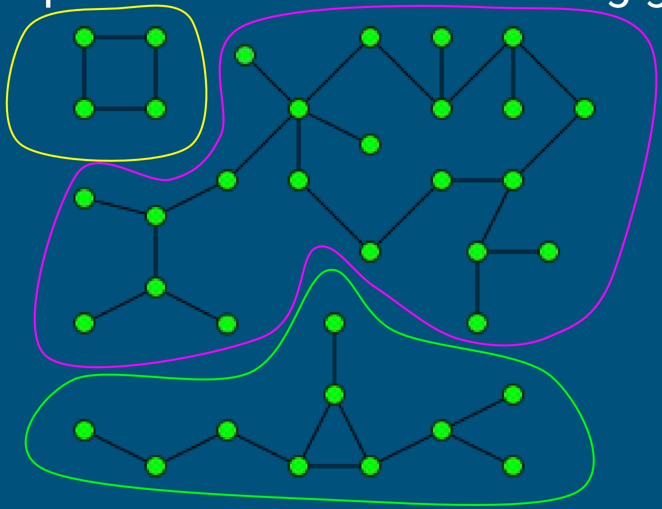
Strongly Connected Components

UF Programming Team

Connected Components

A ***connected component*** (or just ***component***) of an undirected graph is a subgraph in which any two vertices are connected to each other by paths

How many connected components are in the following graph?



How do we find connected components?

Any ideas?

→ BFS/DFS

We'll use the following algorithm

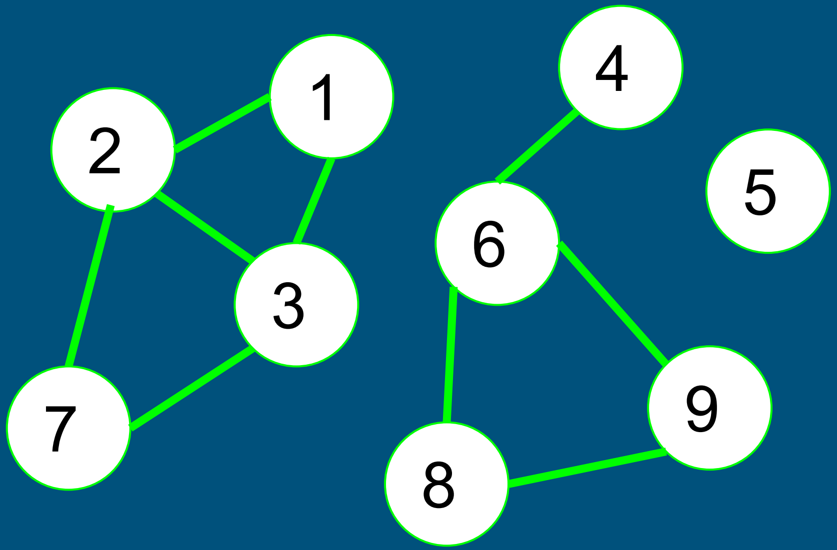
1) Find any node, ***k***, in the graph that is unvisited
2) Start a DFS from ***k*** and see what nodes we can mark as visited during the traversal
3) If there are still nodes left in the graph that are unvisited, go back to step (1)

Example

Let's determine what the connected components of the following graph are

Since all nodes are currently unvisited, we can pick any of them to start a DFS from

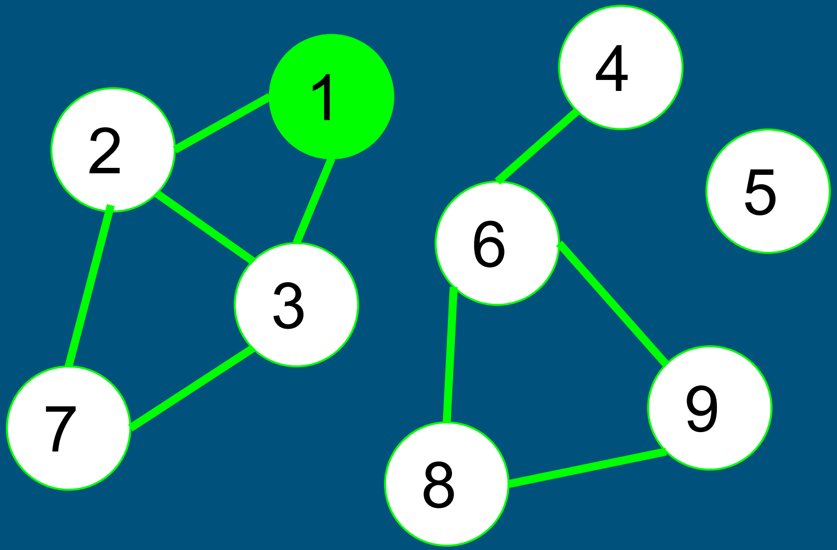
We'll go in ascending order to make things easier, so let's start at node 1



Example

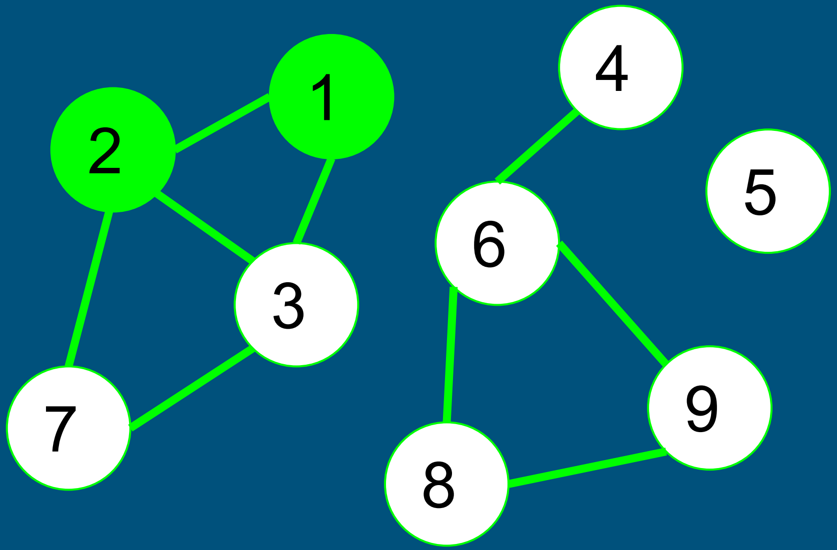
We'll mark the visited nodes with the color green

From node 1, we just need to perform a DFS and see what other nodes we can visit



Example

From node 1 we can visit node 2

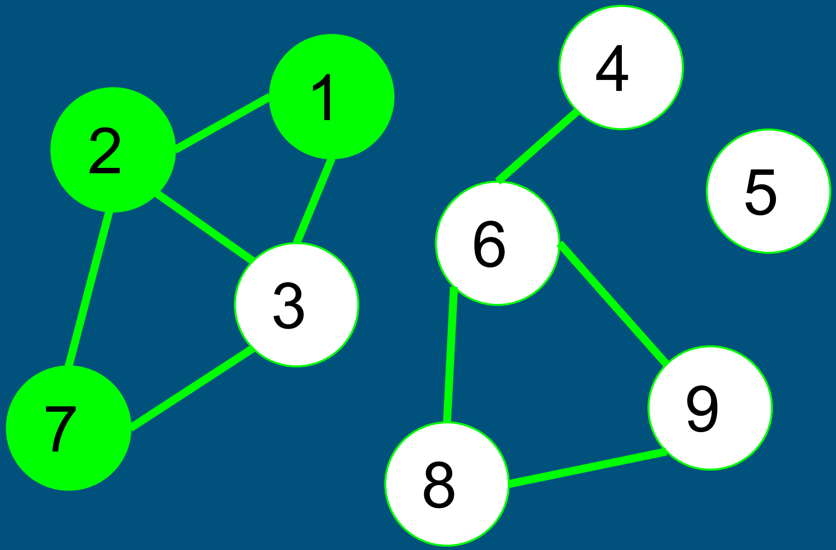


Example

From node 2 we can visit node 7

We can’t go any deeper from node 7, so we backtrack to node 2

We also can’t go any deeper from node 2, so we backtrack to node 1



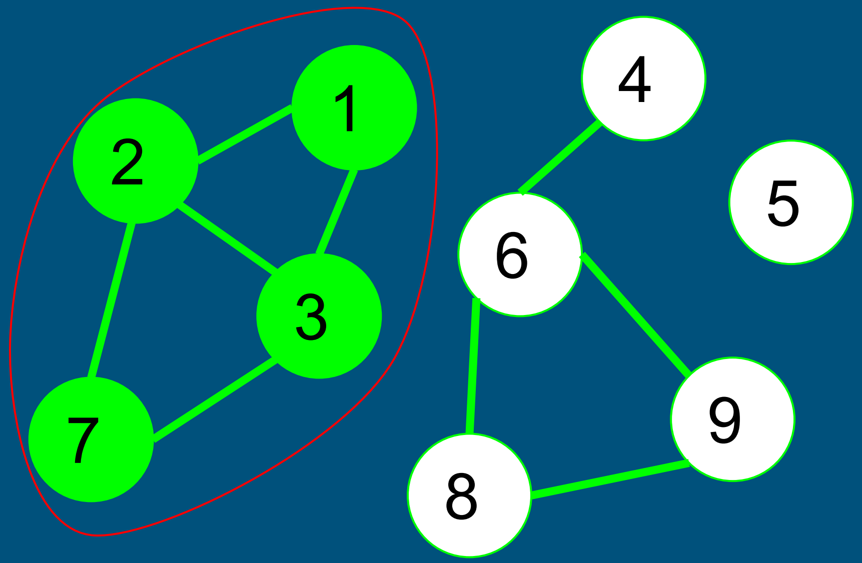
Example

From node 1 we can visit node 3

We can't go any deeper from node 3, so we backtrack to node 1

We can't go any deeper from node 1, so we are finished with DFS

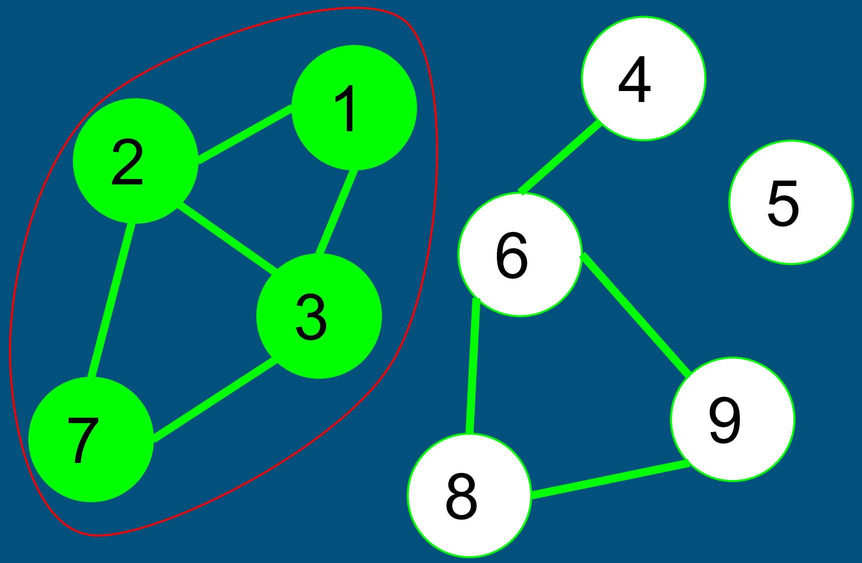
The nodes which are colored green form a connected component!



