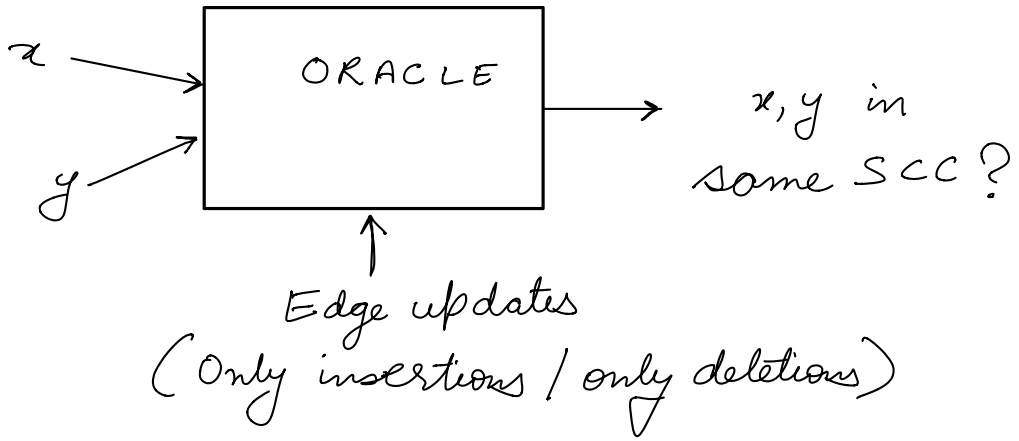
Incremental / Decremental SCC Oracle



Incremental Setting

SSR – $O(m)$ total time

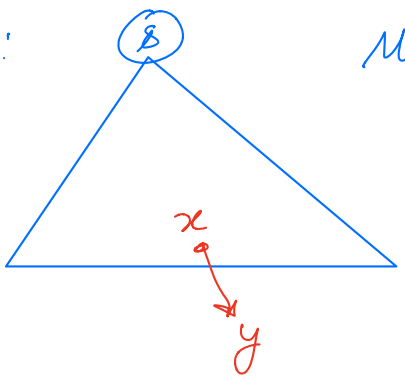
∴, APR – $O(mn)$ total time

SCC – $O(mn)$ total time

(we just need to maintain SSR from all vertices in G.)

How to solve incremental SSR (Single source reachability) in $O(m)$ total time?

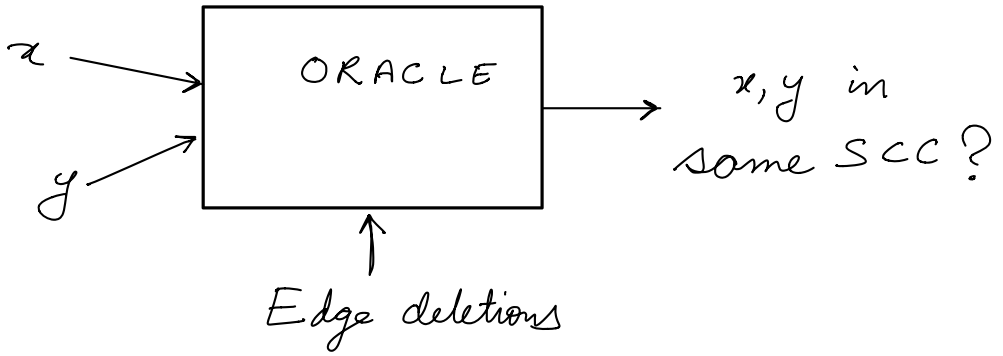
Sol^n:



Maintain vertices reachable from s.

On addition of $e=(x,y)$ such that only x was previously reachable, perform a tree-traversal from y over only those vertices that are first time becoming reachable from s.

Decremental setting



Simpler Problem (Decremental)

Initially : Suppose G is an SCC

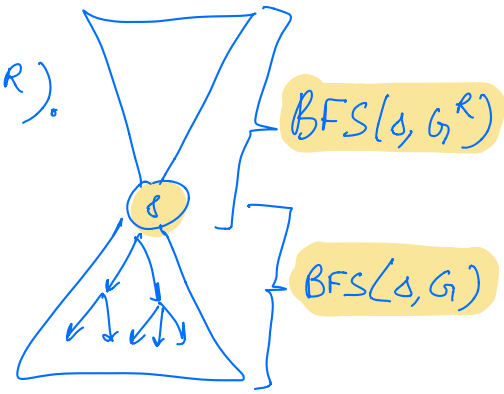
Maintain : Whether G remains an SCC or not. (Yes/No)

An $O(mn)$ total time algorithm is :

(1) choose $s \in V(G)$.

