Design and Analysis of Algorithms

05-05
Directed Graph

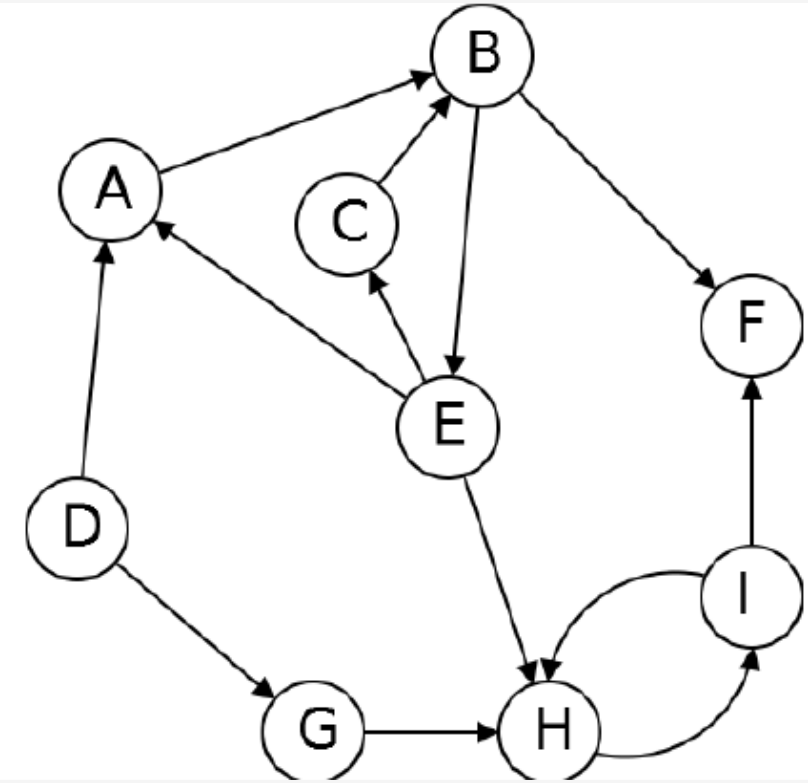
Strongly Connected Components

Imran Ihsan
Assistant Professor, Department of Computer Science
Air University, Islamabad, Pakistan
www.imranihsan.com

Connectivity in Digraphs

In undirected graphs, have connected components.

Directed graphs are more complicated.



Possible Notions

Connected by edges in any direction.

One vertex reachable from another.

Two vertices both reachable from the other.

Strongly Connected Components

Definition

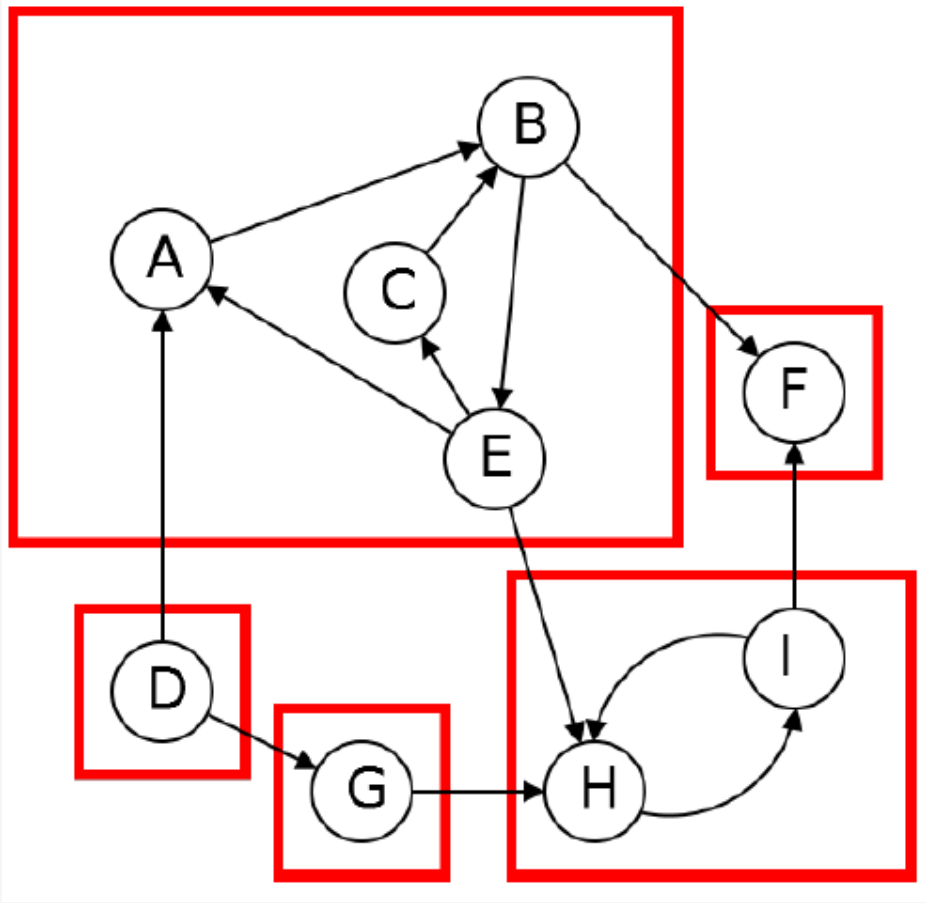
Two vertices v; w in a directed graph are connected if you can reach v from w and can reach w from v.

Result

Theorem

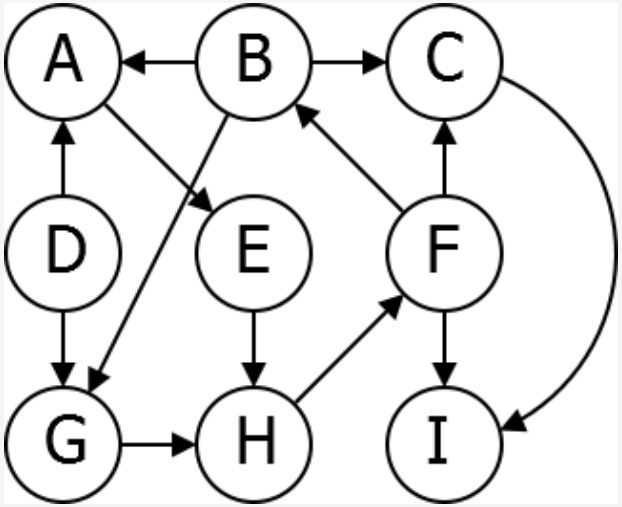
A directed graph can be partitioned into strongly connected components where two vertices are connected if and only if they are in the same component.

Example



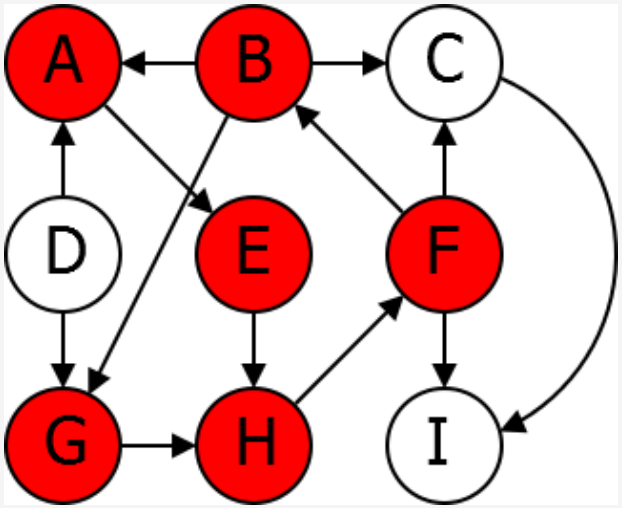
Problem

What is the SCC of A?



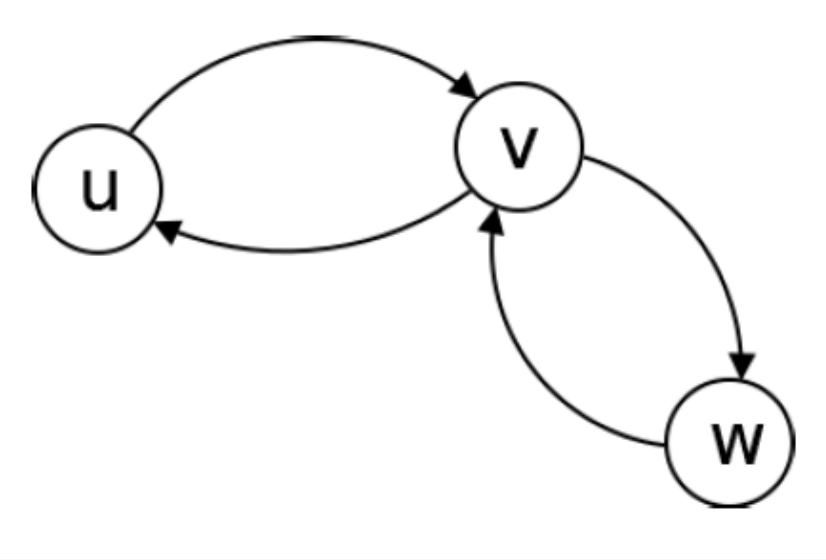
Solution

A, B, E, F, G, H.

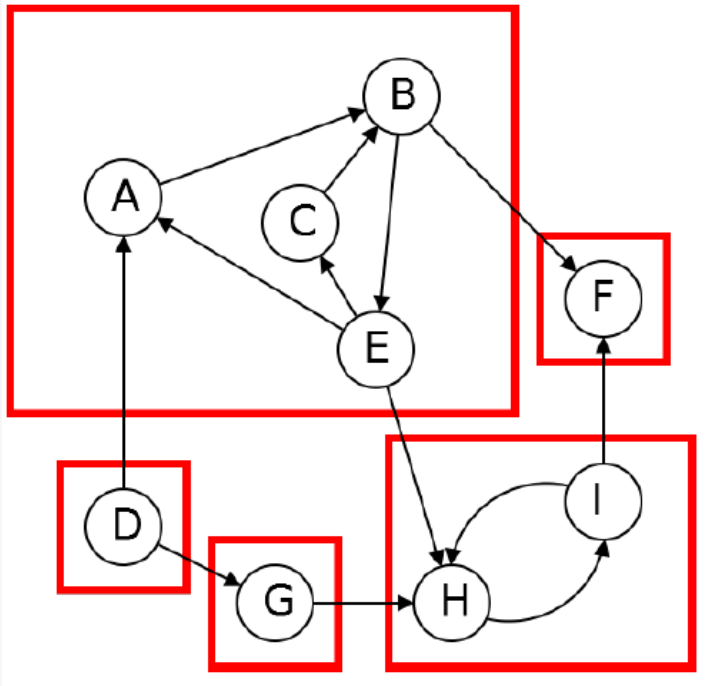


Proof

Need to show an equivalence relation.