Example

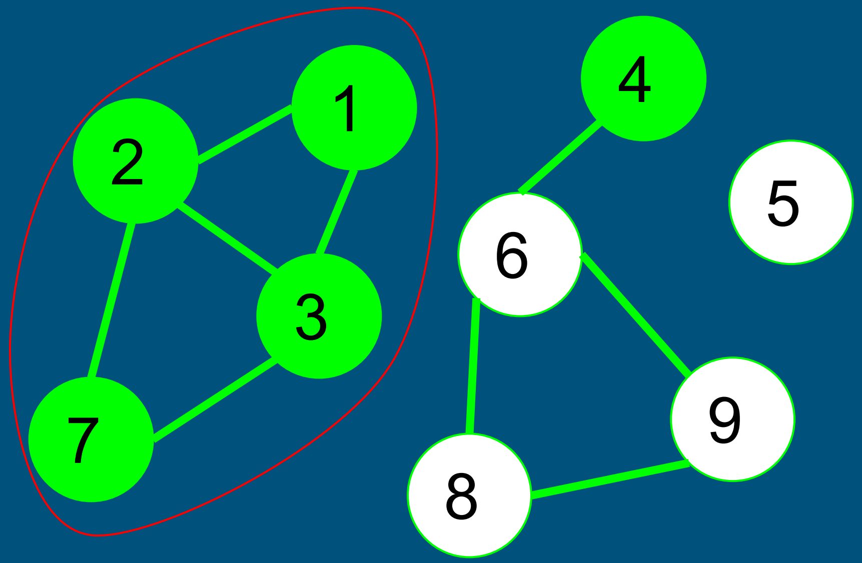
Now we search for nodes which we haven't visited yet to start another DFS

We see that node 4 hasn't been visited yet, so we'll start a DFS from there



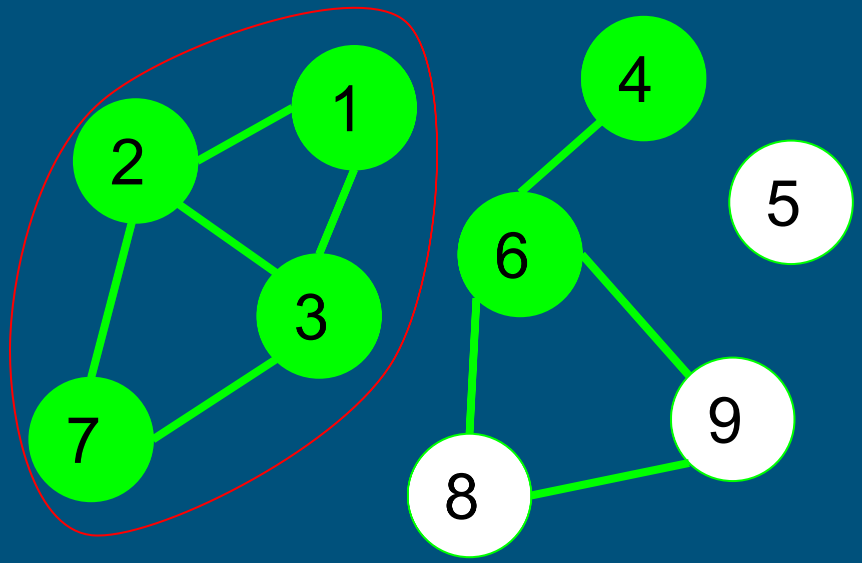
Example

From node 4 we can visit node 6



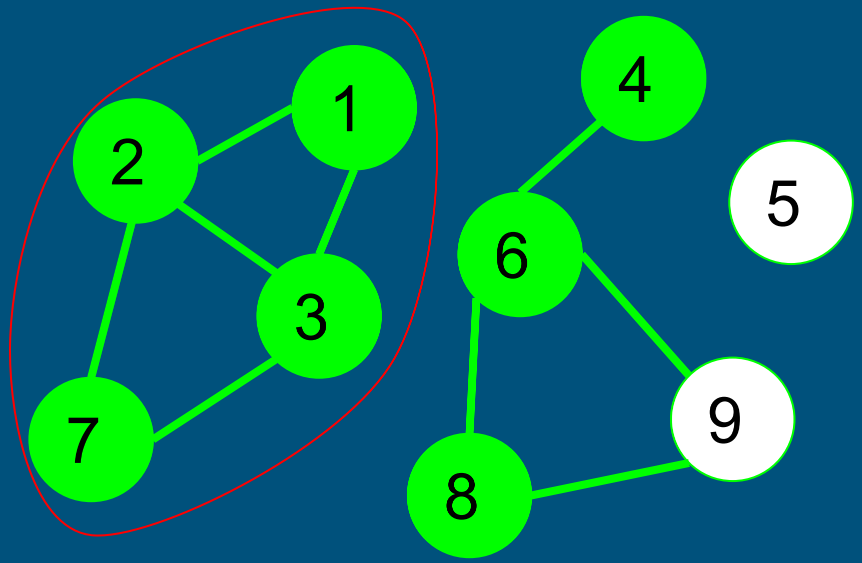
Example

From node 6 we can visit node 8



Example

We can’t go any deeper from node 8, so we backtrack to node 6



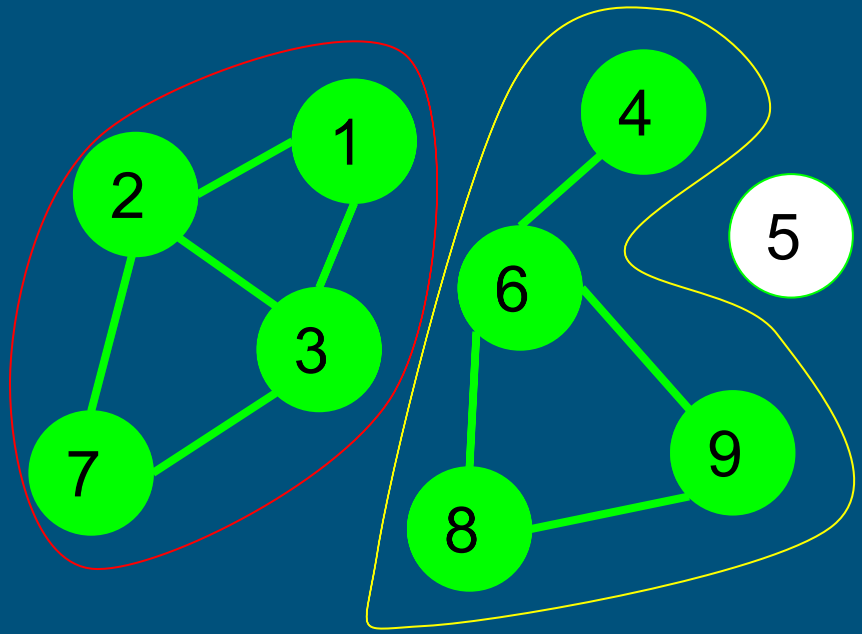
Example

From node 6 we can visit node 9

We can't go any deeper from node 9, so we backtrack to node 6

We can't go any deeper from node 6, so we backtrack to node 4

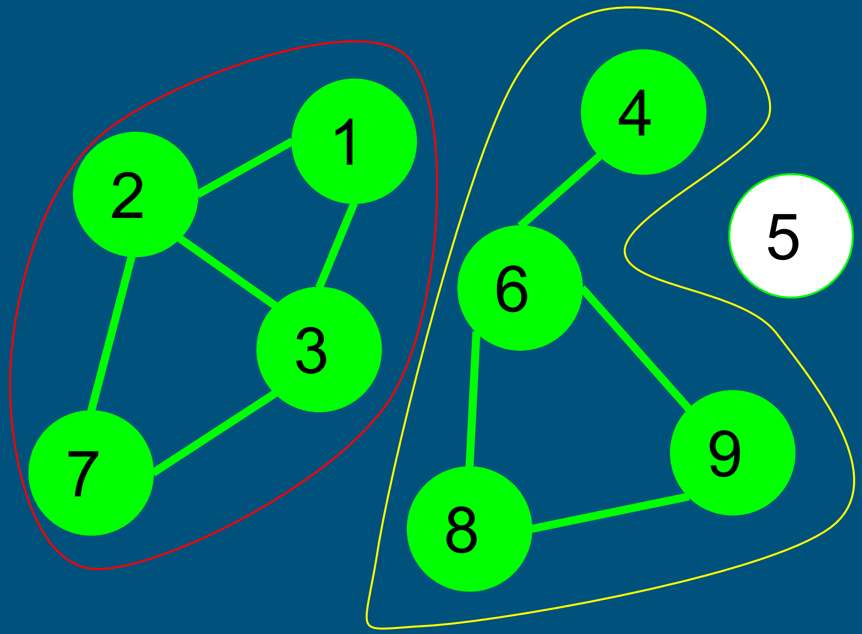
We can't go any deeper from node 4, so we are finished with DFS