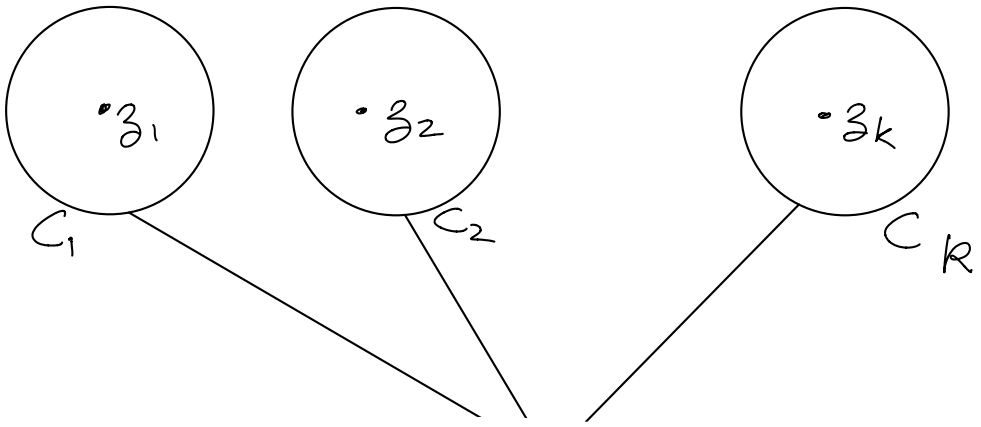
(2) Maintain $BFS(s, G)$ and $BFS(s, G^R)$.

(3) Output 'Yes' if both BFS trees have n vertices, and 'No' if either of trees has $\leq n-1$ vertices



Main Idea:

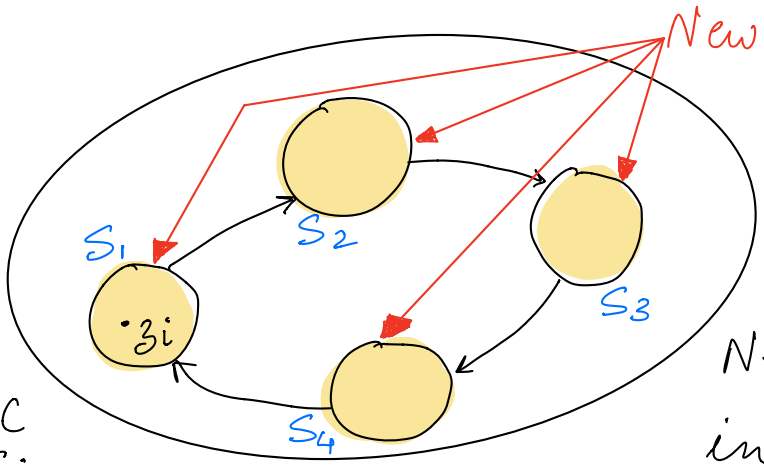
Initially:



SCCs in graph G

(maintain $BFS(z_i, G[C_i])$ and $BFS(z_i, G^R[C_i])$)

On an edge deletion C_i can split:



SCC C_i of size n'

New SCC can be found in $O(m)$ time after split

- Such an event occurs at most $(n-1)$ times

* In exactly one child (new) SCC vertex z_i lies
* At most one child (new) SCC has size $\geq \frac{n'}{2}$
* In the child (new) SCCs that don't contain z_i, we will choose new root and maintain "in" and "out" BFS.

Decremental SCC Algorithm :

Initialization Phase :

(1) Compute the partitioning $\mathcal{C} = \{C_1, C_2 \ldots C_k\}$ of SCCs

(2) Delete from G edges lying b/w SCCs : $C_1, C_2 \ldots C_k$

(3) For $i = 1$ to $|\mathcal{C}|$:

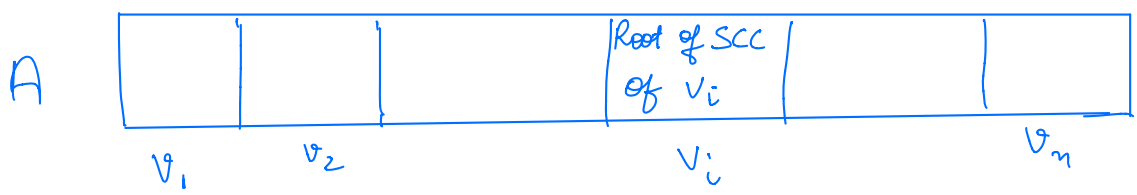
Choose a random vertex z_i in C_i,

Set z_i to be root of C_i.

