Data Structures for Disjoint Sets

Ugur Dogrusoz
Computer Eng Dept, Bilkent Univ
Disjoint-set data structure

- Maintains a collection of disjoint dynamic sets
  \[ S = \{ S_1, S_2, \ldots, S_k \} \]

  - Identify each set with a representative, a member of the set

  - Supported operations:
    - \texttt{MAKE-SET}(x) creates a new set with member x
    - \texttt{UNION}(x,y) unites dynamic sets containing x and y
    - \texttt{FIND-SET}(x) returns a pointer to representative of set containing x

- Analyze running time using \( n \): # of \texttt{MAKE-SET} operations and \( m \): # of total operations (\texttt{MAKE-SET}, \texttt{UNION}, and \texttt{FIND-SET})
An application: connected components

- Construct and answer connected components of an undirected graph

![Graph Diagram]

<table>
<thead>
<tr>
<th>Edge processed</th>
<th>Collection of disjoint sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial sets</td>
<td>{a}</td>
</tr>
<tr>
<td>(b, d)</td>
<td>{a}</td>
</tr>
<tr>
<td>(e, g)</td>
<td>{a}</td>
</tr>
<tr>
<td>(a, c)</td>
<td>{a, c}</td>
</tr>
<tr>
<td>(h, i)</td>
<td>{a, c}</td>
</tr>
<tr>
<td>(a, b)</td>
<td>{a, b, c, d}</td>
</tr>
<tr>
<td>(e, f)</td>
<td>{a, b, c, d}</td>
</tr>
<tr>
<td>(b, c)</td>
<td>{a, b, c, d}</td>
</tr>
</tbody>
</table>
An application: connected components

CONNECTED-COMPONENTS \((G)\)
1. \textbf{for} each vertex \(v \in G.V\)
2. \textbf{MAKE-SET} \((v)\)
3. \textbf{for} each edge \((u, v) \in G.E\)
4. \textbf{if} \text{FIND-SET} \((u) \neq \text{FIND-SET} \((v)\)\)
5. \textbf{UNION} \((u, v)\)

SAME-COMPONENT \((u, v)\)
1. \textbf{if} \text{FIND-SET} \((u) == \text{FIND-SET} \((v)\)\)
2. \textbf{return} \text{TRUE}\)
3. \textbf{else} \textbf{return} \text{FALSE}
Disjoint sets: linked list representation

Make-Set(x) \quad O(1)
Find-Set(x) \quad O(1)
Union(x,y) \quad O(n) \quad \theta(n) amortized running time
### Disjoint sets: linked list representation

<table>
<thead>
<tr>
<th>Operation</th>
<th>Number of objects updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAKE-SET($x_1$)</td>
<td>1</td>
</tr>
<tr>
<td>MAKE-SET($x_2$)</td>
<td>1</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>MAKE-SET($x_n$)</td>
<td>1</td>
</tr>
<tr>
<td>UNION($x_2, x_1$)</td>
<td>1</td>
</tr>
<tr>
<td>UNION($x_3, x_2$)</td>
<td>2</td>
</tr>
<tr>
<td>UNION($x_4, x_3$)</td>
<td>3</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>UNION($x_n, x_{n-1}$)</td>
<td>$n - 1$</td>
</tr>
</tbody>
</table>

$\theta(n^2) / n = \theta(n)$ amortized running time
Disjoint sets: linked list representation

- Weighted union heuristic:
  - always append the shorter list onto the longer, breaking ties arbitrarily
  - a single union will still take $\Omega(n)$ time if both sets have $\Omega(n)$ members
Theorem Using linked-list representation of disjoint sets and weighted-union heuristic, a sequence of $m$ Make-Set, Union, and Find-Set operations, $n$ of which are Make-Set operations, takes $O(m + n \lg n)$ time.

Proof For $k \leq n$, after $x$’s pointer has been updated ⌊lg $k$⌋ times, the resulting set must have at least $k$ members.
Disjoint sets: forest representation

- Sets by rooted trees, with each node containing one member and each tree representing one set.

\[\text{UNION}(e,g) = \{b,c,e,h,d,f,g\}\]
Heuristics: union by rank

- Make root with smaller rank point to root with larger rank during a `UNION` operation.
  - `rank`: an upper bound on height of a node.
  - Only the rank of the roots may change:
    - If both roots have the same rank, rank of new root increases by 1.
    - Otherwise, no change.

- $O(m \log n)$ (every node has rank at most $\lfloor \log n \rfloor$)
Heuristics: path compression

- make each node on the find path point directly to root (during \texttt{FIND-SET})
  - path compression does not change any ranks
  - for a sequence of \( n \) \texttt{MAKE-SET} operations and \( f \) \texttt{FIND-SET} operations, worst-case running time is \( \Theta(n + f(1 + \log_{2+f/n} n)) \)
Disjoint set forests: operations

**MAKE-SET(x)**
1. $x.p = x$
2. $x.rank = 0$

**UNION(x, y)**
1. `LINK(FIND-SET(x), FIND-SET(y))`

**LINK(x, y)**
1. `if x.rank > y.rank`
2. $y.p = x$
3. `else x.p = y`
4. `if x.rank == y.rank`
5. $y.rank = y.rank + 1$

**FIND-SET(x)**
1. `if x $\neq x.p$
2. $x.p = \text{FIND-SET}(x.p)$
3. `return x.p`
Disjoint set forests, both heuristics

- **Worst case running time is** $O(m \alpha(n))$

  - where $\alpha(n)$ is a very slowly growing function (inverse of very fast growing function called Ackermann function $A_k(n)$)

  $\alpha(n) = \begin{cases} 
  0, & \text{for } 0 \leq n \leq 2, \\
  1, & \text{for } n = 3, \\
  2, & \text{for } 4 \leq n \leq 7, \\
  3, & \text{for } 8 \leq n \leq 2047, \\
  4, & \text{for } 2048 \leq n \leq A_4(1) \ [A_4(1) \geq 10^{80}], 
  \end{cases}$