Union Find

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

Philip Bille

Union find. Maintain a dynamic family of sets supporting the following operations:

- **Init**(n): construct sets {0}, {1}, ..., {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.

Init(9)

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

{1, 0, 6} {8, 3, 2, 7} {4, 5}  Union(5,0)  \rightarrow  {1, 0, 4, 5} {8, 3, 2, 7}

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(): return representative.

```
INIT(8)
{3, 1, 2, 3, 4, 5, 6, 7, 8}

id[] = [0, 1, 2, 3, 4, 5, 6, 7, 8]

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

id[] = [1, 1, 1, 1, 1, 1, 3, 3]
```

```
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) time for FIND.
**Quick Union**

- **Quick union.** Maintain each set 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 the tree.
- **INIT(n):** create $n$ trees with one element each.
- **UNION(i,j):** if $FIND(i) \neq FIND(j)$, 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).**

**Time.**

- $O(n)$ time for **INIT**, $O(d)$ for **UNION** and **FIND**, where $d$ is the depth of the tree.

**Bad news.** Depth can be $n-1$.

**Challenge.** Can combine trees to limit the depth?
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.

### Weighted Quick Union

- 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 twice the size.
  - We can double the size of trees at most \( \log_2 n \) times.
  - \( \implies d_i \leq \log_2 n \).
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?

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 Find og Union operations on n elements take O(n + m \(\alpha(m,n)\)) time.
- \(\alpha(m,n)\) is the inverse of Ackermann function. \(\alpha(m,n) \leq 5\) for any practical input.
- **Theorem [Fredman-Saks 1985].** It is not possible to support m Find og Union operations O(n + m) time.
Dynamic Connectivity

- Dynamic connectivity. Maintain a dynamic graph supporting the following operations:
  - \textsc{Init}(n): create a graph \( G \) with \( n \) vertices and no edges.
  - \textsc{Connected}(u,v): determine if \( u \) and \( v \) are connected.
  - \textsc{Insert}(u,v): add edge \((u,v)\). We assume \((u,v)\) does not already exists.

\begin{center}
\begin{tikzpicture}
  \node at (0,0) [circle,fill,inner sep=1pt] (1) {1};
  \node at (1,0) [circle,fill,inner sep=1pt] (2) {2};
  \node at (2,0) [circle,fill,inner sep=1pt] (3) {3};
  \node at (3,0) [circle,fill,inner sep=1pt] (4) {4};
  \node at (4,0) [circle,fill,inner sep=1pt] (5) {5};
  \node at (5,0) [circle,fill,inner sep=1pt] (6) {6};
  \node at (6,0) [circle,fill,inner sep=1pt] (7) {7};
  \node at (7,0) [circle,fill,inner sep=1pt] (8) {8};
  \node at (8,0) [circle,fill,inner sep=1pt] (9) {9};
  \draw (1) -- (2);
  \draw (2) -- (3);
  \draw (3) -- (4);
  \draw (4) -- (5);
  \draw (5) -- (6);
  \draw (6) -- (7);
  \draw (7) -- (8);
  \draw (8) -- (9);
\end{tikzpicture}
\end{center}

\begin{center}
\begin{tikzpicture}
  \node at (0,0) [circle,fill,inner sep=1pt] (1) {1};
  \node at (1,0) [circle,fill,inner sep=1pt] (2) {2};
  \node at (2,0) [circle,fill,inner sep=1pt] (3) {3};
  \node at (3,0) [circle,fill,inner sep=1pt] (4) {4};
  \node at (4,0) [circle,fill,inner sep=1pt] (5) {5};
  \node at (5,0) [circle,fill,inner sep=1pt] (6) {6};
  \node at (6,0) [circle,fill,inner sep=1pt] (7) {7};
  \node at (7,0) [circle,fill,inner sep=1pt] (8) {8};
  \node at (8,0) [circle,fill,inner sep=1pt] (9) {9};
  \draw (1) -- (2);
  \draw (2) -- (3);
  \draw (3) -- (4);
  \draw (4) -- (5);
  \draw (5) -- (6);
  \draw (6) -- (7);
  \draw (7) -- (8);
  \draw (8) -- (9);
  \draw (3) -- (4);
\end{tikzpicture}
\end{center}

\begin{center}
\begin{tikzpicture}
  \node at (0,0) [circle,fill,inner sep=1pt] (1) {1};
  \node at (1,0) [circle,fill,inner sep=1pt] (2) {2};
  \node at (2,0) [circle,fill,inner sep=1pt] (3) {3};
  \node at (3,0) [circle,fill,inner sep=1pt] (4) {4};
  \node at (4,0) [circle,fill,inner sep=1pt] (5) {5};
  \node at (5,0) [circle,fill,inner sep=1pt] (6) {6};
  \node at (6,0) [circle,fill,inner sep=1pt] (7) {7};
  \node at (7,0) [circle,fill,inner sep=1pt] (8) {8};
  \node at (8,0) [circle,fill,inner sep=1pt] (9) {9};
  \draw (1) -- (2);
  \draw (2) -- (3);
  \draw (3) -- (4);
  \draw (4) -- (5);
  \draw (5) -- (6);
  \draw (6) -- (7);
  \draw (7) -- (8);
  \draw (8) -- (9);
  \draw (3) -- (4);
\end{tikzpicture}
\end{center}

Dynamic Connectivity

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

\begin{center}
\begin{tikzpicture}
  \node at (0,0) [circle,fill,inner sep=1pt] (1) {1};
  \node at (1,0) [circle,fill,inner sep=1pt] (2) {2};
  \node at (2,0) [circle,fill,inner sep=1pt] (3) {3};
  \node at (3,0) [circle,fill,inner sep=1pt] (4) {4};
  \node at (4,0) [circle,fill,inner sep=1pt] (5) {5};
  \node at (5,0) [circle,fill,inner sep=1pt] (6) {6};
  \node at (6,0) [circle,fill,inner sep=1pt] (7) {7};
  \node at (7,0) [circle,fill,inner sep=1pt] (8) {8};
  \node at (8,0) [circle,fill,inner sep=1pt] (9) {9};
  \draw (1) -- (2);
  \draw (2) -- (3);
  \draw (3) -- (4);
  \draw (4) -- (5);
  \draw (5) -- (6);
  \draw (6) -- (7);
  \draw (7) -- (8);
  \draw (8) -- (9);
  \draw (3) -- (4);
  \draw[red] (3) -- (4);
\end{tikzpicture}
\end{center}

- 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