Design and Analysis of Algorithms I

Graph Primitives

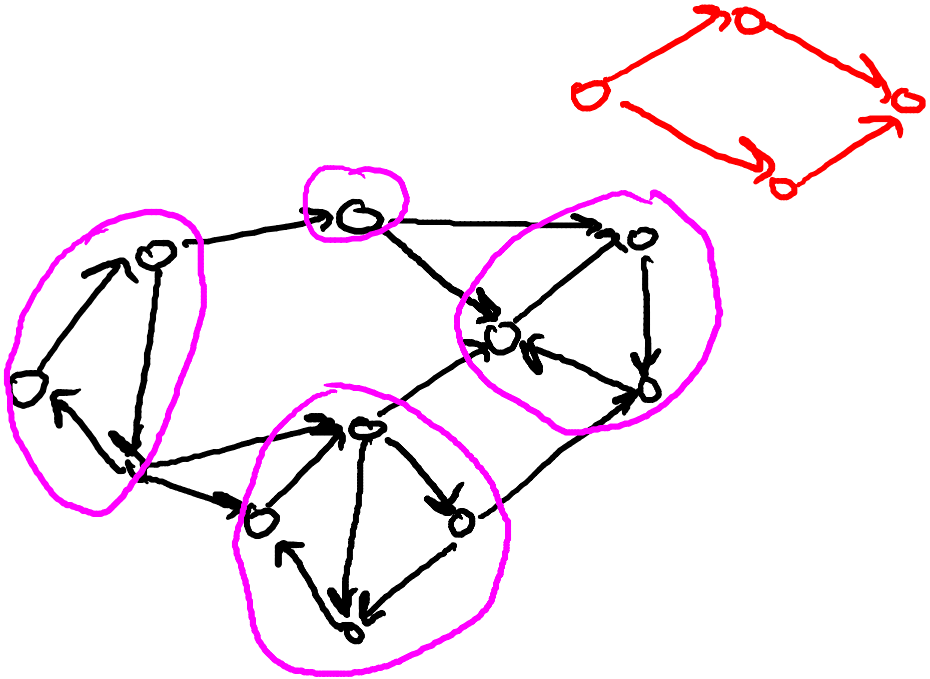
An O(m+n) Algorithm for Computing Strong Components

Strongly Connected Components

Formal Definition : the strongly connected components (SCCs) of a directed graph G are the equivalence classes of the relation

