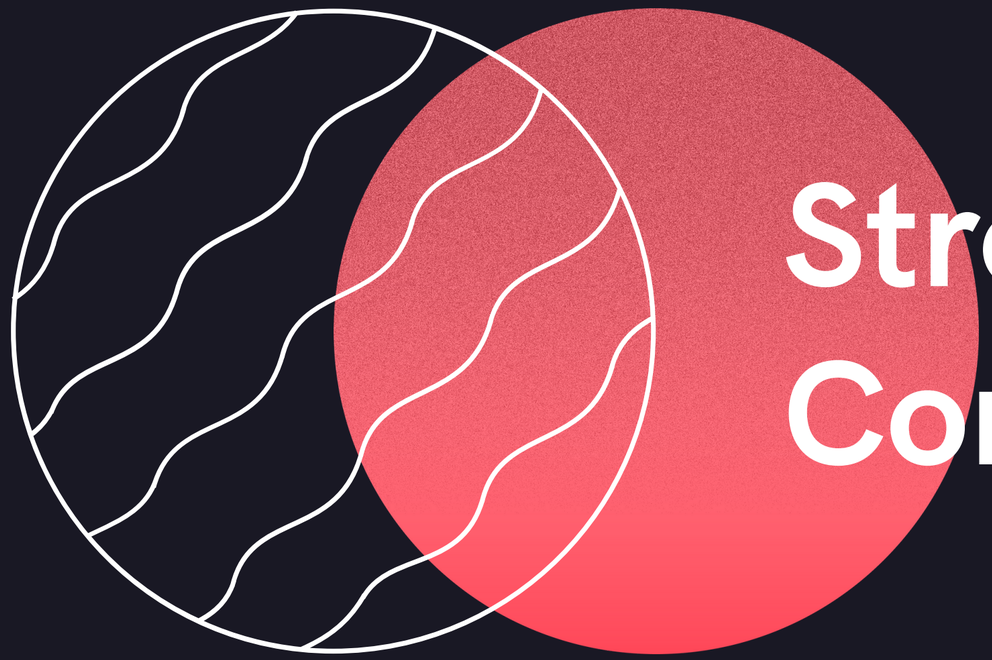
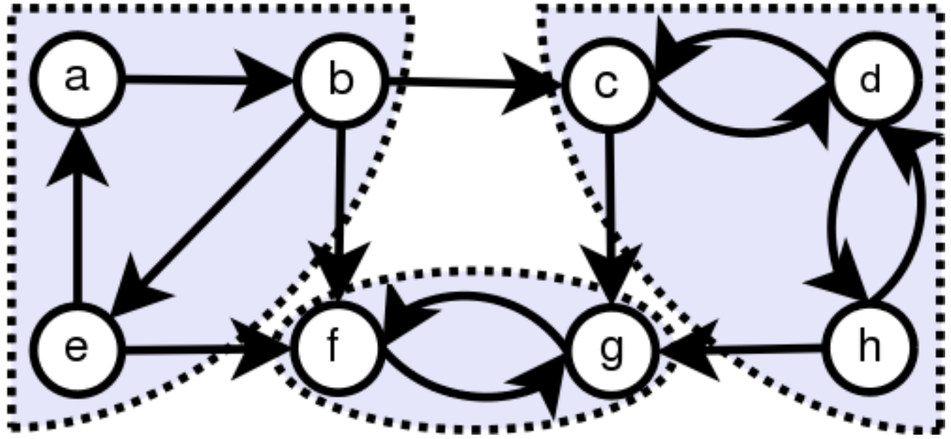


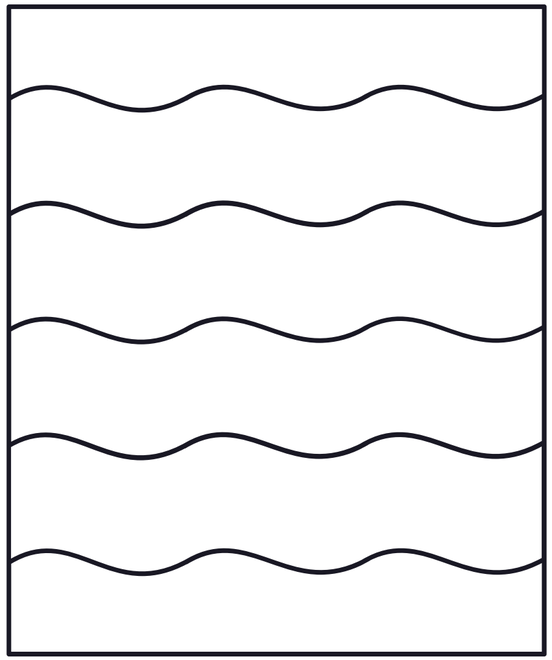
Strongly Connected Components (SCC)

Project work by AlgorithmTeam:
Nurdaulet Daudov
Medet Kurganbayev
Ali Tulebergenov
Adlet Askarov
Zhomart Abdrakhman

