Union Find

- Union Find
- Quick Find
- Quick Union
- Weighted Quick Union
- Path Compression
- Dynamic Connectivity
Union Find

- Union Find
- Quick Find
- Quick Union
- Weighted Quick Union
- Path Compression
- Dynamic Connectivity
Union Find

- **Union find.** Maintain a dynamic family of sets supporting the following operations:
  - **INIT(n):** construct sets \{0\}, \{1\}, \ldots, \{n-1\}
  - **UNION(i, j):** forms the union of the two sets that contain i and j. If i and j are in the same set nothing happens.
  - **FIND(i):** return a representative for the set that contains i.

\[
\text{INIT}(9) \\
\{0\} \{1\} \{2\} \{3\} \{4\} \{5\} \{6\} \{7\} \{8\}
\]

\[
\text{UNION}(5, 0) \\
\{1, 0, 6\} \{8, 3, 2, 7\} \{4, 5\} \rightarrow \{1, 0, 6, 4, 5\} \{8, 3, 2, 7\}
\]
Union Find

• **Applications.**
  • Dynamic connectivity.
  • Minimum spanning tree.
  • Unification in logic and compilers.
  • Nearest common ancestors in trees.
  • Hoshen-Kopelman algorithm in physics.
  • Games (Hex and Go)
  • Illustration of clever techniques in data structure design.
Union Find

- Union Find
- Quick Find
- Quick Union
- Weighted Quick Union
- Path Compression
- Dynamic Connectivity
Quick Find

• **Quick find.** Maintain array id[0..n-1] such that id[i] = representative for i.
  
  • **INIT(n):** set elements to be their own representative.
  
  • **UNION(i,j):** if FIND(i) ≠ FIND(j), update representative for all elements in one of the sets.
  
  • **FIND(i):** return representative.

```
INIT(9)

{0} {1} {2} {3} {4} {5} {6} {7} {8}

id[]

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8
```

```
UNION(5,0)

{1, 0, 6} {8, 3, 2, 7} {4, 5}

id[]

0 1 2 3 4 5 6 7 8

1 1 3 3 5 5 1 3 3
```

```
{1, 0, 6, 4, 5} {8, 3, 2, 7}

id[]

0 1 2 3 4 5 6 7 8

1 1 3 3 1 1 1 3 3
```
Quick Find

**INIT(n):**
for k = 0 to n-1
  id[k] = k

**FIND(i):**
return id[i]

**UNION(i,j):**
iID = FIND(i)
jID = FIND(j)
if (iID ≠ jID)
  for k = 0 to n-1
    if (id[k] == iID)
      id[k] = jID

**Time.**
- O(n) time for INIT, O(n) time for UNION, and O(1) tid for FIND.
Union Find

- Union Find
- Quick Find
- Quick Union
- Weighted Quick Union
- Path Compression
- Dynamic Connectivity
Quick Union

- **Quick union.** Maintain each sets as a rooted tree.
- Store trees as array $p[0..n-1]$ such that $p[i]$ is the parent of $i$ and $p[root] = root$. Representative is the root of tree.
  - $\text{INIT}(n)$: create $n$ trees with one element each.
  - $\text{UNION}(i,j)$: if $\text{FIND}(i) \neq \text{FIND}(j)$, make the root of one tree the child of the root of the other tree.
  - $\text{FIND}(i)$: follow path to root and return root.
Quick Union

- **INIT(n):** create n trees with one element each.
- **UNION(i,j):** if $\text{FIND}(i) \neq \text{FIND}(j)$, make the root of one tree the child of the root of the other tree.
- **FIND(i):** follow path to root and return root.

**Exercise.** Show data structure after each operation in the following sequence.
- INIT(7), UNION(0,1), UNION(2,3), UNION(5,1), UNION(5,0), UNION(0,3), UNION(5,2), UNION(4,3), UNION(4,6).
Quick Union

\textbf{INIT(n)}:
\begin{align*}
\text{for } k = 0 \text{ to } n-1 \\
p[k] = k
\end{align*}

\textbf{FIND(i)}:
\begin{align*}
\text{while } (i \neq p[i]) \\
i = p[i] \\
\text{return } i
\end{align*}

\textbf{UNION(i,j)}:
\begin{align*}
\text{r}_i &= \text{FIND}(i) \\
\text{r}_j &= \text{FIND}(j) \\
\text{if } (\text{r}_i \neq \text{r}_j) \\
\text{p[r}_i] &= \text{r}_j
\end{align*}

- Time.
- \(O(n)\) time for \text{INIT}, \(O(d)\) tid for \text{UNION} and \text{FIND}, where \(d\) is the depth of the tree.
Quick Union

