Strongly Connected Components

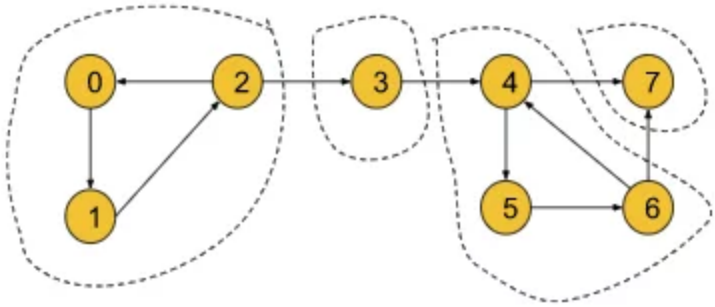
What is Strongly Connected Components ?

> 💡 The concept of strongly connected components is only defined for **directed graphs.**

A component in a directed graph is called a strongly connected component (SCC) if and only if every pair of vertices (u, v) within that component is reachable from each other, i.e., there is a directed path from u to v and a directed path from v to u.

In simple words, A directed graph is strongly connected if there is a path between all pairs of vertices.

In the following directed graph, the SCCs have been marked:



Graph having 4 strongly Connected Components (SCC)

If we take 1st SCC in the above graph, we can observe that each node is reachable from any of the other nodes.

For example, if take the pair (0, 1) from the 1st SCC, we can see that there is a directed path from 0 to 1 (0 → 1), it means 1 is reachable from 0 and There is a directed path from 1 to 0 as well (1 → 2 → 0), it means 0 is reachable from 1. Similarly, this is true for all other pairs of nodes in the SCC like (0,2), and (1,2).

By definition, ***a component containing a single vertex is always a strongly connected component***. For that vertex 3 in the above graph is itself a strongly connected component.

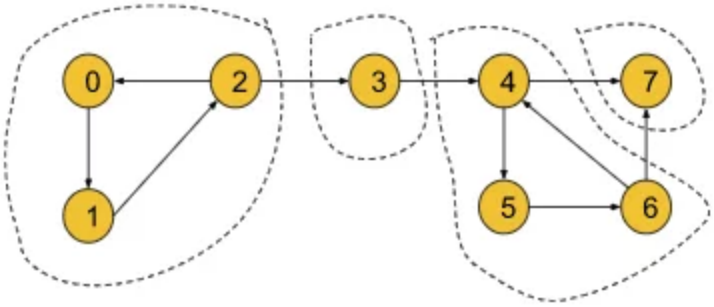
By applying this logic, we can conclude that the above graph contains 4 strongly connected components like (0,1,2), (3), (4,5,6), and (7).

These Strongly Connected Components (SCCs)can be found using **Kosaraju's Algorithm**.

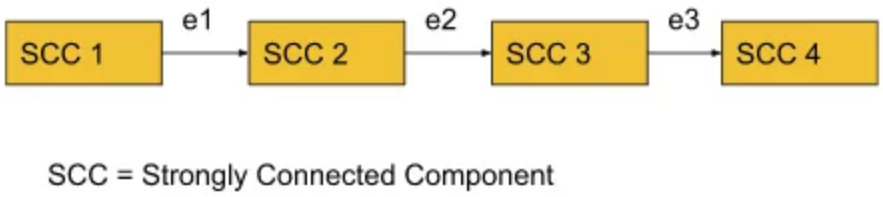
Kosaraju's Algorithm:

To find the strongly connected components of a given directed graph, we are going to use Kosaraju's Algorithm.

Before understanding the algorithm, we are going to discuss the thought process behind it. If we start DFS from node 0 for the following graph, we will end up visiting all the nodes. So, it is impossible to differentiate between different SCCs.