Example

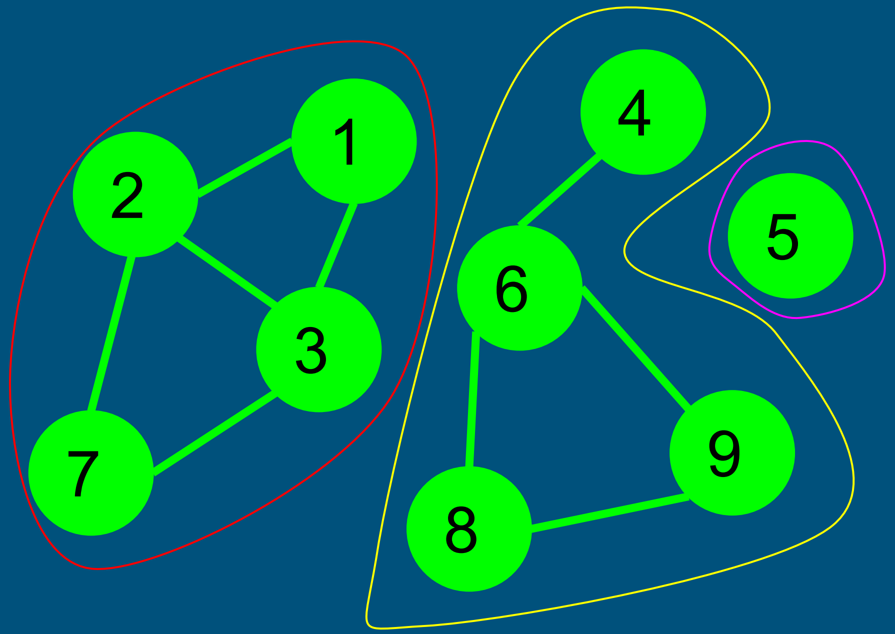
Now we search for nodes which we haven’t visited yet to start another DFS

We see that node 5 hasn’t been visited yet, so we’ll start a DFS from there



Example

Node 5 is not connected to any other nodes, so it forms its own connected component



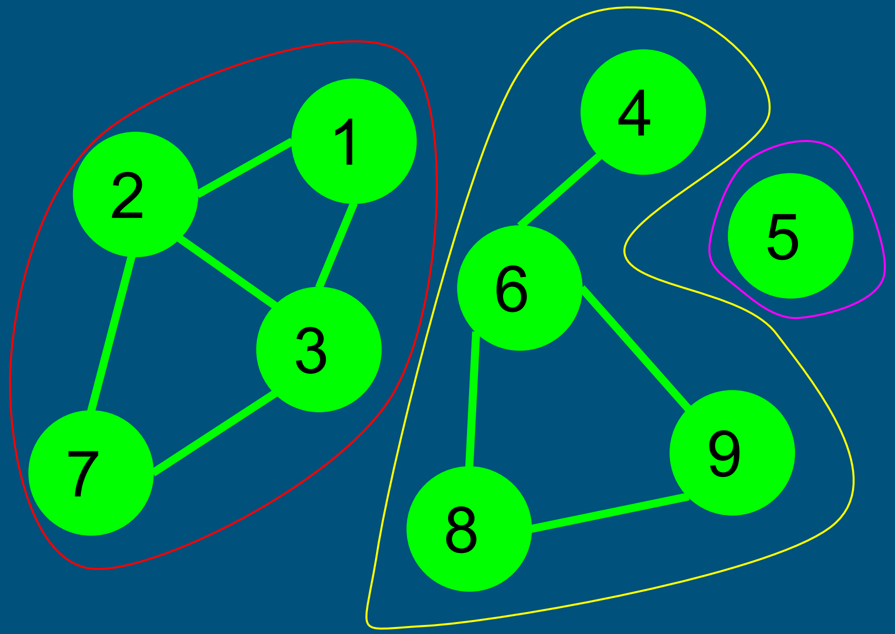
Example

We have now found all connected components of this graph!

{1, 2, 3, 7}

{4, 6, 8, 9}

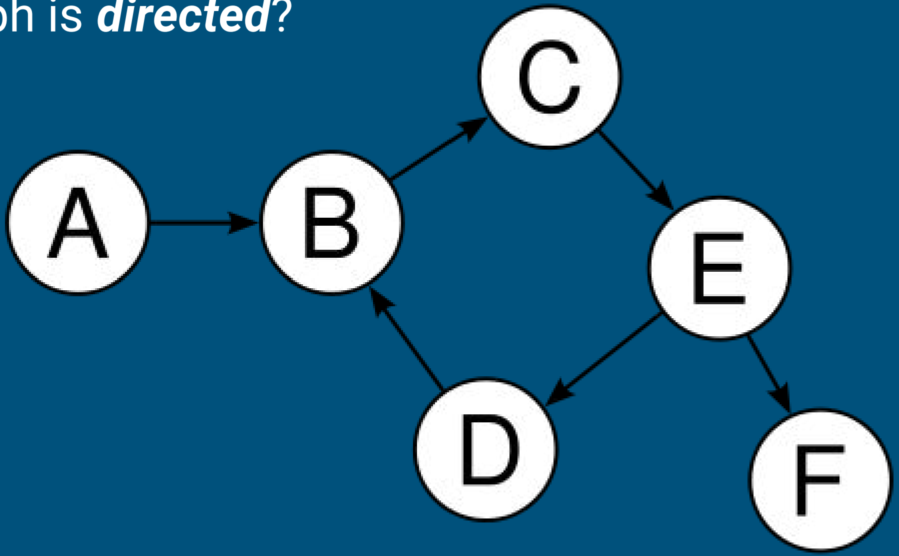
{5}



Directed graph

Finding the connected components of an undirected graph was pretty straightforward, but what if the graph is ***directed***?

How do we go about finding the connected components of this graph?

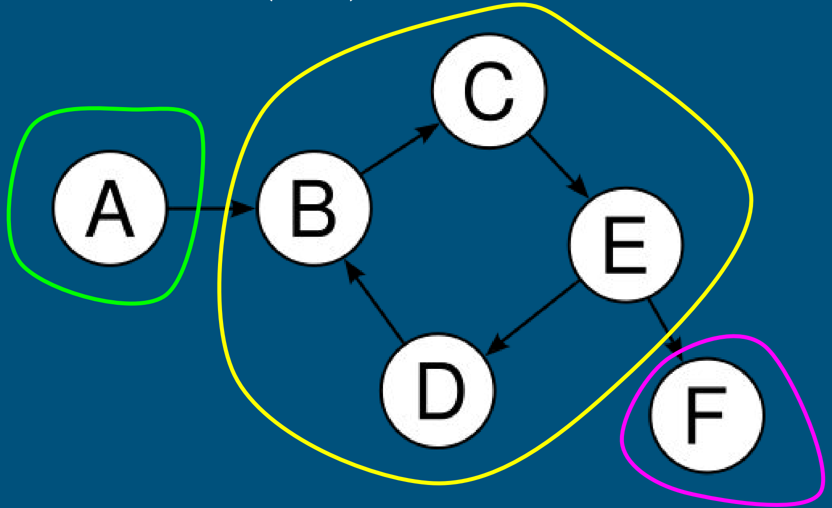


Strongly Connected Components

A ***strongly connected component*** (***SCC***) of a directed graph is a subgraph such that every vertex is reachable from every other vertex

In other words, if we can pick any pair of vertices (***u, v***) in an SCC, we can find a path from ***u*** to ***v***, and vice versa

How many SCCs are there in this graph?



Finding SCCs

Any ideas?

→ Still DFS (with some modifications)

There are two known algorithms to find SCCs in a directed graph

1) Tarjan's algorithm
2) Kosaraju's algorithm

We'll be going over Kosaraju's algorithm today

Kosaraju's Algorithm

Kosaraju's algorithm works as follows:

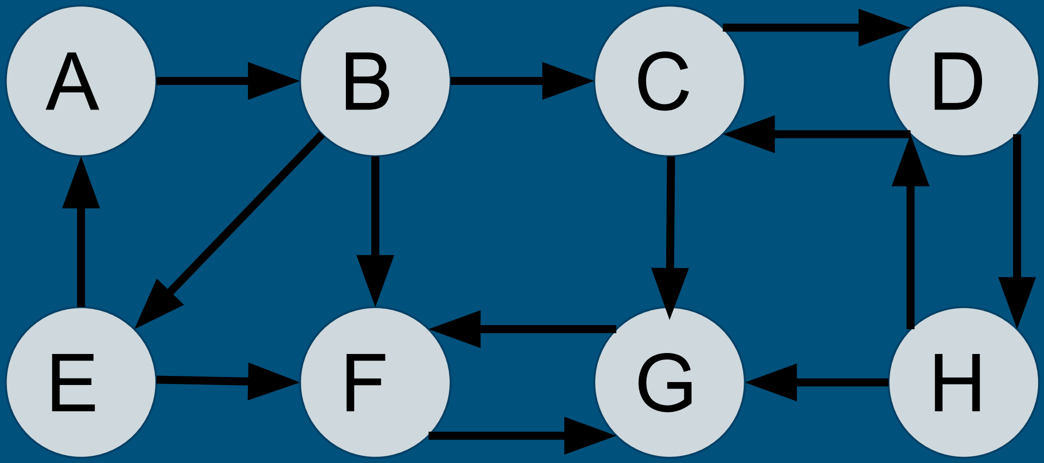
1) Let ***G*** be a directed graph and ***S*** be an empty stack
2) While ***S*** does not contain all vertices:
   a) Choose any vertex ***v*** not in ***S***. Perform a DFS starting at ***v***. Each time that DFS finishes expanding a vertex ***u***, push ***u*** onto ***S***
3) Reverse the directions of all edges in ***G*** (this gives us the transpose graph)
4) While ***S*** is nonempty:
   a) Pop the top vertex ***v*** from ***S***. Perform a DFS starting at ***v*** in the transpose graph. The set of visited vertices will give the SCC containing ***v***; record this and remove all these vertices from the graph ***G*** and the stack ***S***

