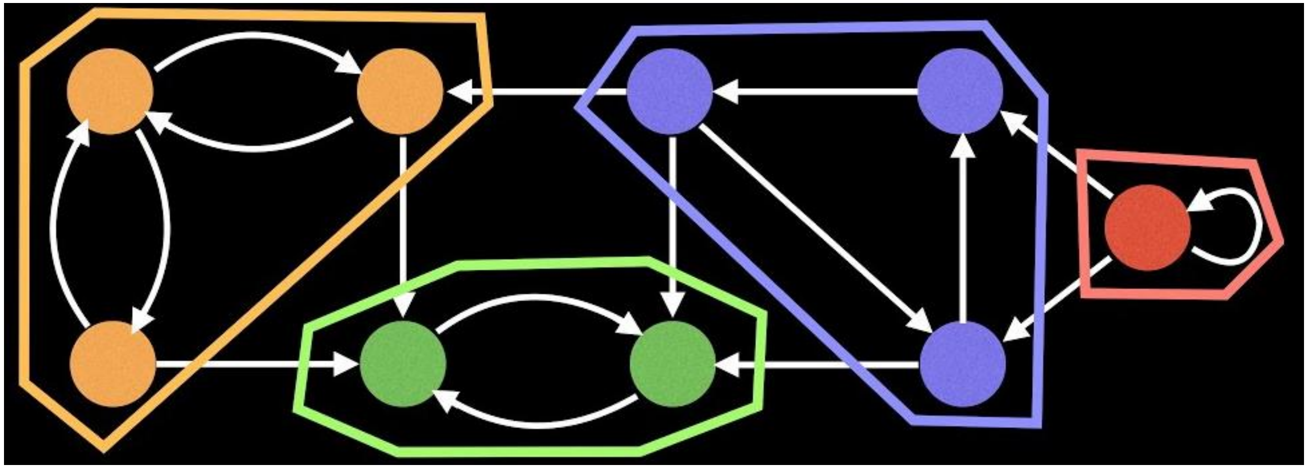
Strongly Connected Components of Digraphs

Textbook *Algorithms: Special Topics*

Chapter 3, Section 3.1, pp. 72-77



Strongly Connected

- Two vertices *u* and *v* in a digraph *D* are *strongly connected* if there is both a directed path from *u* to *v* and a directed path from *v* to *u*.
- *D* is strongly connected if every pair of vertices is strongly connected.

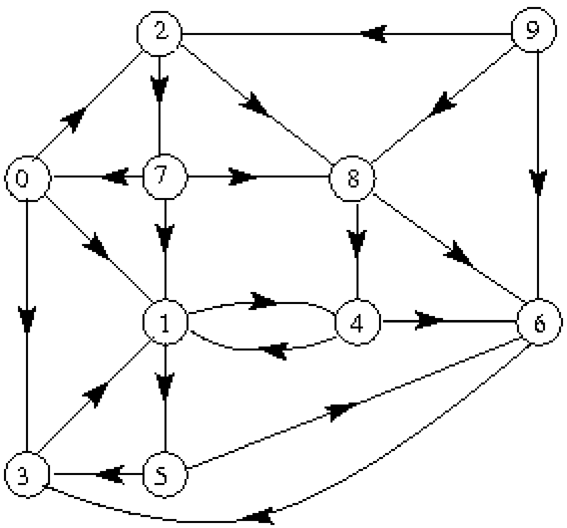
PSN. Design and analyze an algorithm for determining whether a digraph *D* is strongly connected.

