

Property of DAGS

Linearization of a DAG:

Since we know that edges go in the direction of decreasing post numbers, if we order the vertices by decreasing post numbers then we will have a linearization

procedure **linearize**(DAG $G = (V, E)$)

run DFS(G)

return list of vertices in decreasing order of post numbers
(by putting at start of list when post number assigned)

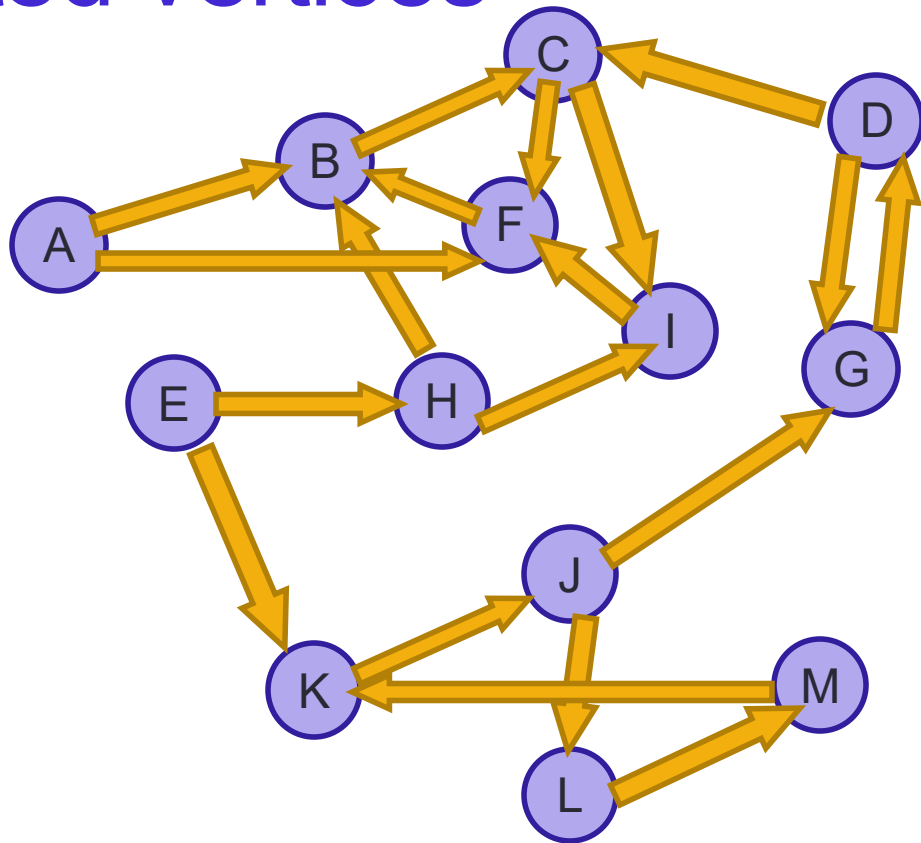
Sources and sinks

- Since all DAGs can be linearized, that means the first vertex in the ordering does not have any edges coming in and the last vertex does not have any edges going out.
- **Definitions:**
 - A vertex with no incoming edges is called a **source**
 - A vertex with no outgoing edges is called a **sink**
- **Theorem:** All DAGs have at least one source and one sink.

Strongly connected vertices

Two vertices u and v in a directed graph are strongly connected if there exists a path from u to v and a path from v to u .

