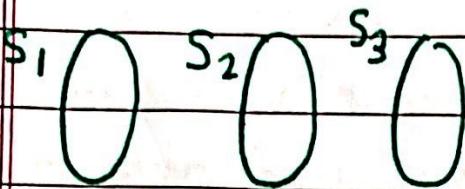


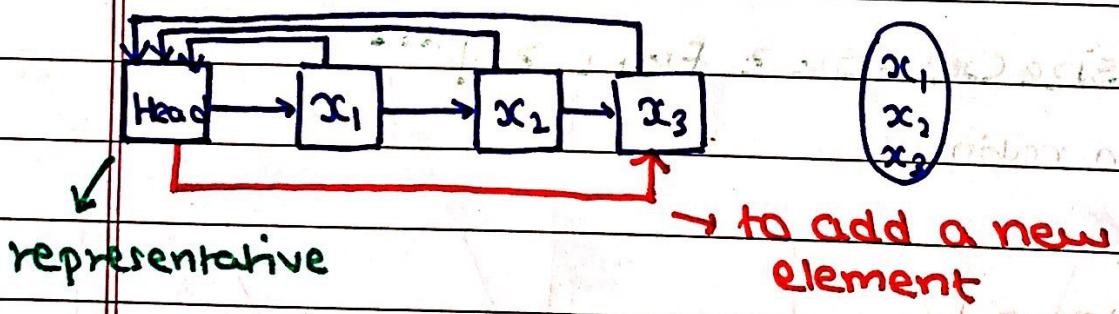
* Disjoint Set Forest



* ADT:

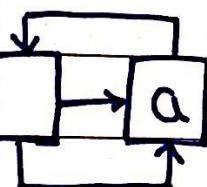
- makeset(x) ... $O(1)$ } \rightarrow n-operations
- findset(x) ... $O(1)$ } \rightarrow m-operations
- union(x, y) ... $O(k)$

Naive : $O(mk)$ - all union

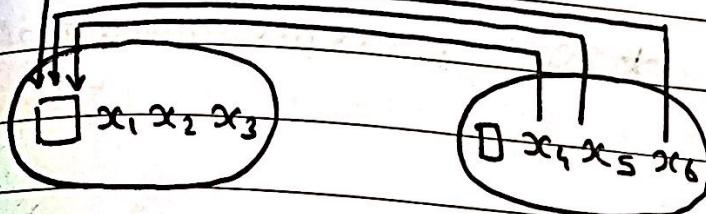


LINKED LIST

* Makeset(a):



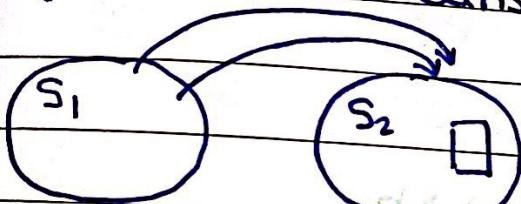
* Union (x, y)



k elements

$O(k)$

→ Weighted Union Heuristic

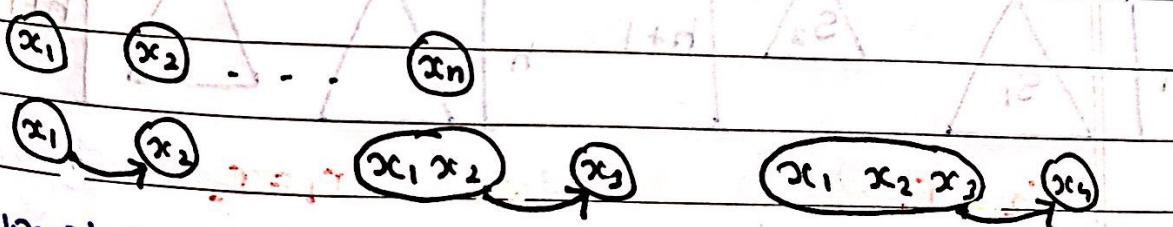


$|S_1| < |S_2|$

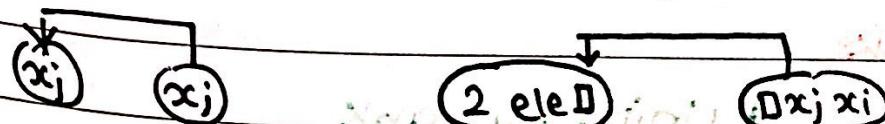
* No. of pointer change : Bh to merge S_1 to S_2

→ makeset : n then union (x, y) : $n-1$ times

→ No weighted union heuristic : $O(m+n^2)$



→ Worst case: $1 + 2 + 3 + \dots + (n-1) = O(n^2)$

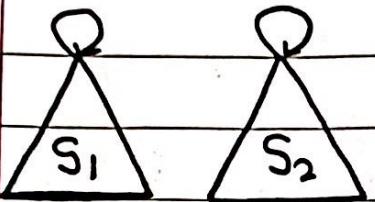


pointer of
 x_j changed
at max $\log n$
times

Weighted Union Heuristic: $O(m+n\log n)$

n choices for
values of j

* Set is represented as a tree (Root point) *

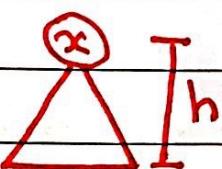


* root as representative

* Makeset :

* rank of a node:

Upper bound on height of subtree rooted at x.

