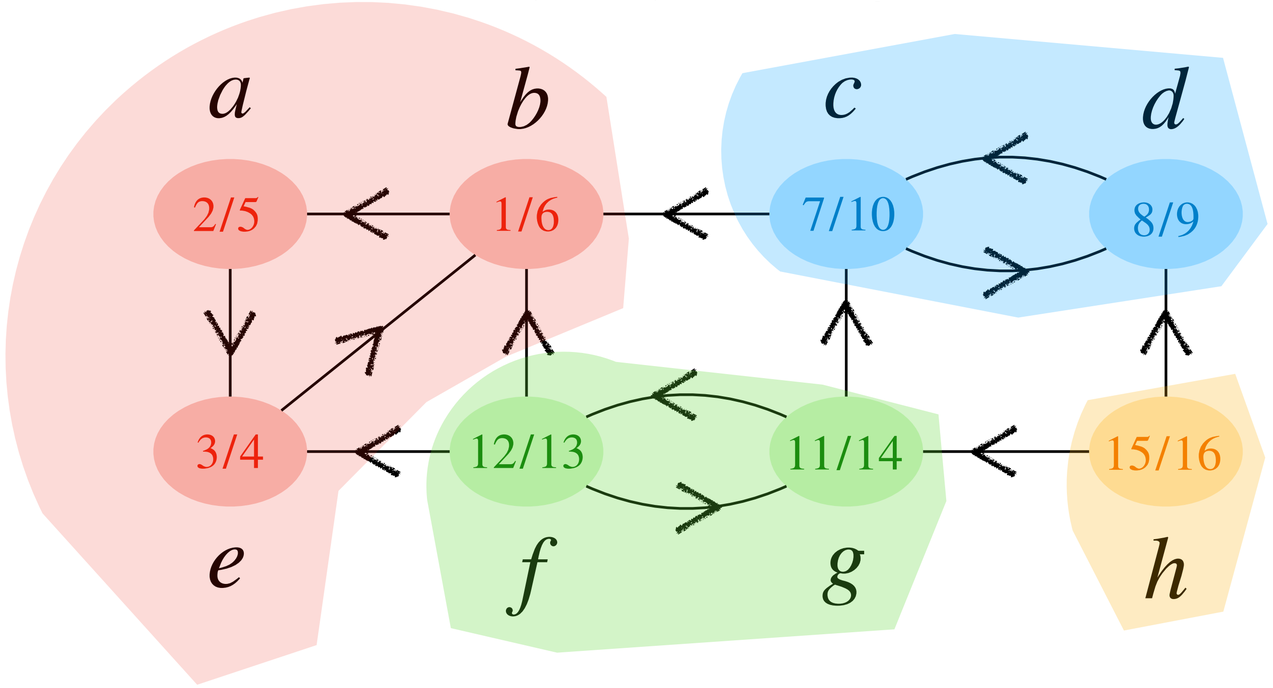
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



$G^T = (V, E^T)$: transpose of G



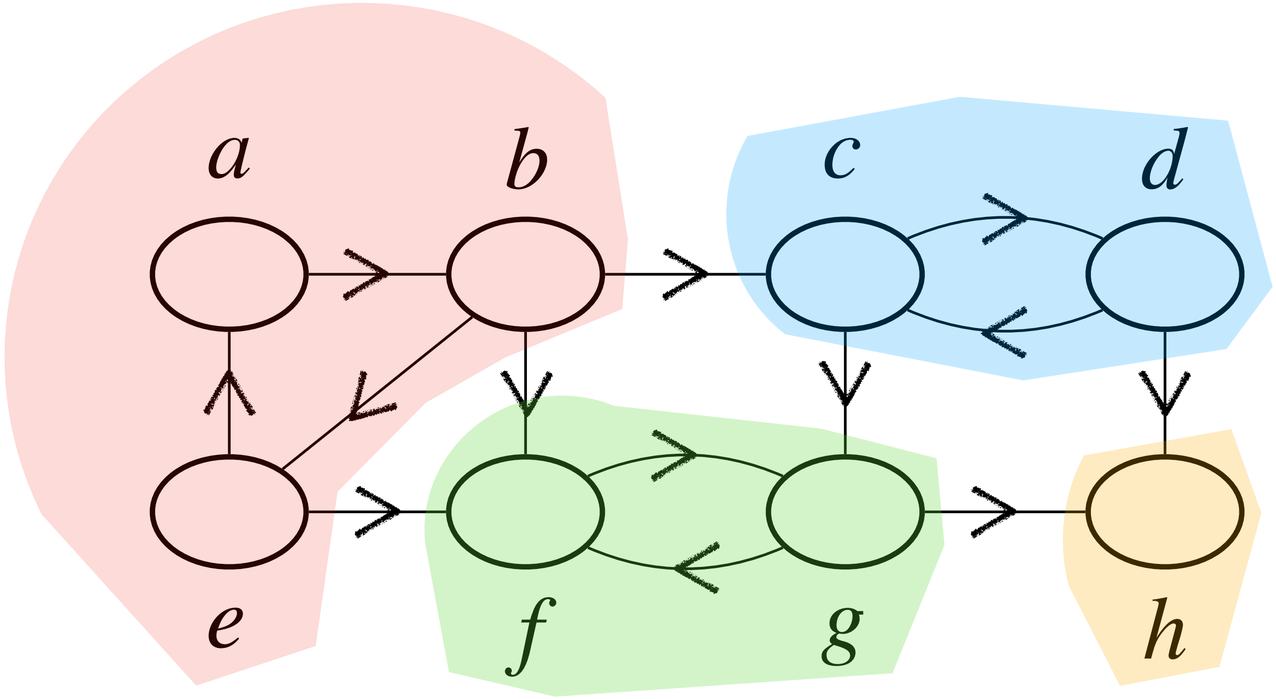
Algorithm for the Strongly Connected Component Problem:

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

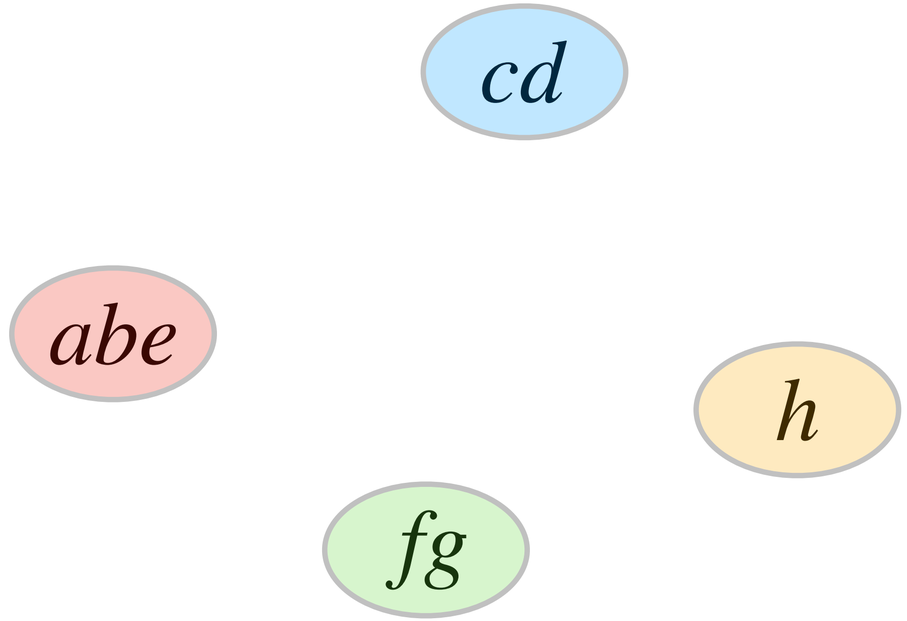
Proof of Correctness

Define Component Graph

$G = (V, E)$

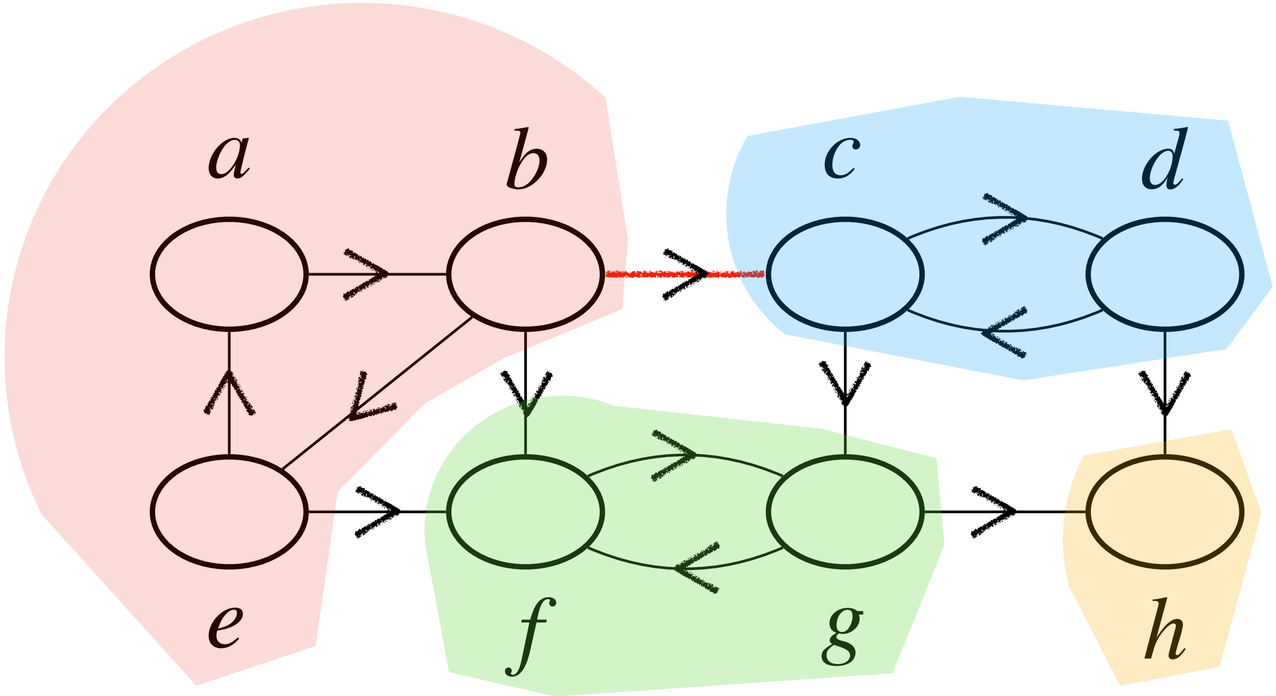


Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$

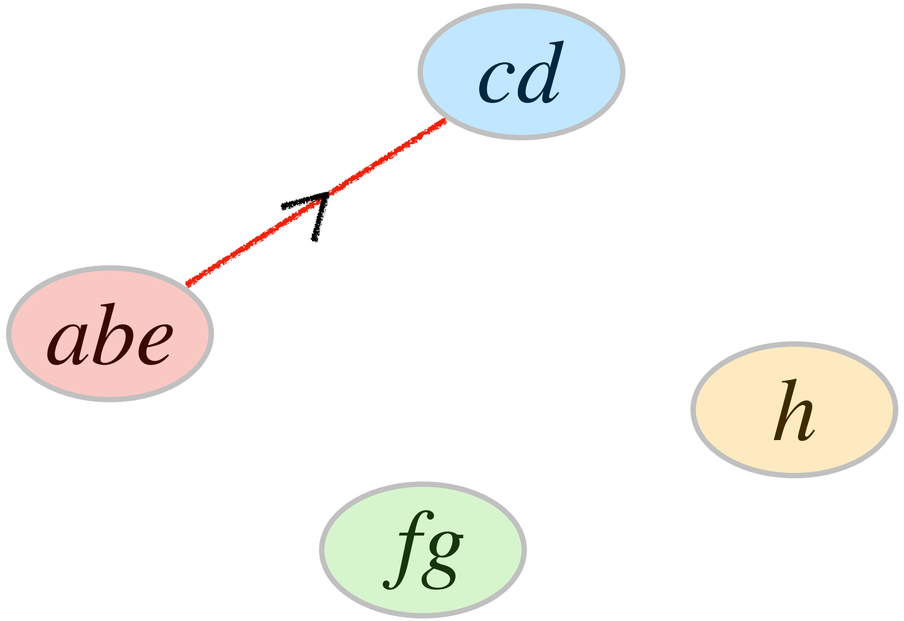


Define Component Graph

$G = (V, E)$

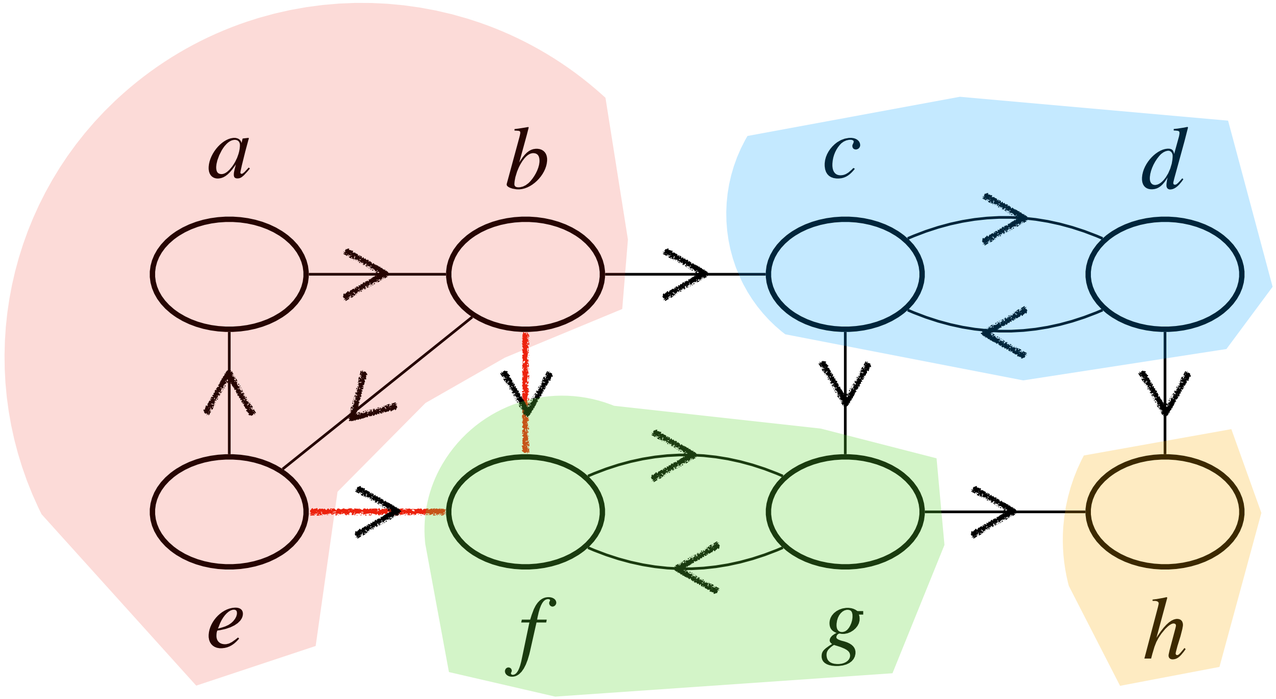


Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$

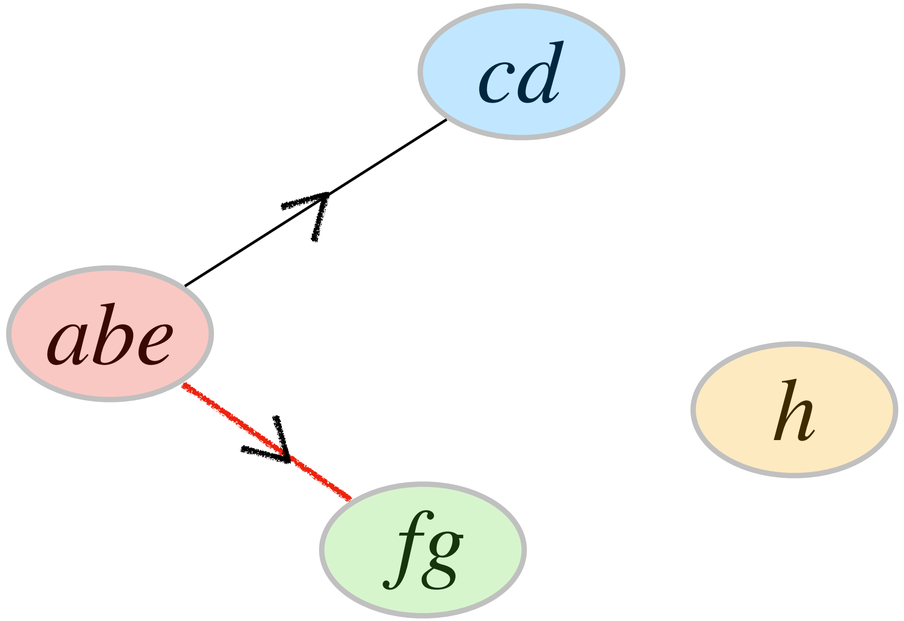


Define Component Graph

$G = (V, E)$

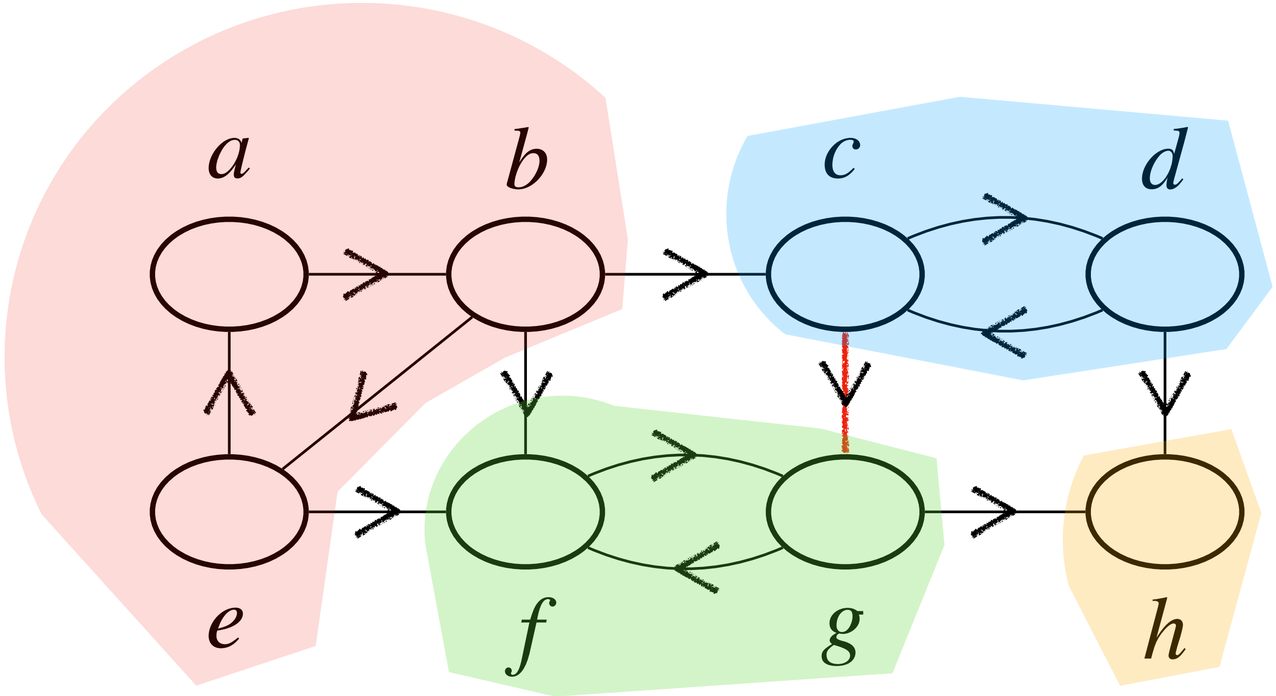


Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$

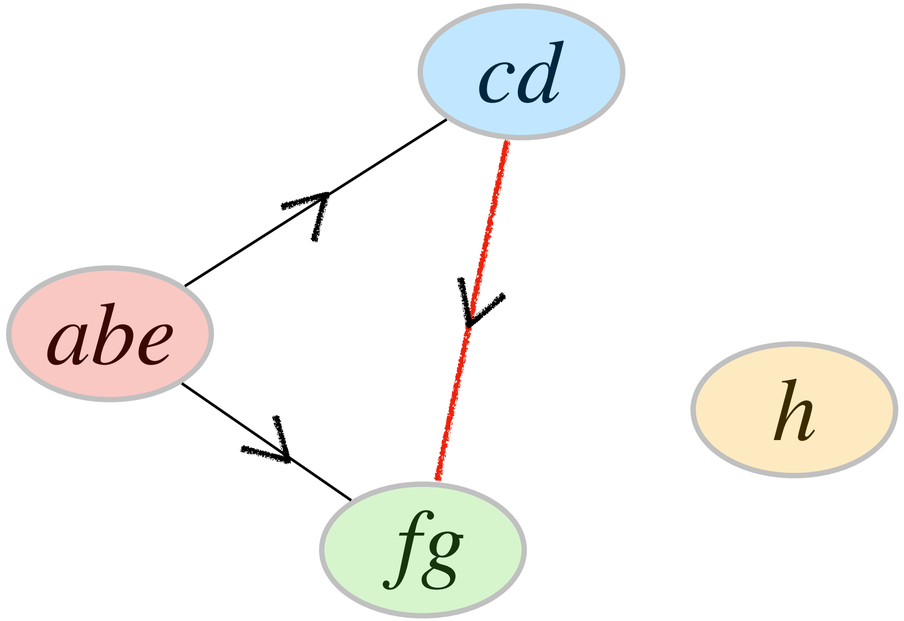


Define Component Graph

$G = (V, E)$

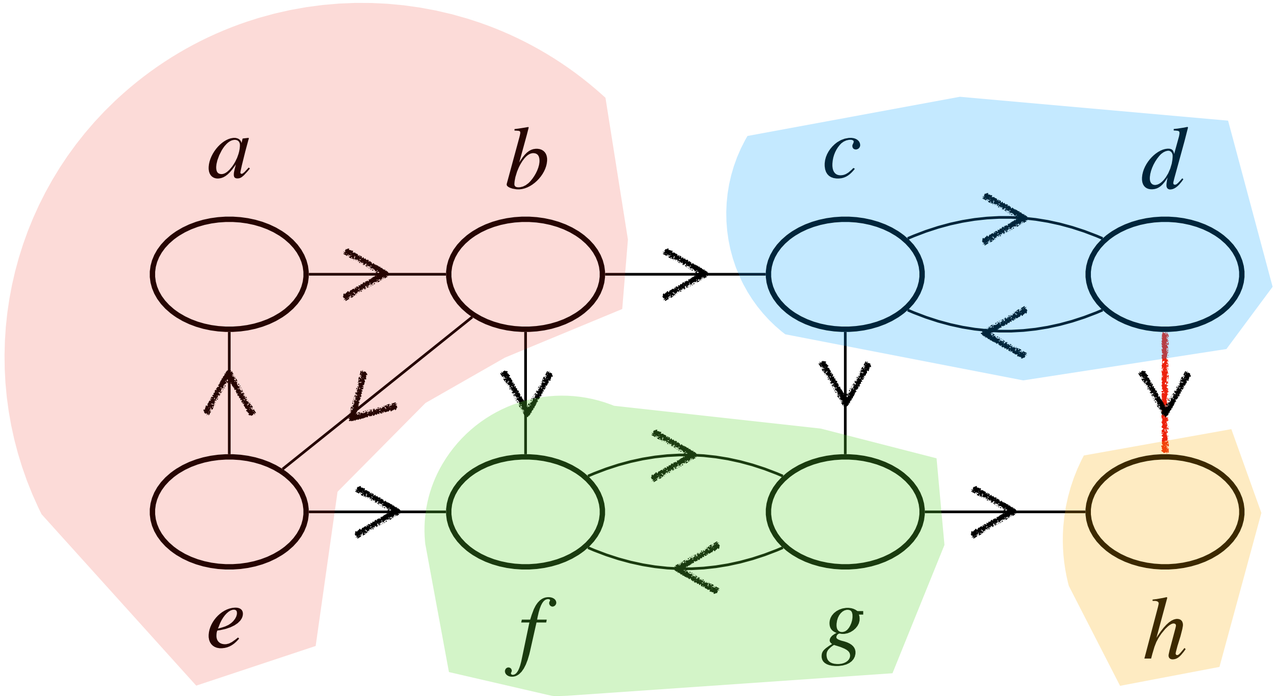


Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$

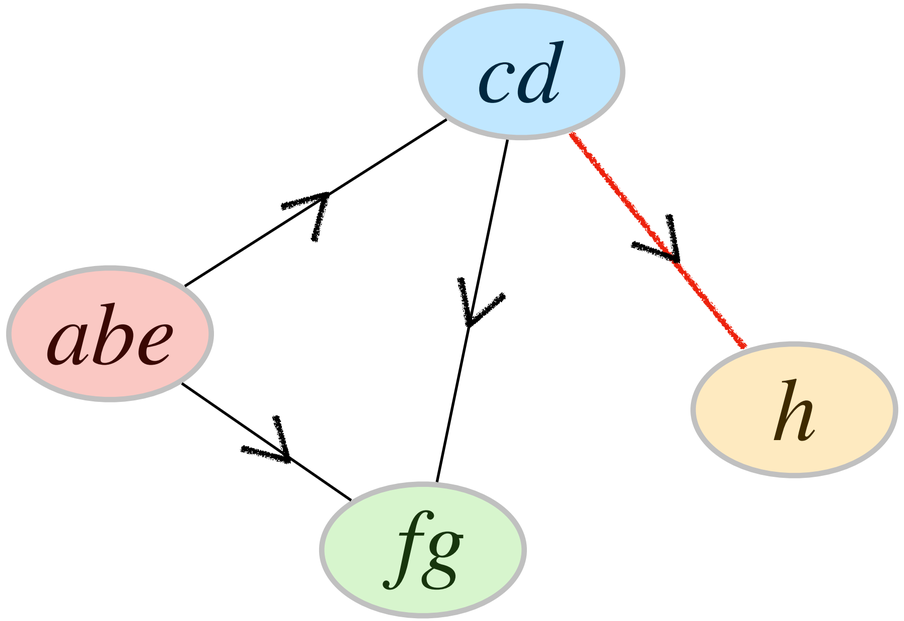


Define Component Graph

$G = (V, E)$

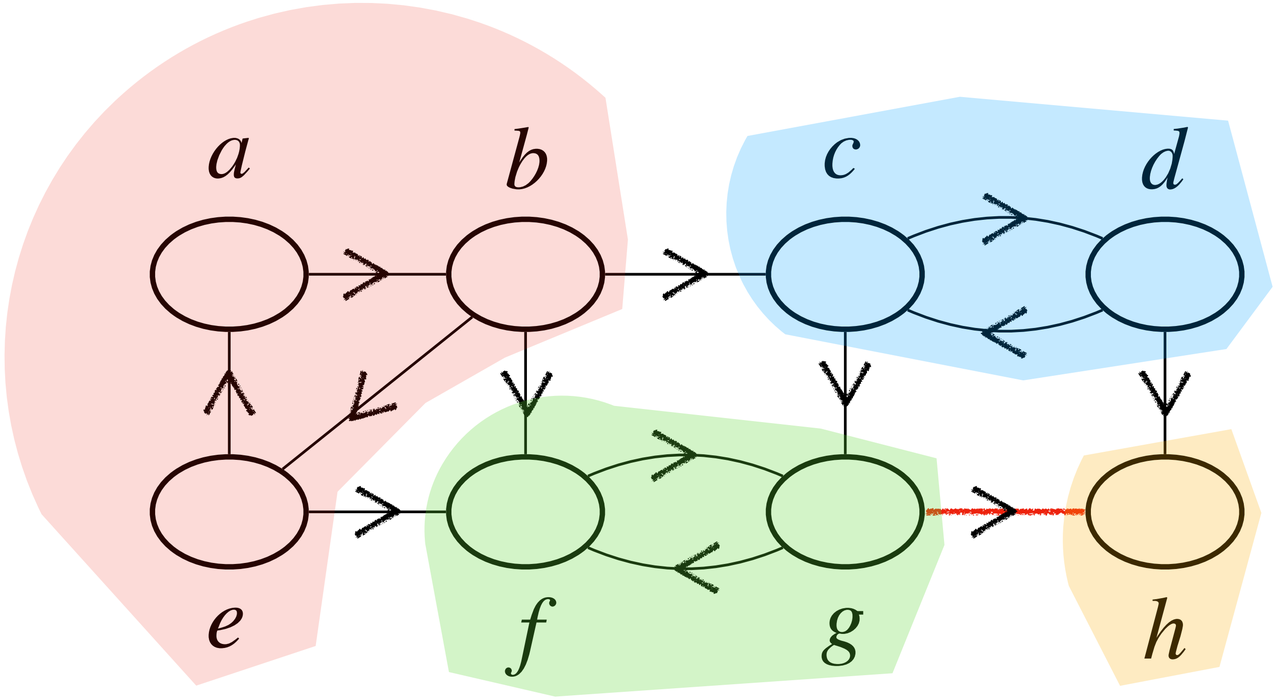


Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$

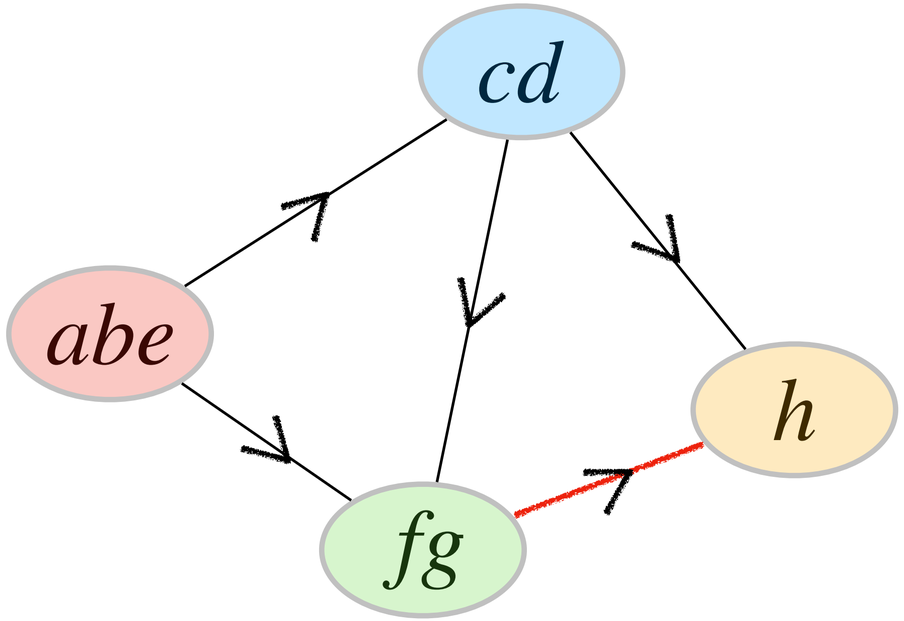


Define Component Graph

$G = (V, E)$



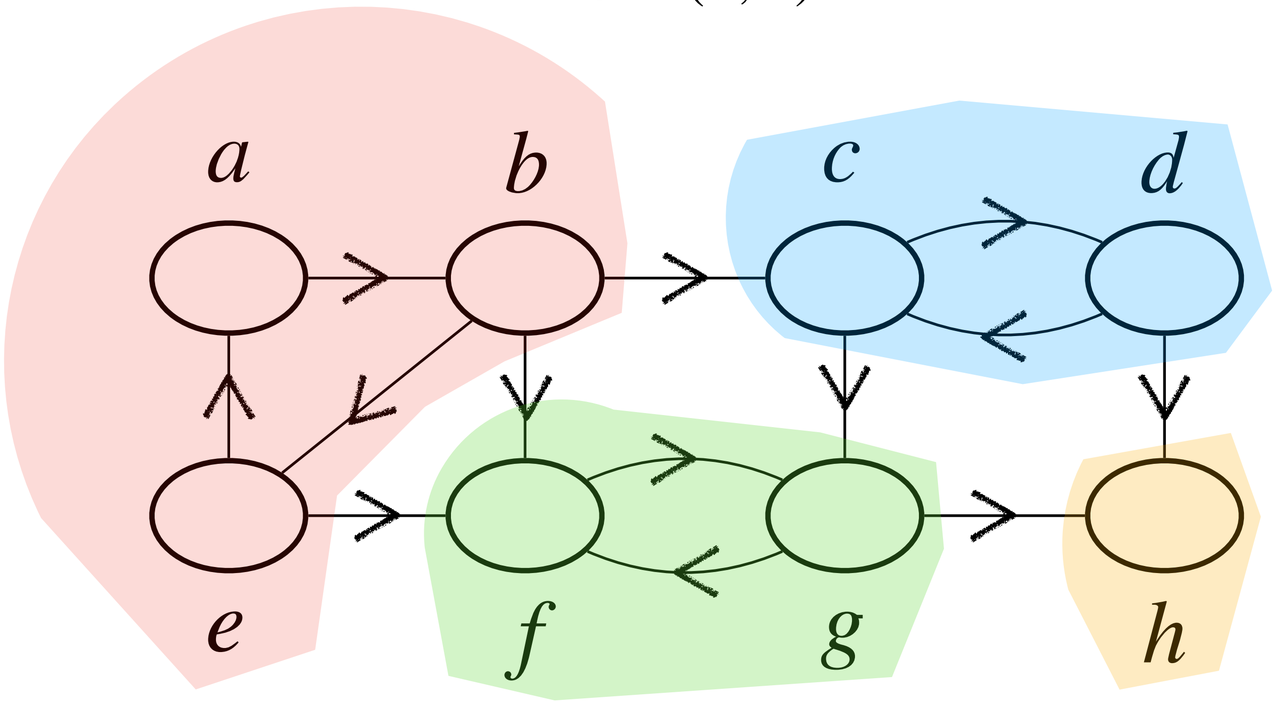
Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$



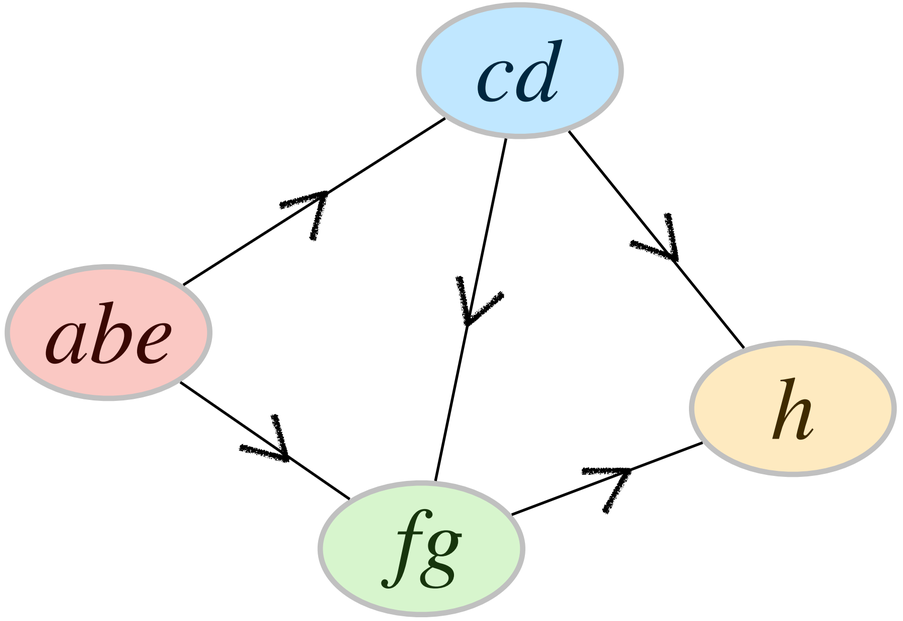
We will see that:

1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

$G = (V, E)$



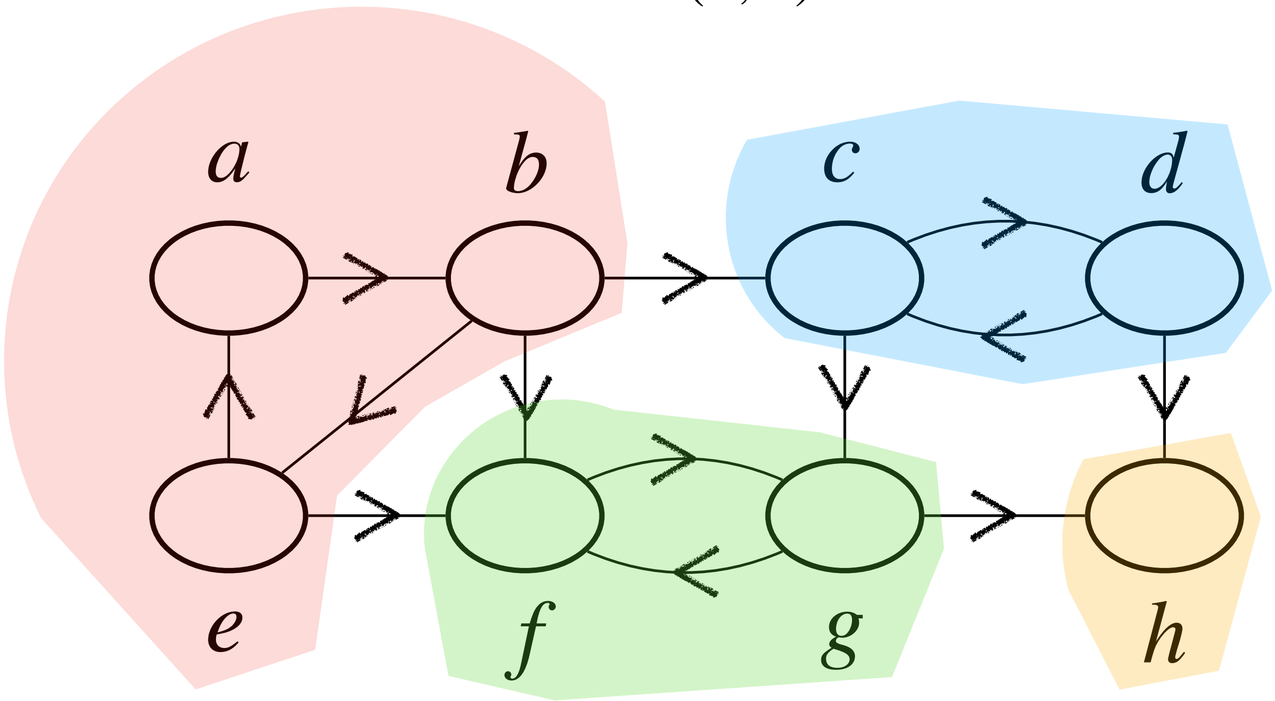
Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$



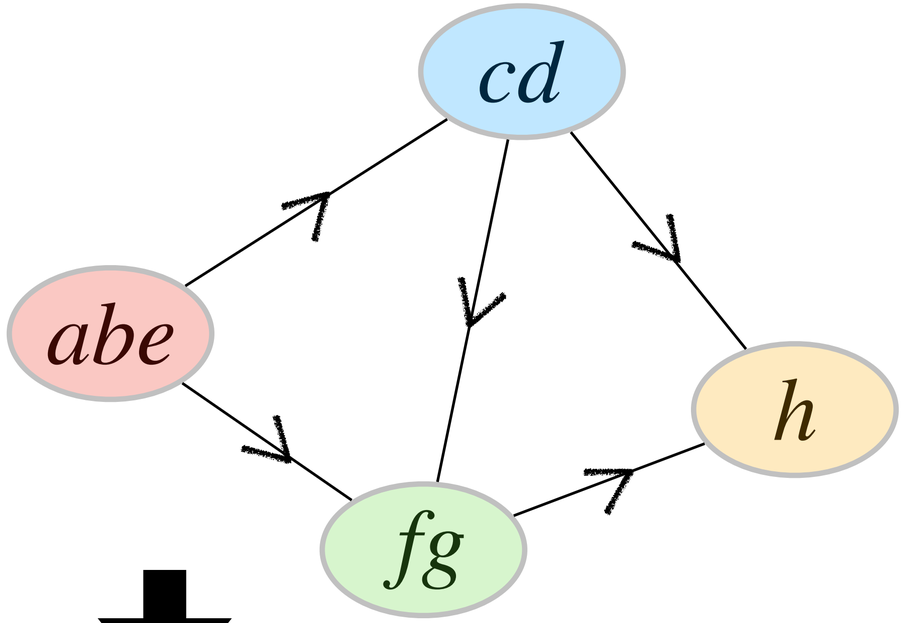
We will see that:

1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

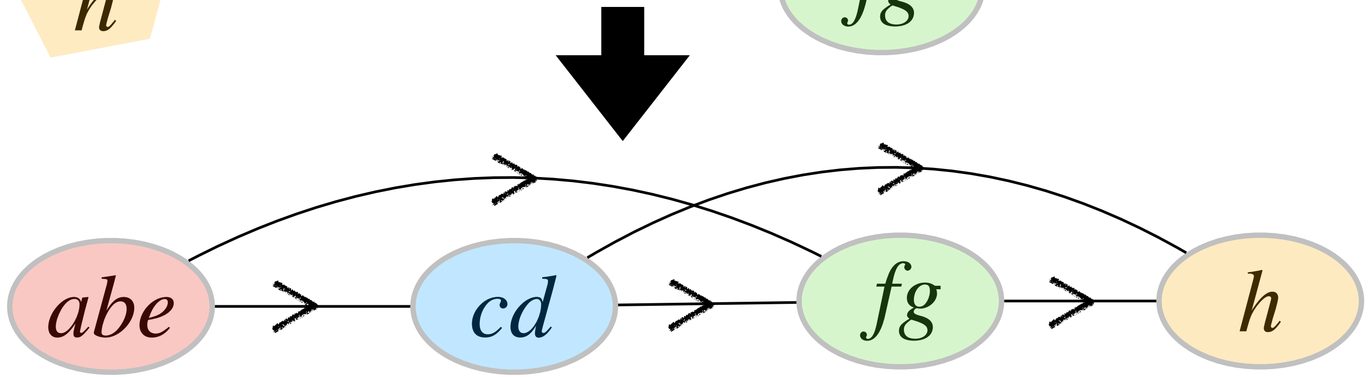
$G = (V, E)$



Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$



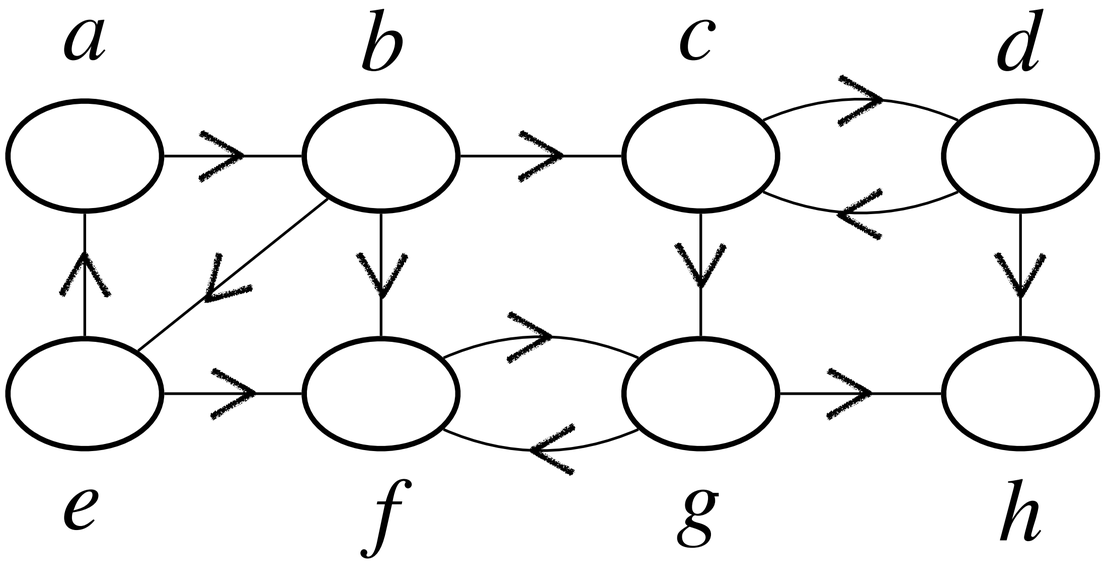
Topological Sort



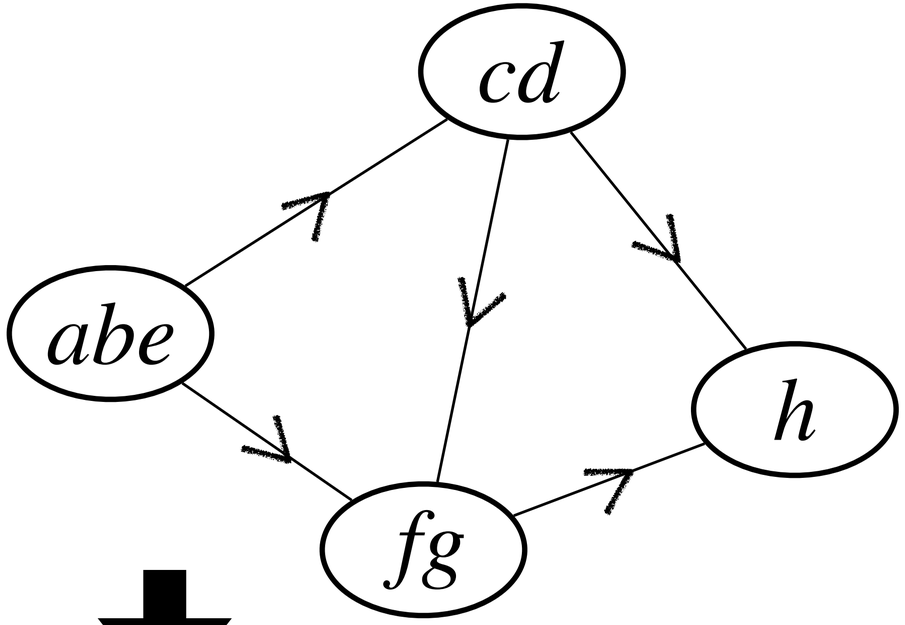
We will see that:

1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

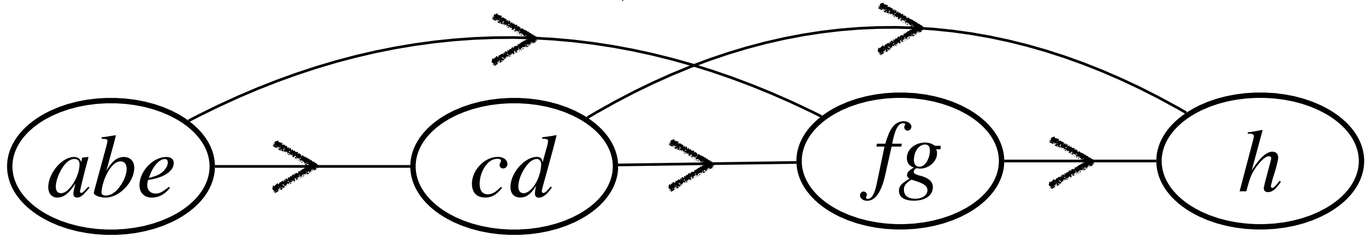
$G = (V, E)$



Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$



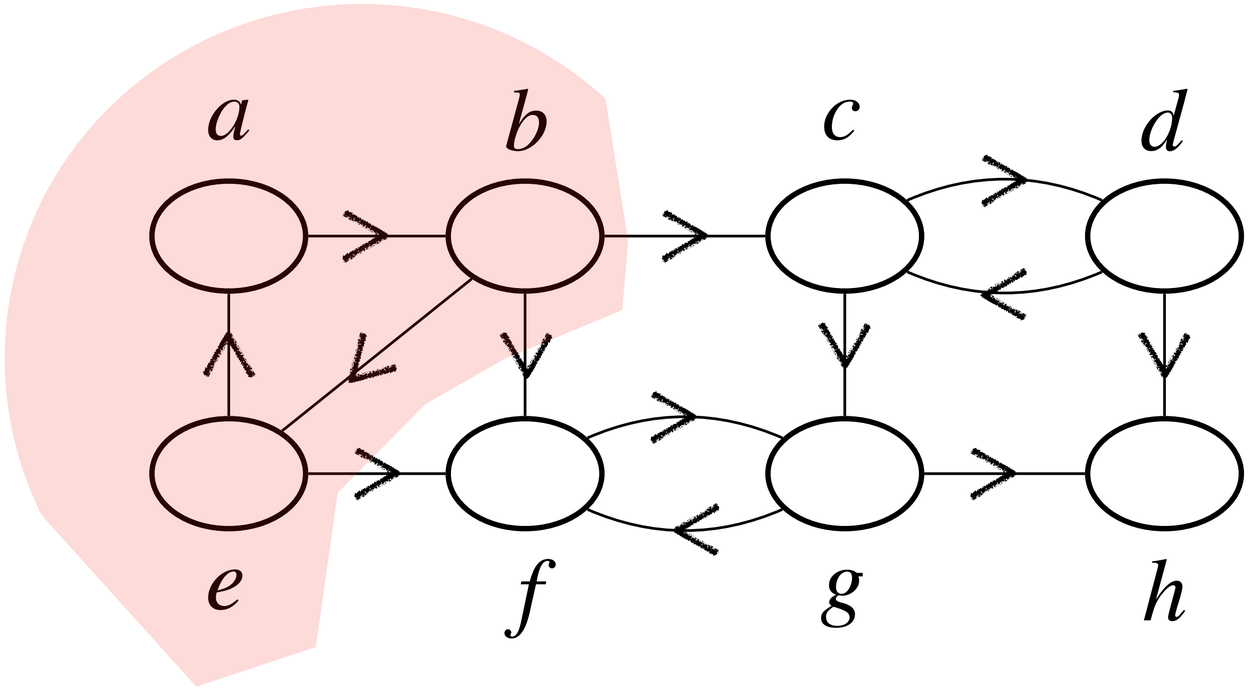
Let's recall how the algorithm worked.



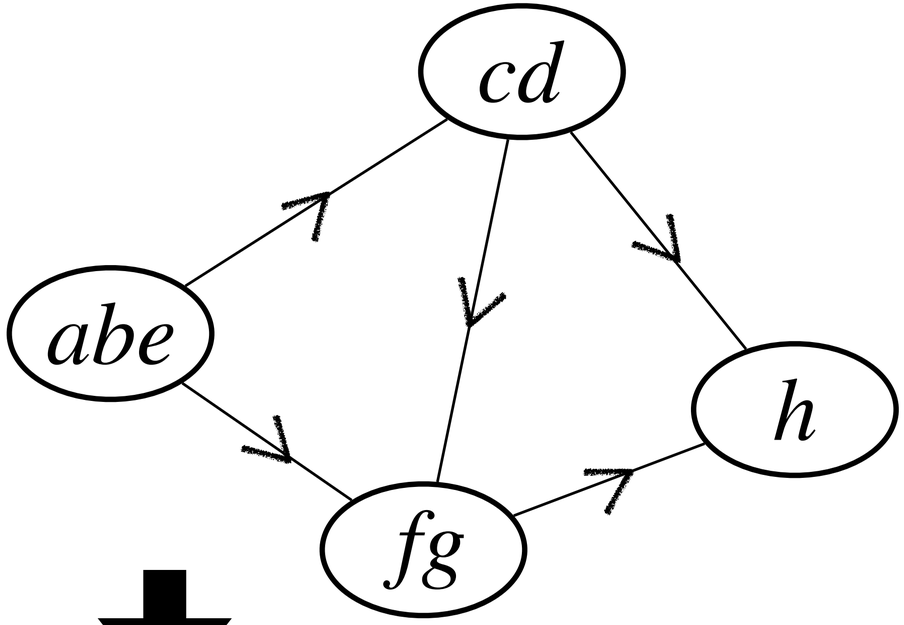
We will see that:

1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

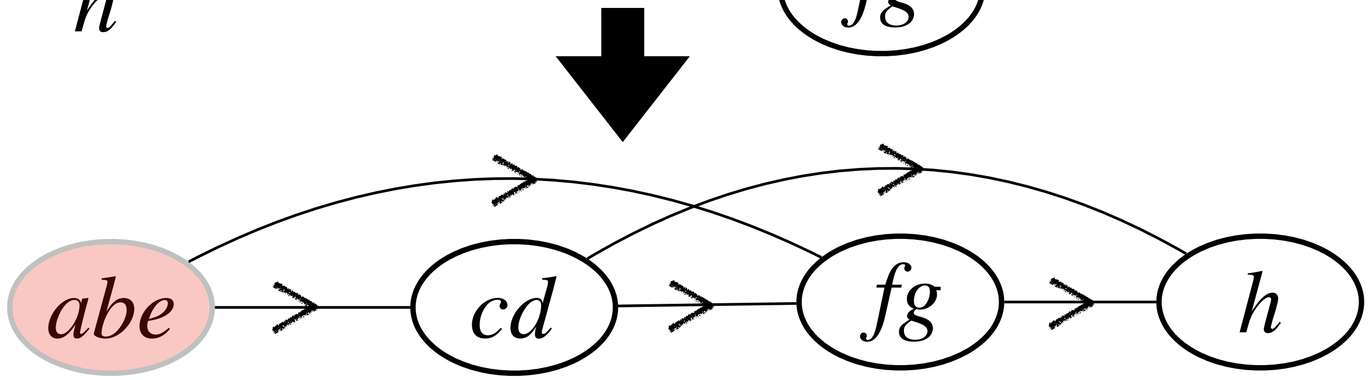
$G = (V, E)$



Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$



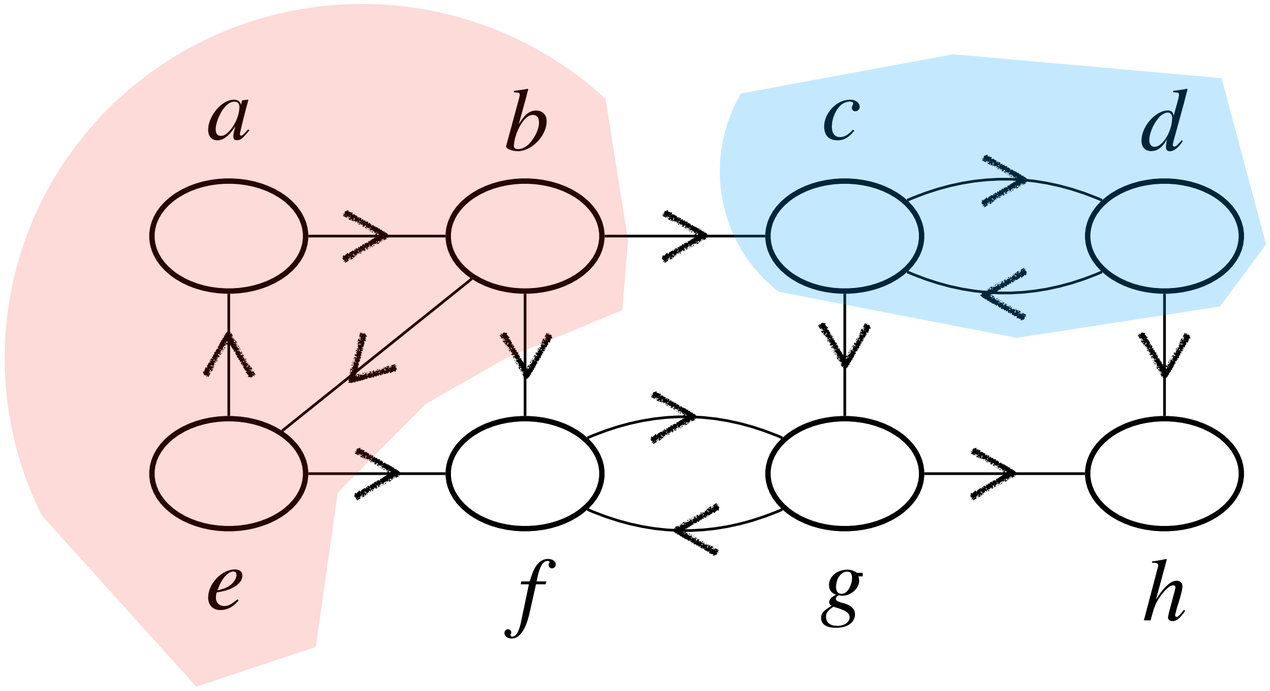
Let's recall how the algorithm worked.



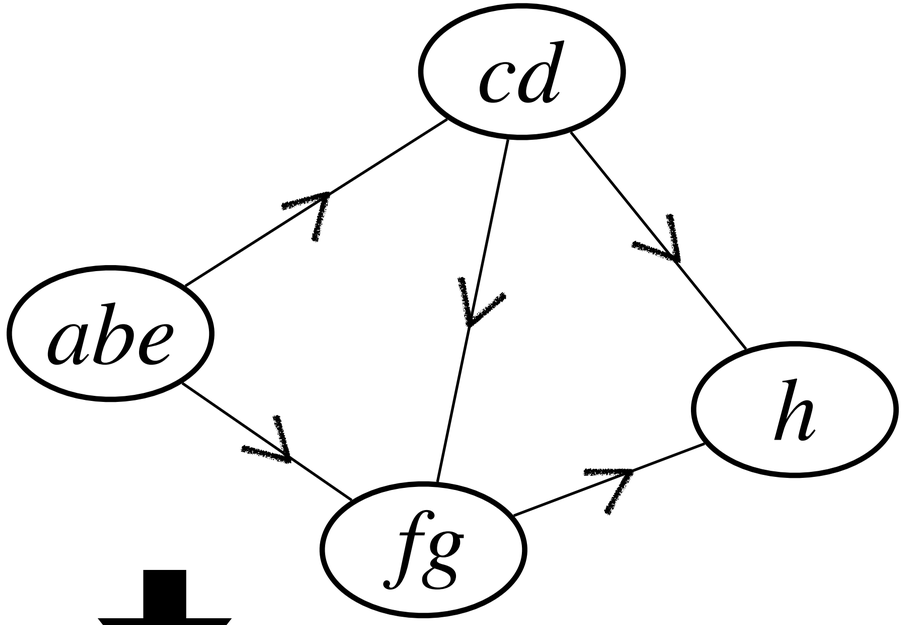
We will see that:

1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

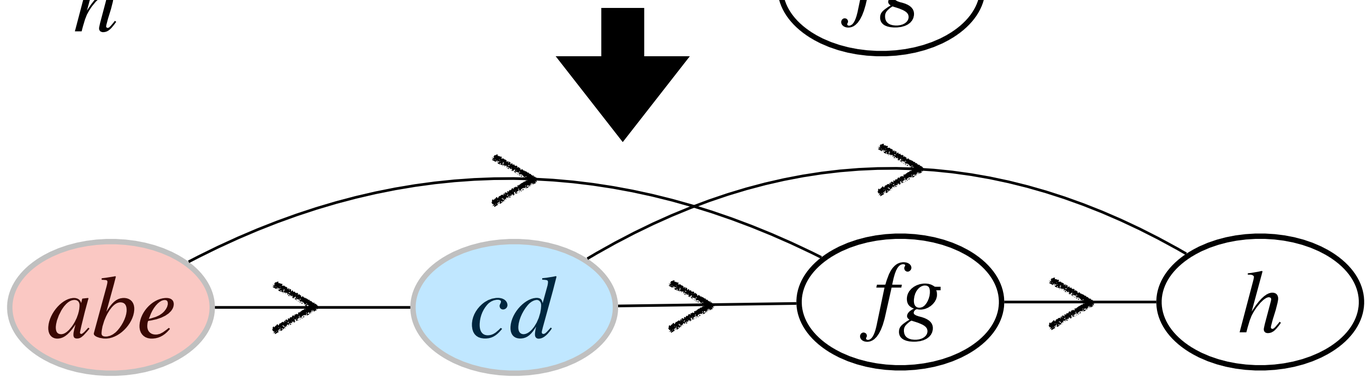
$G = (V, E)$



Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$



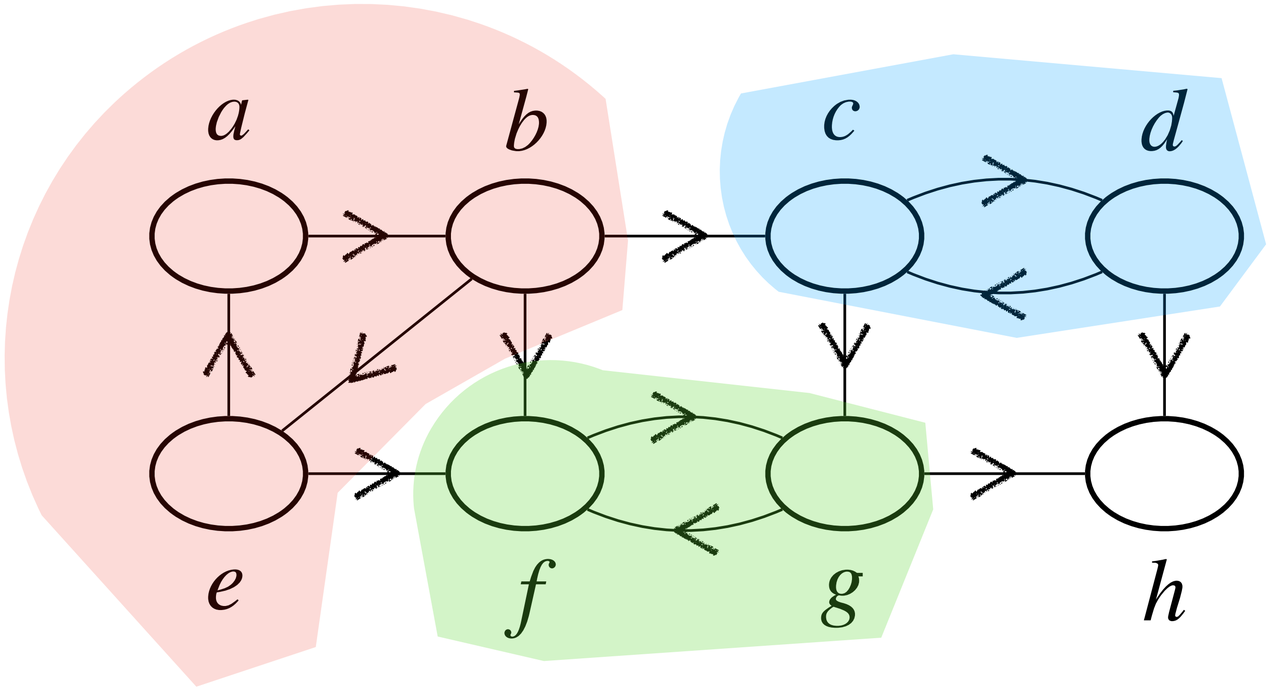
Let's recall how the algorithm worked.



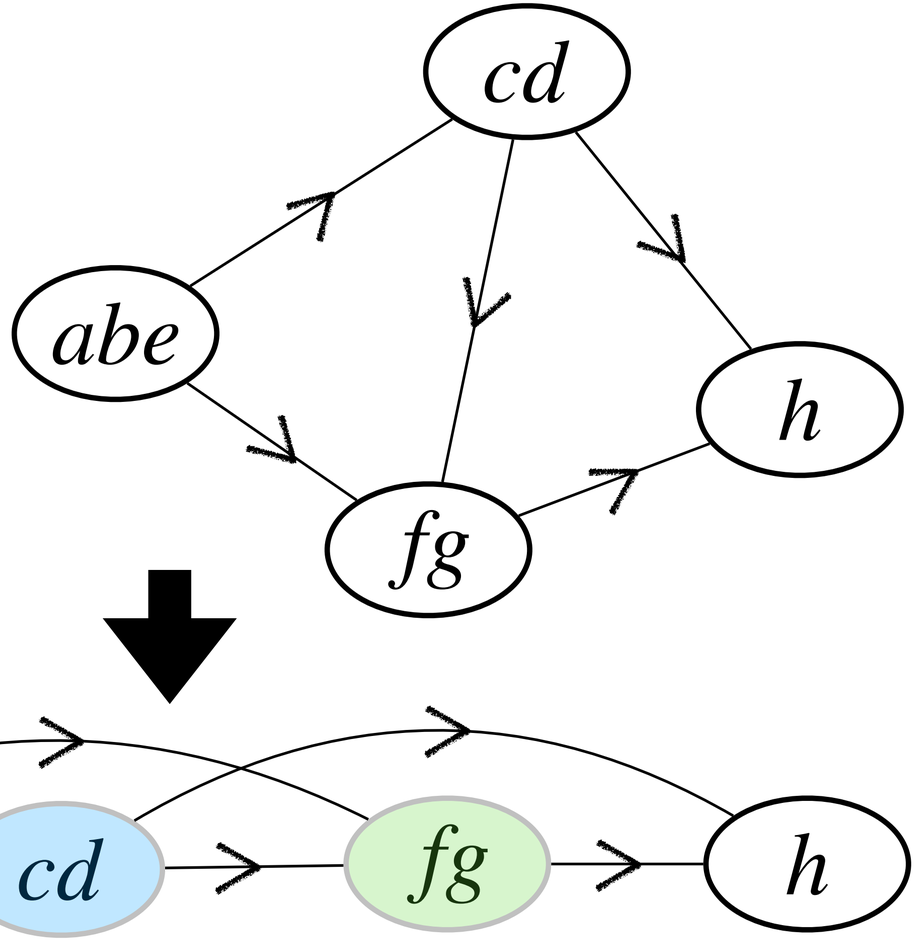
We will see that:

1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

$G = (V, E)$



Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$

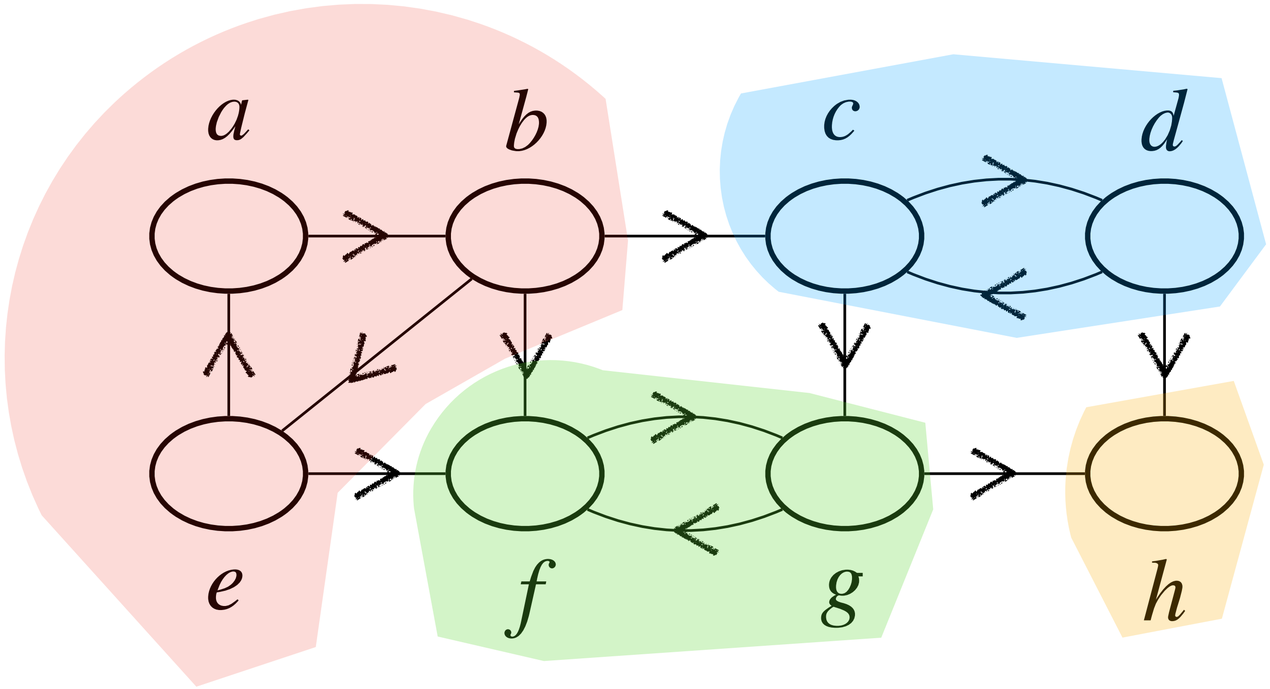


Let's recall how the algorithm worked.

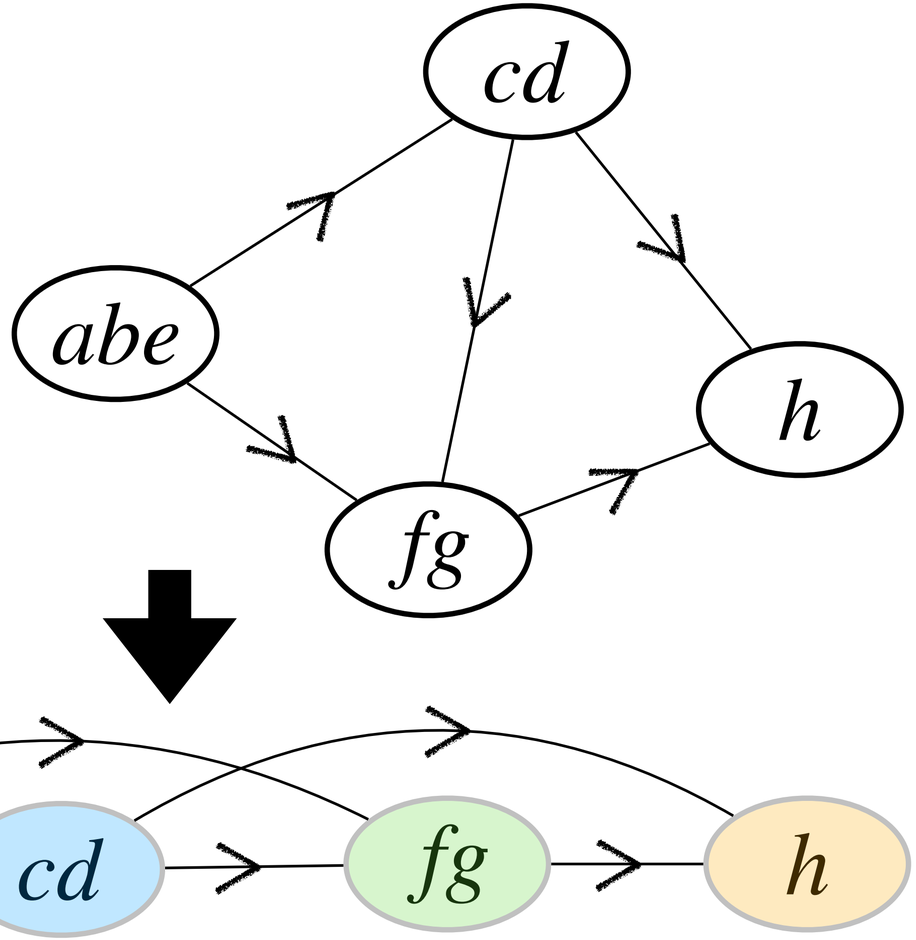
We will see that:

1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

$G = (V, E)$



Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$

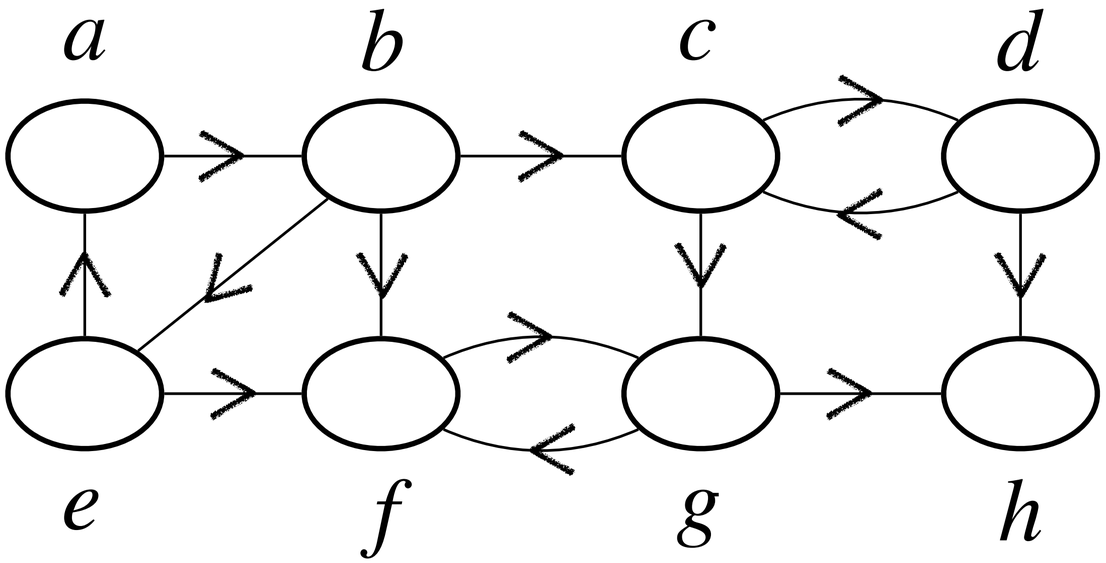


Let's recall how the algorithm worked.

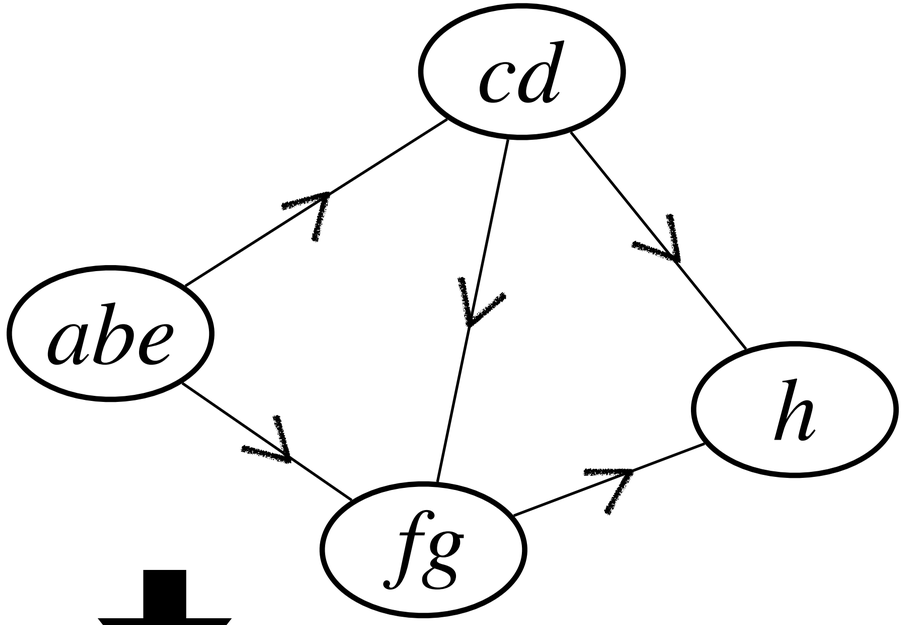
We will see that:

1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

$G = (V, E)$

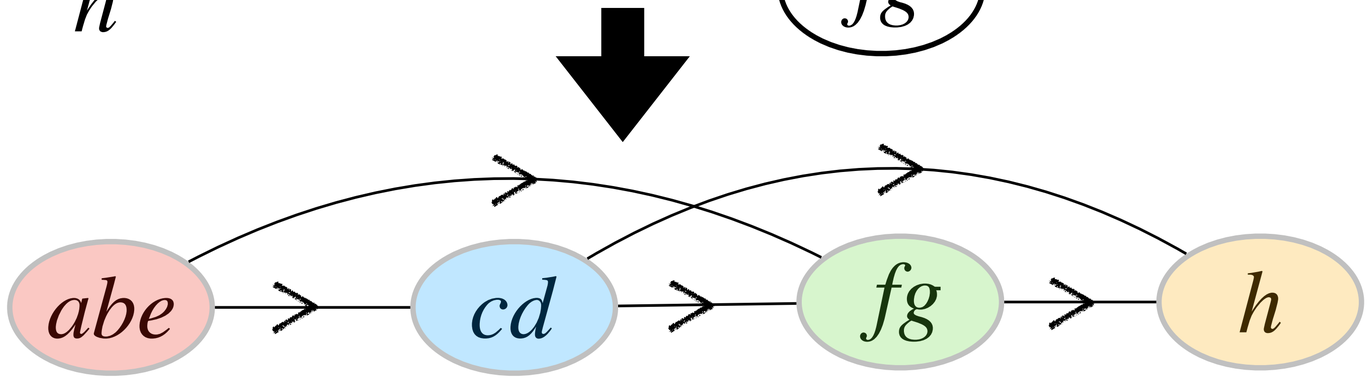


Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$



How can the algorithm work without knowing the strongly connected components?

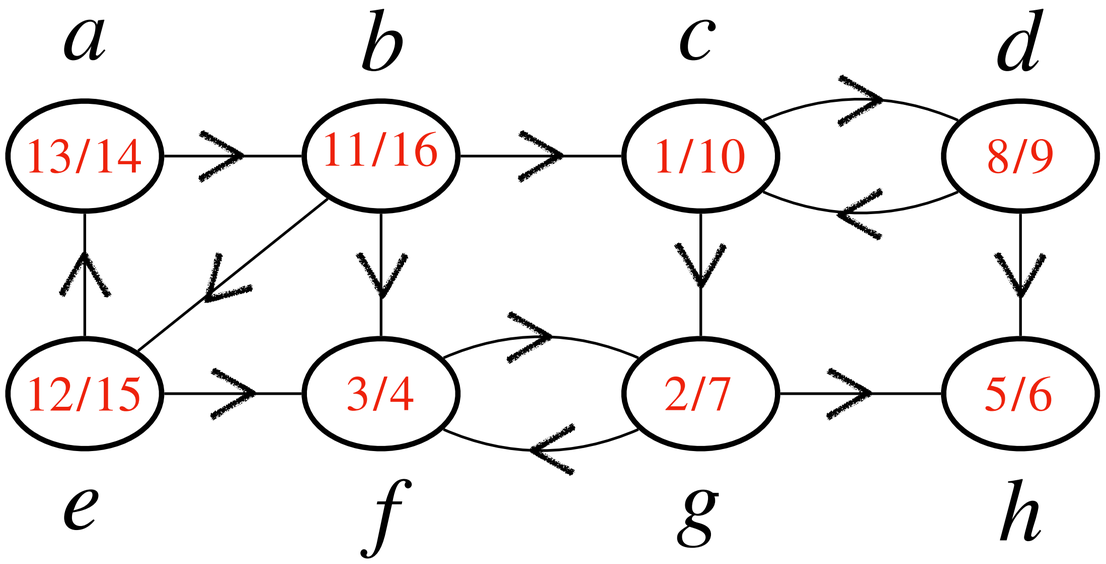
Answer: by using the finish time $u.f$



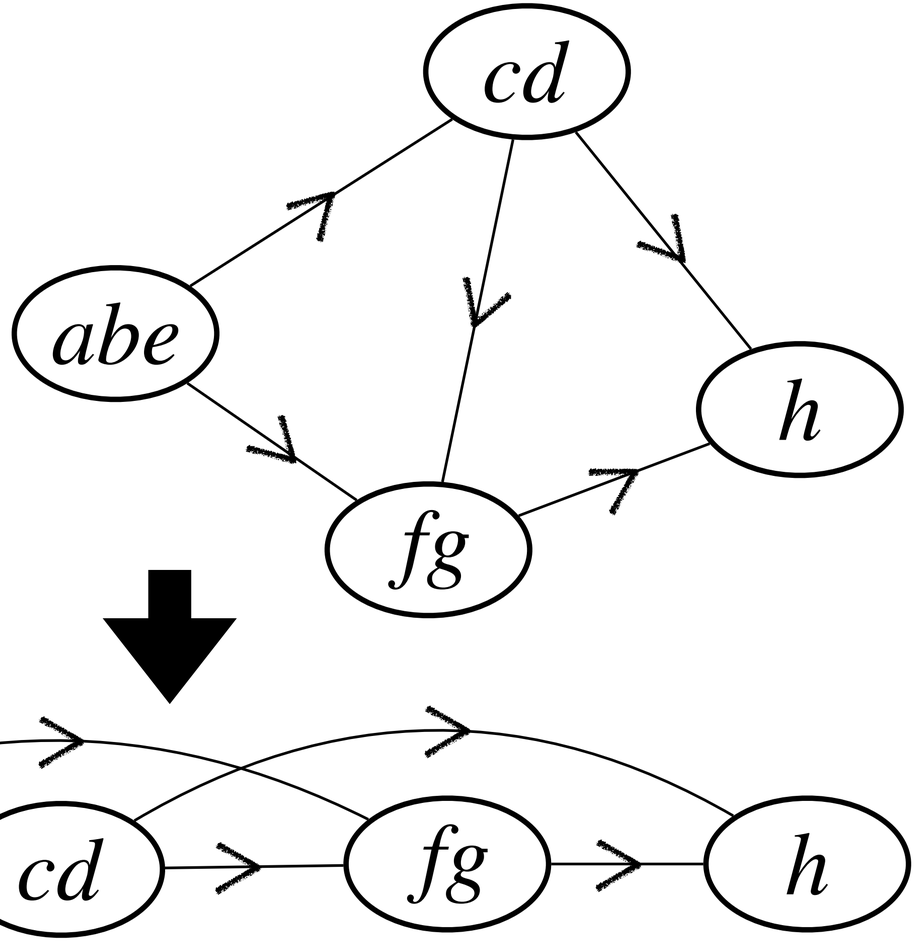
We will see that:

1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

$G = (V, E)$



Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$



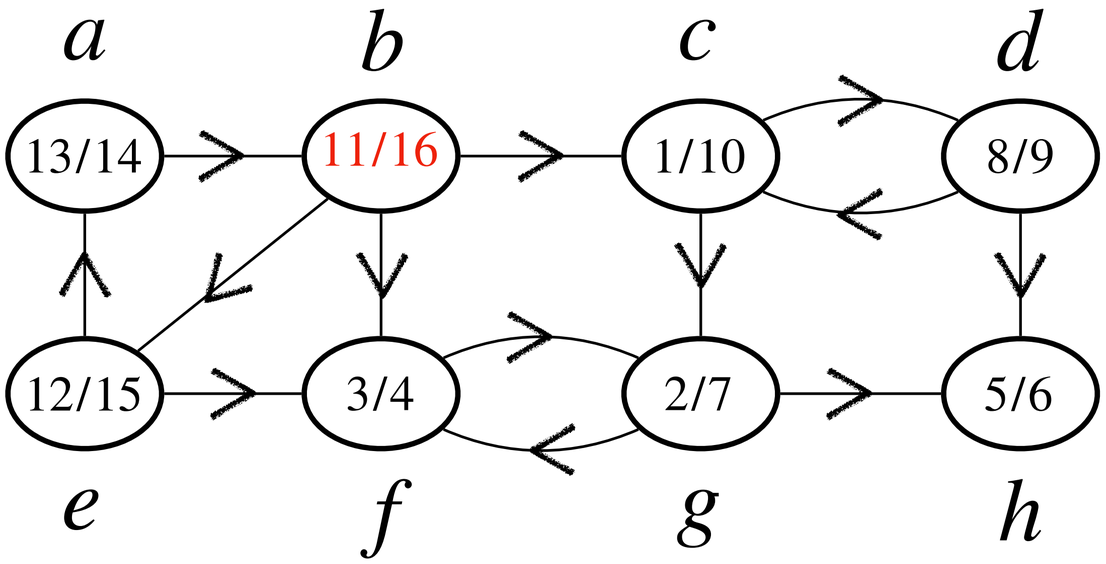
How can the algorithm work without knowing the strongly connected components?