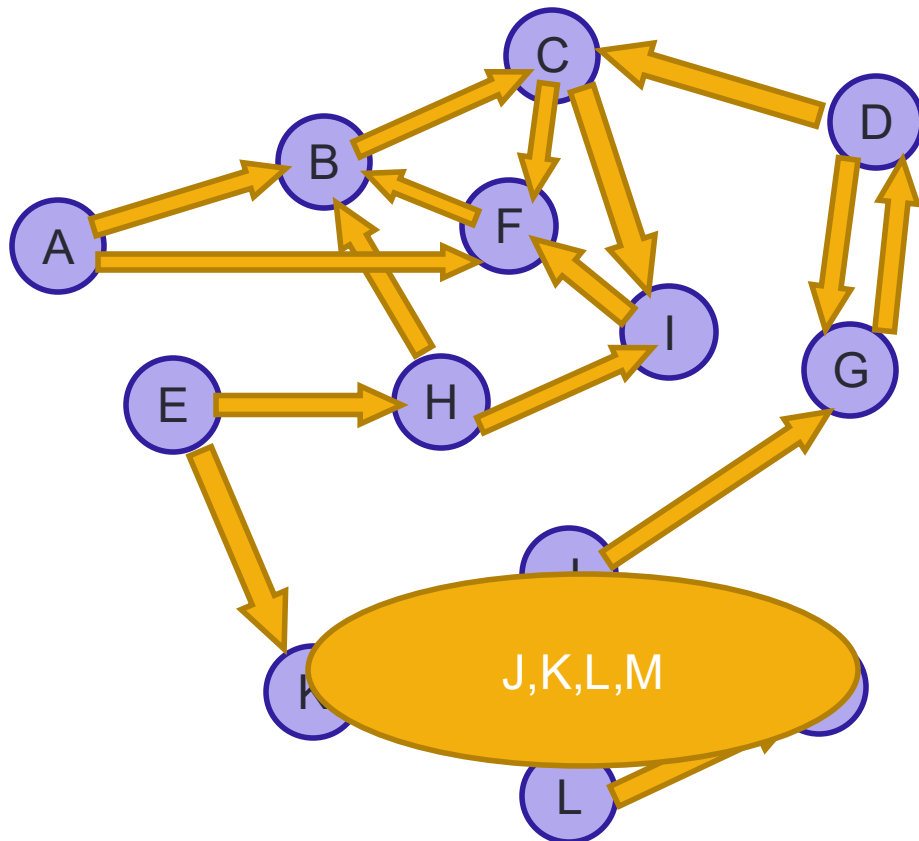
Which vertices are strongly connected to J?



Strongly connected vertices

Two vertices u and v in a directed graph are strongly connected if there exists a path from u to v and a path from v to u .

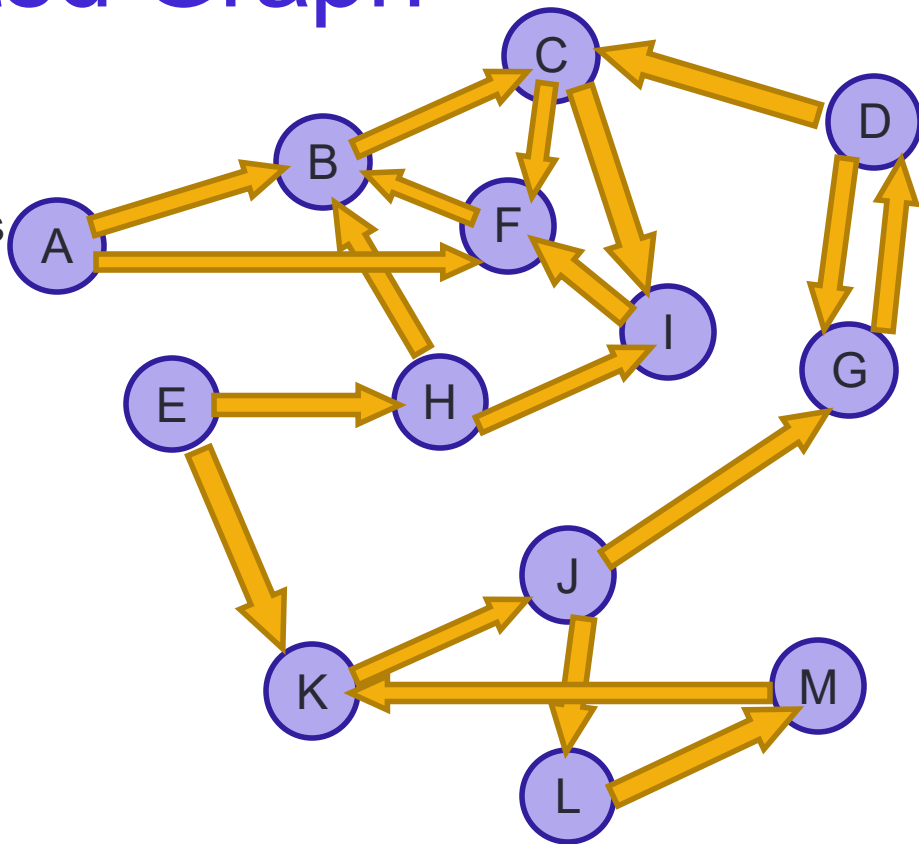
Which vertices are strongly connected to J? {J, K, L, M}



Strongly connected Graph

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

Is this a strongly connected graph?



