Date: 05/05/21

Black Board

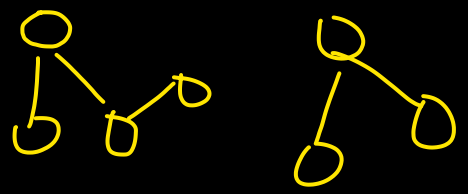
Design and Analysis of Algorithms

Topics:

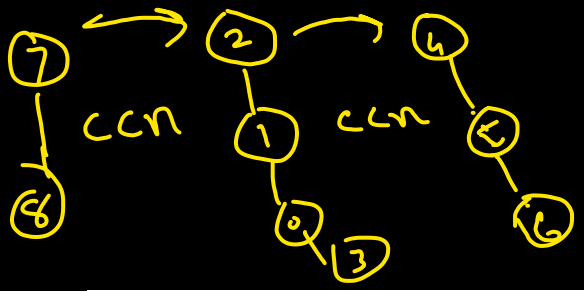
- Graph Algorithms III
  - Connected Components in Undirected Graphs
  - Strongly Connected Components (SCCs) in directed graphs.

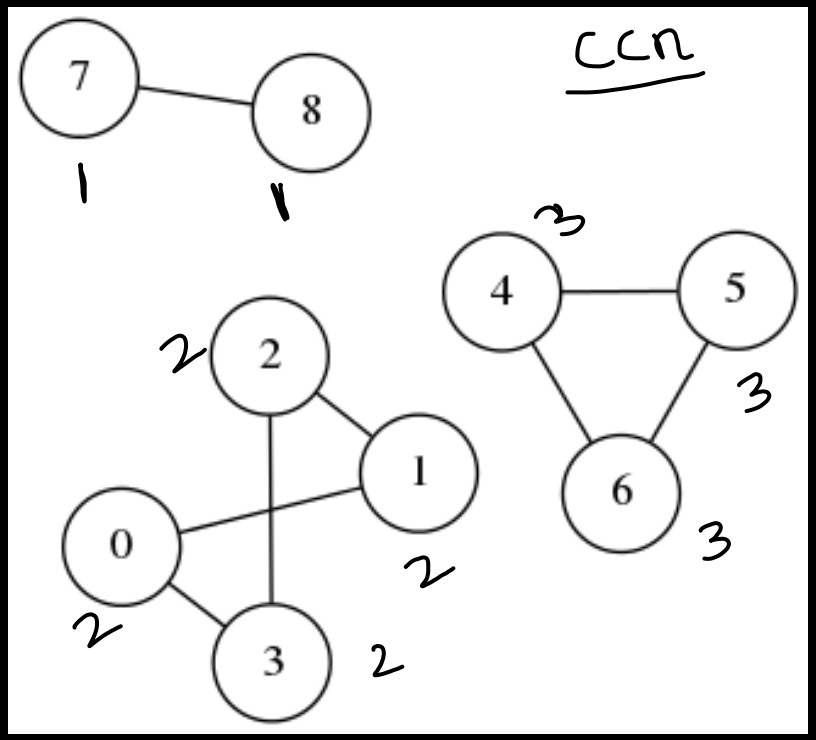
Connected Components and their applications

- A very important way to analyze graph structure
- Example applications:
  - Internet Documents Clustering
  - Social Media Community Detection
  - Protein-Protein Interaction Analysis
  - Image Segmentation
  - Natural Language Processing: word and character separation, etc.



Connected Components in Undirected Graphs





Def:
Two nodes x & y are in the same connected compon - when there is a path x to y in the graph.