Answer: by using the finish time $u.f$

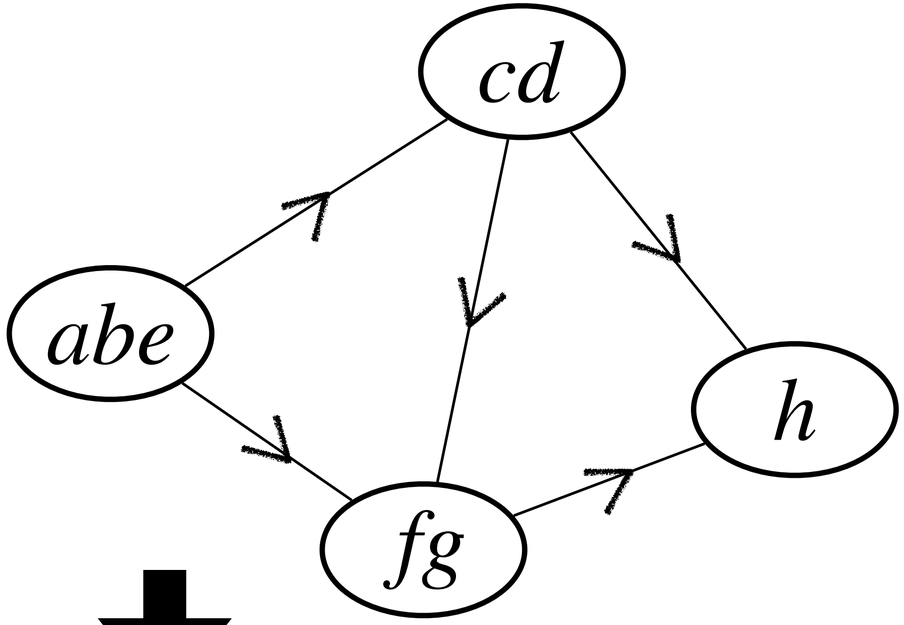
We will see that:

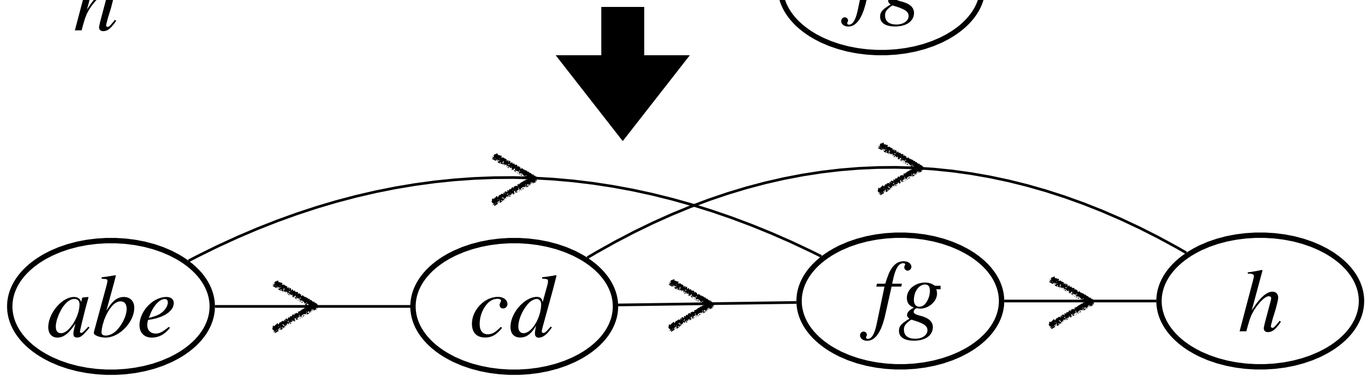
1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

$G = (V, E)$



Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$





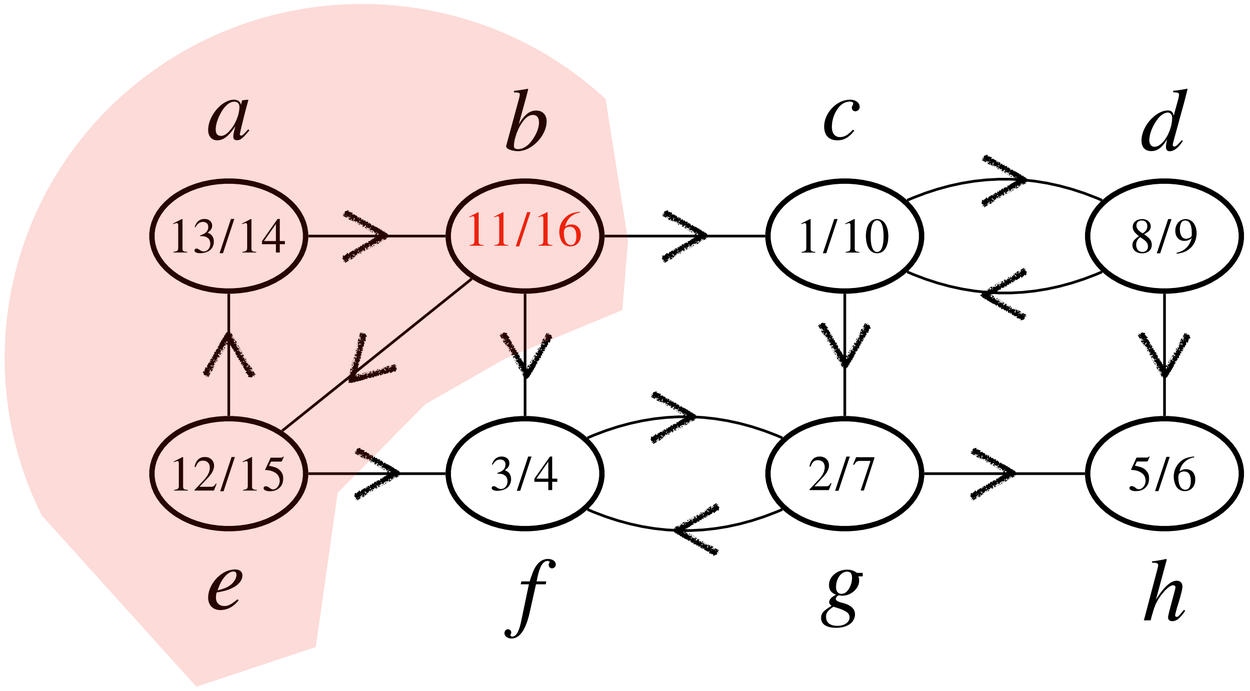
How can the algorithm work without knowing the strongly connected components?

Answer: by using the finish time $u.f$

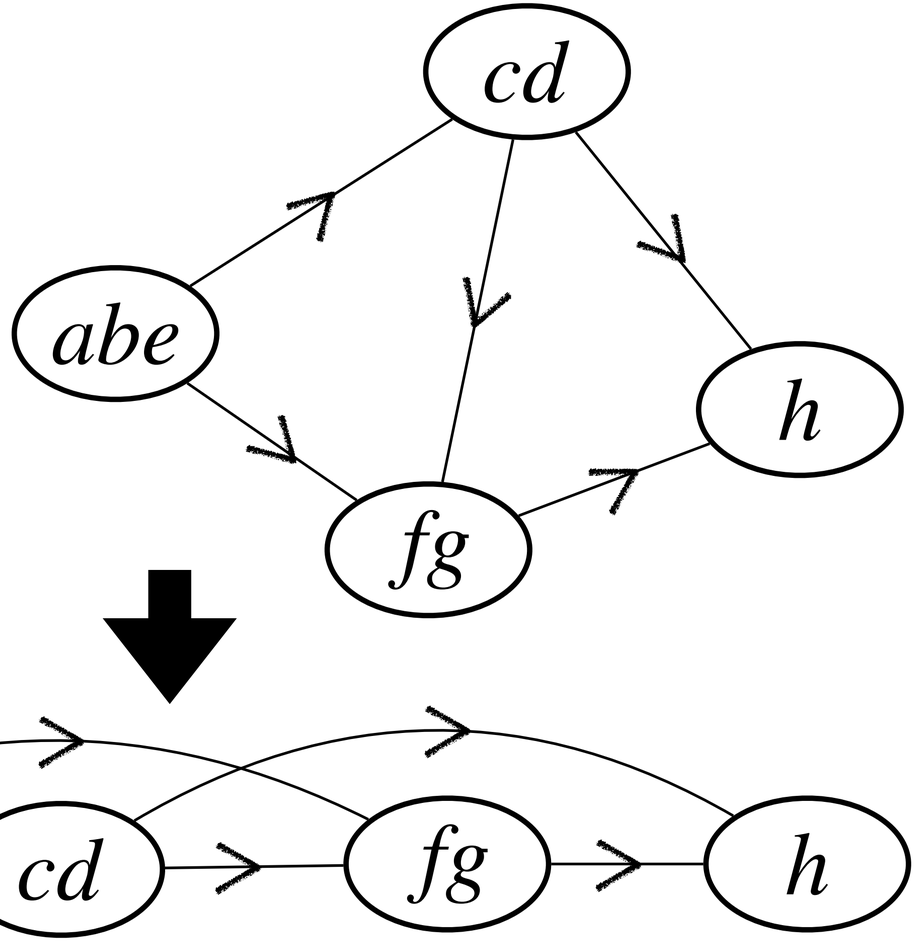
We will see that:

1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

$G = (V, E)$



Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$



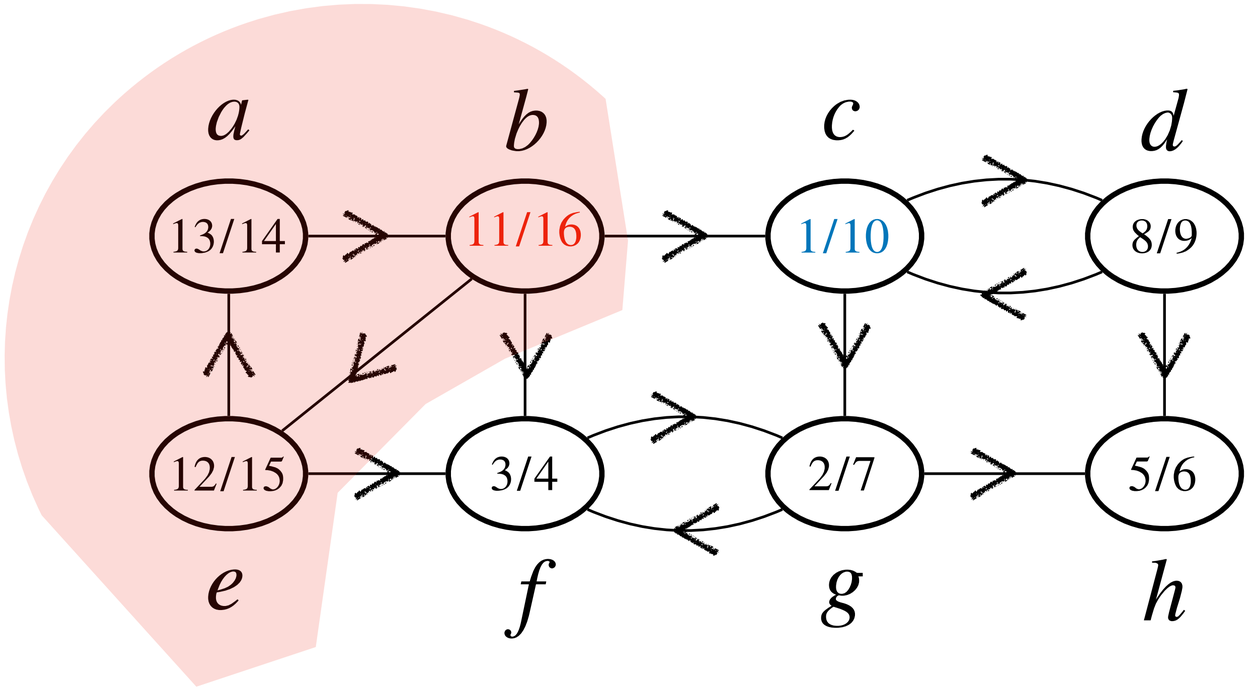
How can the algorithm work without knowing the strongly connected components?

Answer: by using the finish time $u.f$

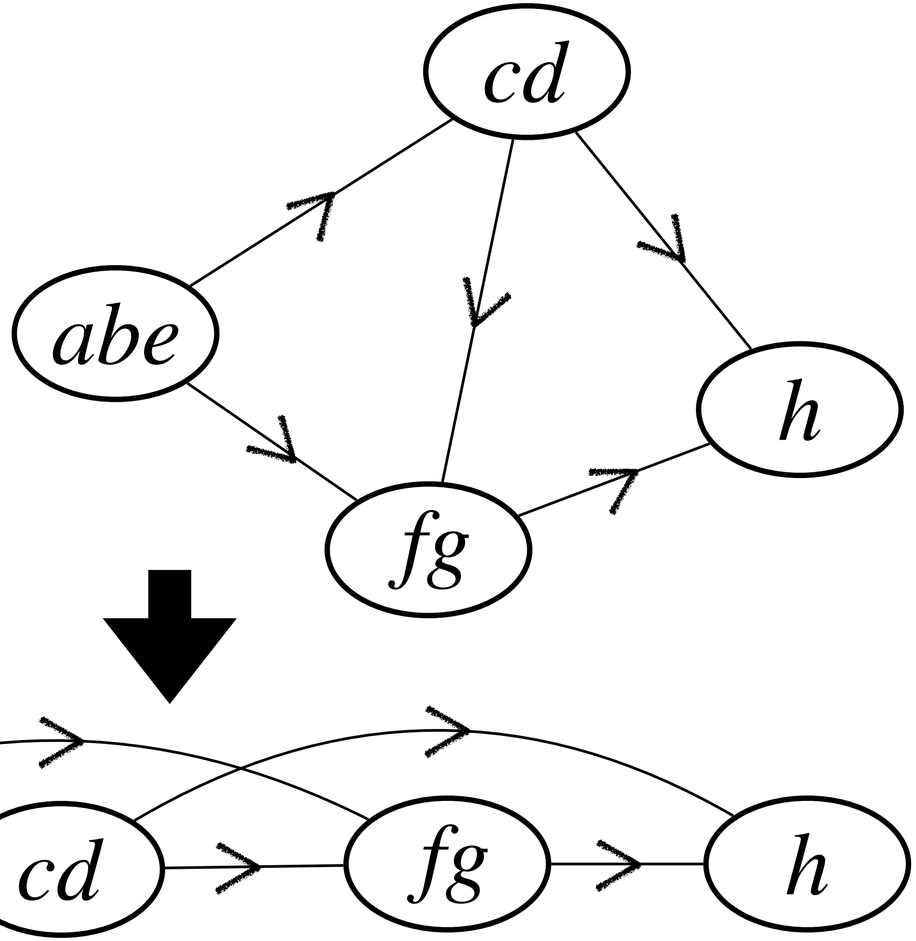
We will see that:

1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

$G = (V, E)$



Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$



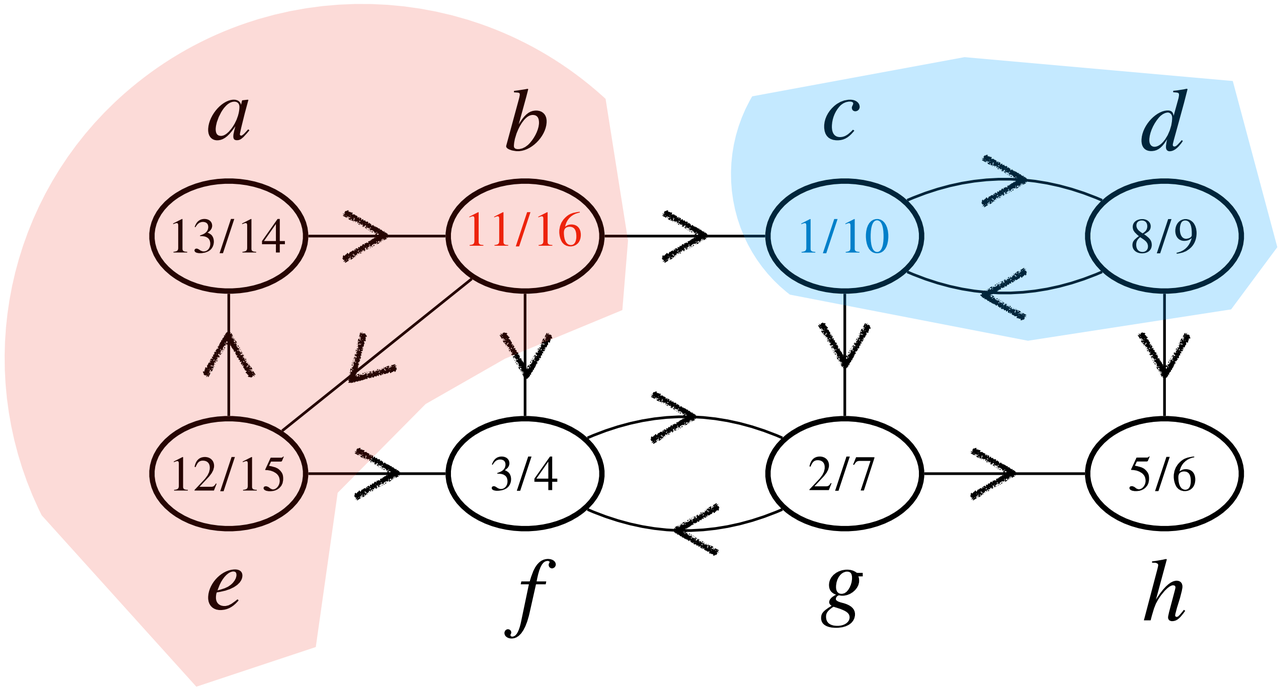
How can the algorithm work without knowing the strongly connected components?

Answer: by using the finish time $u.f$

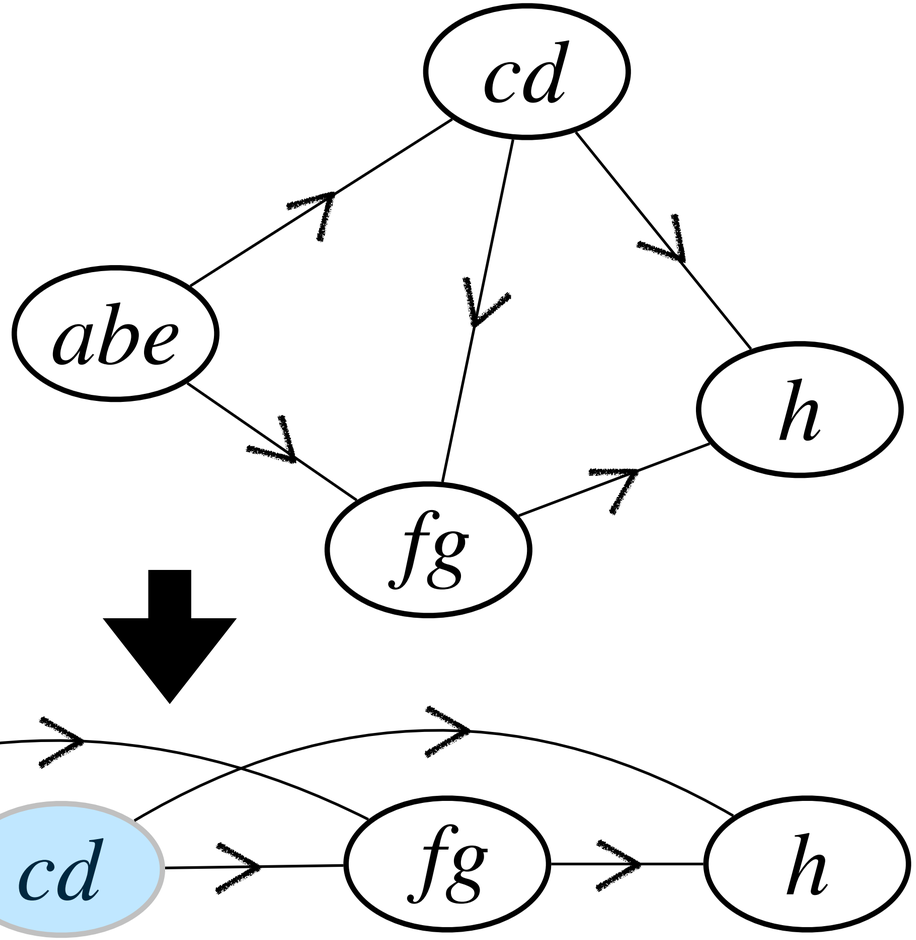
We will see that:

1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

$G = (V, E)$



Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$



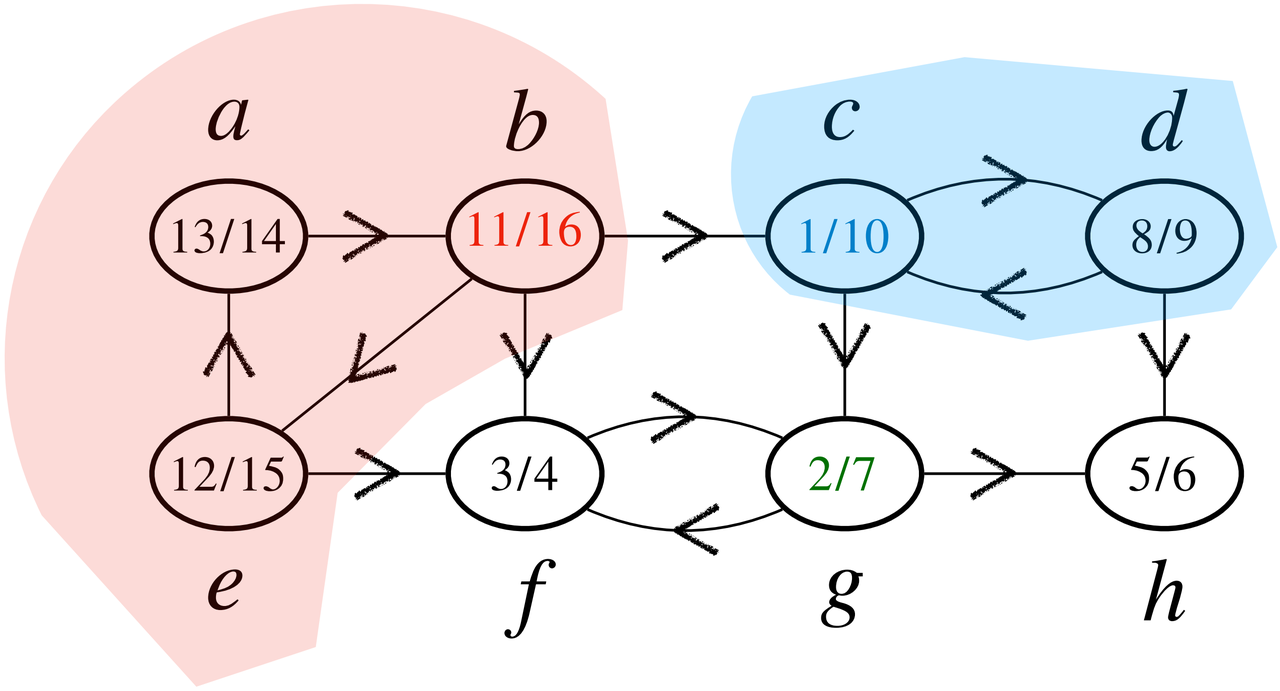
How can the algorithm work without knowing the strongly connected components?

Answer: by using the finish time $u.f$

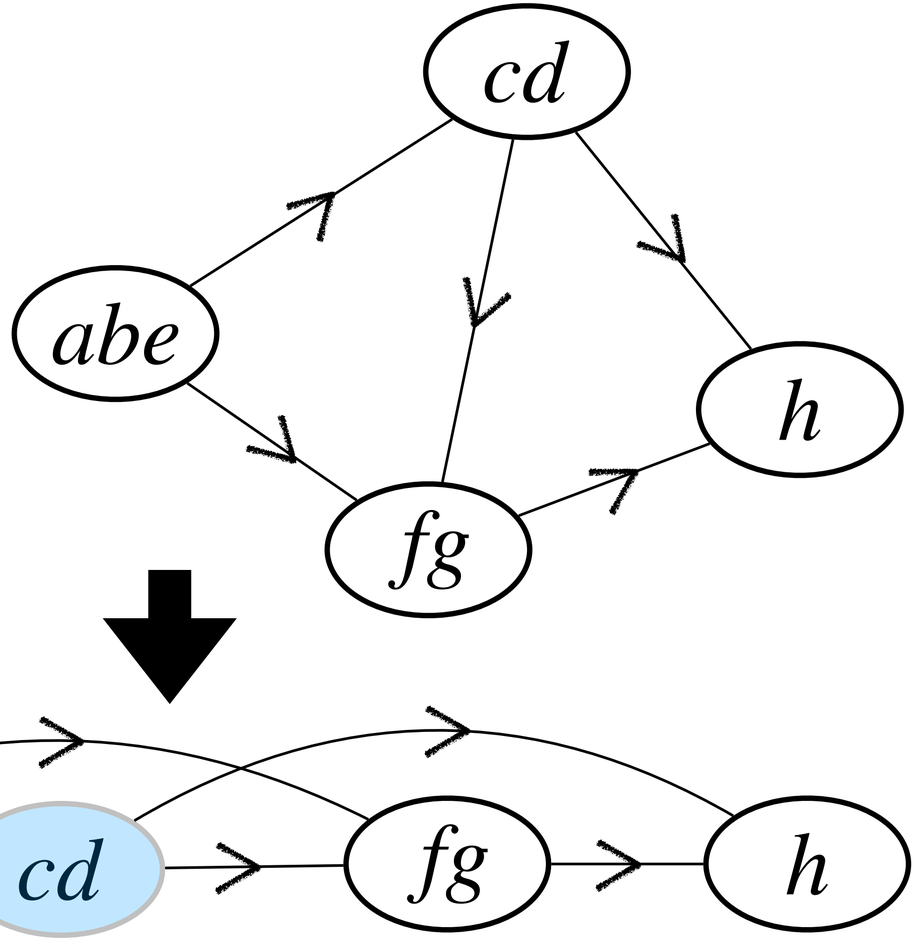
We will see that:

1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

$G = (V, E)$



Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$



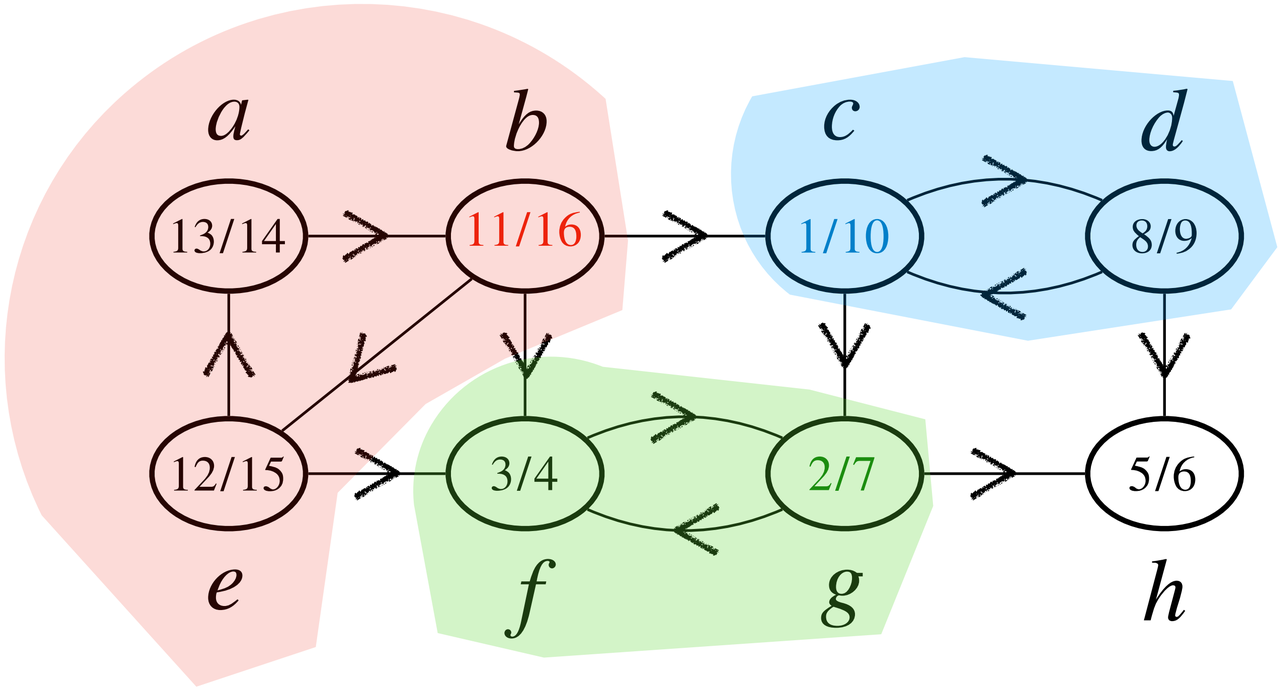
How can the algorithm work without knowing the strongly connected components?

Answer: by using the finish time $u.f$

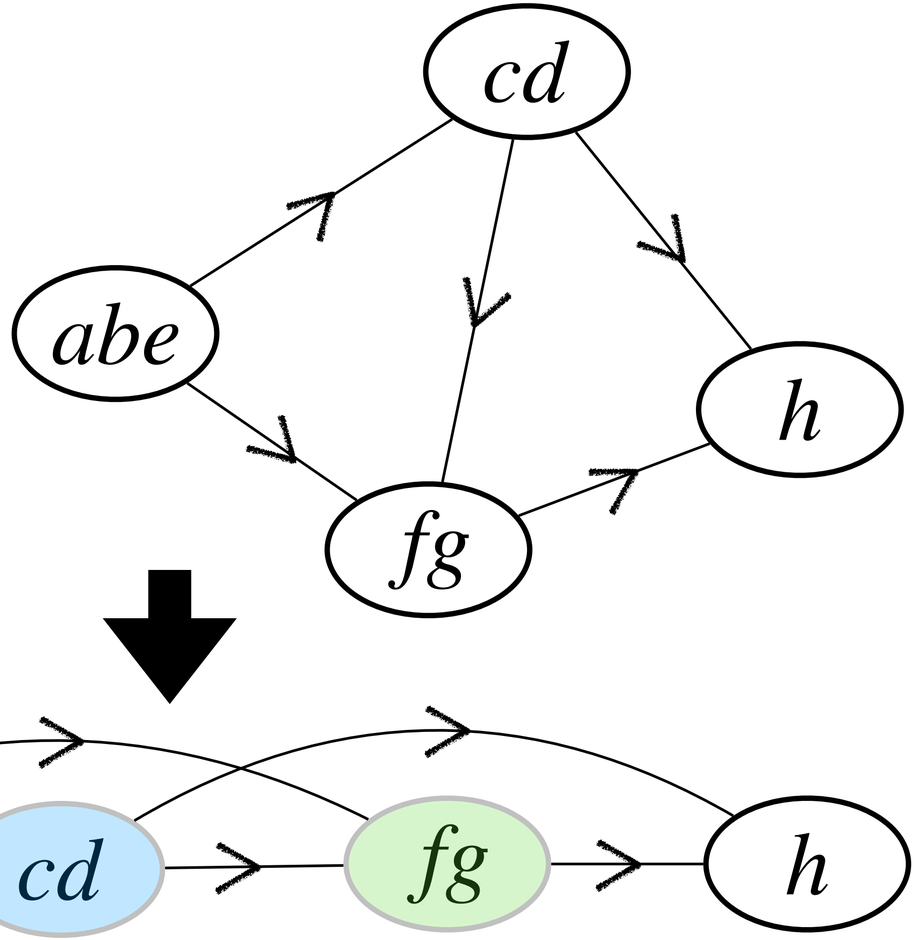
We will see that:

1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

$G = (V, E)$



Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$



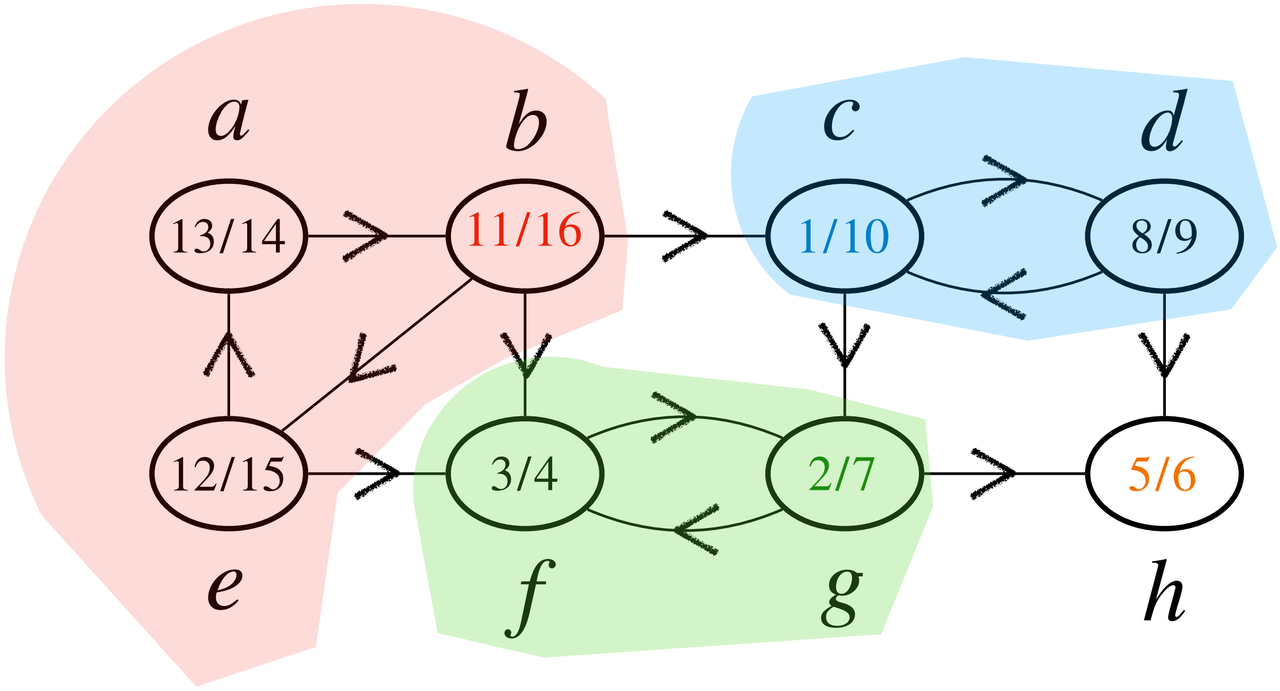
How can the algorithm work without knowing the strongly connected components?

Answer: by using the finish time $u.f$

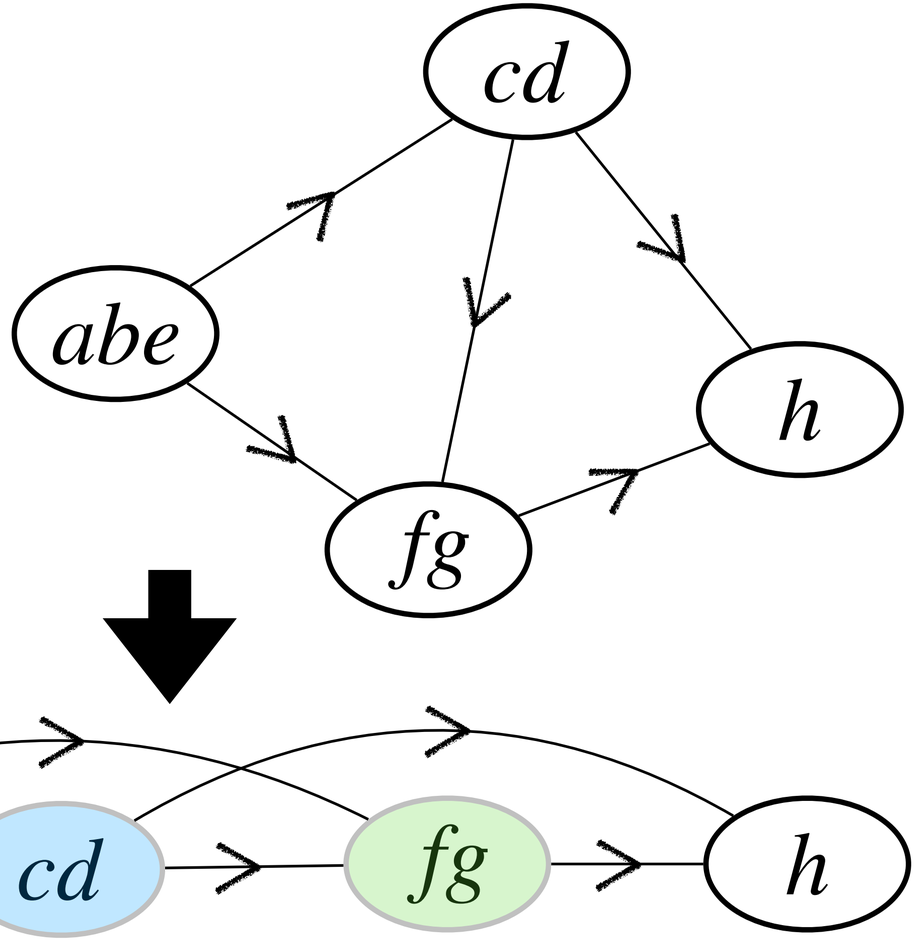
We will see that:

1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

$G = (V, E)$



Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$



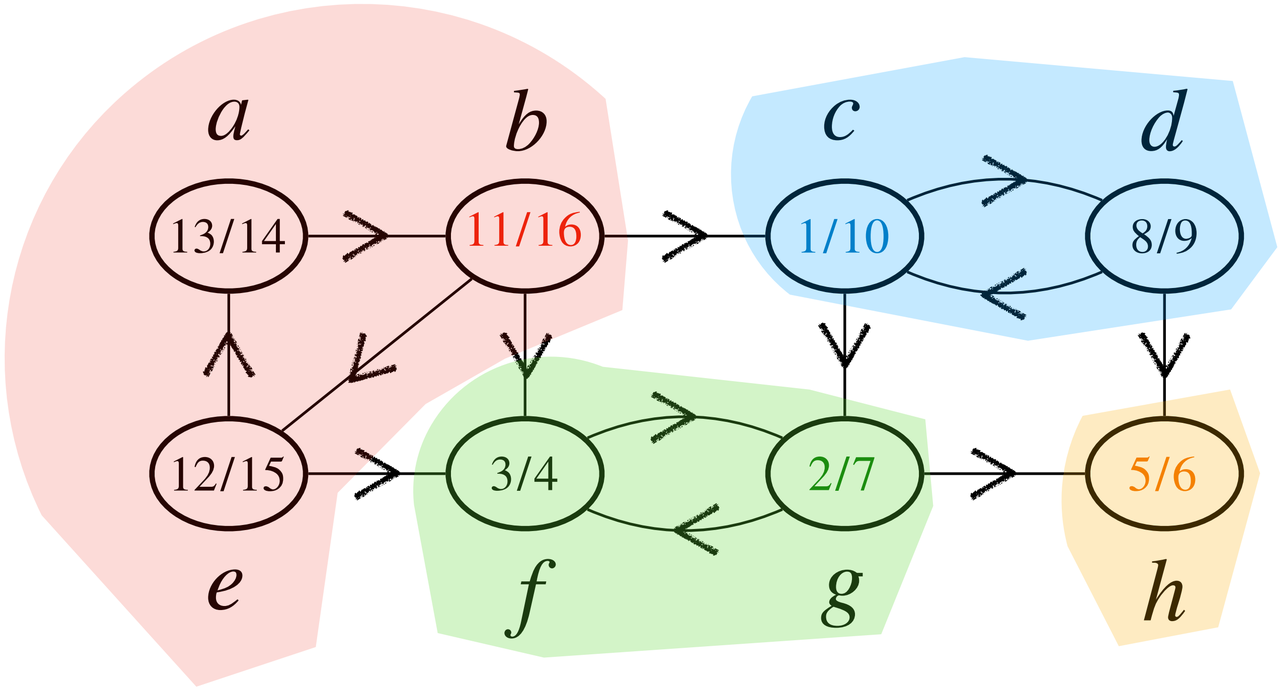
How can the algorithm work without knowing the strongly connected components?

Answer: by using the finish time $u.f$

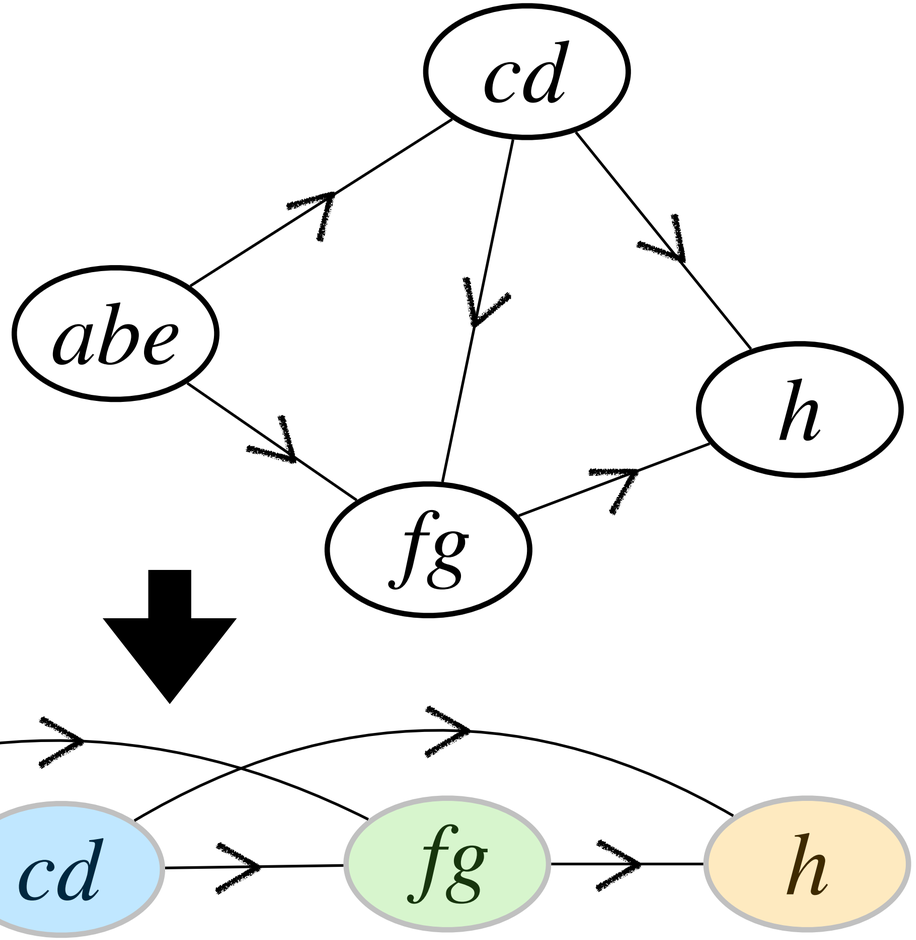
We will see that:

1. The component graph is acyclic.
2. The algorithm visits the vertices of the component graph in topologically sorted order.