Strongly connected components

Consider the relation uRv if u is strongly connected to v .

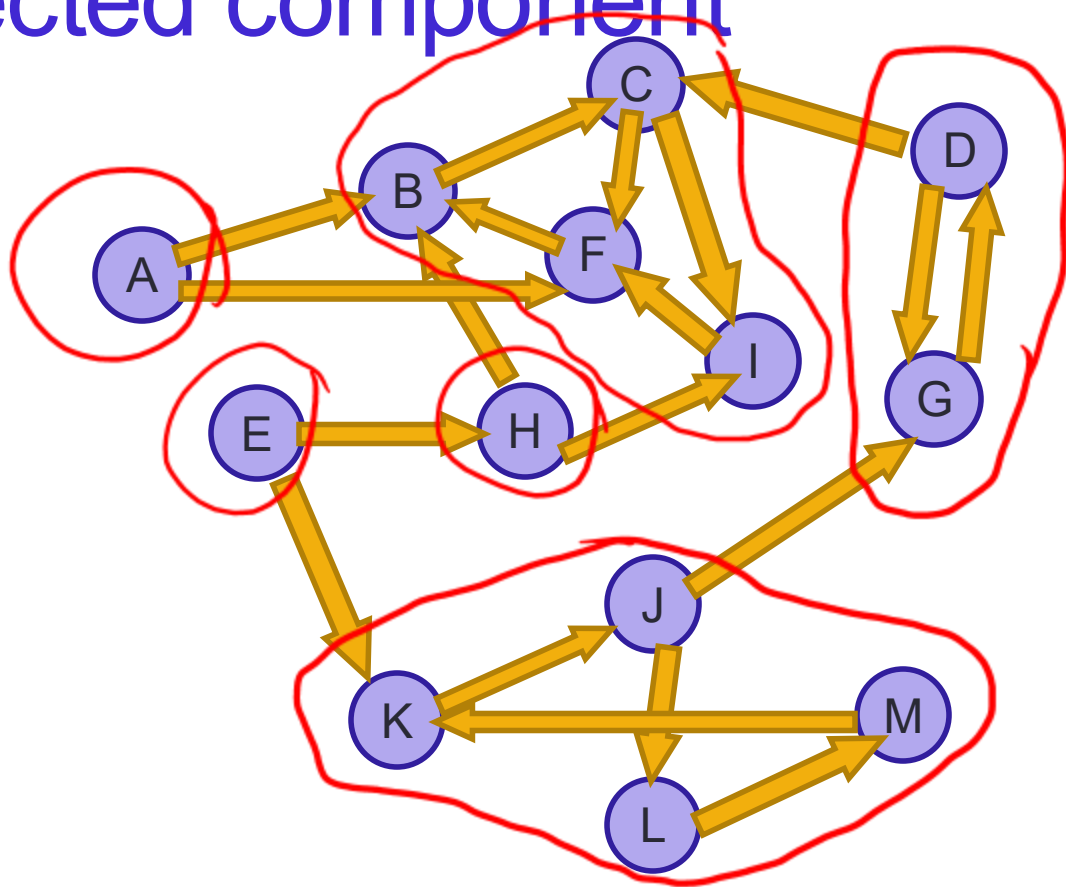
Then R is an equivalence relation. It is reflexive, symmetric and transitive.

So R partitions V , the set of vertices into equivalence classes.