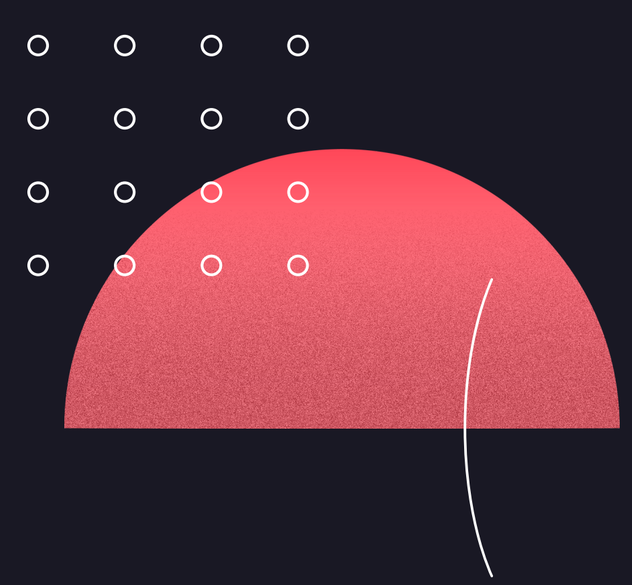
Definition

In the mathematical theory of directed graphs, a graph is said to be strongly connected if every vertex is reachable from every other vertex. The strongly connected components of an arbitrary directed graph form a partition into subgraphs that are themselves strongly connected. It is possible to test the strong connectivity of a graph, or to find its strongly connected components, in linear time (that is, Θ(V + E)).





Algorithms



Several algorithms based on depth first search compute strongly connected components in linear time.

Kosaraju's algorithm

Tarjan's algorithm

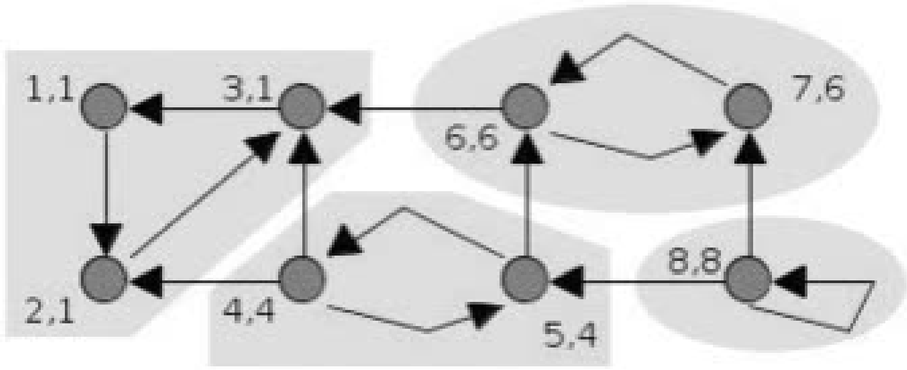
The path-based strong component algorithm

KOSARAJU'S ALGORITHM

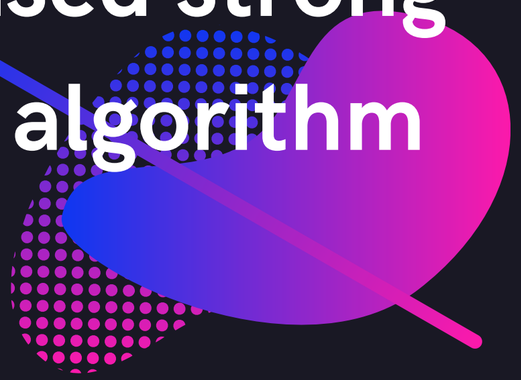
Kosaraju’s is based on DFS, it does DFS two times. The basic concept of this algorithm is that if we are able to arrive at vertex v initially starting from vertex u, then we should be able to arrive at vertex u starting from vertex v, and if this is the situation, we can say and conclude that vertices u and v are strongly connected, and they are in the strongly connected sub-graph.

TARJAN'S ALGORITHM

The algorithm takes a directed graph as input, and produces a partition of the graph's vertices into the graph's strongly connected components. Each vertex of the graph appears in exactly one of the strongly connected components. Any vertex that is not on a directed cycle forms a strongly connected component all by itself: for example, a vertex whose in-degree or out-degree is 0, or any vertex of an acyclic graph.
The basic idea of the algorithm is this: a depth-first search (DFS) begins from an arbitrary start node (and subsequent depth-first searches are conducted on any nodes that have not yet been found). As usual with depth-first search, the search visits every node of the graph exactly once, declining to revisit any node that has already been visited. Thus, the collection of search trees is a spanning forest of the graph. The strongly connected components will be recovered as certain subtrees of this forest. The roots of these subtrees are called the "roots" of the strongly connected components. Any node of a strongly connected component might serve as a root, if it happens to be the first node of a component that is discovered by search.



The path-based strong component algorithm



The path-based strong component algorithm uses a depth first search, like Tarjan's algorithm, but with two stacks. One of the stacks is used to keep track of the vertices not yet assigned to components, while the other keeps track of the current path in the depth first search tree.

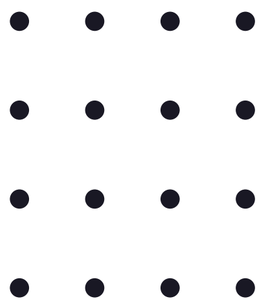
Although Kosaraju's algorithm is conceptually simple, Tarjan's and the path-based algorithm require only one depth-first search rather than two.