$G = (V, E)$



Component Graph $G^{SCC} = (V^{SCC}, E^{SCC})$

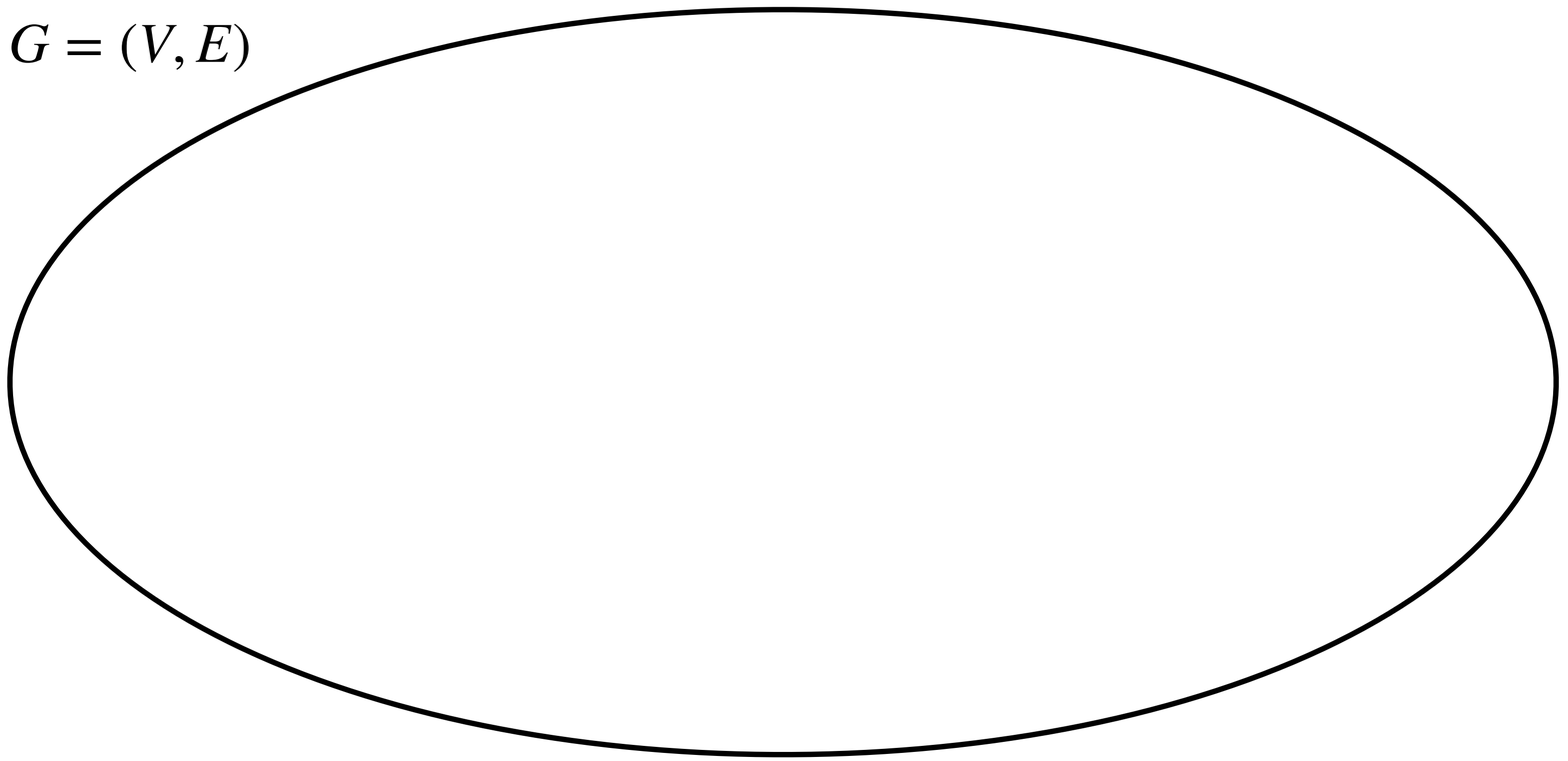


How can the algorithm work without knowing the strongly connected components?

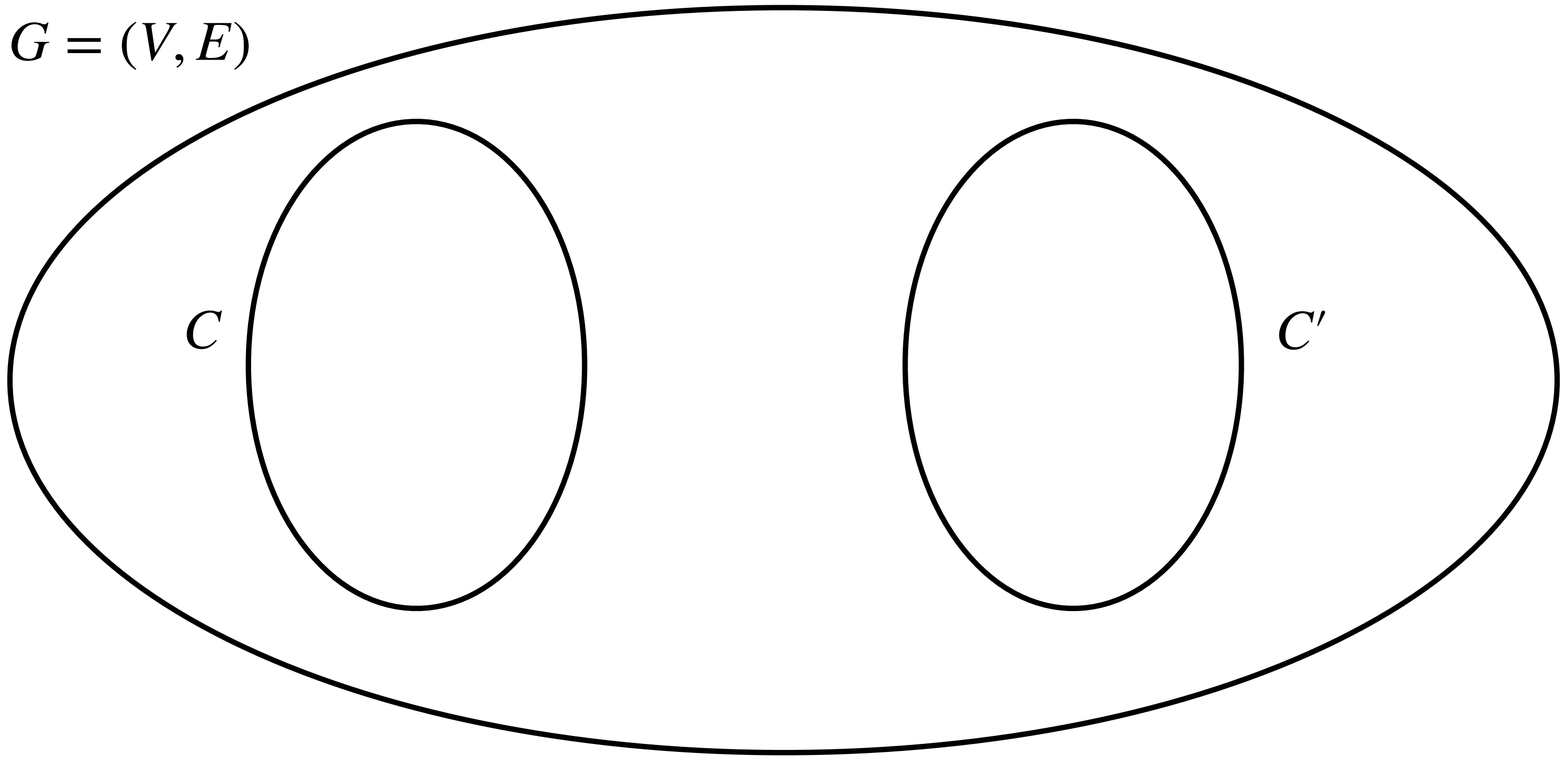
Answer: by using the finish time $u.f$

Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$, let $u, v \in C$, let $u', v' \in C'$, and suppose that G contains a path $u \rightsquigarrow u'$. Then G cannot also contain a path $v' \rightsquigarrow v$.

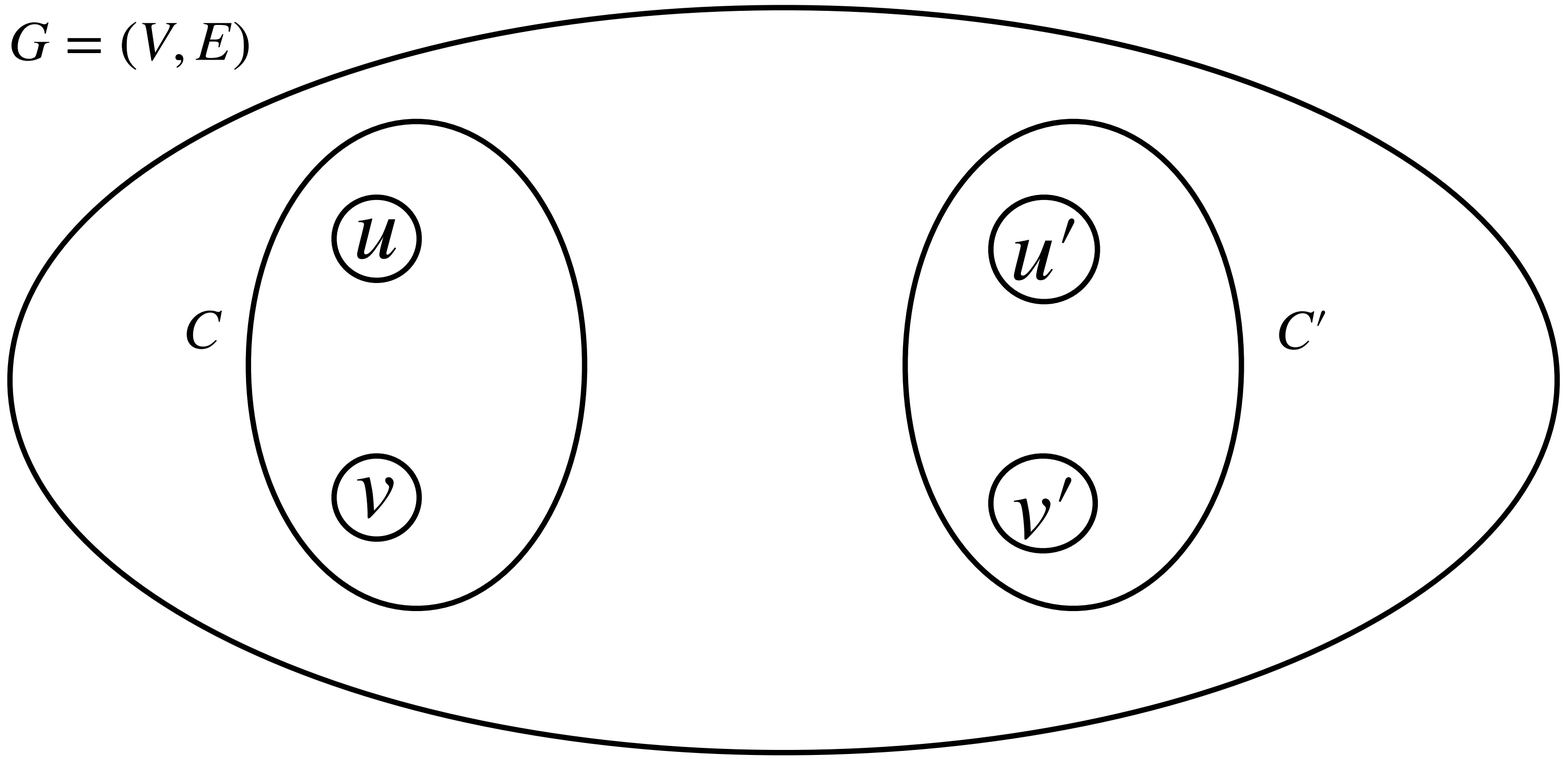
Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$, let $u, v \in C$, let $u', v' \in C'$, and suppose that G contains a path $u \rightsquigarrow u'$. Then G cannot also contain a path $v' \rightsquigarrow v$.



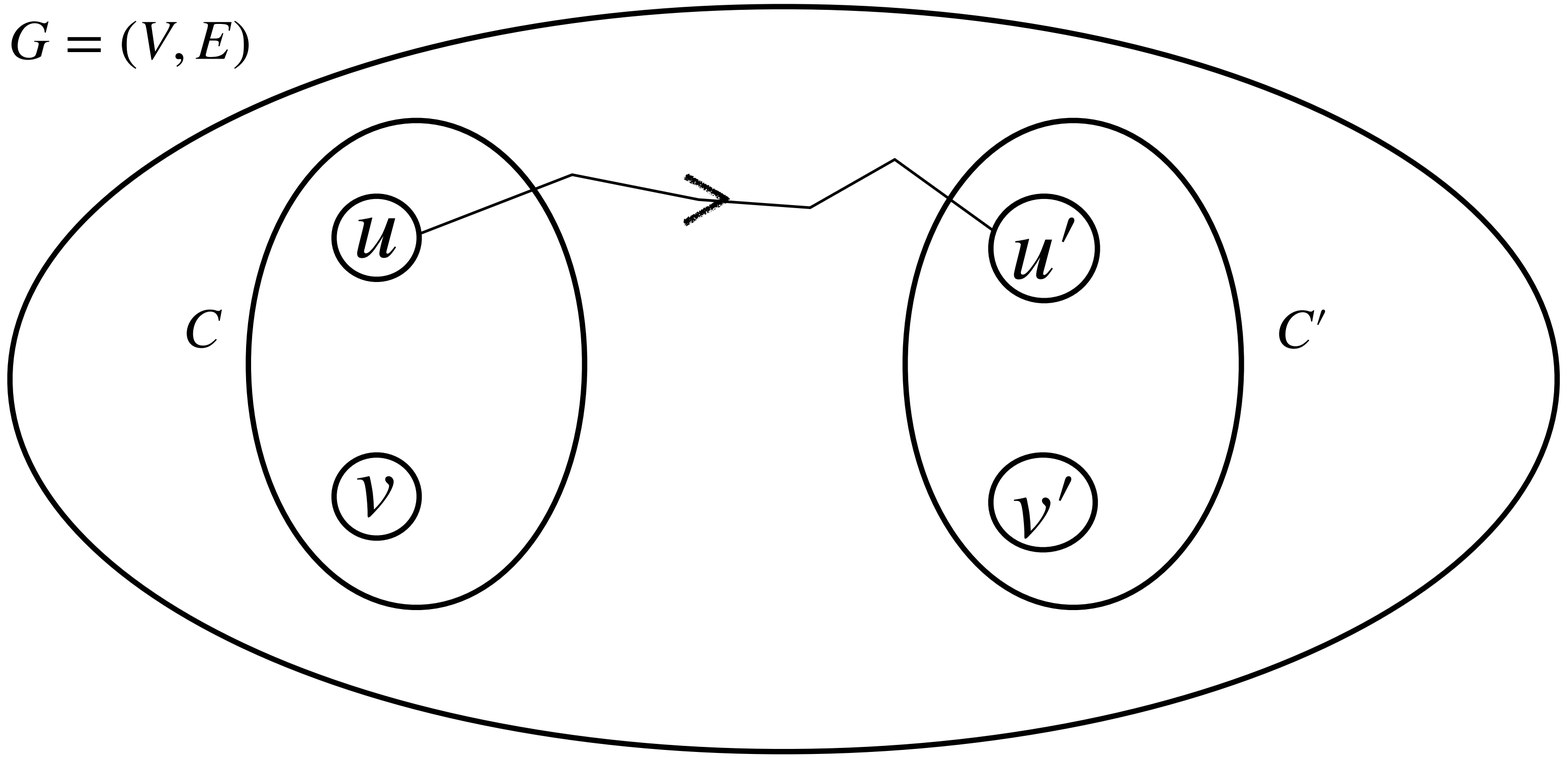
Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$, let $u, v \in C$, let $u', v' \in C'$, and suppose that G contains a path $u \rightsquigarrow u'$. Then G cannot also contain a path $v' \rightsquigarrow v$.



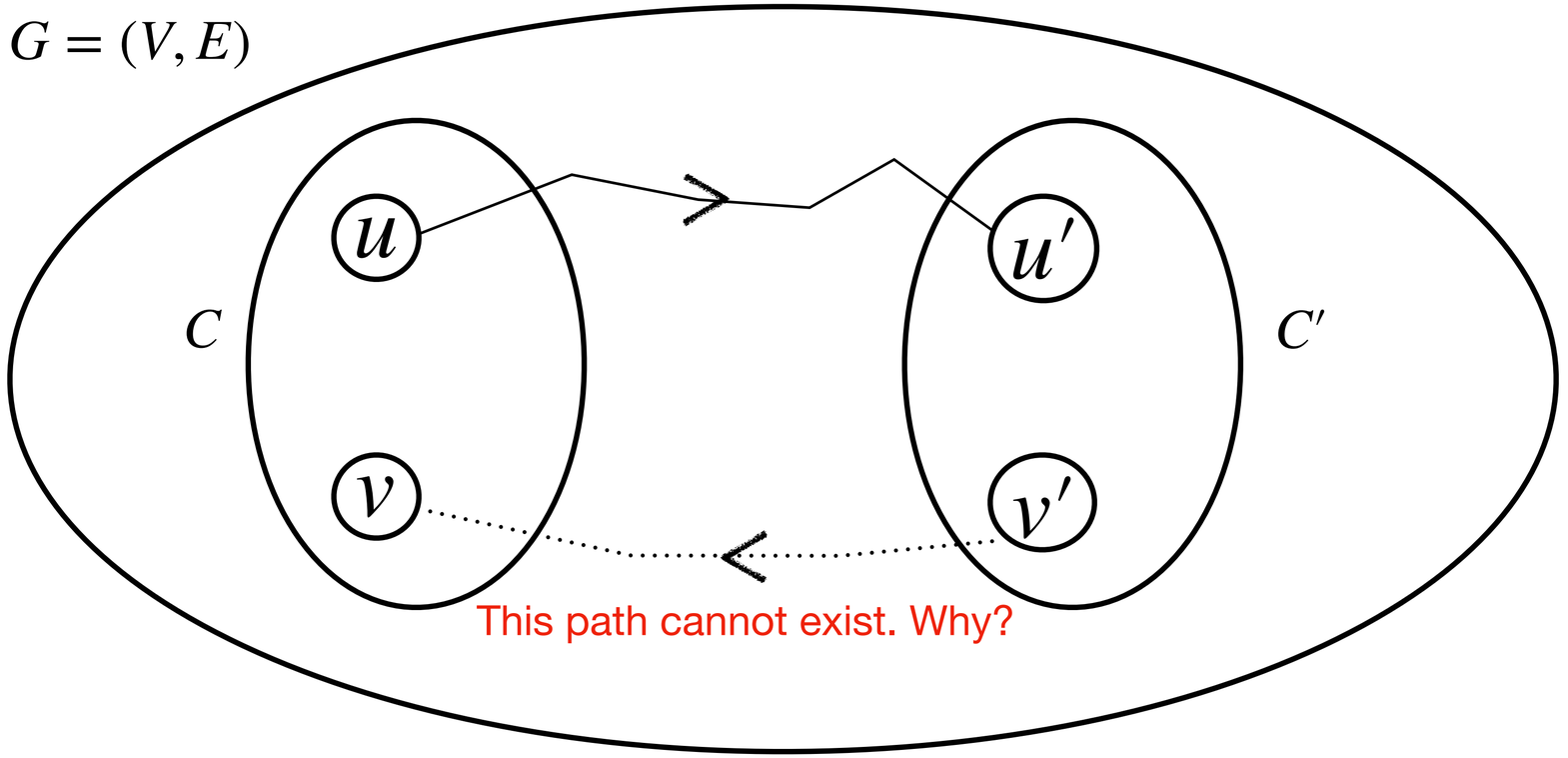
Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$, let $u, v \in C$, let $u', v' \in C'$, and suppose that G contains a path $u \rightsquigarrow u'$. Then G cannot also contain a path $v' \rightsquigarrow v$.



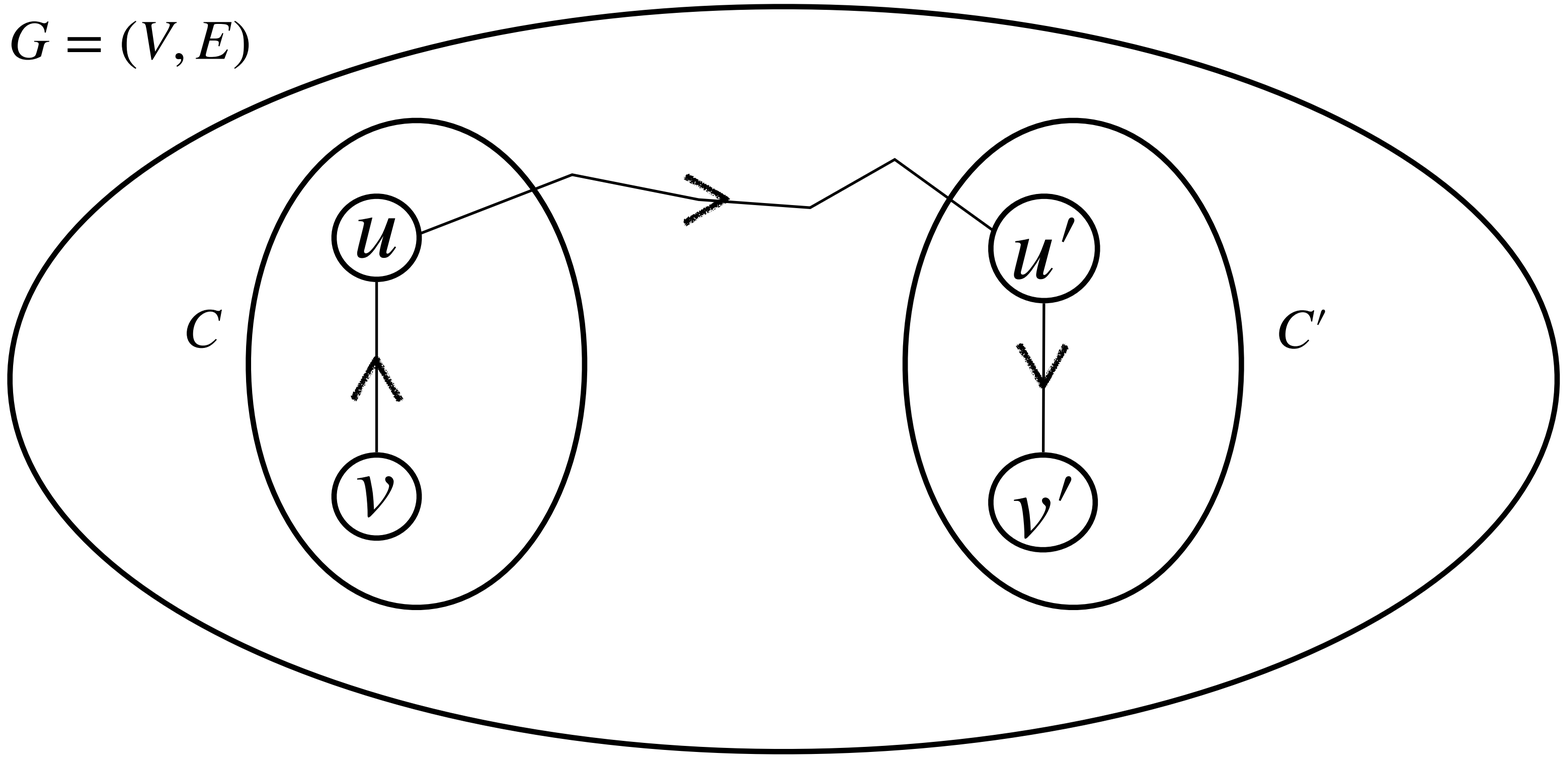
Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$, let $u, v \in C$, let $u', v' \in C'$, and suppose that G contains a path $u \rightsquigarrow u'$. Then G cannot also contain a path $v' \rightsquigarrow v$.



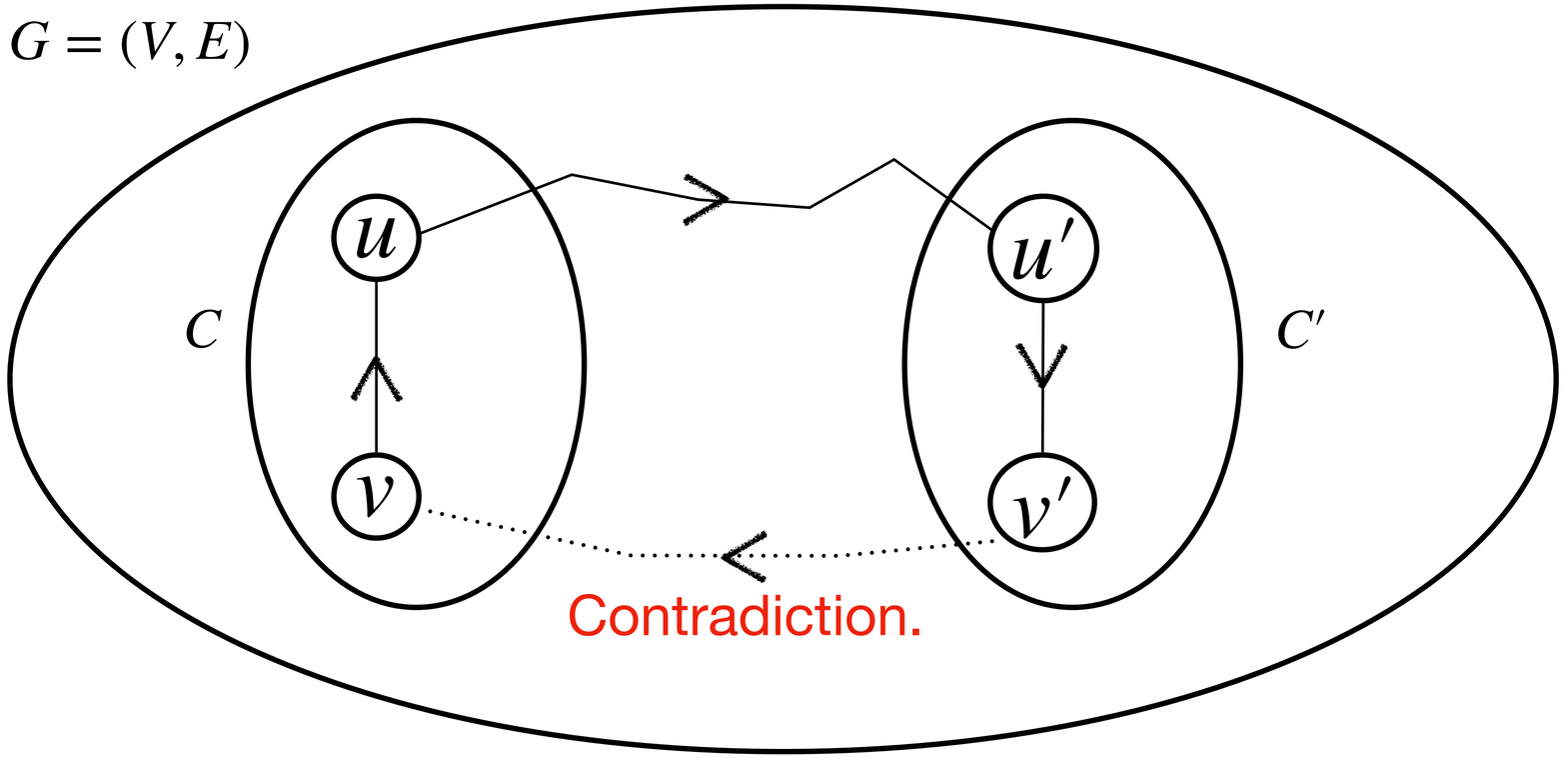
Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$, let $u, v \in C$, let $u', v' \in C'$, and suppose that G contains a path $u \rightsquigarrow u'$. Then G cannot also contain a path $v' \rightsquigarrow v$.



Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$, let $u, v \in C$, let $u', v' \in C'$, and suppose that G contains a path $u \rightsquigarrow u'$. Then G cannot also contain a path $v' \rightsquigarrow v$.



Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$, let $u, v \in C$, let $u', v' \in C'$, and suppose that G contains a path $u \rightsquigarrow u'$. Then G cannot also contain a path $v' \rightsquigarrow v$.



Define “discovery time” and “finish time” for a set of vertices:

For a set U of vertices, $d(U)$ and $f(U)$ are the earliest discovery time and latest finish time, respectively, of any vertex in U. That is:

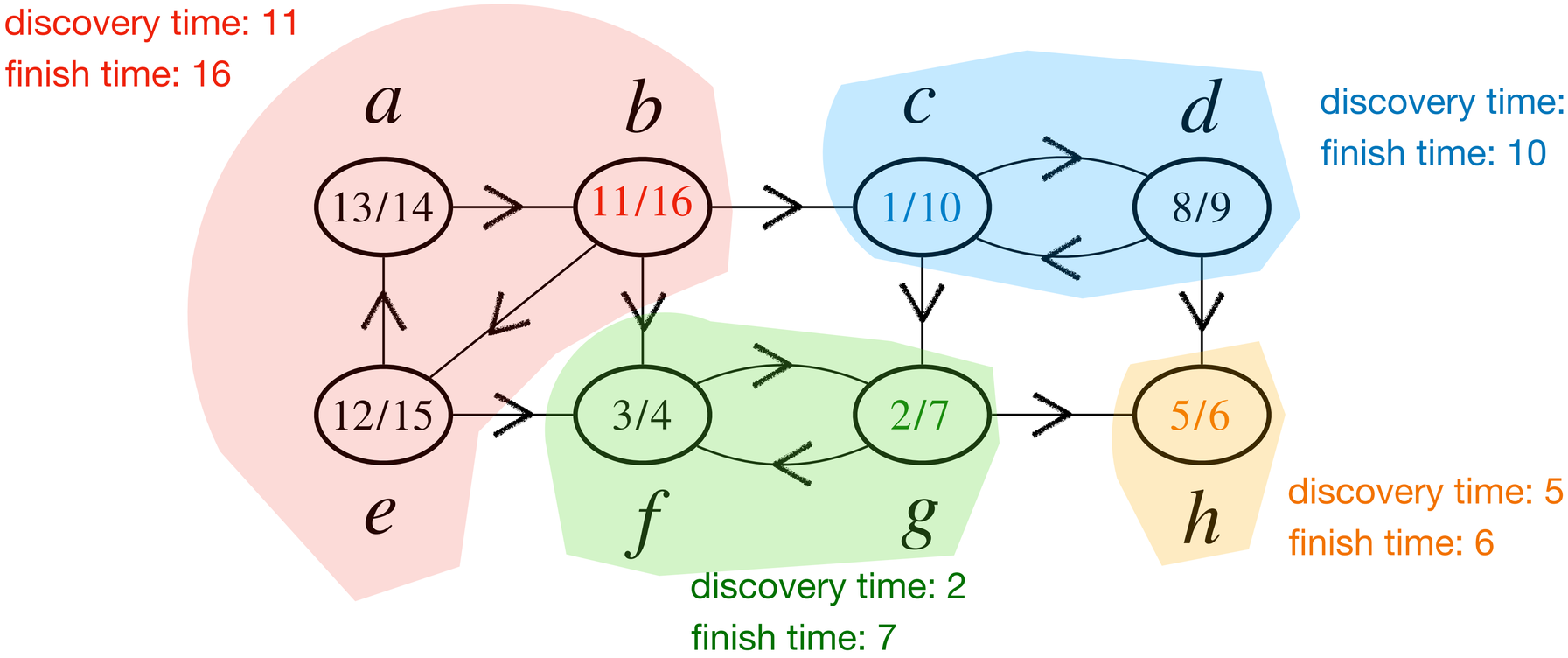
$$d(U) = \min\{u.d \ : \ u \in U\} \qquad f(U) = \max\{u.f \ : \ u \in U\}$$

Define “discovery time” and “finish time” for a set of vertices:

For a set U of vertices, $d(U)$ and $f(U)$ are the earliest discovery time and latest finish time, respectively, of any vertex in U. That is:

$$d(U) = \min\{u.d \ : \ u \in U\} \qquad f(U) = \max\{u.f \ : \ u \in U\}$$

$$G = (V, E)$$

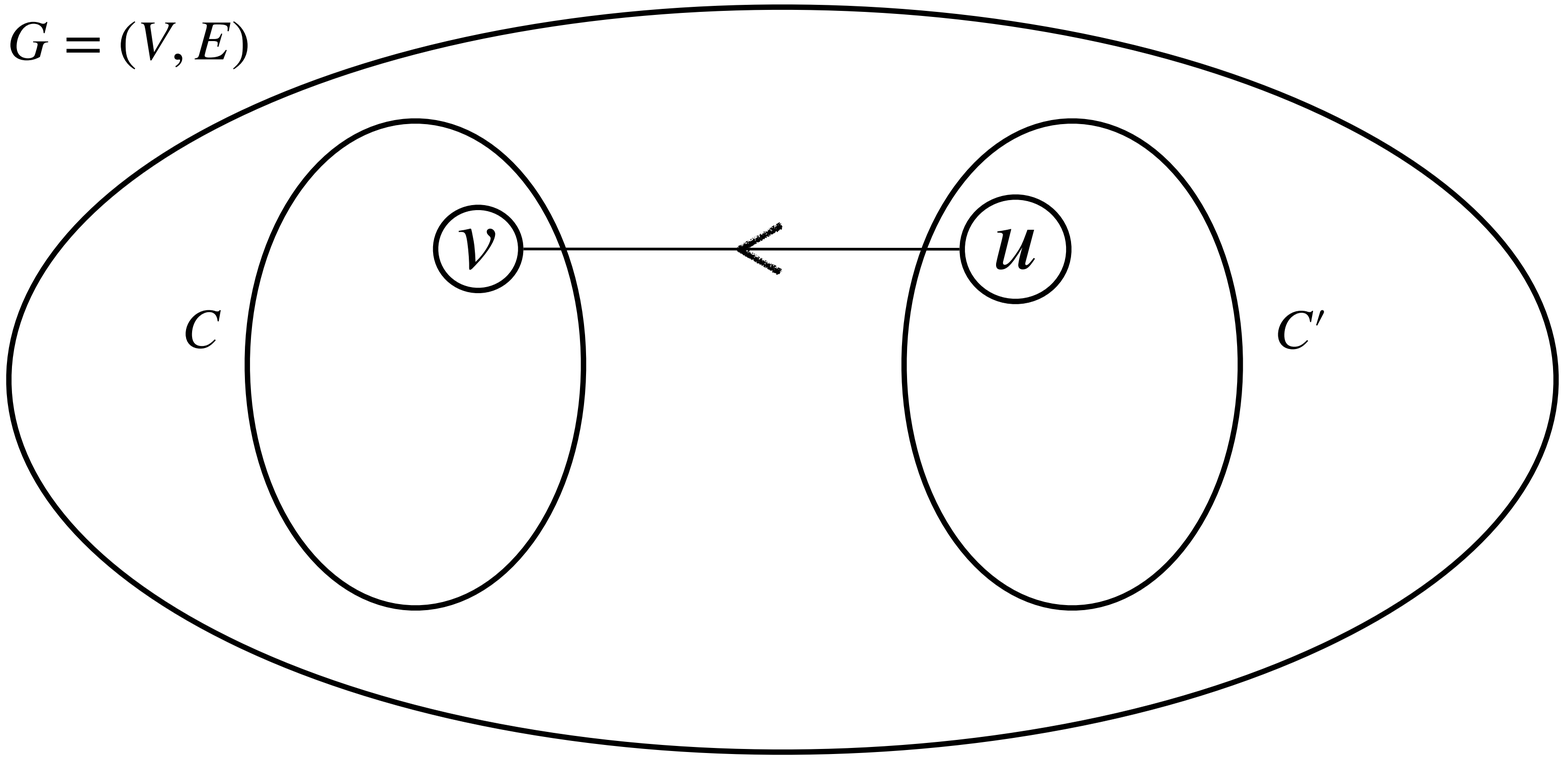


Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$.

Suppose that there is an edge $(u, v) \in E$, where $u \in C'$ and $v \in C$. Then $f(C') > f(C)$.

Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$.

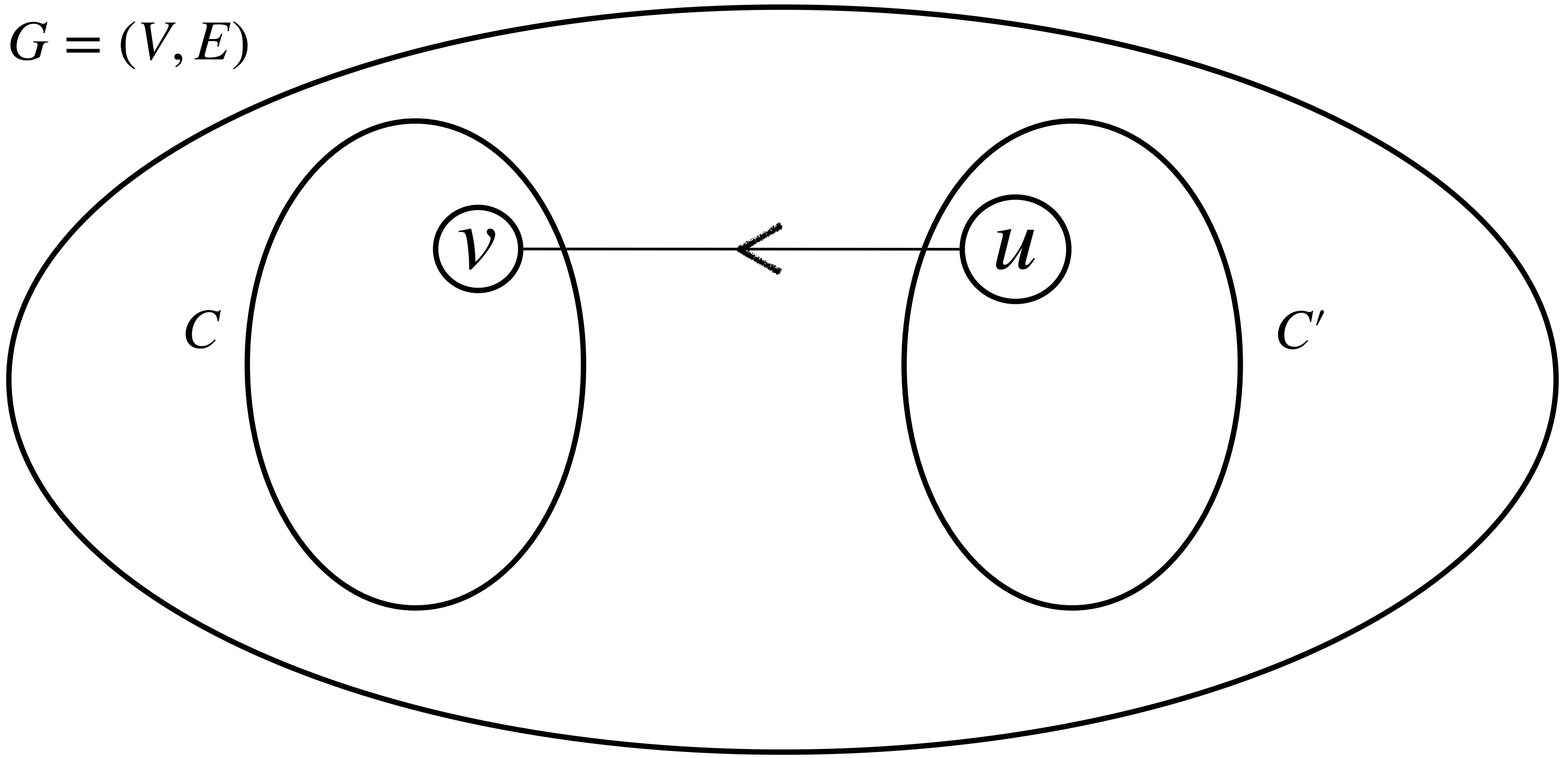
Suppose that there is an edge $(u, v) \in E$, where $u \in C'$ and $v \in C$. Then $f(C') > f(C)$.



Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$.

Suppose that there is an edge $(u, v) \in E$, where $u \in C'$ and $v \in C$. Then $f(C') > f(C)$.

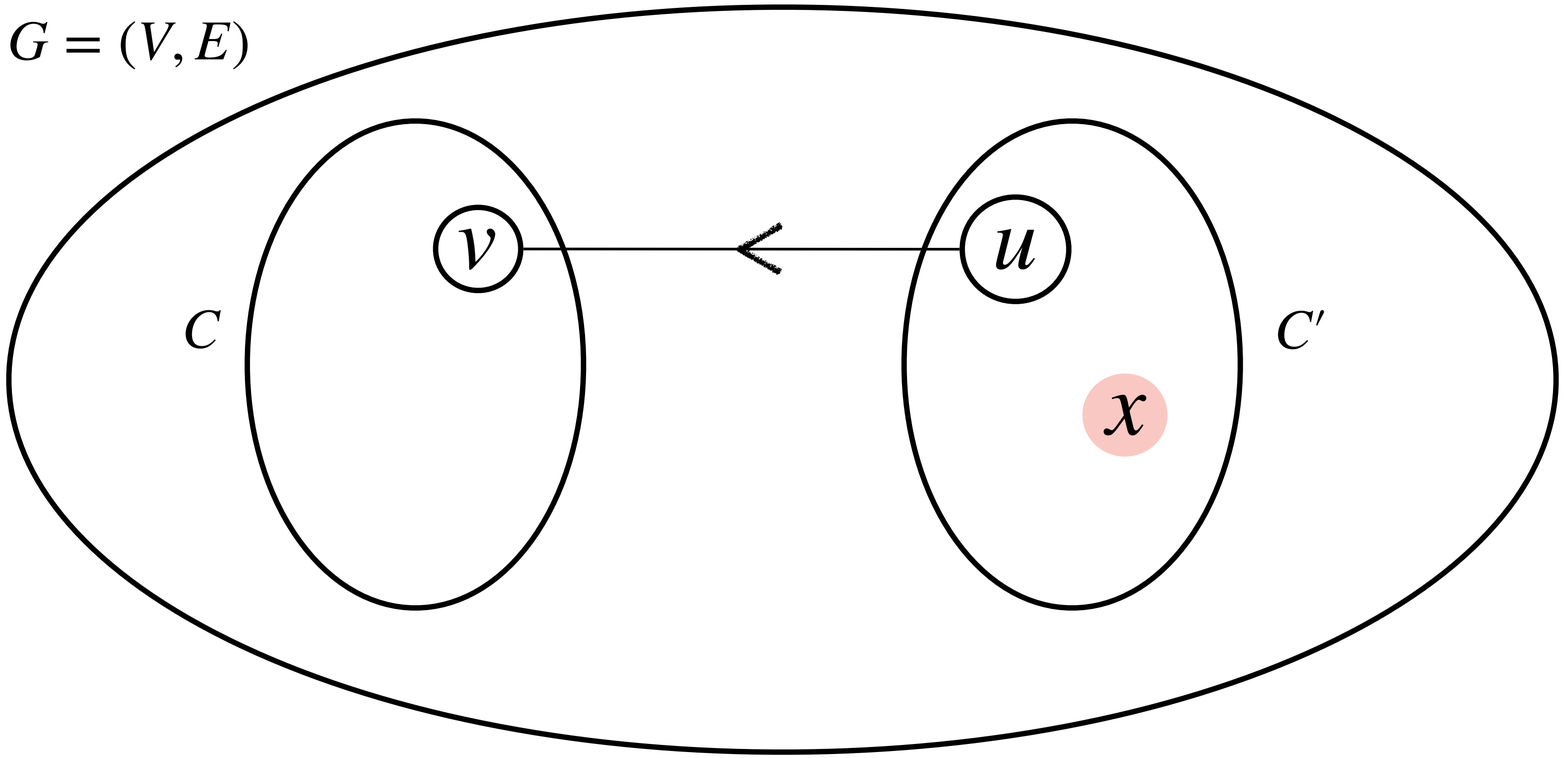
Case 1: $d(C') < d(C)$



Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$.