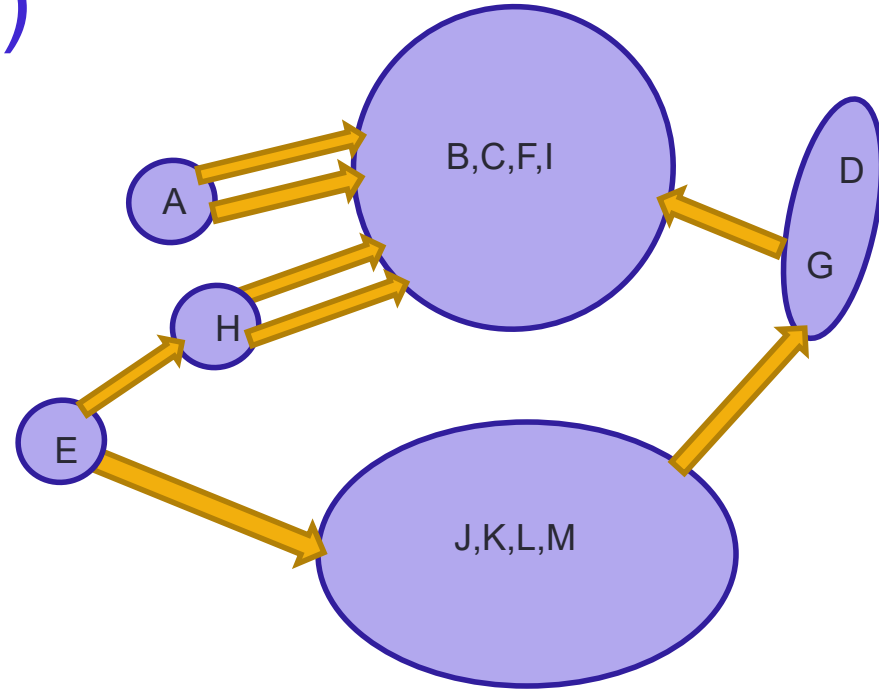
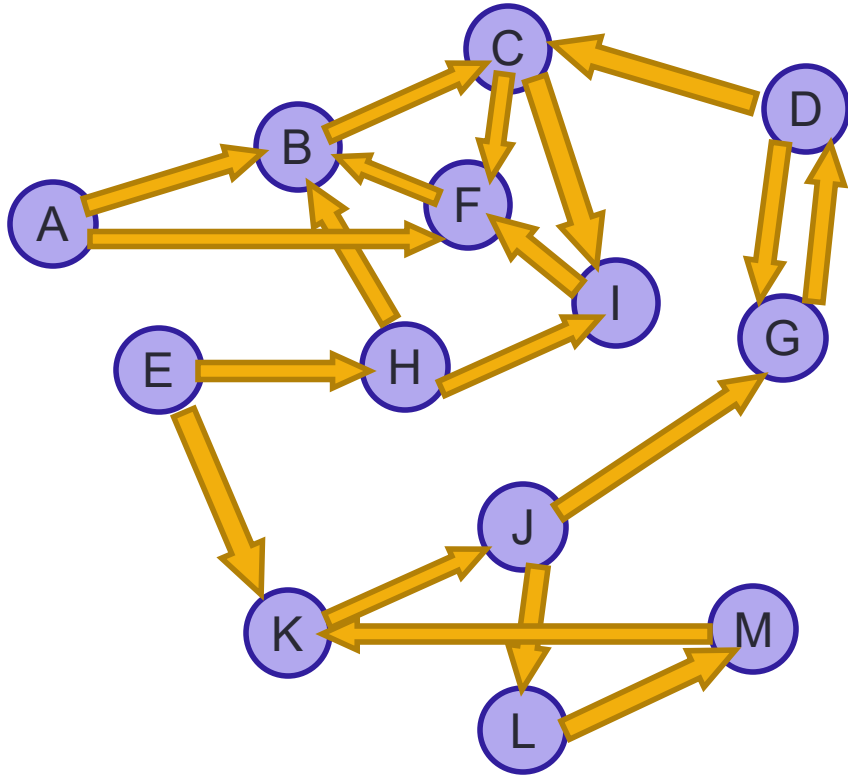
These equivalence classes are called strongly connected components.

Strongly connected component

What are the strongly connected components of this graph?



Strongly connected components as vertices. (Meta-graph)



Directed Graphs as DAGs of SCCs

Every Directed graph is a DAG of its strongly connected components.

Some SCCs are sink SCCs and some are source SCCs.

Decomposition

There is a linear time algorithm that decomposes a directed graph into its strongly connected components.

If explore is performed on a vertex u , then it will visit only the vertices that are reachable by u .

What vertices will be visited when explore is performed on u if u is in a sink SCC?

