Union find using arrays cost $O(m + n \log n)$ for m FINDs and n UNIONs

which is optimal for

$m \geq n \log n$

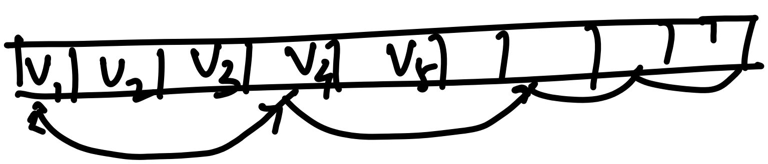
For sparse graphs can we do better?

A variation of storing the ~~Forests~~ (any subsets)

<u>Trees</u> to store subsets (not the Tree of MST)

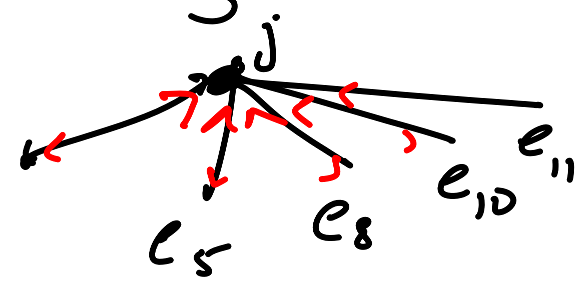
Arrays can be thought as depth one trees

depth 1 trees: stars



Tree S_j

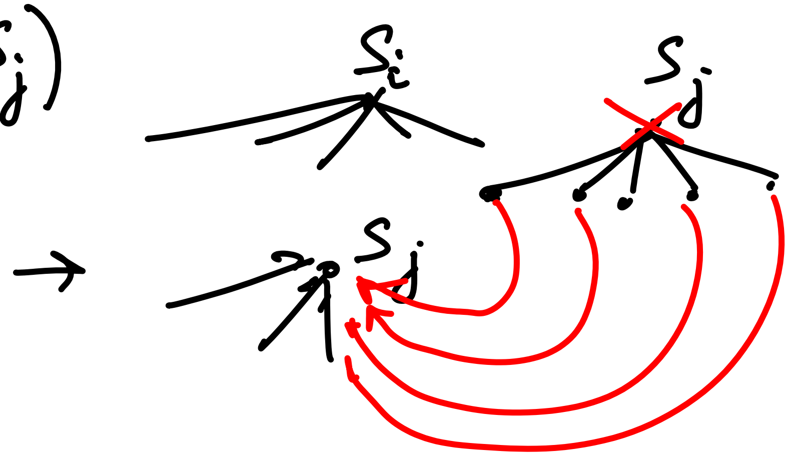
S_j



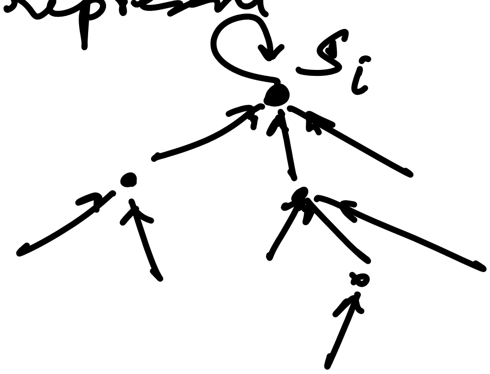
Find(e_5) : report the root (using the parent link)

$=$ parent(e_5) $O(1)$-time

UNION (S_i, S_j, S_j)



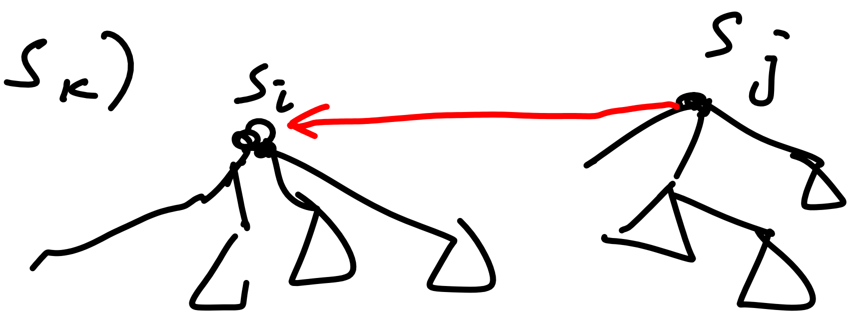
Instead of having depth 1 trees, suppose we allow arbitrary trees to represent the subsets, where each node is an element and they have pointers to their parents



FIND (x) : follow the parent pointers all the way to the root and report the label

Time: proportional to the height of the tree
??

