DAA Project Blog
Kosaraju's algorithm

Jigyasa Bajpai N007
Ritika Chand N009
Divya Chhoriya N010

MBA Tech CS
3rd year 5th semester

Blog Link:
https://algorithms171372765.wordpress.com/2021/10/22/kosarajus-algorithm/

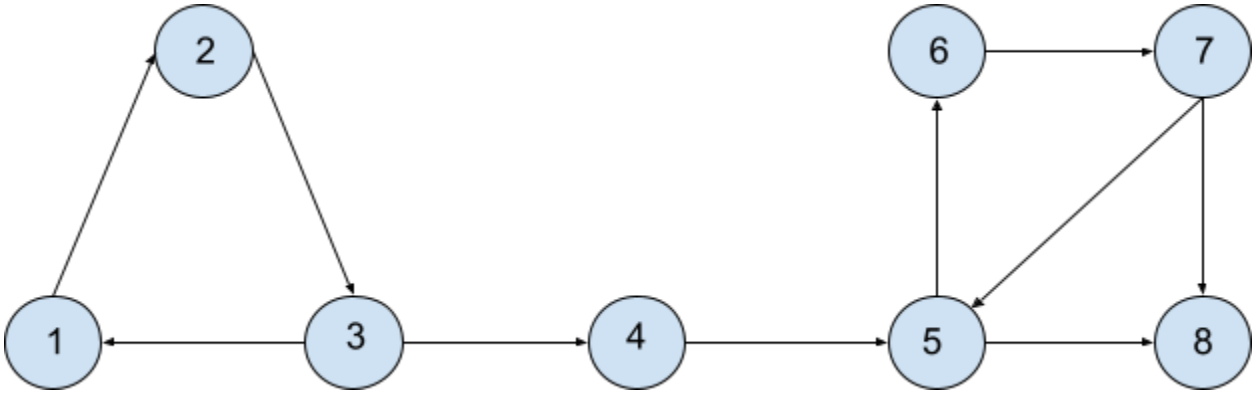
Kosaraju's algorithm

Kosaraju-Sharir's algorithm (also known as Kosaraju's algorithm) is a linear time algorithm used to find the strongly connected components of a directed graph. It makes use of the fact that the transpose graph (the same graph with the direction of every edge reversed) has exactly the same strongly connected components as the original graph.

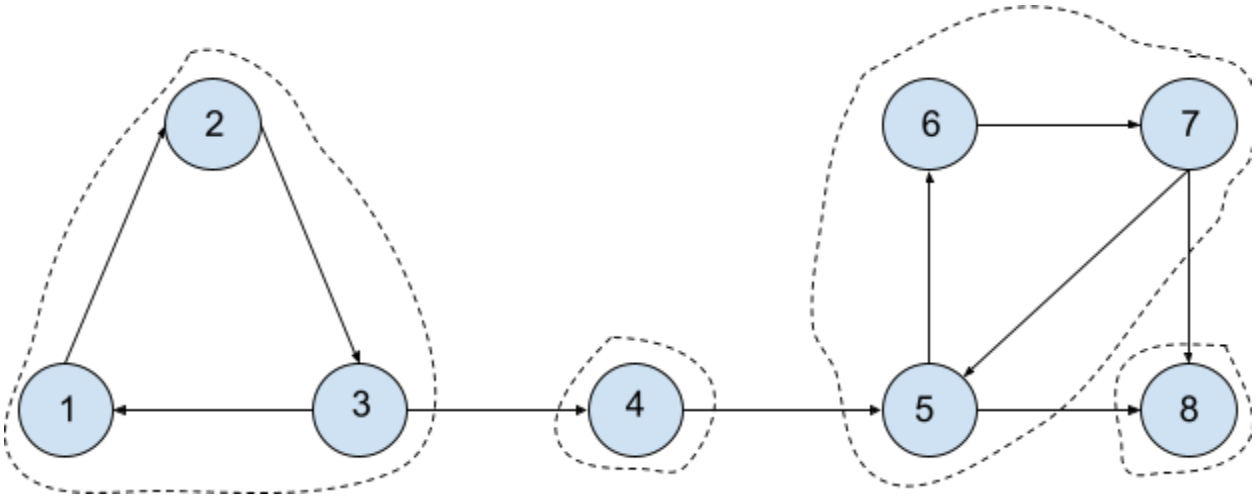
Strongly connected: The graph is strongly connected if you can get from any one vertex to any other vertex and vice-versa.