Example

Let's perform Kosaraju's algorithm on the follow directed graph in order to find all SCCs

Since all vertices are currently unvisited, we can pick any vertex to start a DFS from

We'll start a DFS from A

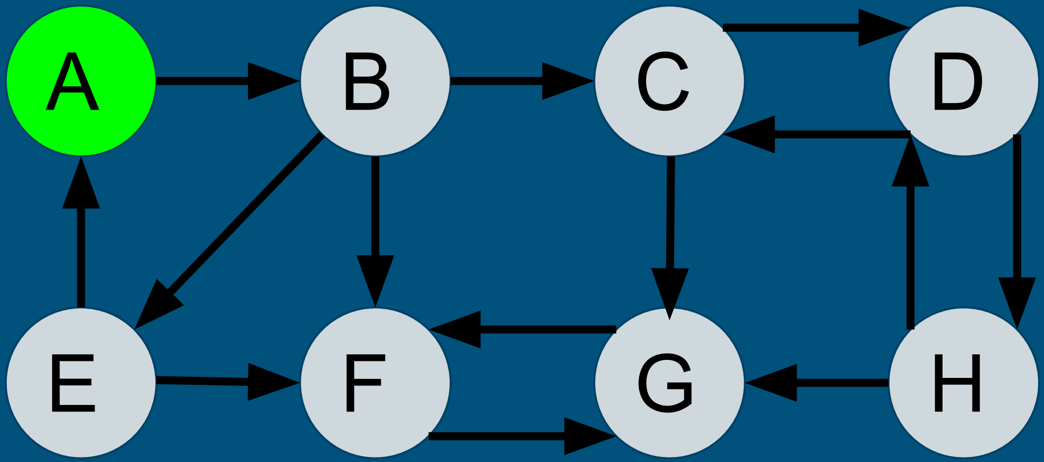


Stack: { }

Example

We will denote a visited vertex by the color green

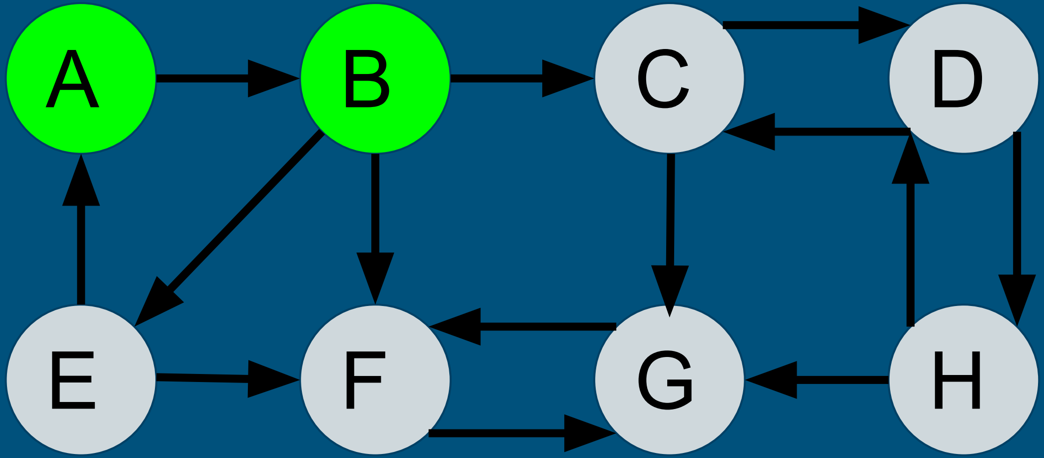
We visit vertex A and visit one of its neighbors; we'll visit vertex B