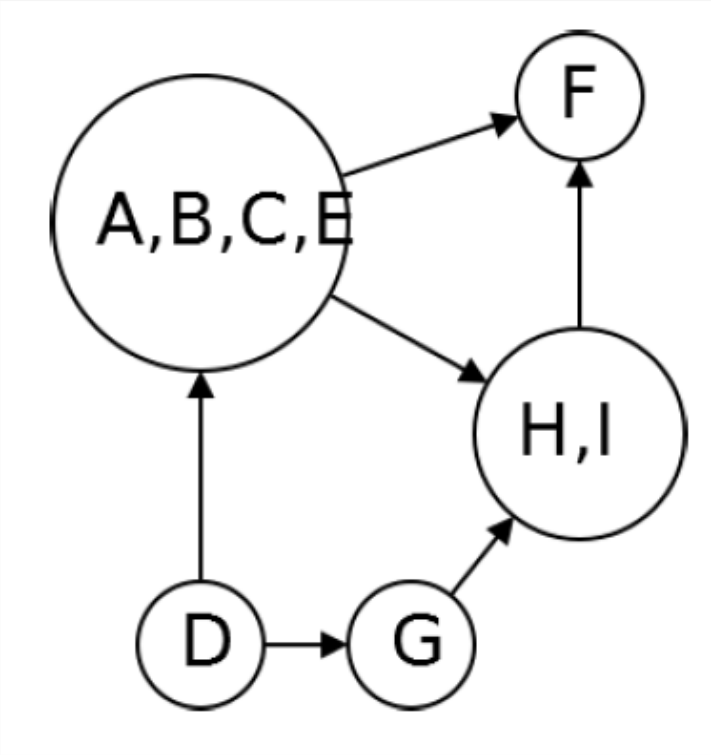


Metagraph



We can also draw a metagraph showing how the strongly connected components connect to one another.

Example



Note: It's a DAG.

DAG

Theorem

The metagraph of a graph G is always a DAG.

Proof

Suppose it's not a DAG.

Must be a cycle C.

Any nodes in cycle can reach any others.

Should all be in same SCC.

Contradiction.

Summary

Can partition vertices into strongly connected components.

Metagraph describes how strongly connected components connect to each other.

Metagraph always a DAG.

Computing SCC

Strongly connected component

Strongly Connected Components

Problem

Input: A directed graph G

Output: The SCC of G.

Easy Algorithm

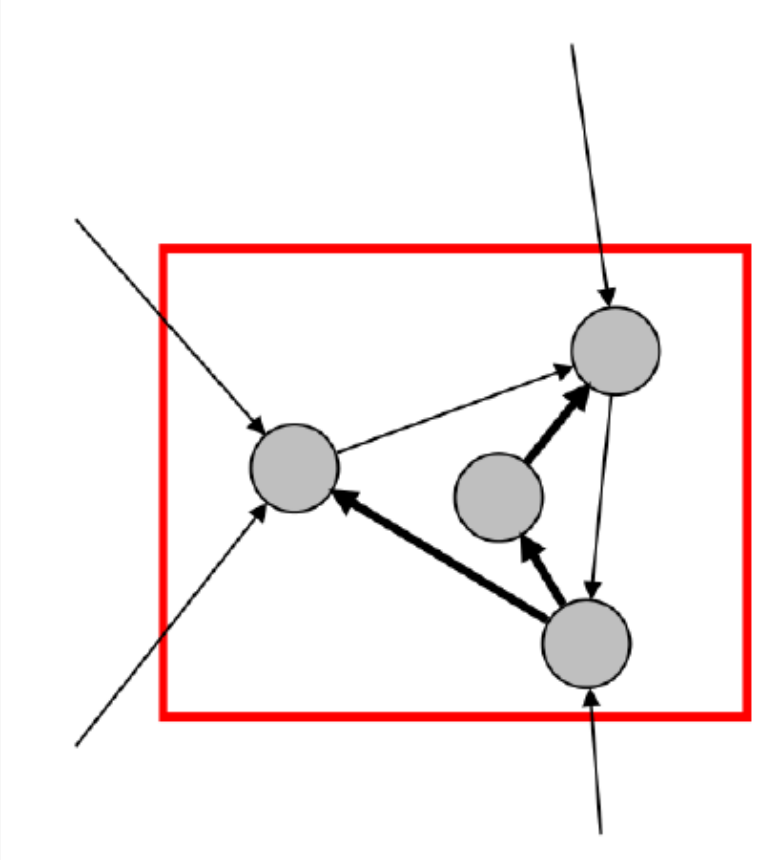
```
EasySCC(G)
   for each vertex v:
      run explore(v) to determine
         vertices reachable from v
   for each vertex v:
      find the u reachable from v that
         can also reach v
   these are the SCCs

//Runtime O(|V|² + |V||E|). Want faster.
```

Sink Components

Idea:

If v is in a sink SCC, explore(v) finds vertices reachable from v.

This is exactly the SCC of v.



Finding Sink Components

Need a way to find a sink SCC.

Theorem

Theorem

If C and C’ are two strongly connected components with an edge from some vertex of C to some vertex of C’, then largest post in C is bigger than largest post in C’.

Proof

Cases:

Visit C before visit C’

Visit C’ before visit C

Case – I

Visit C first.

Can reach everything in C' from C.

Explore all of C' while exploring C.

C has largest post.

Case – II

Visit C’ first.

Cannot reach C from C’

Must finish exploring C’ before exploring C

C has largest post.

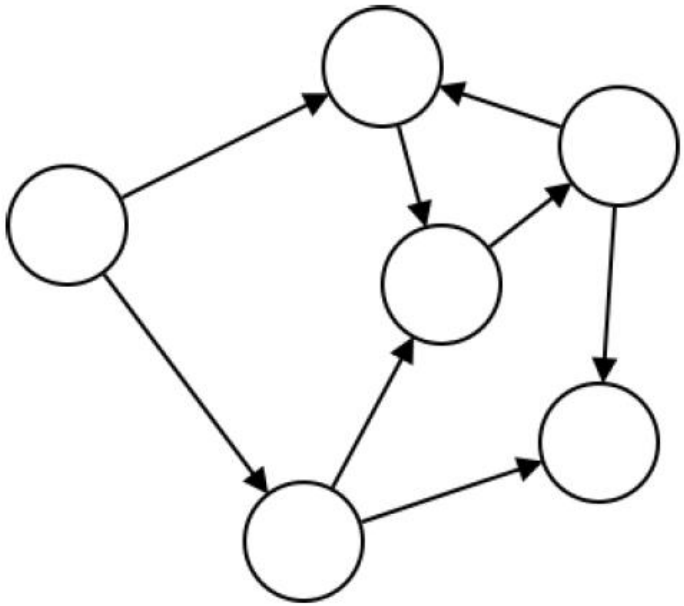
Conclusion

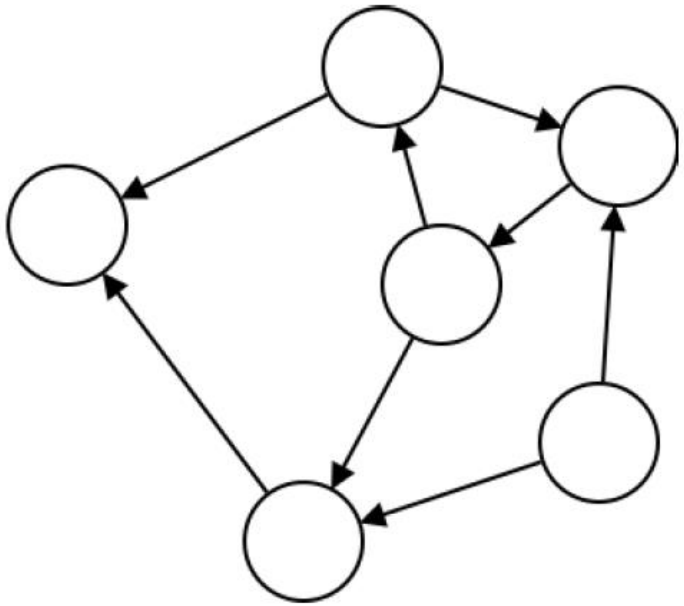
The vertex with the largest postorder number is in a source component!

Problem: We wanted a sink component.

Reverse graph

Let G^R be the graph obtained from G by reversing all of the edges.





Reverse graph Components

G^R and G have same SCCs.

Source components of G^R are sink components of G.

Find sink components of G by running DFS on G^R.

Problem

Which of the following is true?

The vertex with largest postorder in G^R is in a sink SCC of G.

The vertex with the largest preorder in G is in a sink SCC of G.

The vertex with the smallest postorder in G is in a sink SCC of G.

Solution

Which of the following is true?

The vertex with largest postorder in G^R is in a sink SCC of G.

The vertex with the largest preorder in G is in a sink SCC of G.

The vertex with the smallest postorder in G is in a sink SCC of G.

BASIC Algorithm

SCCc(G)

run DFS(G^R)