TIME COMPLEXITY

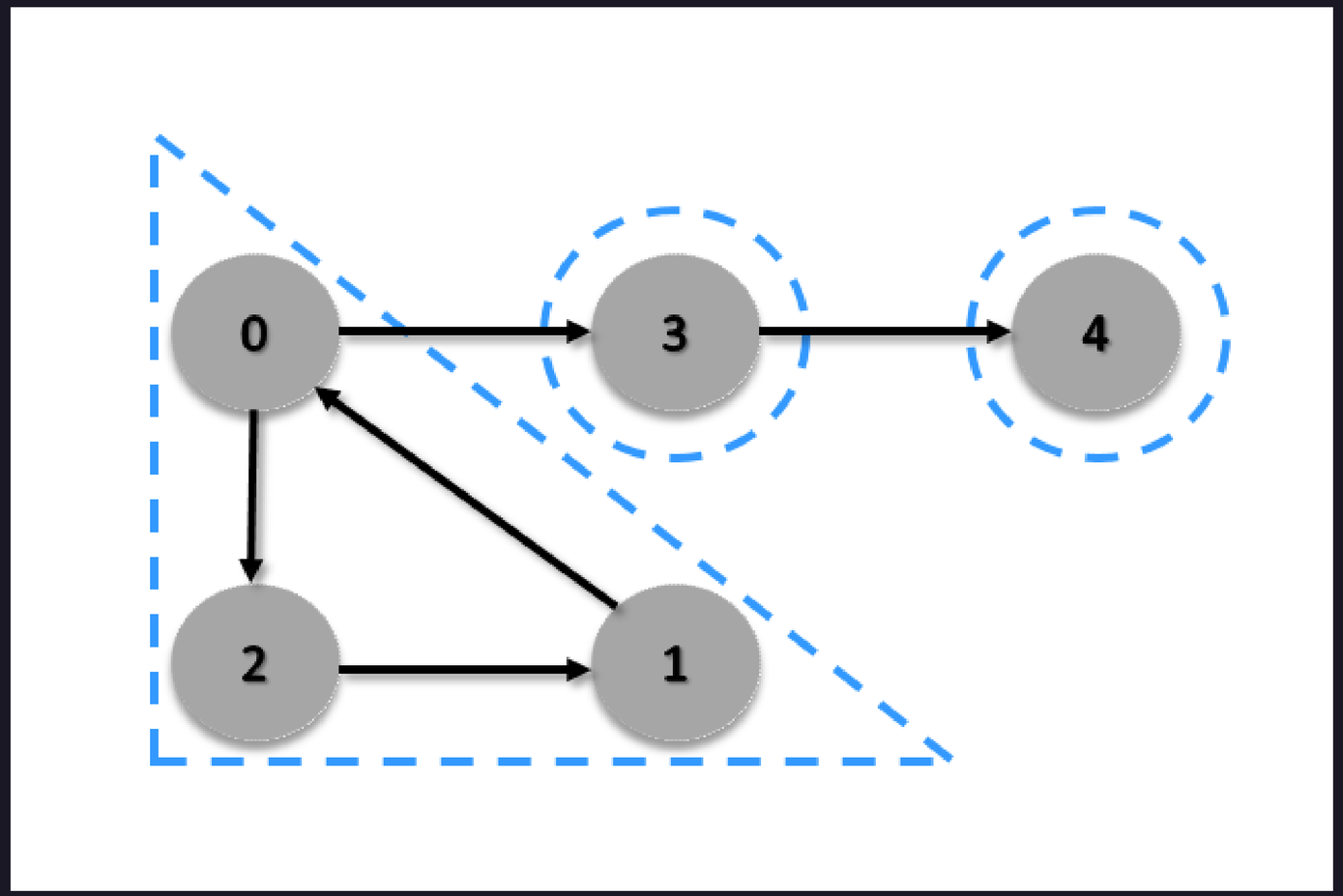
The time complexity of Tarjan’s Algorithm and Kosaraju’s Algorithm will be O(V + E), where V represents the set of vertices and E represents the set of edges of the graph. Tarjan’s algorithm has much lower constant factors w.r.t Kosaraju’s algorithm. In Kosaraju’s algorithm, the traversal of the graph is done at least 2 times, so the constant factor can be of double time. We can print the SCC in progress with Kosaraju’s algorithm as we perform the second DFS. While performing Tarjan’s Algorithm, it requires extra time to print the SCC after finding the head of the SCCs sub-tree.