Strongly connected components (SCC): It is a sub-graph or a portion of graph in which there is a path from each vertex to another vertex.

Consider a directed graph G with 8 vertices.



The strongly connected components of the above graph are:
1st component: 1, 2, 3
2nd component: 4
3rd component: 5, 6, 7
4th component: 8



Kosaraju's algorithm does two DFS (Depth First Search) traversals on a graph.

DFS is an algorithm used for traversing or searching graph data structure. It starts from the root vertex and explores all the possible vertices before backtracking.

Algorithm

1. Perform DFS on graph G
2. Reverse the graph G

3. Perform DFS on reversed graph Grev

Step 1: Perform DFS on graph G.

Consider the above graph G.
Start DFS from vertex 1. When we return from a vertex then we push it into the stack.
Mark the vertex 1 as explored/traversed.

Traversed vertex:

1							

Since from vertex 1, we have only one option to go i.e. vertex 2. So we will go to vertex 2 and mark it as traversed.

Traversed vertex:

1	2						

Then go to vertex 3 and mark it as traversed. Now from vertex 3, we have two options: either we can go to vertex 1 or vertex 4. Since vertex 1 is already marked as traversed we will go to vertex 4 and mark it as traversed. Similarly, go to vertex 5, 6, 7, and 8 and mark them as explored.

Traversed vertex:

1	2	3	4	5	6	7	8

Now from vertex 8, there is no outgoing edge. So we will backtrack to vertex 7. While returning we will push the vertex into the stack.

Stack:

8							

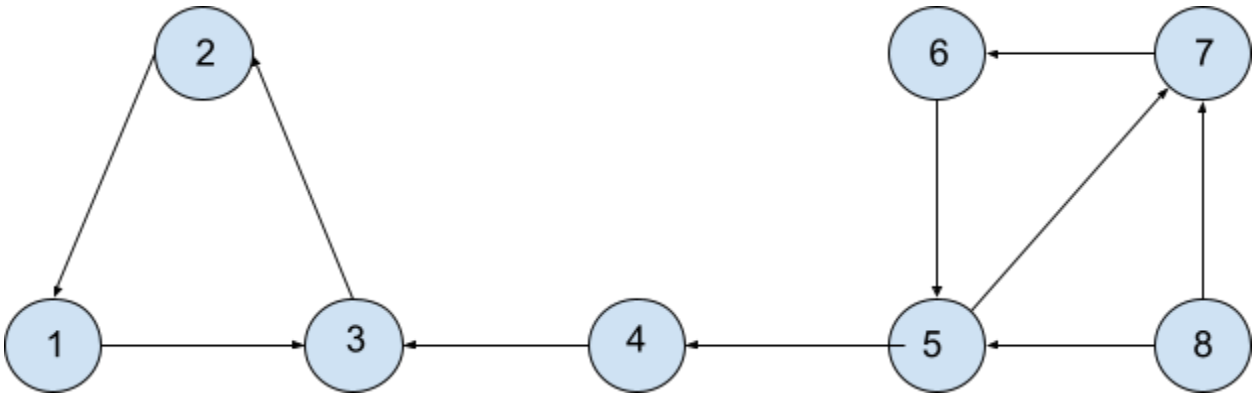
Now from vertex 7, all the vertices of outgoing edges are explored. So again backtrack to vertex 6 and push vertex 7. Similarly, we will backtrack to 6, 5, 4, 3, 2, and 1 and simultaneously push them into the stack.

Stack:

8	7	6	5	4	3	2	1

Step 2: Reverse the graph G

Here we reverse all the directions of the edges. For example, in the original graph we traverse from vertex 1 to 2. In the reverse graph, we traverse from 2 to 1. Similarly, we reverse the whole graph.



Step 3: Perform DFS on reversed graph

Mark all the vertices as untraversed. Now pop the vertex one by one and perform DFS. Each DFS call will give one strongly connected component.

