Union Find

A motivating problem: find all connected components in a graph $n = \# \text{vertices}

m = \# \text{edges}$

Depth first search in $O(n + m)$ time

Dynamic Graph Connectivity - maintain connected components as the graph changes

Answer queries: given vertices $u, v$, are they connected?

Examples

- Social networks, as relationships added/deleted
- Min. Spanning tree
  
  Recall Kruskal's alg:

  Order edges by weight $e_1 \ldots e_m$

  $T = \emptyset$

  for $i = 1 \ldots m$

  if $e_i$ joins different components of $T$

  $T \leftarrow T \cup \{e_i\}$ (and join the components)

  \[\text{joining components} \]

This is a special case of

Incremental Dynamic Connectivity

- Add edges but don't delete any

Today: solutions to using Union Find
Union Find Data Structure

- maintain a collection of disjoint sets subject to
  - Union (A, B) — unite two sets A and B (destroy A, B)
  - Find (e) — which set contains element e

Analysis of Kruskal’s Alg. using Union-Find

\[
\text{sort + } 2m \text{ finds + } n \text{ unions} \\
\text{want this work } \leq O(m \log n) \\
= O(m \log n)
\]

Note: In this analysis, we care about sequence of Union, Find, so amortized analysis is relevant.

Implementation of Union Find

\[
n = \# \text{ elements} \\
m = \# \text{ operations}
\]

First approach

- Use array \( S[i..n] \) \( S[i] = \text{name of set containing i} \)
- Find \( O(1) \)
- Union \( O(n) \) worst case

Tiny improvement:

- Union (A, B) — update \( S[i] \) for \( i \) in smaller of A, B

e.g.

\[
\begin{align*}
\text{A} & : 1, 3 \\
\text{B} & : 2, 7, 6, 5 \\
\text{C} & : 4
\end{align*}
\]

\[
S = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
A & B & A & C & B & B & B
\end{bmatrix}
\]

For Union (A, B), update for A

- Cost of all unions \( \leq O(n \log n) \) because each element changes its set \( \leq \log n \) times
- Cost of \( m \) operations \( \bigO(m + n \log n) \)
- This is best possible if \# finds is \( \Omega(n \log n) \)
for Kruskal we get
\[
O(m \log n) + 2m \text{ Finds } + n \text{ Unions} = O(n \log n)
\]

A better method in case \# finds is small
for Union Find, not Kruskal.

Represent each set as a tree of parent pointers

```
A 3 1 6
B 2 7
C 4 5
```

\text{Find}(e) - walk up the tree from } e \text{ & get set name at root}

\text{Union} - link the 2 trees - make "smaller" tree's
root point to root of other tree

\underline{Path compression}: After \text{Find}(e), for every node } x \text{ on path } v \rightarrow \text{root, set } \text{parent}(x) \leftarrow \text{root}

```
This doubles work of \text{Find},
but same } O(*)
```

Last detail: how to define "smaller"

\text{Want}
\begin{itemize}
  \item bigger height
  \item smaller height
\end{itemize}

\text{But we cannot easily keep track of height}
(p\text{ath compression changes it})
Idea: be lazy, use "rank" $r$

$r(\text{single node}) = 0$

\[ r = \max \{ r_1, r_2 + 1 \} \]

rank = height if no-path compressions are done.

Exercise A node of rank $r$ has $\geq 2^r$ descendants. (*)

Algorithm (above) is simple. But analysis is hard.

Theorem [Tarjan 1975]
The cost of $m$ operations is $\Theta(m \cdot \alpha(m,n))$

i.e. amortized cost per operation is $\Theta(\alpha(m,n))$

$\alpha(m,n)$ — Inverse Ackermann function
  — very slow growing, $\leq 5$ for all practical purposes.

This bound is tight — there is an infinite class of examples where the alg. takes this time.

We will prove a slightly weaker bound of $O(m \log^* n)$ using a charging scheme.

Notes: — for $\Theta(m \cdot \alpha(m,n))$ bound see CLRS
  — for alternate pf. of $O(m \log^* n)$ see Jeff Erickson's notes.
\[
\log^* n = \min \left\{ i : \log \left( \log \cdots \left( \log n \right) \right) \leq 1 \right\}
\]

i.e. define \(2^i n = \frac{2^{2^i}}{n^{2^i}}\)

Then \(\log^* (2^i n) = n\)

<table>
<thead>
<tr>
<th>(n)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\log^* n)</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Cost of \(\text{Find}(v)\) = Distance from \(v\) to root

Idea: change some of this cost to the \(\text{Find}\) and some to the nodes along the path. Then sum up.

\begin{itemize}
  \item \textbf{Claim 1} \(\text{rank}(v) < \text{rank}(\text{parent}(v))\)
  \item \textbf{Claim 2} \# vertices of rank \(r\) \(\leq \frac{n}{2^r}\)
\end{itemize}

Repeat use on previous page

When vertex is assigned rank \(r\), it has \(\geq 2^r\) descendants. Vertices of rank \(r\) have disjoint descendants.

Divide vertices into groups based on rank

- vertex of rank \(r\) \(\rightarrow\) group \(\log^* r\)
- \# groups = \(\log^* n\)
- group \(g\) contains ranks \(2^g + (g-1) + 1 \cdots 2^g\)

\(\leq 2^g\) ranks
Charge for Find \( (v) \):

- for each vertex \( u \) on path \( v \rightarrow \text{root} \)
- if \( u \) has parent \& grandparent and \( \text{group}(u) = \text{group}(\text{parent}(u)) \)
  then charge 1 to \( u \)
- else charge 1 to Find \( (u) \)

Note that this covers cost.

Total charges to Find \( (v) \) \( \leq \log^* n + 1 \)

since group changes \( \leq \log^* n - 1 \) times

+ 2 for root and child of root.

Now add up the charges to vertices:

Total charge to vertex \( u \) in group \( g \):

- if \( u \) is charged then path compression will give it a new parent of higher rank than the old parent by claim 1

- so \( u \) in group \( g \) is charged

\[ c(g) = \left( \text{# ranks in group } g \right) - 1 \]

times before it acquires a parent in a higher group — and after that it is not charged.

\[ c(g) \leq 2^g \text{ from above} \]

Total charge to all vertices in group \( g \):

\[ c(g) \cdot N(g) \quad \text{where } N(g) = \text{# vertices in group } g \]

\[ N(g) \leq \sum_{r = 2^g g + 1}^{\infty} \frac{n}{2^r} \leq \frac{n}{2^{2^g g + 1}} \leq \sum_{0}^{\infty} \frac{1}{2^i} = \frac{n}{2^g} \]

\[ \text{# vertices of rank } r \]

Thus \( c(g) \cdot N(g) \leq n \)
Total charge to all vertices
\[ n \cdot \log^* n \]
charge to \( \text{group} \) # groups

Total charge to finds and vertices
\[ O\left( \frac{m (\log^* n + 1)}{\text{# finds}} \right) + n \log^* n = O(m \log^* n) \]
\[ \text{for find} \]
\[ \text{vertices} \]

note standard assumption that \( m \geq n \)