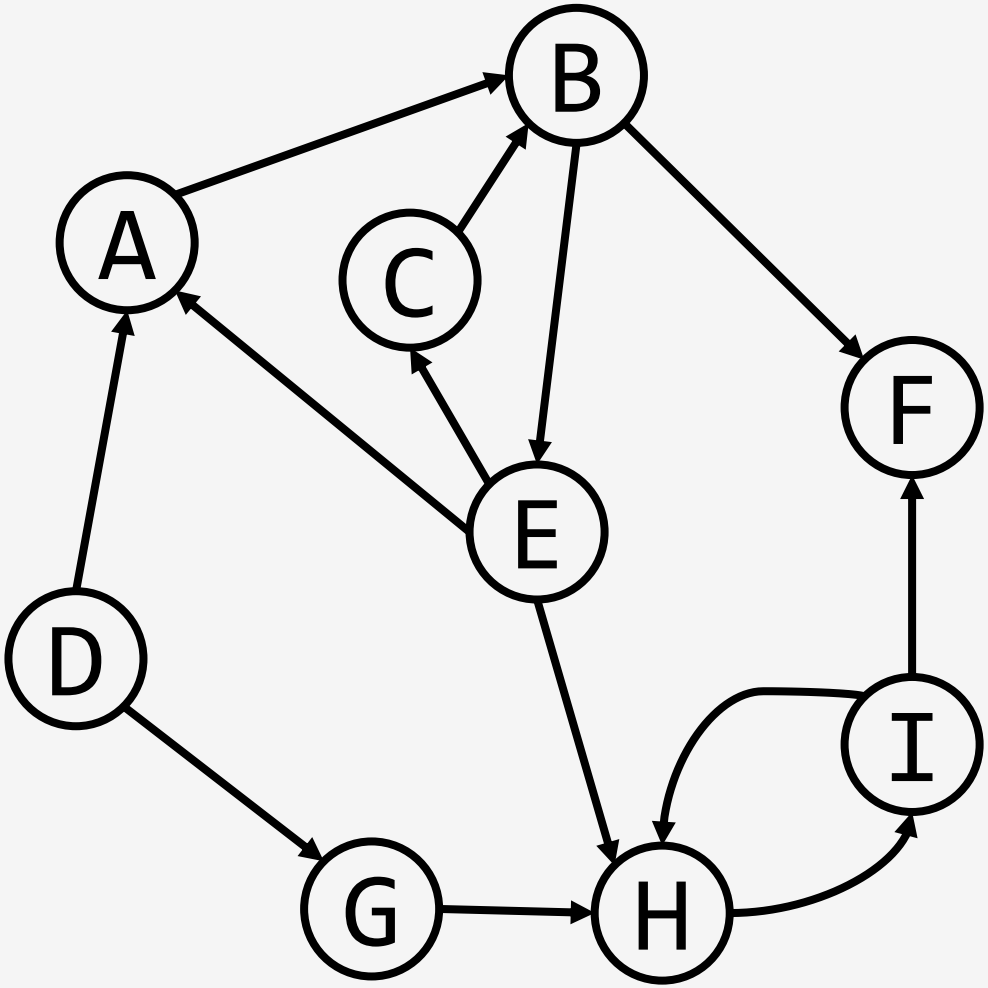
let v have largest post number

run Explore(v)

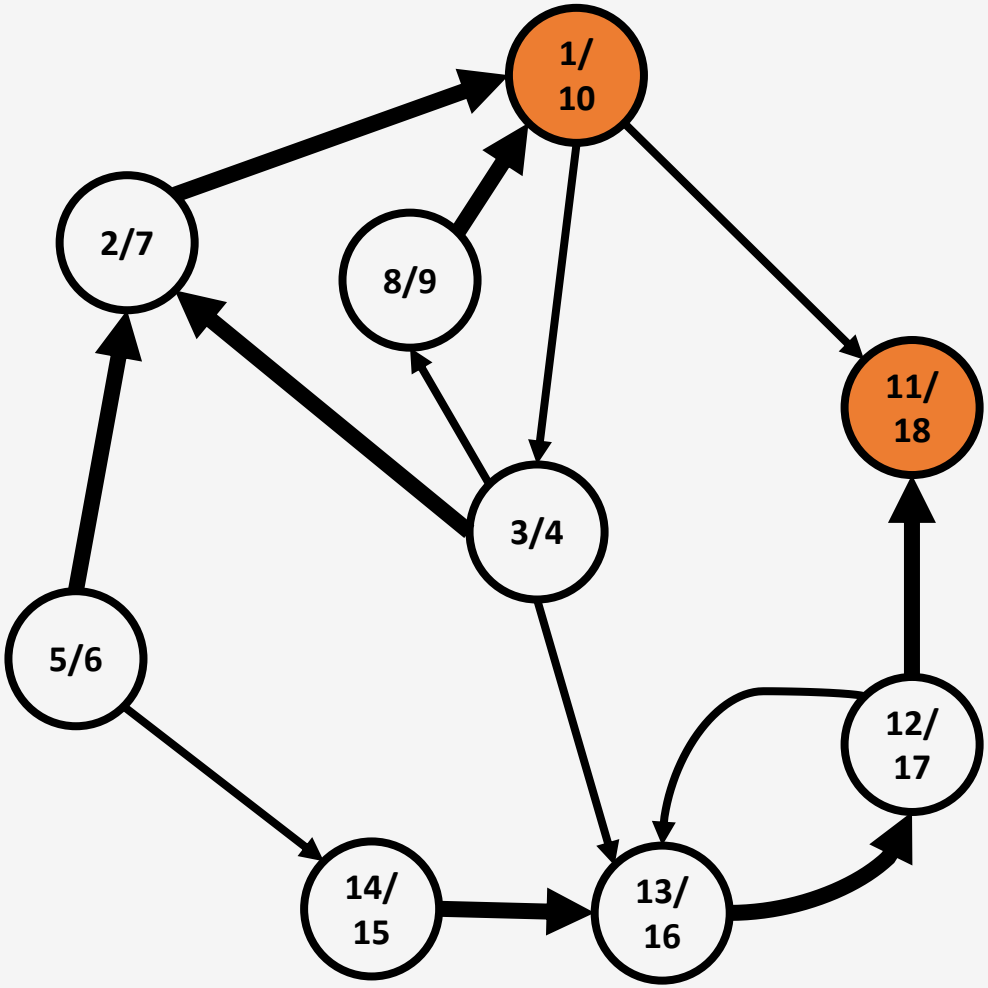
vertices found are first SCC

Remove from G and repeat.

