Disjoint Sets

Definition
Linked List Representation
Disjoint-Set Forests
A disjoint set data structure supports the following operations. $x$ and $y$ are elements.

- **MAKE-SET($x$):** Creates a new set $\{x\}$. $x$ must not be in any other set.
- **UNION($x, y$):** Combine the set that contains $x$ with the set that contains $y$.
- **FIND-SET($x$):** $\text{FIND-SET}(x) = \text{FIND-SET}(y)$ iff $x$ and $y$ are in the same set.
Example: Connected components of an undirected graph

\[
\text{CONNECTED-COMPONENTS}(G) \\
\text{for each vertex } v \text{ in graph } G \\
\text{MAKE-SET}(v) \\
\text{for each edge } (u, v) \text{ in graph } G \\
\text{if } \text{FIND-SET}(u) \neq \text{FIND-SET}(v) \\
\text{then UNION}(u, v)
\]
Linked List Representation

- Disjoint sets example
- Linked list operations analysis 1
- Analysis 2
- Disjoint-set forests operations 1
- Operations 2
- $O(\lg n)$ analysis 1
- $O(\lg n)$ analysis 2
- $O(\lg \lg n)$ analysis 1
- $O(\lg \lg n)$ analysis 2
- $O(\lg \lg n)$ analysis 3
**Linked List Operations**

- **MAKE-SET**(x)
  
  ```
  set ← new set
  set.head ← x
  set.tail ← x
  set.size ← 1
  x.set ← set
  x.next ← NIL
  ```

- **FIND-SET**(x)
  
  ```
  return x.set
  ```

- **UNION**(x, y)
  
  ```
  sx ← FIND-SET(x)
  sy ← FIND-SET(y)
  if sx = sy return
  if sx.size < sy.size then exchange sx ↔ sy
  sx.tail.next ← sy.head
  sx.tail ← sy.tail
  sx.size ← sx.size + sy.size
  while y ≠ NIL
    y.set ← sx
    y ← y.next
  ```
Assume $m$ operations including $n$ MAKE-SETS. Analyze number of assignments to $set$ field.

Using accounting method of amortized analysis:
Use amortized cost $1 + \lg n$ units per MAKE-SET.
Consider an arbitrary element $x$.
Use one unit immediately for MAKE-SET($x$).
Use one unit each time UNION modifies $x.set$. 
**Amortized Analysis Continued**

**UNION** changes *set* fields of the smaller set, so a change to \( x.set \) at least doubles \( x \)'s set size.

The size of a set cannot exceed \( n = 2^{\lg n} \), so the cost \( \lg n + 1 \) covers all changes to \( x.set \).

Over \( n \) elements, the total amortized cost is \( n(\lg n + 1) \).

There can be \( O(m) \) **FIND-SET** operations, so total time is \( O(m + n \lg n) \).

Easy to show \( n - 1 \) **UNIONS** can make \( (n/2) \lg n \) changes to *set* fields, so \( O(m + n \lg n) \) is tight.
Disjoint-Set Forests

- Disjoint sets
- Example
- Linked list
- Operations
- Analysis 1
- Analysis 2

- Disjoint-set forests

operations 1
- $O(\lg n)$ analysis 1
- $O(\lg n)$ analysis 2

operations 2
- $O(\lg \lg n)$ analysis 1
- $O(\lg \lg n)$ analysis 2
- $O(\lg \lg n)$ analysis 3

(a)

(b)
Disjoint-Set Forest Operations

Assume $x$ has fields $parent$ and $rank$.

**MAKE-SET($x$)**

$x.parent \leftarrow x$

$x.rank \leftarrow 0$

**FIND-SET($x$)**

if $x = x.parent$

then return $x$

else $y \leftarrow$ FIND-SET($x.parent$)

$x.parent \leftarrow y$

return $y$
**Union**

\[ \text{Union}(x, y) \]

\[ x \leftarrow \text{Find-Set}(x) \]
\[ y \leftarrow \text{Find-Set}(y) \]
\[ \text{if } x = y \text{ return} \]
\[ \text{if } x.\text{rank} > y.\text{rank} \]
\[ \text{then } y.\text{parent} \leftarrow x \]
\[ \text{else } x.\text{parent} \leftarrow y \]
\[ \text{if } x.\text{rank} = y.\text{rank} \]
\[ \text{then } y.\text{rank} \leftarrow y.\text{rank} + 1 \]
The rank $r$ is $\geq$ the height $h$ of the tree:

Basis: When $r = 0$, then $h = 0$.

Assume: When $r = k$, then $h \leq k$.

Show: When $r = k + 1$, $h \leq k + 1$.

Induction: $h$ can increase only when combining two trees with same $r$. 

A tree with rank $r$ has $\geq 2^r$ nodes.

**Basis:** When $r = 0$, there is $2^0 = 1$ node.

**Assume:** When $r = k$, there are $\geq 2^k$ nodes.

**Show:** When $r = k + 1$, there are $\geq 2^{k+1}$ nodes.

**Induction:** $r$ increases only when combining two trees with same $r$. Their union must have $\geq 2(2^k) = 2^{k+1}$ nodes.

Without considering compression, this implies that $\text{FIND-SET}$ will traverse $\leq \lg n$ links/call.
Let $f$ be the number of \texttt{FIND-SET}s excluding recursion.

Let $l_1, \ldots, l_f$ be the number of links compressed by calls to \texttt{FIND-SET}. Want to bound $\sum_{i=1}^{f} l_i$. Why?

If a call to \texttt{FIND-SET} compresses $l$ links, then $\geq 2^{l-1}$ nodes are closer to the root. Proof: A subtree of rank $\geq l - 1$ now points to the root. This subtree has $\geq 2^{l-1}$ nodes.
Let \( l \) be the average of \( l_1, \ldots, l_f \).
The number of recursive calls is \( fl \).

It can be shown that
\[
\sum_{i=1}^{f} 2^{l_i-1} \geq f 2^{l-1}.
\]
For example, note that
\[
2^{l-1} + 2^{l+1} > 2^l + 2^l.
\]

There is \( \leq n \lg n \) “distance” to compress because each node is \( \leq \lg n \) away from the root.

\[
\leq n \lg n \text{ distance to compress implies } f 2^{l-1} \leq n \lg n.
\]
Case 1: \( f \leq n/(\lg n) \)
\[
f \leq n/(\lg n) \text{ and } l \leq \lg n \text{ imply } fl \leq n.
\]

Case 2: \( f \geq n/(\lg n) \)

Start with \( f2^{l-1} \leq n \lg n \).

Implies \((\lg f) + l - 1 \leq (\lg n) + (\lg \lg n) \)

Implies \(l - 1 \leq (\lg n) + (\lg \lg n) - (\lg f) \)

\[
\leq (\lg n) + (\lg \lg n) - \lg(n/\lg n)
\]

\[
= (\lg n) + (\lg \lg n) - (\lg n) + (\lg \lg n)
\]

\[
= 2 \lg \lg n.
\]

so \( l \) is \( O(\lg \lg n) \), which implies that the total cost of \( f \) FIND-SETS is \( O(n + fl \lg \lg n) \).