Sink SCCs

If explore is performed on a vertex that is in a sink SCC, then only the vertices from that SCC will be visited.

This suggests a way to look for SCCs.

- Start explore on a vertex in a sink SCC and visit its SCC.
- Remove the sink SCC from the graph and repeat.

Source SCCs

Ideally we would like to find a vertex in a sink SCC.
Unfortunately, there is not a direct way to do this.

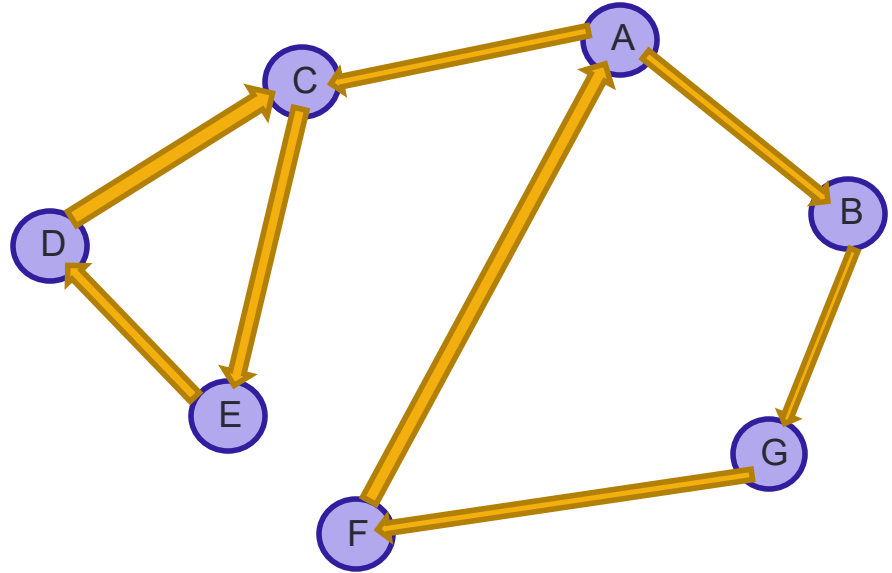
Source SCCs

However, there is a way to find a vertex in a source SCC.

The vertex with the greatest post number in any DFS output tree belongs to a source SCC.

The vertex with the least post number in a dfs output does not necessarily belong to a sink SCC.

Example of low post number not in a sink.



Vertices in Source SCCs

The vertex with the greatest post number in any DFS output tree belongs to a source SCC.

To prove this, we will state a more general property:

If C and C' are strongly connected components and there is an edge from a vertex in C to a vertex in C' then the highest post number in C is greater than the highest post number in C'

Vertices in S SCCs

The vertex with the greatest post number in any DFS output tree belongs to a source SCC.

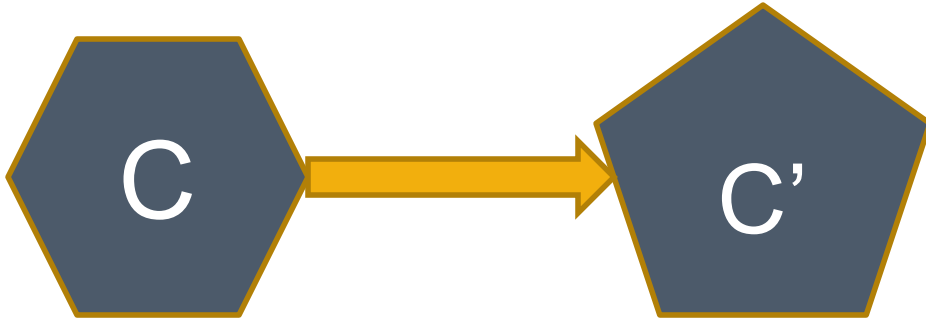
To prove this, we will state a more general property:

If C and C' are strongly connected components and there is an edge from a vertex in C to a vertex in C' then the highest post number in C is greater than the highest post number in C'

Proof

Case 1: DFS searches C before C' :

Then at some point dfs will cross into C' and visit every edge in C' then it will retrace its steps until it gets back to the first node in C it started with and assign it the highest post number