Suppose that there is an edge $(u, v) \in E$, where $u \in C'$ and $v \in C$. Then $f(C') > f(C)$.

Case 1: $d(C') < d(C)$

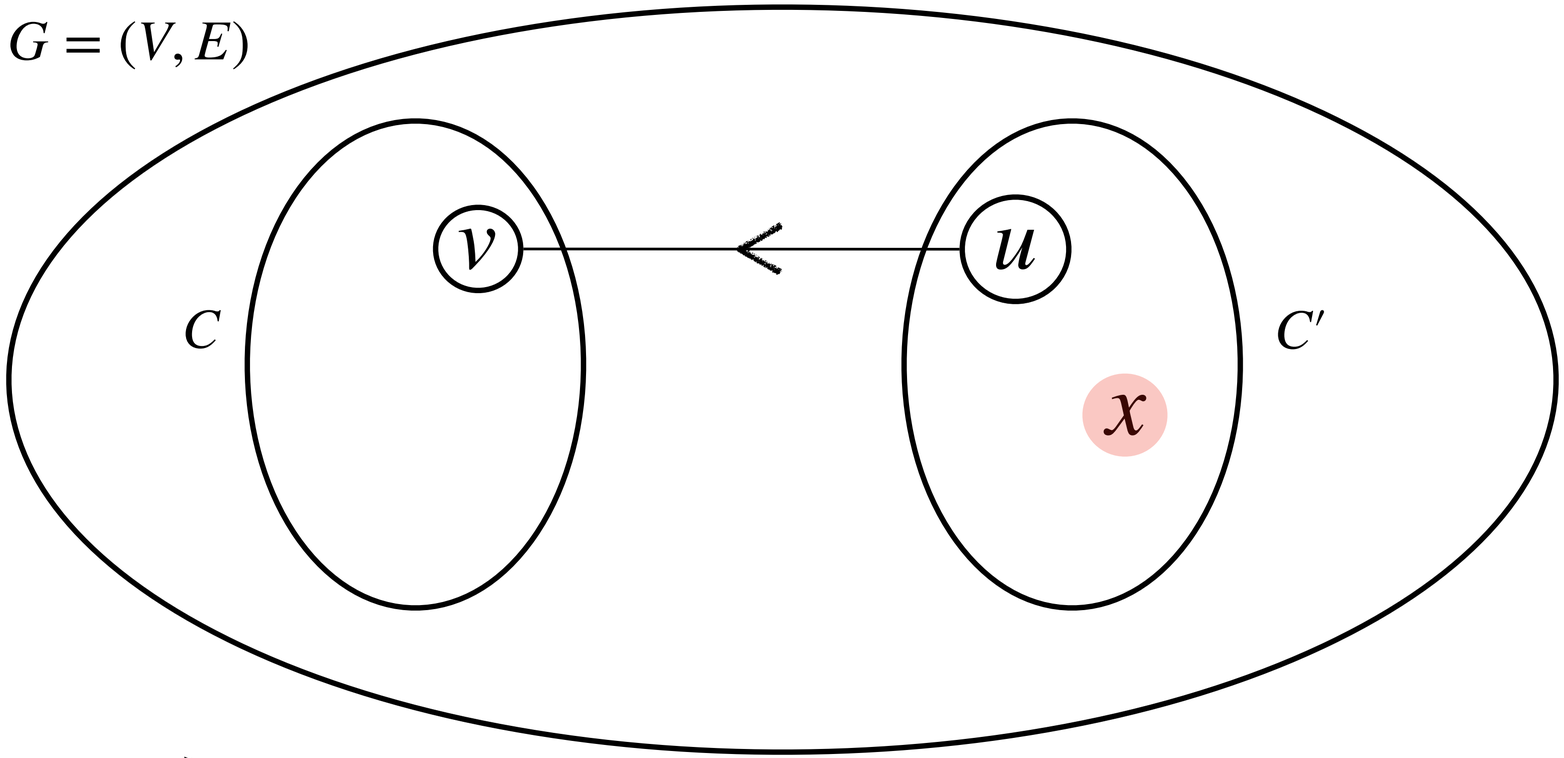


x : first vertex discovered in C'

Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$.

Suppose that there is an edge $(u, v) \in E$, where $u \in C'$ and $v \in C$. Then $f(C') > f(C)$.

Case 1: $d(C') < d(C)$

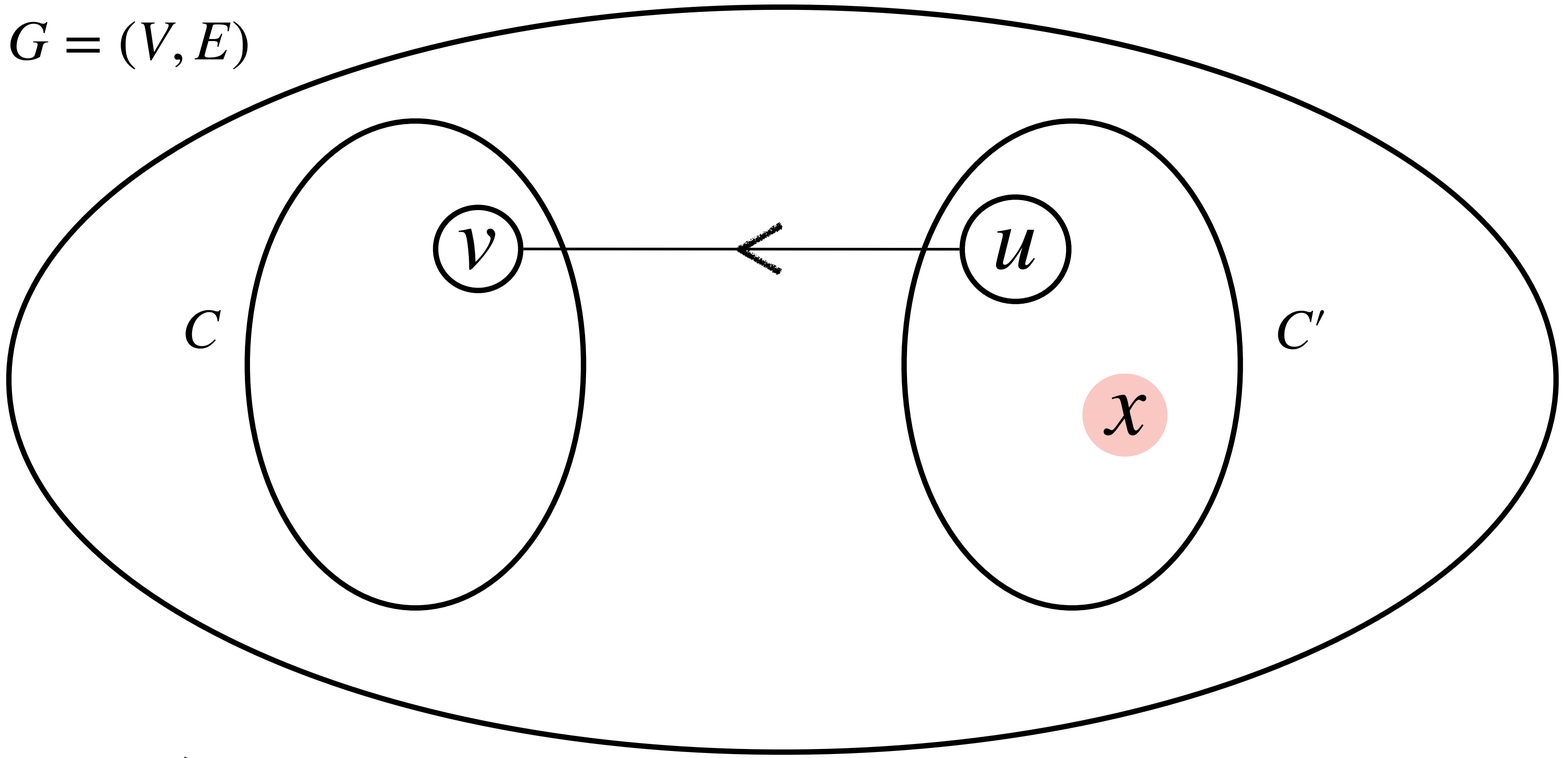


x : first vertex discovered in C' ➡ x has white paths to all vertices in C' and C

Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$.

Suppose that there is an edge $(u, v) \in E$, where $u \in C'$ and $v \in C$. Then $f(C') > f(C)$.

Case 1: $d(C') < d(C)$



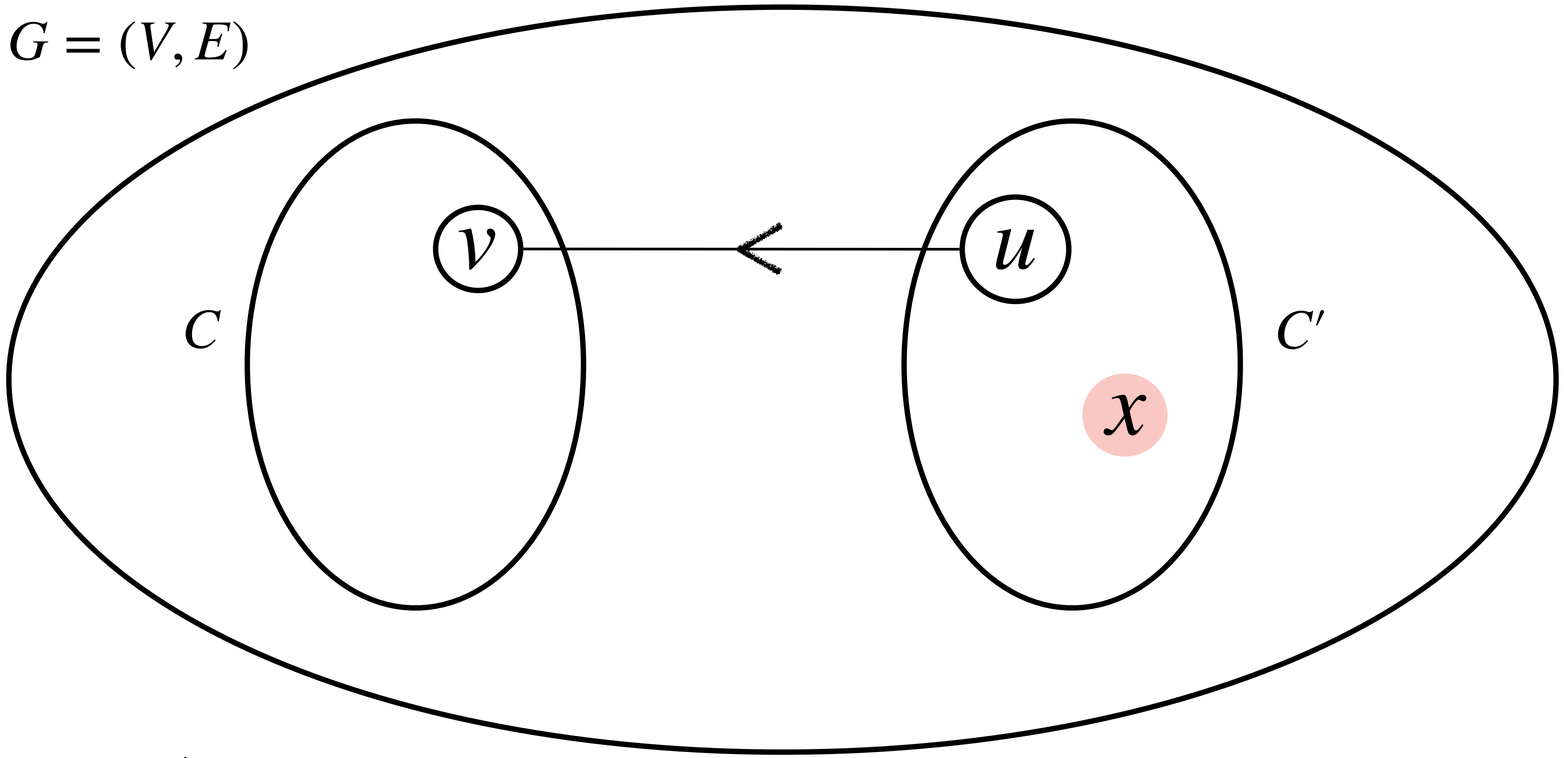
x : first vertex discovered in C' → x has white paths to all vertices in C' and C

→ x becomes ancestor of all vertices in C' and C, and has a later finish time than all of them

Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$.

Suppose that there is an edge $(u, v) \in E$, where $u \in C'$ and $v \in C$. Then $f(C') > f(C)$.

Case 1: $d(C') < d(C)$



x : first vertex discovered in C' → x has white paths to all vertices in C' and C

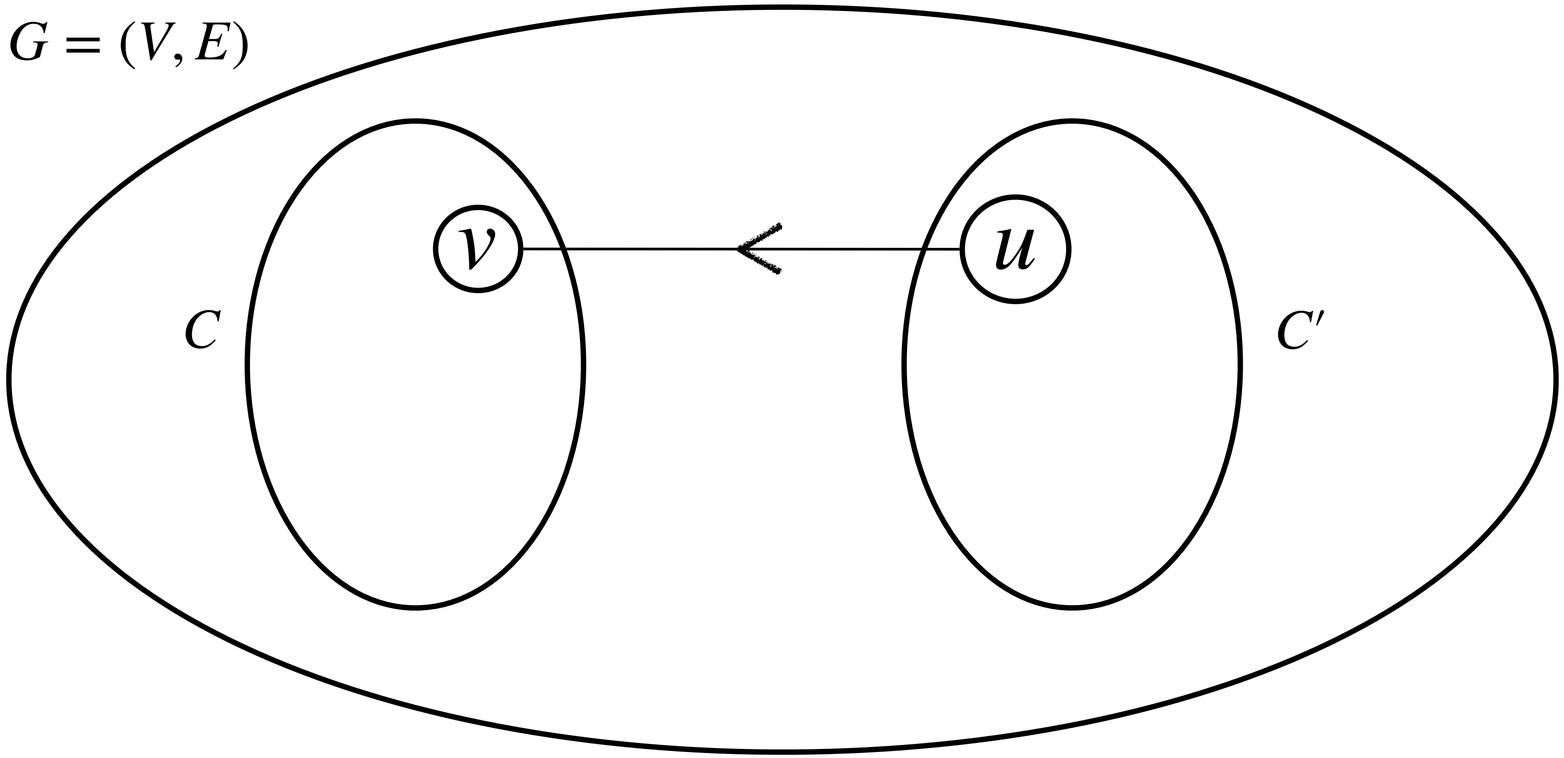
→ x becomes ancestor of all vertices in C' and C, and has a later finish time than all of them → $f(C') > f(C)$

Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$.

Suppose that there is an edge $(u, v) \in E$, where $u \in C'$ and $v \in C$. Then $f(C') > f(C)$.

Case 1: $d(C') < d(C)$

Case 2: $d(C) < d(C')$

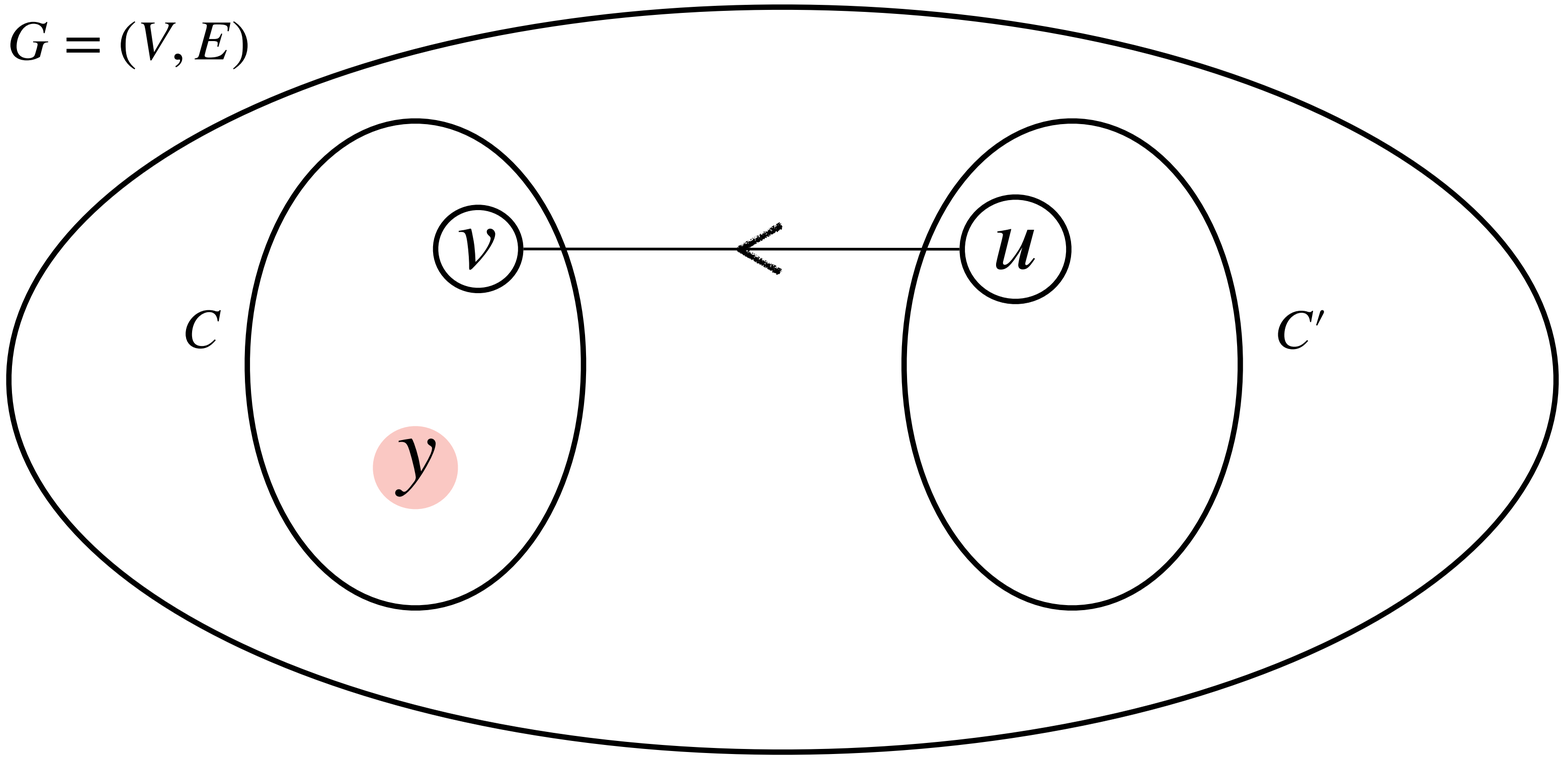


Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$.

Suppose that there is an edge $(u, v) \in E$, where $u \in C'$ and $v \in C$. Then $f(C') > f(C)$.

Case 1: $d(C') < d(C)$

Case 2: $d(C) < d(C')$



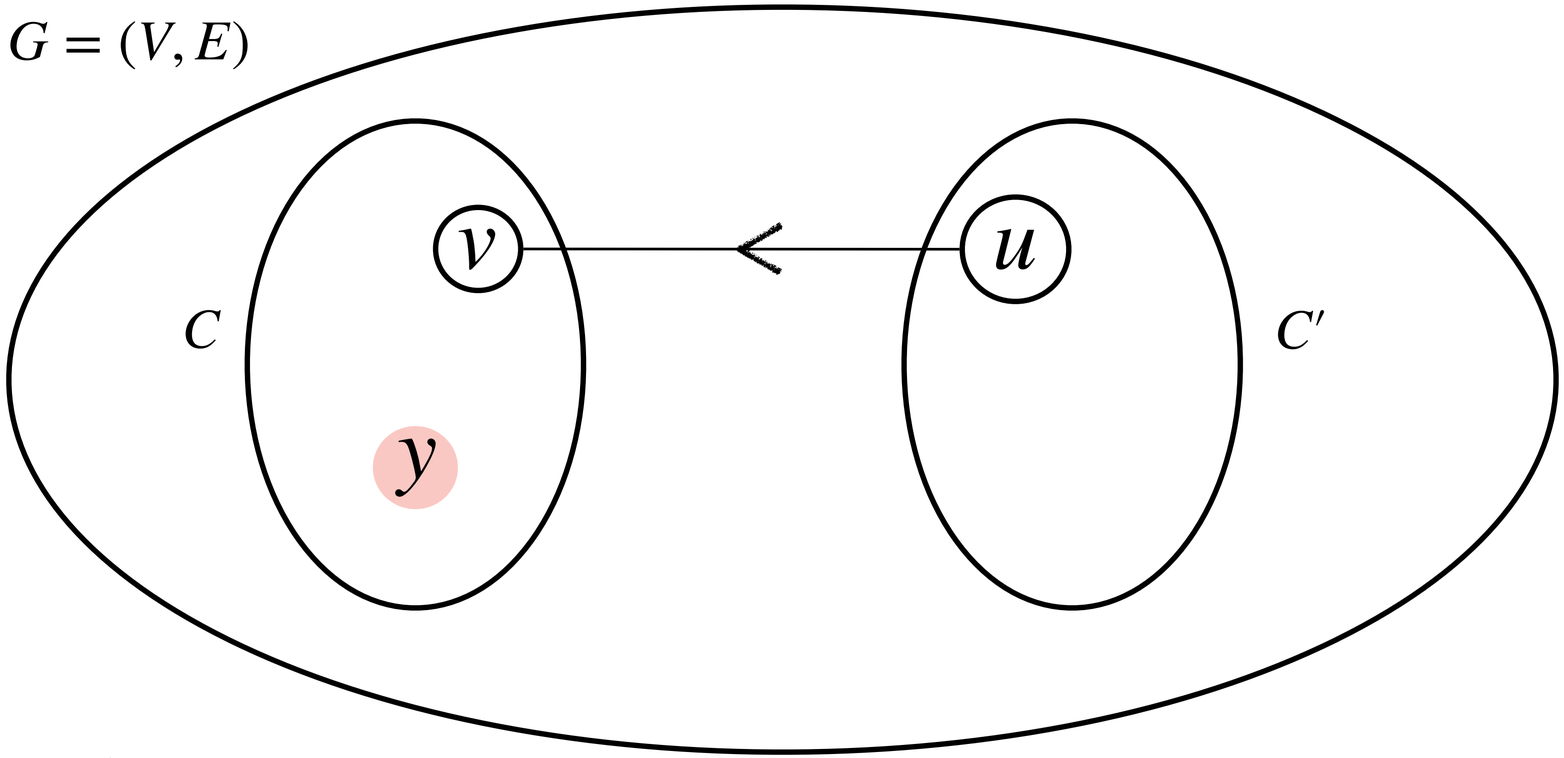
y : first vertex discovered in C

Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$.

Suppose that there is an edge $(u, v) \in E$, where $u \in C'$ and $v \in C$. Then $f(C') > f(C)$.

Case 1: $d(C') < d(C)$

Case 2: $d(C) < d(C')$



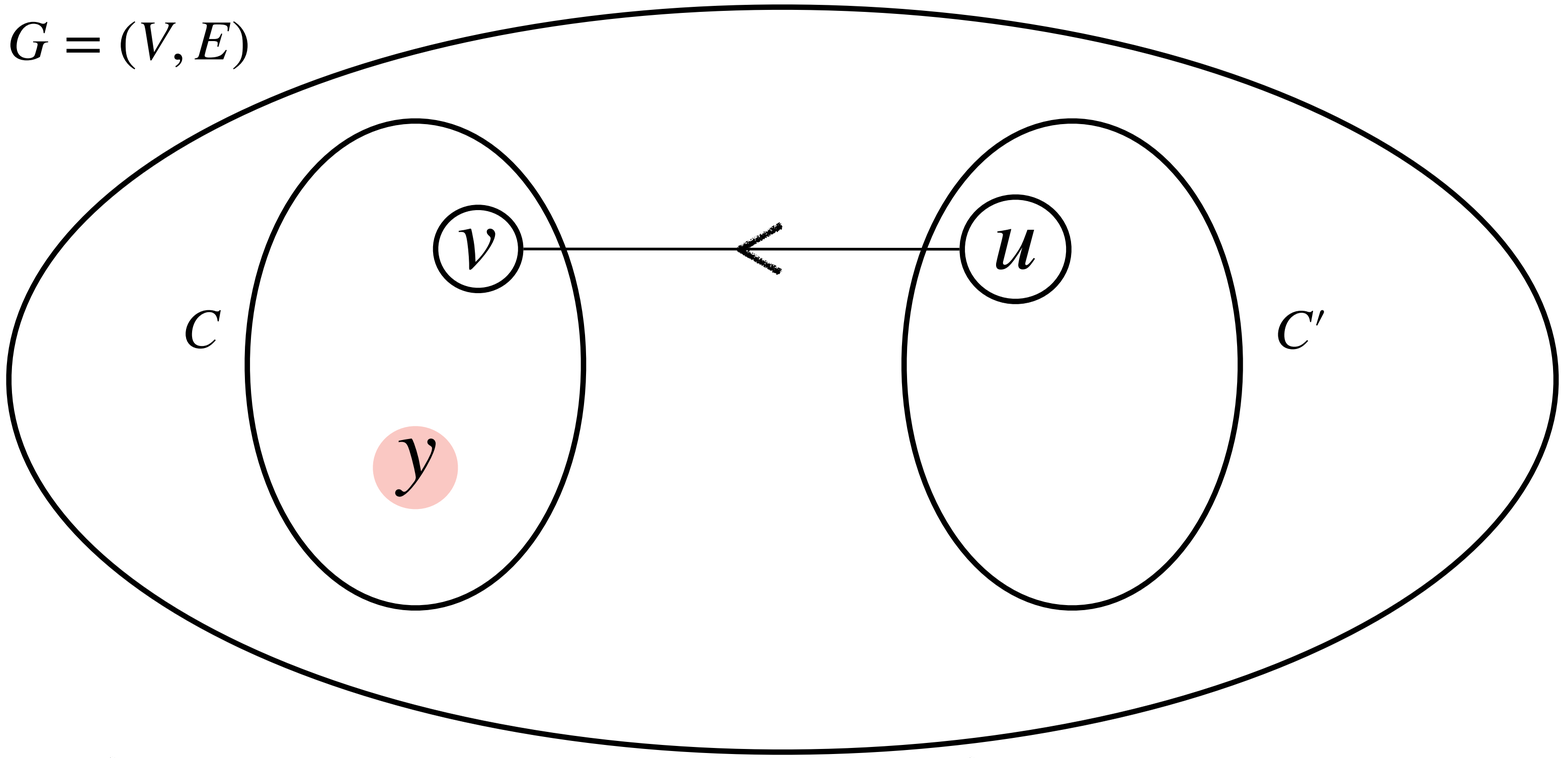
y : first vertex discovered in C ➡ y cannot reach C' via any path

Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$.

Suppose that there is an edge $(u, v) \in E$, where $u \in C'$ and $v \in C$. Then $f(C') > f(C)$.

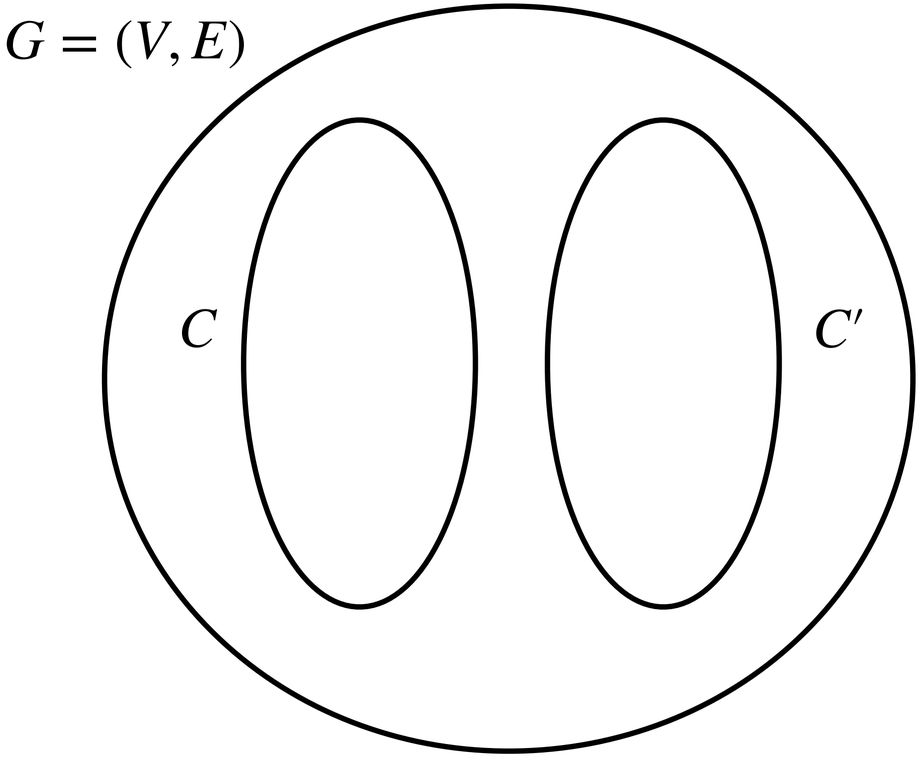
Case 1: $d(C') < d(C)$

Case 2: $d(C) < d(C')$

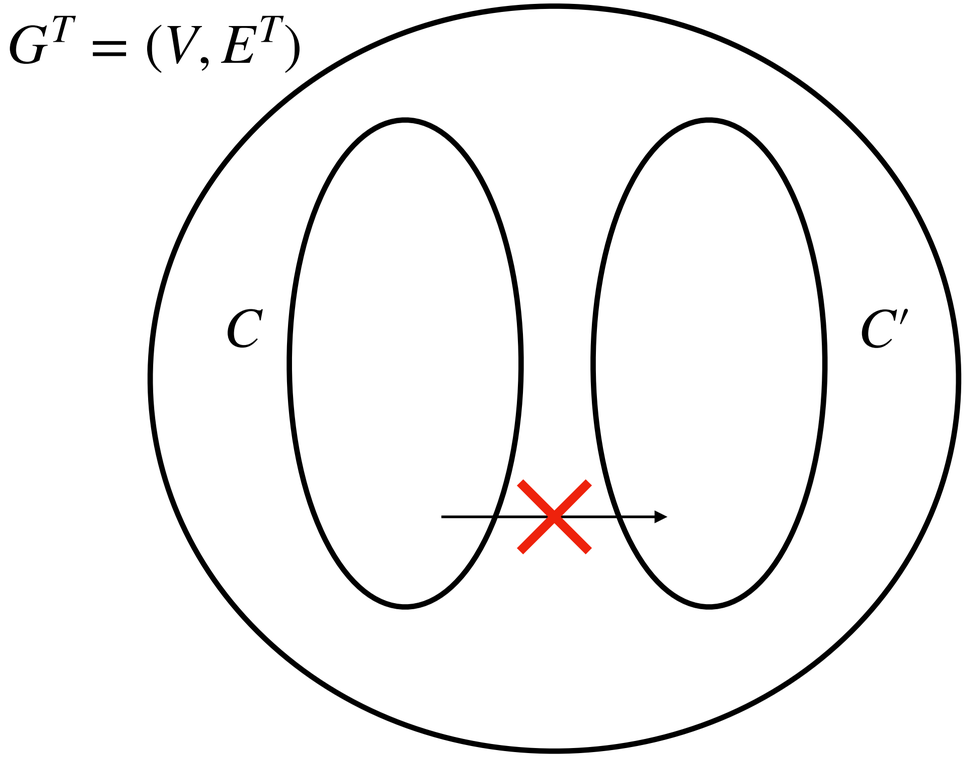


y : first vertex discovered in C → y cannot reach C' via any path → $f(C) = y.f < d(C') < f(C')$

Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$, and suppose that $f(C) > f(C')$. Then E^T contains no edge (v, u) such that $u \in C'$ and $v \in C$.

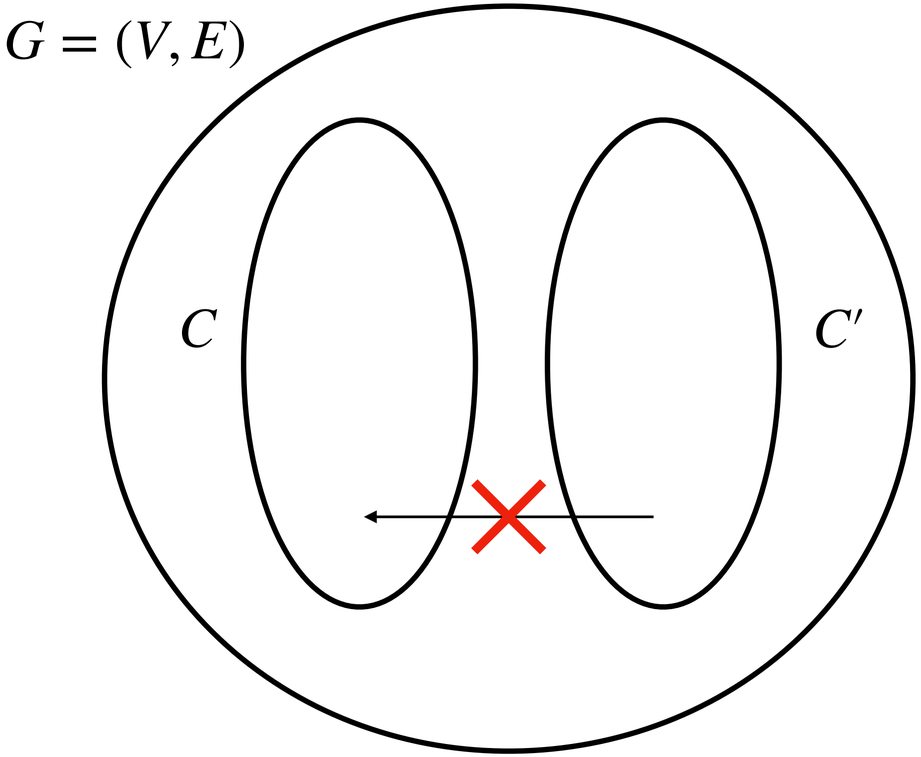


$f(C) > f(C')$

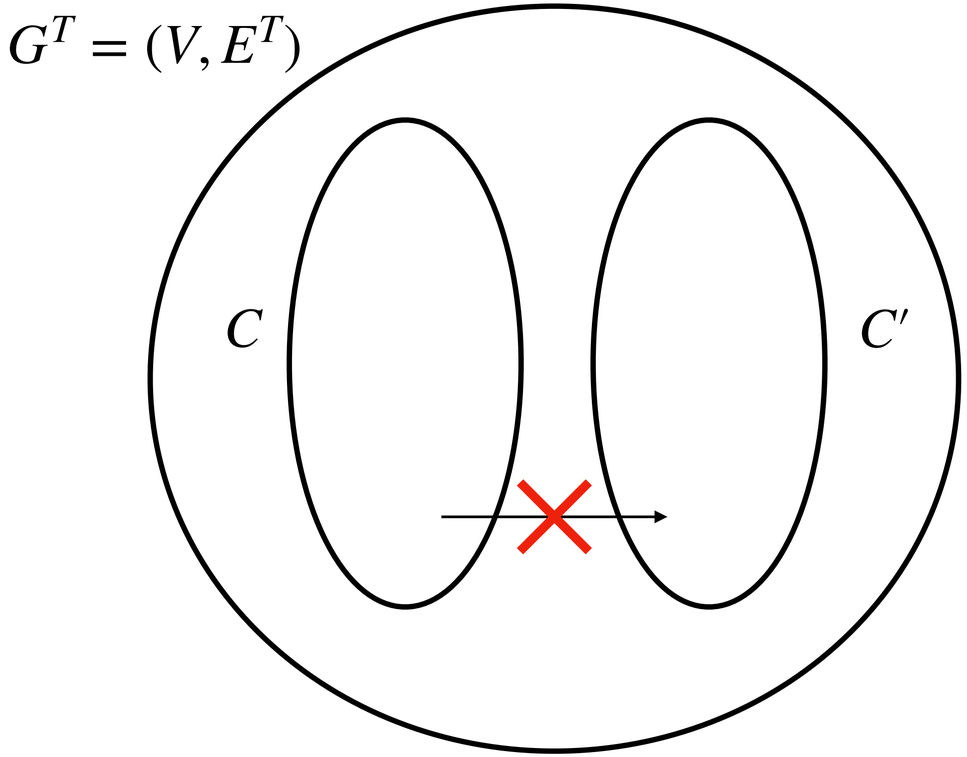


Why?

Lemma: Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$, and suppose that $f(C) > f(C')$. Then E^T contains no edge (v, u) such that $u \in C'$ and $v \in C$.



$f(C) > f(C')$

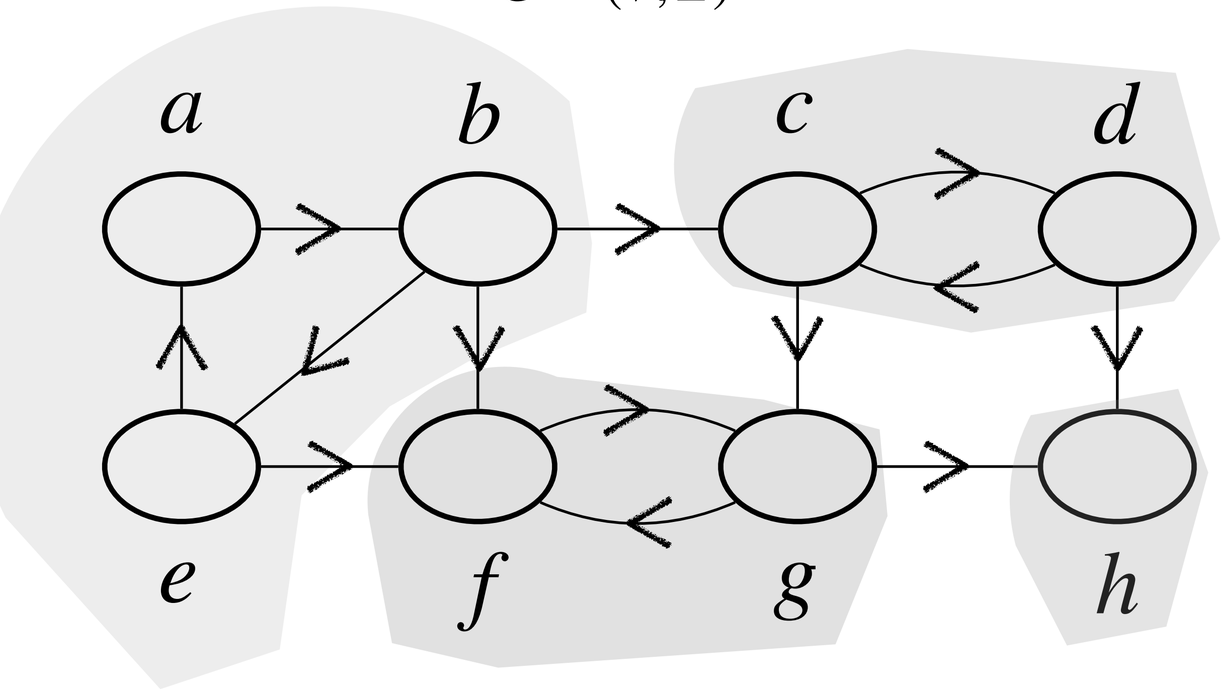


Algorithm for the Strongly Connected Component Problem:

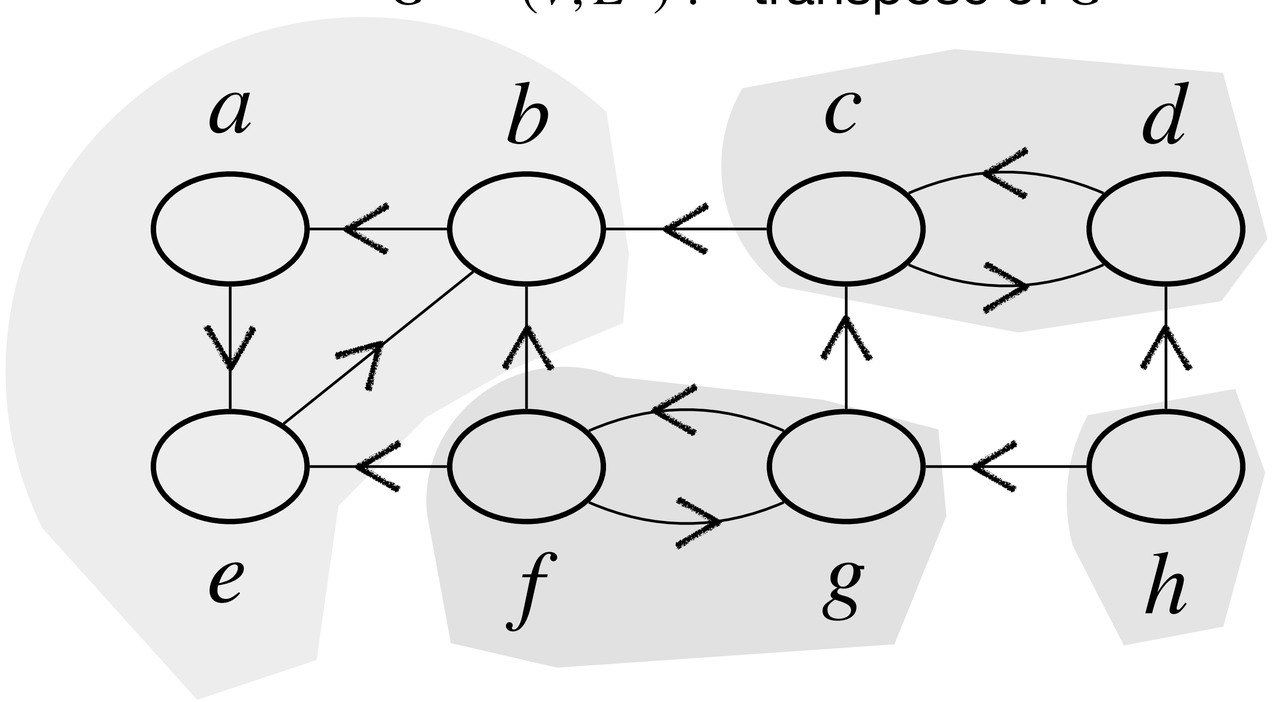
Let's prove the algorithm's correctness.

