Lecture 16: Advanced Data Structures - Disjoint Sets

- "Union-Find" data structures (see Lect 04)
- amortized analysis

READING: CLRS Chpt 21 (§21.4 is optional)

General situation - want to maintain dynamic collection of elements that grow and merge but do not shrink (elements appear but none disappear)

→ Problem arises in a number of data handling and analysis situations

→ Especially for Minimum Spanning Tree (Kruskal)

Problem: Want to maintain dynamic collection of pairwise disjoint sets \( S = \{S_1, S_2, ... S_r\} \) with one representative element per set, rep \([S_i]\).

Supported Operations:
- Make-Set\( (u) \) - create new set with \( u \) as representative
- Find-Set\( (u) \) - returns representative of set containing element \( u \), rep \([S_u]\)
- Union\( (u, v^-) \) - replace \( S_u \) and \( S_v \) with \( S_u \cup S_v \) in \( S \)
  
  for any \( u, v \) in distinct sets \( S_u, S_v \)
  
  update representative
Simple Linked-list Solution - unordered

\[ \text{Rep} \]

\[ \text{SU: } \]

\[ u_1 \leftarrow \rightarrow u_2 \rightarrow u_3 \rightarrow \ldots \rightarrow u_n \]

\[ \text{Make-Set}(u) - \text{initialize lone node} \quad \Theta (1) \]
\[ \text{Find-Set}(u) - \text{walk left to head} \quad \Theta (n) \]
\[ \text{Union}(u, v) - \text{walk to head of one, tail of other, insert pointers; automatically updates representative} \quad \Theta (n) \]

\{ worst case \}

Why are \text{Find-Set} and \text{Union} so slow?
Can we eliminate all that walking? \ldots teleportation

---

SU:

- \text{Make-Set} \quad \Theta (1)
- \text{Find-Set} \quad \Theta (1)
- \text{Union} \quad \Theta (n)

---

1. Can I remove the "\( \leftrightarrow \)" arrows?
2. Is the \( \Theta (n) \) for union a problem that needs improving - 2 hops to head?

3. What is the total cost of \( n \) \text{MAKE-SETS} followed by \( n \) \text{UNIONS}?
   - Initial sets \( \{1, 2, 3, \ldots, n\} \)
   - \text{union}(n-1, n), \text{union}(n-2, n-1), \ldots, \text{union}(1, 2)
   - \text{union}(n-i, n-i+1) modifies a list of length \( i \)
   - Total union cost is \( \sum_{i=1}^{n-1} i = \Theta (n^2) \) \text{Amortized}

4. Can we do the union so as to minimize cost?
   - merge "smaller into larger" so have fewer pointer updates
   - need to store and update weight = size of set.
Amortized analysis for "smaller into larger" improvement

Let \( m \) = total number of operations

\( N \) = total number of elements (MAKE-SETs)

Will show:

Cost of all Unions is \( \Theta(n \lg n) \)
Total cost is \( \Theta(m + n \lg n) \)

- Focus on element \( u \) in set \( S_u \)
- When created with MAKE-SET, weight \([S_u]\) = 1
- Whenever \( S_u \) merges with another set \( S_v \)
  - If \( \text{weight}[S_v] \geq \text{weight}[S_u] \)
    - pay 1 unit cost to update head pointer for \( u \)
    - \( \text{weight}[S_u] \) at least doubles
  - Else
    - pay no cost
    - \( \text{weight}[S_u] \) increases

- So analysis of Unions gives:
  - Each time we pay 1, weight at least doubles
  - Maximum weight is \( n \)
  - Maximum cost \( \leq \lg n \) for one element \( u \),
    or \( \Theta(n \lg n) \) overall

So total cost of \( m \) operations on \( n \) elements is \( \Theta(m + n \lg n) \)
Forest of Trees Solution

MAKE-SET (u) - initializes u as lone node - \( \Theta(1) \)