Strongly Connected Components

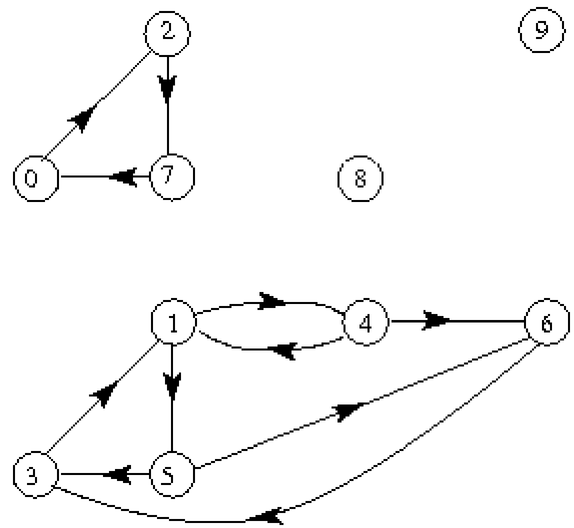
- A **strongly connected component** is a maximal subdigraph such that every pair of vertices in the subdigraph is strongly connected, where **maximal** means that adding any edge (and its incident vertices) will result in a subdigraph that is no longer strongly connected.

- The notion of a strongly connected component in a digraph $D = (V,E)$ is a generalization of the notion of a connected component in a graph.

Example of Strongly Connected Components



Digraph D



Strongly connected components

Equivalence relation

Another way to think of vertex sets for the strongly connected components is the equivalence classes for the relation S on the vertex set V, where two vertices u and v are related, i.e., uSv, iff they are strongly connected.

PSN. Show this relation S is an equivalence relation.

Naïve Algorithm for Strongly-Connected Components

For each vertex *v* that is unvisited, perform *BFSout* (*v*) and *BFSin*(*v*) (or *DFSout* (*v*) and *DFSin*(*v*)) .

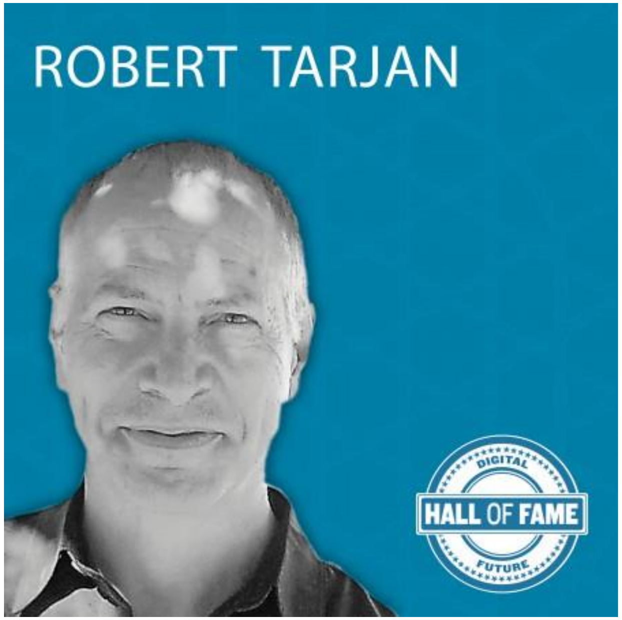
The set of vertices that are visited by both these searches determine the vertex set of the strongly connected component that contains vertex *v*.

Complexity Analysis. We perform an in-directed and an out-directed BFS searches. Since *BFSout* an *BFSin* have worst-case complexity $\Theta(m + n)$, where n is the number of vertices and m is the number of edges, our algorithm for computing the strongly connected components has worst-case complexity

$$W(n) \in \Theta(mn)$$

Faster Algorithm

- We now design an $O(m + n)$ algorithm *StrongComponents* for obtaining the set of strongly connected components based on the notion of the post numbering of the vertices of a digraph. This algorithm is due to the famous computer scientist and mathematician Robert Tarjan.