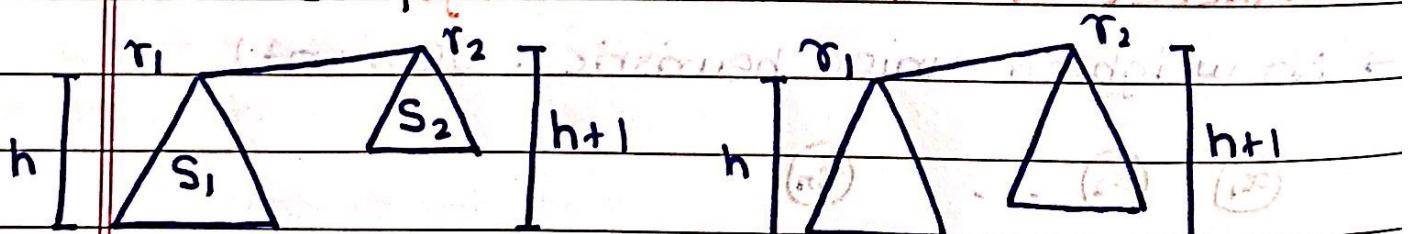


$\text{rank}(x) \geq h$

* Heuristics: if $r_1 > r_2$ spread a single rank to all

→ Union By Rank:

→ Path Compression:



→ No need to change then $r_2 = r_2 + 1$ for the ranks

* Union By Rank

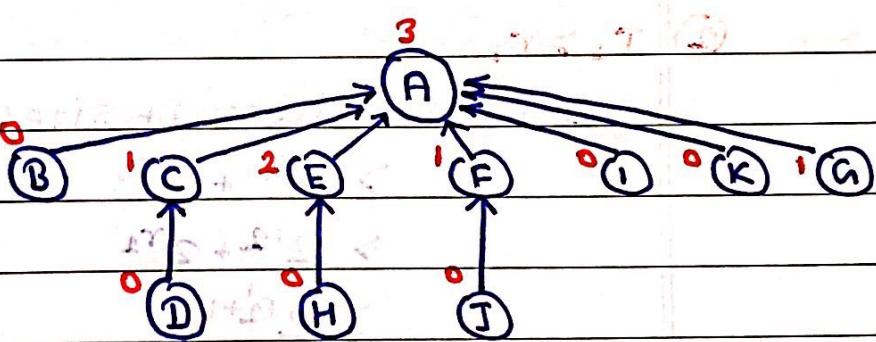
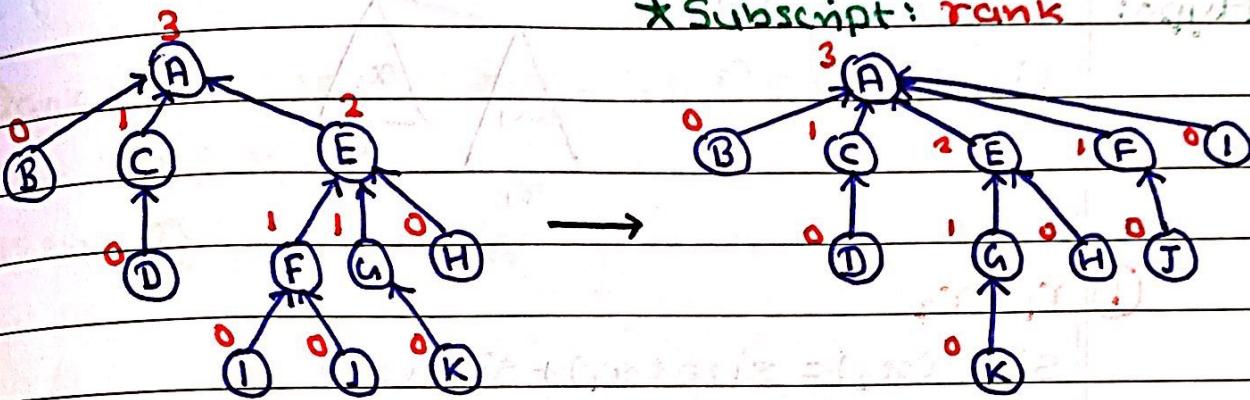
*Path Compression: ~~can do one step at a time~~

$\pi(x)$: parent node of x

Actually we promote $x, \pi(x), \pi(\pi(x)), \dots$

Find(i) followed by Find(k) ~~is same as~~ $\pi(i) = k$

*Subscript: rank



*For root node rank is same as depth

*Once a node becomes a non-root, its rank stays fixed

→ even when path compression decreases its depth

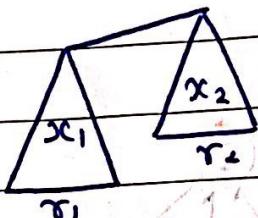
* For any node x

$$\text{size}(x) \geq 2^{\text{rank}(x)}$$

Basis: (1) $\text{rank} = 0 \Rightarrow \text{size}(x_0) = 1$ (by definition)

