Generic greedy algorithm is basically Kruskal’s MST in disguise

Borůvka’s algorithm

For every vertex, we find the closest vertex and add that edge to the MST.

- It is easy to parallelize
- There is linear time randomized algorithm for MST based on Borůvka’s algorithm
The time complexity of the greedy algorithm is dependent on:

1. How quickly we can detect independence
2. If so, actually update

In the context of MST we want to detect cycles due to the addition of an edge.

For any edge \((u, v)\) if \(u, v\) belong to different component, we can add:

\[
\text{Find}(u) \quad \text{Find}(v) \quad \text{else discard}
\]

\[
\text{Return the component} \neq \text{u/v}
\]

Process of adding \((u, v)\) is known as join component of \(u\) and \(v\).
Data Structure for Union Find

We have a family of (disjoint) subsets $\{S_i\}$; given any element $x$, we want to return $j$ such that $n$: total # elements $x \in S_j$.

$S_k \leftarrow S_i, U S_i_2$; Union

We can rename the union in the way that it suits us.

$S_i \leftarrow S_i, U S_i_2$

Approach 1

Based on maintaining an array of the label of each element.

$X_1, X_2, \ldots, X_i, \ldots, X_n$

$A$

$\text{Find } x_j := \text{return } A[j]\; \text{ O(1) time}$

Union
The elements of a set can be recovered by scanning the array: $O(n)$

\[ S_1, S_2, S_3, \ldots, S_k \]

\[ x_1, x_2, x_3, \ldots, x_s \]

Union $S_1, S_2$

Cost of union is the number of label changes $\leq$ size of the smaller set $\leq O(n)$ (worst case)

What is cost of $n$ unions and $m$ finds:

$O(n^2) + \frac{O(m)}{O(n \log n)}$
For any fixed element \( x \), what is the number of times its label can change?

By changing the labels of the elements of the smaller set, the number of changes is bounded by \( O(\log n) \) \( \Rightarrow \) \( O(n \log n) \) for all elements \( \Rightarrow \) cost of \( n \) unions is \( O(n \log n) \).

Tree representation of sets.

```
Si ----> Sj
```

Cost of `Find` : length of the path to root \( : O(n) \)

Cost of `Union` : \( O(1) \)
Let us link the "smaller" tree to the "larger" tree. (height is more relevant)

**rank of a tree**: height

**rank of a singleton node** is 0

Two trees \( T_1, T_2 \) with ranks \( r_1, r_2 \)

\( r_1 < r_2 \), if we make

\( T_1 \) the child of the root of \( T_2 \)

then \( \text{rank}(T_2) = r_2 \)

If \( r_1 = r_2 \), then \( \text{rank}(T_2) = r_2 + 1 \)

**Union-hemistic**: Make the tree with smaller rank - the child of the root of the other tree.

**Claim**: The # of nodes in a tree with rank \( r \geq 2^r \)

Then cost of find op is bound by \( O(\text{bfzn}) \)

\[ O(n + m \log n) \]
To rigorously analyze the benefits:

1. The rank function is preserved.
2. The rank of an internal node remains fixed henceforth.
3. Along any node in the root path, the ranks are monotonically increasing.

\[ r_{i_1} < r_{i_2} < r_{i_3} < \ldots < r_{i_k} = \text{root} \]
The path compression heuristic gives us an $O(n \log^* n + m \log^* n)$ bound on $m$ finds + $n$ unions.

$log^* n = i$ if $n \leq 2^{2^i}$

$2^{100} < 2^{2^{2^2}}$

$log^* (2^{100}) < 4$

$log^* (2^{2^{100}}) < 5$