Initialize and maintain $BFS(z_i, G)$, $BFS(z_i, G^R)$.

(4) Compute an array A of size n such that

$A[q]$ = root of SCC of vertex v_q, $1 \le q \le n$.



—— x ——

Deletion Subroutine ($e=(u,v)$)

① If $A[v] \neq A[u] \Rightarrow$ Return.

② Let $z = A[v]$ be root of SCC of v.

③ C_0 = Vertices in $BFS(z, G)$ } Total time of this step $= O(n) \times m$.

(C_0 is old SCC of z as we don't keep inter SCC edges.)

④ Delete edge (u,v) from $BFS(z, G)$, $BFS(z, G^R)$

⑤ If size of tree $BFS(z,G)$ or $BFS(z,G^R)$ decreases:

→ Compute new (child) SCCs $S_1 \dots S_\alpha$ } Total time of this step $= O(m) \times n$.

→ Remove all edges lying b/w $S_1\ S_2 \dots S_\alpha$
and update $BFS(z, G)$ and $BFS(z, G^R)$

→ For $i=1$ to α :

If $z \notin S_i$:

→ choose a random root $w_i \in S_i$

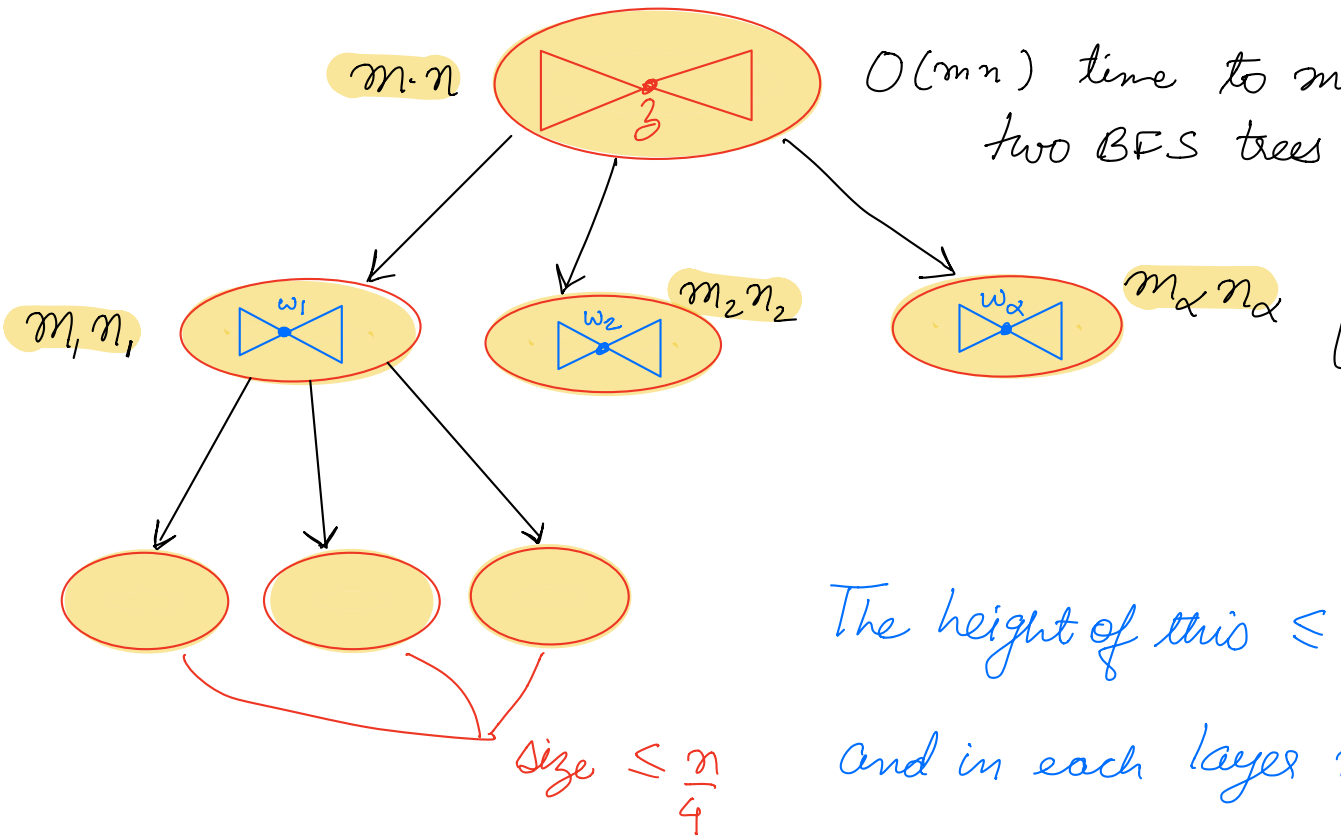
→ $\forall$ vertices in S_i update entry in A to root w_i

→ Compute & decrementally maintain
$BFS(w_i, G)$ and $BFS(w_i, G^R)$.