UNION (S_i, S_j, S_k)



$O(1)$ -time

Make the Tree with smaller height point to the Tree with larger height. Height : "rank"

$\Rightarrow$ rank will increase by 1 when both trees have the identical ranks

<u>Claim</u> : By using union by rank heuristic $size(T) \geq 2^{rank(T)}$

$$\Rightarrow \quad rank(T) \leq \log[size(T)]$$

$$\Rightarrow \quad \leq \log n$$

$\Rightarrow$ m FINDS + n unions will cost

$$O(m \log n + n)$$

(instead of $O(m + n \log n)$)

<u>Proof of the claim</u> : (By induction)

rank = 0   1 node

Suppose true for all ranks $\leq k-1$

Union two trees with ranks $\leq k-1$,

say j, l, they have

at least 2^j and 2^l nodes resp.

(from I.H.)

Case : $l > j$: rank of the final tree $= l$

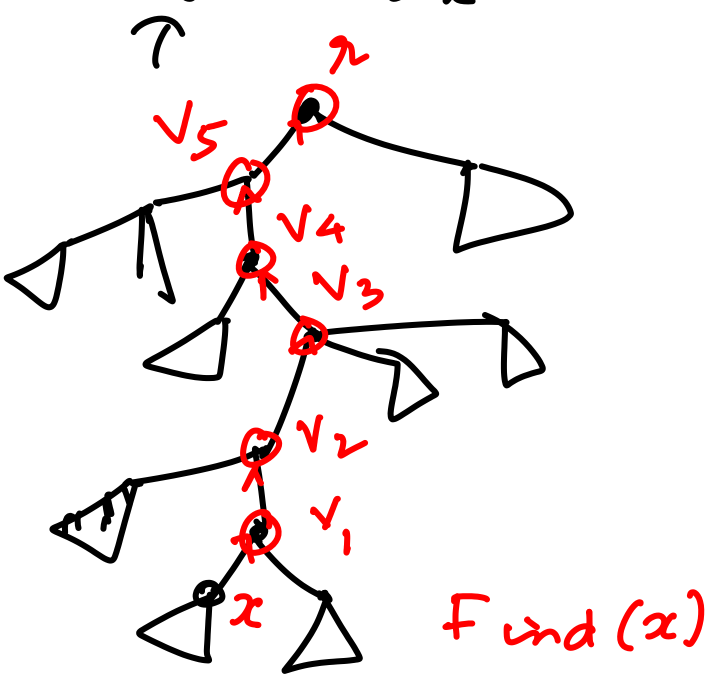
node $2^j + 2^l \geq 2^l$

Case $l = j$ rank becomes $l+1$

nodes $= 2^j + 2^l \geq 2 \cdot 2^l$

$= 2^{l+1}$

For improving the bound further, we will use path-compression heuristic



Make all the nodes visited during the Find (x) as children of the root

Use of path compression heuristic in conjunction with the rank heuristic results in $O((m+n)\log^* n)$ cost for m FINDs and n UNION

The function log "star"

$\log^* 1$ is 0

$$\log^* 2^x = \log^* x + 1$$

Ex.

$$\log^* 2 = \log^* 1 + \underline{1} = 1$$

$$\log^* 4 = \log^* 2^2 = 1+1 = 2$$

$$\log^*(2^4) = \log^*(16) = 2+1 = 3$$

$$\log^*\left(\underbrace{2^{16}}_{\approx 65K}\right) = 3+1 = 4$$

$$\log^*\left(2^{2^{16}}\right) = 4+1 = 5$$

$$\log^*\left(2^{2^{2^{16}}}\right) = 6$$

$$2^{2^{2^{2}}}\Big\}\, i \quad : \text{Tower of 2 function}$$

$\log^*$ is roughly i

Special case of Ackerman's function

inverse Ackerman function

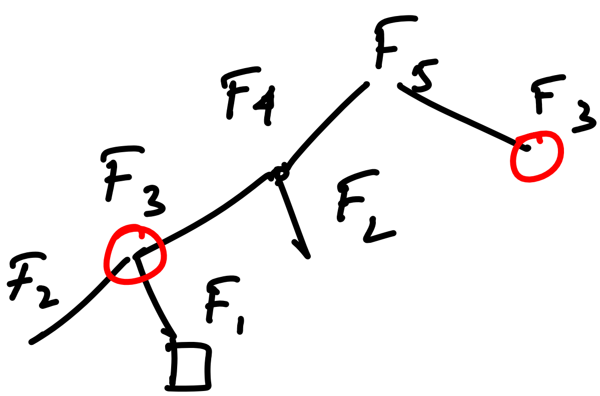
$\boxed{\text{What if we use path compression heuristic but not the union by rank heuristic?}}$

Dynamic Programming

1. We use a recurrence to capture the soln of a problem

$$F_n = F_{n-1} + F_{n-2}$$

$F_0 = 0 \quad F_1 = 1$



Recurrence for Divide and Conquer

Avoiding Repetitive calls to the same functions

$F_0 - F_1 \longrightarrow F_2 \rightarrow F_3 \longrightarrow F_4 \; . \; - \; - \; -$