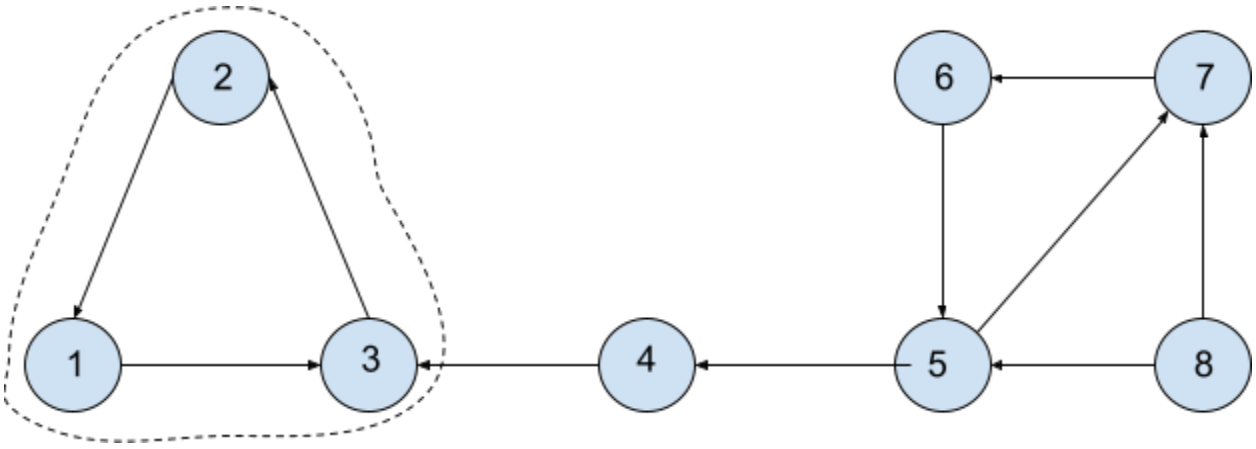
Pop vertex 1 from the stack. Start DFS from vertex 1 and mark it as traversed. From vertex 1 go to vertex 3 and mark it as traversed. Then go to vertex 2 and mark it as traversed. Now from vertex 2, we can't go to vertex 1 because it is already traversed. So backtrack to vertex 3. Again there are no outgoing edges from vertex 3 so backtrack to vertex 1. Now all the outgoing edges from vertex 1 are traversed. So after this DFS loop, all the vertices which were traversed will form one strongly connected component with vertex 1, 2, and 3.

Stack:

8	7	6	5	4	3	2	

Traversed vertex:

1	3	2					



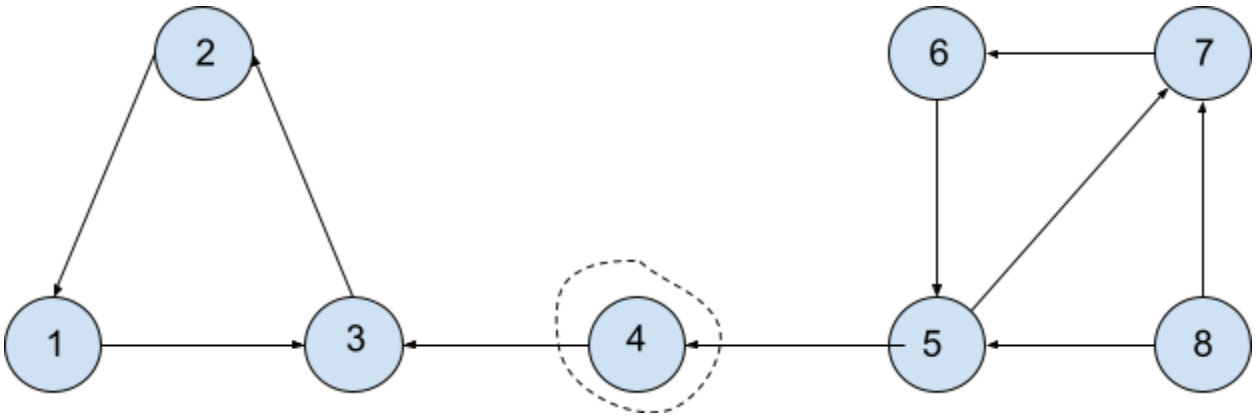
Now pop vertex 2 but it is already traversed so we will not make a DFS call. Then pop vertex 3 but it is also traversed. Then pop vertex 4. Since it is not traversed we can make a DFS call. Mark vertex 4 as traversed. Now from vertex 4, there is only one outgoing edge i.e. to vertex 3 but it is already traversed. And there are no more outgoing edges from vertex 4 so this DFS loop is completed and hence we got another SCC with vertex 4.

