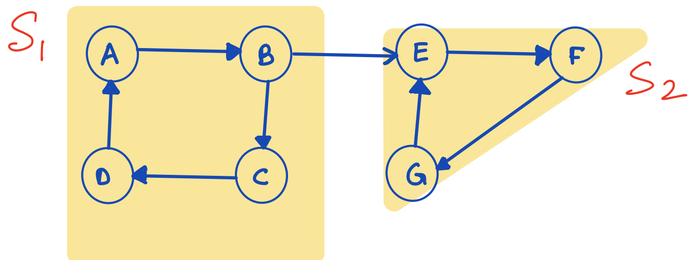
COL 351: Analysis and Design of Algorithms

Lecture 9

SCCs in Directed Graph

Definition: Two vertices x, y are "strongly-connected" if there is x to y, as well as y to x path in G.



Strongly-connected-component (SCC):
Maximal subset S of G such that all vertices in S are strongly connected.

Trivial algorithm to compute SCCs:

For each x compute:

$R_{out}(x)$ = vertices reachable from x

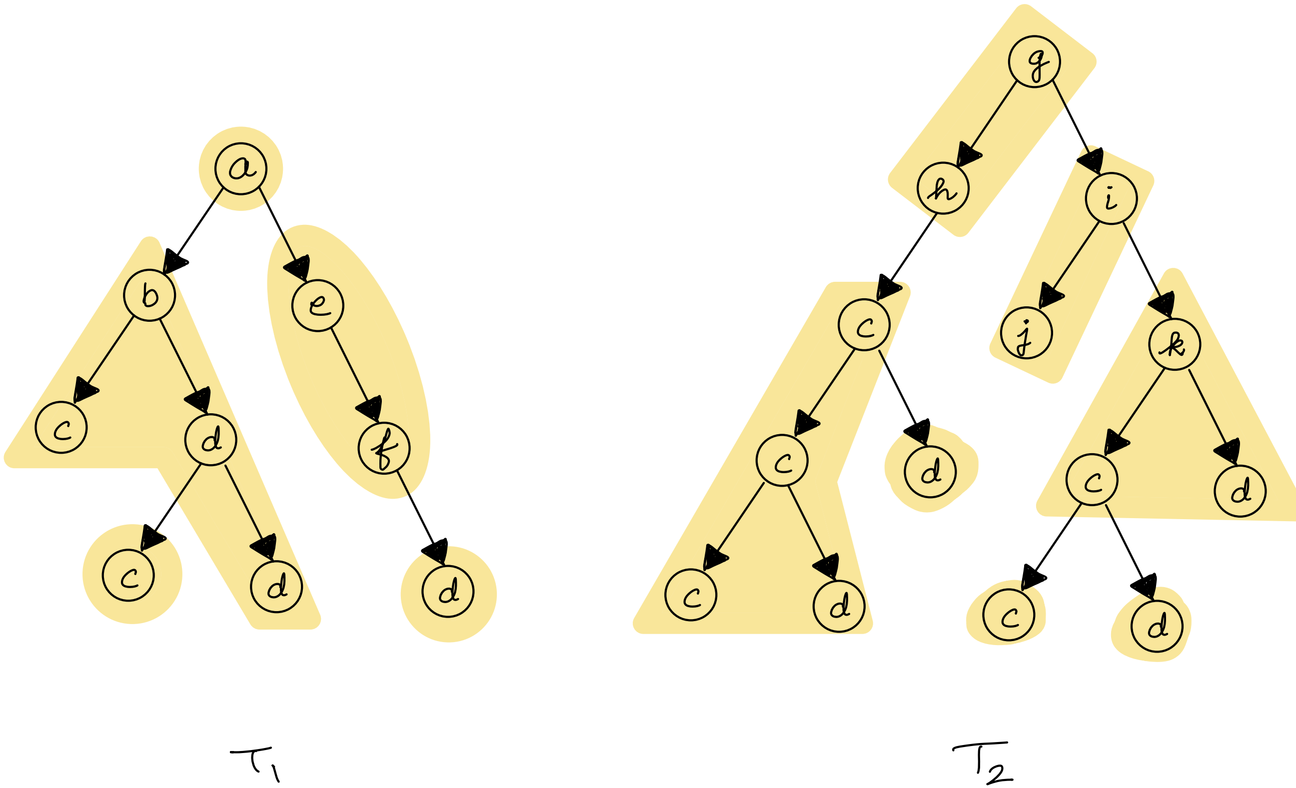
$R_{in}(x)$ = vertices having path to x

Then, $SCC(x) = R_{out}(x) \cap R_{in}(x)$

Total time $= O(mn)$

Structure of SCCs

Theorem: Let **T** be a DFS tree of G and ***S*** be an SCC in G, then **T**[***S***] is a contiguous subtree of **T**.