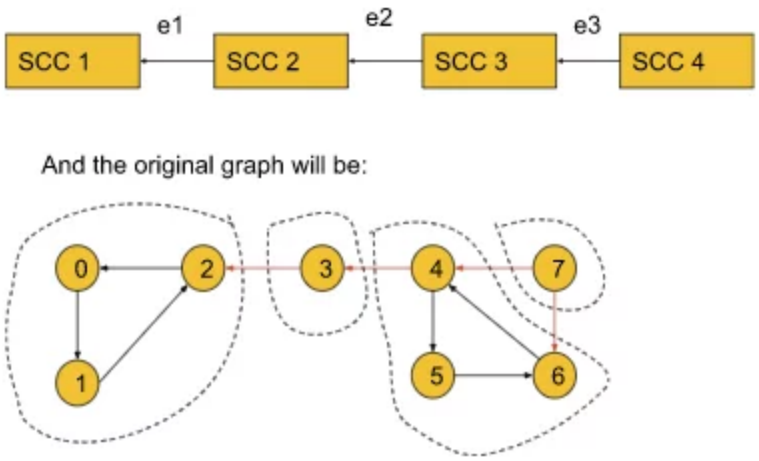
Now, we need to think in a different way. We can convert the above graph into the following illustration:



SCC = Strongly Connected Component

By definition, within each SCC, every node is reachable. So, if we start DFS from a node of SCC1 we can visit all the nodes in SCC1 and via edge e1 we can reach SCC2. Similarly, we can travel from SCC2 to SCC3 via e2 and SCC3 to SCC4 via e3. Thus all the nodes of the graph become reachable.

But if we reverse the edges e1, e2, and e3, the graph will look like the following:

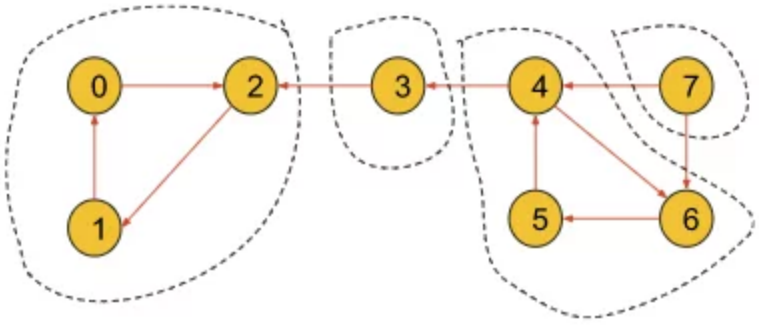


Now in this graph, if we start DFS from node 0 it will visit only the nodes of SCC1. Similarly, if we start from node 3 it will visit only the nodes of SCC2. Thus, by reversing the SCC-connecting edges, the adjacent SCCs become unreachable. Now, the DFS will work in such a way, that in one DFS call we can only visit the nodes of a particular SCC. So, ***the number of DFS calls will represent the number of SCCs***.

Until now, we have successfully found out the process of getting the number of SCCs.

But here, comes a new problem i.e. if we do not know the SCCs, how the edges will be reversed?

To solve this problem, we will simply try to reverse all the edges of the graph like the following:



If we carefully observe, the nodes within an SCC are reachable from each one to everyone even if we reverse the edges of the SCC. So, the SCCs will have no effect on reversing the edges. Thus we can fulfill our intention of reversing the SCC-connecting edge without affecting the SCCs.

Now, the question might be like, if node 0 is located in SCC4 and we start DFS from node 0, again we will visit all the SCCs at once even after reversing the edges. This is where ***the starting time and the finishing time*** concept will come in.

Now, we have a clear intuition about reversing edges before we move on to the starting and the finishing time concept in the algorithm part.

Algorithm:

The algorithm steps are as follows:

1. ***Sort all the nodes according to their finishing time:***
    a. To sort all the nodes according to their finishing time, we will start DFS from node 0 and while backtracking in the DFS call we will store the nodes in a stack data structure. The nodes in the last SCC will finish first and will be stored in the last of the stack. After the DFS gets completed for all the nodes, the stack will be storing all the nodes in the sorted order of their finishing time.
2. ***Reverse all the edges of the entire graph.***
3. ***Perform the DFS and count the no. of different DFS calls to get the no. of SCC.***