———— × ————

Before analyzing above algorithm we will analyze some special cases.

SPECIAL CASE 1: Whenever an SCC splits the new components have size at most half of original SCC.

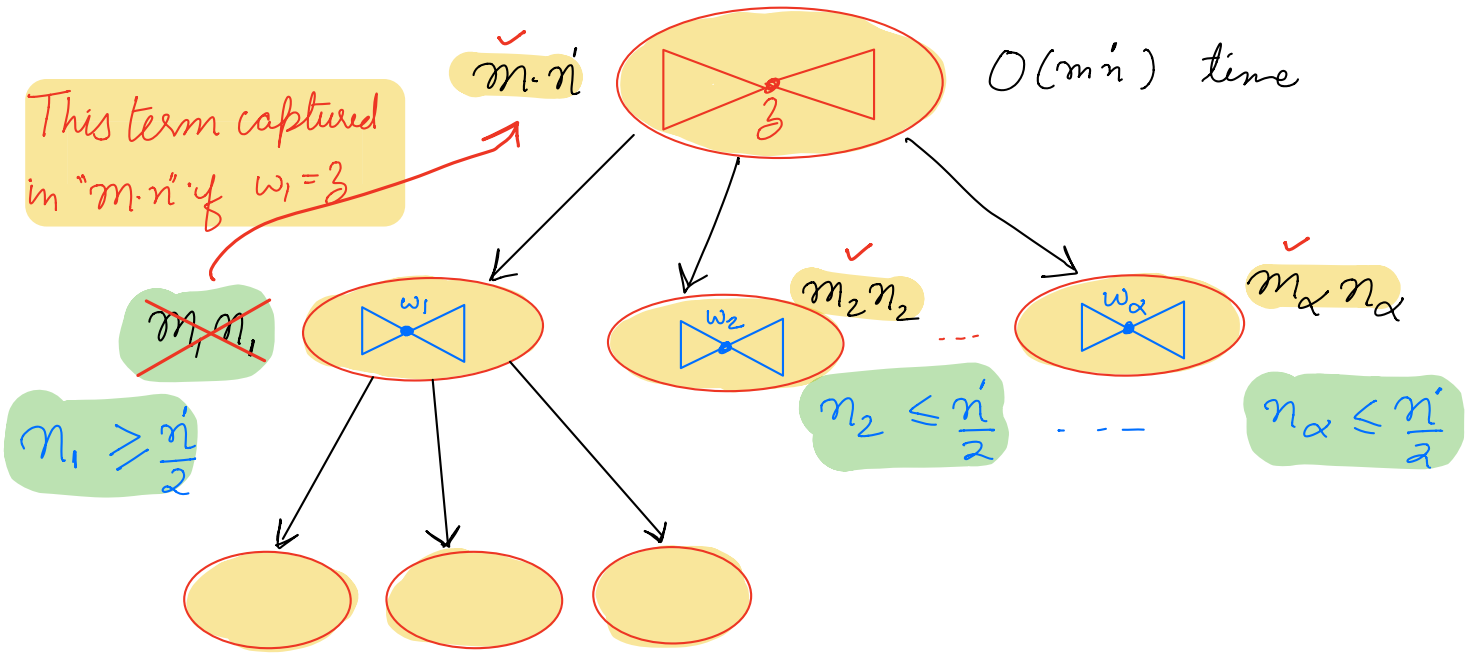


The height of this $\leq \log_2 n$ and in each layer we spent in total $O(mn)$ time

$\Rightarrow$ Total time spent $= O(mn \log n)$.

can we eliminate this?

SPECIAL CASE 2: Whenever an SCC splits the root of original SCC goes into largest component.