Consider two vertices y, z in an SCC "S".

$\boxed{\text{Claim 1: If } y \text{ is ancestor of } z\text{. Then vertices of } \text{TREEPATH}(y, z) \in S\text{.}}$

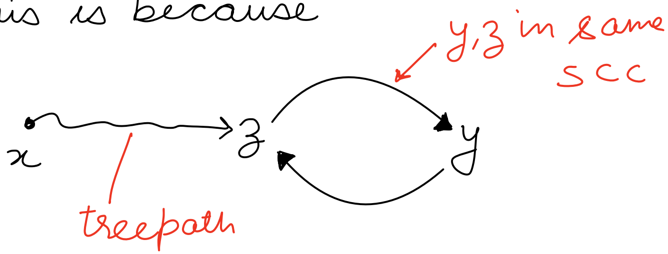
<u>Proof:</u>

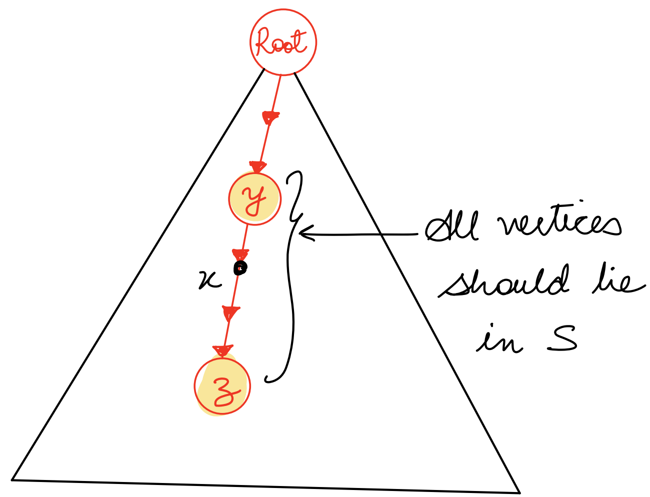
Consider a vertex $x \in \text{treepath}(y, z)$

1. $\exists\ y \rightsquigarrow x$ path (Trivial proof)

2. $\exists\ x \rightsquigarrow y$ path.

This is because





Consider two vertices y, z in an SCC "S".

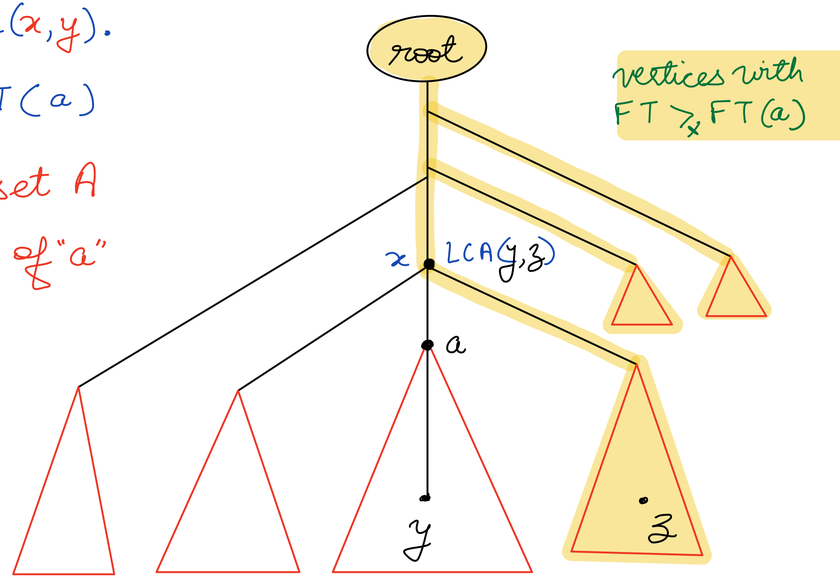
Claim 2: If y, z do not have ANCESTOR-DESCENDANT relationship.
Then $\exists$ at least one common ancestor of y, z that lie in S.

Proof

Let $x = LCA(y, z)$ be lowest common ancestor of y, z. (Suppse y is visited before z.)

Let a = child of x on treepath(x, y).

Def $A :=$ Vertices with $FT \le FT(a)$



Claim: Any edge going out from set A terminates to an ancestor of "a"

Proof: Non-tree edges can't be anti-cross edge.

Let "P" be a $y \rightarrow z$ path.

