Lecture 12

Instructor: Karteek Sreenivasaiah

28th September 2018

Disjoint Set

Maintain a collection $\mathcal{F} = \{S_1, S_2, \ldots, S_k\}$ of disjoint sets.
One element from each set serves as a 'representative' for that set.

Disjoint Set supports the following procedures:

- $\textsc{MakeSet}(x)$ – Creates a singleton set with element x.
- $\textsc{Union}(x, y)$ – Performs union on sets containing x and y.
- $\textsc{FindSet}(x)$ – Find the set containing x.

Recap: List Implementation

Disjoint Set using linked lists:

- For each set S, maintain:
  - a node with metadata
  - a linked list L_S with the objects in the set.
- The "Metadata Node" stores:
  - Head and tail pointers to the linked list.
- Each node in the linked list consists of:
  - The value of the element.
  - A pointer to the next element.
  - A pointer to the Metadata Node.

The head of L_S is the representative of S.

List Implementation - Union by Rank heuristic

Disjoint Set using linked lists, union by rank:

- For each set S, maintain:
  - a node with metadata
  - a linked list L_S with the objects in the set.
- The "Metadata Node" stores:
  - Head and tail pointers to the linked list.
  - Size of the set.
- Each node in the linked list consists of:
  - The value of the element.
  - A pointer to the next element.
  - A pointer to the Metadata Node.

The head of L_S is the representative of S.

List Implementation - Union by Rank heuristic

UNION(X,Y)

Union of sets containing x and y.

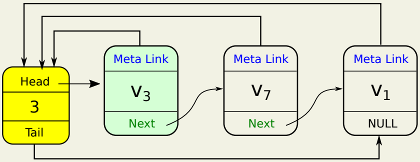
Let $x \in S$ and $y \in T$.
If $|S| \leq |T|$,

- Append list of S to tail end of list of T.
- Representative of new set is same as that of T.
- Update meta pointers of nodes in S
- Update tail pointer in metadata node of T.
- Update size of set in the metadata node.

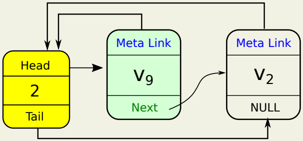
Else, do the opposite.

Implementation - Union by Rank heuristic

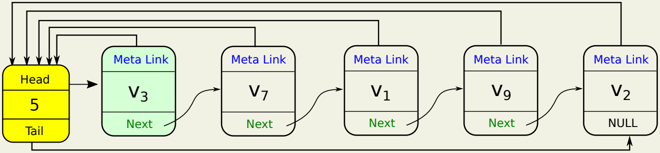
Union of sets $S = \{v_1, v_3, v_7\}$ and $T = \{v_2, v_9\}$.



S



T



Union(v_1,v_2)

Analysis - Union by Rank heuristic

Theorem

A sequence of m operations in total, n of which are MakeSet takes $O(m + n \log n)$ time.

Analysis - Union by Rank heuristic

Observation 1

Updating the meta pointers takes the most time.

Observation 2

The meta pointer of a node x is updated only when union happens with a bigger set.

Proof strategy

- Fix an element x.
- Count number of times the meta pointer on node x is updated.

Analysis - Union by Rank heuristic

Observation 2 (informal)

If x lived inside a set of size s,
and a union operation updated its meta pointer,
then x now lives inside a set of size at least $2s$.

- Initially, x starts off as a singleton set.
- After k many updates to its meta pointer, it lives inside a set of size at least 2^k.
- Total number of elements is n. So $2^k \leq n$.
- This means $k \leq \log n$

Hence, for each element the meta pointer can be updated at most $k \leq \log n$ many times.
Worst case total number of updates to meta pointer across all n elements is $n \log n$.

$\square$

Implementation - disjoint forests

The disjoint forest implementation:

- Each set S is implemented as a rooted tree.

A node corresponding to an $x \in S$ contains:

- The value (or pointer to) x.
- A pointer to its parent.

Implementation - disjoint forests

The disjoint forest implementation:

- Each set S is implemented as a rooted tree.

A node corresponding to an $x \in S$ contains:

- The value (or pointer to) x.
- A pointer to its parent.

Note:

- There are no pointers to children nodes!
- There is no dedicated metadata node for each set.
- Convention: Parent of root will be itself.
- Root node is also the representative.

Implementation - disjoint forests

Example picture on whiteboard

MakeSet(x)

MakeSet(x) involves creating a new tree with a single node for x.

MakeSet(x)

Whiteboard

FindSet(x)

FindSet(x):

- Start at node x.
- Follow the parent pointer starting from x.
- Return the root.

$\textsc{FindSet}(x)$

$\textsc{Union}(x, y)$:

- $r_x \leftarrow \textsc{FindSet}(x)$
- $r_y \leftarrow \textsc{FindSet}(y)$.
- $\text{parent}(r_x) \leftarrow r_y$.