Stack: { }

Example

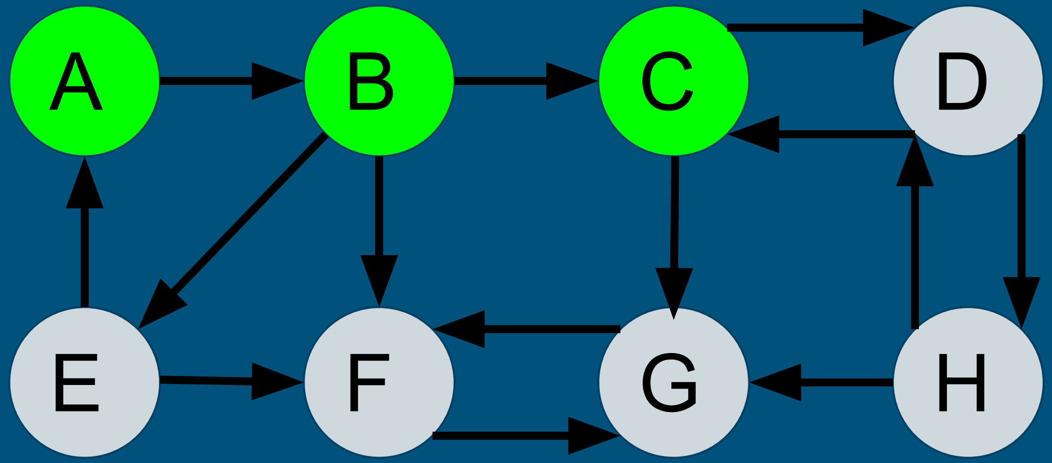
From vertex B, we can visit vertex C



Stack: { }

Example

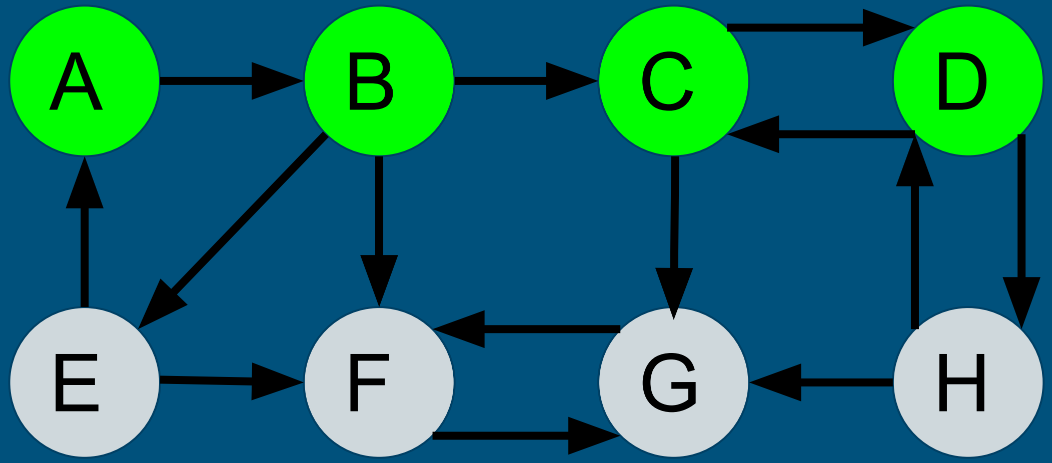
From vertex C, we can visit vertex D



Stack: { }

Example

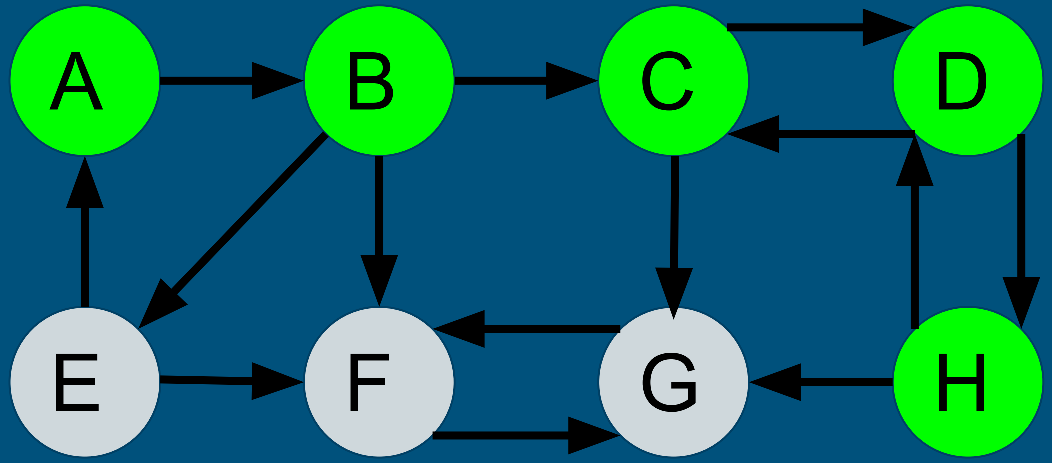
From vertex D, we can visit vertex H



Stack: { }

Example

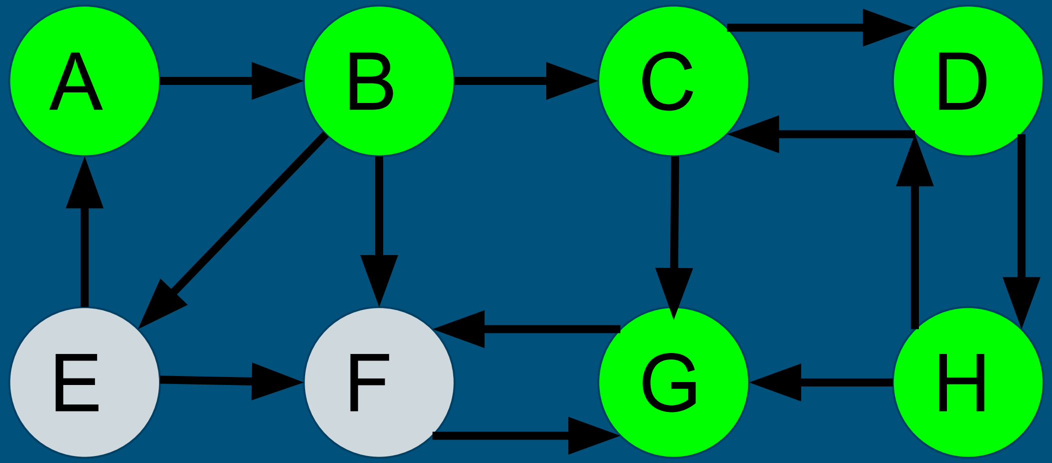
From vertex H, we can visit vertex G



Stack: { }

Example

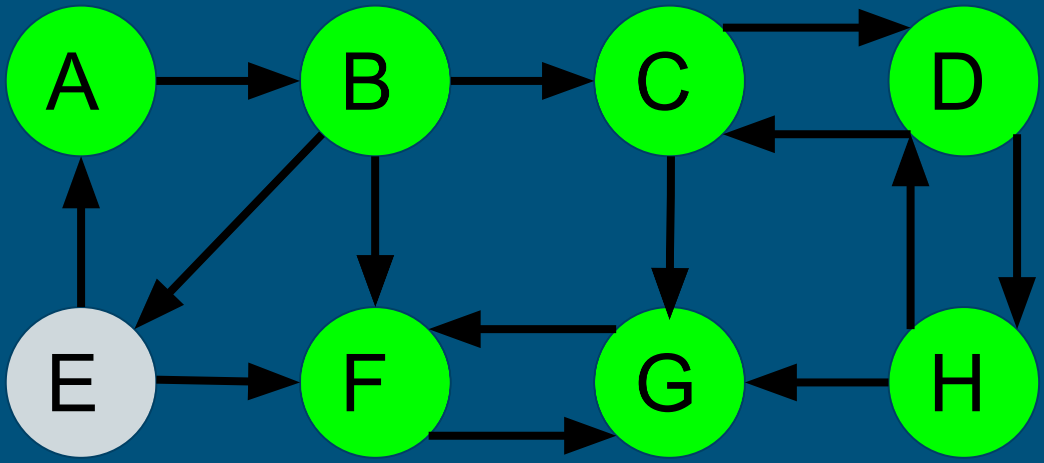
From vertex G, we can visit vertex F



Stack: { }

Example

There are no more vertices for us to visit, so we need to add vertices to the stack while we backtrack

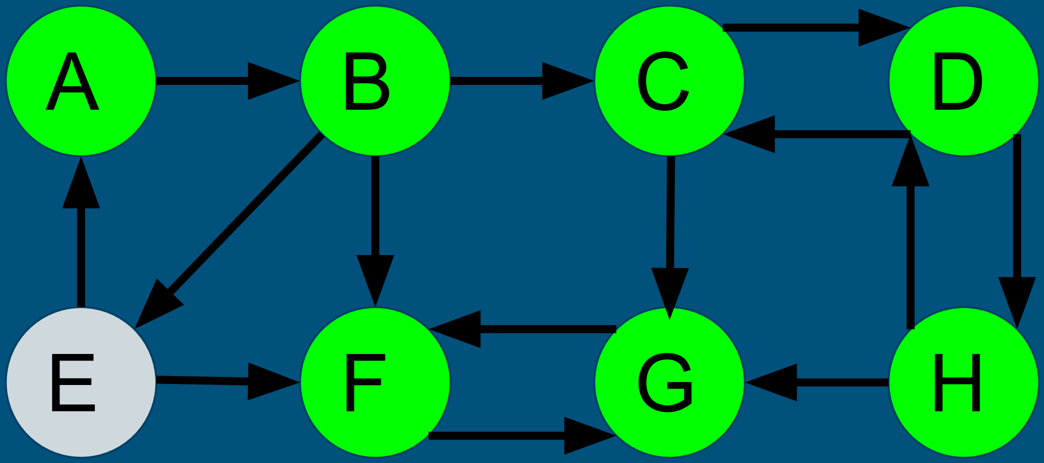


Stack: { }

Example

We stop the backtracking at B since it still has neighbors that haven't been visited

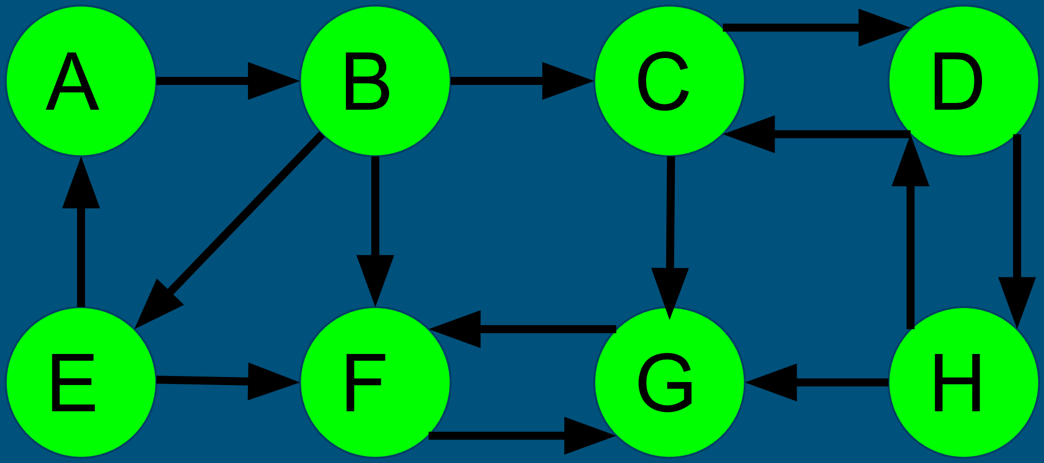
From vertex B, we can visit vertex E



Stack: {F, G, H, D, C}

Example

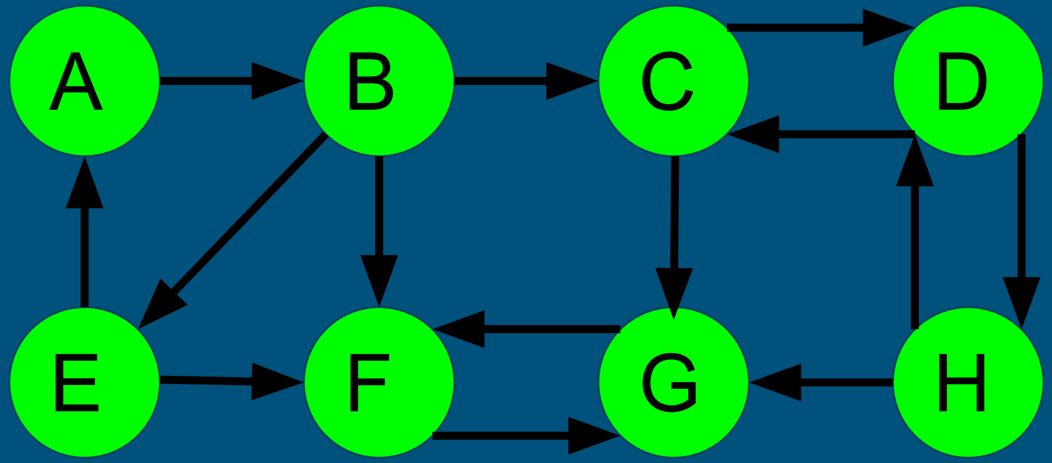
There are no more vertices left to visit, so we backtrack to B, adding the vertices to the stack along the way



Stack: {F, G, H, D, C}

Example

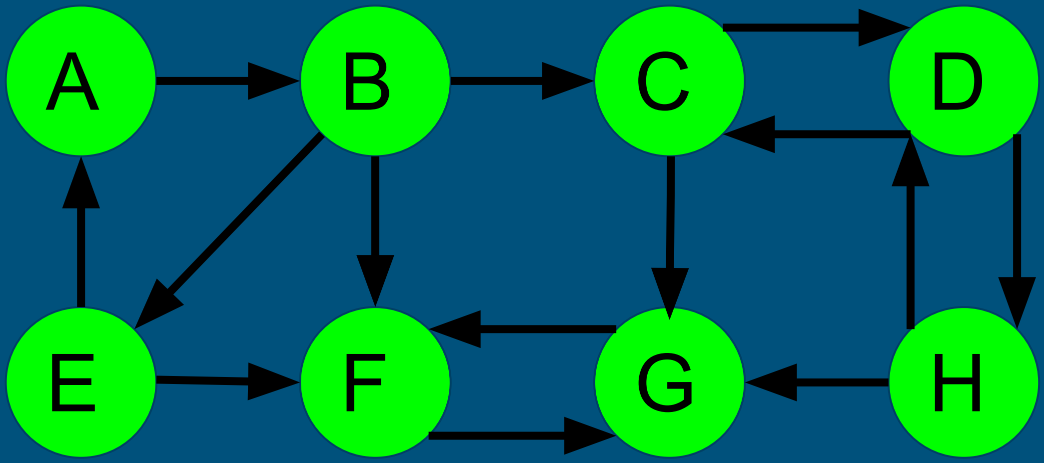
When we get back to our starting vertex, and it has no more vertices left to visit, we add vertex A to the stack and stop the DFS process



Stack: {F, G, H, D, C, E, B}

Example

Now that we are done and have our stack, we need to reverse all of the edges in the graph to construct a transpose graph