Undirected Graphs

```
DFS(G=(V, E))
  For each x ∈ V:
      visited[x] = false
  ccn = 1
  For each x ∈ V:
      if !visited[x]
          explore(G, x)
          ccn = ccn + 1
```

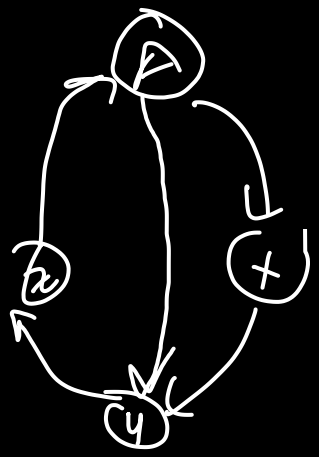
```
Explore(G=(V, E), x)
  visited[x] = true
  comp[x] = ccn
  For each (x, y) ∈ E:
      if !visited[y]
          explore(G, y)
```

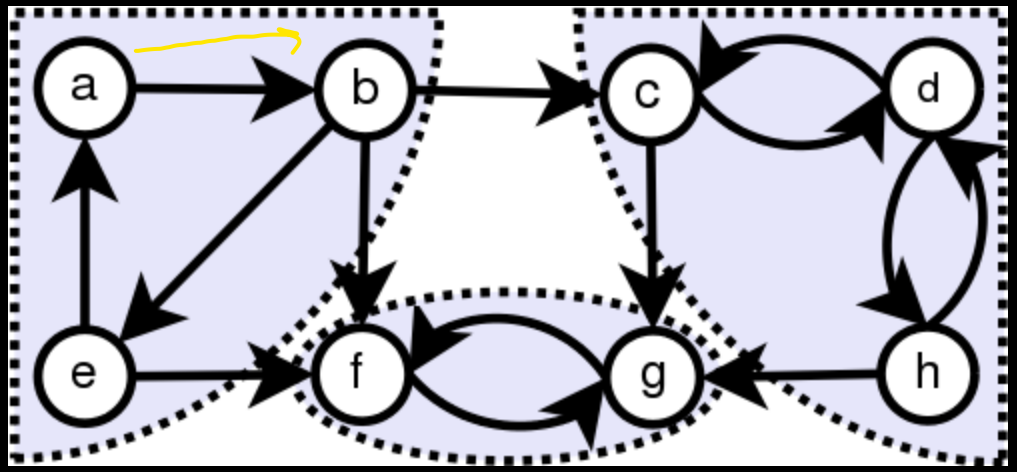
Connected Components in Undirected Graphs

Can be found in $O(|V| + |E|)$ with small additions to the regular DFS code.

Connected Components in Directed Graphs

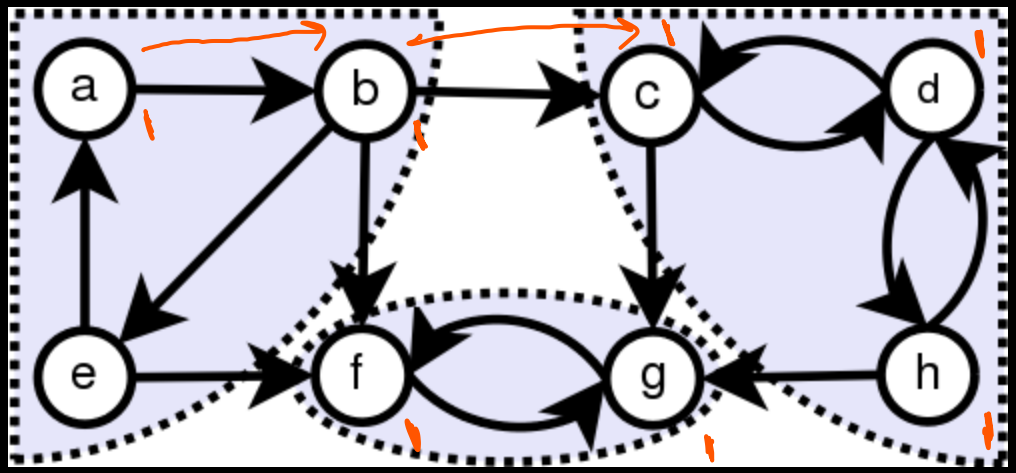
- **Strongly Connected Components (SCCs)**



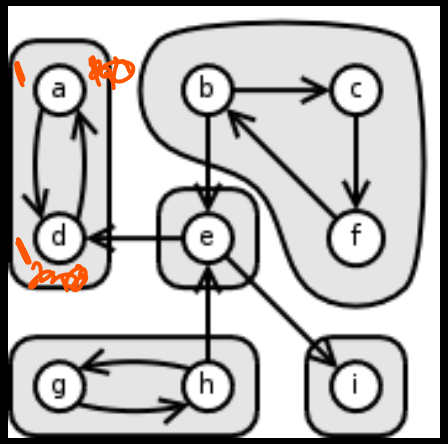


SCC:
For Any pair of vertices x, y in a SCC of G there is a path from x to y & a path from y to x.

