

Lecture 04

Union-Find and Amortization

L04-1
6.046
2/16/2017

Data Structure Analysis
 Technique

Motivating Problem

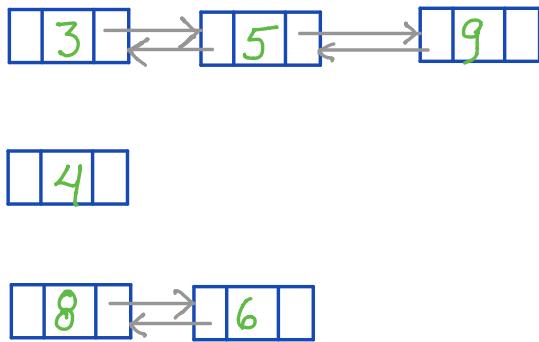
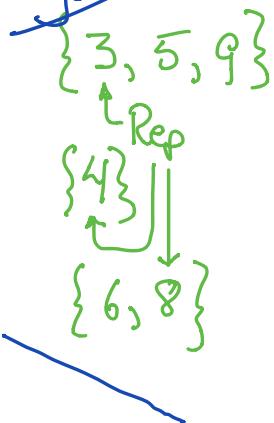
- maintain dynamic collection of pairwise disjoint sets, $S = \{S_1, S_2, \dots, S_r\}$ with one (arbitrary) representative element per set, $\text{Rep}[S_i]$
- support operations
 - $\text{MAKE-SET}(x)$: add set $\{x\}$ to collection with x as representative
 - $\text{FIND-SET}(x)$: returns representative of set $S(x)$ containing element x , $\text{Rep}[S(x)]$
 - $\text{UNION}(x, y)$: replaces sets $S(x), S(y)$ containing elements x, y with $S(x) \cup S(y)$, having single representative (if $S(x), S(y)$ distinct)

Simple Implementation:

LO4.2

- Each set is linked list
- Representative is head of list

e.g.



Running Time (Worst-case cost for n elements)

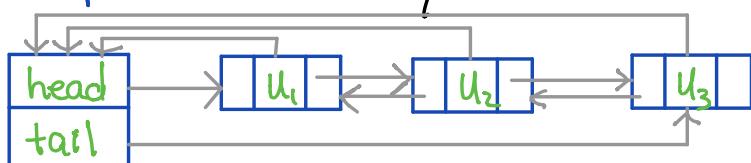
$\mathcal{O}(1)$ MAKE-SET(x): create single node containing x

$\mathcal{O}(n)$ FIND-SET(x): walk "left" from x to head
and return "left-most" node

$\mathcal{O}(n)$ UNION(x, y):

- walk to tail of list containing x and head of list containing y
- join lists with head-tail pointers
- note $\text{Rep}[S(x)]$ is Rep of new set

Candidate Improvement: Easy access to list head & tail



Now $\text{FIND-SET}(x)$: follow 2 pointers to head LO3.3
 $\text{UNION}(x,y)$:
 - find tail of $S(x)$ $O(1)$
 - find head of $S(y)$ $O(1)$
 - concatenate lists $O(1)$
 - update all pointers from $S(y)$ elements to new head $O(n)$

Adversary could ask: $\text{MAKE-SET}(0), \text{MAKE-SET}(1), \dots, \text{MAKE-SET}(n-1),$
 $\text{UNION}(1,0), \text{UNION}(2,0), \dots, \text{UNION}(i,0), \dots, \text{UNION}(n-1,0)$
 Takes $O(i)$

Total time: $O(n + \sum_{i=1}^{n-1} i) = O(n^2)$

due to $\Omega(n)$ UNION operations taking $\Omega(n)$ time each.

How might we improve this?

- always concatenate smaller into larger list
 - by maintaining length attribute for list
- - would make each $\text{UNION } O(1)$ in above example
 - but adversary could construct more balanced sets, and UNION of 2 sets of size $\Omega(n)$ still takes $\Omega(n)$ time.
 - so worst-case running time for UNION did not improve.

L04.4

But how common is it that a sequence of operations on n elements involves taking UNION of sets of size $\Omega(n)$?

