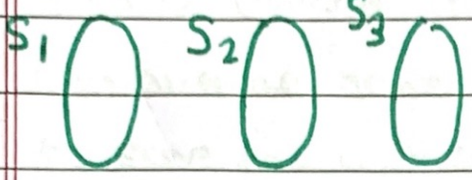
★ Disjoint Set Forest



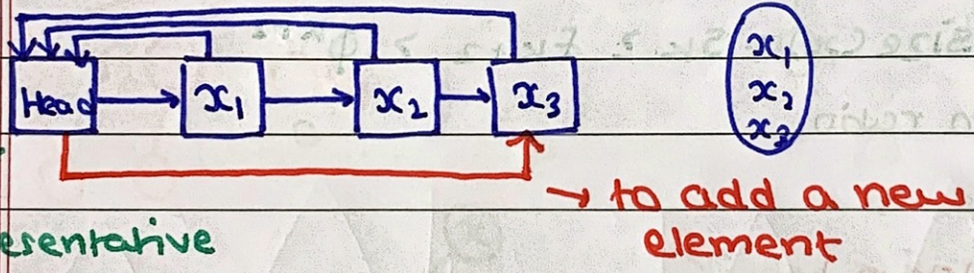
★ ADT:

→ makeset(x) ... O(1)
→ Findset(x) ... O(1)
→ union(x,y) ... O(k)

} n-operations
} m-operations

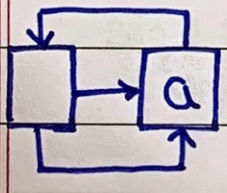
Naive : O(mk) – all union

$n \leq m$

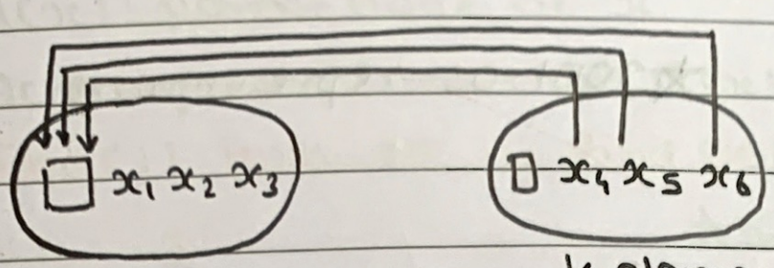


LINKED LIST

★ Makeset(a):

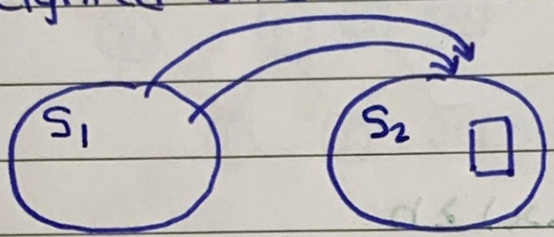


★ Union (x, y)



k elements
O(k)

→ Weighted Union Heuristic

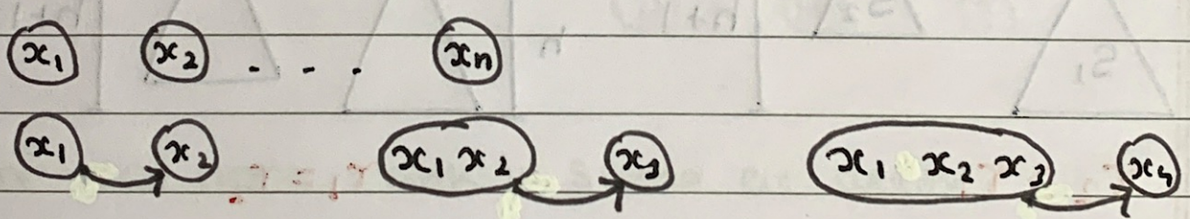


$|S_1| < |S_2|$

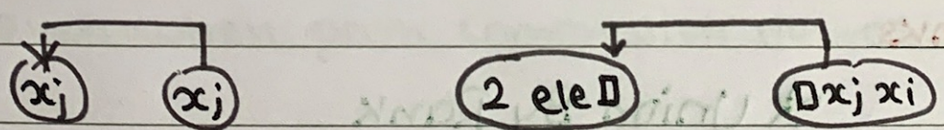
★ No. of pointer change : Bkr to merge S_1 to S_2

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

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



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



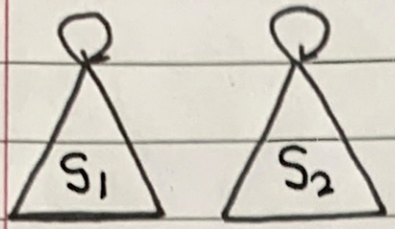
pointer of x_j changed at max log n times

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

↓

n choices for values of j

★ Set is represented as a tree

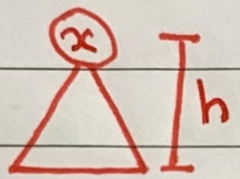


★ root as representative

★ Makeset : (a)

★ rank of a node:

Upper bound on height of subtree rooted at x.