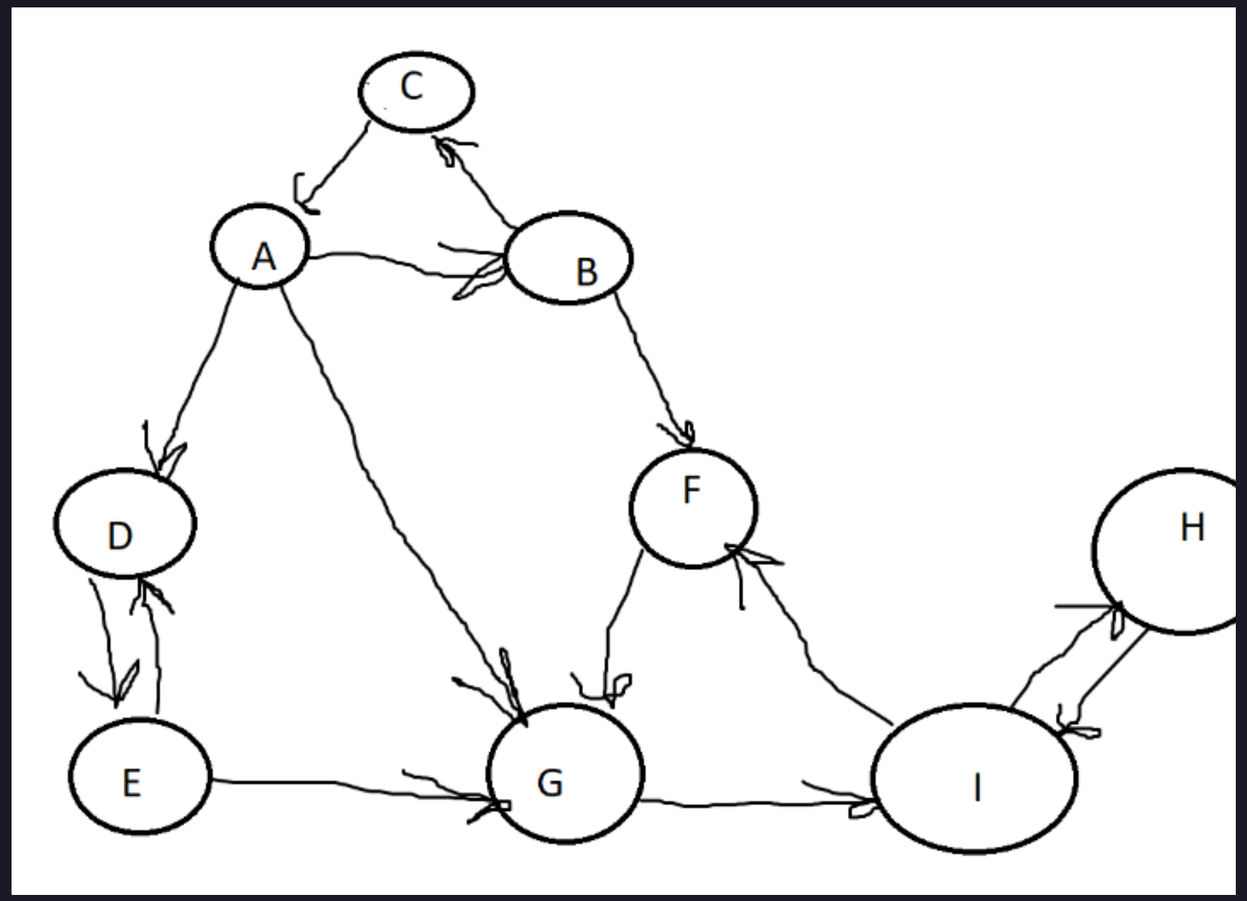
SUMMARY

Both the methods have the same linear time complexity, but the techniques or the procedure for the SCC computations are fairly different. Tarjan’s method solely depends on the record of nodes in a DFS to partition the graph whereas Kosaraju’s method performs the two DFS (or 3 DFS if we want to leave the original graph unchanged) on the graph and is quite similar to the method for finding the topological sorting of a graph.