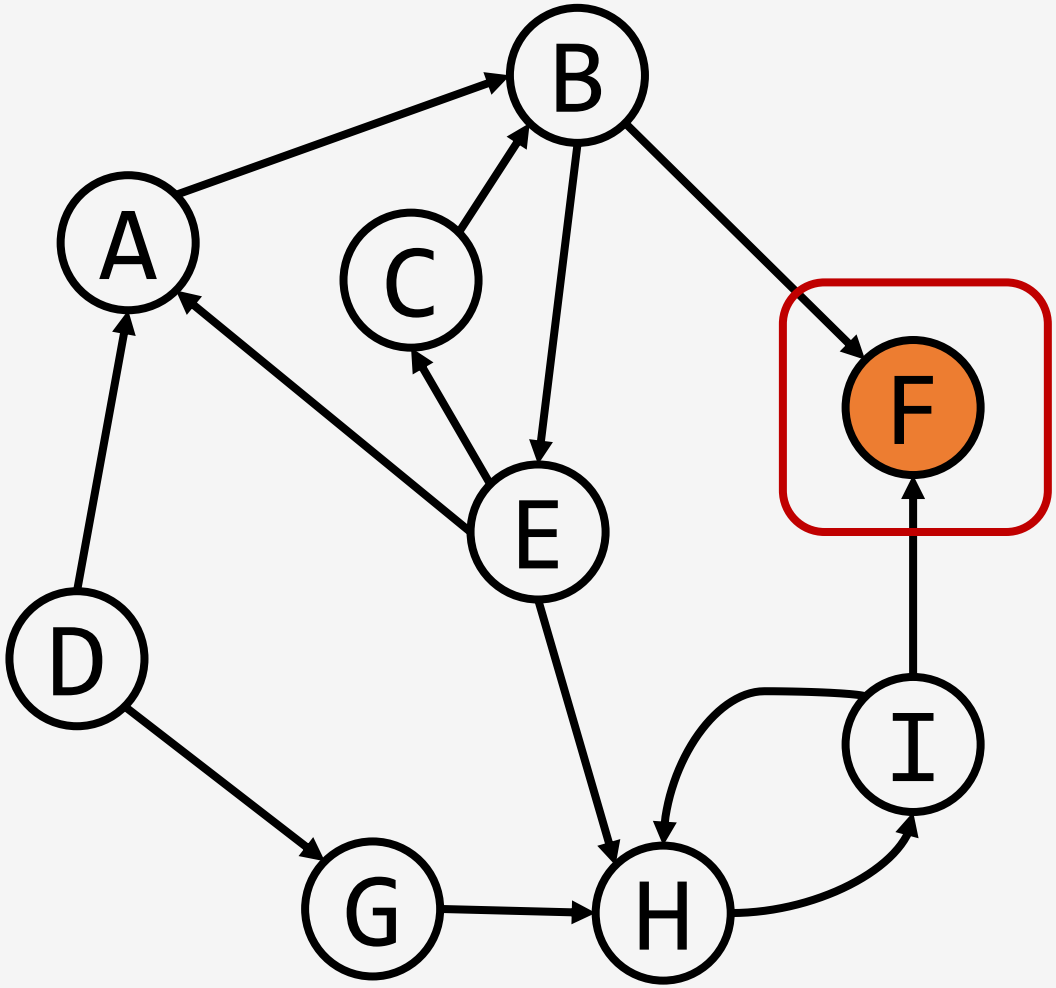
EXAMPLE



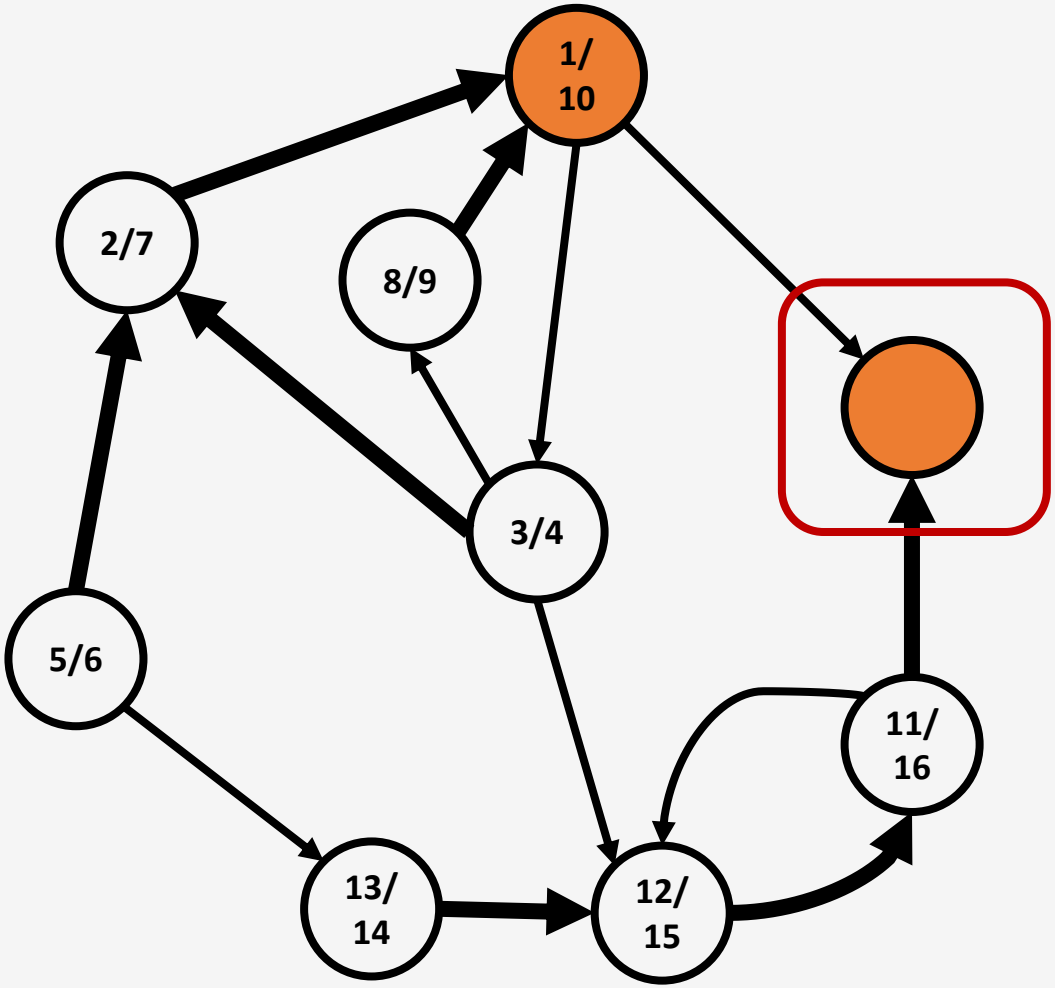
EXAMPLE



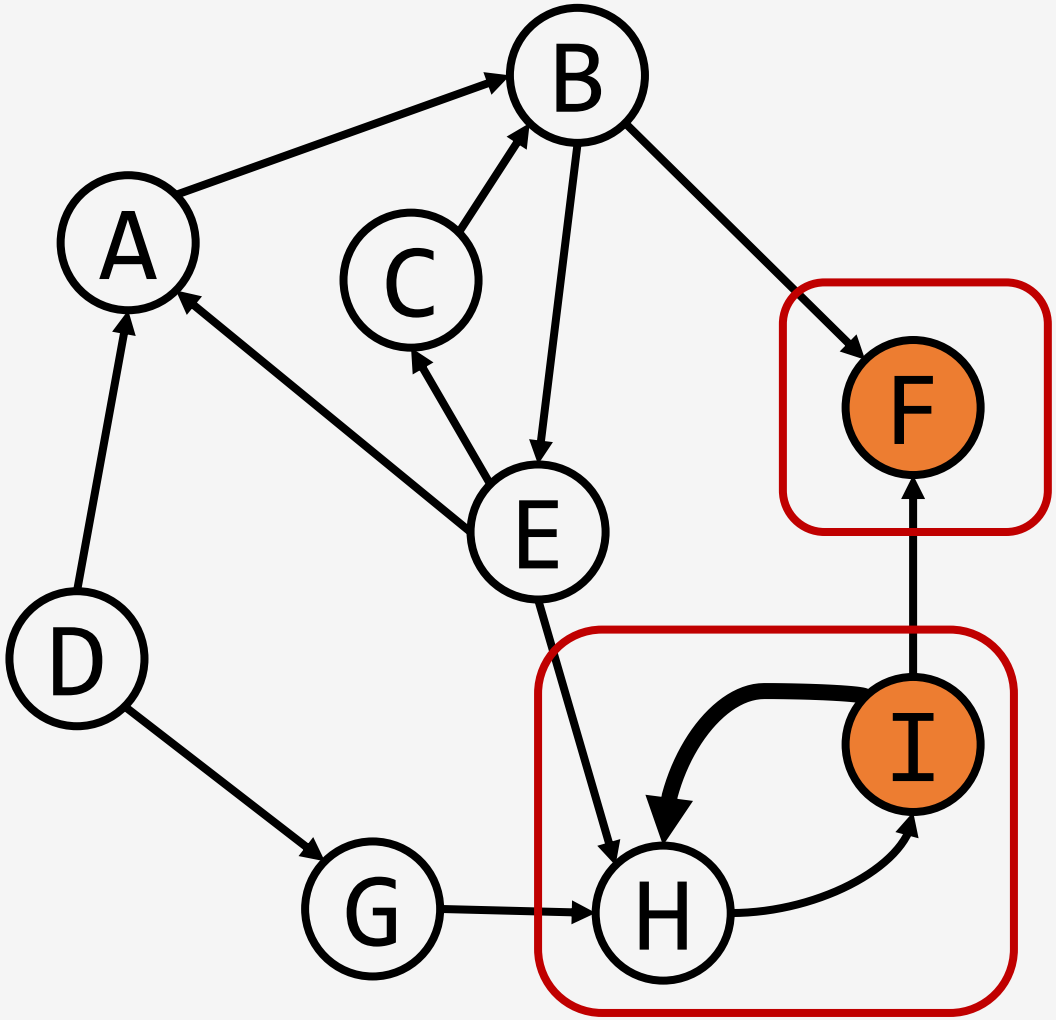
EXAMPLE



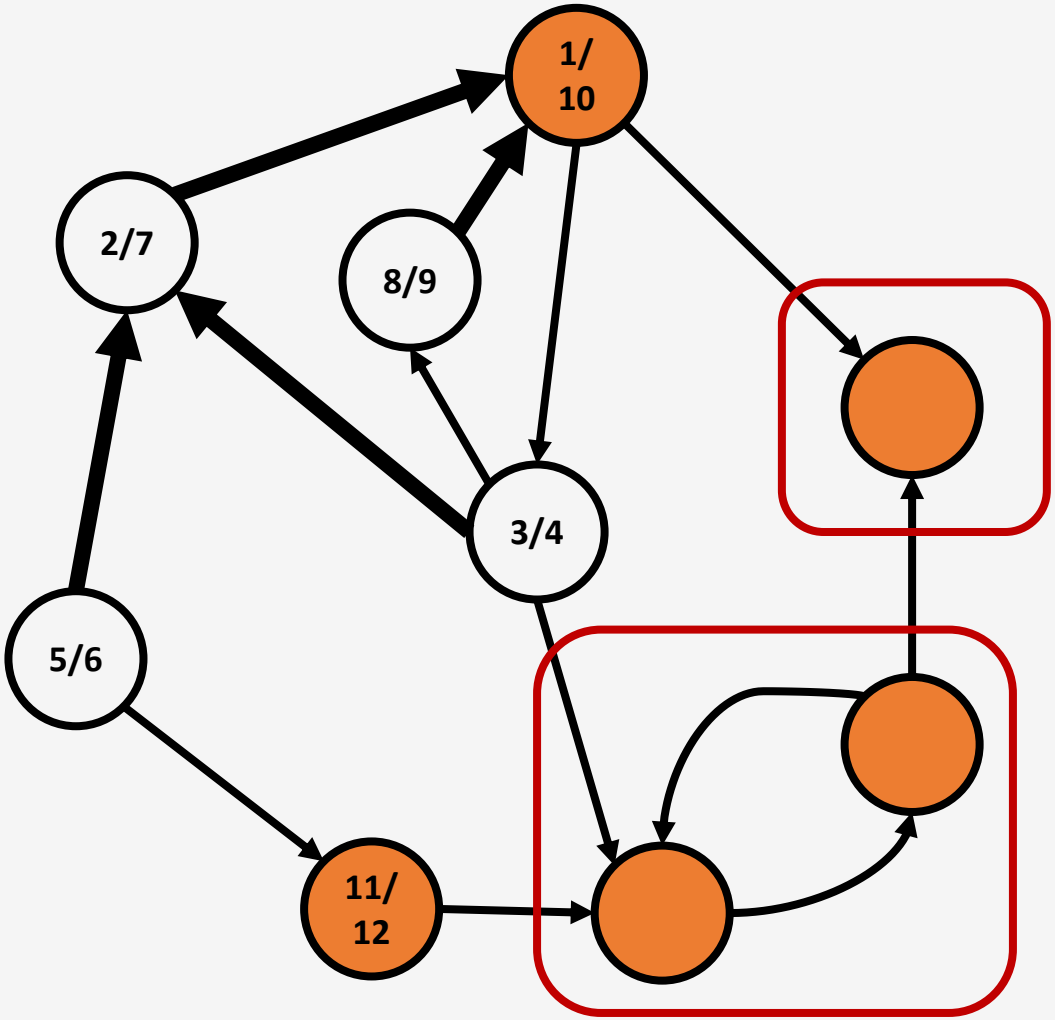
EXAMPLE