FIND-SET (u) - walks up to root - \( \Theta(\text{height}) \)

\[ \text{WEAK-UNION}(u,v) \] - merges representative nodes u,v - \( \Theta(1) \)

\[ \text{UNION}(u,v) = \text{WEAK-UNION}(\text{FIND-SET}(u), \text{FIND-SET}(v)) \] - \( \Theta(1) + 2T_{\text{FIND-SET}} \)

So, short trees \( \Rightarrow \) efficient algorithm \( \Rightarrow \) How can we keep the trees short

Idea 1: "Smaller into larger" \( \Rightarrow \) "Shorter into Taller" Trees

Merge shorter tree into larger, keeping track of rank (upper based on height of tree).

Move pointer \( v \) to \( u \), \( \Rightarrow \) \( u \), as new parental pointer.

Increase rank of \( u \) by 1 if

\[ u_{\cdot \text{rank}} = v_{\cdot \text{rank}} \]

"Union by Rank"
What does this do for running time?

It can be shown that the maximum rank of a node is \( \lg n \), where \( n \) is the number of elements, so

\[
\begin{align*}
\text{MAKE-SET}(u) & \quad \Theta(1) \\
\text{FIND-SET}(u) & \quad \Theta(\lg n) \\
\text{UNION}(u, v) & \quad \Theta(1) \\
\text{UNION}(u) & \quad \Theta(\lg n)
\end{align*}
\]

So \( m \) operations on \( n \) elements is \( \Theta(m \lg n) \)

So what else can we do?

What about FIND-SET?

Idea 2: Path Compression

When we execute a FIND-SET, we walk up a path of nodes to the root (representative). By re-directing the parental pointer of each of those nodes, they become direct children of the root. This locally flattens the tree.

Path compression doesn't update ranks, which is why they are upper bounds.
With path compression alone (but not union by rank), it can be shown that

\[ n \text{ MAKE-SET operations (and so at most n-1 UNIONS)} \]

and \( f \) FIND-SET operations has worst-case running time

\[ \Theta(n + f(1 + \log_{2+f/n} n)) \]

---

Really impressive result... Both improvements together lead to spectacular behavior

Define \( A_k(j) = \begin{cases} 
  j+1 & \text{for } k=0 \\
  A_{k-1}(A_{k-1}(j)) & \text{for } k \geq 1
\end{cases} \) Ackermann's function

\[ k \geq 0, j \geq 1 \text{ integers} \]

\[ A_1(j) = 2j+1 \quad \rightarrow \quad A_1(1) = 3 \]

\[ A_2(j) = 2^{j+1} - 1 \quad \rightarrow \quad A_2(1) = 7 \]

\[ A_3(1) = 2047 \]

\[ A_4(1) \gg 10^{80} \quad \text{\# atoms in observable universe} \]

\[ \rightarrow \text{This is an extremely rapidly increasing function} \]
Take its inverse:
\[ \lambda(n) = \min \{ k : A_k(i) \geq n \} \]
\[ \lambda(n) \] is the lowest level \( k \) for which \( A_k(i) \geq n \).

\[ \lambda(n) = \begin{cases} 
0 & \text{for } 0 \leq n \leq 2 \\
1 & \text{for } n=3 \\
2 & \text{for } 4 \leq n \leq 7 \\
3 & \text{for } 8 \leq n \leq 2047 \\
4 & \text{for } 2048 \leq n \leq A_4(i) \\
\infty & \text{for } n > A_4(i) 
\end{cases} \]

Inverse Ackermann Function

\[ \lambda(0) \gg 10^{60} \]

for practical situations, \( \lambda(n) \leq 4 \).

**Theorem:** Any sequence of \( m \) operations with \( n \) elements using the forest of trees data structure has worst-case running time \( \Theta(m \lambda(n)) \) and amortized running time \( \Theta(\lambda(n)) \).

just barely super-linear (effectively linear)

(effectively constant)

Proof in CLRS §21.4 \( \leftarrow \) You are not responsible for.