Designing efficient implementation of Kruskal's algorithm

1. Assume elements are presented

2. Check if $T \cup \{e\}$ remains independent

   If so $T \leftarrow T \cup \{e\}$

   **Cycle test**

   Union the edge to the forest

   $e = (u,v)$

   If the next edge $e$ connects two -trees, then no cycle.

   else if it connects a tree and another forest

   If $u,v$ belong to different -trees, then add

   else discard
Find(v) : Return the label \( T \) s.t. \( v \in T \)

if \( \text{Find}(u) \neq \text{Find}(v) \) then
add

Union \((E = (u,v))\) : Since \( u \) and \( v \) belong to different trees, now we must join them into a single tree.

We need a suitable data structure that represents a forest, i.e., a set of trees and that supports Find and Union ops.

\[ T_1, T_2, T_3, T_4, \ldots, T_k \]
\[ V(T_1), V(T_2), V(T_3), \ldots, V(T_k) \]

Subsets of vertices: disjoint

1. Find returns the label of the tree
2. Union joins \( T_i, T_j \) to produce a new tree \( T_k \)
In Kruskal's algorithm, we will perform
1. \( \leq 2m \) `FIND` ops
2. \( n-1 \) `UNION` ops starting with \( n \) singleton sets

Total Running Time: \( O(m \cdot \text{cost of } \text{FIND}) + O(n \cdot \text{cost of } \text{UNION}) \)

Array for vertices: \( T_1, T_2, T_3, T_4 \)

\[
\begin{array}{ccccccc}
V_1 & V_2 & V_3 & V_4 & \cdots & V_n \\
\uparrow & \uparrow & \uparrow & \uparrow & \cdots & \uparrow \\
T_3 & T_4 & T_2 & T_1 & & T_1 \\
\end{array}
\]

label of the tree

: `FIND` takes \( O(1) \) time

: `UNION` \( T_3, T_4 \rightarrow T_1 \) We need to change the labels of the vertices that belong to \( T_3, T_4 \)
How many labels would change?
That is the cost of union.
Let us change the labels if only one of the heur, say $T_4 \leftarrow T_3 \cup T_4$
$T_3 \leftarrow T_3 \cup T_4$
Choose min $\{ |T_3|, |T_4| \}$
Size could be $O(n)$

?? How do we know smaller $T_3, T_4$

How do we locate the vertices of some tree

[Diagram showing set operations with vertices $V_1, V_2, V_3, V_4, V_5$, and sets $T_1$, $T_2$, $T_3$, $T_4$.]

Union($T_3, T_4$) should be maintained.
Cost of union is proportional to the size of the smaller tree.

Worst case calculations yield

\[ O(m + n \cdot n) = O(m + n^2) \cdot O(|E| + |V|^2) \]

If graph is dense \(|E| \approx |V|^2\)

What about sparse graphs?

Ideal performance \(O(|E| + |V|)\)

\[ n(v_i) = \# \text{time the label of } v_i \text{ changes} \]

Total cost of all unions \(\sum_i n(v_i) \leq O(\log n)\)

Question: How many times does the label of a vertex change?

Claim \(n(v_i) \leq O(\log n)\)
3) Total time for all unions
   \[ = O(n \log n) \]

2) Kyuva's algorithm takes
   \[ O(m + n \log n) \] using the
   data structure