Disjoint Sets

\[ n \text{ elements} \quad \text{Partitioned into } k \text{ sets (disjoint sets)} \]

\[ X = \{x_1, x_2, \ldots, x_n\} \quad S_1, S_2, \ldots, S_k \]

Operations

- \text{Find-Set}(x) - return set containing \( x \)
- \text{Union}(x, y) - make a new set by combining the sets containing \( x \) and \( y \) (destroying them)
- \text{MakeSet}(x) - add a new element \( x \) and make it into a singleton set

Disjoint Sets (cont)

Each set is identified by its representative element \( x \)

- \( \{13, 21, 15, 2, 4, 20, 12\} \)
- \( \{8, 6, 16, 14, 18\} \)
- \( \{9\} \quad \{1\} \quad \{11\} \)
- \( \{10, 3, 5, 17, 7, 19\} \)

\text{Find-Set}(4) \quad \rightarrow 13
\text{Find-Set}(9) \quad \rightarrow 9

Disjoint Sets (cont)

\( \text{Union}(4, 16) \)
\text{Find-Set}(4) \quad \rightarrow 8

Disjoint Sets (cont)

\text{MakeSet}(22)

\( \{8, 15, 14, 16, 18\} \)
**Disjoint Sets (cont)**

**Version 1**

Each set is a linked list (with pointer to tail)
List head is the representative
Each element has a pointer to the head

\{3, 21, 15, 2, 4, 20, 12\}
\{8, 6, 16, 14, 18\}
\{9\}
\{11\}
\{1\}
\{10, 3, 5, 17, 7, 19\}

**Find-Set(x)** - traverse head pointer \(O(1)\)

**MakeSet(x)** - create \(\Omega(1)\)

**Union(x, y)** - Append \(S_x\) to \(S_y\) and update head pointers \(O(|S_x|)\)

**Disjoint Sets (cont)**

**Version 2** (weighted-union heuristic)

Modify Version 1 by keeping track of the size of each set and appending the smaller to the larger.

Each time \(x\)'s head pointer is updated the size of \(x\)'s set doubles.

A head pointer has been updated at most \(\lg n\) times if there are a total of \(n\) elements in the sets.

A sequence of \(m\) instructions in which there are \(n\) MakeSets takes time \(O(m + n \lg n)\).
Disjoint Sets (cont)  

**Version 3**

Each set is a tree with parent pointers only
The root of the tree is the representative

**Version 4** (weighted-union and path compression)

MakeSet(x) - create

Union(x,y) - Link(FindSet(x),FindSet(y))

- \( O(1) \) + time for 2 FindSets

\[
\begin{align*}
\text{MakeSet}(x) & : p(x) \leftarrow x, r(x) \leftarrow 0 \\
\text{Union}(x,y) & : \\
& \text{if } r(x) > r(y) \\
& \quad p(y) \leftarrow x \\
& \text{else } p(x) \leftarrow y \\
& \quad \text{if } r(x) = r(y) \\
& \qquad r(y) ++ \\
& \text{return } p(x)
\end{align*}
\]
Disjoint Sets (cont)  Version 4

FindSet(x) - Traverse parent pointers to reach root (till \( x = \text{parent}(x) \))

\[
\text{Find-Set}(4) \rightarrow 13
\]

Disjoint Sets (cont)  Version 4 Analysis

Lemma If \( x \neq p(x) \) then \( \text{rank}[x] < \text{rank}[p(x)] \).

Proof: By induction on the number of operations.

Base: 0 Link operations. All nodes are roots with size 1 and rank 0.

Step: Assume property holds after \( i \) Links and consider the Link \( i + 1 \). Link \( (x, y) \).

Case 1 \( \text{rank}(x) > \text{rank}(y) \) (or vice versa). Then \( \text{rank}(x) \) does not change and \( \text{size}(x) \) increases so inequality still holds.

Case 2 \( \text{rank}(x) = \text{rank}(y) = r \).