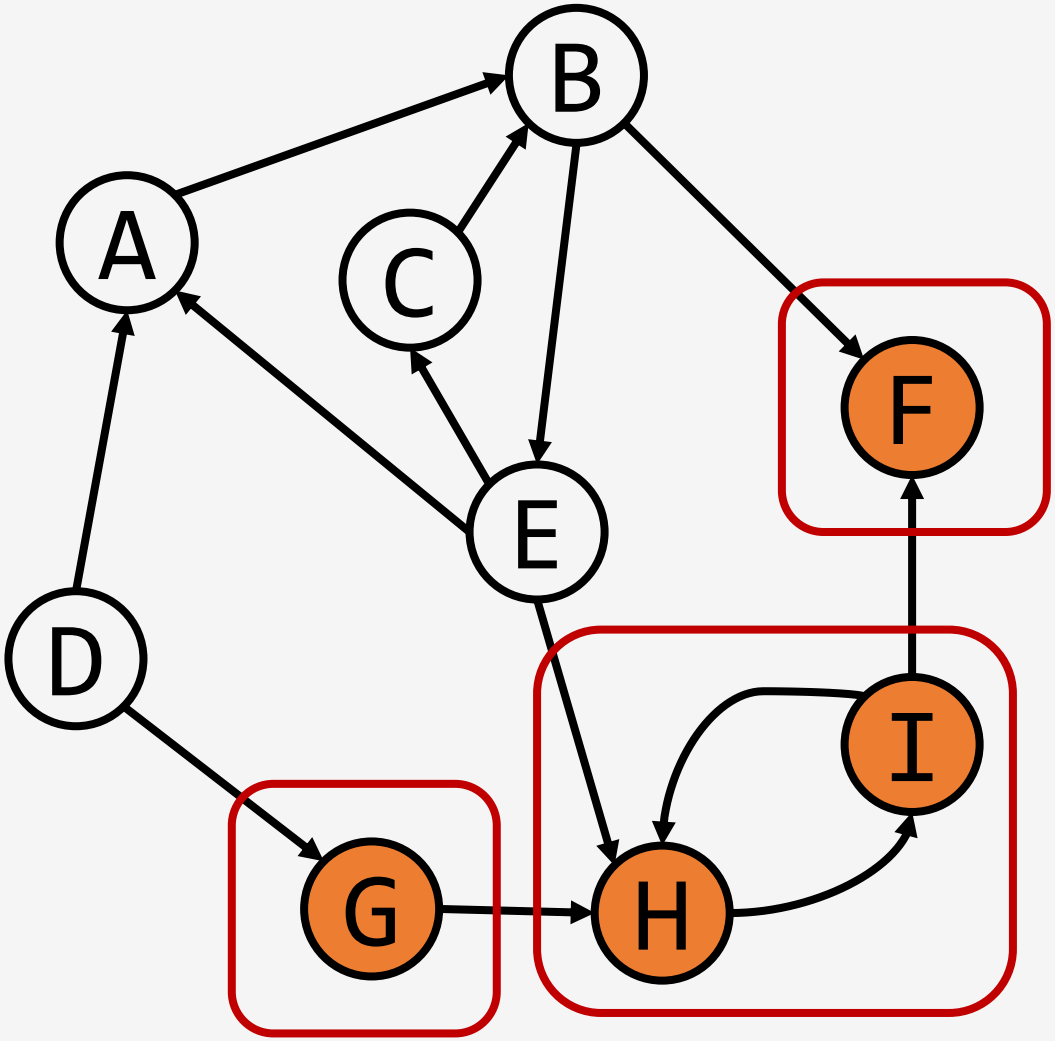
EXAMPLE



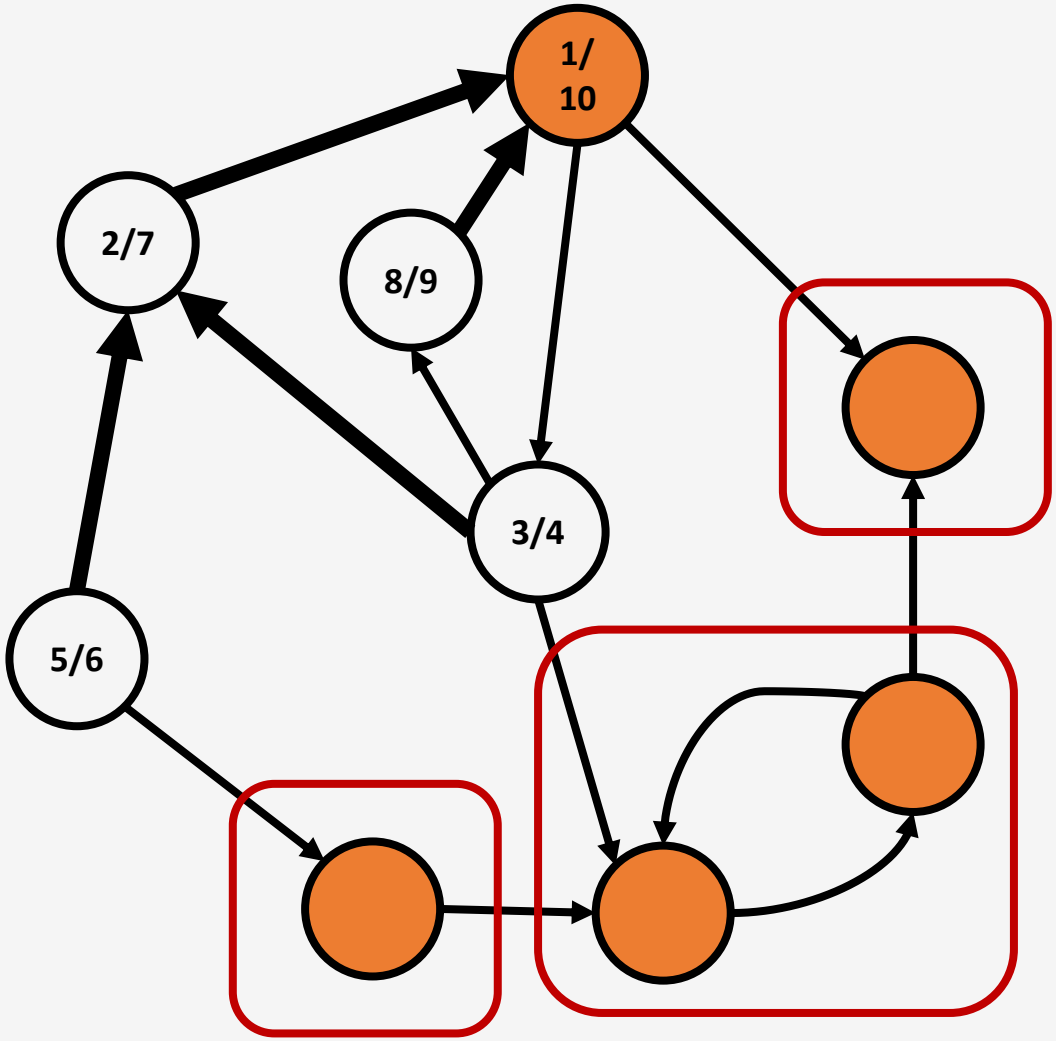
EXAMPLE



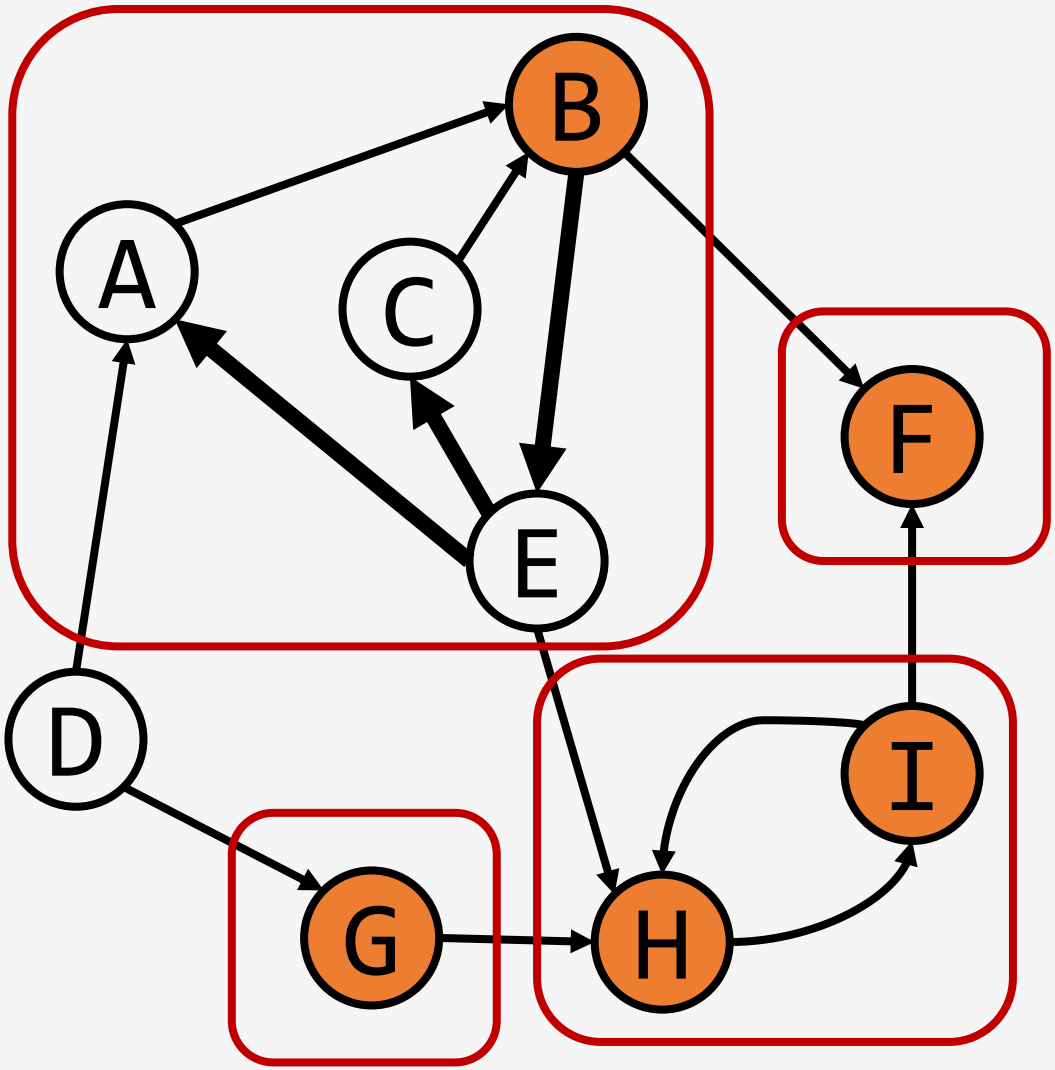
EXAMPLE



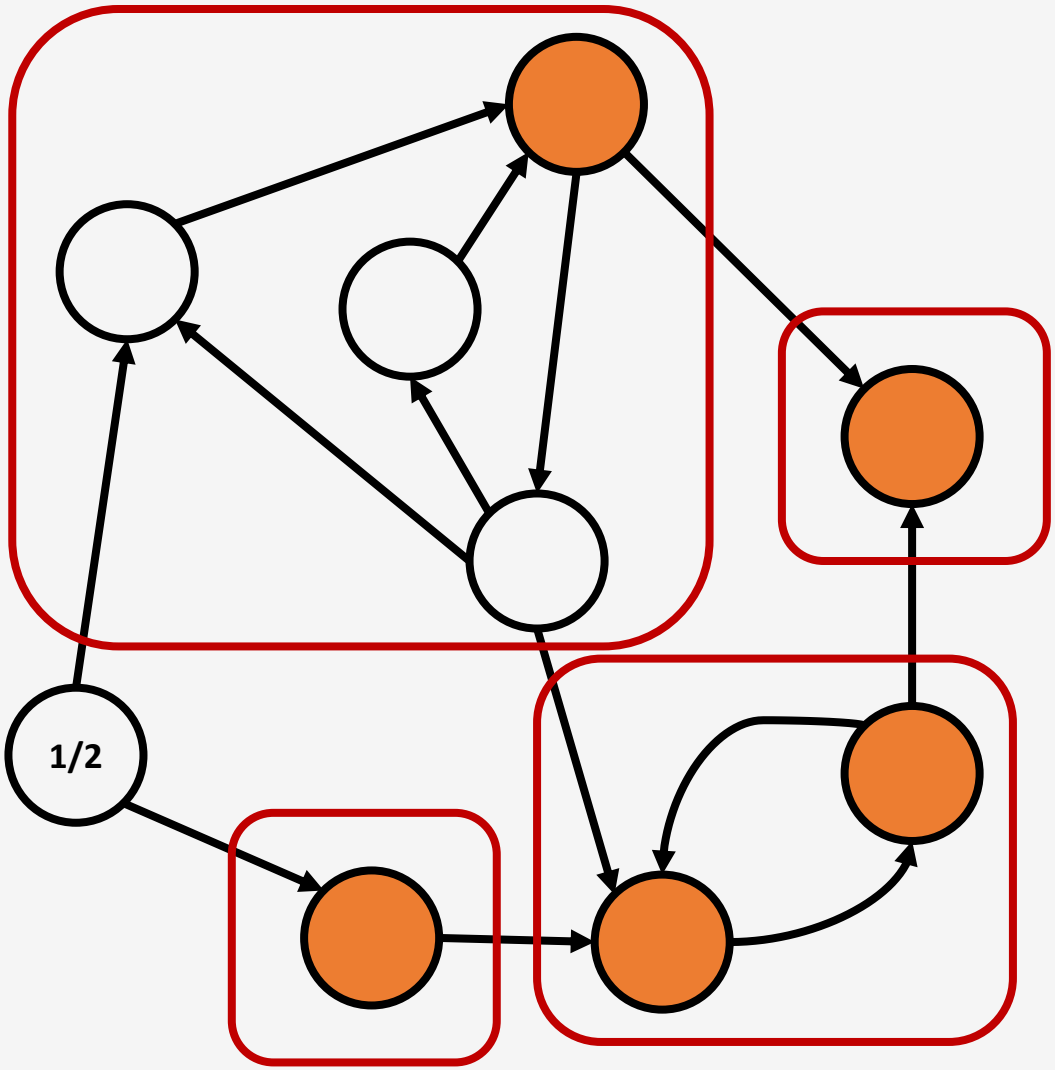
EXAMPLE