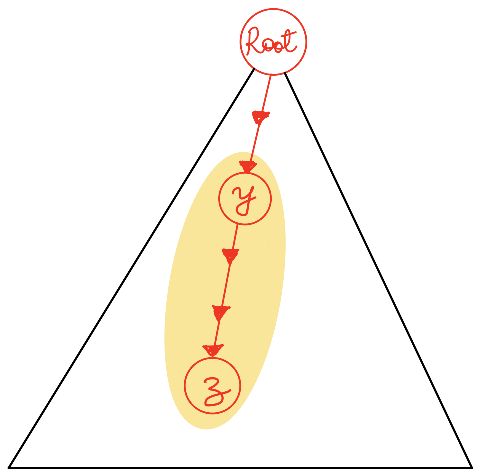
By above claim, P passes through a common ancestor of y, z (say x')

Then $y \underset{\text{due to P}}{\rightsquigarrow} x' \underset{\text{treepath}}{\rightsquigarrow} z \underset{\text{same SCC}}{\longrightarrow} y$ (so x', y, z are in same SCC "S")

Structure of SCCs

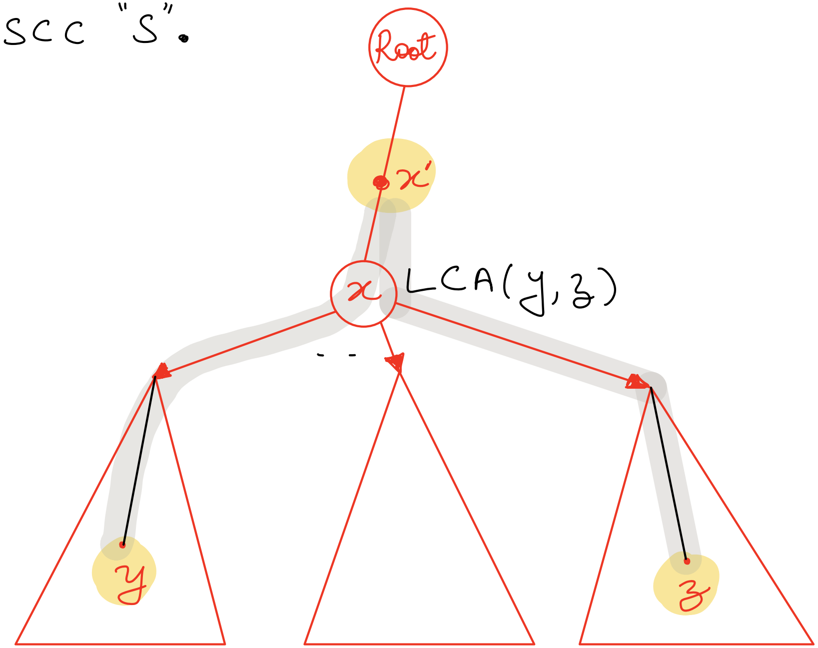
Theorem: Let **T** be a DFS tree of G and ***S*** be an SCC in G, then **T**[***S***] is a contiguous subtree of **T**.

Proof: Suppose y, z lie in an SCC "S".