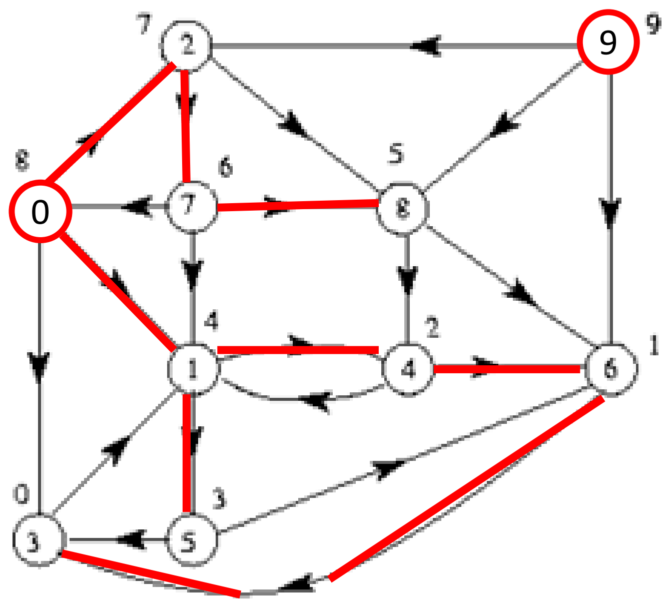
Post Numbering

Digraph $D = (V, E)$, where $V = \{0,1, \ldots, n-1\}$.

- A post numbering of a digraph *D* is determined by performing an out-directed depth-first traversal of *D*.
- A vertex *u* of *D* becomes *explored* when the traversal accesses *u* having visited all vertices in the out-neighborhood of *u*.
- The *post number* of *u*, denoted *PostNum*(*u*), is the integer i where *u* is the $(i+1)^{st}$ vertex to be explored in an out-directed depth-first traversal of *D*, *i* = 0,…,*n* – 1.

Post Numbering in Example Digraph

When perform the out-directed depth-first traversal when there is a choice the smallest label node is chosen. Post number is shown outside each node



i	0	1	2	3	4	5	6	7	8	9
$PostNum[i]$	8	4	7	0	2	3	1	6	5	9

Observation

Vertex u has post number i if u is the $(i+1)^{st}$ vertex to be visited in a postorder traversal of the DFT out-forest, $i = 0,\dots,n-1$.

Post Number Inverse

The array $PostNumberInv$ is defined by

$$PostNumberInv[i] = j \text{ iff } PostNumber[j] = i$$

i	0	1	2	3	4	5	6	7	8	9
$PostNum[i]$	8	4	7	0	2	3	1	6	5	9
$PostNumInv[i]$	3	6	4	5	1	8	7	2	0	9

Observation. PostNuminv lists the vertices in the order they were explored by the out-directed depth-first traversal

Algorithm *StrongComponents*

- Algorithm *StrongComponents* finds the strongly connected components D in two stages.

- In the first stage, *StrongComponents* performs an out-directed depth-first traversal to compute the array $PostNumInv[0{:}n - 1]$.

- In the second stage, *StrongComponents* performs an in-directed depth-first traversal, where we scan in the order $PostNumInv[n - 1], \ldots, PostNumInv[0]$ and we keep track of the DFT in-forest F using the array $InForest[0{:}n - 1]$.

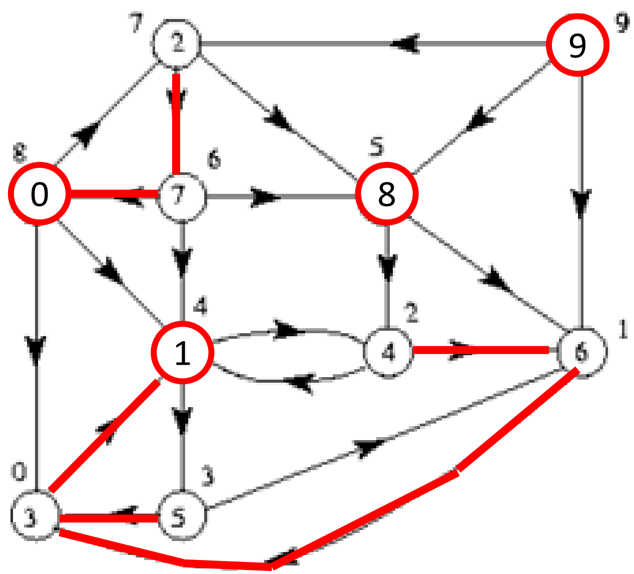
- Each tree in F spans a strongly connected component.