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



$G^T = (V, E^T)$: transpose of G

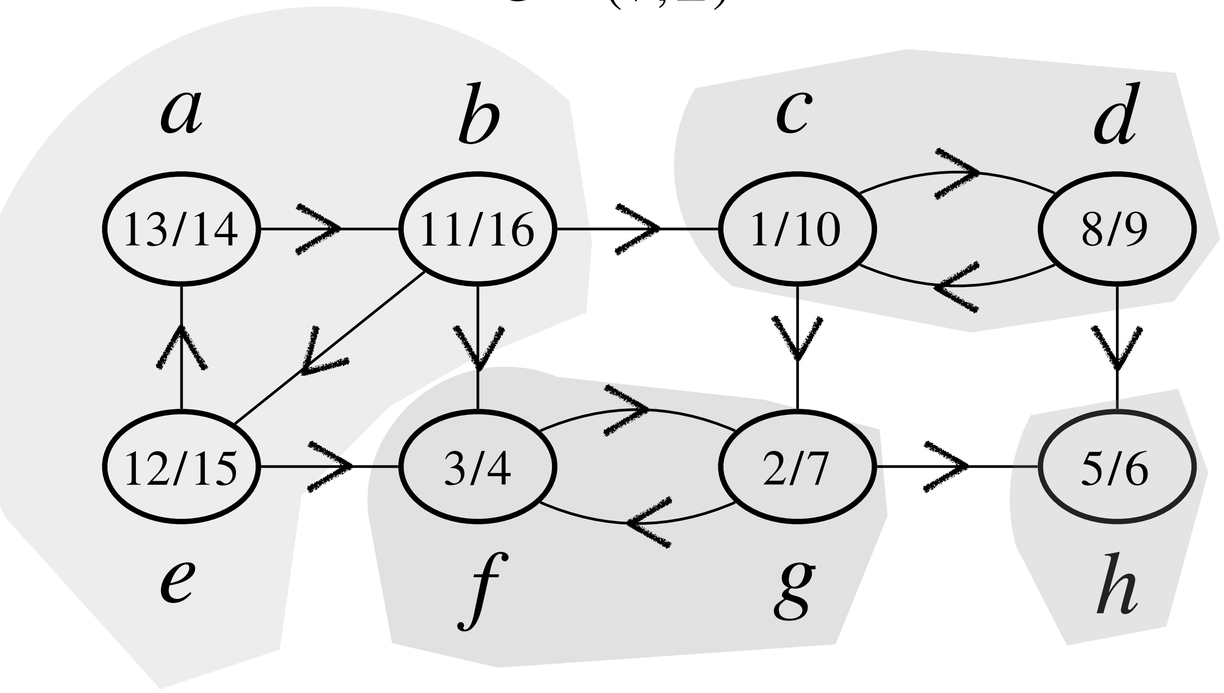


Algorithm for the Strongly Connected Component Problem:

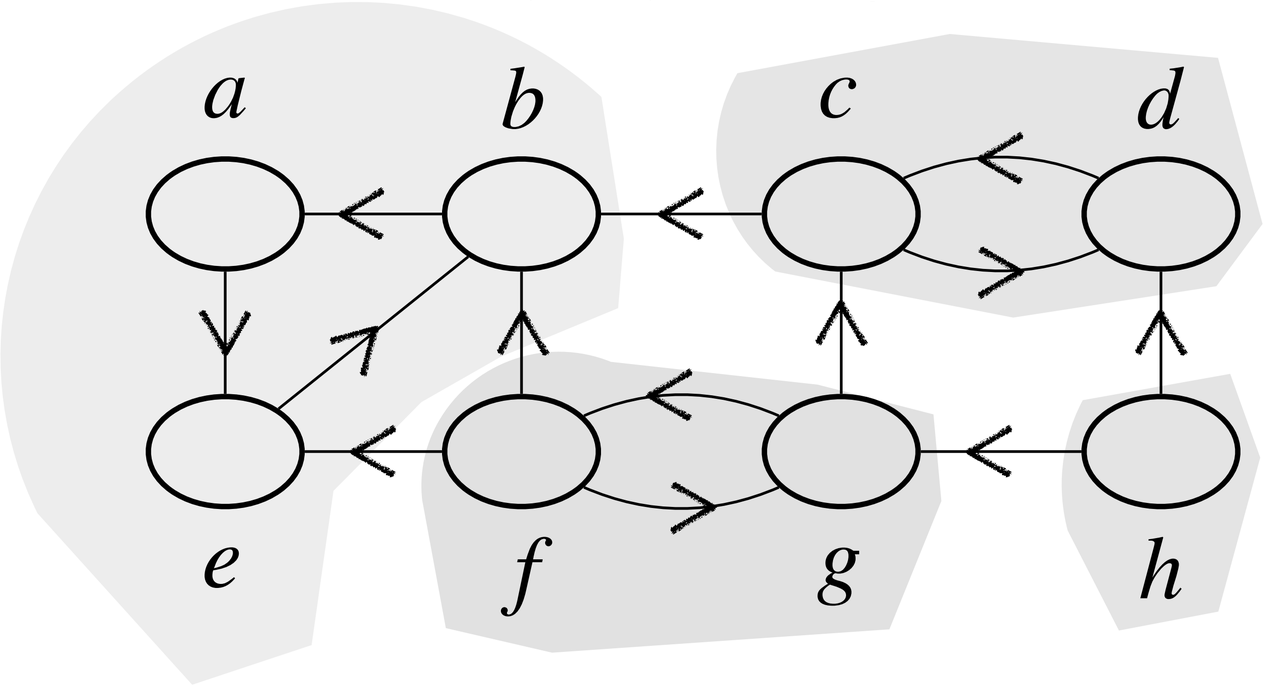
Let's prove the algorithm's correctness.

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



$G^T = (V, E^T)$: transpose of G



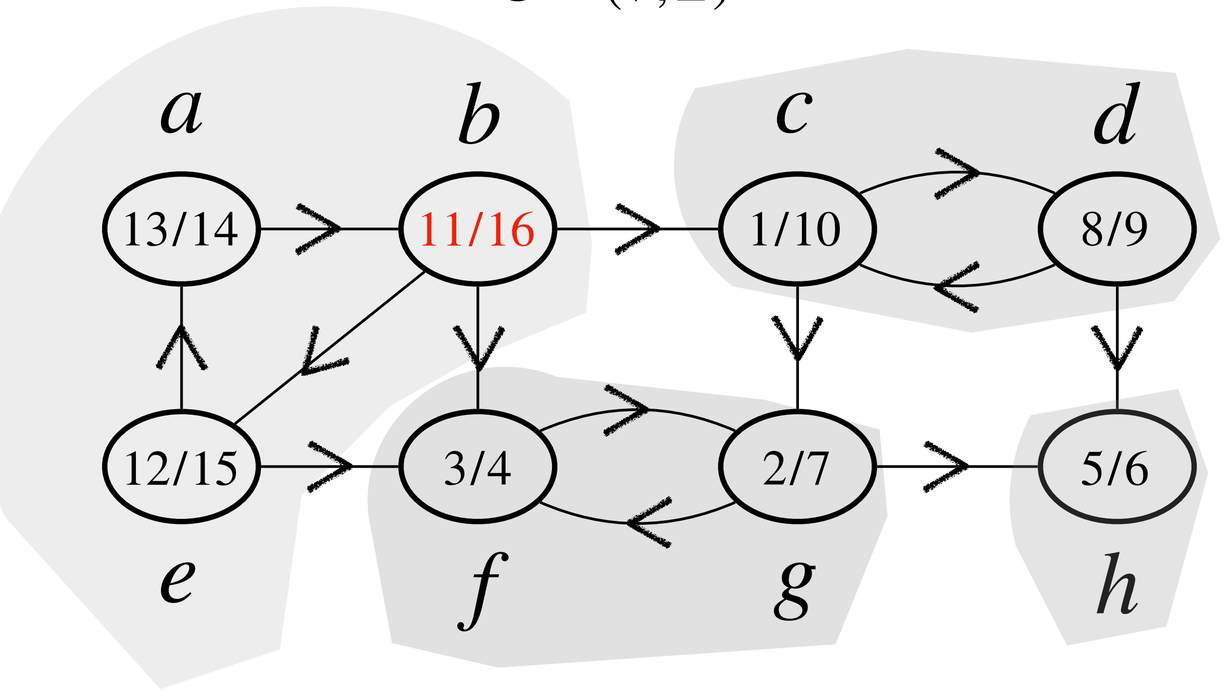
Algorithm for the Strongly Connected Component Problem:

Let's prove the algorithm's correctness.

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

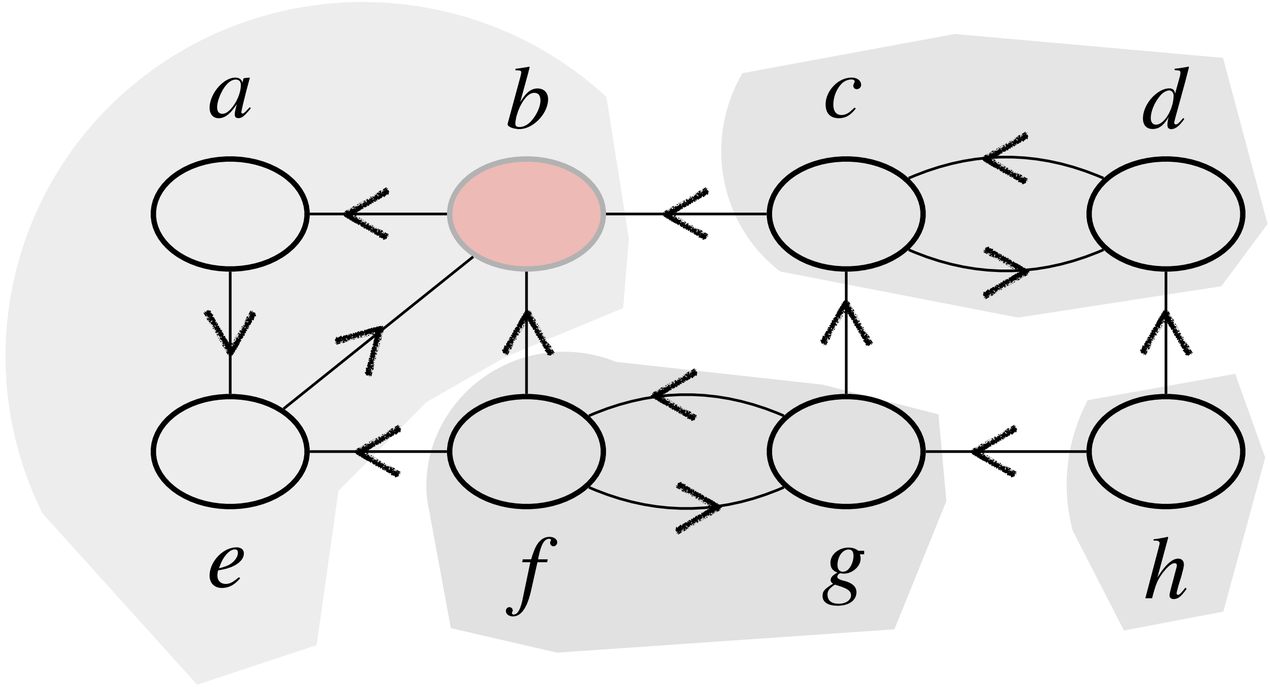
The SCC containing b has the greatest finish time.

$G = (V, E)$



The SCC containing b has no edge toward other SCC

$G^T = (V, E^T)$: transpose of G



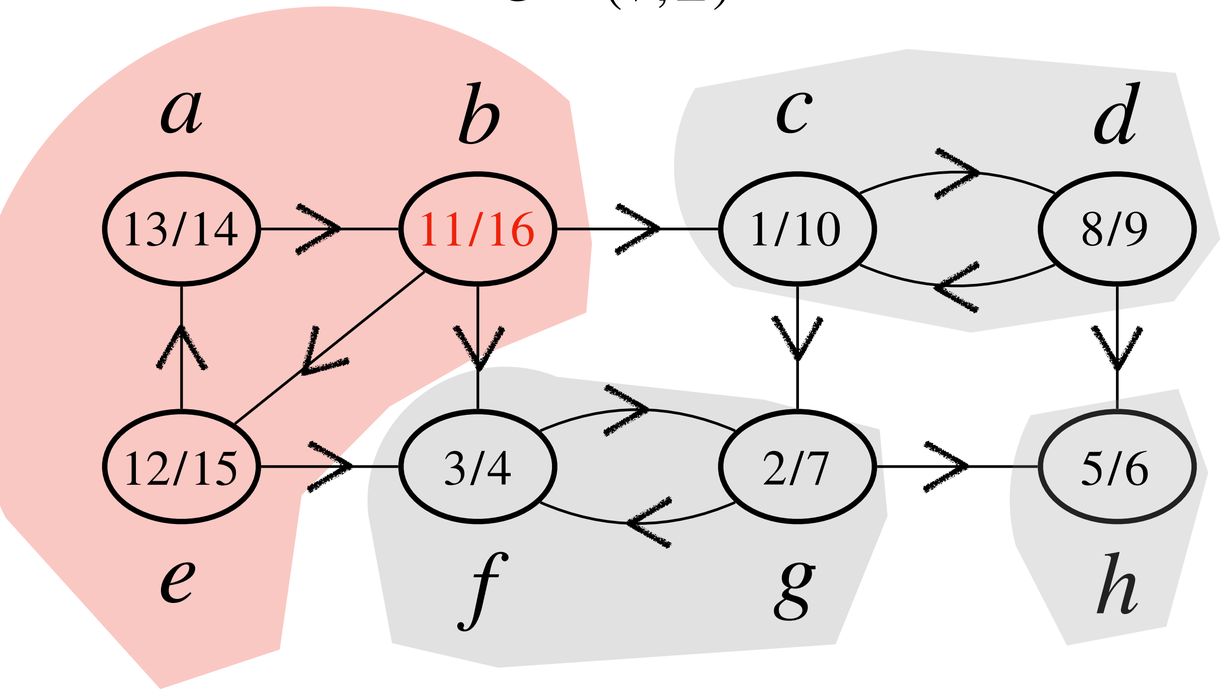
Algorithm for the Strongly Connected Component Problem:

Let's prove the algorithm's correctness.

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

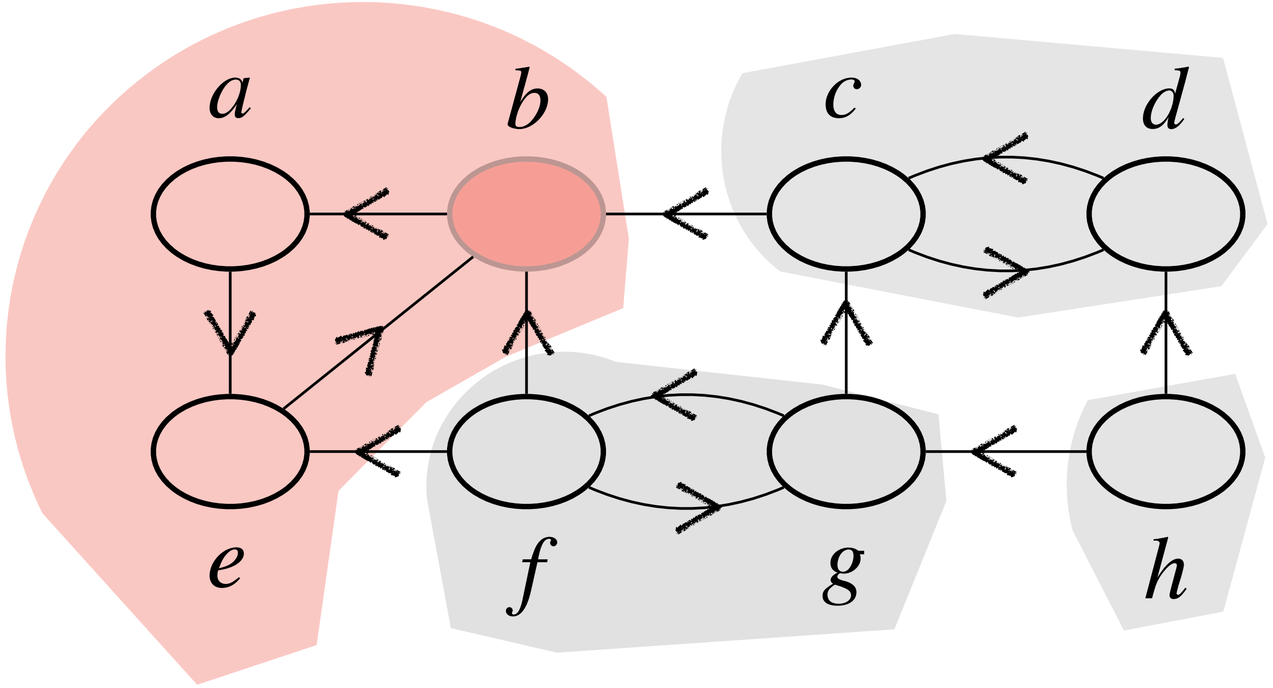
The SCC containing b has the greatest finish time.

$G = (V, E)$



The SCC containing b has no edge toward other SCC

$G^T = (V, E^T)$: transpose of G

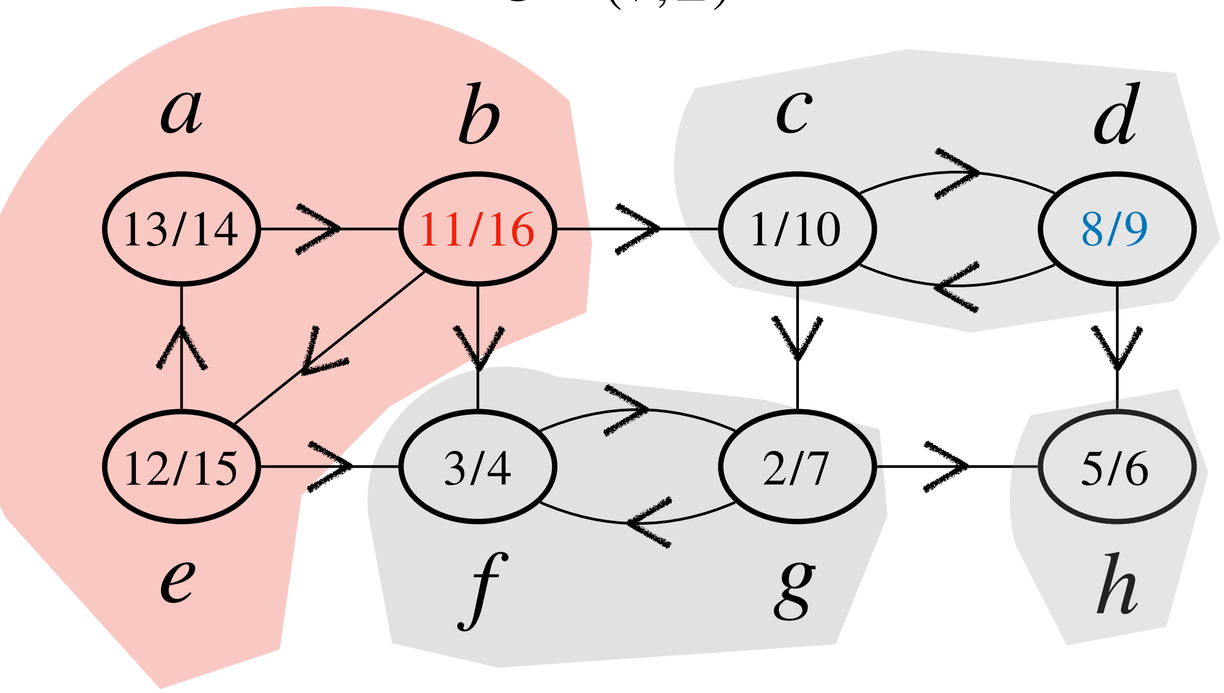


Algorithm for the Strongly Connected Component Problem:

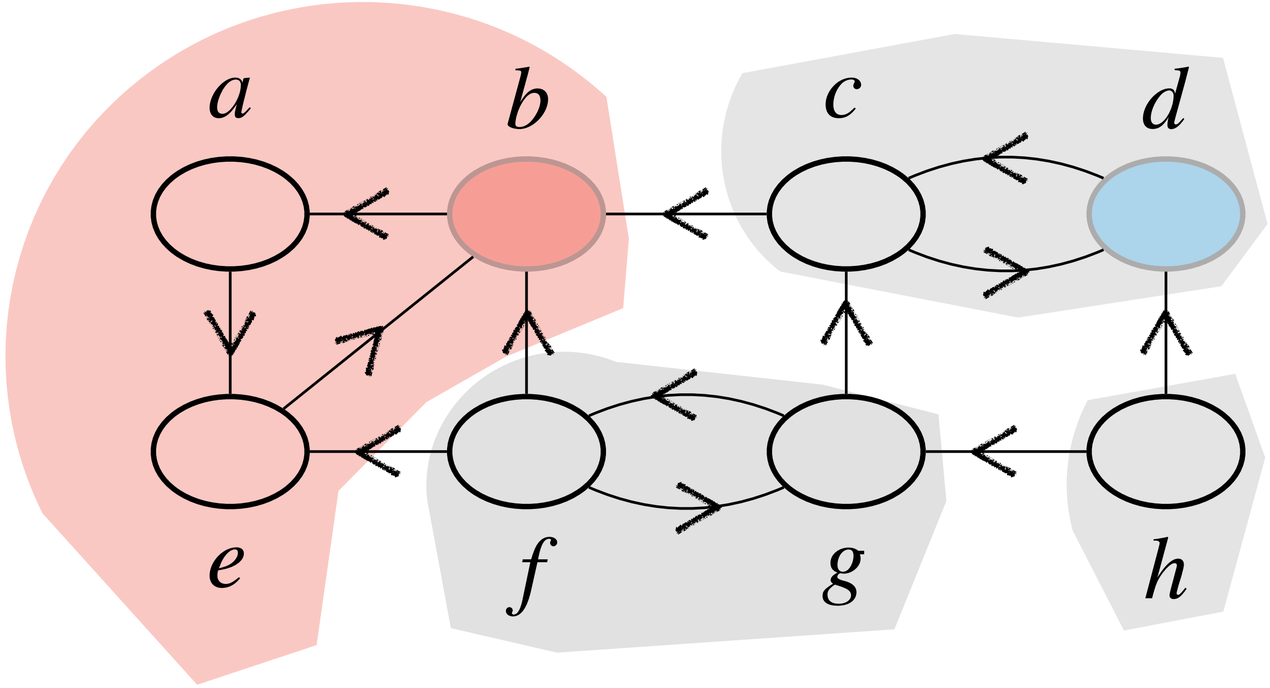
Let's prove the algorithm's correctness.

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



$G^T = (V, E^T)$: transpose of G

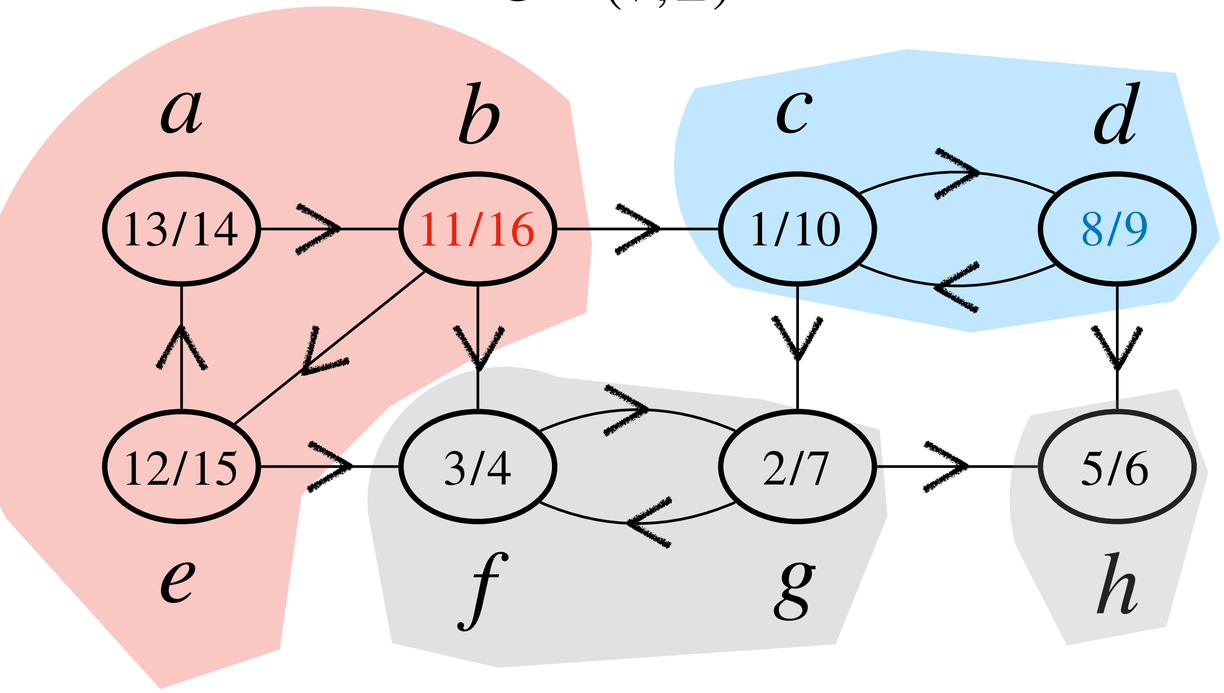


Algorithm for the Strongly Connected Component Problem:

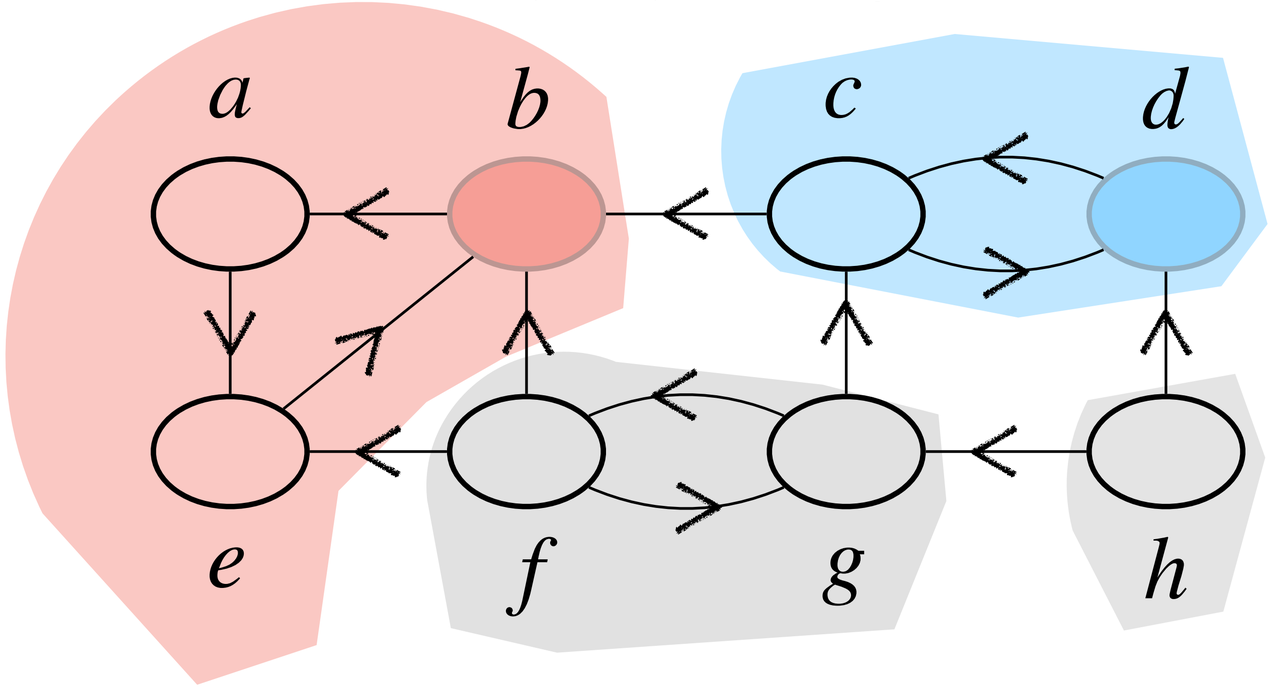
Let's prove the algorithm's correctness.

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



$G^T = (V, E^T)$: transpose of G

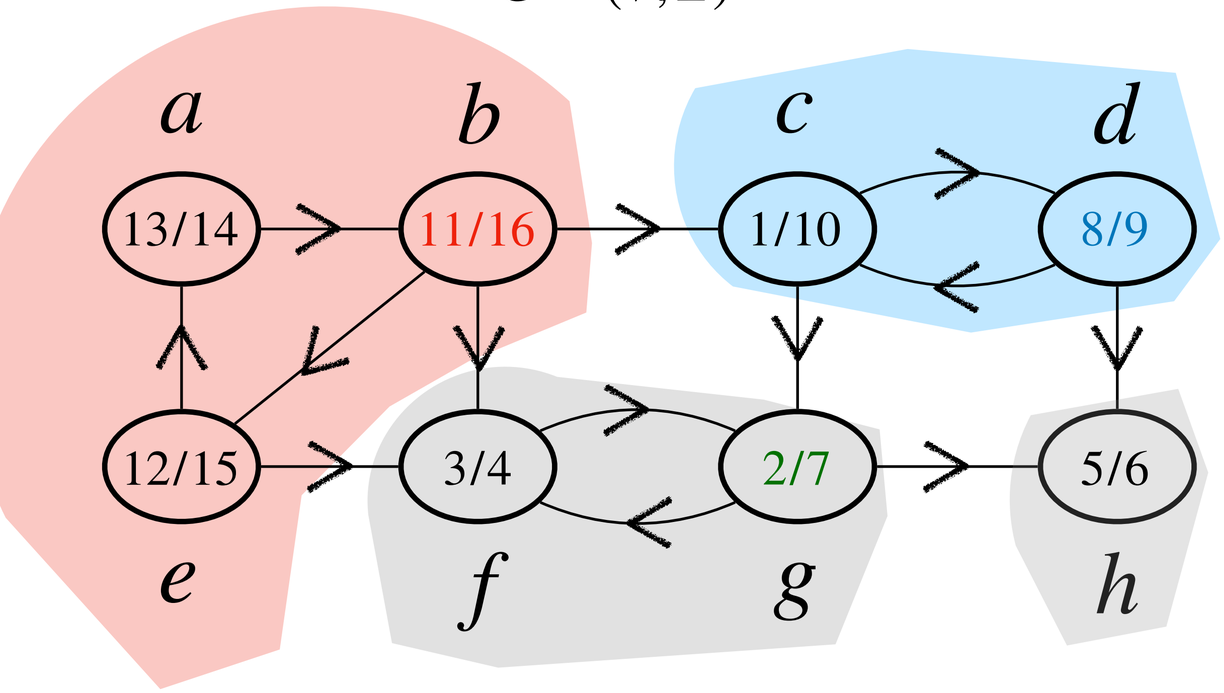


Algorithm for the Strongly Connected Component Problem:

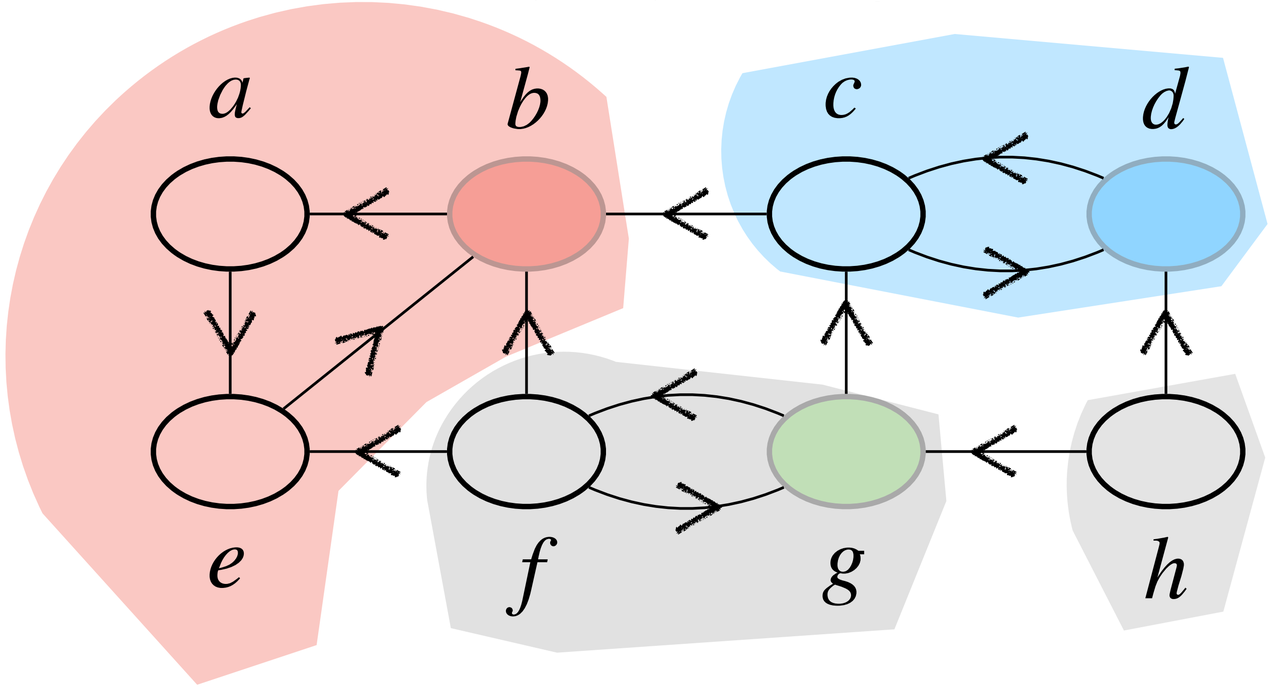
Let's prove the algorithm's correctness.

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



$G^T = (V, E^T)$: transpose of G

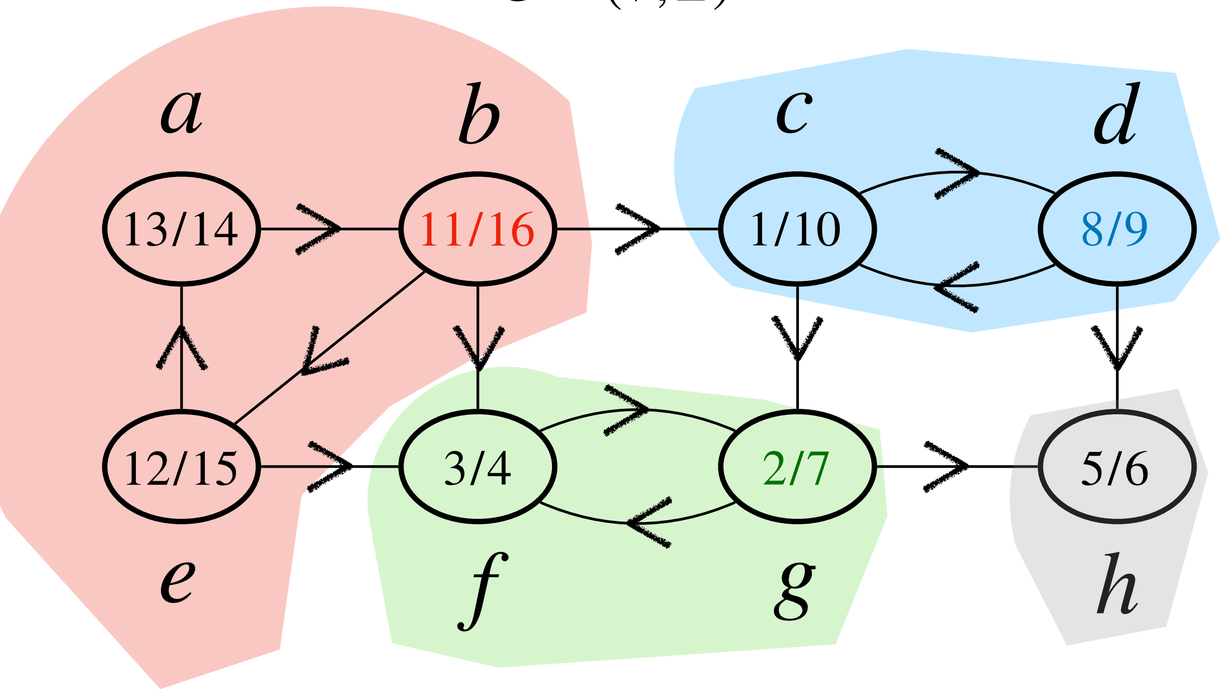


Algorithm for the Strongly Connected Component Problem:

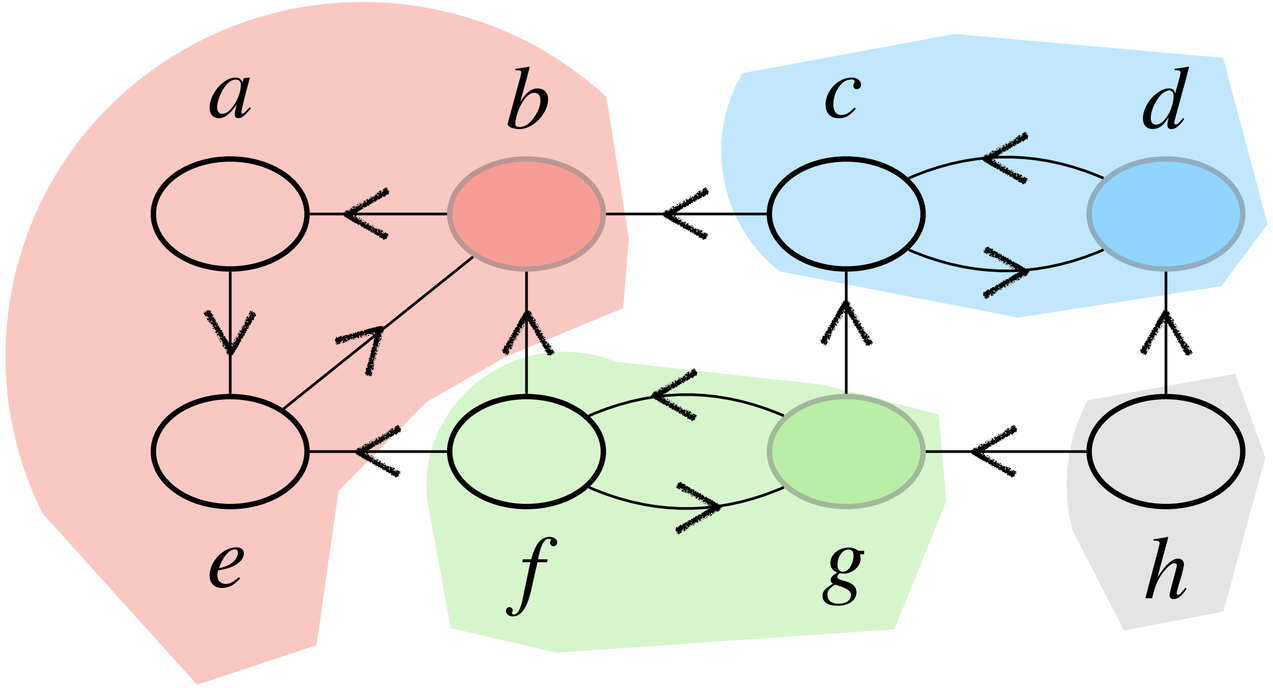
Let's prove the algorithm's correctness.