Stack:

8	7	6	5				

Traversed vertex:

1	3	2	4				



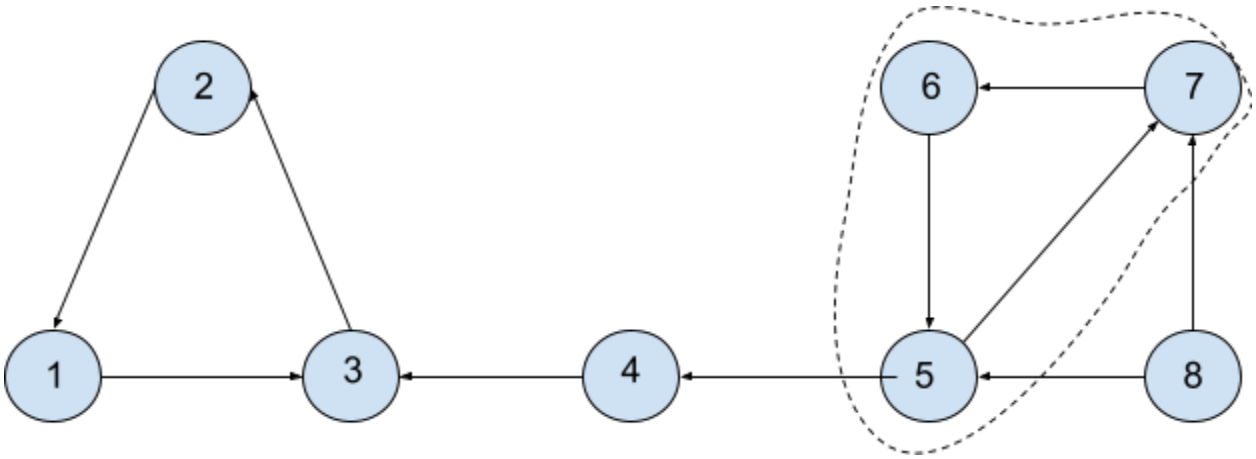
Similarly, now we will pop vertex 5 and make a DFS call. Mark it as traversed. Go to vertex 7 mark it as traversed. Go to vertex 6 mark it as traversed. Now all the outgoing edges are traversed so backtrack to vertex 7 again backtrack to vertex 5. From this vertex, there are no more outgoing edges. So after this loop, we found our 3rd SCC with vertex 5, 6, and 7.

Stack:

8	7	6					

Traversed vertex:

1	3	2	4	5	7	6	

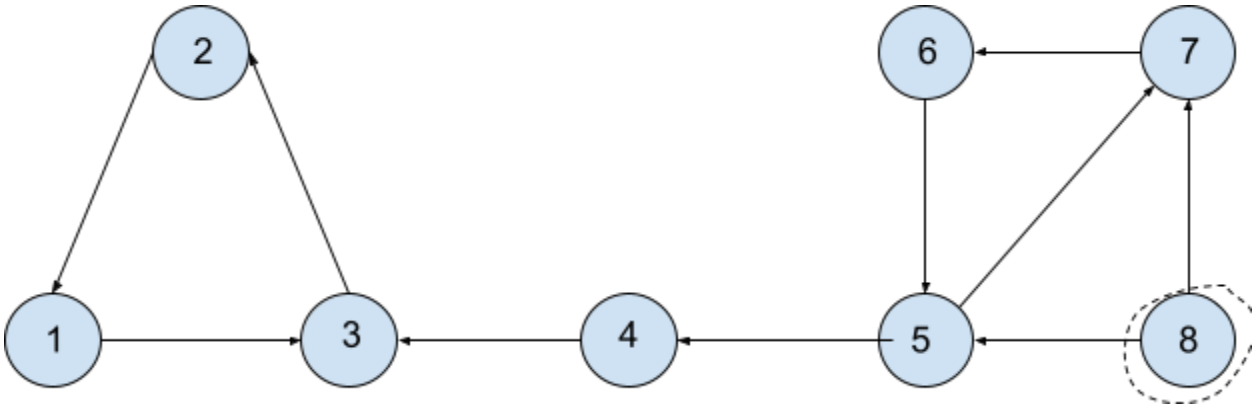


Now pop vertex 6 but it is already traversed. Vertex 7 is also traversed. So now pop vertex 8 and make another DFS call and mark it as traversed. From 8 there is only one outgoing edge that is already traversed. So from this loop, we got our 4th SCC with vertex 8.