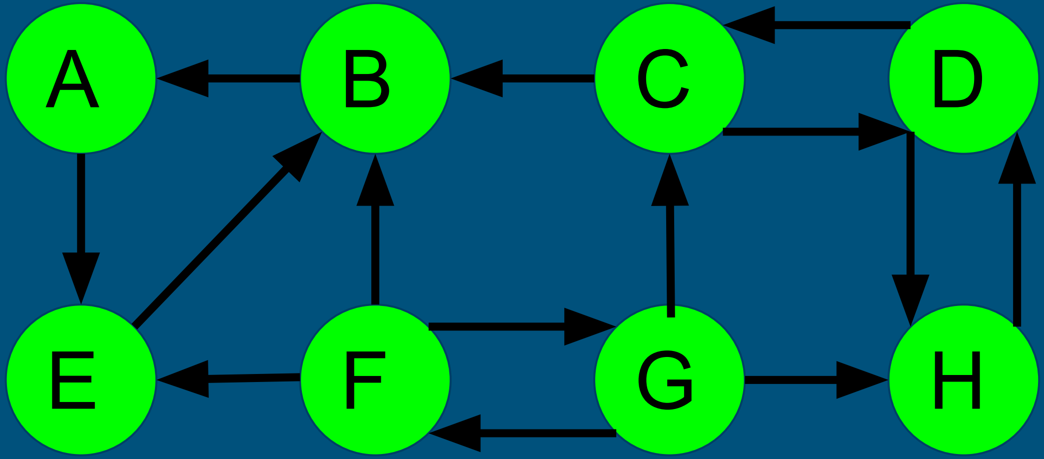
Stack: {F, G, H, D, C, E, B, A}

Example

Now that we have reversed the edges, we are going to pop the stack to get some vertex ***u***

We will perform a DFS from ***u*** and see which unvisited vertices we can reach from ***u***

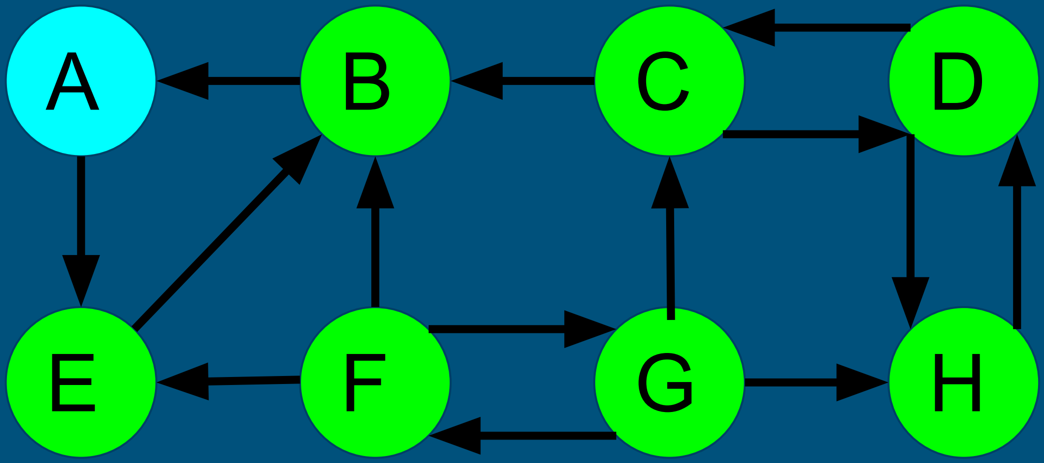
All vertices that we reach are included in an SCC along with ***u***



Stack: {F, G, H, D, C, E, B, A}

Example

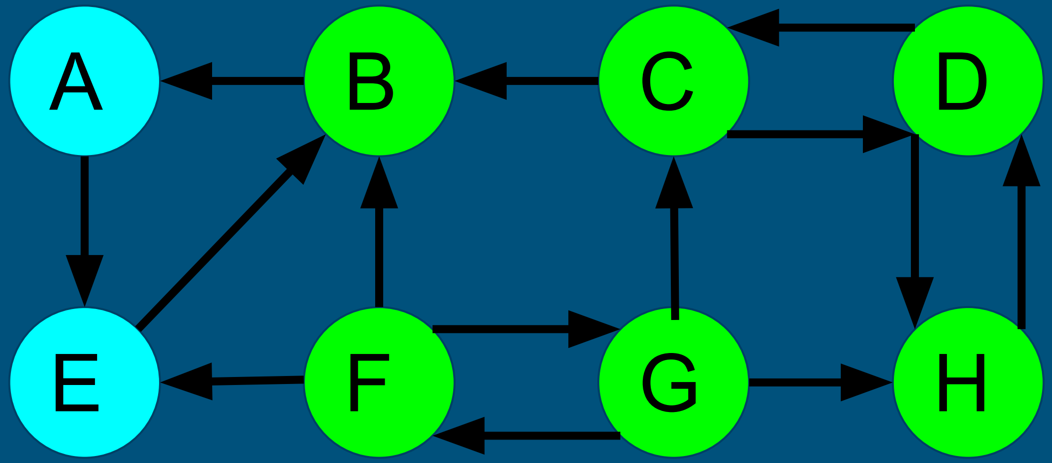
From vertex A we can visit E



Stack: {F, G, H, D, C, E, B}

Example

From vertex E we can visit B



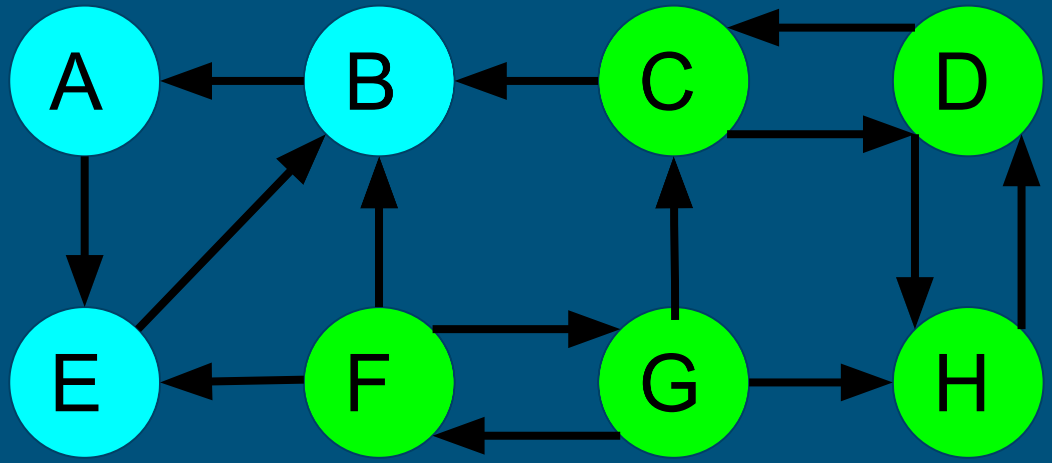
Stack: {F, G, H, D, C, E, B}

Example

When we reach vertex B, there are no more vertices for us to look at, so we backtrack

Vertex E has no other vertices to look at, so we backtrack

Vertex A has no other vertices to look at, so we are done with DFS

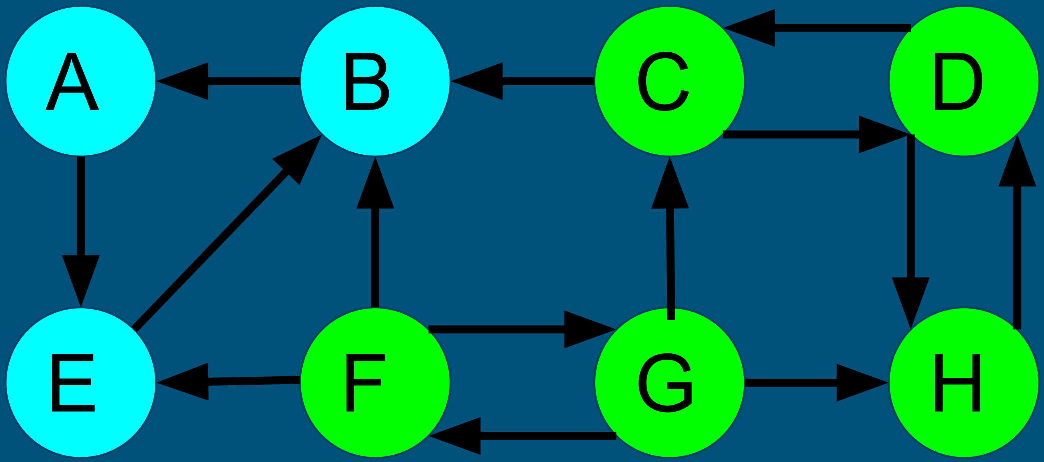


Stack: {F, G, H, D, C, E, B}

Example

All vertices that we visited in that DFS are contained in the same SCC

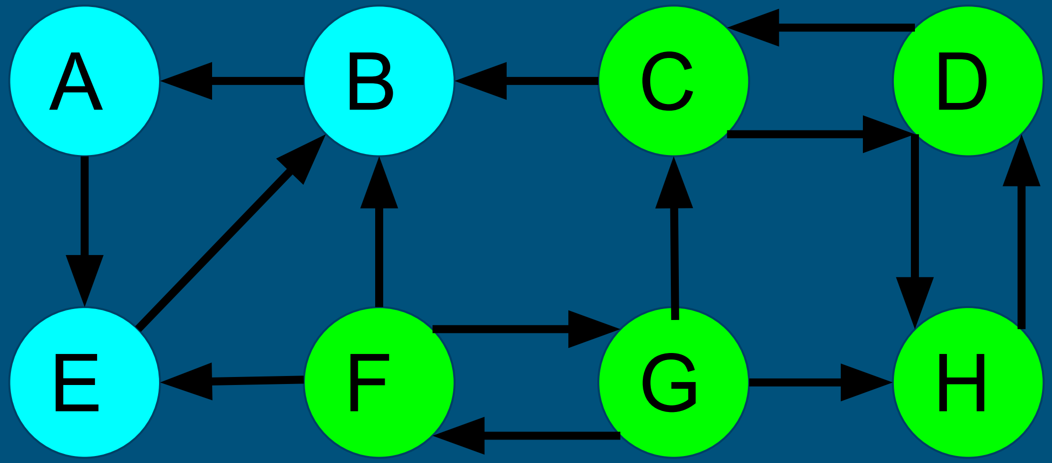
We now removed all vertices from the stack that we visited during the last DFS