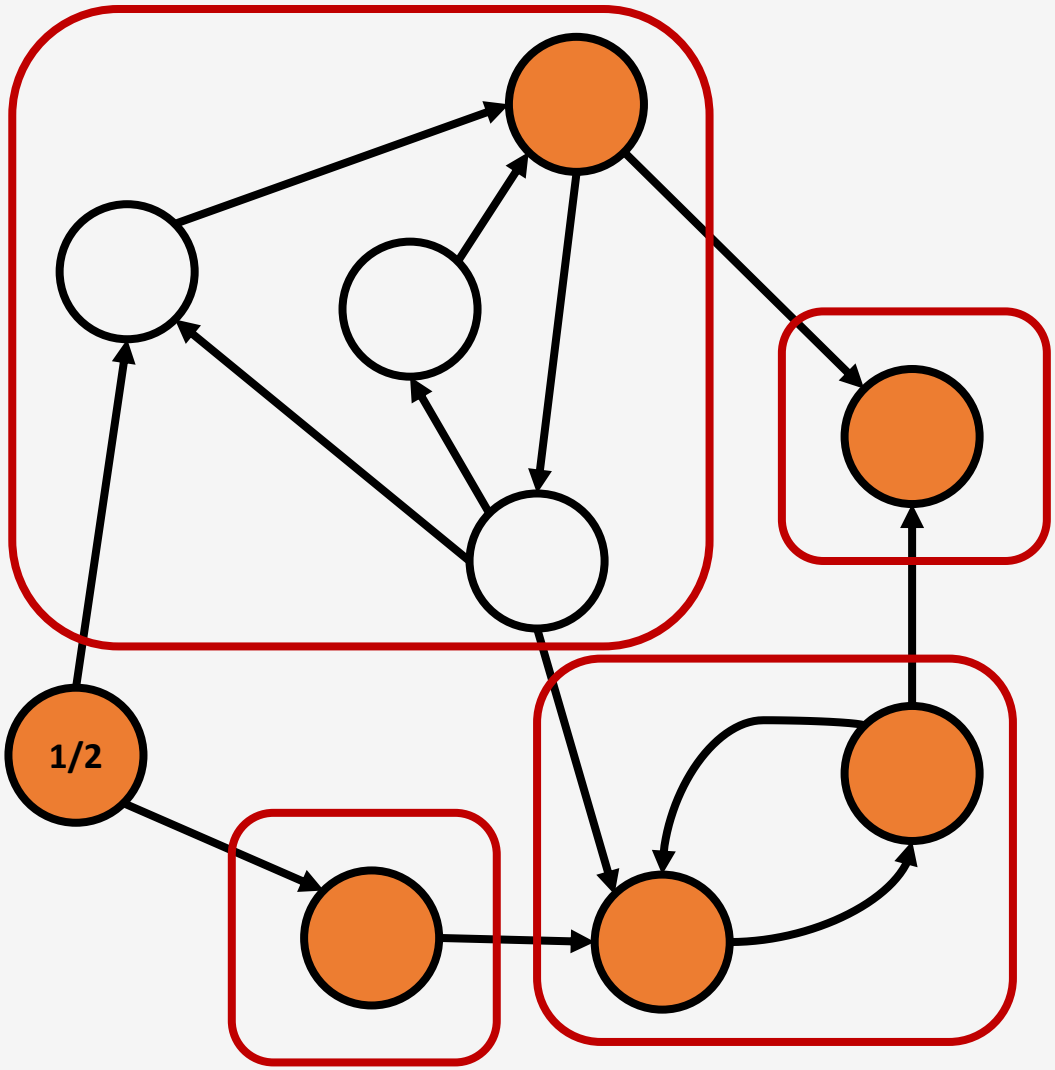
EXAMPLE



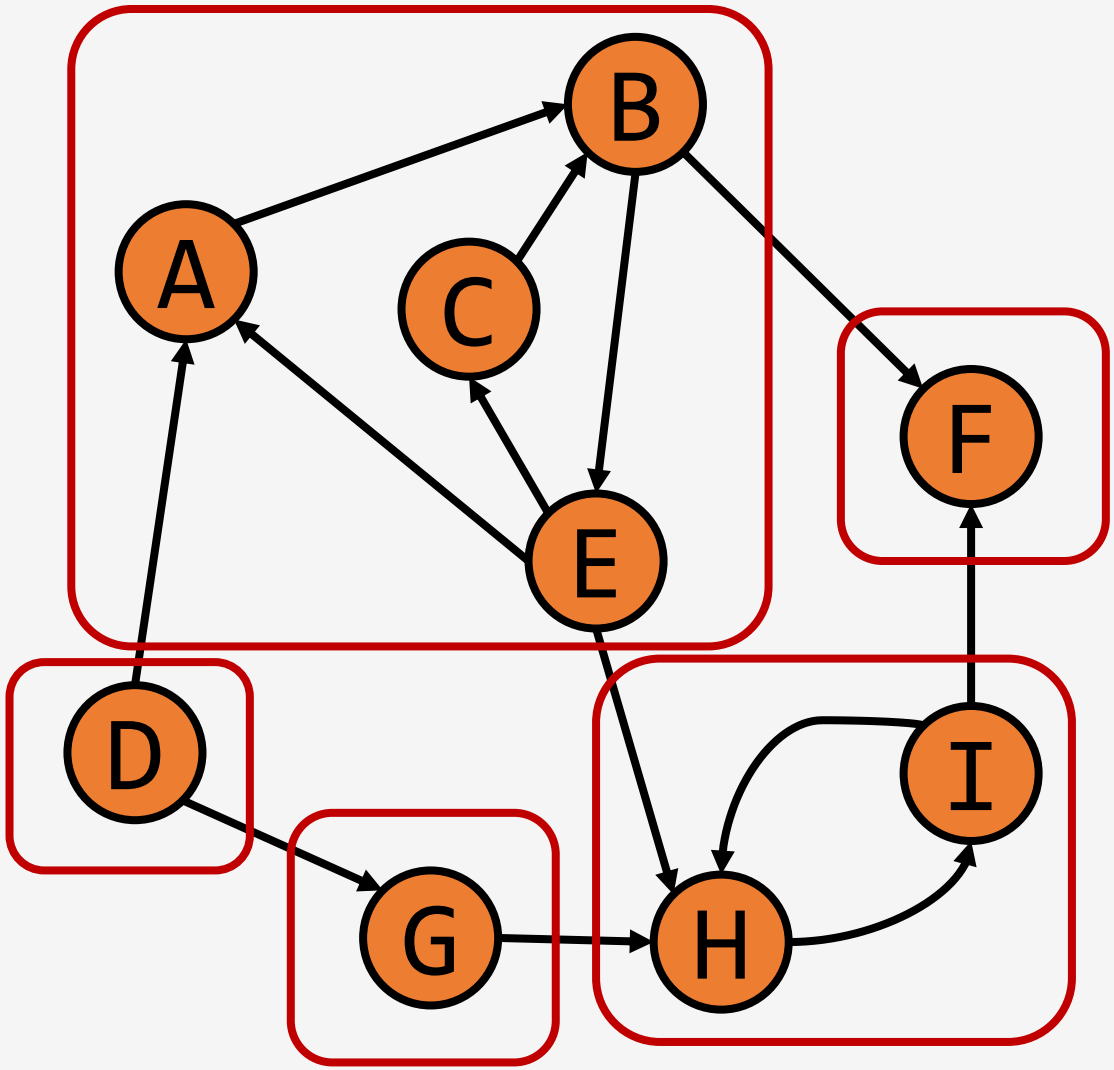
EXAMPLE



EXAMPLE



EXAMPLE



Improvement

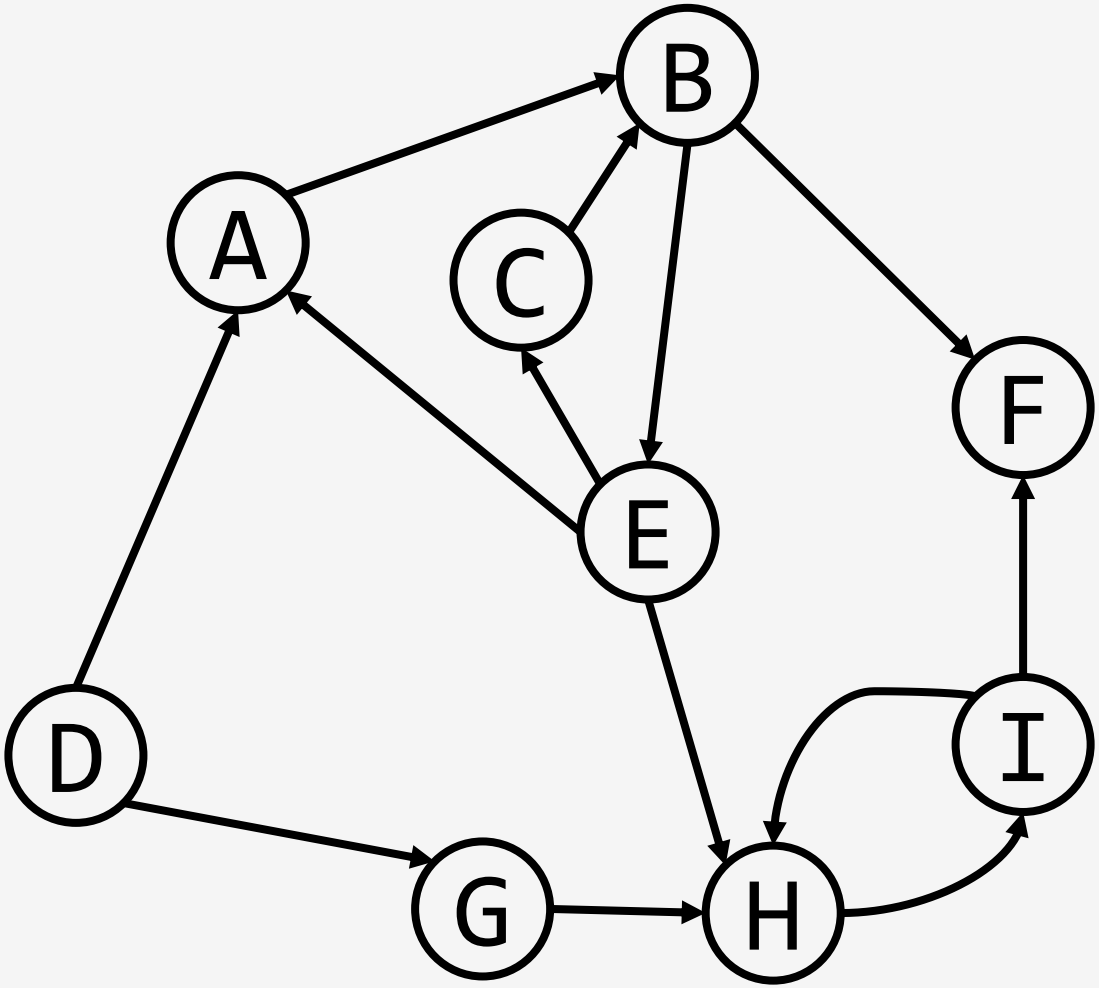
Don't need to rerun DFS on G^R.