The Challenge of finding SCCs in Linear Time



Example 1



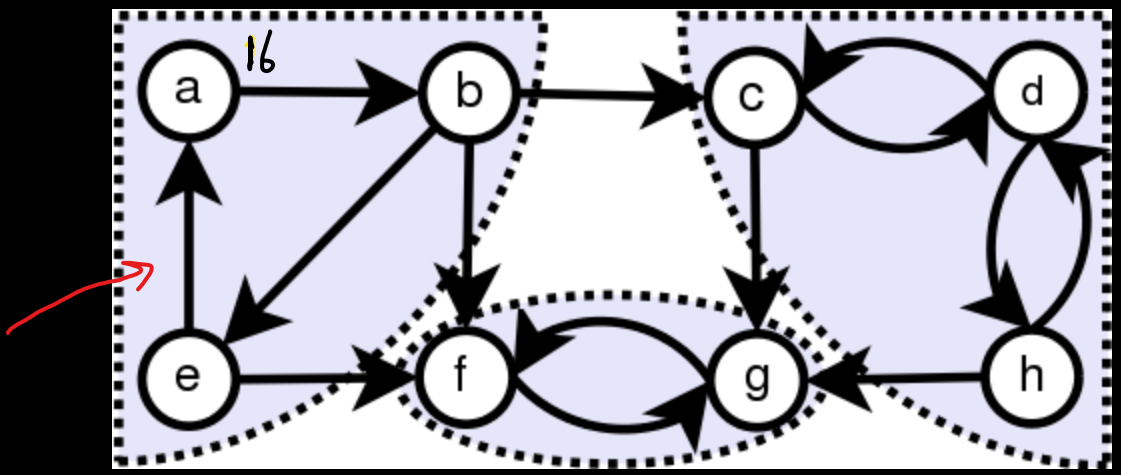
Example 2

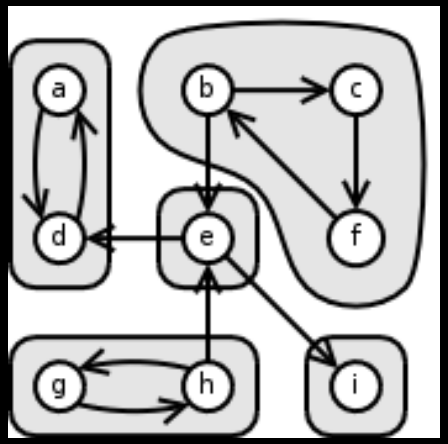
CCN=1

(Return to DFS)

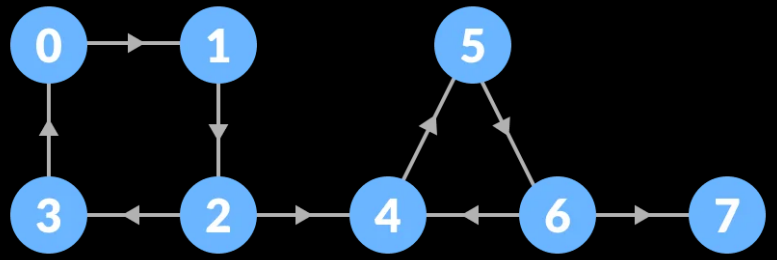
How to stop the chain – explores again labeling every component separately.

Relationship of SCCs with DAGs





What is the Component DAG of the following graph?