Stack: {F, G, H, D, C, E, B}

Example

Now we pop the stack and perform another DFS with the new starting vertex, C

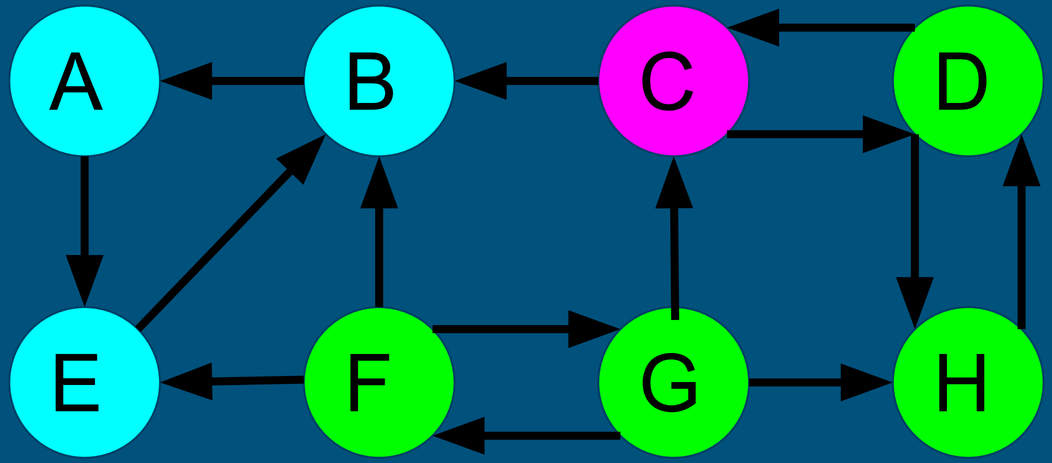


Stack: {F, G, H, D, C}

SCCs: {A, B, E}

Example

From vertex C we can visit vertex D

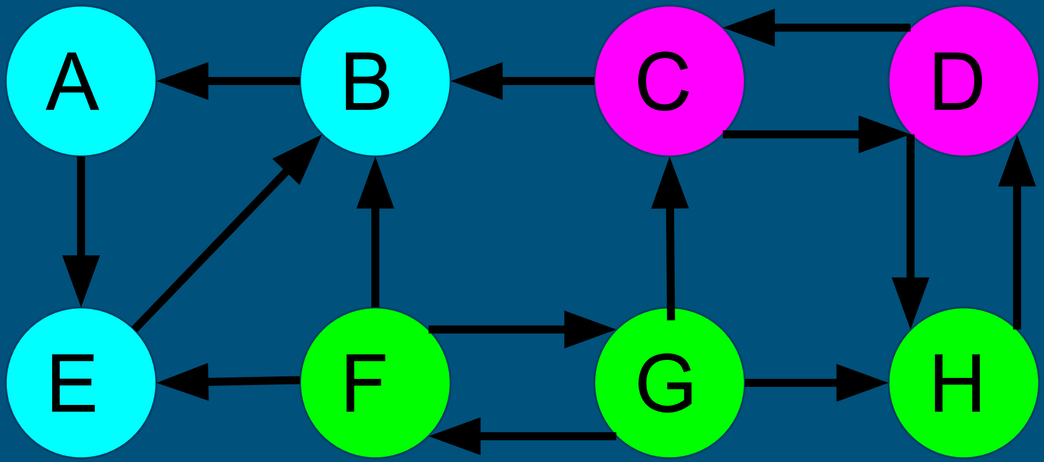


Stack: {F, G, H, D}

SCCs: {A, B, E}

Example

From vertex D we can visit vertex H



Stack: {F, G, H, D}

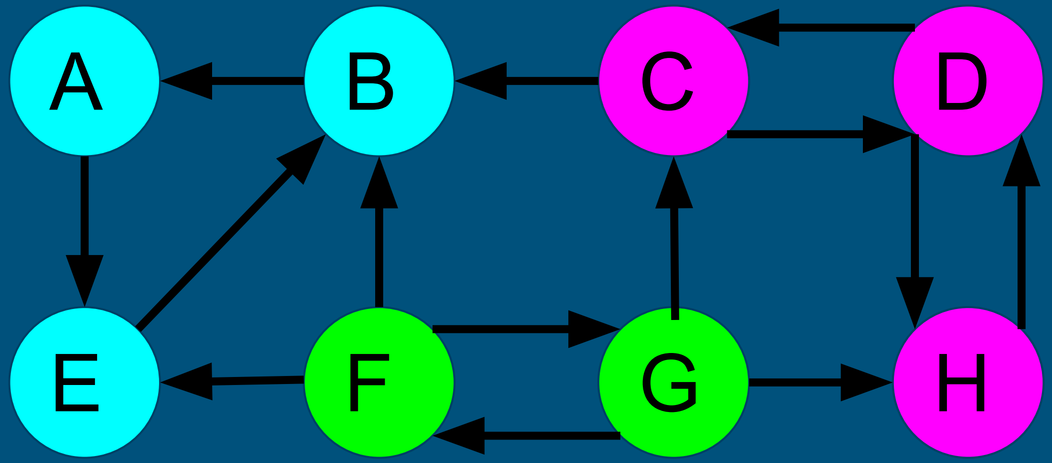
SCCs: {A, B, E}

Example

When we reach vertex H, there are no other vertices for us to look at, so we backtrack

Vertex D has no other vertices for us to look at, so we backtrack

Vertex C has no other vertices for us to look at, so we are done with the DFS

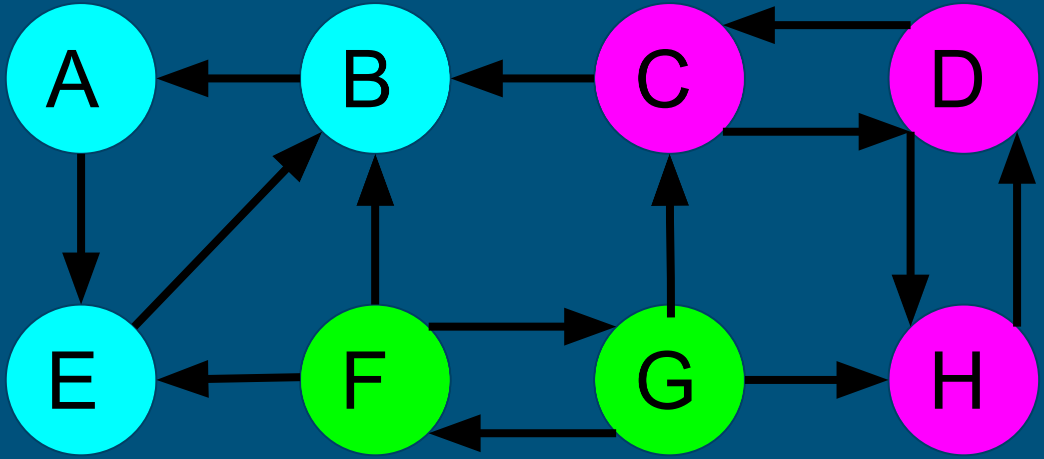


Stack: {F, G, H, D}

SCCs: {A, B, E}

Example

We placed the vertices we saw in our last DFS into their own set, and removed them from the stack