Let $n = \text{total # of elements } (\text{MAKE-SET ops})$
 $m = \text{total # of operations } (m \geq n)$

Claim: Cost of all UNIONS = $O(n \log n)$
Total cost = $O(m + n \log n)$

Proof:

- focus on specific element u
- when created with MAKE-SET, length $[S(u)] = 1$
- whenever $S(u)$ merges with another set $S(v)$
 - if length $[S(v)] \geq \text{length}[S(u)]$
 - then
 - u 's head pointer must be updated cost
 - $\&$ length $[S(u)]$ at least doubles
 - else
 - u 's head pointer not updated no cost
 - length $[S(u)]$ increases

∴ Each time we pay cost to update u 's head pointer; $\text{length}[S(u)]$ at least doubles can not exceed n and never decreases
• Head pointer of u is updated $\leq \underline{\underline{\lg(n)}}$ times !!

So, total cost of all UNIONS is $\mathcal{O}(n \log n)$ L64.5
elements $\xrightarrow{\text{↑}} \frac{\text{max cost per element}}{\epsilon}$
total cost overall is $\mathcal{O}(m + n \log n)$

Because all m operations together take $\mathcal{O}(m + n \log n)$ time,
we say each is $\mathcal{O}(\log n)$ in some average sense.
We call this the Amortized Cost recall ($m \geq n$)

Amortized Analysis

Def: An operation has amortized cost $T(n)$
iff any k operations have cost $\leq k \cdot T(n)$

Several methods are used to bound amortized cost
of operations:

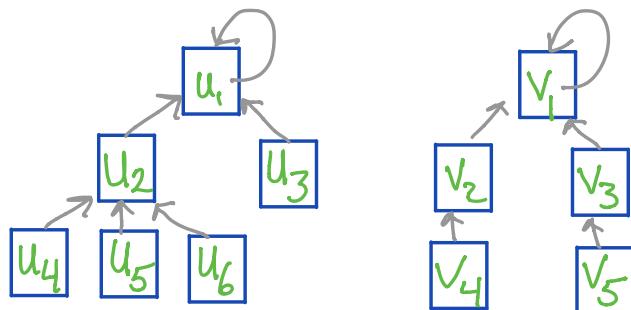
Aggregate Method • compute total cost of k ops
and divide by k (as above)

Accounting Method (recitation & book) • plan ahead by storing "credit"
today in small increments
to use to cover "expensive"
ops in future.

Potential Method • quantify karma! (later today)

UNION-FIND with forest of trees soln.

104.6



- each set is a tree
- possibly unbalanced
- not necessarily binary
- root is representative

$\text{MAKE-SET}(u)$: Initialize single node

$O(1)$

$\text{FIND-SET}(u)$: Walk up tree & return root

$O(\text{height}[S(u)])$

$\text{UNION}(u, v)$: $\bar{u} \leftarrow \text{FIND-SET}(u)$

$O(\text{height}[S(u)])$

$\bar{v} \leftarrow \text{FIND-SET}(v)$

$+ O(\text{height}[S(v)])$

$\bar{v}.\text{parent} \leftarrow \bar{u}$

Regrettably, we find the same worst-case behavior:

Per operation: $O(n)$

Per sequence of ops: $O(n^2)$

Consider: $\text{MAKE-SET}(0), \text{MAKE-SET}(1), \dots, \text{MAKE-SET}(n-1),$
 $\text{UNION}(1,0), \text{UNION}(2,0), \dots, \text{UNION}(i,0), \dots, \text{UNION}(n-1,0)$



Can we improve this solution?

LO4.7

Idea 1: Merge shorter trees into taller ones

- maintain height of tree at root

UNION(u, v):

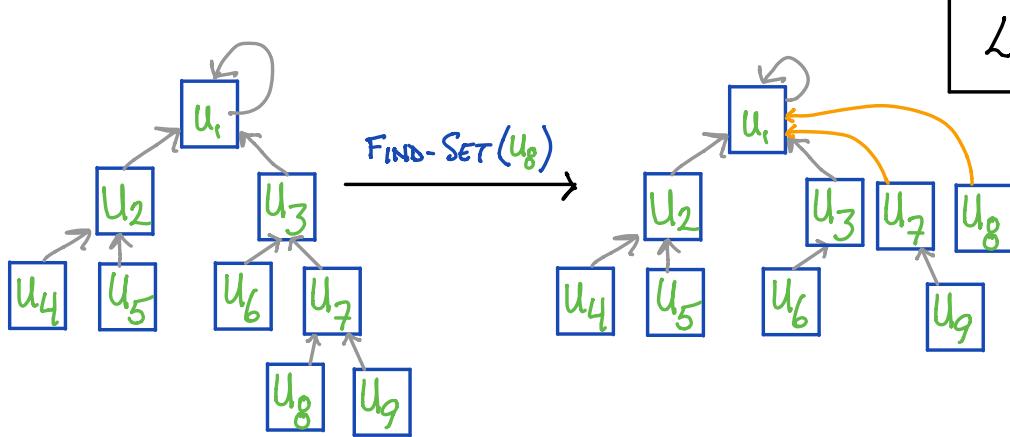
```
 $\bar{u} \leftarrow \text{FIND-SET}(u)$ 
 $\bar{v} \leftarrow \text{FIND-SET}(v)$ 
if  $\bar{u}.\text{rank} = \bar{v}.\text{rank}$ 
   $\bar{u}.\text{rank} \leftarrow \bar{u}.\text{rank} + 1$ 
   $\bar{v}.\text{parent} \leftarrow \bar{u}$ 
else if  $\bar{u}.\text{rank} > \bar{v}.\text{rank}$ 
   $\bar{v}.\text{parent} \leftarrow \bar{u}$ 
else
   $\bar{u}.\text{parent} \leftarrow \bar{v}$ 
```