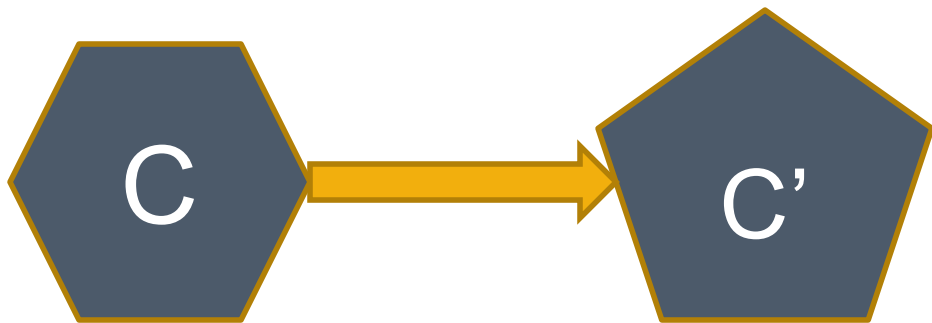


Proof

Case 2: DFS searches C' before C :

Then DFS will visit all vertices of C' before getting stuck and assign a post number to all vertices of C' .

Then it will visit some vertex of C later and assign post numbers to those vertices.



Corollary

The strongly connected components can be linearized by arranging them in decreasing order of their highest post numbers.

How to find sink SCCs

Given a graph G , let G^R be the reverse graph of G .
Then the sources of G^R are the sinks of G ,

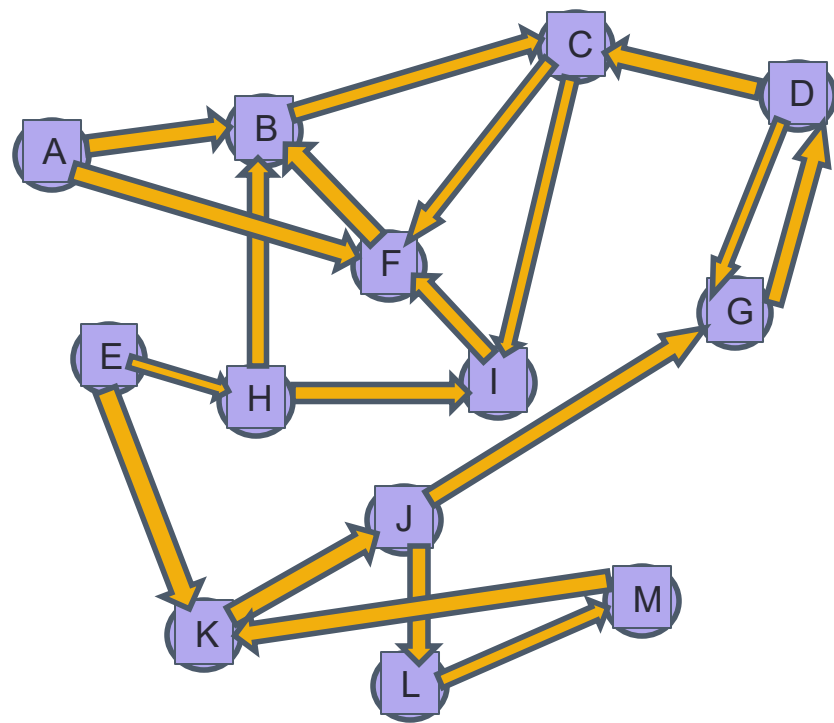
So if we perform DFS on G^R then the vertex with the highest post number is in a source. This means that this vertex will be in a sink of G .