u<->v <==> there exists a path u->v and a path v->u in G

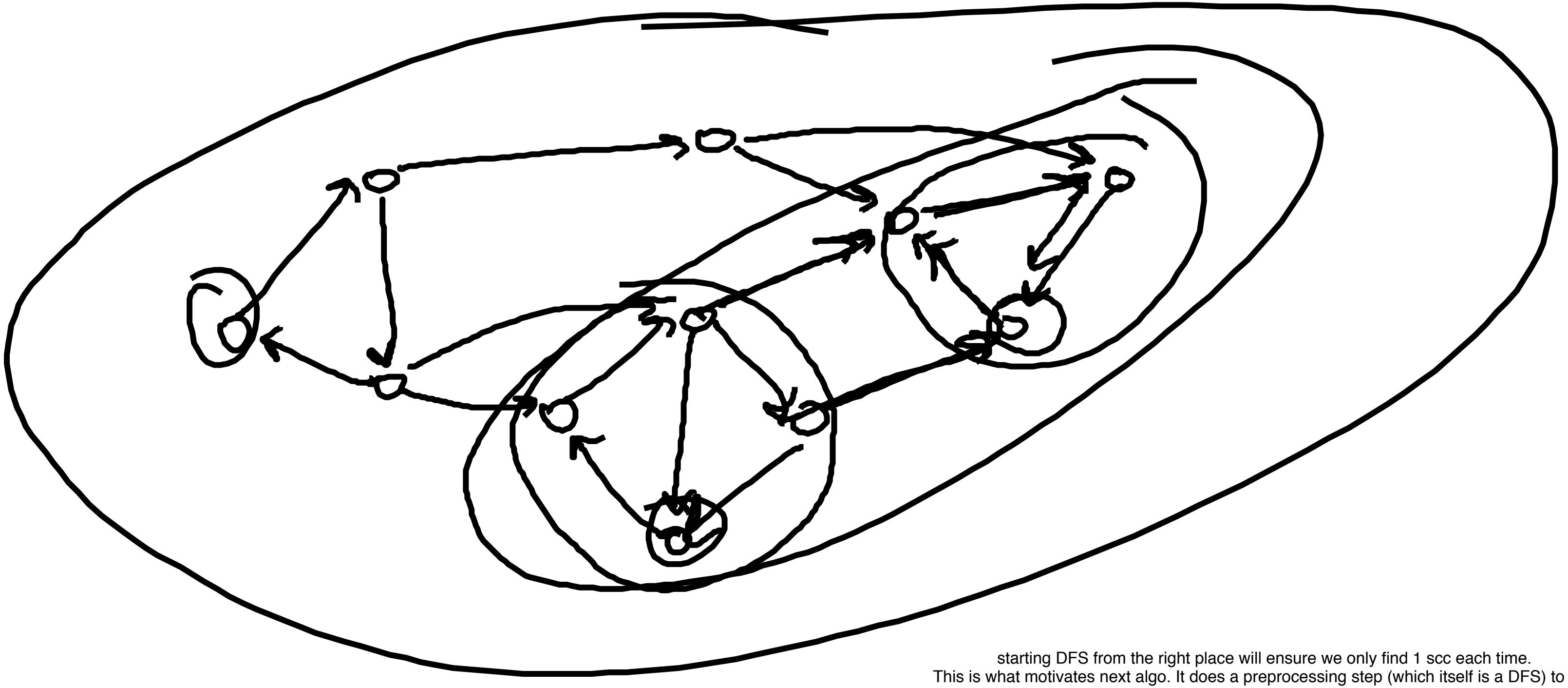
You check : <-> is an equivalence relation



Calling DFS/BFS on any of the pink circles will find 1 or more scc circles.
Depending on the nodes and kind of graph, it will find “a union of 1 or more scc.”. Goal-> just 1 SCC on each DFS pass and nothing else
In worst case it will explore the entire graph (if we start from left most edge). Output entire graph gives no insight about SCC at all
So it is important to call DFS from the right places. So that it discovers just 1 SCC on every invocation of DFS

Tim Roughgarden

Why Depth-First Search?



starting DFS from the right place will ensure we only find 1 scc each time.
This is what motivates next algo. It does a preprocessing step (which itself is a DFS) to find the places from where we should start DFS.

Kosaraju's Two-Pass Algorithm

Theorem : can compute SCCs in O(m+n) time.

Algorithm : (given directed graph G)

1. Let Grev = G with all arcs reversed
2. Run DFS-Loop on Grev

Goal : compute "magical ordering" of nodes

Let f(v) = "finishing time" of each v in V

Goal : discover the SCCs one-by-one

1. Run DFS-Loop on G

processing nodes in decreasing order of finishing times

[SCCs = nodes with the same "leader"]

DFS-Loop

<u>DFS-Loop (graph G)</u>
Global variable t = 0
[# of nodes processed so far]
Global variable s = NULL
[current source vertex]
Assume nodes labeled 1 to n
For i = n down to 1
    if i not yet explored
        s := i
        DFS(G,i)

For finishing times in 1st pass
on the reverse G

For leaders in 2nd pass
on the forward G

DFS (graph G, node i)
-- mark i as explored
-- set leader(i) := node s
-- for each arc (i,j) in G :
    -- if j not yet explored
        -- DFS(G,j)
-- t++
-- set f(i) := t