Whiteboard

Analysis – disjoint forests

Worst case running times under disjoint forest implementation:

- MakeSet(x) –

Analysis – disjoint forests

Worst case running times under disjoint forest implementation:

- MakeSet(x) –$O(1)$
- FindSet(x) –

Analysis – disjoint forests

Worst case running times under disjoint forest implementation:

- $\textsc{MakeSet}(x)$ $-O(1)$
- $\textsc{FindSet}(x)$ $-O(n)$
- $\textsc{Union}(x, y)$ –

Analysis – disjoint forests

Worst case running times under disjoint forest implementation:

- $\textsc{MakeSet}(x)$ $-O(1)$
- $\textsc{FindSet}(x)$ $-O(n)$
- $\textsc{Union}(x, y)$ $-O(n)$

where n is the number of elements handled.

Disjoint Forests – Union by Rank heuristic

To use the Rank heuristic:

- Each node will contain a “rank”.
- Every node starts with a rank of 1.
- Update rank only when Union is called.

Disjoint Forests – Union by Rank heuristic

UNION(x, y):
Let $x \in S$, $y \in T$ with representatives r_S and r_T respectively.

If $\text{rank}(r_S) > \text{rank}(r_T)$, then:

- $\text{parent}(r_T) \leftarrow (r_S)$.

Else:

- $\text{parent}(r_S) \leftarrow (r_T)$.

If $\text{rank}(r_S) = \text{rank}(r_T)$, then:

- Increment $\text{rank}(r_S)$.

Note: A set S rooted at x has at least as many elements as the height of x.

Analysis – Union by Rank

Theorem

A sequence of m operations in total, n of which are MakeSet takes $O(m + n \log n)$ time.

Analysis – Union by Rank

Observation 1

The rank of a node is exactly the height of the node.

Proof

Let x be a node. Induction on number of updates to rank x:
Base case: After MAKESET(x), rank is 1.
Step: The rank of x is incremented when:

- Union is called involving a set (tree) S with root x and T with root y.
- $\text{rank}(x) = \text{rank}(y)$.

Analysis – Union by Rank

Observation 1

The rank of a node is exactly the height of the node.

Proof

Let x be a node. Induction on number of updates to rank x:
Base case: After $\textsc{MakeSet}(x)$, rank is 1.
Step: The rank of x is incremented when:

- Union is called involving a set (tree) S with root x and T with root y.
- $\text{rank}(x) = \text{rank}(y)$.
- From induction $\text{rank}(y) = \text{height}(y)$.
- y's parent becomes x. Hence $\text{height}(x)$ increases by one.
- The increment done to $\text{rank}(x)$ reflects this change of height.

Analysis – Union by Rank

Lemma 1

Every node has rank at most $\log n$

Proof

Fix a node x. If $\text{rank}(x)$ is incremented, then:

- Union was called on $S \ni x$ and a set T with root y.
- Node x was the root of the tree representing S.
- We have $|S| \geq \text{height}(x) = \text{rank}(x)$ (Obs 1).
- $\text{height}(y) = \text{rank}(y) = \text{rank}(x)$.

Analysis – Union by Rank

Lemma 1

Every node has rank at most $\log n$

Proof

Fix a node x. If $\text{rank}(x)$ is incremented, then:

- Union was called on $S \ni x$ and a set T with root y.
- Node x was the root of the tree representing S.
- We have $|S| \geq \text{height}(x) = \text{rank}(x)$ (Obs 1).
- $\text{height}(y) = \text{rank}(y) = \text{rank}(x)$.
- So $|T| \geq \text{height}(y)$= $\text{rank}(x)$.
- Hence $|S \cup T| \geq 2\ \text{rank}(x)$.

So after k increments to $\text{rank}(x)$, the cardinality of the set in which x lives is at least 2^k

Since total number of elements in n, the claim follows.

Disjoint Forests – Path Compression heuristic

When FindSet(x) is called:

- Follow the parent pointer from x to root.
- Change the parent pointer of every node on this path to directly point to root.

Disjoint Forests – Path Compression heuristic

Whiteboard

Analysis – Union by Rank and Path Compression

Theorem

A sequence of m operations in total, n of which are MakeSet takes $O(m\alpha(n))$ time where $\alpha(n)$ is the inverse Ackermann

Ackermann function

The Ackermann function $A(m, n)$ is defined as:

- $n + 1$ if $m = 0$
- $A(m - 1, 1)$ if $m > 0$ and $n = 0$
- $A(m - 1, A(m, n - 1))$ if $m > 0$ and $n > 0$

Ackermann function

The Ackermann function $A(m, n)$ is defined as:

- $n + 1$ if $m = 0$
- $A(m - 1, 1)$ if $m > 0$ and $n = 0$
- $A(m - 1, A(m, n - 1))$ if $m > 0$ and $n > 0$

Example values:

- $A(0, 0) = 1$
- $A(1, 1) = 3$
- $A(2, 2) = 7$
- $A(3, 3) = 61$
- $A(4, 4) = 2^{2^{65536}} - 3$

(That escalated quickly!)

Ackermann function

The Ackermann function $A(m, n)$ is defined as:

- $n + 1$ if $m = 0$
- $A(m - 1, 1)$ if $m > 0$ and $n = 0$
- $A(m - 1, A(m, n - 1))$ if $m > 0$ and $n > 0$

Example values:

- $A(0, 0) = 1$
- $A(1, 1) = 3$
- $A(2, 2) = 7$
- $A(3, 3) = 61$
- $A(4, 4) = 2^{2^{65536}} - 3$

(That escalated quickly!)

The inverse Ackermann $\alpha(n)$ is the smallest k for which $n < A(k, k)$.