Sinks, Sources, DAGs

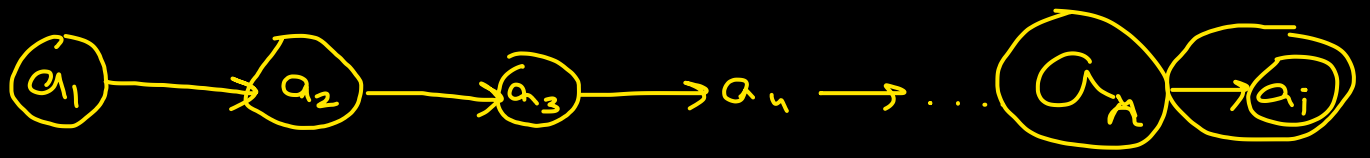
- Where should be start each **explore** call?

— From a sink scc.

FACT: Every DAG has at least one source and at least one sink

- **Proof:**

→ think of a DAG $G = (V, E)$ $V = \{a_1, a_2, \ldots, a_n\}$

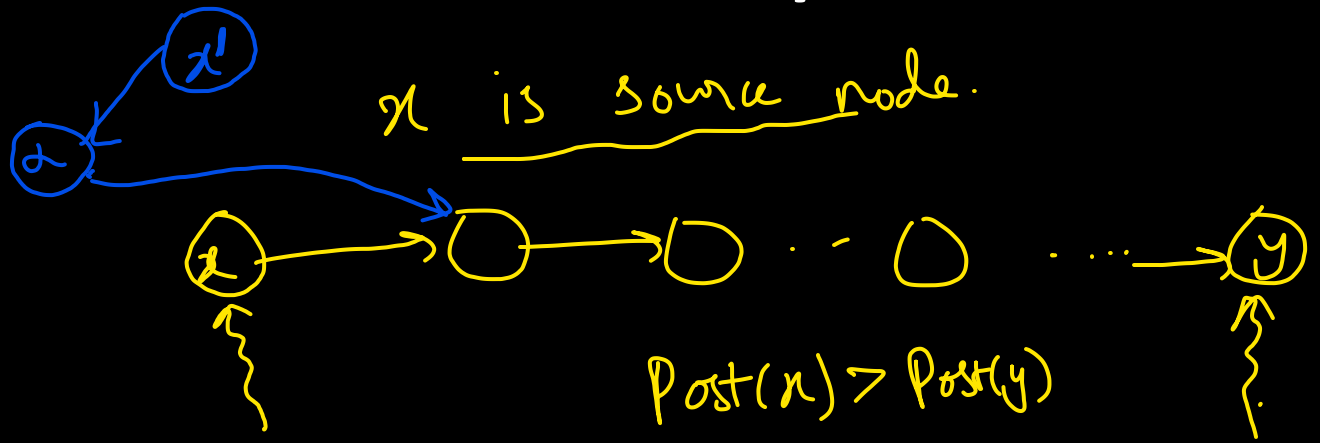


has no out

Our Component DAG will always contain a Sink Component & a So c. Comp.

- So the component DAG has at least one *sink component node* and at least one *source component node*.