Then before the link \( \text{size}(x) \geq 2^r \) and \( \text{size}(y) \geq 2^r \).

After link \( \text{rank}(y) = r + 1 \) and \( \text{size}(y) \geq 2^r + 2^r = 2^{r+1} = 2^{\text{rank}(y)} \).

QED

Disjoint Sets (cont)  Version 4 Analysis

Lemma For any \( r \geq 0 \), the # of nodes of rank \( r \) is at most \( \frac{n}{2^r} \).

Proof: A node gets rank \( r \) as the result of a Link of two trees of rank \( r-1 \).

This node will have size at least \( 2^r \) at that time.

No node can contribute more than once to a rank change from \( r-1 \) to \( r \).

Since there are \( n \) nodes, at most \( \frac{n}{2^r} \) nodes can attain rank \( r \).

Corollary For any node \( x \), \( \text{rank}(x) \leq \lfloor \lg n \rfloor \).
Disjoint Sets (cont) **Version 4 Analysis**

**First try at bounding cost of m instructions**

Maximum cost of a FindSet is $O(\lg n)$.

A sequence of $m$ operations costs at most $O(m \lg n)$.

Over estimate since FindSets are destructive, reducing height of trees.

Cannot find sequence with $\Omega(m)$ FindSets each with cost $\Omega(\lg n)$.

**Idea**: Bound number of parent updates (as in Version 2).

**Second try at bounding cost of m instructions**

How many times can node $x$ of rank $r$ get a new parent?

At most $\lg n - r$ times. (Its parents increase in rank from $r+1$ to $\lg n$.)

The cost of FindSet is at most $O(m + \# of parent updates)$.

# parent updates $\leq \sum_{r=0}^{\lg n} (\lg n - r) \frac{n}{2^r}$

$\leq n \lg n \sum_{r=0}^{\lg n} \frac{1}{2^r} - n \sum_{r=0}^{\lg n} \frac{r}{2^r}$

$\leq n \lg n \left( 2 - \frac{1}{2^{\lg n}} \right) - n \left( 2 - \frac{\lg n + 2}{2^{\lg n}} \right)$

$\leq 2n \lg n - \lg n - 2n + \lg n + 2 = \Theta(n \lg n)$.

Cost of $m$ operations is $O(m + n \lg n)$.

Over estimate! Not many branches with consecutive ranks.

Cannot find $\Omega(n)$ nodes with $\Omega(\lg n)$ parent updates.

**Third try at bounding cost of m instructions**

Charge some parent updates to the node and the rest to the operation.
**Disjoint Sets (cont)  Version 4 Analysis**

Amortized Analysis using Aggregate Method

If \( \text{FindSet} \) traverses nodes \( x_1, x_2, \ldots, x_p \), then the cost for traversing \( x \) is charged:

- to the node if \( x \) and \( p(x) \) are in the same rank group (before) and \( p(x) \) is not the root.
- otherwise to the operation.

Path Charge

Block Charge

Total cost = total path charges + total block charges

Maximum block charges to single \( \text{FindSet} \) is \( f^{-1}(\lg n) \)

Maximum block charges to a sequence of \( m \) operations is \( O(m + m \cdot f^{-1}(\lg n)) \)

---

**Disjoint Sets (cont)  Version 4 Analysis**

Amortized Analysis using Aggregate Method

To bound path charges:

If node \( x \) has a parent in a different group, this will remain the case. \( x \) in group \( i \) can be charged at most \( f(i) - (f(i-1)+1) \) path charges.

Total path charges \( \leq \sum_{i=1}^{f^{-1}(\lg n)} \sum_{r=f(i-1)+1}^{f(i)} \frac{n}{2^r} \left( f(i) - (f(i-1)+1) \right) \)

\( \leq n \sum_{i=1}^{f^{-1}(\lg n)} f(i) \left( \frac{1}{2^{f(i-1)}} - \frac{1}{2^{f(i)}} \right) \)

Time to choose \( f() \)