Lecture 12

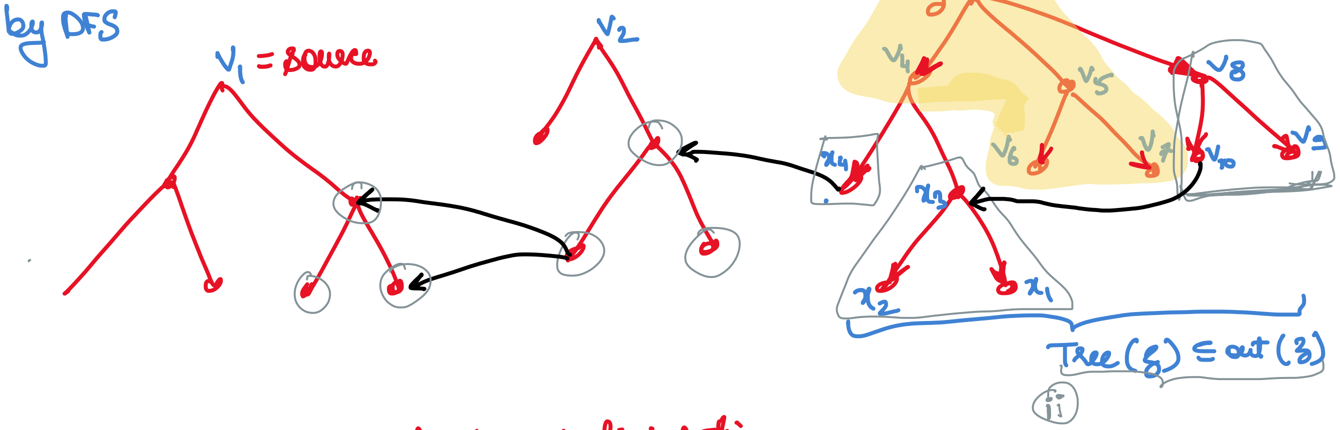
Trees (3 trees) obtained by DFS

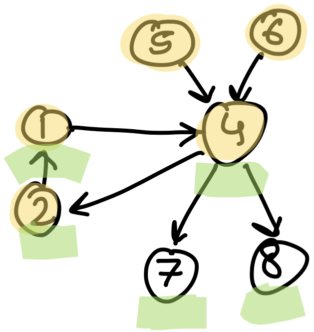
$SCC(z) = IN(z) \cap OUT(z)$

$C \leftarrow$ collection of SCCs (initialized to empty)

$IN(v)$ = vertices having path to $v$

$OUT(v)$ = vertices reachable from $v$




Let $z$ be vertex with largest finish time

Find $IN(z)$ = vertices having path to $z$

Vertices in order of finish time

| | | | | | | |
|---|---|---|---|---|---|---|
| ..... | $v_7$ | $v_5$ | $v_{10}$ | $v_9$ | $v_8$ | $v_3$ |

← Array A

CLAIM: $IN(z) = SCC(z)$

Proof: As there are no left → right cross edges, $\boxed{IN(z) \subseteq Tree(z)}$. (i)

Tree(z) has no incoming edges from trees lying in left side.

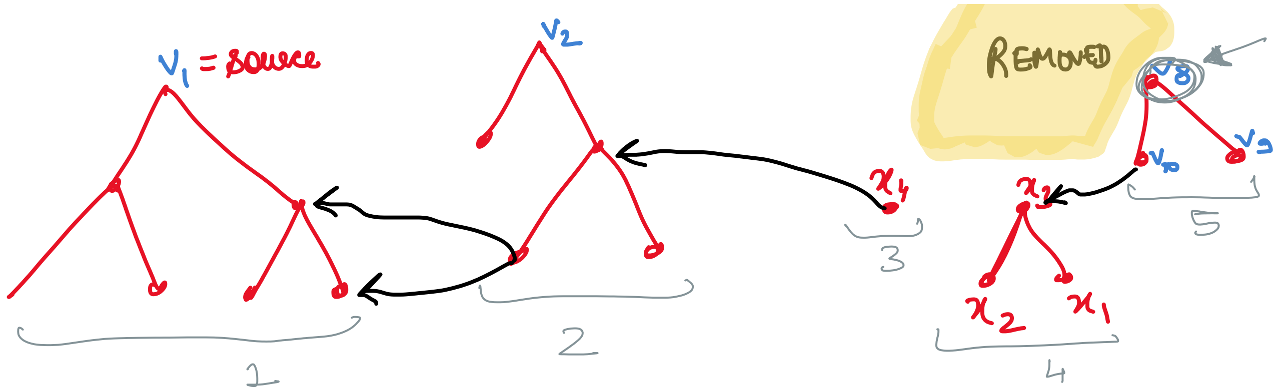
From (i), (ii) $\Rightarrow In(z) \subseteq out(z)$