- **UNION** and **FIND** depend on the depth of the tree.
- **Bad news.** Depth can be n-1.
- **Challenge.** Can combine trees to limit the depth?
Union Find

- Union Find
- Quick Find
- Quick Union
- Weighted Quick Union
- Path Compression
- Dynamic Connectivity
Weighted Quick Union

- **Weighted quick union.** Extension of quick union.
- Maintain extra array sz[0..n-1] such sz[i] = the size of the subtree rooted at i.
  - **INIT:** as before + initialize sz[0..n-1].
  - **FIND:** as before.
  - **UNION(i,j):** if FIND(i) ≠ FIND(j), make the root of the smaller tree the child of the root of the larger tree.
- **Intuition.** UNION balances the trees.

\[
\text{sz}[r_j] = \text{sz}[r_i] + \text{sz}[r_j]
\]
Weighted Quick Union

\[ \text{UNION}(i, j): \]
\[
\begin{align*}
    r_i & = \text{FIND}(i) \\
    r_j & = \text{FIND}(j) \\
    \text{if } (r_i \neq r_j) \\
    \quad & \text{if } (\text{sz}[r_i] < \text{sz}[r_j]) \\
    \quad & \quad \text{p}[r_i] = r_j \\
    \quad & \quad \text{sz}[r_j] = \text{sz}[r_i] + \text{sz}[r_j] \\
    \quad & \text{else} \\
    \quad & \quad \text{p}[r_j] = r_i \\
    \quad & \quad \text{sz}[r_i] = \text{sz}[r_i] + \text{sz}[r_j]
\end{align*}
\]
Lemma. With weighted quick union the depth of a node is at most $\log_2 n$.

Proof.

- Consider node $i$ with depth $d_i$.
  - Initially $d_i = 0$.
  - $d_i$ increases with 1 when the tree is combined with a larger tree.
  - The combined tree is at least \textit{twice} the size.
  - We can double the size of trees at most $\log_2 n$ times.
  - $\implies d_i \leq \log_2 n$. 

Weighted Quick Union
## Union Find

<table>
<thead>
<tr>
<th>Data structure</th>
<th>UNION</th>
<th>FIND</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick find</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>quick union</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>weighted quick union</td>
<td>O(log n)</td>
<td>O(log n)</td>
</tr>
</tbody>
</table>

- **Challenge.** Can we do even better?
Union Find

- Union Find
- Quick Find
- Quick Union
- Weighted Quick Union
- Path Compression
- Dynamic Connectivity
Path Compression

- **Path compression.** Compress path on FIND. Make all nodes on the path children of the root.
- No change in running time for a single FIND. Subsequent FIND become faster.
- Works with both quick union and weighted quick union.
Path Compression

- **Theorem [Tarjan 1975].** With path compression any sequence of $m$ \textsc{find} and \textsc{union} operations on $n$ elements take $O(n + m \alpha(m,n))$ time.
- $\alpha(m,n)$ is the inverse of Ackermann's function. $\alpha(m,n) \leq 5$ for any practical input.
- **Theorem [Fredman-Saks 1985].** It is not possible to support $m$ \textsc{find} and \textsc{union} operations $O(n + m)$ time.
Union Find

- Union Find
- Quick Find
- Quick Union
- Weighted Quick Union
- Path Compression
- Dynamic Connectivity
Dynamic Connectivity

- **Dynamic connectivity.** Maintain a dynamic graph supporting the following operations:
  - `INIT(n)`: create a graph $G$ with $n$ vertices and no edges.
  - `CONNECTED(u,v)`: determine if $u$ and $v$ are connected.
  - `INSERT(u, v)`: add edge $(u,v)$. We assume $(u,v)$ does not already exist.

![Graph Diagram]

1. BEFORE `INSERT(3,4)`:
   - Vertices: 1, 2, 3, 4, 5, 6, 7, 8, 9
   - Edges: (1,3), (3,7), (1,6), (2,7), (2,8), (3,9)

2. AFTER `INSERT(3,4)`:
   - Vertices: 1, 2, 3, 4, 5, 6, 7, 8, 9
   - Edges: (1,3), (3,4), (4,5), (3,7), (1,6), (2,7), (2,8), (3,9)
Dynamic Connectivity

- Implementation with union find.
  - \textsc{Init}(n): initialize a union find data structure with \(n\) elements.
  - \textsc{Connected}(u,v): FIND(u) == FIND(v).
  - \textsc{Insert}(u, v): \textsc{Union}(u,v)

- Time
  - \(O(n)\) time for \textsc{Init}, \(O(\log n)\) time for \textsc{Connected}, and \(O(\log n)\) time for \textsc{Insert}
Union Find

- Union Find
- Quick Find
- Quick Union
- Weighted Quick Union
- Path Compression
- Dynamic Connectivity