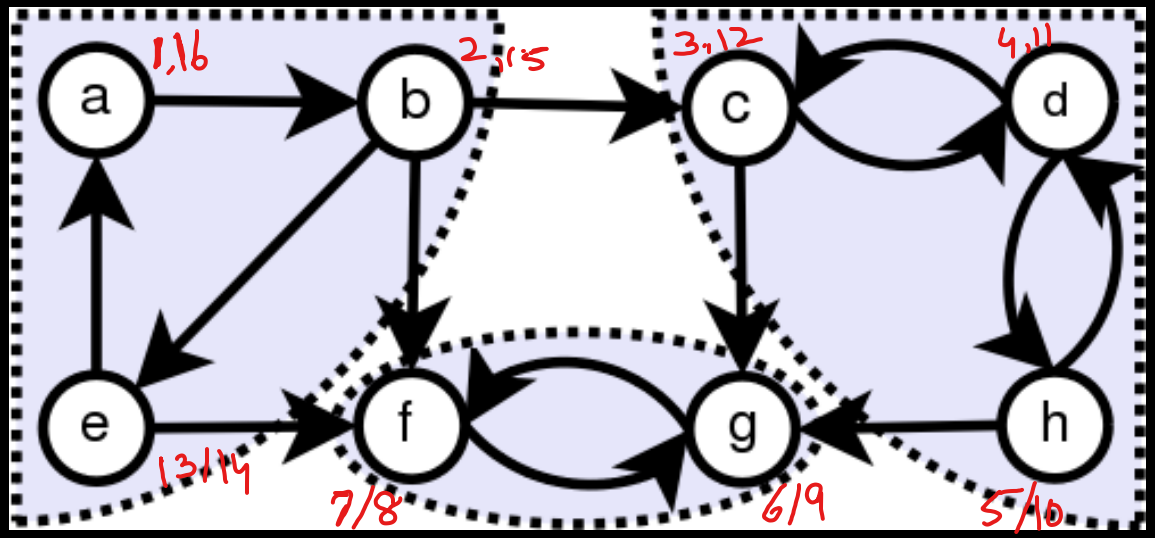
- ***How to detect a sink component node?***

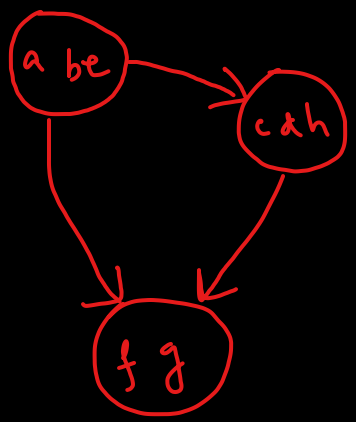


The highest post # must belong to a source node.