Largest remaining post number comes from sink component.

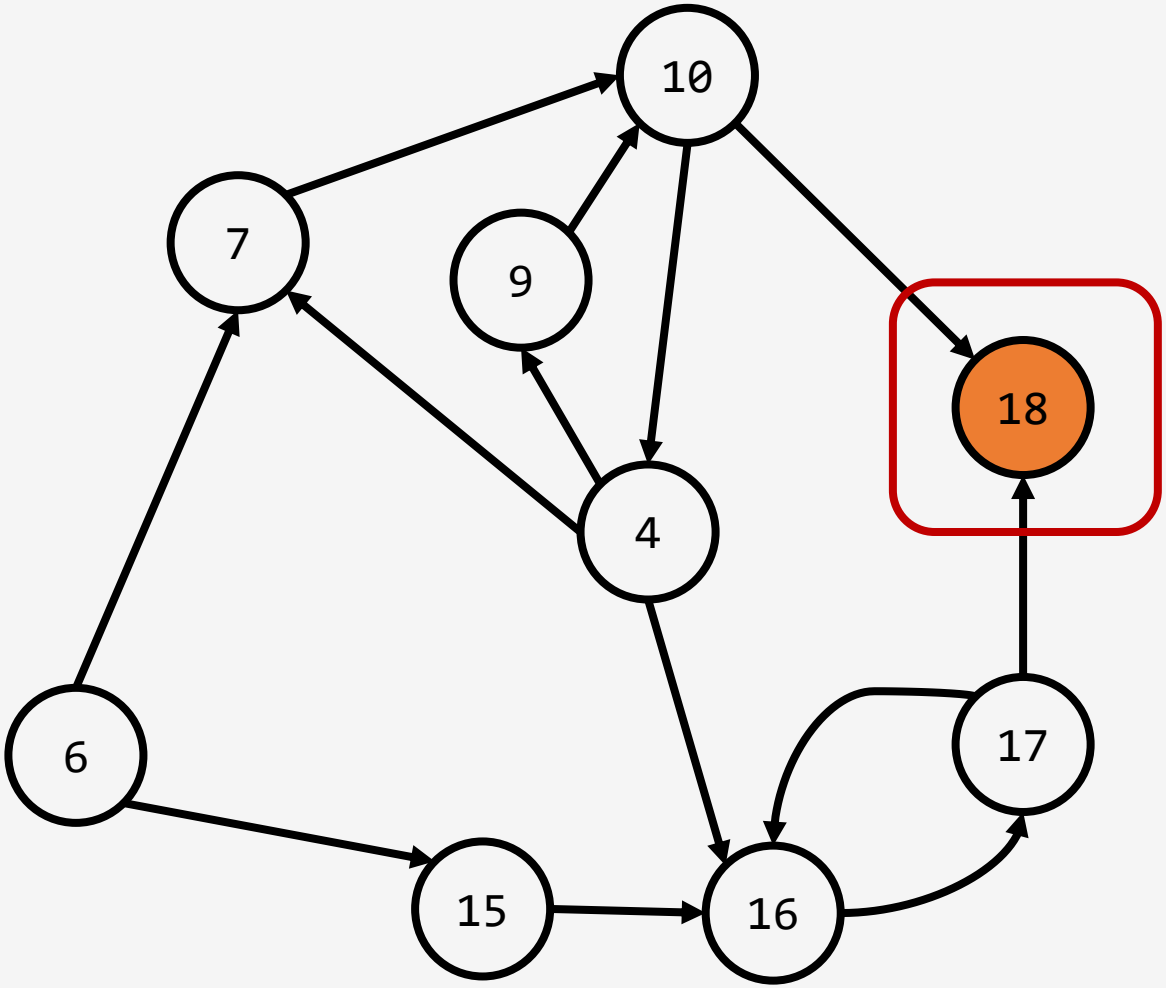
New Algorithm

```
SCCc(G)
    run DFS(G^R)
    for v ∈ V in reverse postorder:
       if not visited(v):
          Explore(v)
          mark visited vertices as new SCC.
```