Pseudocode

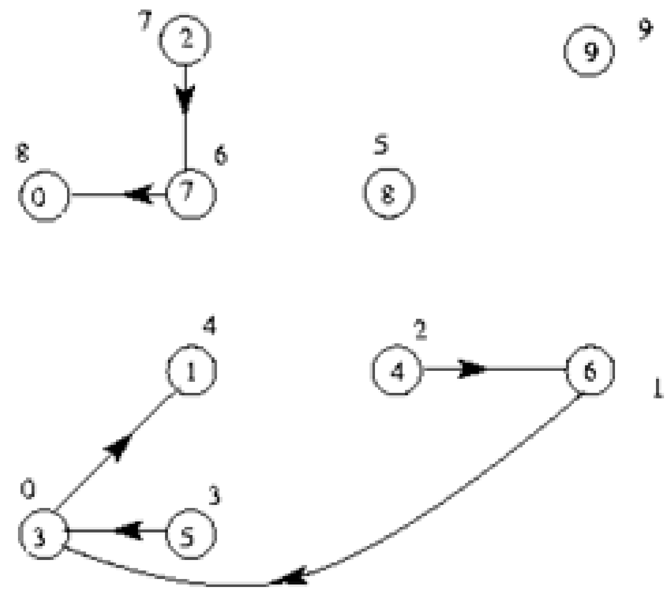
We first call *DFTout* to compute *PostNumInv*

```
procedure StrongComponents(D,PostNumInv[0:n – 1])
Input: D (a digraph with n vertices and m arcs)
          PostNumInv[0:n – 1] (PostNumInv[i] is the vertex u where
                                                        PostNum[u] = i)
Output:  InForest[0:n – 1] (an array giving parent representation of forest of
                                          in-trees T1, T2, . . . , Tk )
        dcl  Mark[0:n – 1]  //a 0–1 array, InForest[0:n – 1]
        for v ← 0 to n – 1 do
            Mark[v] ← 0
            InForest[v] ← 0
        endfor
        for i ← n – 1 downto 0 do     //perform an in-directed depth-first traversal
            v ← PostNumInv[i]         //according to reverse order of post numbers
            if Mark[v] = 0 then
                DFSInTree(D,v,InForest)  //perform an in-directed depth-first
                                         //search rooted at vertex v and store associated
            endif                        //depth-first in-tree as part of InForest[1:n]
        endfor
end StrongComponents
```

Action for Example Digraph



Digraph D with the post number shown outside each node



In–forest generated by *StrongComponents*

i	0	1	2	3	4	5	6	7	8	9
PostNum[*i*]	8	4	7	0	2	3	1	6	5	9
PostNumInv[*i*]	3	6	4	5	1	8	7	2	0	9
InForest[*i*]	-1	-1	7	1	6	3	3	0	-1	-1

Arrays *PostNum*[0:9], *PostNumInv*[0:9], and *InForest*[0:9] for the digraph *D*

Complexity Analysis

StrongComponents performs two depth-first traversals each having worst-case complexity $\Theta(m + n)$. Therefore it has worst-case complexity

$$W(n) \in \Theta(m + n).$$

Strong Interview

Interviewer : What's your biggest strength?

Student : I'm good at Machine Learning.

Interviewer : Okay, what’s 22 + 19.

Student : It’s 5.

Interviewer : Not even close. It’s 41.

Student : It’s 28.

Interviewer : I said it’s 41.

Student : It’s 39.

Interviewer : It's still 41....

Student : It’s 41.

Interviewer : Hired!