Union-Find

Given n sets, each of size one:

\[ A_1 = \{1\}, \ A_2 = \{2\}, \ldots, \ A_n = \{n\} \]

Process a sequence of operations, where each operation is one of:

- Union \((A, B, C)\): Set \(C = A \cup B\); \(A = \emptyset\); \(B = \emptyset\)
- Find \((x)\): return the name of the set that contains \(x\).

The sequence consists of

\(n-1\) Union operations \(\text{ (in any order) }\)

\(m\) Find operations

We are interested in the total time to process any such sequence.
Store each set in a list:

* First node stores the name of the set.
* Each other node stores one element of the set.
* Each node $u$ stores two pointers:
  
  $\text{next}(u)$: next node in the list
  
  $\text{back}(u)$: first node in the list

$A = \{1, 4, 5, 7, 9\}$

Diagram:

```
head -> A -> 5 -> 9 -> 1 -> 4 -> 7
```

Tail
Start: for each set $A = \{x\}$:

- $A$  \[\rightarrow x \leftarrow \]

Total time $= O(n)$

Union $(A,B,C)$:

- $A$  \[\rightarrow a_1 \rightarrow a_2 \ldots \rightarrow a_k \leftarrow \]
- $B$  \[\rightarrow b_1 \rightarrow b_2 \ldots \rightarrow b_l \leftarrow \]

Combine the lists for $A$ and $B$, give each element in the $B$-list a pointer to the head of the new list, change the name in the head of the new list from $A$ to $C$:
Time = \( O(1 + l) = O(l) = O(\text{size of } B) \).

**Find \( x \):** follow the back pointer from the node storing \( x \) to the head of the list; return the name stored at the head.

Time = \( O(1) \).
Example:

<table>
<thead>
<tr>
<th>Union</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}, {2}</td>
<td>1</td>
</tr>
<tr>
<td>{3}, {1,2}</td>
<td>2</td>
</tr>
<tr>
<td>{4}, {1,2,3}</td>
<td>3</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>{n}, {1,\ldots,n-1}</td>
<td>n-1</td>
</tr>
</tbody>
</table>

\[ \Theta(n^2) \]

Better solution:

for each list: head stores \( \Rightarrow \) name of the set

size of the set
Find \( f(x) \) takes \( O(n) \) time, as before.

**Union \( (A,B,C) : \)**

*if \(|A| \geq |B| : \)*

\[
\begin{align*}
\text{C} & \quad \rightarrow \quad \text{list for A} \quad \rightarrow \quad \text{list for B} \\
|A|+|B| & \quad \rightarrow
\end{align*}
\]

*if \(|A| < |B| : \)*

\[
\begin{align*}
\text{C} & \quad \rightarrow \quad \text{list for B} \quad \rightarrow \quad \text{list for A} \\
|A|+|B| & \quad \rightarrow
\end{align*}
\]

\[
\text{Time} = O\left(\min(|A|, |B|)\right)
\]

\[
= O(\text{number of back-pointers that are changed}),
\]
What is the total time for a sequence of $n-1$ Union operations:

$$\text{total time} = \text{total number of back-pointer changes}$$

$$= \sum_{x=1}^{n} \text{total number of times back}(x) \text{ is changed}.$$

Consider an element $x$. How many times do we change back$(x)$:

Start: $x$ is in a set of size 1.

First time back$(x)$ is changed:

- the set containing $x$ is merged with a set of size $\geq 1$.
- new set containing $x$ has size $\geq 2$.

Second time back$(x)$ is changed:

- the set containing $x$ is merged with a set of size $\geq 2$.
- new set containing $x$ has size $\geq 4$. 

Third time back$(x)$ is changed:
the set containing $x$ is merged with a set of size $\geq 4$
° new set containing $x$ has size $\geq 8$.

etc., etc.

Since there are $n$ elements: back$(x)$ is changed $\leq \log n$ times.