Using post numbers to find sink component nodes

G

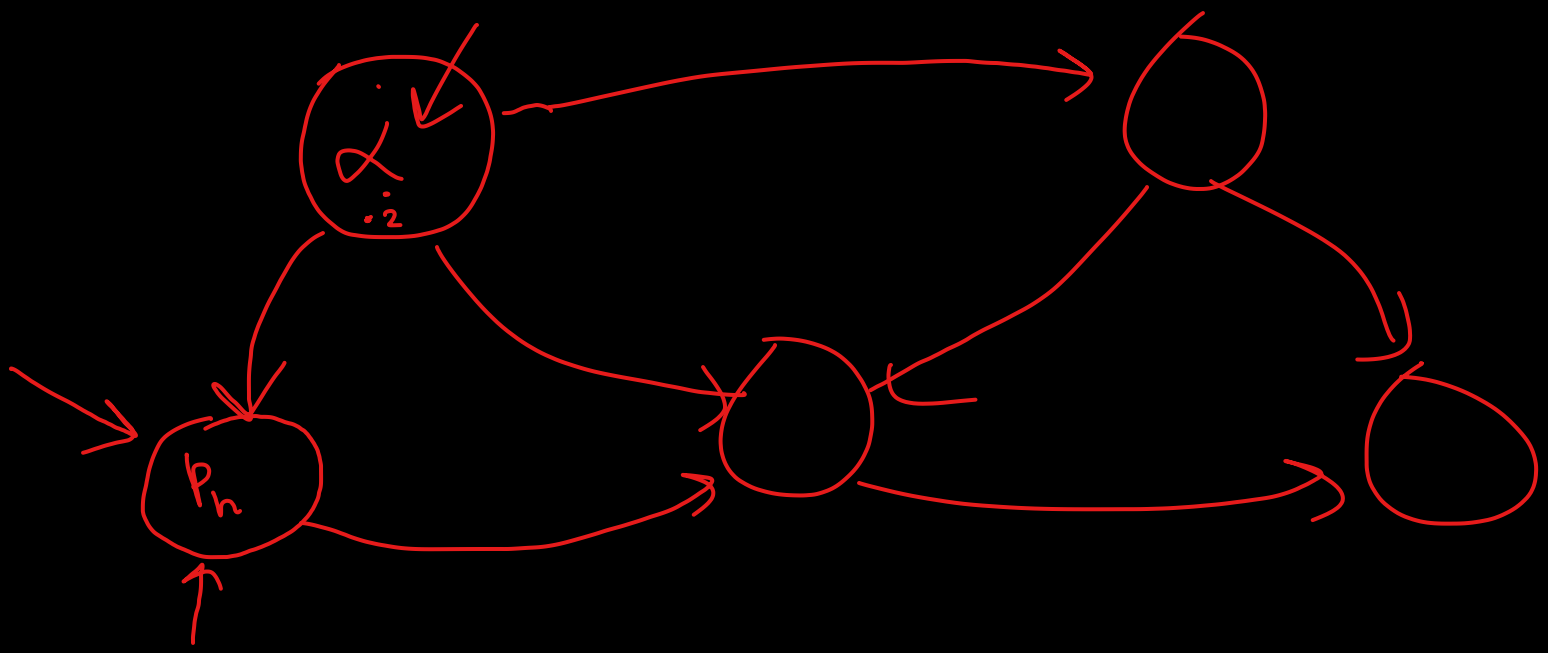




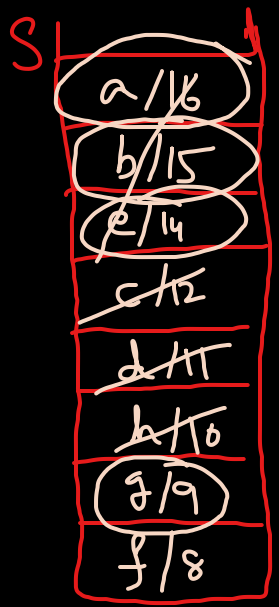
Claim: The node with the highest post# must be in a Source Component

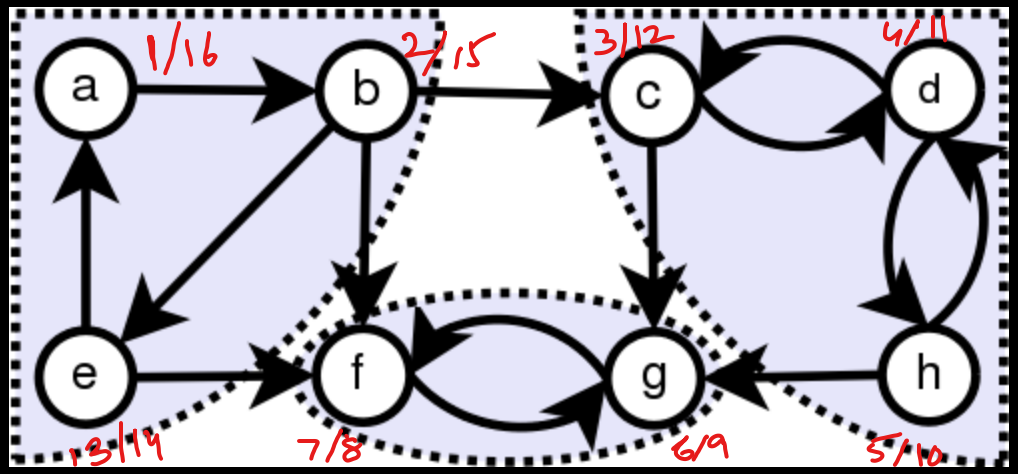
Post Numbers and Source Components

- FACT: the highest post number must be in a source SCC.
- Proof:



