Union find using arrays cost $O(m \cdot \log n)$ for $m$ FINDs and $n$ UNIONS which is optimal for $m \gg n \cdot \log n$.

For sparse graphs can we do better?

A variation of storing the forests (any subset)

Trees & store subsets (not the tree of MST)

Arrays can be thought as depth one trees

Depth 1 trees: stars

Find ($e_5$): report the root (using the parent link) = parent ($e_5$) $O(1)$-time
Union \((S_i, S_j, S_k)\) 

\[
\begin{array}{c}
\text{Si} \\
\downarrow \\
\text{Sj} \\
\end{array}
\]

Instead of having depth 2 trees, suppose we allow arbitrary trees to represent the subsets, where each node is an element and they have pointers to their parents.

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

Time: proportional to the height of the tree

\text{O}(1) - time

\text{Union}(S_i, S_j, S_k)
Make the Tree with smaller height point to the Tree with larger height. Height : “rank” by ↑

⇒ rank will increase when both trees have the identical ranks

Claim : By using union by rank heuristic size(τ) ⇒ rank(τ) ≥ 2

⇒ rank(τ) ≤ log [size(τ)]

⇒ ≤ log n

⇒ m finds + n unions will cost O ( m log n + n )

( instead of O ( m + n log n )

Proof of the claim : ( By induction )

rank = 0 I node

Suppose true for all ranks ≤ k-1
Union two trees with ranks \( \leq k-1 \), say \( j, k, \) they have at least \( 2^j \) and \( 2^k \) nodes resp. (from I.H.)

Case 1: \( l > j \) : rank of the final tree = \( l \)

\[ \text{node} \geq 2^j + 2^k \geq 2^l \]

Case 2: \( l = j \) : rank becomes \( l+1 \)

\[ \text{nodes} = 2^j + 2^l > 2 \cdot 2^l = 2^{l+1} \]

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

\[ \text{Find}(x) \]

Make all the nodes visited during the \text{Find}(x) as children of the root
Use of path compression heuristic rank handled results in \(O((m + n) \log^* n))\) cost for \(m\) \textsc{finds} and \(n\) \textsc{unions}.

The function \(\log^*\) "star"

\[
\log^* 1 = 0 \\
\log^* 2 = \log^* x + 1
\]

Ex.

\[
\log^* 2 = \log^* 1 + 1 = 1 \\
\log^* 4 = \log^* 2^2 = 1 + 1 = 2 \\
\log^* (2^4) = \log^* (16) = 2 + 1 = 3
\]
\[
\log^* (2^{16}) = 3 + 1 = 4
\]

\[
\log^* (2^{2^{16}}) \approx 65k
\]

\[
\log^* (2^{2^{2^{16}}}) = 5
\]

\[
2^{2^{2^{2^i}}} : \text{Tower of 2 functions}
\]

\[
\log^* \approx \text{roughly } i
\]

\[
\text{Special case of Ackermann's function}
\]

\[
\text{Inverse Ackermann function}
\]

What if we use path compression on

hemispheres but not the underlying
hemispheres?
Dynamic Programming

1. We use a recurrence to capture the solution 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 function

\[ F_0 - F_1 \rightarrow F_2 \rightarrow F_3 \rightarrow F_4 \ldots \]