° total time for $n-1$ Union operations

\[ = O(n \log n). \]

Conclusion: Any sequence of $n-1$ Union and $m$ Find operations:

\[ O(m + n \log n) \] time.
Minimum Spanning Trees

\[ G = (V, E), \] undirected, connected graph, each edge \( \{u, v\} \) in \( E \) has weight \( w(u, v) \).

Compute a subgraph \( G' \) such that

* the vertex set of \( G' \) is \( V \),
* \( G' \) is connected, and
* \( \text{weight}(G') = \text{sum of the weights of the edges in } G' \) is minimum.

Claim: \( G' \) must be a tree (i.e., connected and no cycles).

\( G' \) is called a minimum spanning tree of \( G \).
Lemma: Split V into A and B.

\{u,v\} : shortest edge connecting A and B.

Then: there is an MST of G that contains the edge \{u,v\}.
Proof: Let $T$ be an MST of $G$.

If $\{u,v\}$ is an edge in $T$: done.

Assume $\{u,v\}$ is not an edge in $T$.

$T$ is connected :: there is a path in $T$ between $u$ and $v$. This path contains an edge $\{a,b\}$ with $a \in A$ and $b \in B$.

\[ \begin{array}{c}
\text{A} \\
\text{ } \\
\text{ } \\
\{a\} \\
\text{ } \\
\{b\} \\
\text{B}
\end{array} \]

\[ u \quad \text{---} \quad v \]

Define $T' = T$ minus $\{a,b\}$ plus $\{u,v\}$

Observe: $T'$ is a tree.
\[
\text{weight}(T) \leq \text{weight}(T') \quad \text{if } T \text{ is MST}
\]

\[
= \text{weight}(T) - \text{wt}(a,b) + \underbrace{\text{wt}(u,v)}_{\leq \text{wt}(a,b)} \\
\leq \text{weight}(T).
\]

\[
\therefore \text{weight}(T') = \text{weight}(T)
\]

\[
\therefore T' \text{ is an MST that contains the edge } \{u,v\}.
\]
Kruskal (1956):

**Approach:** Maintain a forest. In each step, add an edge of minimum weight that does not create a cycle.

**Start:** forest = one tree for each vertex

**One iteration:**

Combine 2 trees using an edge of minimum weight.
Sort edges by weight:

BC, CD, BD, CF, DF, EF, AD, AB, CE, AC

Illustrate the algorithm.
Kruskal: $G = (V, E)$, $V = \{x_1, \ldots, x_n\}$, $m = 1E1$

Sort the edges of $E$ by weight: $e_1, e_2, \ldots, e_m$.

for $i = 1$ to $n$:
  $V_i = \{x_i\}$, $T_i = \emptyset$;

for $k = 1$ to $m$:
  let $u_k$ and $v_k$ be the vertices of $e_k$;
  $i = \text{index such that } u_k \in V_i$;
  $j = \text{index such that } v_k \in V_j$;
  if $i \neq j$:
    $V_i = V_i \cup V_j$; $V_j = \emptyset$;
    $T_i = T_i \cup T_j \cup \{\{u_k, v_k\}\}$; $T_j = \emptyset$.

Running time:

Sort: $O(m \log m) = O(m \log n)$, because $m \leq (n \choose 2)$.

First for-loop: $O(n)$. 
Second for-loop:
store each $T_i$ as a linked list: $O(n)$ total time to maintain these lists.

how to store the sets $V_i$: Union-Find structure.

\[
\begin{align*}
\text{Find: } & 2m \text{ times } \\
\text{Union: } & n-1 \text{ times }
\end{align*}
\]
\[
\{ = O(m + n \log n)
\]

Total time: $O(m \log n) + O(n) + O(m + n \log n)$

\[
= O(m \log n)
\]
\[
\Rightarrow m \geq n-1 \text{ because G is connected.}
\]

Conclusion: Kruskal's algorithm computes MST in $O(m \log n)$ time.