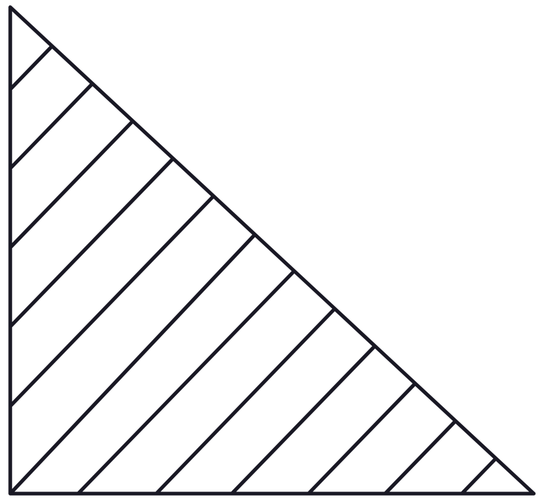


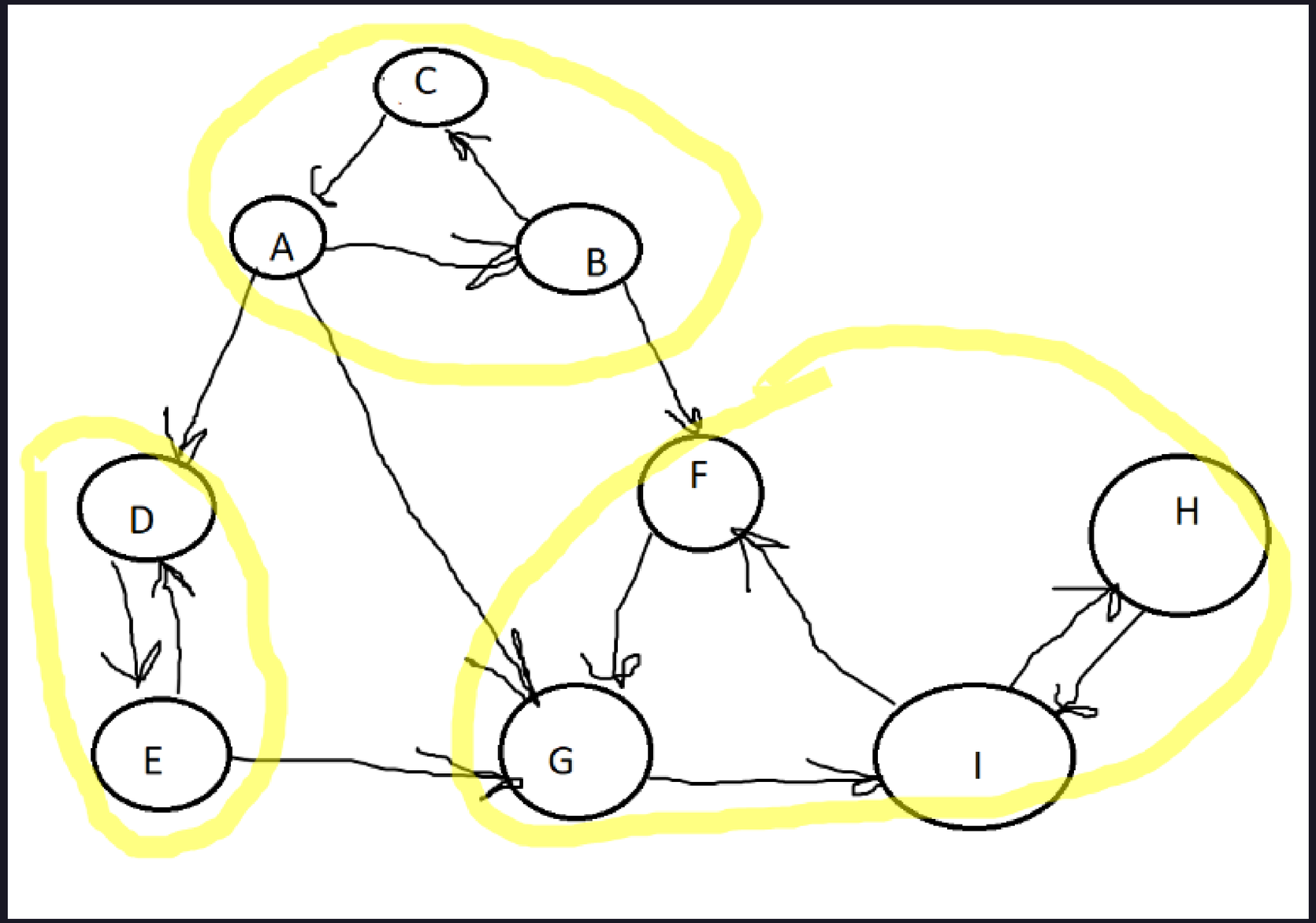
GRAPH EXAMPLE





LETS FIND STRONGLY CONNECTED COMPONENTS TOGETHER

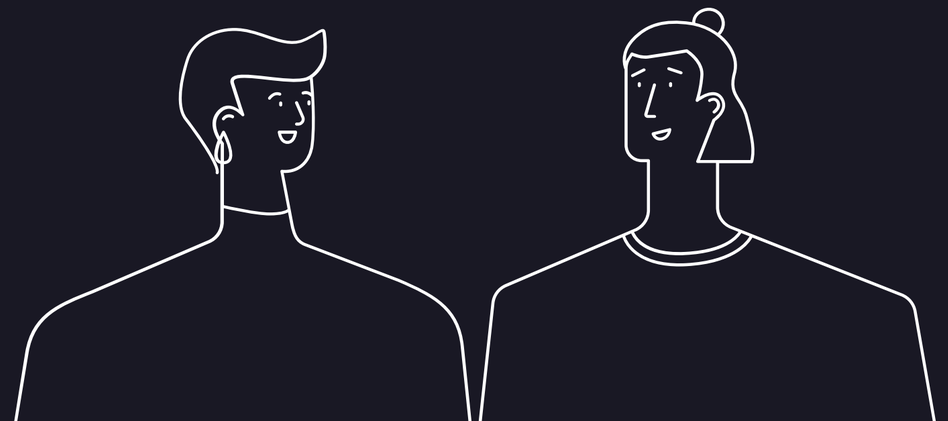


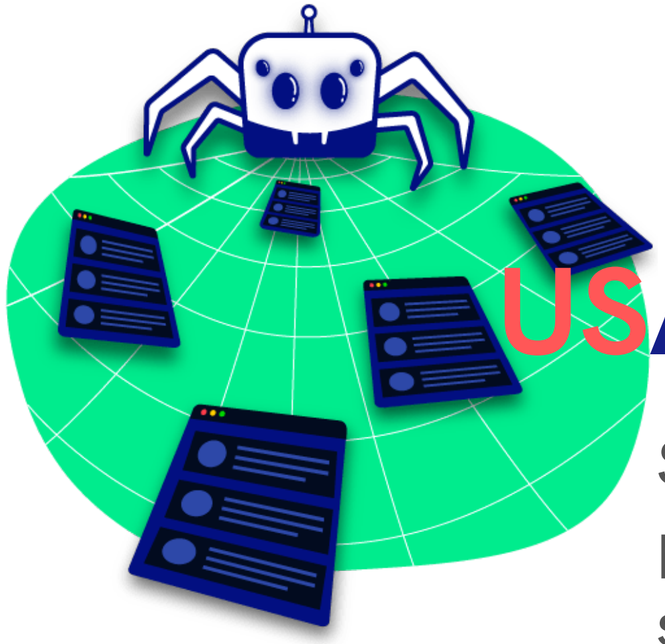

C
A
B
D
F
H
E
G
I

USAGE OF SCC

SCC algorithms can be used as a first step in many graph algorithms that work only on strongly connected graph.

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 like some common pages or play common games. The SCC algorithms can be used to find such groups and suggest the commonly liked pages or games to the people in the group who have not yet liked commonly liked a page or played a game.

