Kosaraju’s algorithm

* A linear time algorithm to find the strongly connected components of a directed graph
* Complexity O(V+E) as DFS takes O(V+E) and reversing a graph takes O(V+E)

Strongly connected

* A directed graph is strongly connected if every vertex is reachable from every other vertex
* **Strongly connected components** form a partition of an arbitrary graph into subgraphs that are themselves strongly connected
  + Must be maximal with this property, meaning no additional edges or vertices from the graph can be included in the subgraph without breaking its property of being strongly connected

Pseudocode

* Create an empty stack ‘S’ and do DFS on the graph, pushing the traversed vertices onto the stack ‘S’
* Reverse direction of edges to get a transposed graph ‘GT’
* Pop vertices from ‘S’ until it is empty. Let the popped vertices be ‘v’, take ‘v’ as the root and do DFS on ‘GT’. The traversed vertices are in the strongly connected component of ‘v’

Explanation[[1]](#footnote-1)

* Property: strongly connected components of G is same as strongly connected components of GT
* DFS of a graph produces a single tree if all vertices are reachable from the DFS starting point
  + Otherwise DFS produces a forest (Multiple trees)
  + So DFS of a graph with only one SCC always produces a tree
* DFS may produce a tree or a forest when there are more than one SCCs depending upon the chosen starting point
* To find and print all SCCs we want to start DFS from a sink vertex to a sink vertex …
  + Sink vertex just means a vertex with no outgoing edges
  + In this scenario we want to start DFS with 4, then 3 then any of the remaining vertices
* But there is not direct way of getting this desired sequence
* However, if we do a DFS of graph and store vertices according to their finish times, we make sure that the finish time of a vertex that connects to other SCCs (other that its own SCC), will always be greater than finish time of vertices in the other SCC (See [this](http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/strongComponent.htm) for proof). For example, in DFS of above example graph, finish time of 0 is always greater than 3 and 4 (irrespective of the sequence of vertices considered for DFS). And finish time of 3 is always greater than 4. DFS does not guarantee about other vertices, for example finish times of 1 and 2 may be smaller or greater than 3 and 4 depending upon the sequence of vertices considered for DFS. So to use this property, we do DFS traversal of complete graph and push every finished vertex to a stack. In stack, 3 always appears after 4, and 0 appear after both 3 and 4.
* In the next step, we reverse the graph. Consider the graph of SCCs. In the reversed graph, the edges that connect two components are reversed. So the SCC {0, 1, 2} becomes sink and the SCC {4} becomes source. As discussed above, in stack, we always have 0 before 3 and 4. So if we do a DFS of the reversed graph using sequence of vertices in stack, we process vertices from sink to source (in reversed graph). That is what we wanted to achieve and that is all needed to print SCCs one by one.

1. https://www.geeksforgeeks.org/strongly-connected-components/ [↑](#footnote-ref-1)