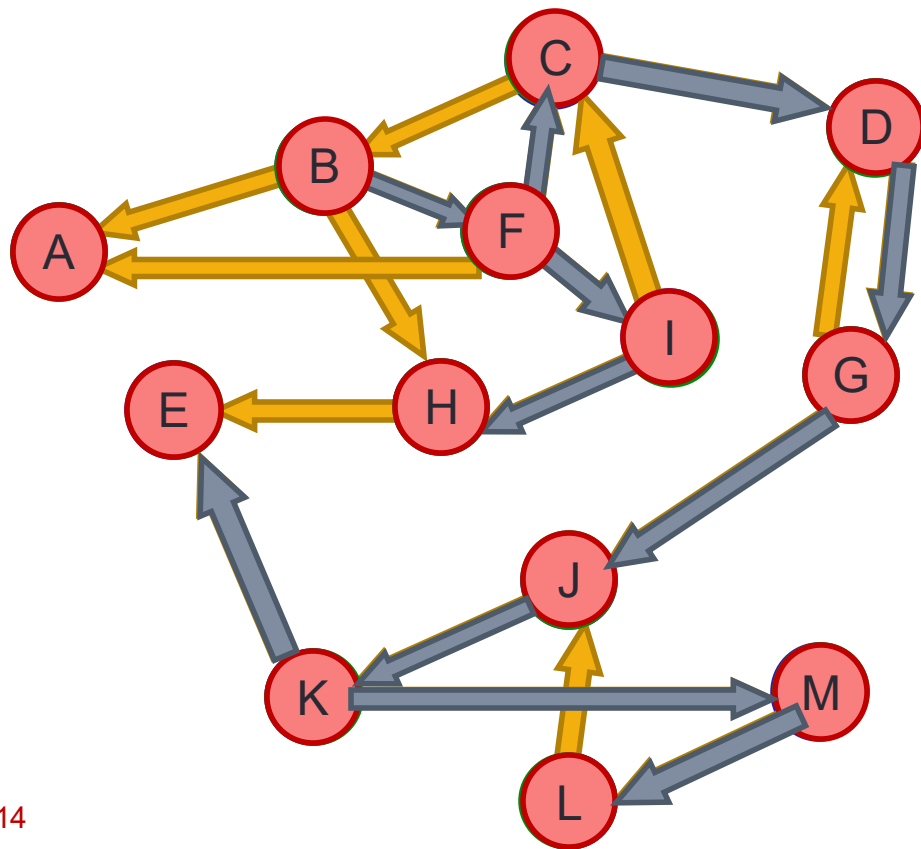
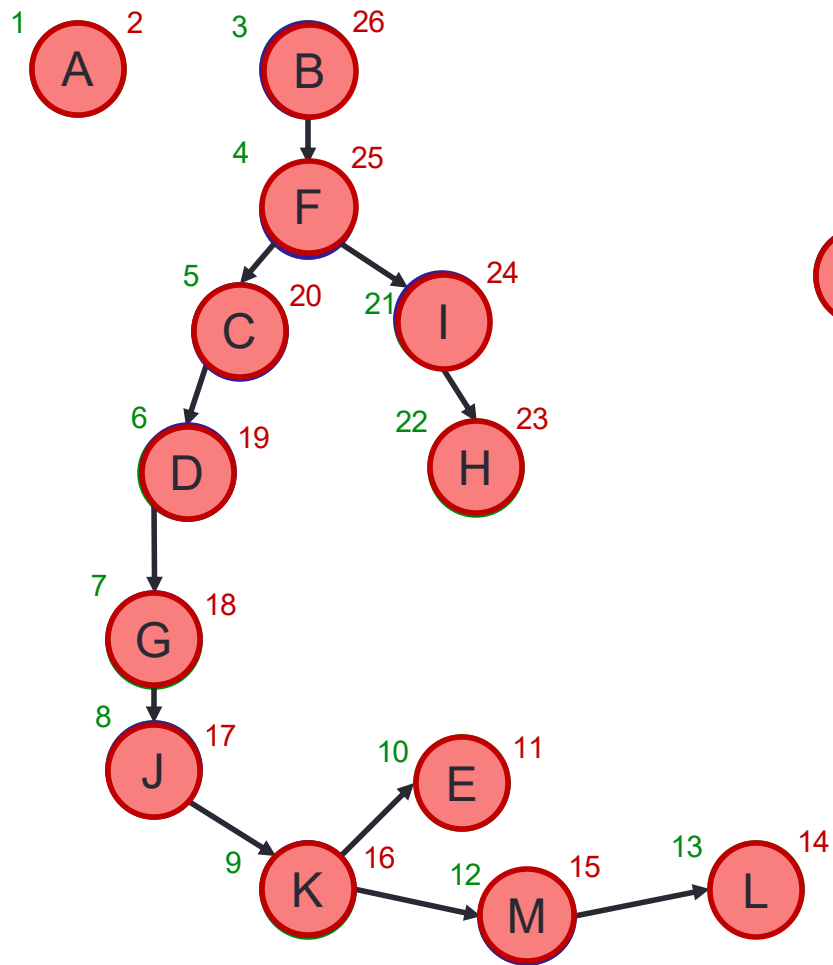
So start with this vertex and explore the SCC.

