Strongly Connected Components

This is another application of DFS. In a directed graph two vertices u and v are strongly connected if and only if there exists a path from u to v and there exists a path from v to u. The strong connectedness is an equivalence relation.

- a vertex is strongly connected with itself.
- if a vertex u is strongly connected to vertex v, then v is strongly connected to u.
- if a vertex u is strongly connected to a vertex v, and v is strongly connected to a vertex x, then u is strongly connected to x.

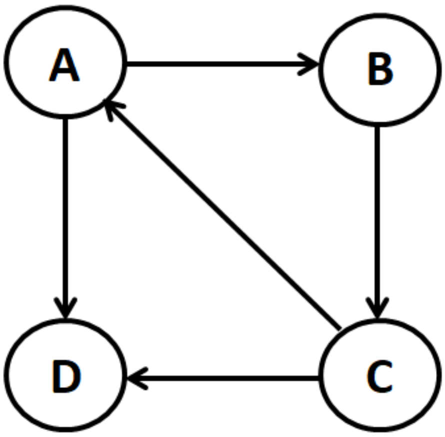
What this says is for a given directed graph we can divide it into strongly connected components. This problem can be solved by performing two depth-first searches. With two dfs searches, we can test whether a given directed graph is strongly connected or not. We can also produce the subset of the vertices that are strongly connected.

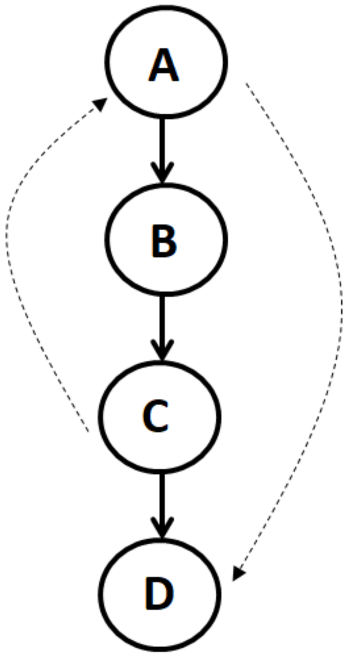
Algorithm

- Perform dfs on a given graph g
- Number the vertices of graph G according to a post-order traversal of depth-first spanning forest.
- Construct the G_r by reversing all edges in g.
- Performed on G_r: start a new DFS (initial call to visit) at the highest number vertex.
- Each tree in the resulting depth-first spanning forest corresponds to a strongly connected component.

Why does this algorithm work?

Let us consider two vertices,v and w. If they are in the same strongly connected component, then there are paths from v to w and from w to v in the original graph G, and hence also in the G_r , if two vertices v and w are not in the same depth-first spanning tree G_r , clearly they cannot be in the same strongly connected component. As an example consider the graph shown below on the left. Let us assume this graph is G.

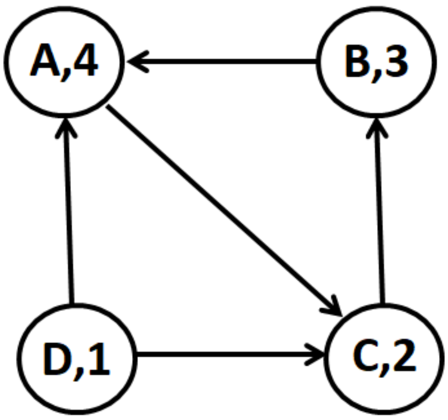




Now as per the algorithm performing, DFS on graph G gives the following diagram. The dotted line from C to A indicates a back-edge. Now, performing post-order traversal on the tree gives D, C, B and A.

Vertex	Post Order Number
A	4
B	3
C	2
D	1

Now reverse the given graph G and call it G_r and at the same time assign a postorder number to the vertices. The reversed G_r will look like this:



The last step is performing DFS on this graph G_r. While doing DFS. need to consider the vertex which has the largest DFS number. So, first, we start at A and with DFS we go to C and then B. At B we cannot move further. This says that {A, B, C} is strongly connected. Now, the only remaining element is D and we end our second DFS at D. So, the other component is {D}. Thus, the strongly connected companies are {A, B, C} and {D}.

```
list adj, adj_rev;                          // two lists for adjacency list, and reverse adj list
list used;                                  // list of boolean type to store visited vertices

//   order: to store post order traversal, component: to store components
list order, component;

function dfs1(int v) {
            used[v] = true;          // mark visited

            for child of vertex v in adj
                    if (used[child] is false)
                            dfs1(child);

            order.push(v);           // add vertex v to order vector
}

//   performing DFS on the reversed graph in the order of DFS1 stack(order)
function dfs2(int v) {
            used[v] = true;
            component.push(v);

            for child of vertex v in adj_rev
                    if (used[child] is false)
                            dfs2(child);
}

//   n : number of vertices
function connectedComponents(n) {

            // set the used list to false
            used.set(n, false);

            for all vertices u from 0 to n
                    if (used[i] is false)
                            dfs1(i);

            used.set(n, false);
            // now reverse the order list
            order.reverse();

            for all vertices v in the order
                    if (used[v] is false) {
```

```
                                        dfs2 (v);

                                        // ... processing next component ...
                                       component.clear();
    }
}
```

Time Complexity: The above algorithm calls DFS, finds reverse of the graph and again calls DFS. DFS takes O(V+E) for a graph represented using adjacency list. Reversing a graph also takes O(V+E) time. For reversing the graph, we simple traverse all adjacency lists.