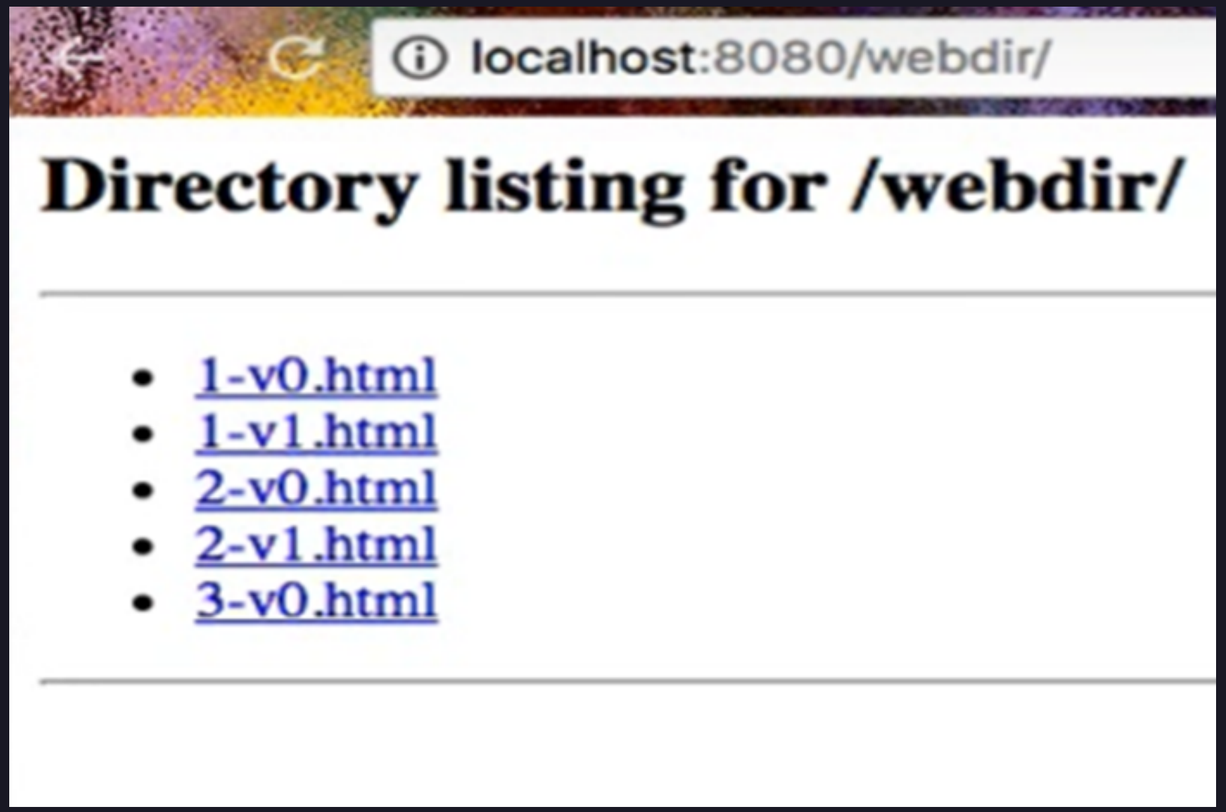




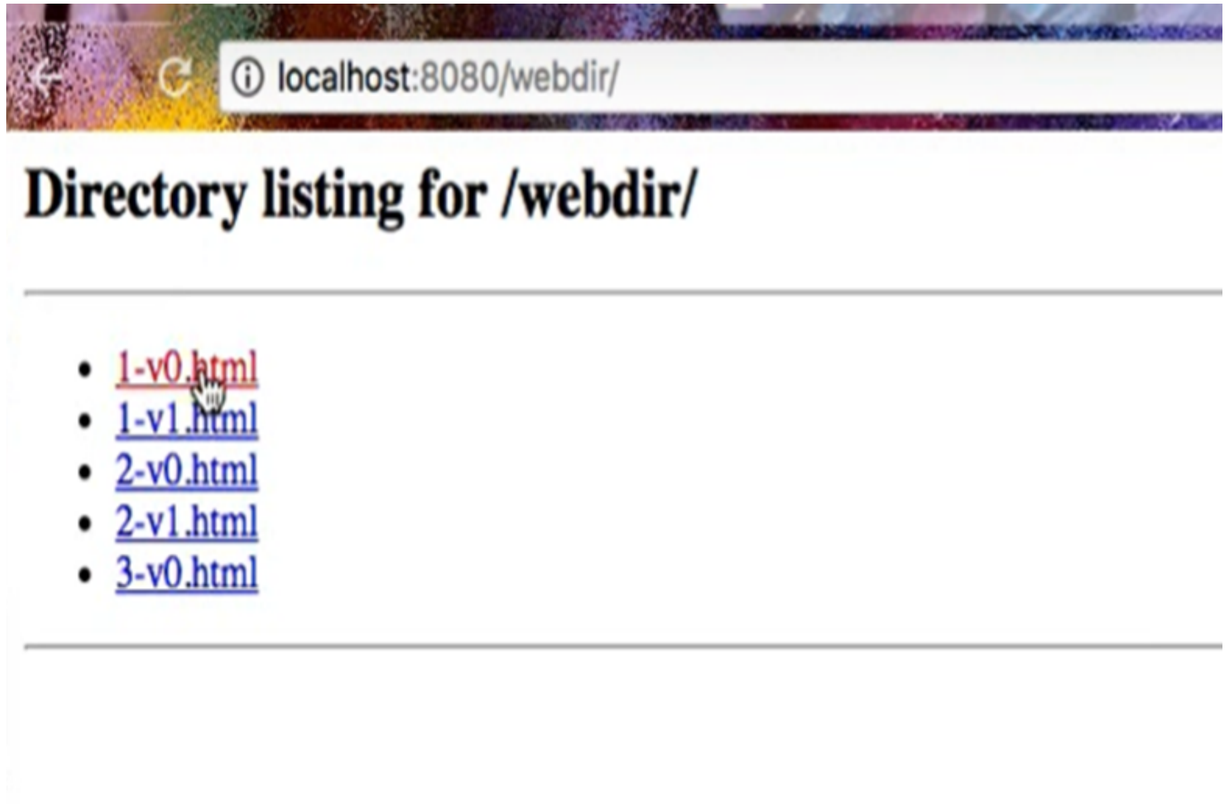
USAGE OF SCC

Strongly connected components in a directed graph is particularly interesting. If your site have links to site B and site B gives back link, search engine sometimes give the sites better rating. We can model the internet as a directed graph, each site is a vertex and outbound links to other sites are directed graph edges, then we can use kosaraju's algorithms to find the strongly connected components. This program instead of crawling the internet, created a limited intranet to test the kosaraju's algorithms. In order to go internet, the crawler need to be updated for infinite number of sites and know when to stop