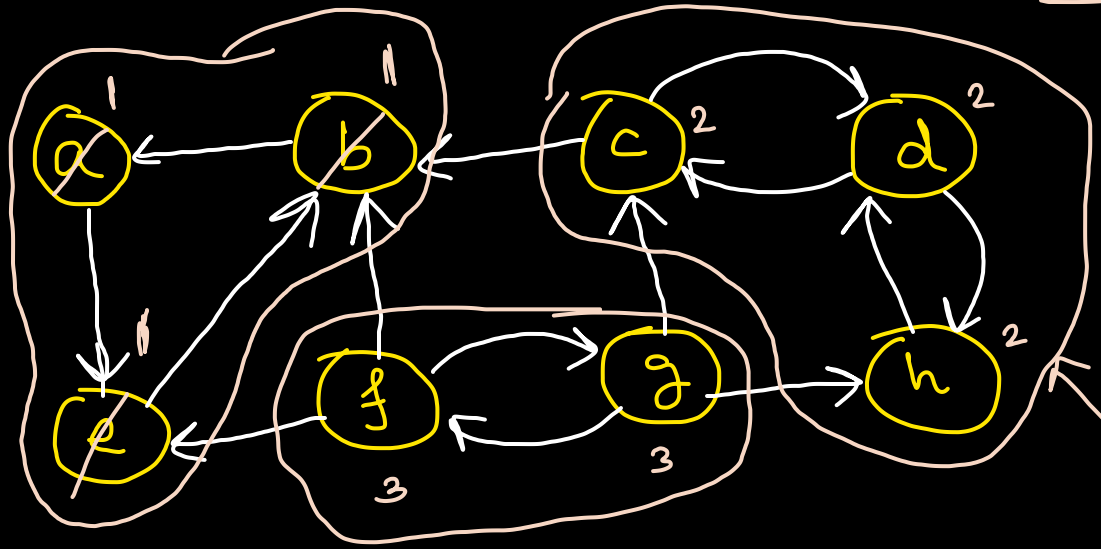
The Graph Reversal Trick





$G=(V, E)$

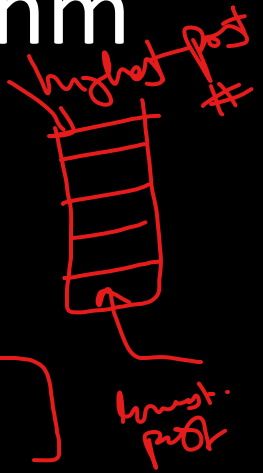
ccnum[x] = ccn



$G_R=(V, E_R)$

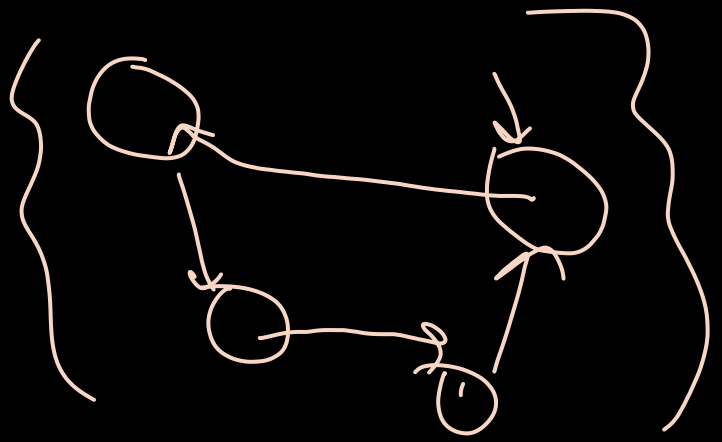
f, g

The Linear time SCC finding Algorithm (Kosaraju's Algorithm)



First Two Steps:

- Perform regular DFS on G, and get vertices in decreasing order of post numbers in a stack S
- Create reverse Graph $G_R = (V, E_R)$ in $O(|V|+|E|)$
- Call ModifiedDFS($G_R=(V, E_R)$, S)