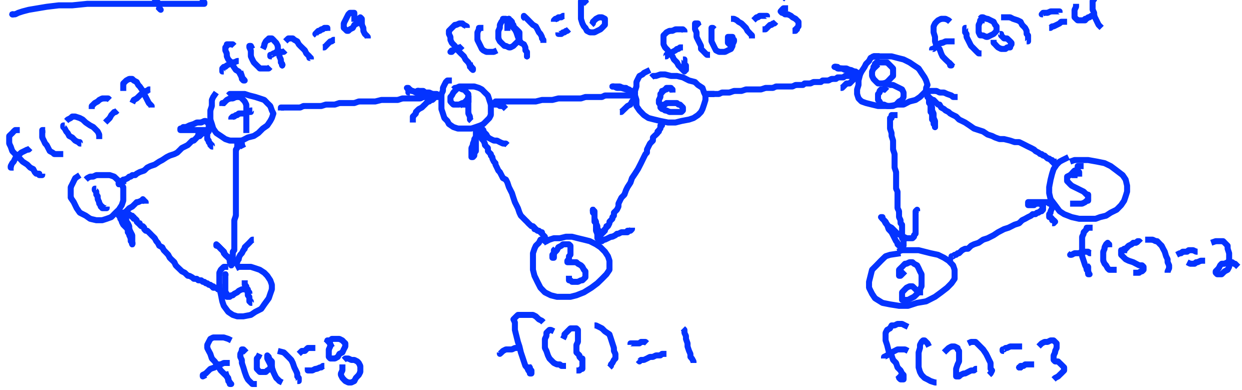
For rest of DFS-Loop

i's finishing time

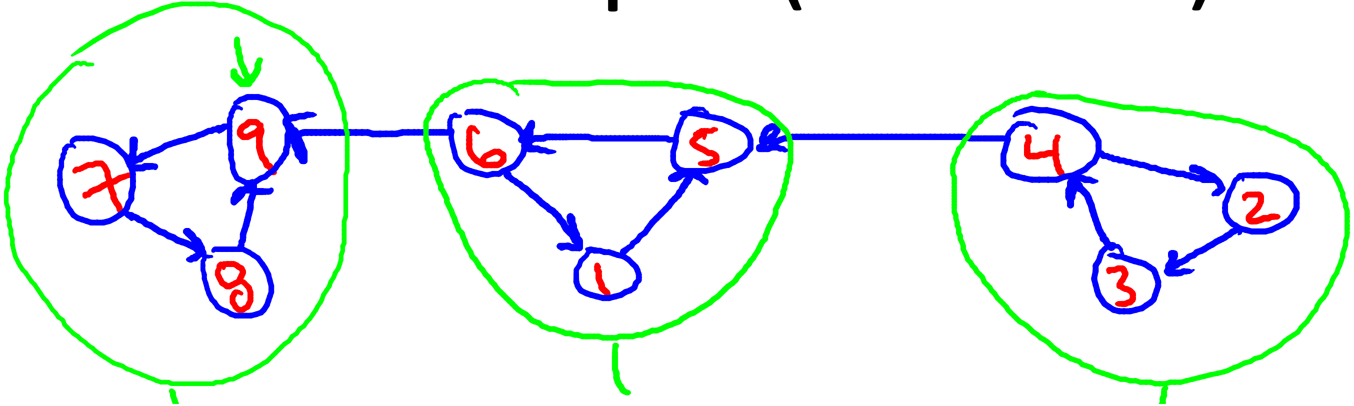
Only one of the following is a possible set of finishing times for the nodes 1,2,3,…,9, respectively, when the DFS-Loop subroutine is executed on the graph below. Which is it?

Example: [1st DFS-Loop on Grev]

- 9,8,7,6,5,4,3,2,1
- 1,7,4,9,6,3,8,2,5
- 1,7,9,6,8,2,5,3,4
- 7,3,1,8,2,5,9,4,6



Example (2^{nd} Pass)



Leader = 9

Leader = 6

Leader = 4

Replace node number by finishing times from the 1st pass and fix the edge directions back to original

Running Time : 2*DFS = O(m+n)