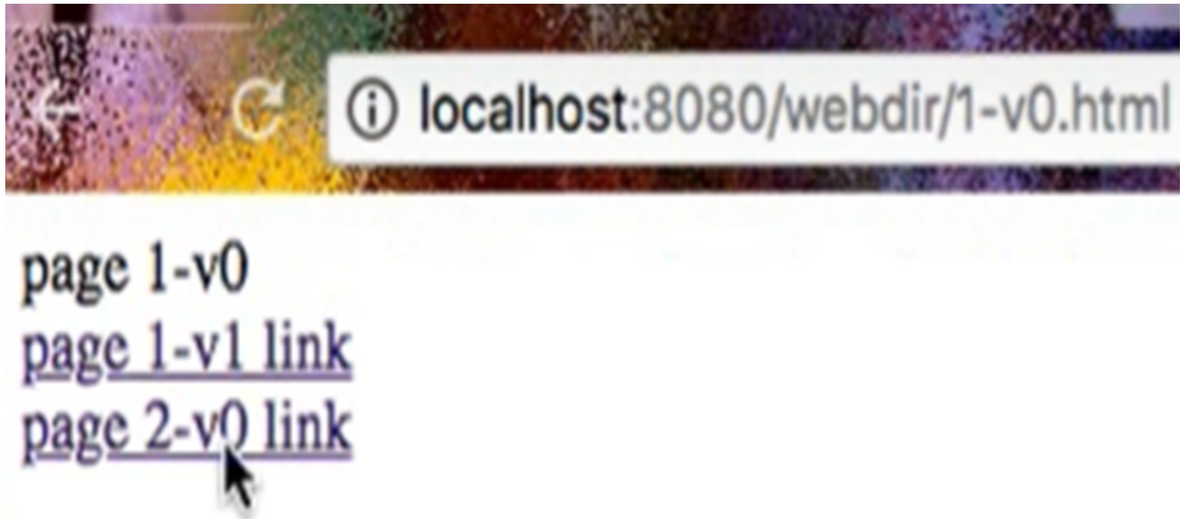
SCC EXAMPLE



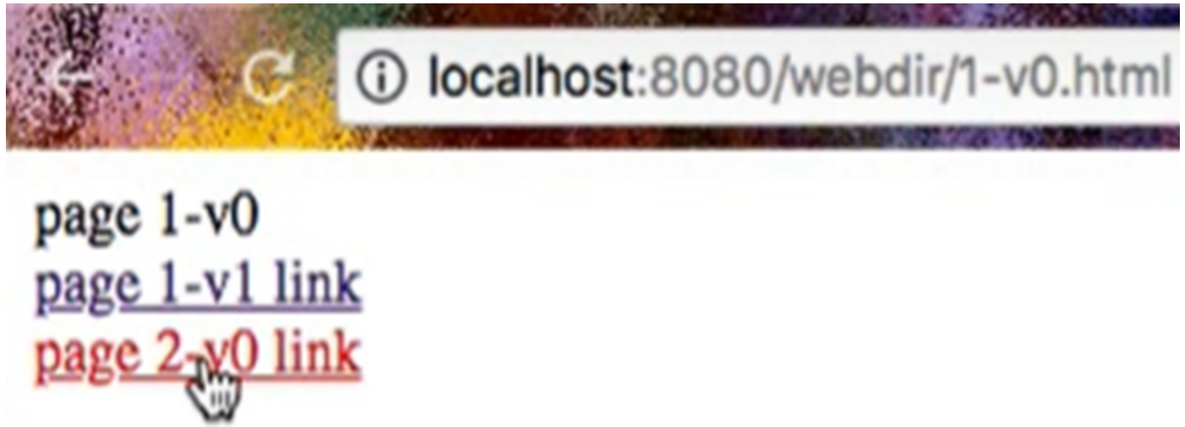
1



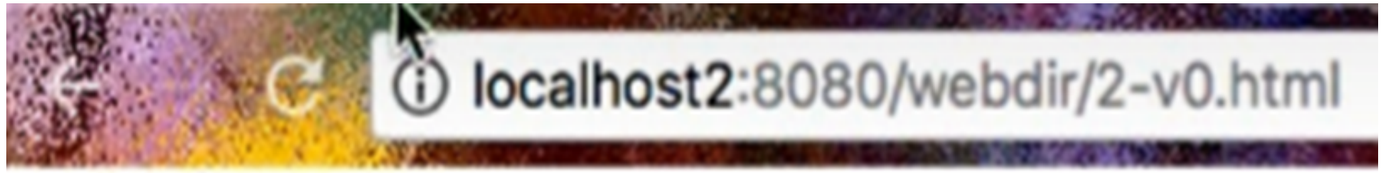
2


localhost:8080/webdir/1-v0.html
page 1-v0
page 1-v1 link
page 2-v0 link

3

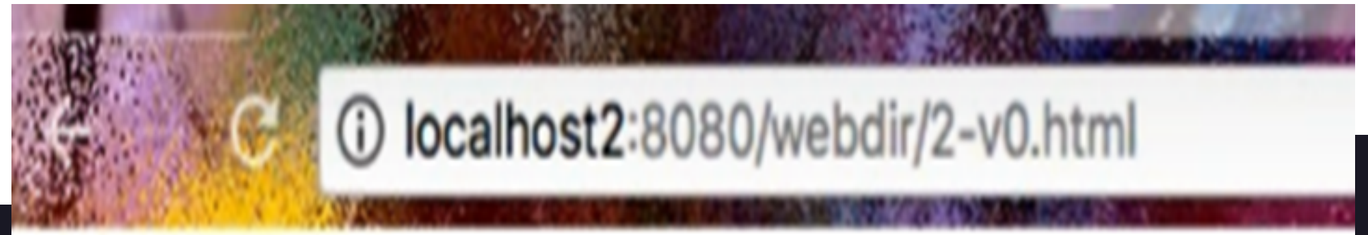

localhost:8080/webdir/1-v0.html
page 1-v0
page 1-v1 link
page 2-v0 link

4



page 2-v0
page 1-v1 link
page 2-v1 link
page 2-v3 link

5



page 2-v0
page 1-v1 link
page 2-v1 link
page 2-v3 link

6

page 1-v1

page 1-v0 link

page 1-v4 link

page 1-v5 link

Problems Javadoc Declaration Search Console Call Hierarchy Debug

<terminated> ParallelCrawler (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_31.jdk/Contents/Home/bin/java (Mar 6, 2018, 5:52:1

```
http://localhost2:8080/webdir/2-v0.html
http://localhost1:8080/webdir/1-v1.html
http://localhost3:8080/webdir/3-v1.html
visited 9 nodes out of total 9
digraph: {localhost3=[], localhost2=[localhost3, localhost1], localhost1=[localhost2]}
reverse digraph: {localhost3=[localhost2], localhost2=[localhost1], localhost1=[localhost2]}

pre order
localhost3 -> localhost2 -> localhost1 ->
post order
localhost1 -> localhost2 -> localhost3 ->
reversePost order
localhost3 -> localhost2 -> localhost1 ->
strongly connected components: {localhost3=0, localhost2=1, localhost1=1}
```