Hypo: $\text{size}(x_1) \geq 2^{r_1}$

$$\text{size}(x_2) \geq 2^{r_2}$$



$$(1) r_1 = r_2$$

$$\text{size}'(x_2) = \text{size}(x_1) + \text{size}(x_2)$$

$$\geq 2^{r_1} + 2^{r_2} = 2 \cdot 2^{r_2} = 2^{r_2+1} = 2^{r_2}$$

$$(2) r_2 > r_1$$

$$\text{size}'(x_2) = \text{size}(x_1) + \text{size}(x_2)$$

$$\geq 2^{r_1} + 2^{r_2}$$

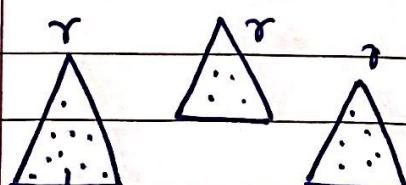
$$> 2^{r_2}$$

$$\geq 2^{r_2}$$

* If $\text{rank}(x) \leq \lfloor \log n \rfloor$

→ For any rank $r \exists$ almost $n/2^r$ nodes of rank r

* Once a node is marked it is not marked again.



nodes marked in subtree
if rank of subtree is r

* $\text{size}(x) \geq 2^r$

Let rank $r := \frac{n}{2^r}$ nodes

$$\therefore \text{size}(x) \geq \left(\frac{n}{2^r}\right)(\geq 2^r) > n$$

* Rank of a node is almost $\lceil \log n \rceil$

Let $r > \lceil \log n \rceil$

$$\text{then } \frac{n}{2^{r+1}} < 1 \quad (\#)$$

* \log^* :

$$\log^* n = \min \{ i \geq 0 : \log^{(i)} n \leq 1 \}$$

$\rightarrow \log^* n \leq 5$ virtually for any n of practical relevance

$$\log^*(2^{65536}) = \log^*(2^{2^{2^2}}) = 5$$

$2^{65536} \approx 20k$ digit number

We treat $\log^* n$ as $O(1)$

\rightarrow Rank of any node falls in the range $[0, \lceil \log n \rceil]$

Divide this range into following groups:

$$[1], [2], [3-4], [5-16], [17-2^{16}]$$

* rank: 0
belongs to group 0.

Each range is from $[k-2^{k-1}]$

\rightarrow let $G(v)$ be the group $\text{rank}(v)$ belongs to $G(v)$

$$= \log^* (\text{rank}(v))$$

Inkuu user block
Sir

* Given an allowance to a node when it becomes a non-root.

This allowance will be used to pay costs of path compression operations involving this node.

For a node whose rank $\in [k, 2^k - 1]$

allowance is 2^{k-1}

\rightarrow No. of nodes of rank $r \leq \frac{n}{2^r}$
 \therefore Total allowance handed out to nodes with ranks $[k - 2^{k-1}]$ is,

$$\rightarrow 2^{k-1} \left(\frac{n}{2^k} + \frac{n}{2^{k+1}} + \dots + \frac{n}{2^{2^{k-1}}} \right) \leq 2^{k-1} \cdot \frac{n}{2^{k-1}} = n = O(n)$$

\therefore Total no. of ranges is $\log^* n$

\therefore Total allowance ~~cost~~ granted to all nodes is $n \log^* n$

$$\rightarrow \text{Allowance for Block 0: } \frac{n}{2^1} + \frac{n}{2^0} = \frac{3n}{2} = O(n)$$

* We will spread this cost across all the n operations thus contributing $O(\log^* n)$ to each operation.

* Charge the cost of updating $\pi(p)$ to:

* Case 1:

→ $C_n(\pi(p)) \neq C_n(p)$ then charge it to current find operation (# Block/Group Leader)
Can apply $\log^* n$ times (# no. of Groups)
Adds only $\log^* n$ to cost of find

∴ $O(m \log^* n)$: for m findset operations

* Case 2: Charge it to p 's allowance

→ If $\pi(p)$ is updated rank of p 's parent ↑

→ Let p be involved in series of find's with q_i being parent after i^{th} find

$$\text{rank}(p) < \text{rank}(q_0) < \text{rank}(q_1) < \dots$$

→ Let m be the number of operations before p 's parent

has higher C_n -value than p ($C_n(p) = C_n(q_m) < C_n(q_{m+1})$)

$$C_n(p) = r = C_n(q_m), r \in [k, 2^{k-1}]$$

$$\therefore q_m \leq 2^{k-1}$$

∴ there is enough allowance for promotion of p

* After $m+1^{th}$ find, the find operation would pay for pointer updates as $C_n(\pi(p)) > C_n(p)$ from here on.