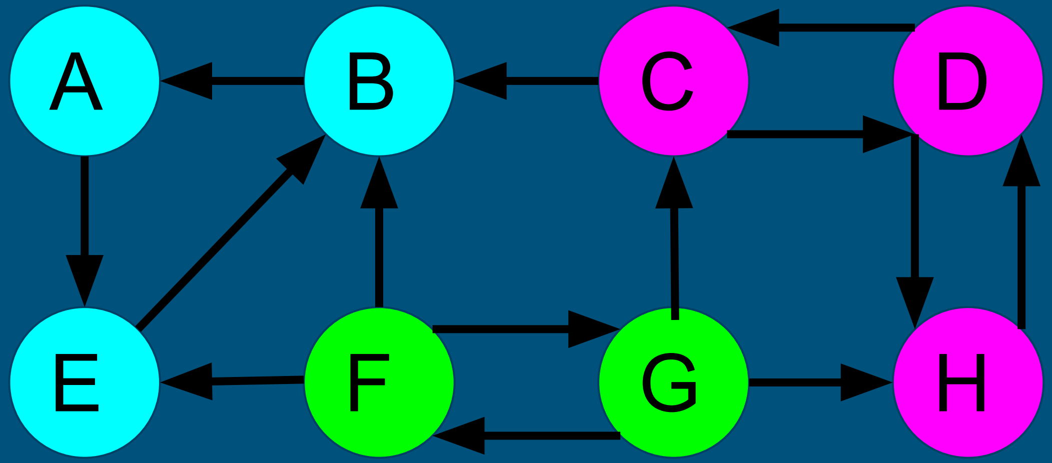


Stack: {F, G, H, D}

SCCs: {A, B, E}, {C, D, H}

Example

Again, we pop the stack and perform a DFS from vertex G

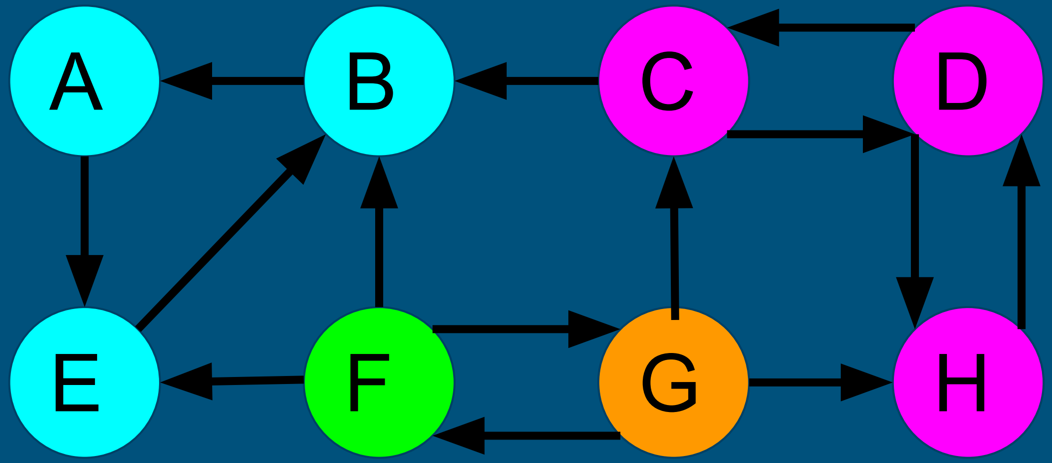


Stack: {F, G}

SCCs: {A, B, E}, {C, D, H}

Example

From vertex G we can visit vertex F



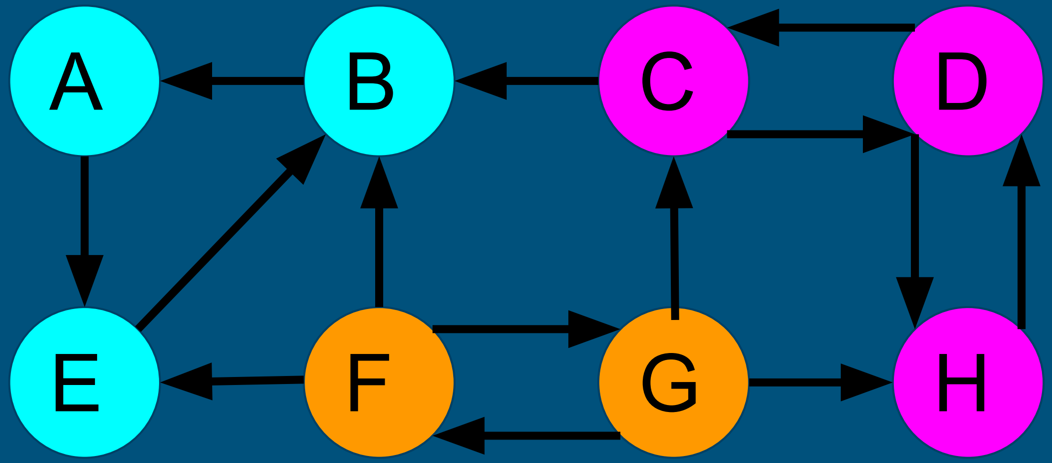
Stack: {F}

SCCs: {A, B, E}, {C, D, H}

Example

When we reach vertex F, there are no other vertices for us to look at, so we backtrack

Vertex G has no other vertices for us to look at, so we are done with the DFS

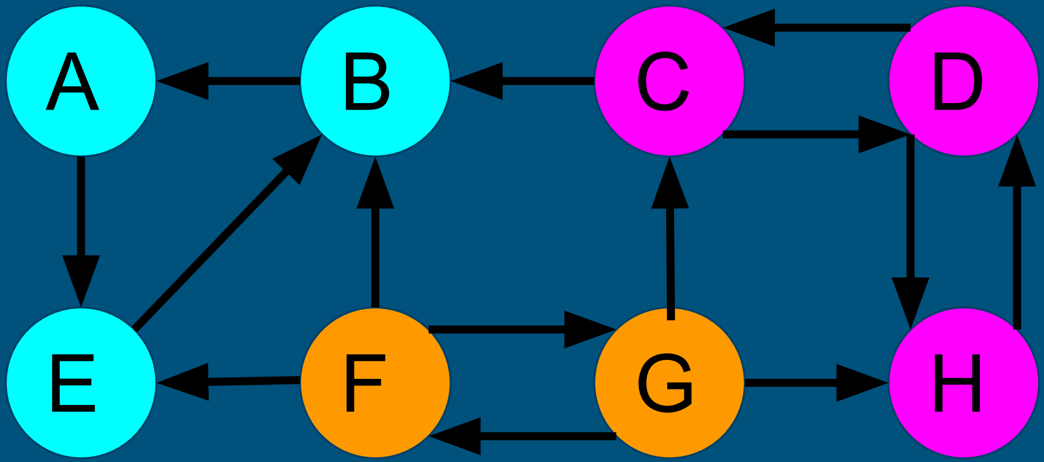


Stack: {F}

SCCs: {A, B, E}, {C, D, H}

Example

We add the vertices we looked at in the last DFS to their own set and removed them from the stack

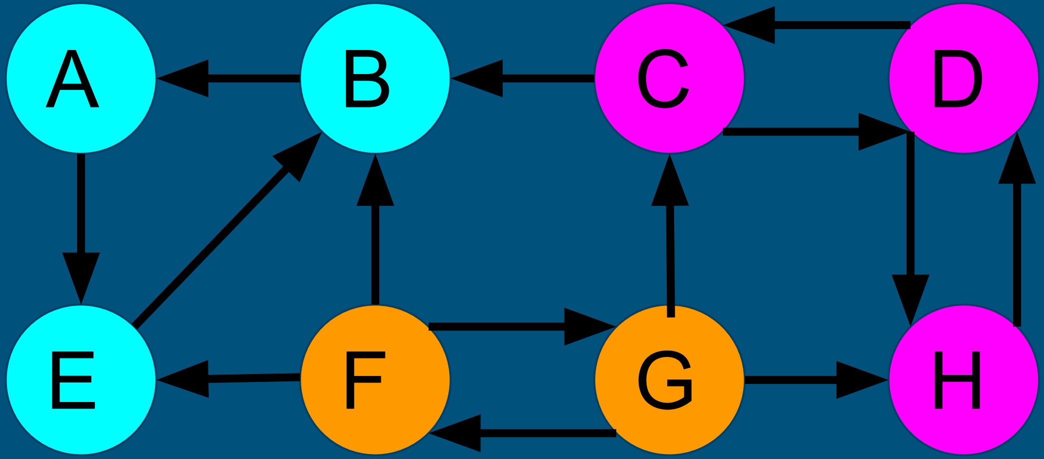


Stack: {F}

SCCs: {A, B, E}, {C, D, H}, {F, G}

Example

The stack is now empty, so we are done with Kosaraju's algorithm and now found ***all*** SCCs in the original graph!



Stack: {}

SCCs: {A, B, E}, {C, D, H}, {F, G}

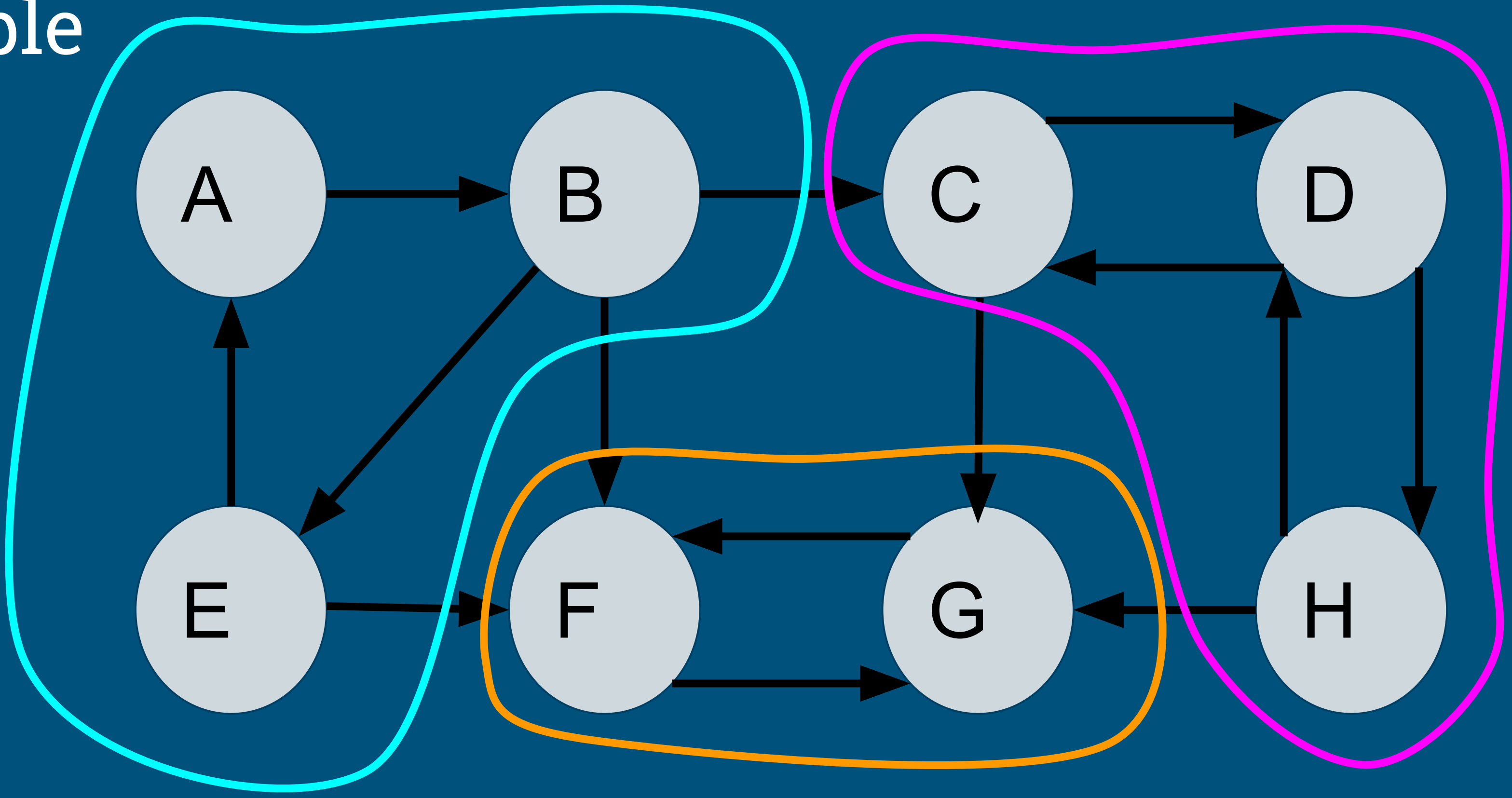
Example

SCCs

A, B, E

C, D, H

F, G



Problem

Break up into groups of 2-3 and work on the following problem

https://open.kattis.com/problems/cantinaofbabel

Other problems:

→ spoj.com/problems/MOWS (make sure to use Parser for faster I/O)

(http://pastebin.com/LaVRcsVa)