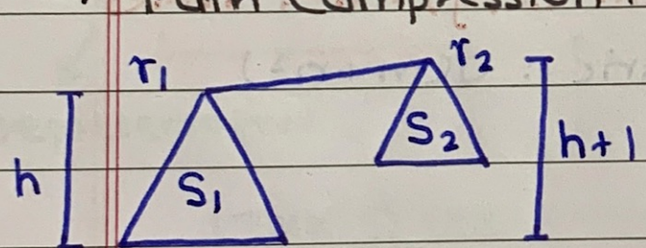


rank(x) ≥ h

★ Heuristics:

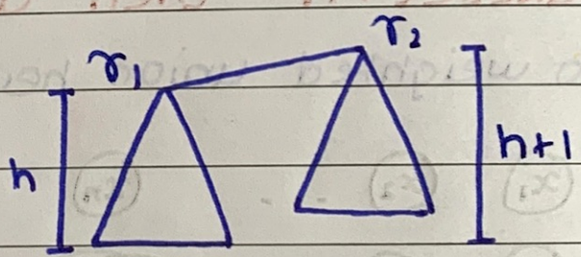
→ Union By Rank:

→ Path Compression:



$r_2 > r_1$

→ No need to change the ranks



$r_1 = r_2$ then $r_2 = r_2 + 1$

★ Union By Rank

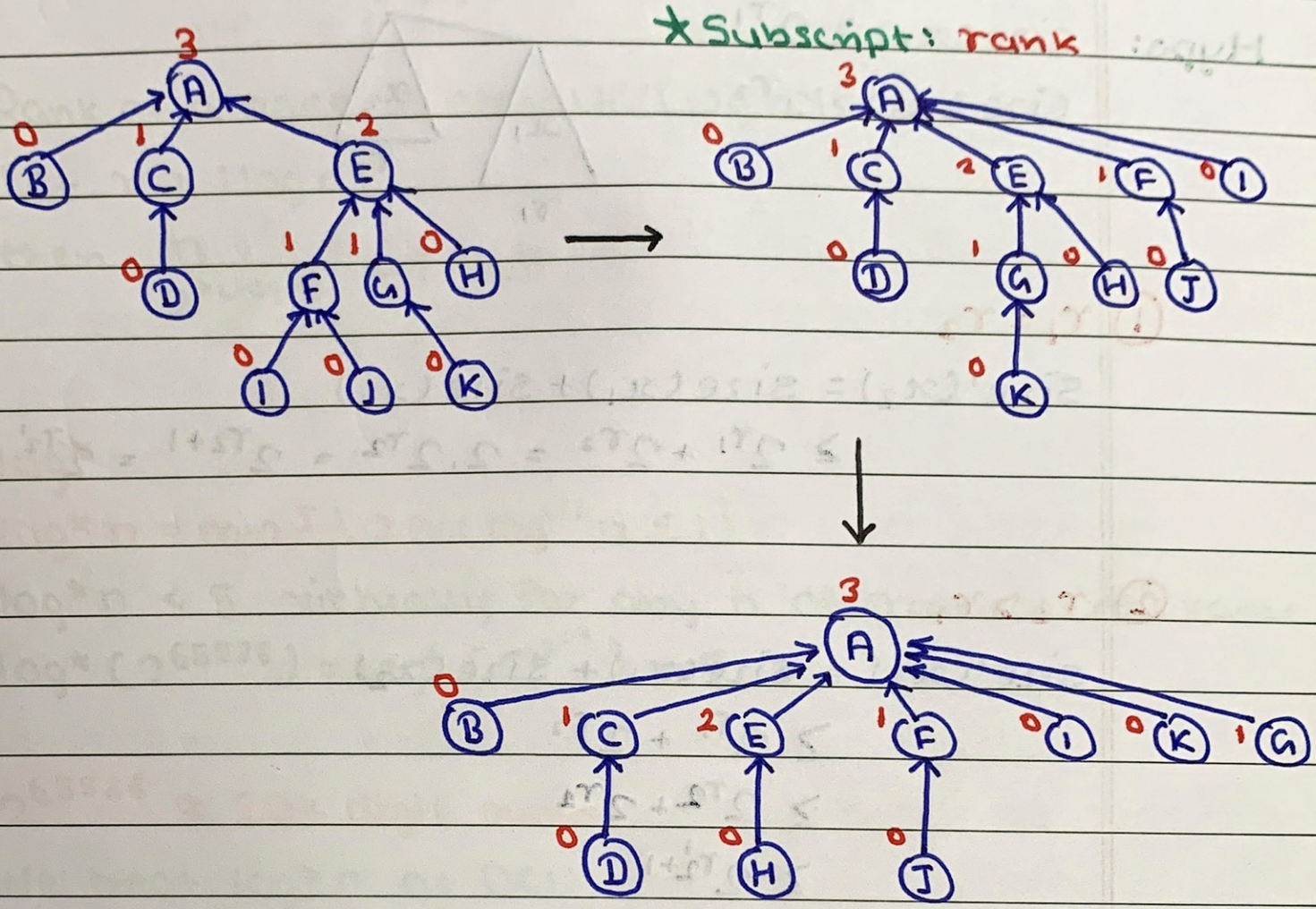
*Path Compression:

→ $\pi(x)$: parent node of x

→ Actually we promote $x, \pi(x), \pi(\pi(x)), \ldots$

Find(I) followed by Find(K)

*Subscript: rank



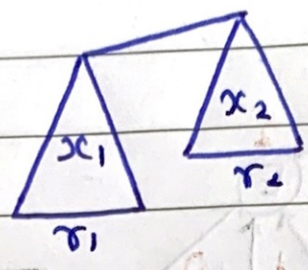
* For root node rank is same as depth
* Once a node becomes a non-root, its rank stays fixed
  → even when path compression decreases its depth

★ For any node x
$size(x) \geq 2^{rank(x)}$

Basis: (x) rank = 0 size(x) = 1

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

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



① $r_1 = r_2$

$size'(x_2) = size(x_1) + size(x_2)$

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

② $r_2 > r_1$

$size'(x_2) = size(x_1) + size(x_2)$

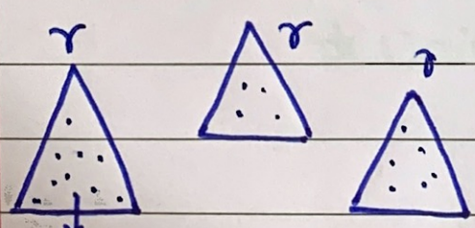
$\geq 2^{r_1} + 2^{r_2}$

$\geq 2^{r_2}$

$\geq 2^{r_2'}$

★ $\forall x \; rank(x) \leq \lfloor \log n \rfloor$

→ For any rank r ∃ atmost $n/2^r$ nodes of rank r



nodes marked in subtree if rank of subtree is r

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

* size(x) $\geq 2^r$

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

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

* Rank of a node is almost $\lfloor \log n \rfloor$

Let $r > \lfloor \log n \rfloor$

then $\frac{n}{2^{>\lfloor \log n \rfloor}} < 1$ (#)

* $\log^*$:

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

→ $\log^* n \leq 5$ virtually for any n of pratical relevance

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

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

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

→ Rank of any node falls in the range $[0, \lfloor \log n \rfloor]$

Divide this range into following groups:

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

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

* rank: 0 belongs to group 0.

→ Let G(v) be the group rank(v) belongs to G(v)

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

↓ Inkulu uses 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}

→ No. of nodes of rank $r \leq n/2^r$
∴ Total allowance handed out to nodes with ranks $[k - 2^{k-1}]$ is.

→ $2^{k-1}\left(\frac{n}{2^k} + \frac{n}{2^{k+1}} + \cdots + \frac{n}{2^{2^k-1}}\right) \leq 2^{k-1}\frac{n}{2^{k-1}} = n = O(n)$

∵ Total no. of ranges is $\log^* n$
∴ Total allowance ~~Credit~~ Cost granted to all nodes is $n \log^* n$

→ 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.

* Inverse Ackermann function: $\alpha(m,n)$

$$A(m,n) = \begin{cases} n+1 & m = 0 \\ A(m-1,1) & m > 0 \text{ \& } n = 0 \\ A(m-1, A(m,n-1)) & m > 0 \text{ \& } n > 0 \end{cases}$$

$$\alpha(m,n) = \min \{ i \geq 1 \mid A(i, \lfloor \tfrac{m}{n} \rfloor) \geq \log_2 n \}$$

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

* Case 1:

→ $G(\pi(p)) \neq G(p)$ 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

$rank(p) < rank(q_0) < rank(q_1) < \ldots$

→ Let m be the number of operations before p's parent has higher G-value than p $G(p) = G(q_m) < G(q_{m+1})$

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

∴ $q_m \leq 2^{k-1}$

∴ there is enough allowance for promotion of p

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