Then the vertex with the next greatest post number in G^R is in the next SCC in linear order so start with that one next.

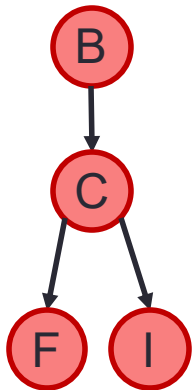
How to decompose a graph into its SCCs:

- Construct G^R .
- Run DFS on G^R and keep track of the post numbers.
- Run DFS on G and order the vertices in decreasing order of the post numbers from the previous step.
Every time DFS increments **cc**, you have found a new SCC!!





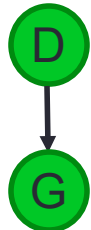
cc = 1



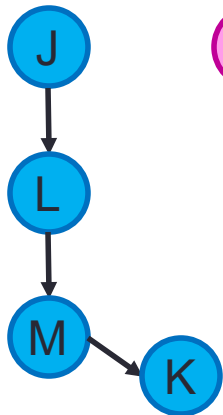
cc = 2



cc = 3



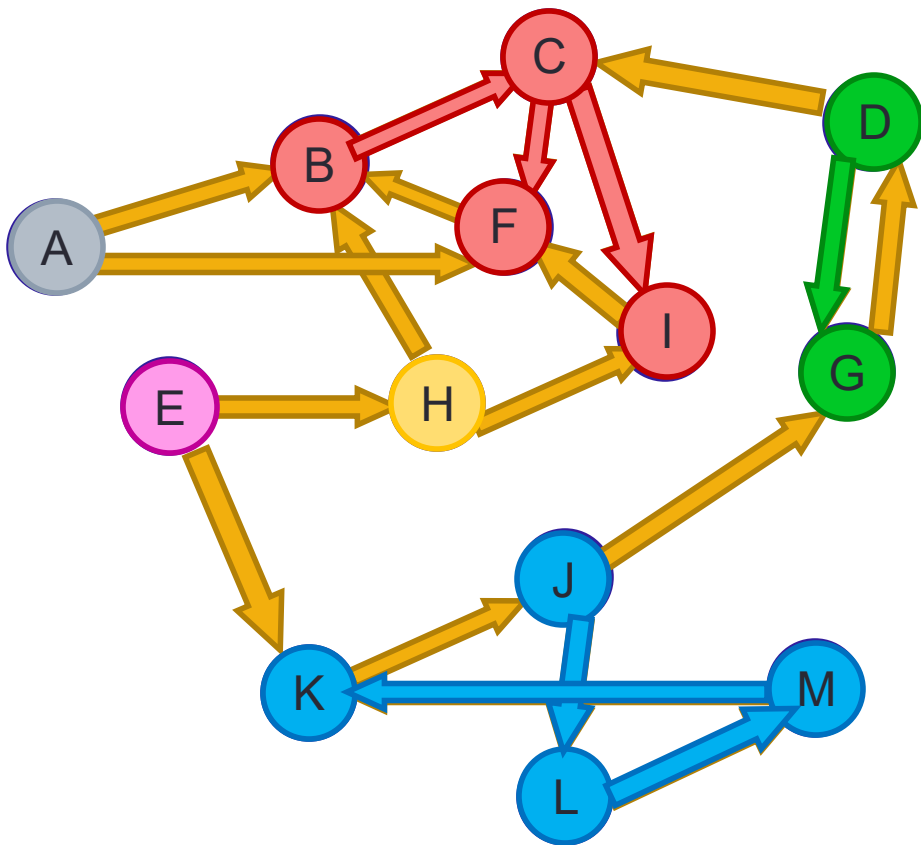
cc = 4



cc = 5



cc = 6



B, F, I, H, C, D, G, J, K, M, L, E, A

How to decompose a graph into its SCCs:

- Run DFS on G^R and keep track of the post numbers.
- Run DFS on G and order the vertices in decreasing order of the postnumbers from the previous step. Every time DFS increments **cc**, you have found a new SCC!!

How long does this take?

I claim it is linear time for each step and so it is linear time in general