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



$G^T = (V, E^T)$: transpose of G

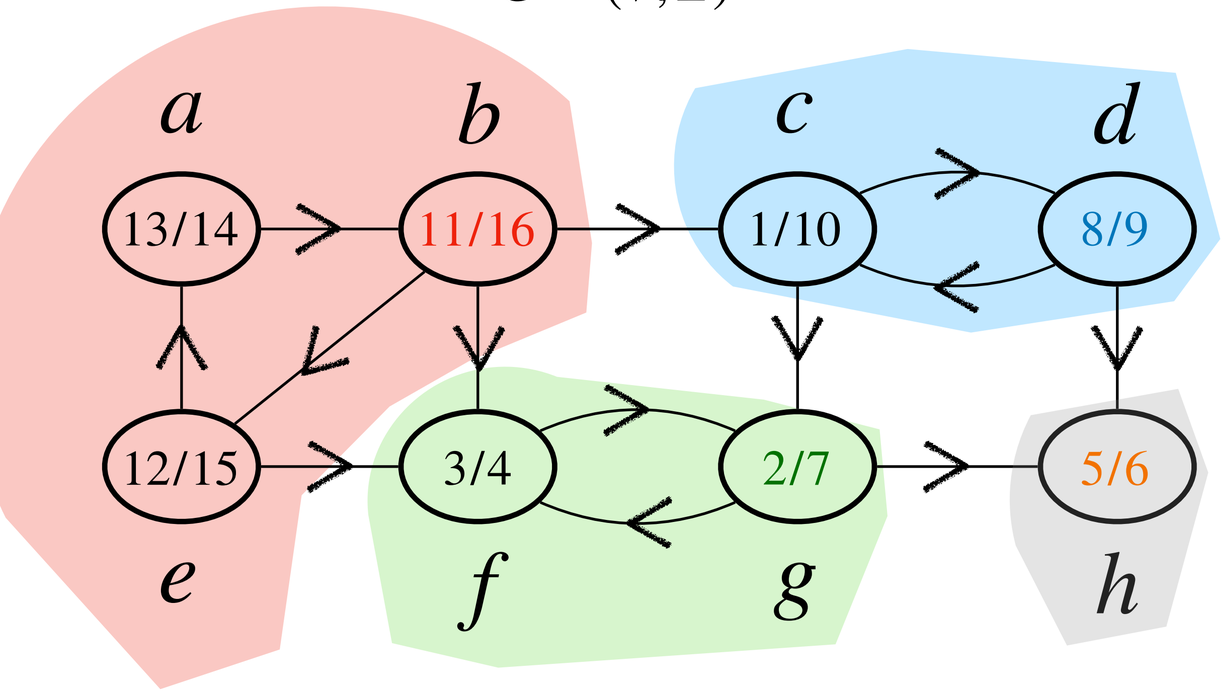


Algorithm for the Strongly Connected Component Problem:

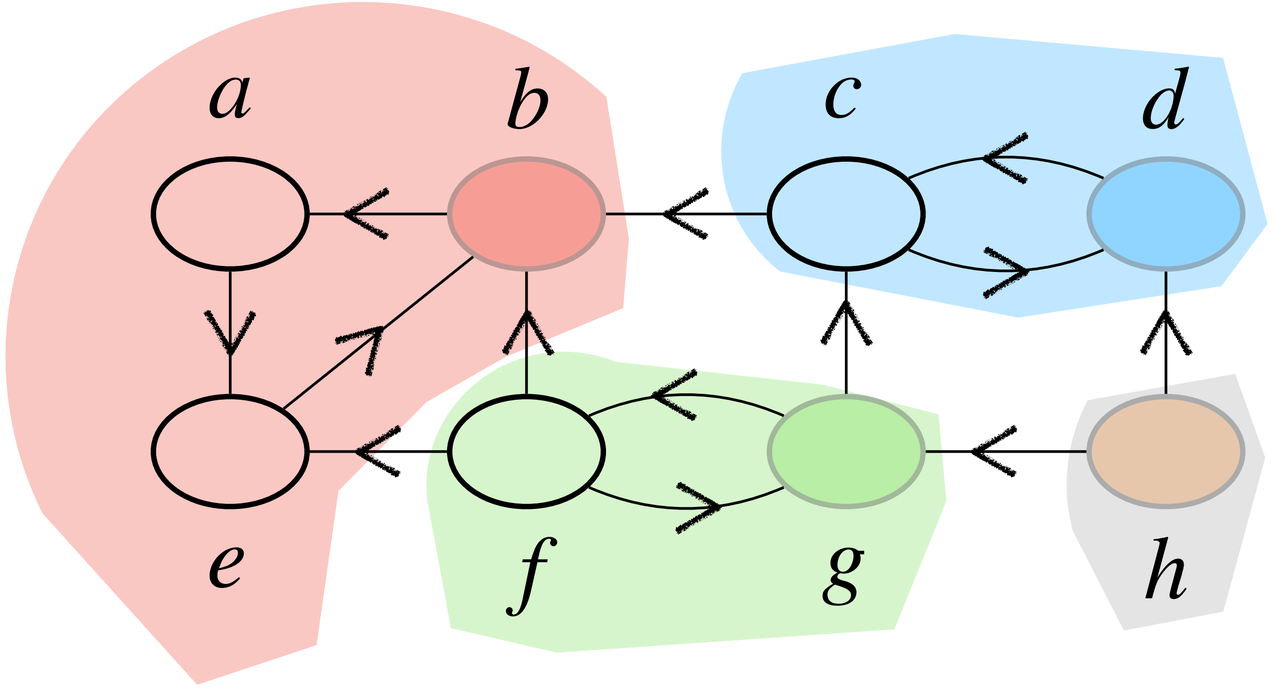
Let's prove the algorithm's correctness.

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

$G = (V, E)$



$G^T = (V, E^T)$: transpose of G

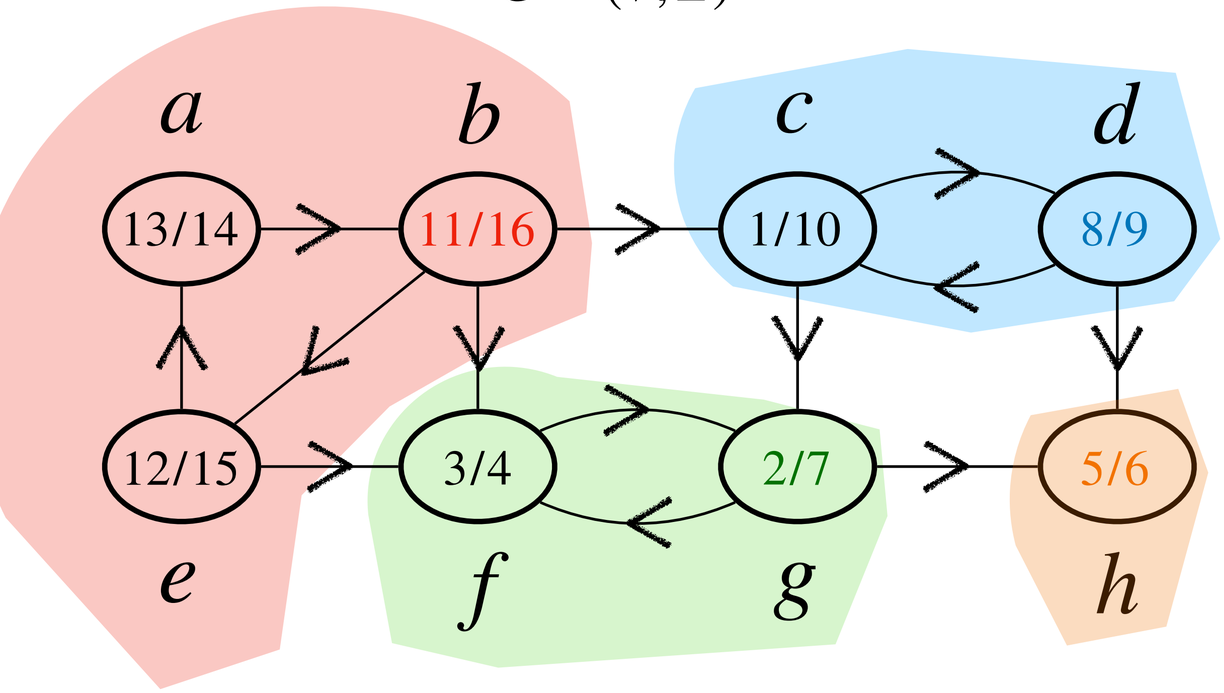


Algorithm for the Strongly Connected Component Problem:

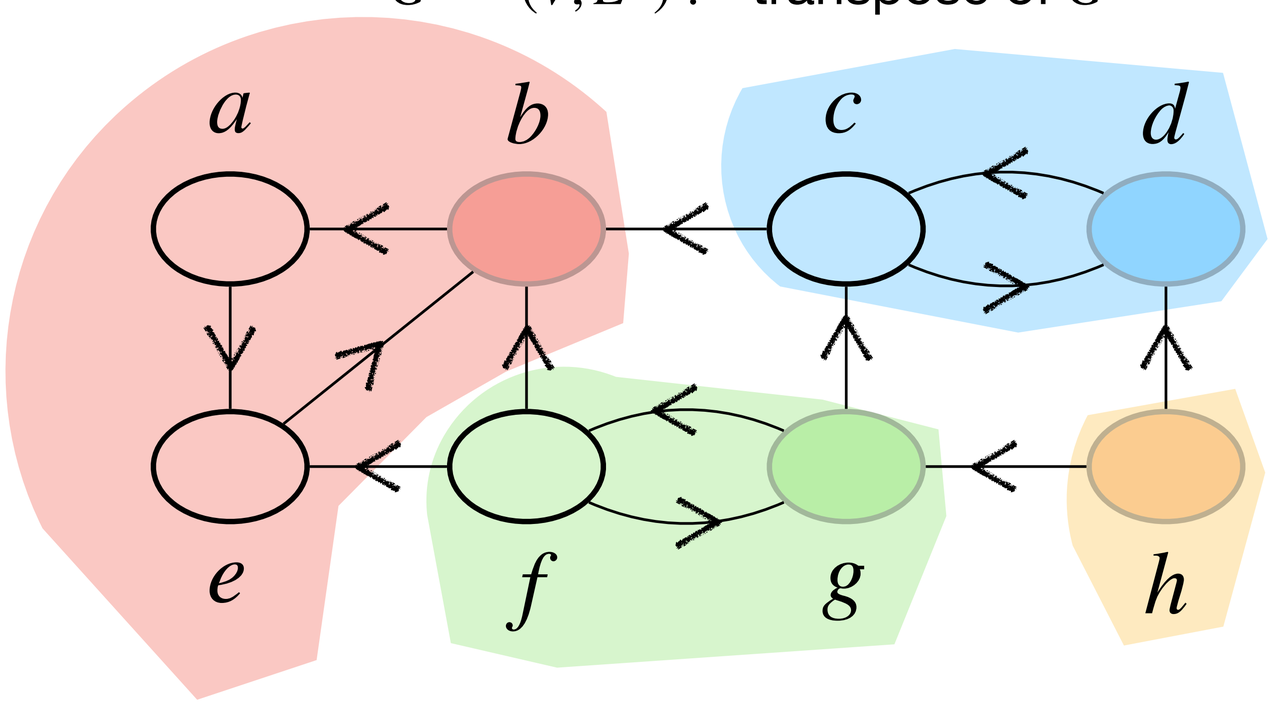
Let's prove the algorithm's correctness.

1. run DFS on G to compute finish times $u.f$ for each vertice u
2. create the transpose graph G^T
3. run DFS on G^T, but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4. output the vertices of each tree in the DFS forest formed in line 3 as a separate SCC.

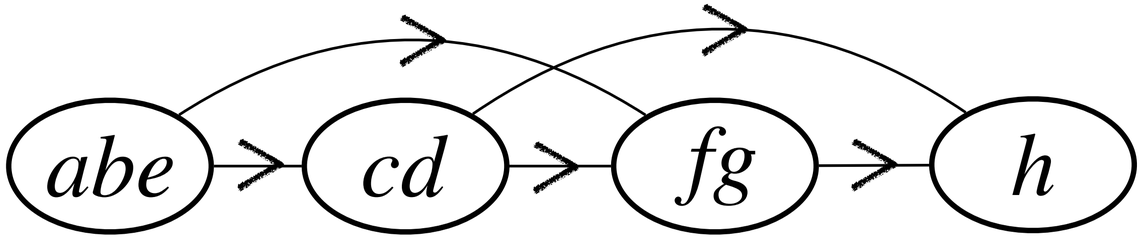
$G = (V, E)$

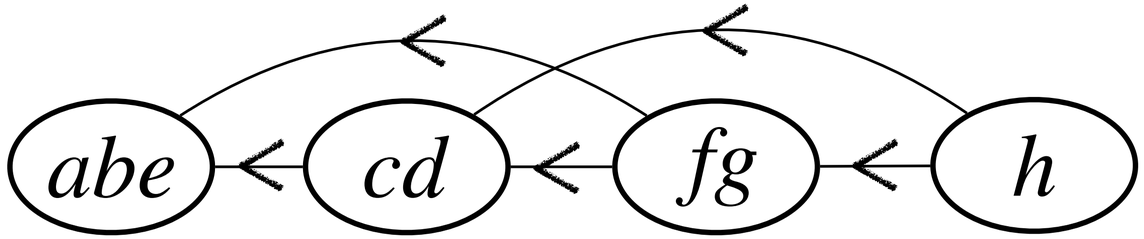


$G^T = (V, E^T)$: transpose of G

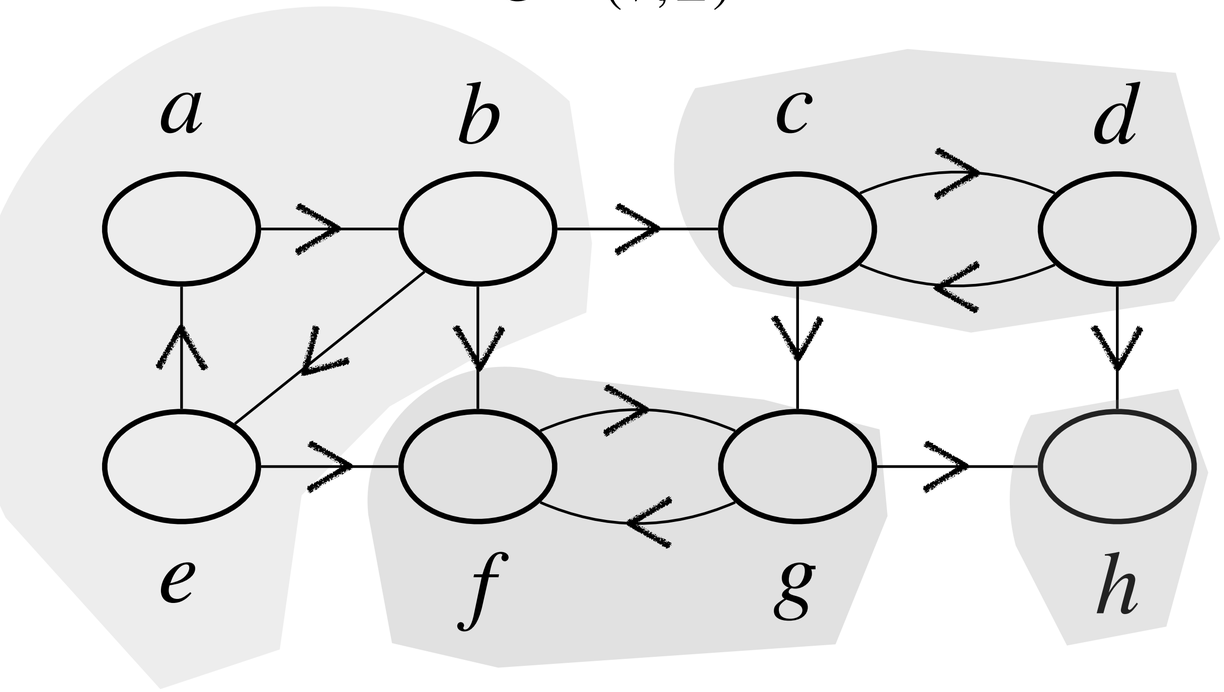


Another way to understand the algorithm's correctness.