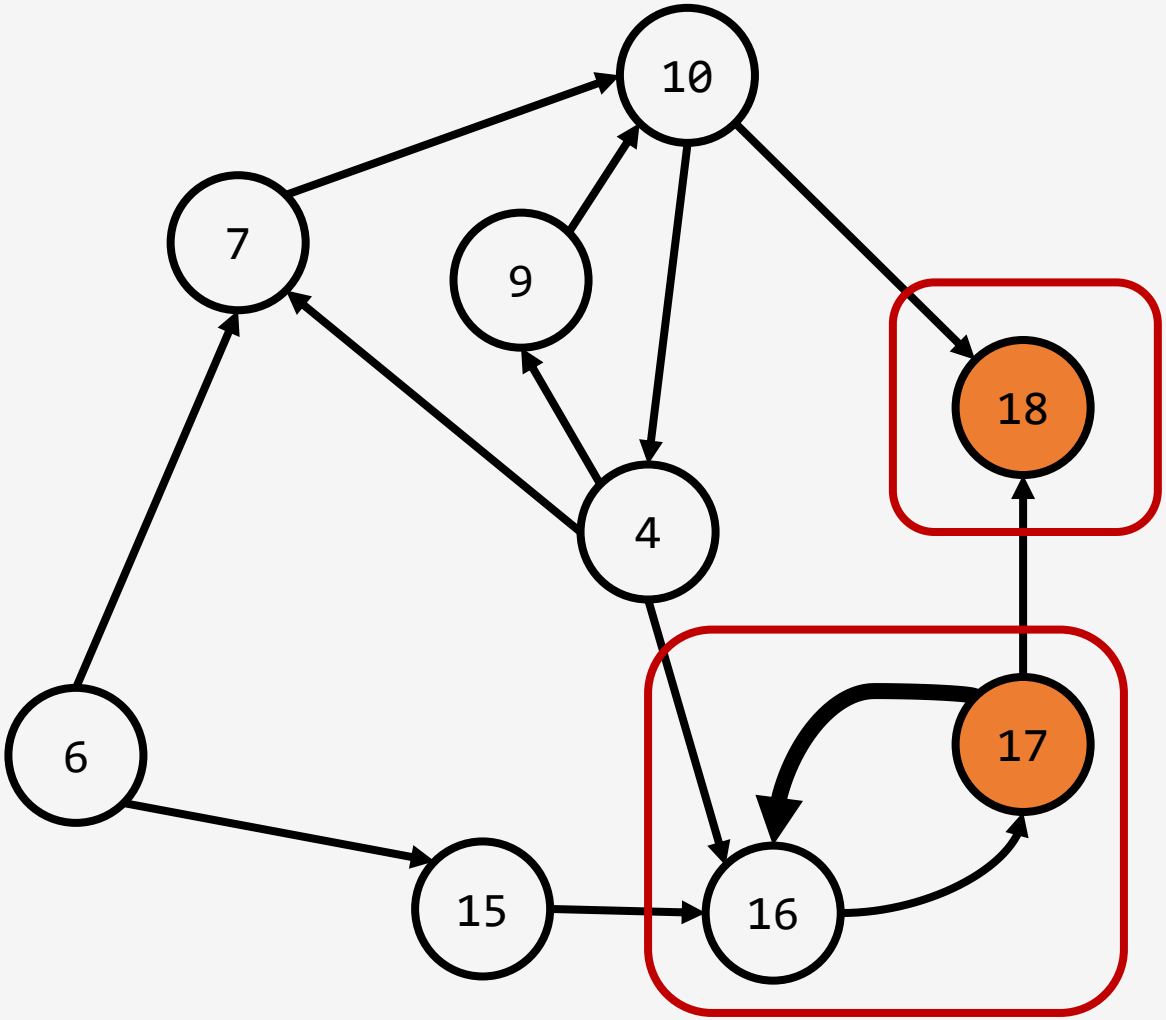
EXAMPLE



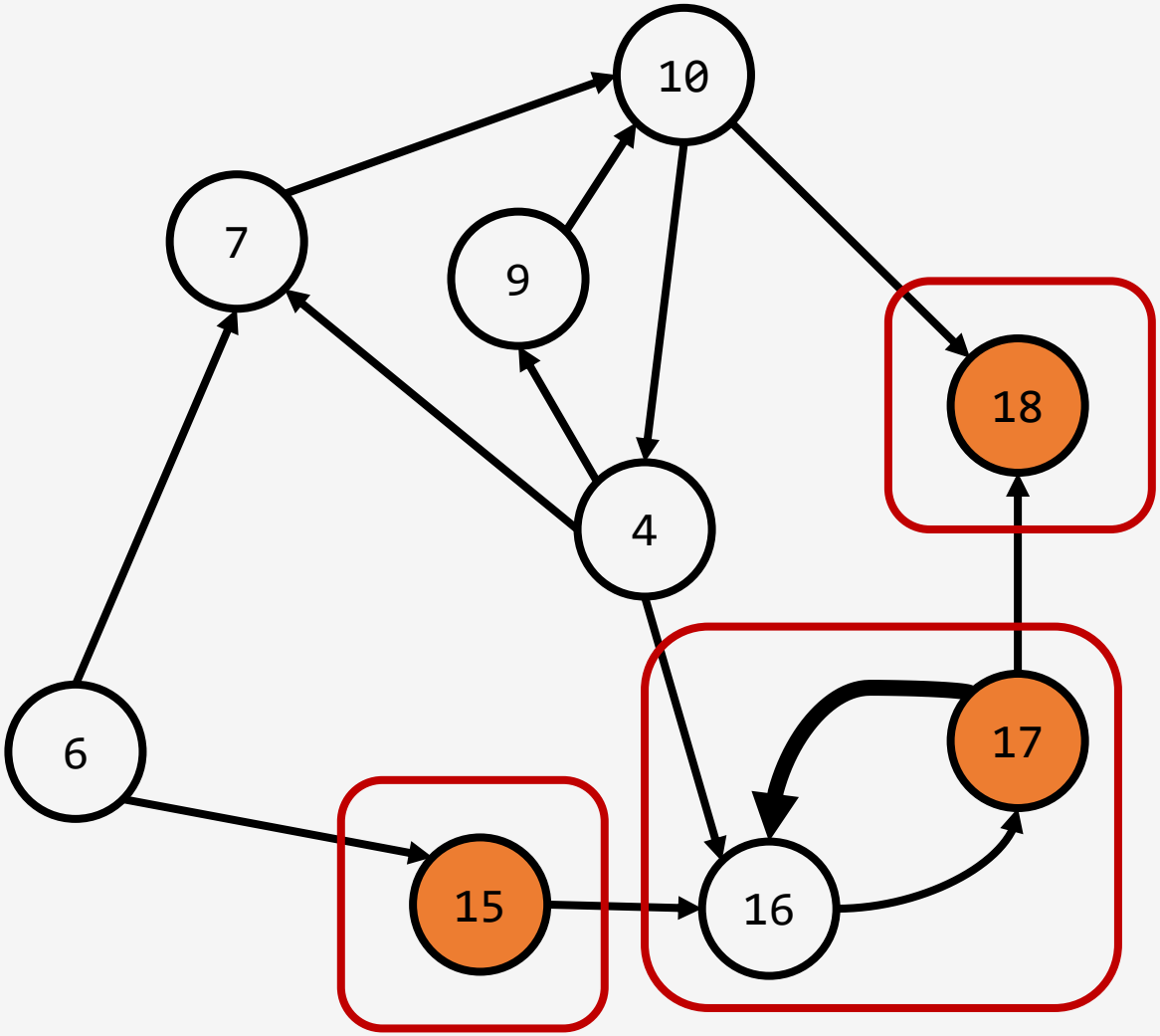
EXAMPLE



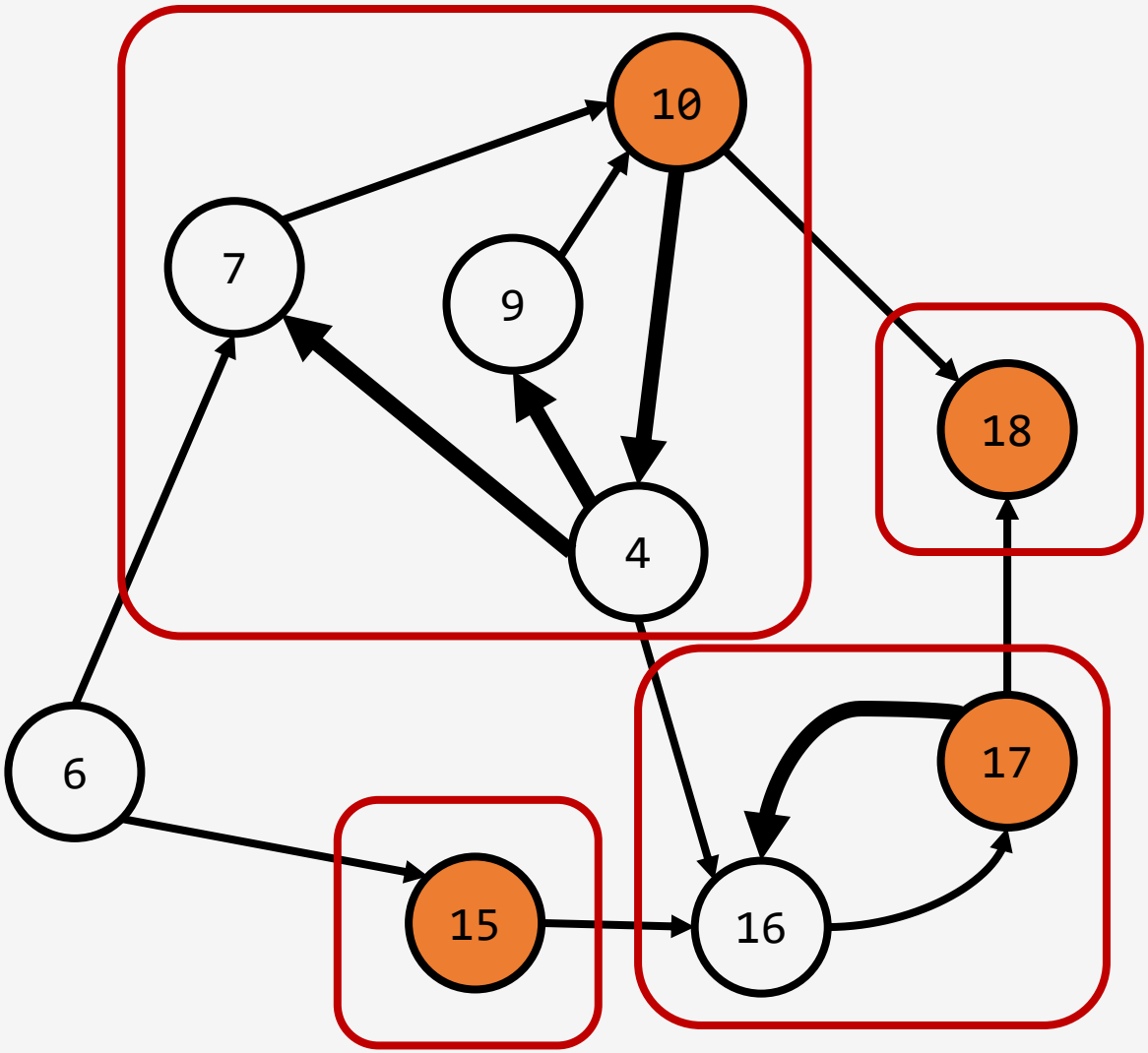
EXAMPLE



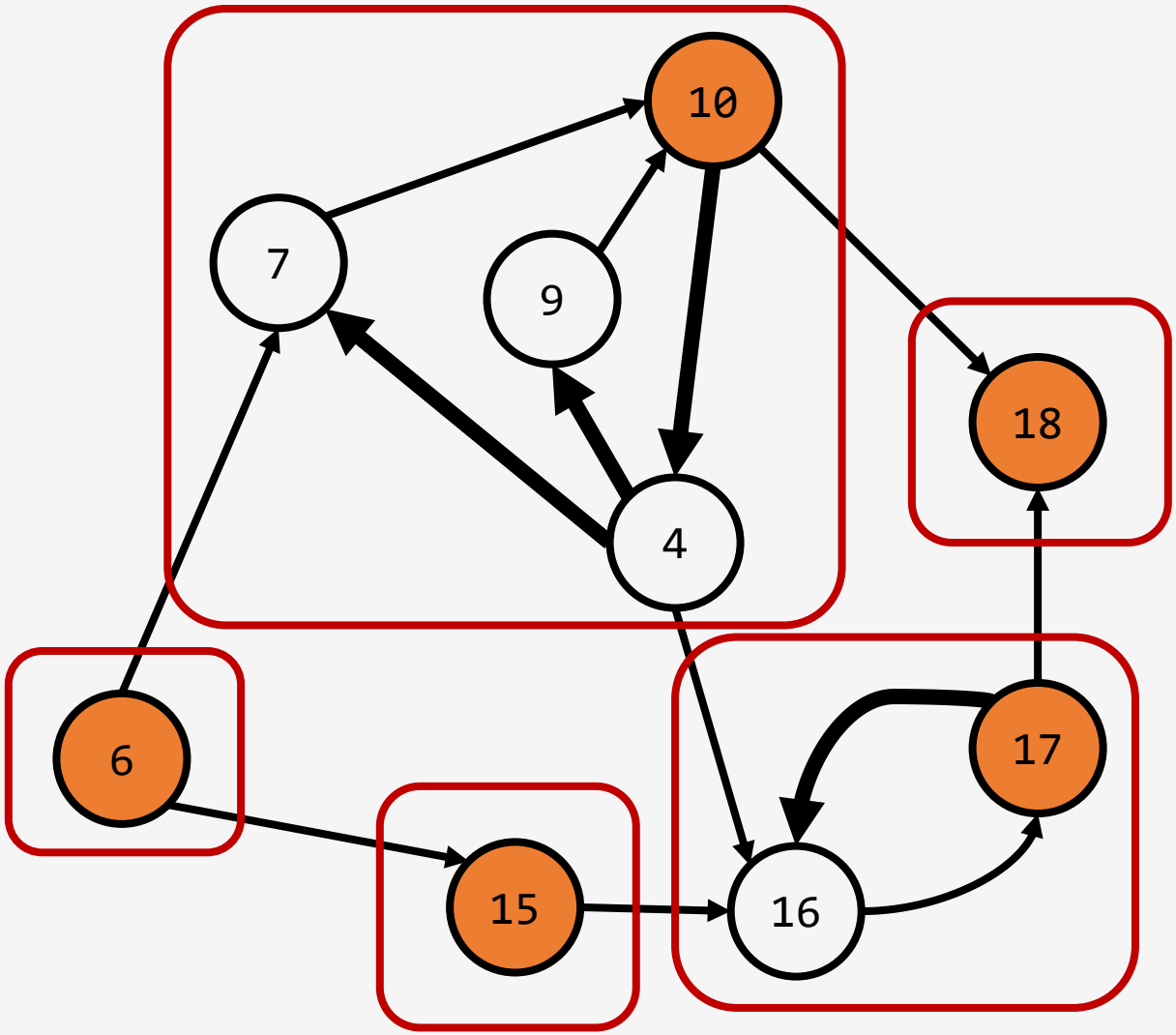
EXAMPLE



EXAMPLE



EXAMPLE



Runtime

Essentially DFS on G^R and then on G.

Runtime O(|V| + |E|).