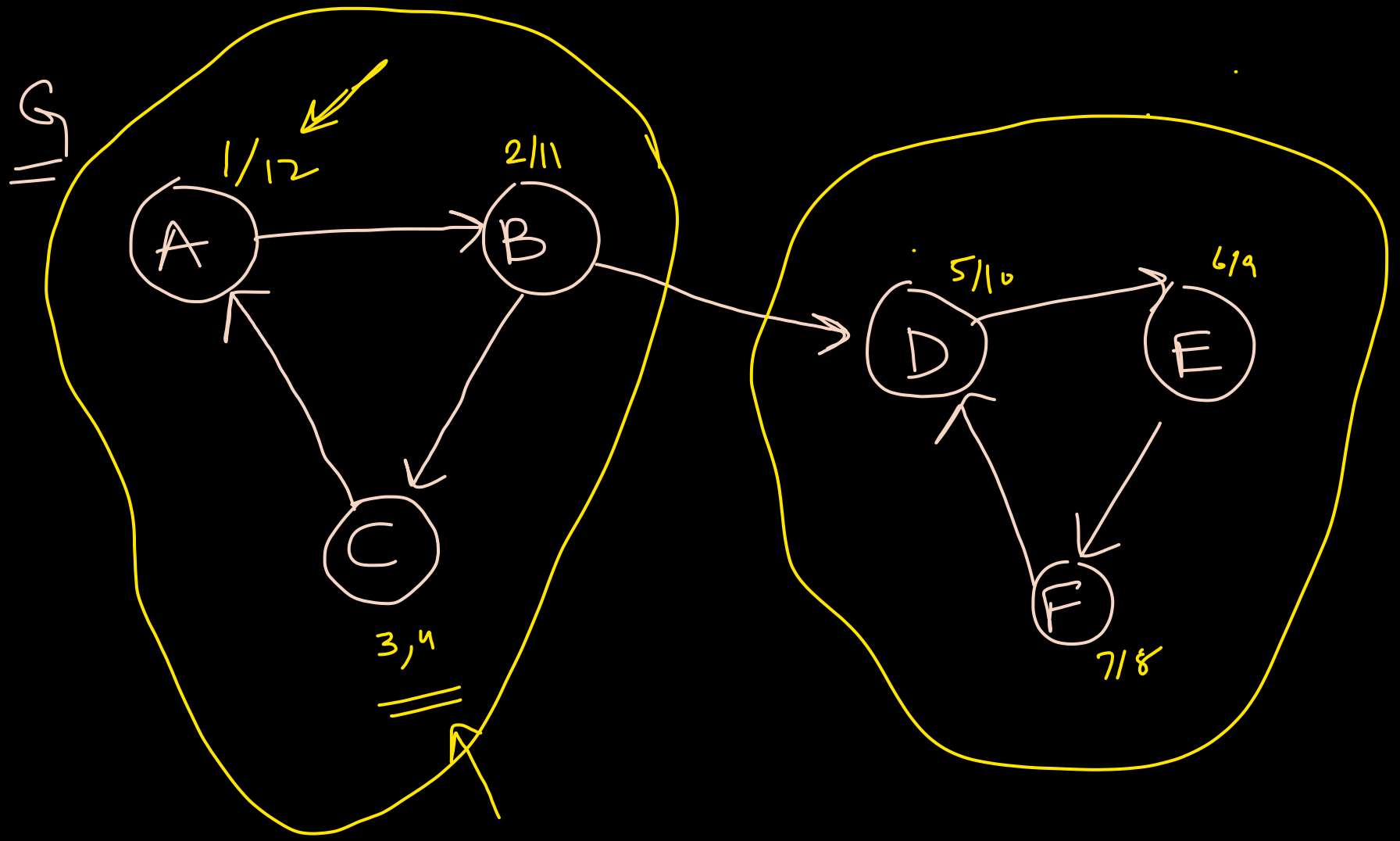


$T(G=(V,E)) = O(|V|+|E|)$

Modified DFS and Explore Methods

```
ModifiedDFS(G=(V, E), S)
  For each x ∈ V:
        visited[x] = false
  ccn = 1
  WHILE !S.empty():
        x ← S.top()
        if !visited[x]
              Explore(G, x)
              ccn = ccn + 1
        S.pop()
```

```
Explore(G=(V, E), x)
  visited[x] = true
  comp[x] = ccn
  For each (x, y) ∈ E:
        if !visited[y]
              explore(G, y)
```


G
1/12
2/11
A
B
C
3,4
5/10
6/9
D
E
F
7/8