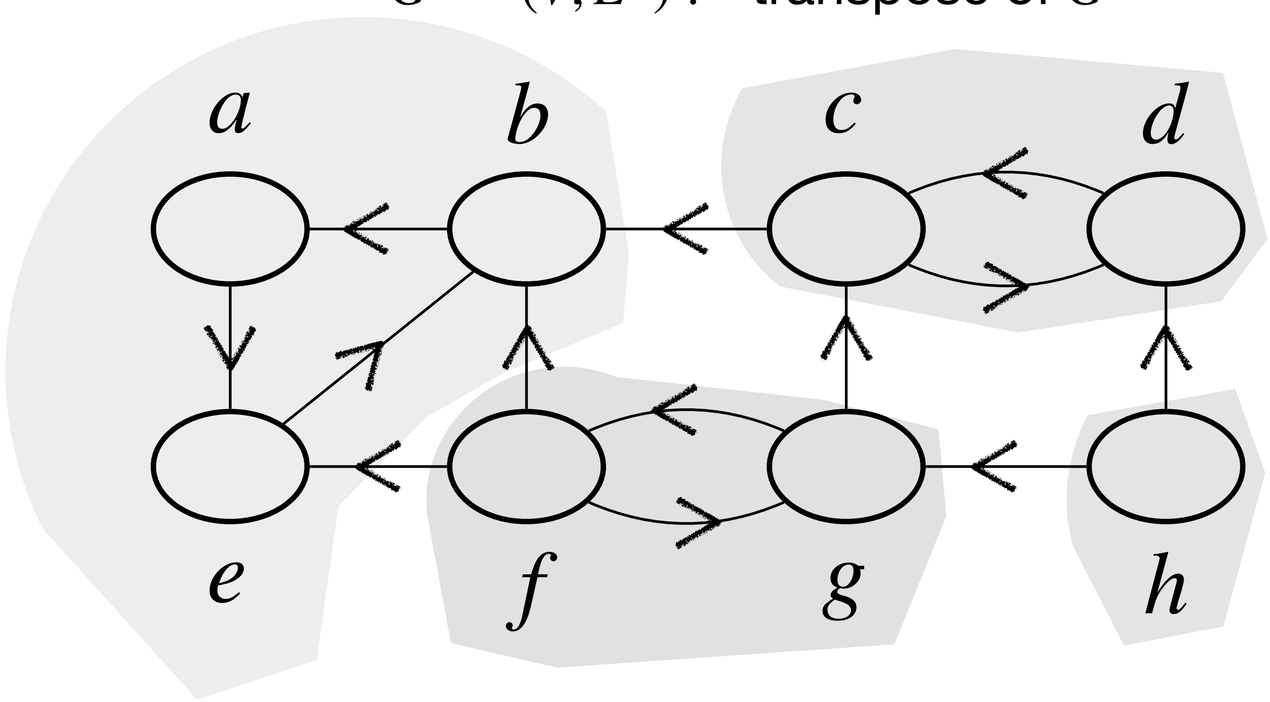


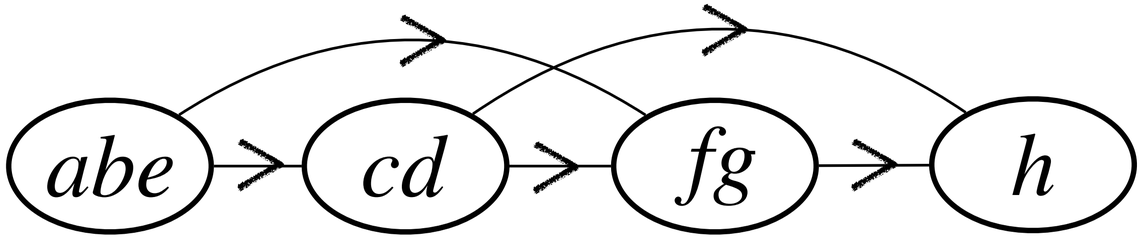


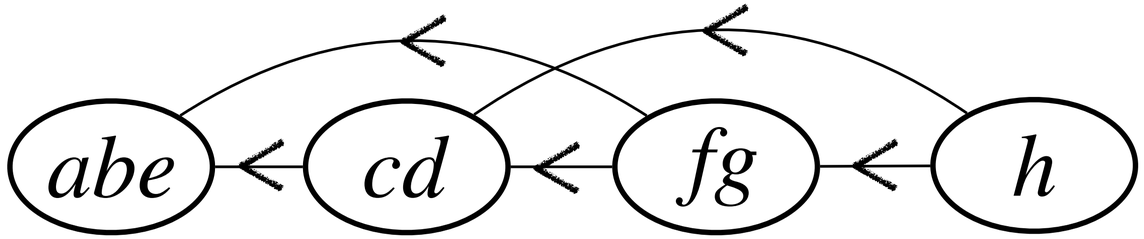
$G = (V, E)$



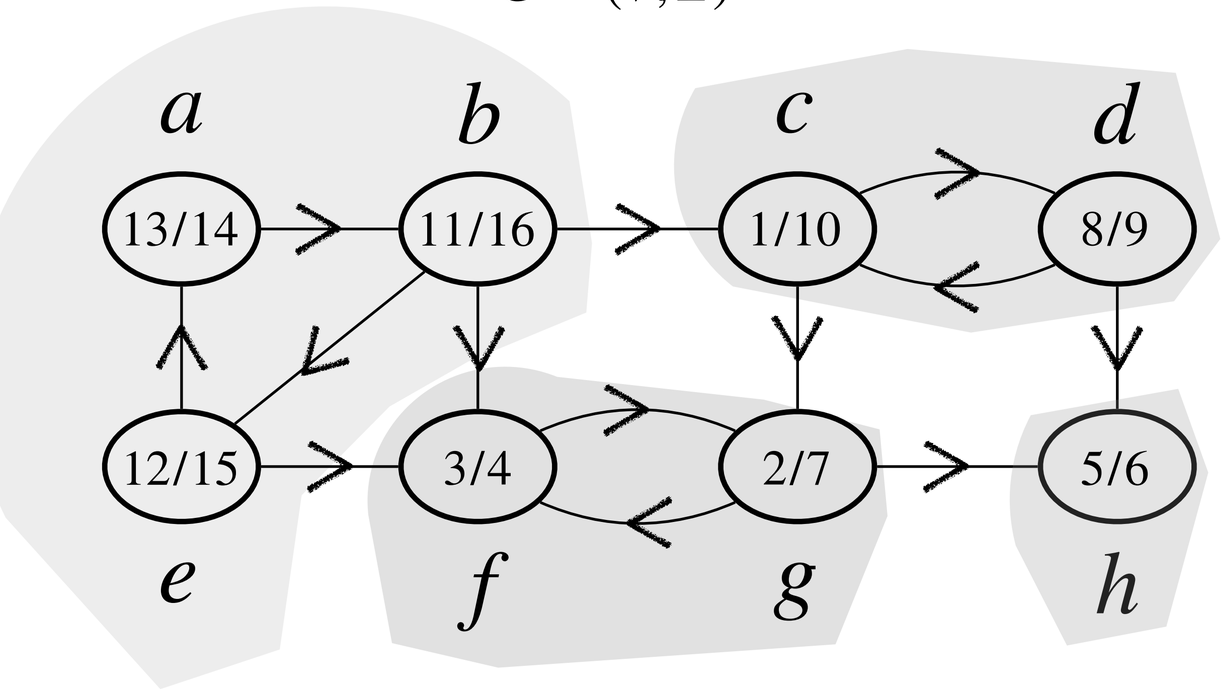
$G^T = (V, E^T)$: transpose of G



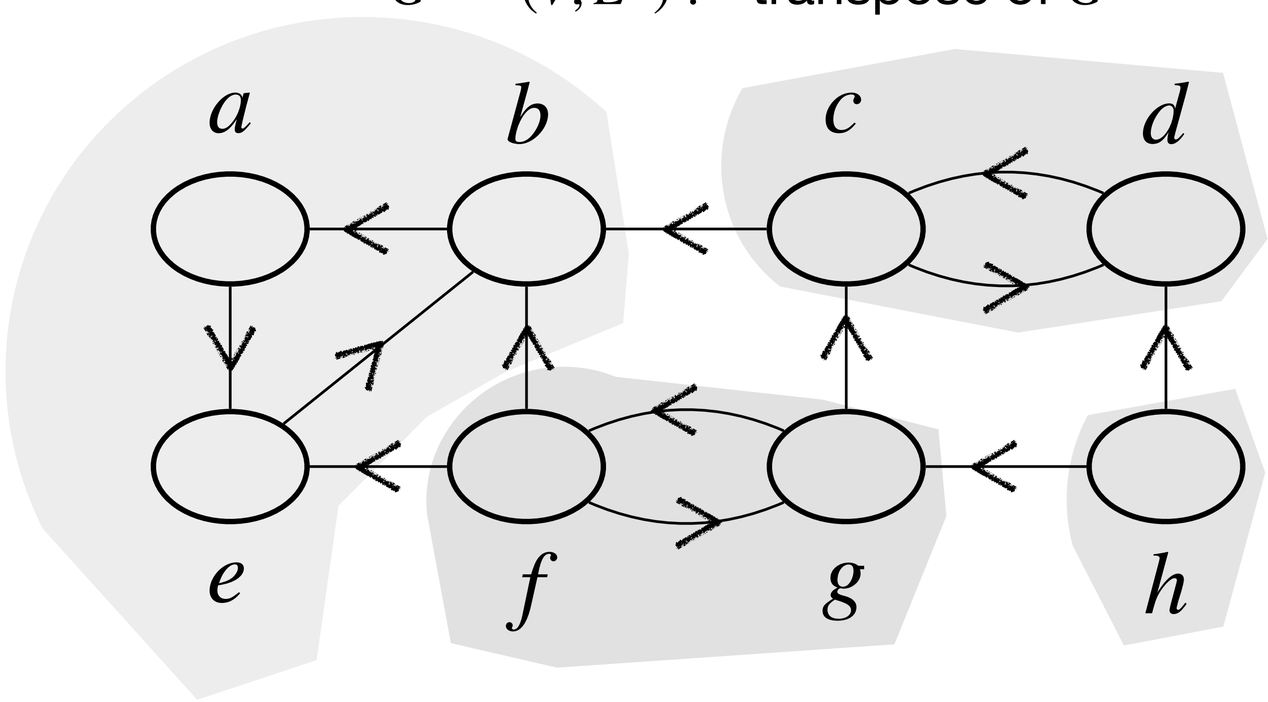


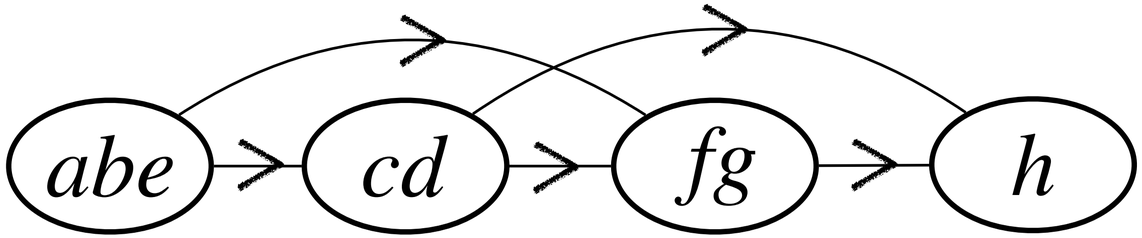


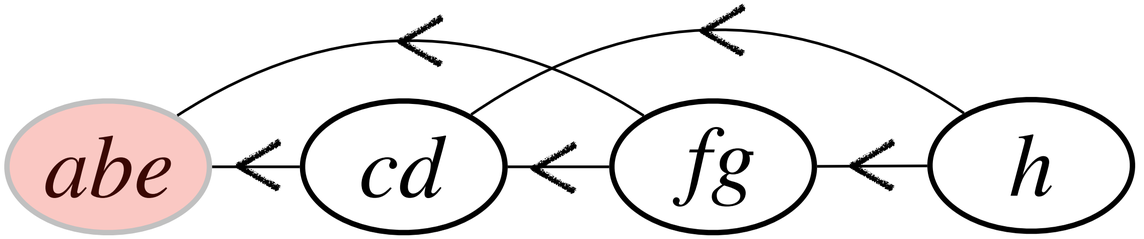
$G = (V, E)$



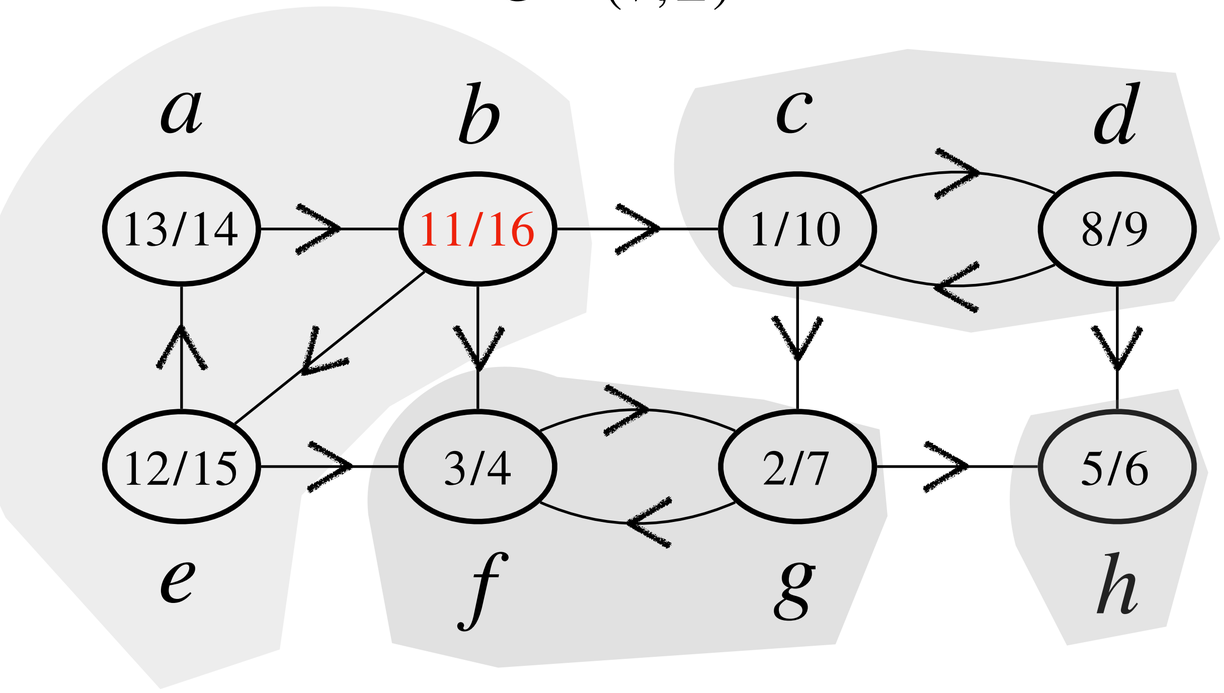
$G^T = (V, E^T)$: transpose of G



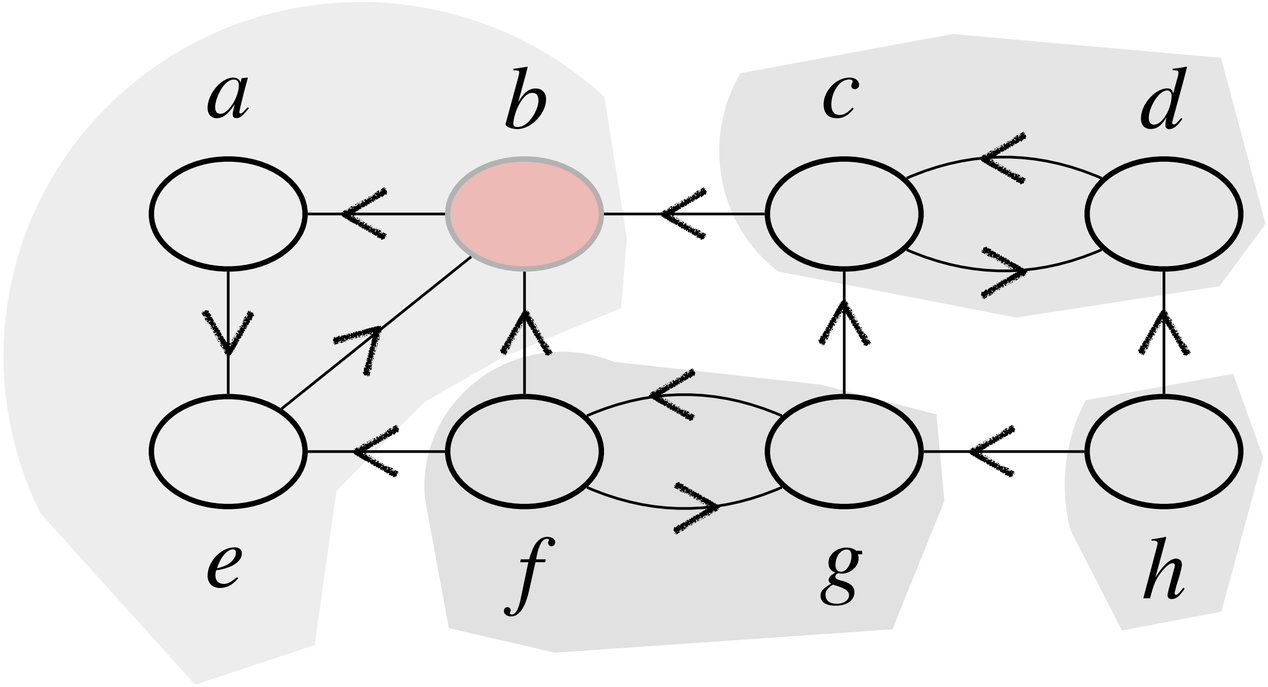


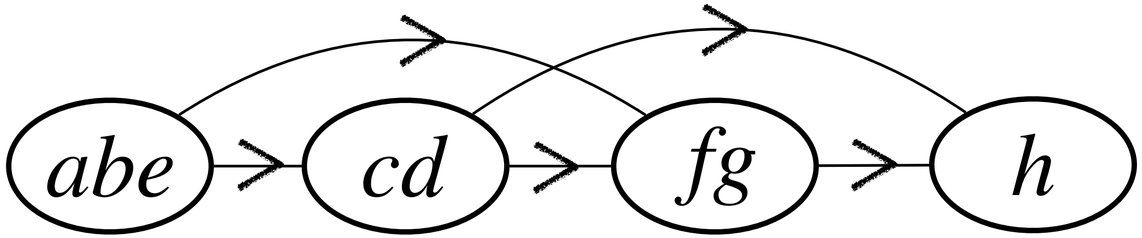


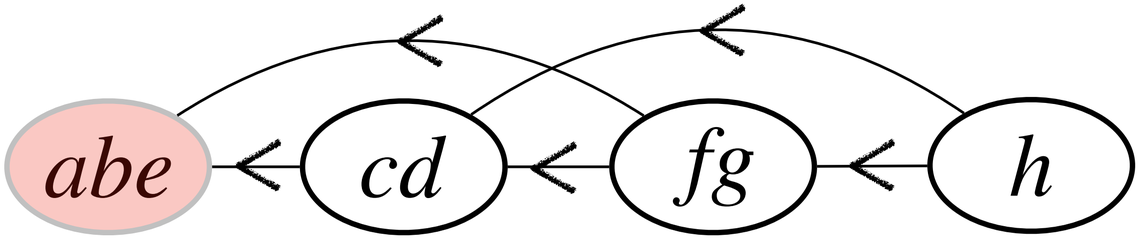
$G = (V, E)$



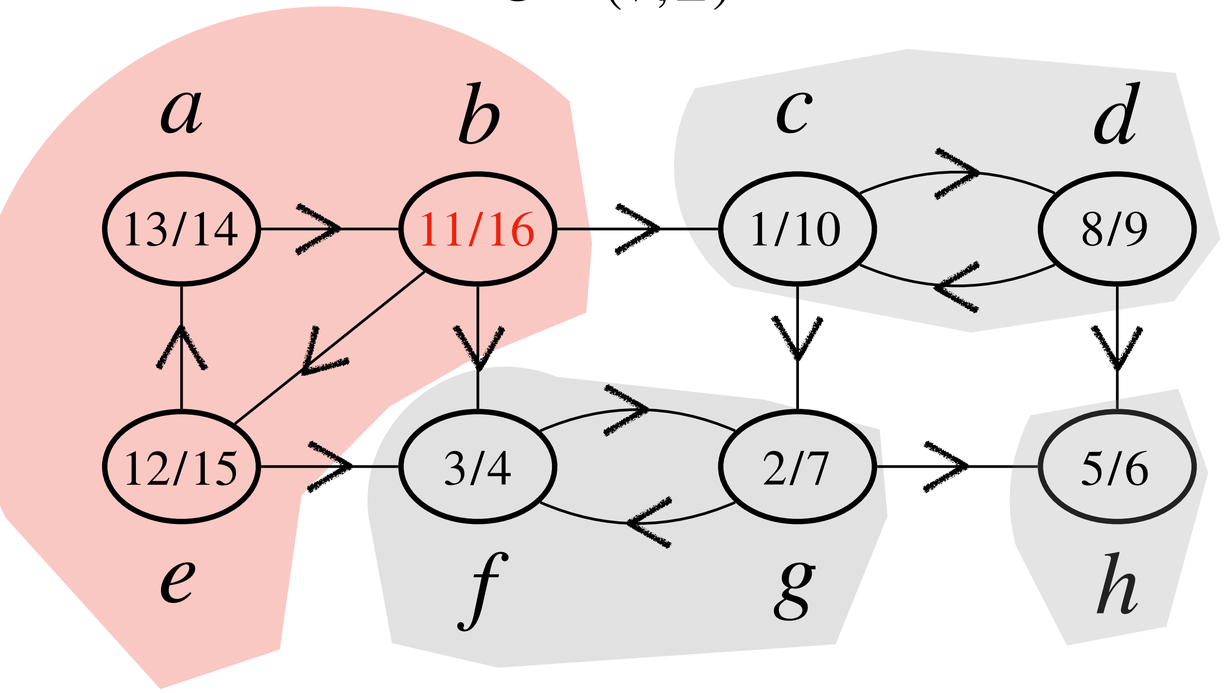
$G^T = (V, E^T)$: transpose of G



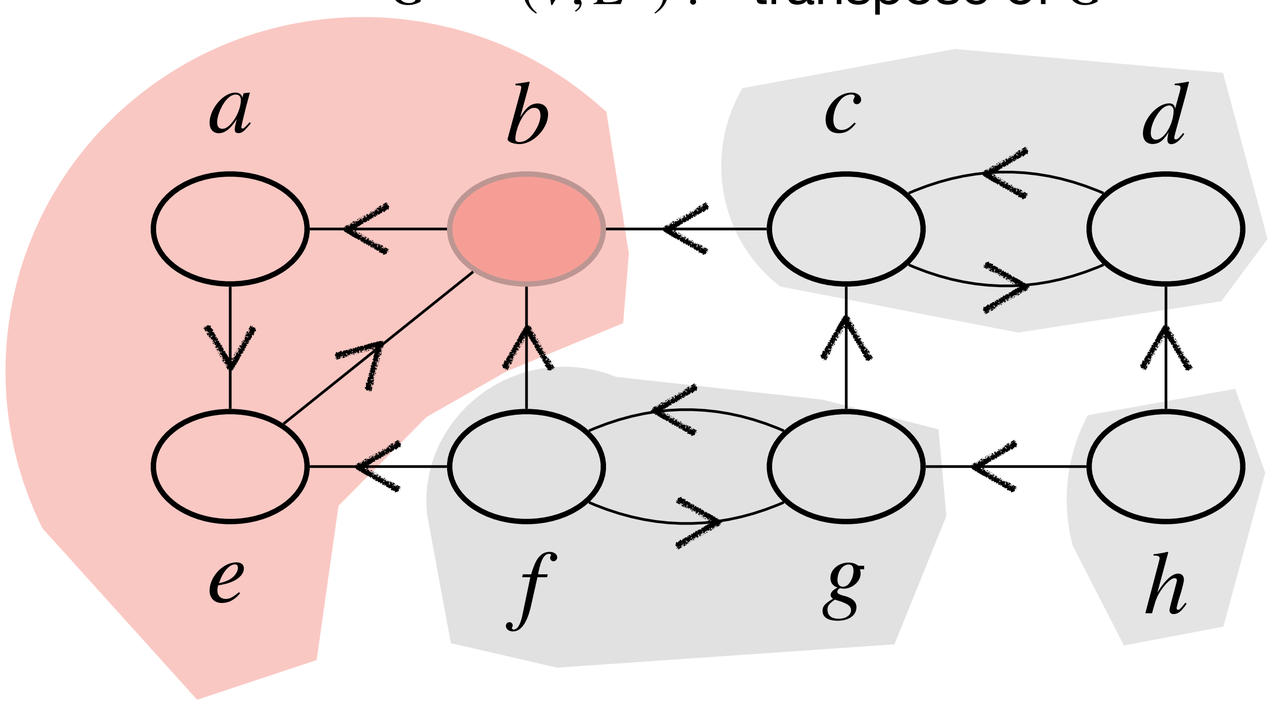


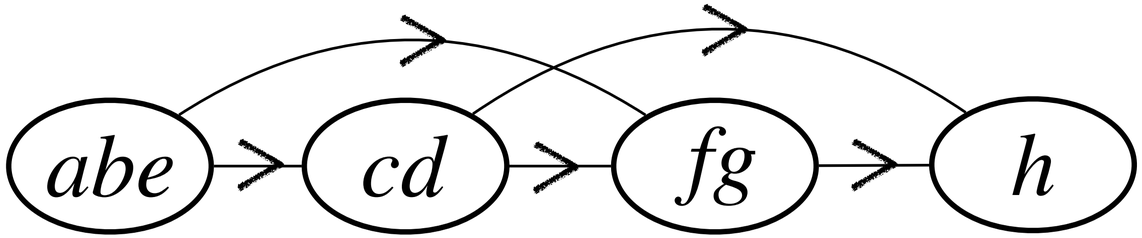


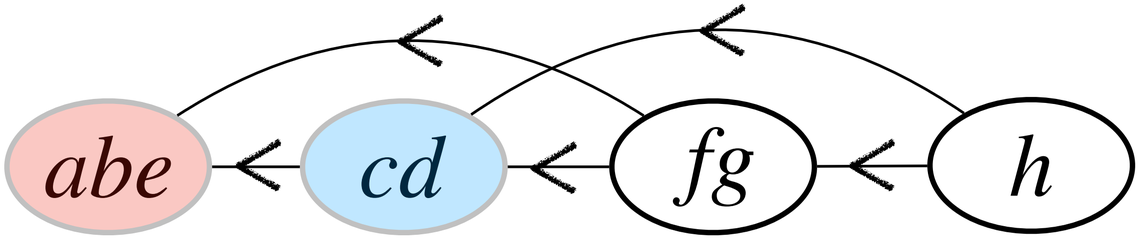
$G = (V, E)$



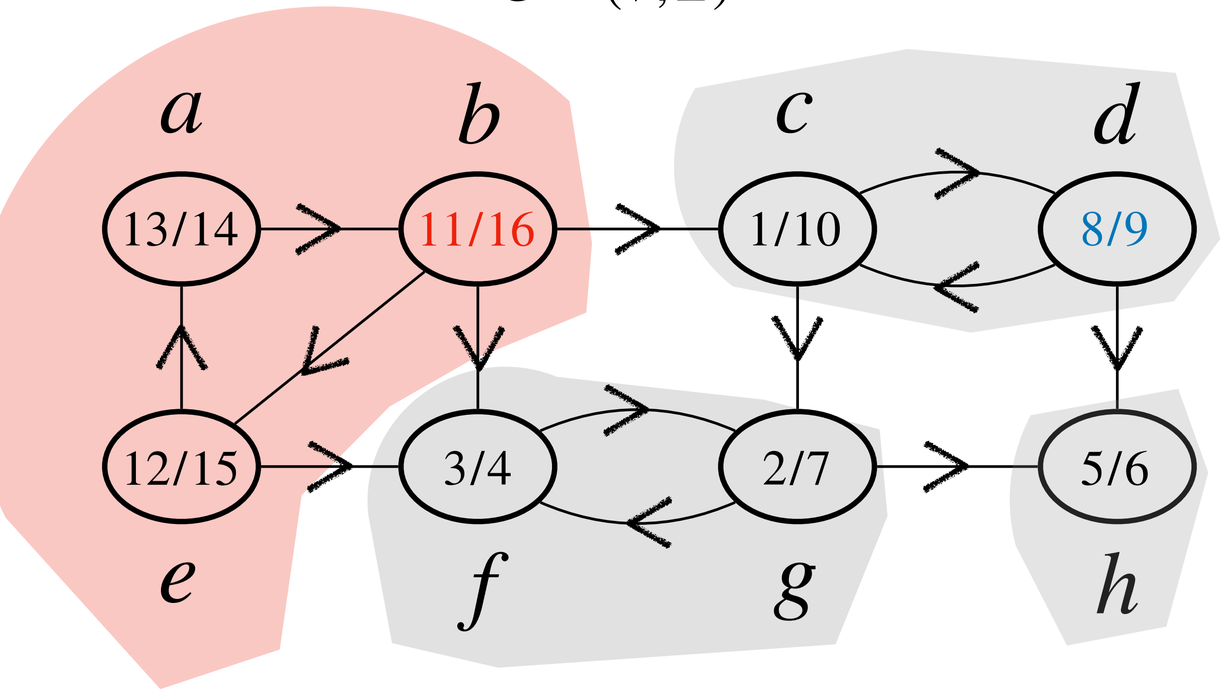
$G^T = (V, E^T)$: transpose of G



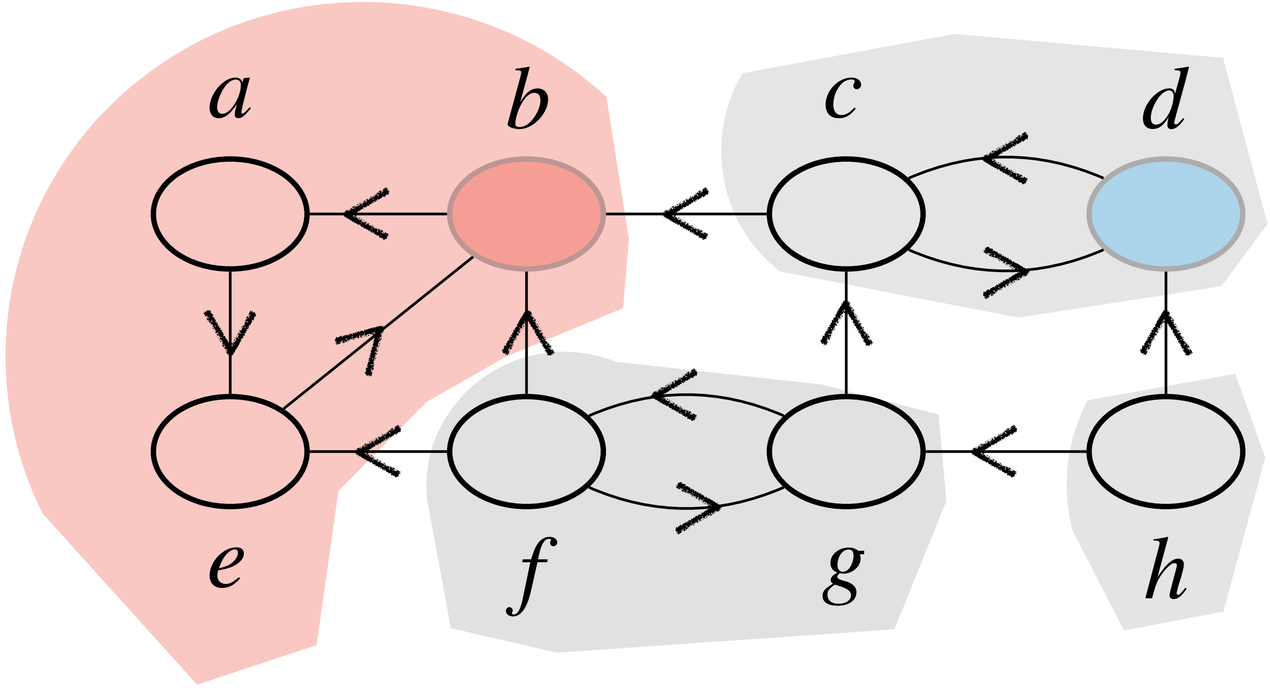


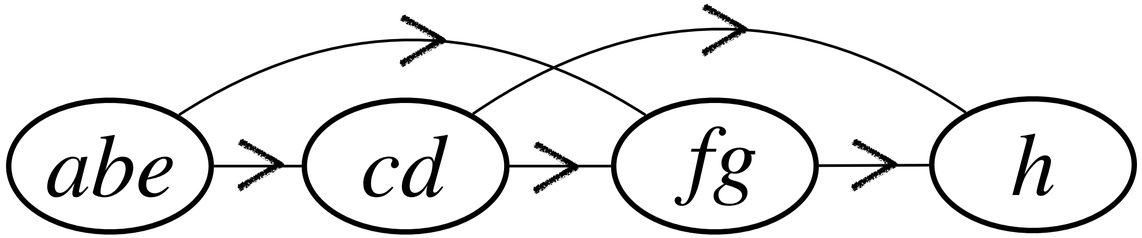


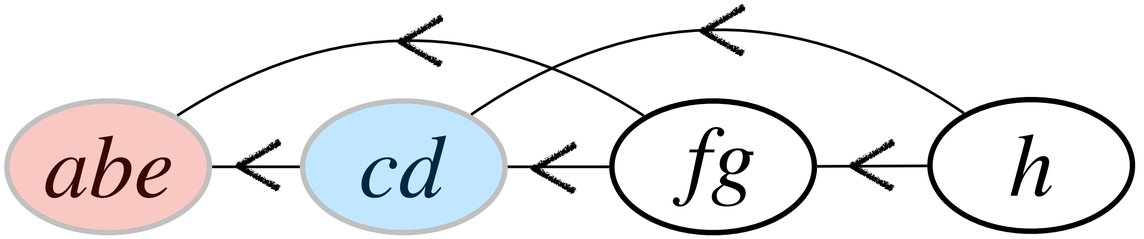
$G = (V, E)$



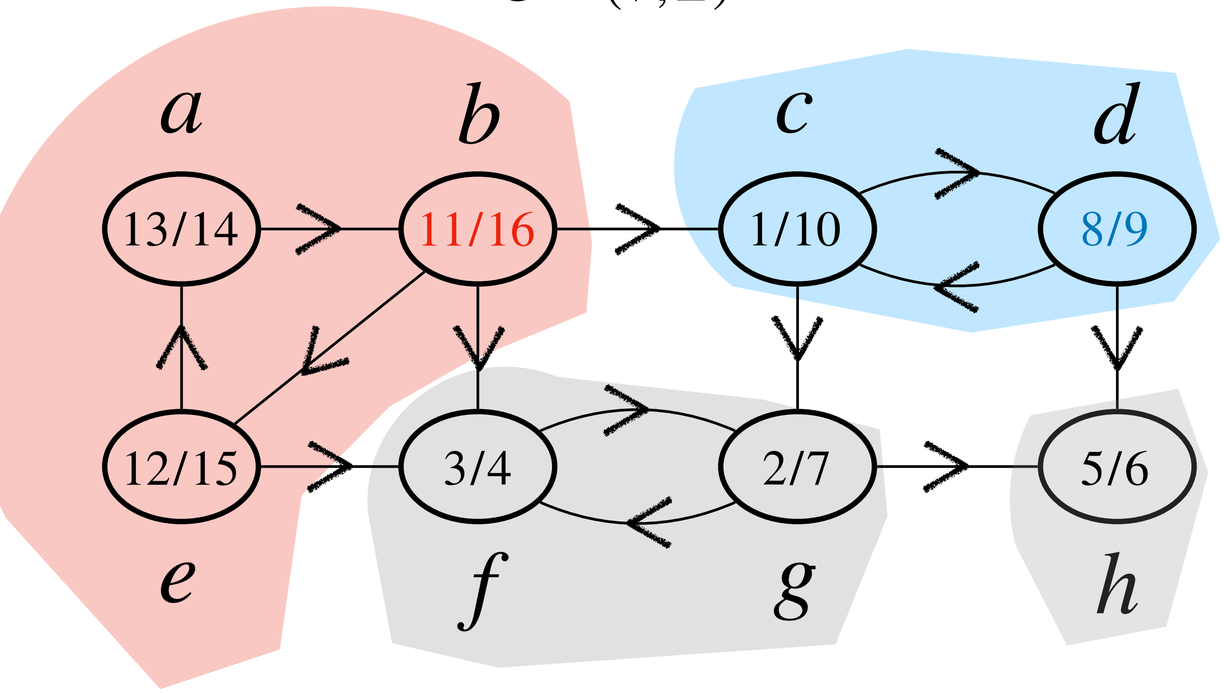
$G^T = (V, E^T)$: transpose of G



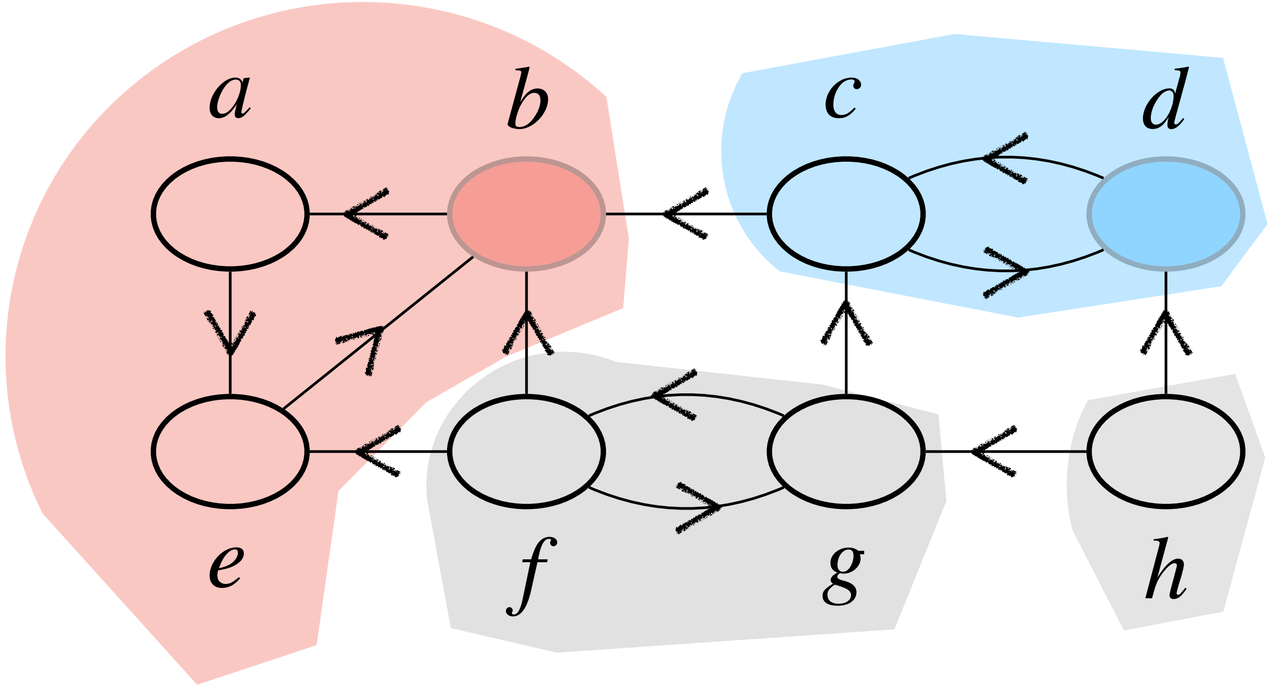


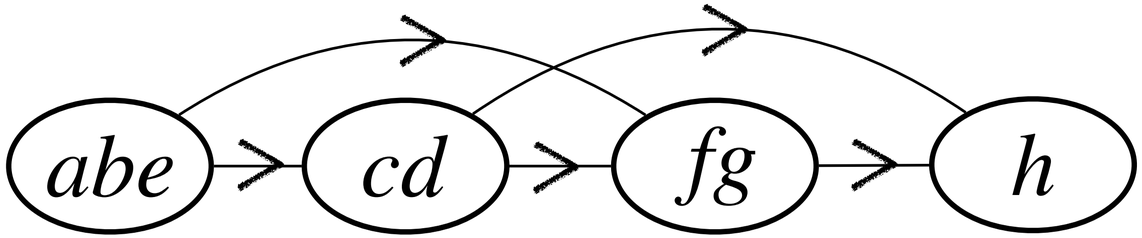


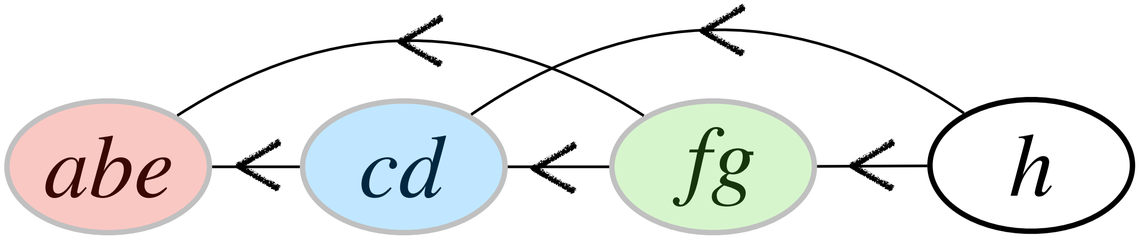
$G = (V, E)$



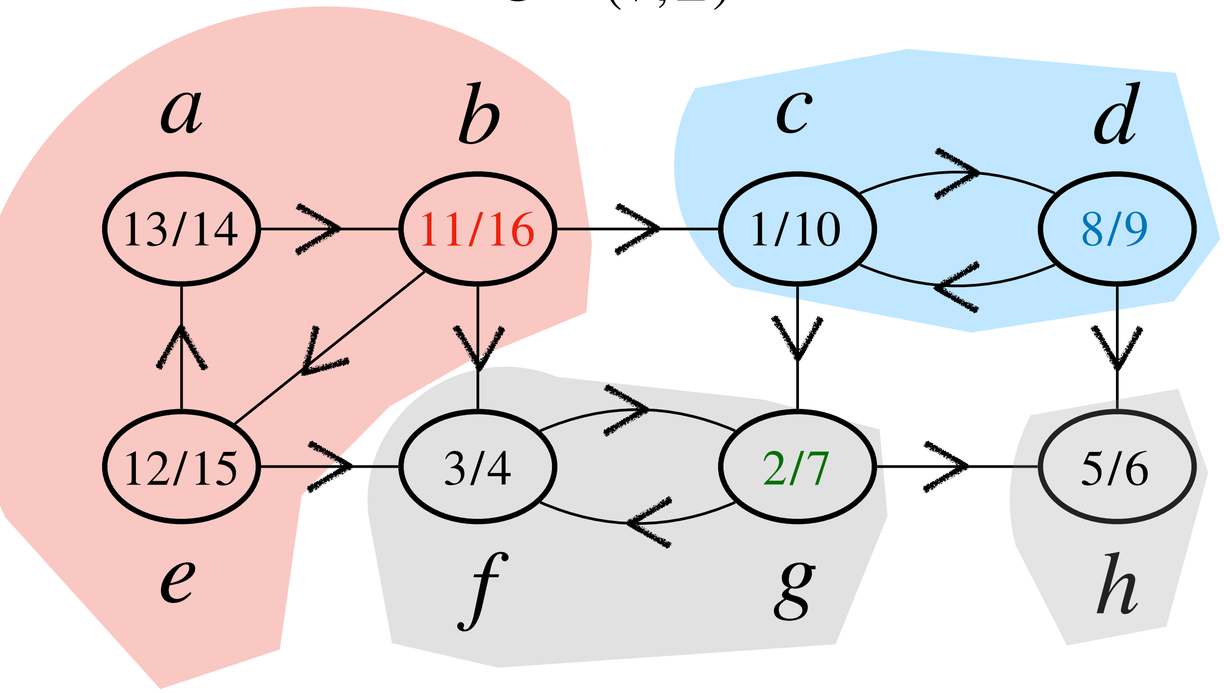
$G^T = (V, E^T)$: transpose of G



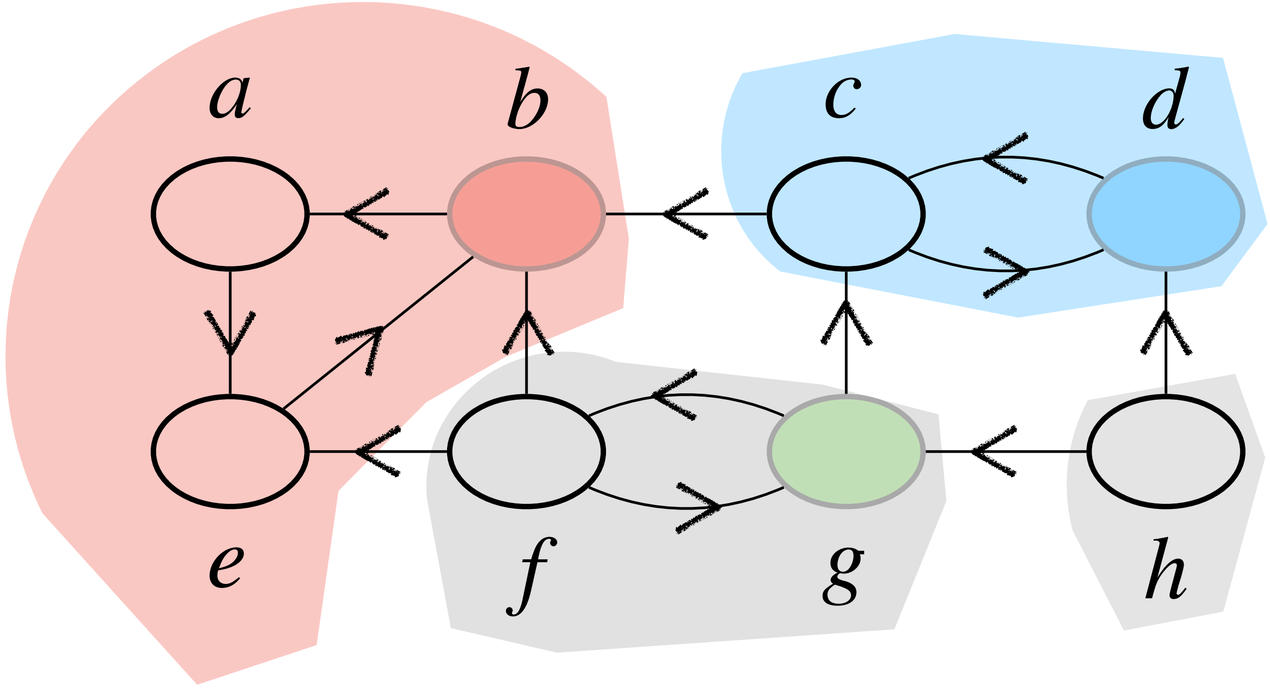


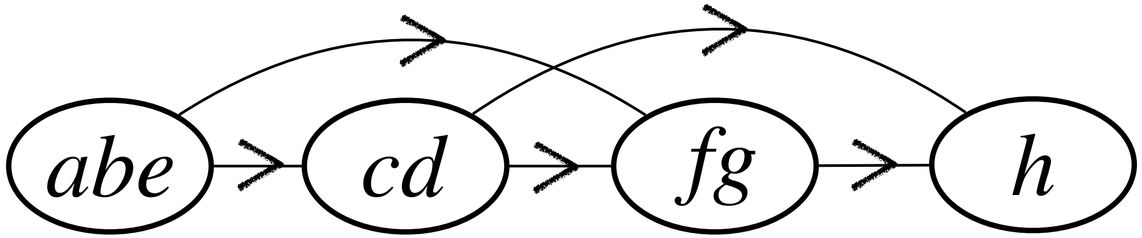


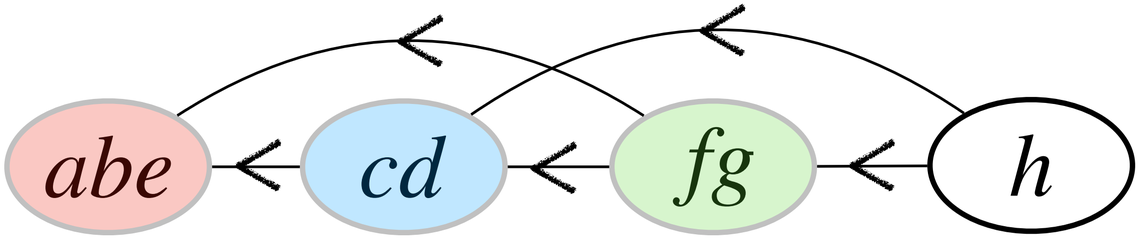
$G = (V, E)$



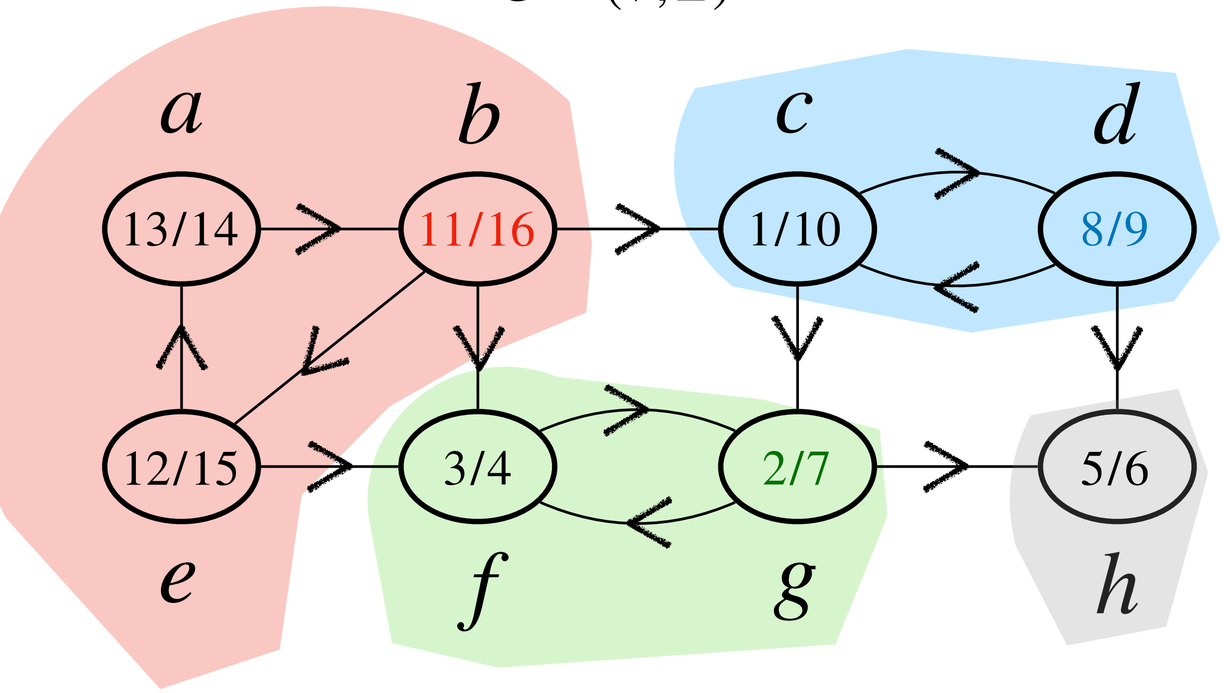
$G^T = (V, E^T)$: transpose of G



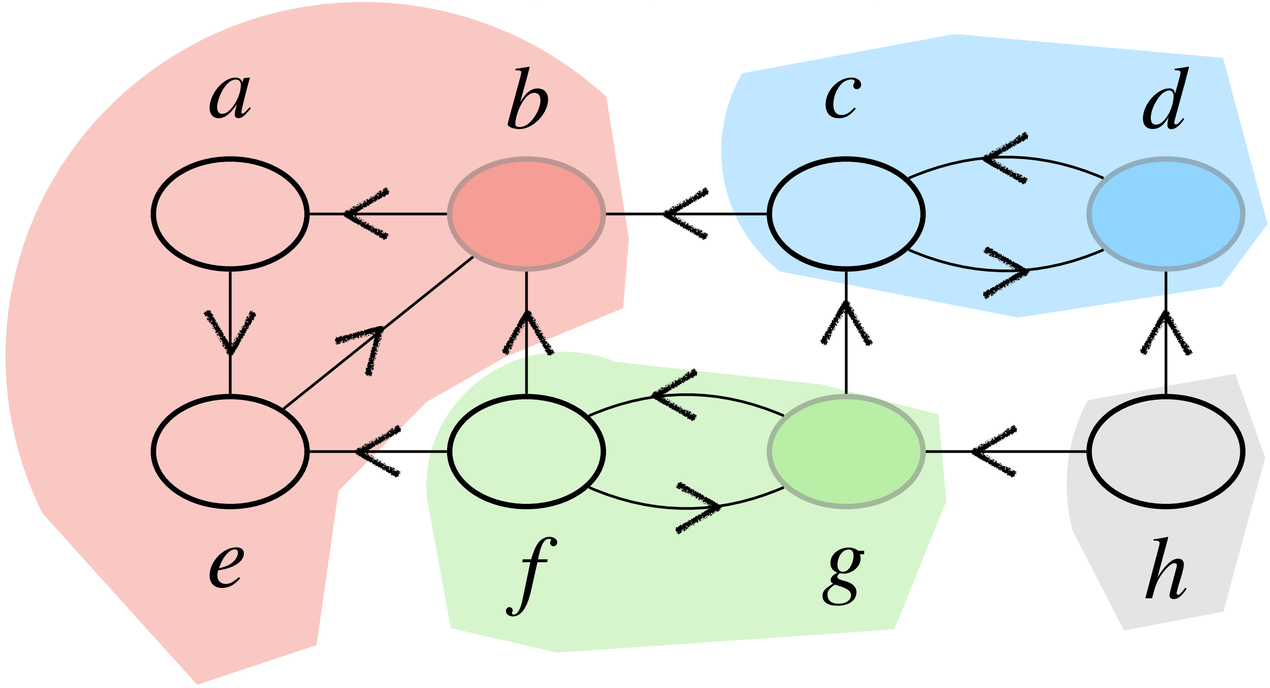


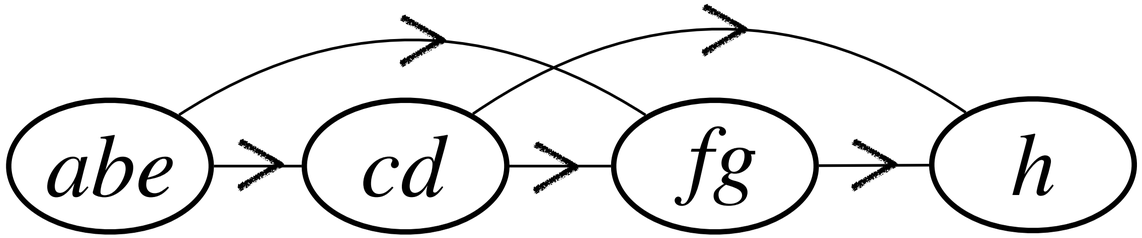


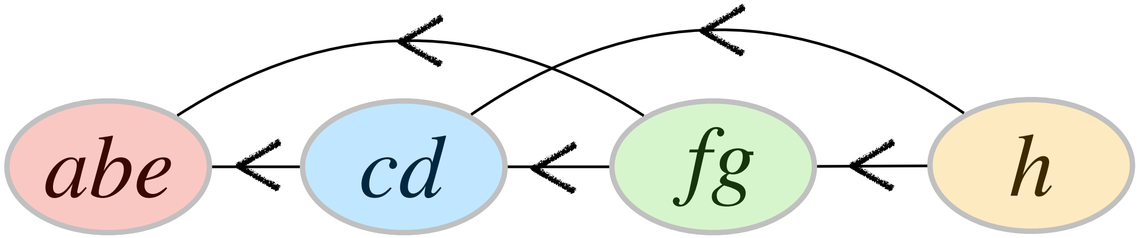
$G = (V, E)$



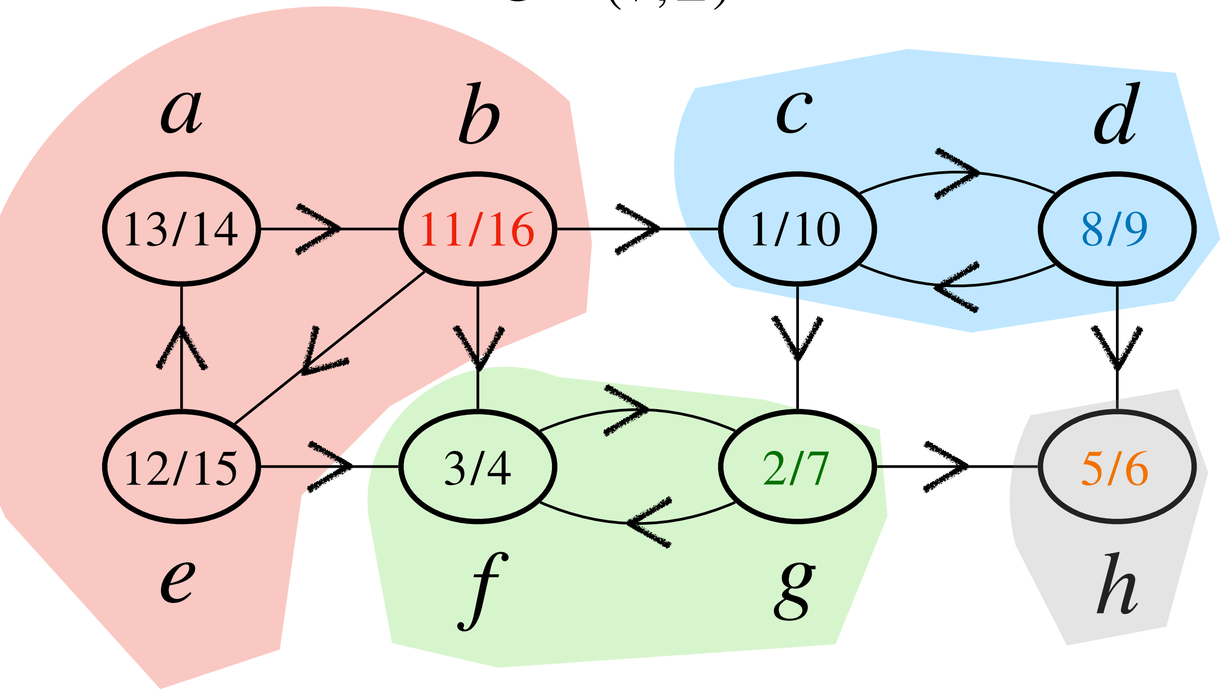
$G^T = (V, E^T)$: transpose of G



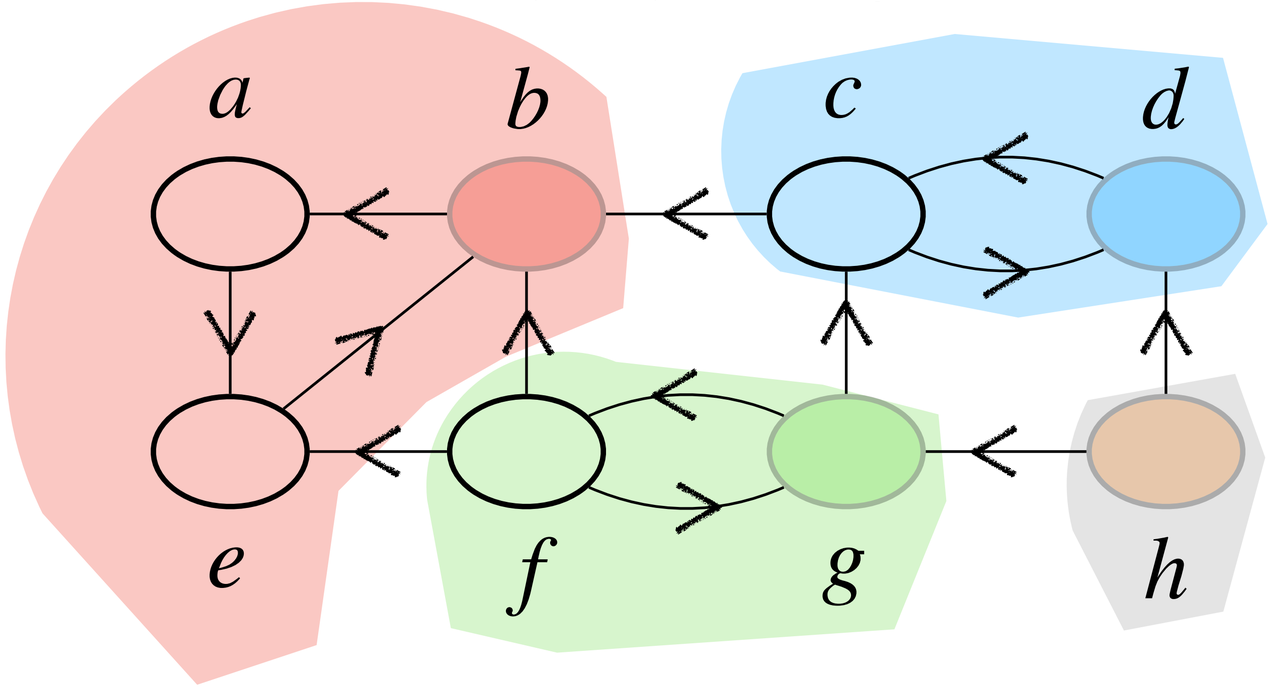


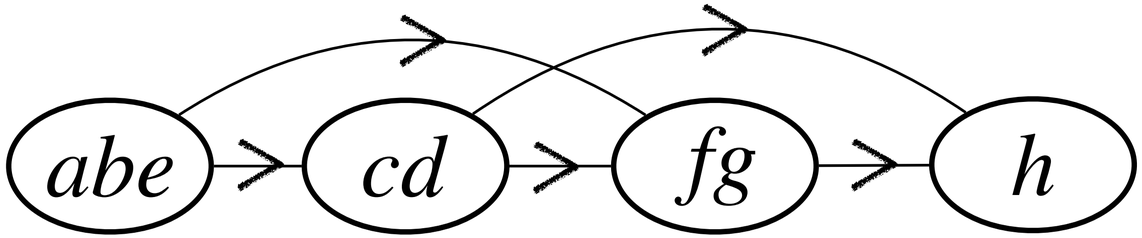


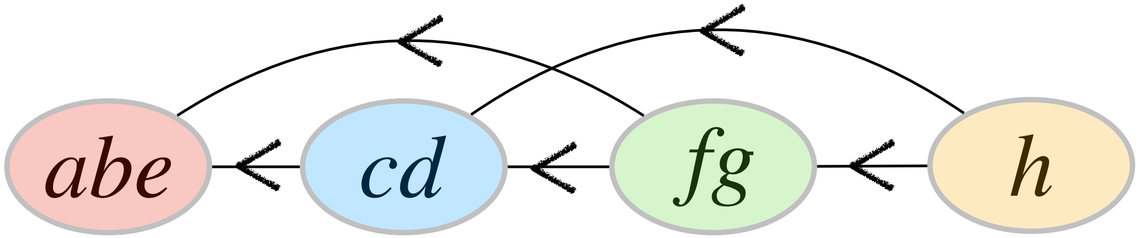
$G = (V, E)$



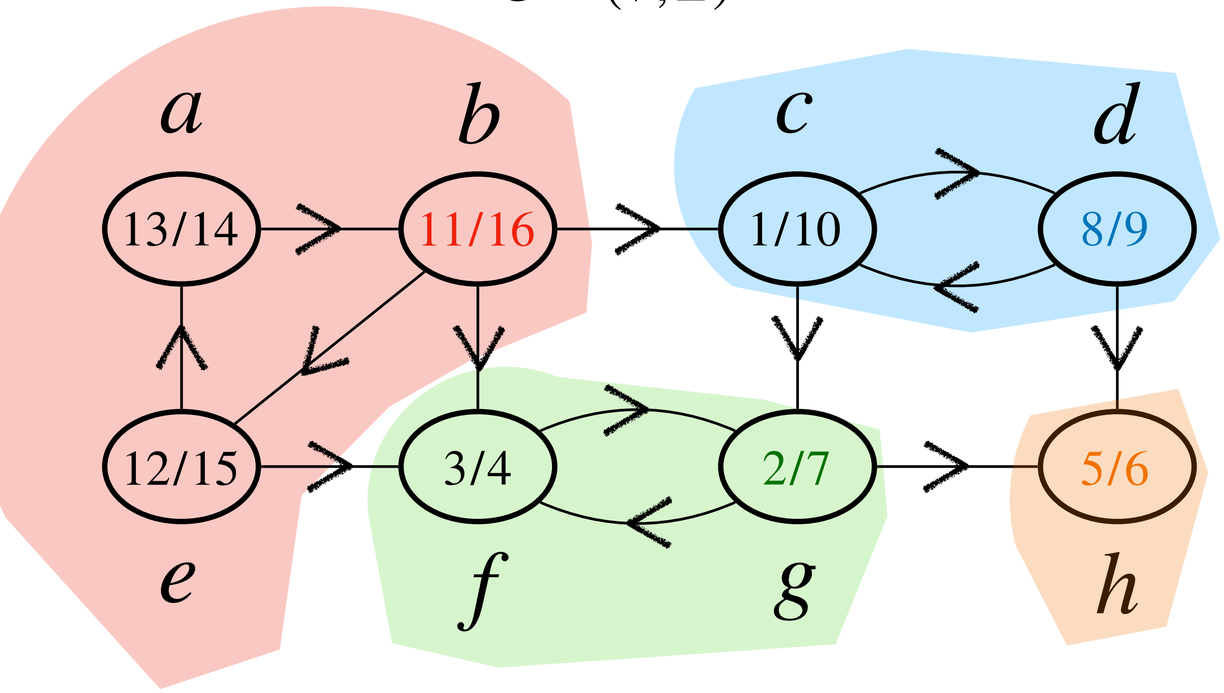
$G^T = (V, E^T)$: transpose of G



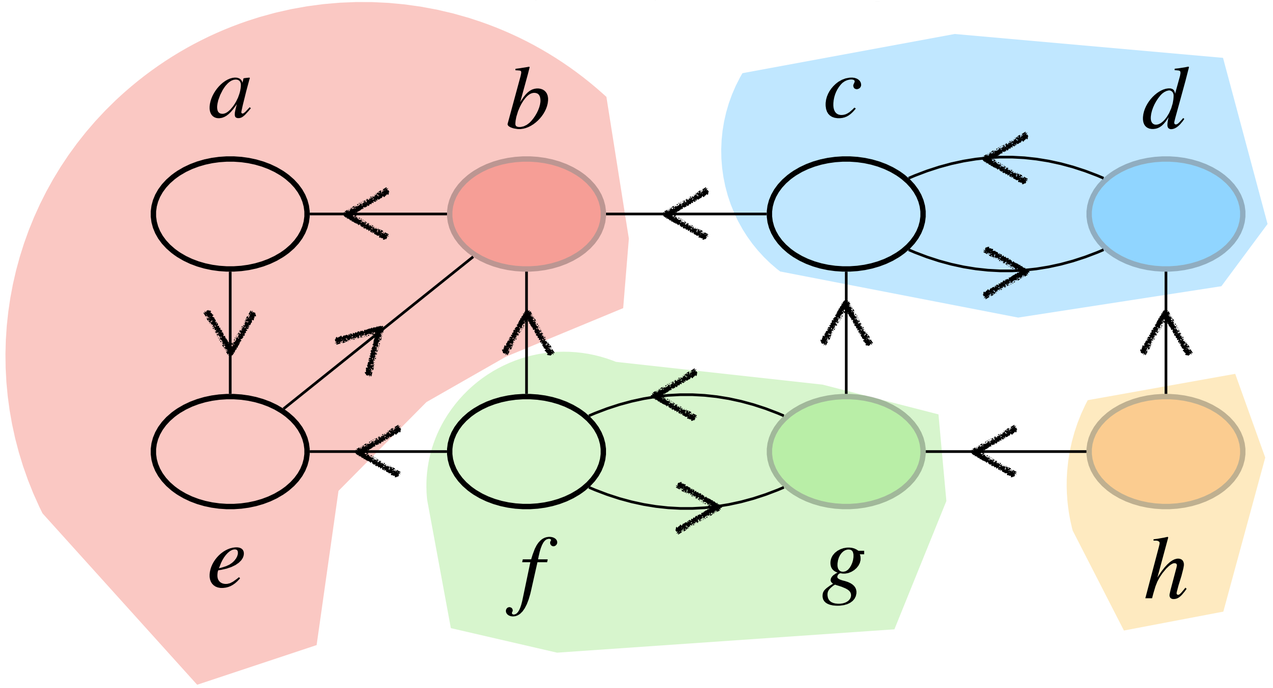




$G = (V, E)$



$G^T = (V, E^T)$: transpose of G



Quiz questions:

1. For two strongly connected components, what is the relationship between their finish times if they are connected by an edge?

2. How is the above property used by the algorithm to find strongly connected components?