$\Rightarrow In(z) = SCC(z)$.

Suppose we remove SCC(z) from G:

Ordering of roots

$v_1$




Start time of original DFS traversal.

$v_2$
$x_4$
$x_3$
$v_8$

CLAIM 1: Let $F$ be forest of DFS tree, and $z$ be vertex of largest finish time.
Then $F \setminus SCC(z)$ will form DFS tree collection of graph $G \setminus SCC(z)$.

Repeat Same Idea
└ compute SCC of vertex with Highest Finish time & remove SCC(z) from G.

ALGORITHM-SKETCH

① Perform DFS traversal and compute an array A where vertices are ordered in increasing order of Finish-Time.

② Set all vertices as Unmarked, and initialize $\mathcal{C} = \{\}$. To store sets corresponding to SCCs

③ While (A) contains unmarked vertex:

$z$ = Unmarked vertex with highest finish time. ( $z$ is rightmost unmarked vertex ) ii Find $IN(z)$ = vertices having path to $z$ in "current" G.

iii Add set '$IN(z)$' to $C$.

iv Remove vertices of $IN(z)$, and their incident edges from G.

v Mark vertices of $IN(z)$

Property: The order of finish time of vertices don't change on removing SCC of vertex with largest finish time.

Proof: (By Claim 1)

# DYNAMIC PROGRAMMING

## NEW - PROBLEM

Example of sequence: $S = (a, c, b, c, d, a)$

Subsequence of S = sequence obtained by deleting some elements from S.

Eg. $(a, b, d, a)$ is subsequence of S.

... two sequences $A = (a_1\ a_2 .. a_n)$ and $B = (b_1\ b_2 .. b_m)$, find

**Problem:** Given two sequences ... find

a Longest-Common-Subsequence (LCS) of A and B.

Eg. $A = (\underline{a}, c, \underline{b}, \underline{c}, \underline{d}, a)$
$B = (c, \underline{a}, \underline{b}, a, \underline{c}, \underline{d})$ $\Big\}$ $(a, b, c, d) = LCS(A, B)$

A[n-1]
$\boxed{a\ b\ c\ c\ d\ c}$ b

$\boxed{b\ c\ a\ a\ d\ a\ c}$ b
B[m-1]

**Simple Recursive Algo:**

LCS(A, B, n, m):

If (A[n] = B[m]): Return LCS(A, B, n-1, m-1) • A[n]

(Assumption: indices are starting from 1)

Else:

ans1 = LCS(A, B, n, m-1)
ans2 = LCS(A, B, n-1, m)

WILL INVOKE $\boxed{LCS(A, B, n-1, m-1)}$

a b c d c ~~b~~
b c c a b ~~a~~

If LENGTH(ans1) > LENGTH(ans2): Return ans1
Else: Return ans2

Time-complexity $\stackrel{?}{=}$ O(mn) No

**Time complexity**

$$T(n, m) = \begin{cases} T(n-1, m-1) + 1 & \text{if } A[n] = B[m] \\ T(\ldots) + T(n, m-1) + 1 & \text{o/w.} \end{cases}$$

$\left( T(n-1, m) + T(n, m-1) \right.$

$\boxed{\text{H.W.1}}$ $T(n, m) = \underline{O(2^{n+m})}$

It is too large b coz we are SOLVING same problem multiple times.

## Dynamic Programming:

- Break problem into subproblems.
- Don't solve subproblem again & again. Instead STORE solution of subproblems.

$\boxed{\text{H.W. 2}}$ Modify the algorithm to have polynomial time complexity.