Stack:

Traversed vertex:

1	3	2	4	5	7	6	8



Depth First Search (DFS) of a Graph

Depth First Search (DFS) algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

Algorithm

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until STACK is empty

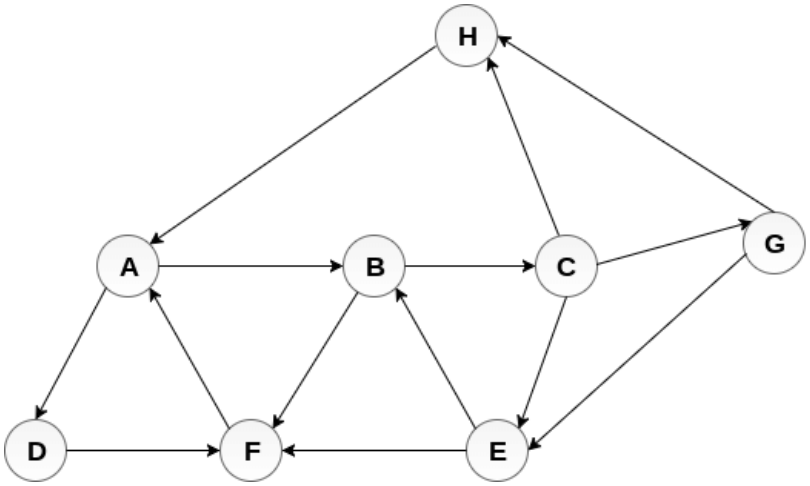
Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

Step 5: Push on the stack all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state) [END OF LOOP]

Step 6: EXIT

Example:

Consider the graph G along with its adjacency list, given in the figure below. Calculate the order to print all the nodes of the graph starting from node H, by using depth first search (DFS) algorithm.



Adjacency Lists

A : B, D
B : C, F
C : E, G, H
G : E, H
E : B, F
F : A
D : F
H : A

Solution:

Push H onto the stack
STACK: H

POP the top element of the stack i.e. H, print it and push all the neighbours of H onto the stack that are in ready state.
Print H
STACK: A

Pop the top element of the stack i.e. A, print it and push all the neighbours of A onto the stack that are in ready state.
Print A
Stack: B, D

Pop the top element of the stack i.e. D, print it and push all the neighbours of D onto the stack that are in ready state.
Print D
Stack: B, F

Pop the top element of the stack i.e. F, print it and push all the neighbours of F onto the stack that are in ready state.
Print F
Stack: B

Pop the top of the stack i.e. B and push all the neighbours
Print B
Stack: C

Pop the top of the stack i.e. C and push all the neighbours.
Print C
Stack: E, G

Pop the top of the stack i.e. G and push all its neighbours.
Print G
Stack: E
Pop the top of the stack i.e. E and push all its neighbours.
Print E
Stack:

Hence, the stack now becomes empty and all the nodes of the graph have been traversed.
The printing sequence of the graph will be :
H → A → D → F → B → C → G → E

Complexity of Kosajaru's Algorithm

The time complexity of this algorithm is **O(V+E)**, where V is the number of vertices and E is the number of edges.

Applications of Kosajaru's Algorithm

- Kosaraju's algorithm is used to find the Strongly Connected Components in a graph in linear time.
- SCC algorithms can be used as a first step in many graph algorithms that work only on strongly connected graphs. In social networks, a group of people are generally strongly connected (For example, students of a class or

any other common place). Many people in these groups generally play common games. The SCC algorithms can be used to find such groups and suggest the commonly liked games to the people in the group who have not yet liked a commonly played game.

- Vehicle routing applications
- Maps

References

- https://www.javatpoint.com/depth-first-search-algorithm
- https://www.youtube.com/watch?v=5wFyZJ8yH9Q
- https://www.geeksforgeeks.org/strongly-connected-components/
- https://en.wikipedia.org/wiki/Kosaraju%27s_algorithm

Conclusion

Hence Kosajaru's Algorithm is used to find the strongly connected components of a directed graph. It makes use of 2 concepts - transpose of a graph, and depth first search of a graph. This algorithm also has several real life applications.