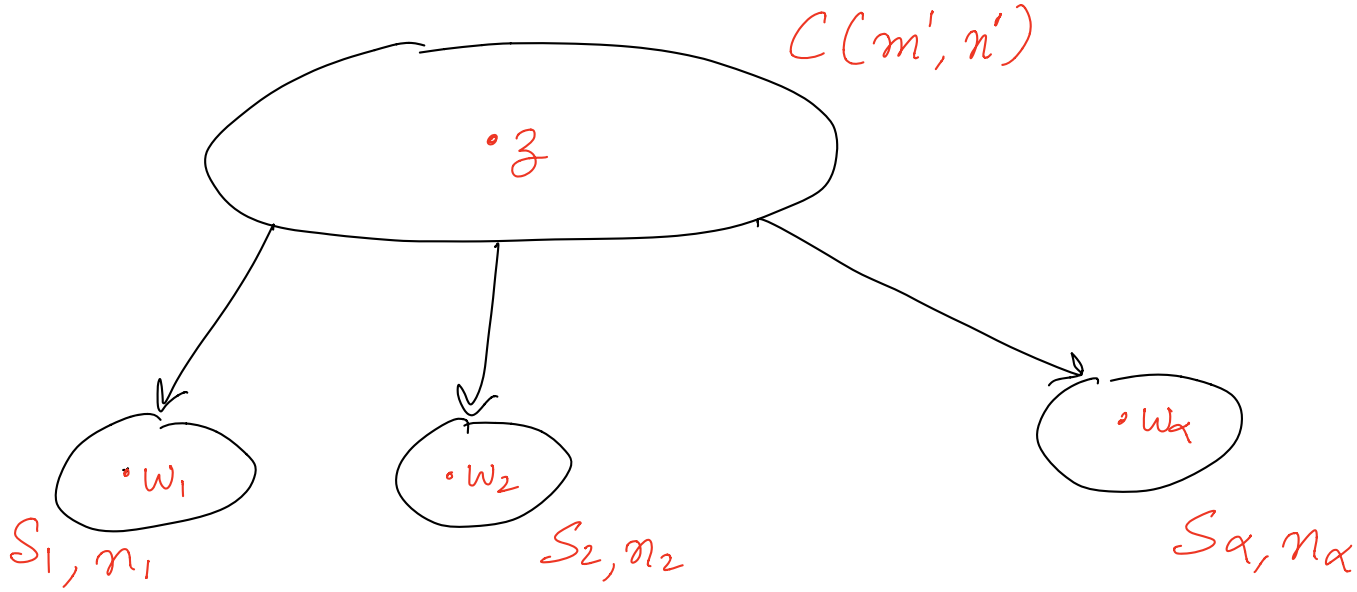


Note: With probability half (at least) 3 is passed to component having size $\geq \frac{n'}{2}$.

We next perform analysis of Algorithm

Analysis :

* Cost of maintaining BFS trees in different component is main bottle neck.



$$n' = n_1 + n_2 + \dots + n_\alpha$$

Expected cost of maintaining S_i

$$= \underbrace{\left(1 - \frac{n_i}{n'}\right)}_{\text{Prob}(z \notin S_i)} * m_i n_i \quad \leq \left(\frac{n' - n_i}{n'}\right)^{(m_i)(n)}$$

So, each edge of S_i gets cost $= \left(\frac{n' - n_i}{n'}\right) \cdot n$

In general, suppose edge e lies in t components in total:

$$C_1 \longrightarrow C_2 \longrightarrow C_3 \longrightarrow \cdots \longrightarrow C_t$$

$$(n_1) \quad (n_2) \quad (n_3) \quad (n_t)$$

$$\left(\frac{n_1 - n_2}{n_1}\right) \cdot n \qquad \left(\frac{n_2 - n_3}{n_2}\right) \cdot n \qquad \left(\frac{n_{t-1} - n_t}{n_{t-1}}\right) \cdot n$$

Total expected cost over e $= \left(\left(\frac{n_1 - n_2}{n_1}\right) + \cdots + \left(\frac{n_{t-1} - n_t}{n_{t-1}}\right)\right) \cdot n$

$$\frac{n_1 - n_2}{n_1} \leq \underbrace{\frac{1}{n_1} + \frac{1}{n_1 - 1} + \frac{1}{n_1 - 2} + \cdots + \frac{1}{n_2 + 1}}_{(n_1 - n_2) \text{ terms}}$$

$$\frac{n_2 - n_3}{n_2} \leq \frac{1}{n_2} + \frac{1}{n_2 - 1} + \frac{1}{n_2 - 2} + \cdots + \frac{1}{n_3 + 1}$$

$(n_2 - n_3)$ terms

Total expected cost over c $\leq \left(\frac{1}{n_1} + \frac{1}{n_1 - 1} + \cdots + 1\right) \cdot n = O(n \log n)$

Thus, expected time complexity of algorithm $= O(mn \log(n))$.