OT TODAY: 1:15 - 2:15 pm.

Union-Find Data Structure

Motivation: Kruskal’s algorithm. When we process an edge \((u,v)\), we want to know if \(u\) and \(v\) are in different connected components or not.

We need a data structure that maintains disjoint sets where each set represents one CC in our (partial) solution. Our data structure will support the following operations:

- \text{makeSet}(x) : Create a singleton set
containing x

Find (x) : returns the name of the set containing x

Union (x, y) : merges sets containing x & y.

Kruskal

\[
\begin{align*}
\text{for each } u \in V \text{ do } & \quad \begin{cases}
\text{make-set } (u) \\
\end{cases} \\
\mathcal{O}(n) \\
\end{align*}
\]

\[
\begin{align*}
\text{Sort edges } \mathcal{E} \text{ in order of weight } & \quad \begin{cases}
\mathcal{O}(m \lg m) \quad & \mathcal{O}(m \lg n^2) = \mathcal{O}(m \lg m) \\
\end{cases} \\
\text{for each edge } (u, v) \text{ do } & \quad \begin{cases}
\text{if } \text{Find } (u) \neq \text{Find } (v) \text{ then } \\
\text{Call find(\text{union}(u, v))} \\
\end{cases} \\
\mathcal{O}(m) \\
\end{align*}
\]
We will represent a set using "directed trees", in which all elements in the same set belong to the same tree. The root of the tree will be the representative.

Make set \( \pi(x) \)

\[
\pi(x) \leftarrow x
\]

\[
\text{rank}(x) \leftarrow 0
\]

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

\[
\text{while } x \neq \pi(x) \text{ do}
\]

\[
\text{x} \leftarrow \pi(x)
\]
\[
\begin{align*}
&\{ \\
&\quad x \leftarrow \pi(x) \\
&\quad \text{return } \pi(x) \\
&\} \\
&\text{Find Cost } \alpha \\
&\text{hit } f \text{ the tree proportional.} \\
\end{align*}
\]

Union \( (x, y) \)

\[
\begin{align*}
&\quad r_x \leftarrow \text{Find}(x) \\
&\quad r_y \leftarrow \text{Find}(y) \\
&\quad \text{if } r_x = r_y \text{ then return.} \\
&\quad \text{else if } \text{rank}(r_x) > \text{rank}(r_y) \text{ then} \\
&\quad \quad \pi(r_y) \leftarrow r_x \\
&\quad \text{else} \\
&\quad \quad \pi(r_x) \leftarrow r_y \\
&\quad \text{if } \text{rank}(r_x) = \text{rank}(r_y) \text{ then}
\end{align*}
\]
Union \((a, b)\) → 
\[\text{rank}(b) = 1\]
\[\text{rank}(a) = 0\]

Find \((a)\) → \(b\)

Union \((a, c)\) → 

Union \((a, e)\) → 

\[\vdots\]

\[\text{rank}(d) + 1\]

\[\text{rank}(y) + 1\]
Properties.

(1) For any non-root node \( x \),
\[
\text{rank}(\pi(x)) > \text{rank}(x) .
\]

(2) For any (root) node with rank
exactly equal to \( k \), the
no. of nodes in the tree rooted
at the node \( \geq 2^k \).

Proof idea: Consider the moment
when rank of node \( x \) became \( k \).
(3) \[ \text{# nodes of rank exactly } k \leq \frac{n}{2^k}. \]
\# nodes of rank exactly \( y_n \)

\[
\leq \frac{\lambda}{\frac{1}{2} \ln n} = 1
\]

\# nodes of rank \( > y_n \) = 0.

\[{}\Rightarrow\text{max rank \# any node is} \leq y_n.\]

Running time of Kruskal: \( O(m \ln n) \).

Suppose sorted edges were given to us for free. In this case, can we implement the operations more efficiently?

\( O(m) \) find operations.
$O(\log n)$ time per find. Can we do better?

Find($x$)

if $x \neq \pi(x)$ then

$\pi(x) \leftarrow$ Find($\pi(x)$)

return $x$

Analysis (Union by rank using path compression).

Amortized analysis to find the
Cost of an find by considering a seq. of find & insert on an empty data structure.

\( \log^* n \) : \# times we apply \( \log \) to bring down the value of \( n \) down to 1 (or less).

**Ex:** \( \log^* 1024 \)

\[
\begin{array}{c}
2 \\
1024 \\
\log_2 1024 \rightarrow 10 \rightarrow 4 \rightarrow 2 \rightarrow 1
\end{array}
\]

We will show that the amortized
Cost of finding \( L_n^* \).

1. For any non-root node \( x \),
   \[ \text{rank}(\pi(x)) > \text{rank}(x). \]

2. For any root node with rank exactly equal to \( k \), the
   no. of nodes in the tree rooted at the node \( \geq 2^k \).

3. \# nodes of rank exactly \( k \) \( \leq \frac{n}{2^k}. \)
Note that $\text{rank}(x) \neq \text{ht}(x)$.

Map $\text{rank}$ of any node $\leq \lg n$. 

A node is "frozen" once a node ceases to be a root. 

Partition the rank $k$ of all nodes as follows:

\[
\{1\}, \{2\}, \{2^1\}, \{2^1, 2\}, \{5, 6, \ldots, 2^2\},
\]

\[
\{17, 18, \ldots, 2^3\}, \{2, 3\}, \{16\}, \{2^1, \ldots, 2^4\}, \{2^2\}, \ldots,
\]

\[
\# \text{rank intervals} \leq \lg n.
\]
# Find operator = $O(mn)$.  
Who pays for all the find operators?

**Budget**: $O(my^*n) + 

**Pocket money**: $O(nly^*n)$.

$1 = \text{one operation or one hop in find.}$

Suppose budget + pocket money pays for all the find operators then do we agree that total cost = $O(my^*n)$. 

Consider an interval
\[ \{ k+1, k+2, \ldots, 2^k \} \]

If a node has a rank belonging to the above interval then it gets pocket money \( \geq 2^k \).

Total # nodes whose ranks belong to the above interval are

\[
\leq \frac{N}{2^k} + \frac{N}{2^{k+2}} + \ldots
\]

\[
= \frac{N}{2^k} \left( \frac{1}{2} + \frac{1}{2^2} + \ldots \right)
\]
\[ \leq \left\lfloor \frac{\sqrt{1}}{2^k} \right\rfloor \]

Total pocket money given to nodes whose ranks are in the interval \( \leq \frac{1}{2^k} \).

\[ 2^k = \lceil \sqrt{n} \rceil \]

Total # intervals = \( \lceil \sqrt{n} \rceil \)

\[ \implies \text{Total pocket money} \leq \pi^* n \]

Big jump:

Small jump:

\[ \pi(n) \text{ has rank in a higher interval.} \]
All hops are big jumps.

\[ \# \text{ big jumps in one find} \leq \lg n. \]

Total \# finds = \( O(n) \)

\[ \implies \text{Budget takes can fit all big jumps}. \]
$u \xrightarrow{\pi(u)} \pi(u) \xrightarrow{1} \{k+1, \ldots, 2^k\} \xrightarrow{2^k} 2^n \leq 2^n$