(i) In this case we have

$\text{treepath}(y, z) \in S$



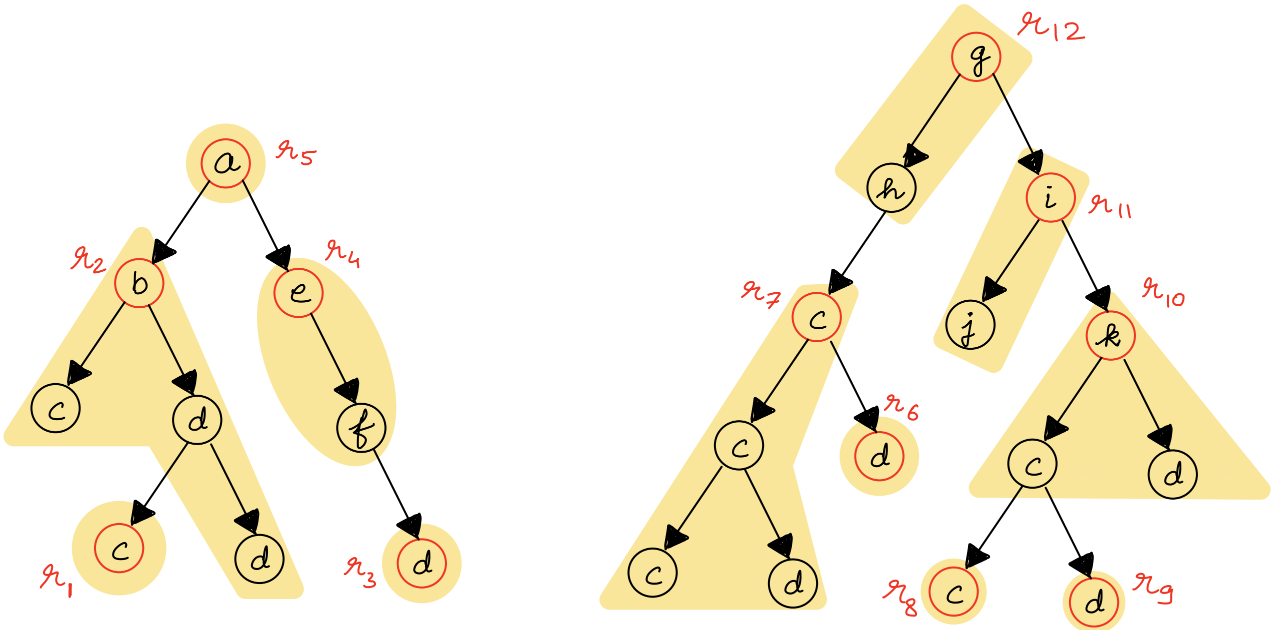
(ii) In this case we have

$\text{treepath}(x', y) \in S$

$\text{treepath}(x', z) \in S$

Computing SCCs

Concept of SCC's root: The vertex of SCC that is visited first by DFS traversal.

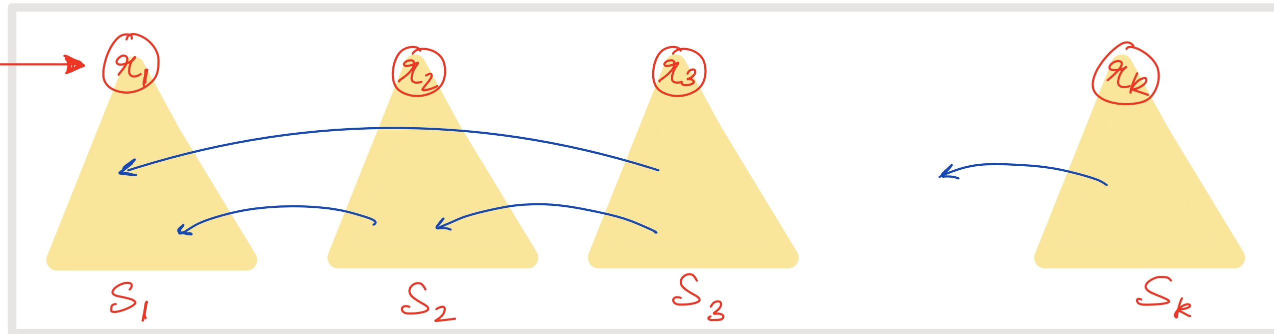


Remark 1: $r_1 \dots r_{12}$ are roots of SCCs in increasing order of F.T.

Remark 2: r_{12} is also vertex of largest F.T. in G.

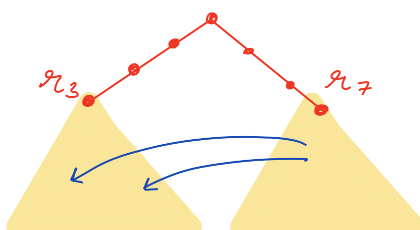
Theorem: Let $S_1 \dots S_k$ be SCCs of G, and let $r_i = Root(S_i)$ in DFS. If $FT(r_1) < \dots < FT(r_k)$, then for any edge $(x,y) \in S_i \times S_j$, we have $i > j$.

Roots of SCCs in increasing order of FT.



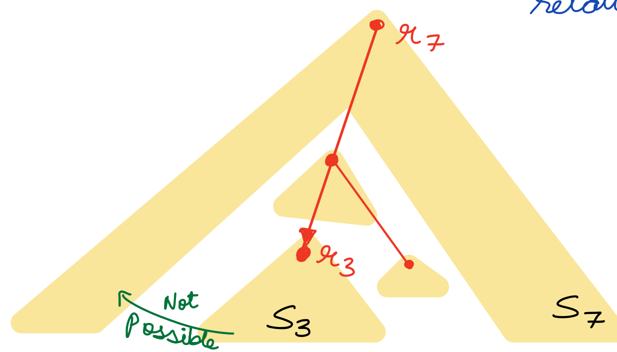
Proof of Theorem:

CASE 1: Roots don't have ancestor descendant relationship



In this case all edges are cross edges, so from $T(r_7)$ to $T(r_3)$

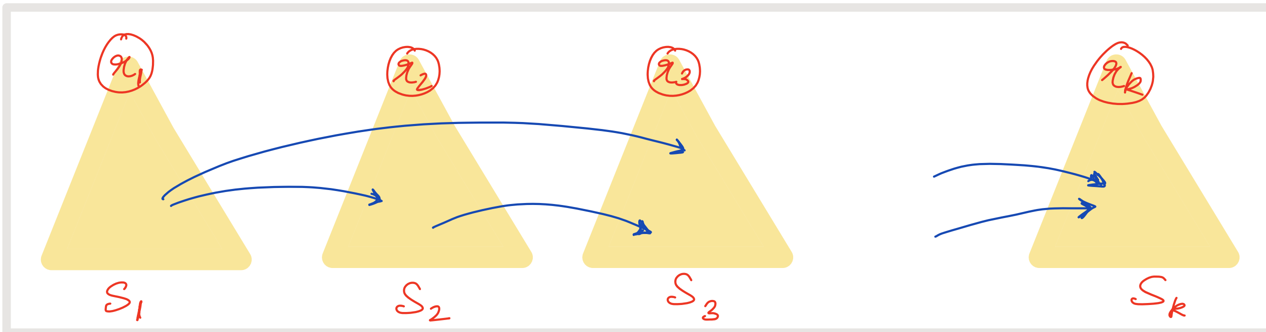
CASE 2: Roots have ancestor-descendant relationship



As $r_7 \rightsquigarrow r_3$, we can't have any edge from S_3 to S_7 b'coz otherwise it will contradict that r_3, r_7 are in different SCCs

Theorem: Given the roots $r_1, \ldots, r_k$ of SCCs of G satisfying $FT(r_1) < \cdots < FT(r_k)$, we can find SCCs of G in $O(m+n)$ time.

Depiction of inter-SCC edges in graph G^R.



Algorithm:

1. Mark all vertices as Unvisited.

2. For $i = k$ to 1:

    - Compute S_i = Vertices reachable from r_i in Unvisited part of G^R.

    - Mark vertices of S_i as visited.

3. Return $(S_1 \ldots S_k)$

G^R is graph obtained on reversing edges of G

Proof:

Suppose we have marked vertices of $S_{i+1}, \ldots, S_k$ as VISITED, for some $i \leq k$.

Then, the only vertices reachable from r_i in G^R that are yet UNVISITED will be those in SCC S_i

This proves algo is correct.

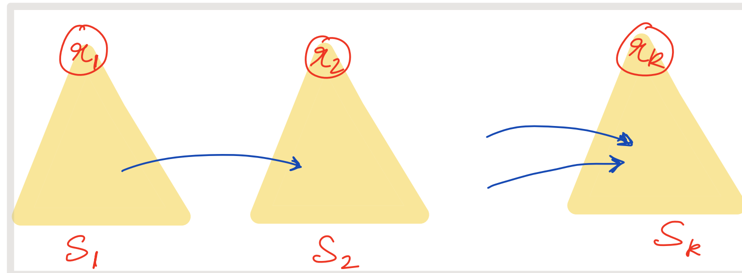
Time complexity is $O(m+n)$.

Theorem: Given the roots $r_1, \ldots, r_k$ of SCCs of G satisfying $FT(r_1) < \cdots < FT(r_k)$, we can find SCCs of G in $O(m+n)$ time.

Algorithm:

1. Mark all vertices as unvisited.
2. For $i = k$ to 1:
   - Compute S_i = Vertices reachable from r_i in Unvisited part of G^R.
   - Mark vertices of S_i as visited.
3. Return $(S_1 \ldots S_k)$

G^R is graph obtained on reversing edges of G



Question: How to find r_i, given $(r_{i+1}, \ldots, r_k)$ and $(S_{i+1}, \ldots, S_k)$?

Ans:

For $i \leq k$, r_i is vertex of largest FT in graph $G \setminus (S_{i+1} \ldots S_k)$.

This is because FT of vertices in $S_j \leq FT(r_j)$ for $j = 1$ to k.

So, if L = vertices of G in decreasing order of FT, then the first vertex in L lying in $G \setminus (S_{i+1} \ldots S_k)$ will be r_i, for $i \leq k$.

Computing SCCs in Directed graph (Kosaraju's Algorithm)

Step 1 : Perform DFS traversal on G.

Step 2 : Let L = vertices of G in decreasing order of finish time.

Step 3 : Perform tree traversal on Reverse(G), by picking new roots in ordering defined by L.

Step 4 : We have shown that the trees computed by second traversal correspond to the SCCs of G, so we simply output them.