Claim: Worst-case cost of each operation is $O(\log n)$
if at most n MAKE-SET operations

Proof: By induction, show rank of tree is
always $O(\log n)$.

Idea 2: Path Compression

- For each FIND-SET op, we not only walk up the tree to the root,
- We also re-direct the parental pointer of each node we visit on that walk to point to the root.
"Flattening the tree"



L04.8

Claim: Amortized cost of m operations, n of which are MAKE-SET, is $\mathcal{O}(\log n)$.

Proof: Using potential function method

- Define potential function ϕ mapping data-structure configuration \hat{c} \rightarrow non-negative integer $c + \Delta\phi$ $\begin{matrix} \text{corresponds} \\ \equiv \\ \text{to "potential energy"} \end{matrix}$
- Define make-believe-cost \equiv true cost + $\Delta\phi$

$$\sum \text{make-believe-cost}_S = \left(\sum \text{true costs} \right) + \phi(\text{final}) - \phi(\text{init.})$$

$\overbrace{\phi(\text{DS after op}) - \phi(\text{DS before})}$

Here we select $\phi(\text{DS}) = \sum_u (\log(u.\text{size}))$ size of subtree rooted at u

Check our conditions:

MAKE-SET: $\hat{c} = \underbrace{1}_{\text{true cost}} + \underbrace{0}_{\text{change in potential}}$

204.9

UNION: Cost of $2 \times \text{FIND-SETS}$ + Cost of linking trees

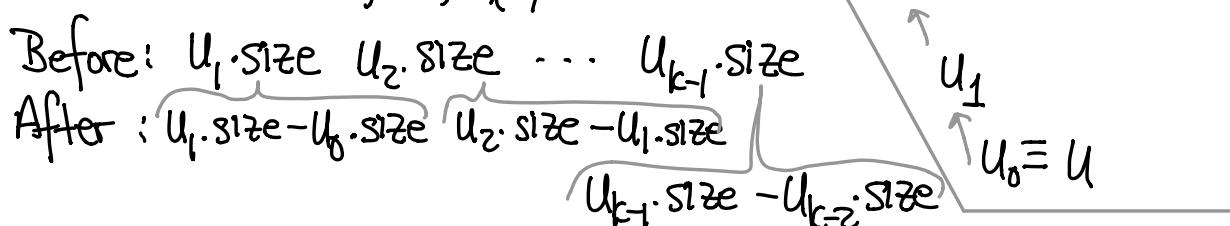
$$\hat{C}_{\text{linking}} \leq 1 + \log(\text{Rep}[S(u)].\text{size} + \text{Rep}[S(v)].\text{size})$$

only $\text{Rep}[S(u)].\text{size}$ changed

$$\leq O(\log n)$$

FIND-SET: Suppose traverse

- only nodes whose size is affected are u_1, \dots, u_{k-1}



$$\text{So, } \hat{C} \leq k + \phi(\text{DS after}) - \phi(\text{DS before})$$

$$= k + \sum_{i=1}^{k-1} \log(S_i - S_{i-1}) - \sum_{i=1}^{k-1} \log(S_i)$$

$$= 1 + \sum_{i=1}^{k-1} \left[1 + \log\left(\frac{S_i - S_{i-1}}{S_i}\right) \right]$$

Lemma: The function $1 + \log \frac{S_i - S_{i-1}}{S_i}$ is ≤ 1

and non-negative only if $S_i \geq 2S_{i-1}$, which can hold for at most $\lg n$ indices i .

Hence $\hat{C} \leq O(\log n)$.

So, all our operations satisfy $\hat{C} = O(\log n)$ LOG.10
 \Rightarrow Sum of make-believe-costs of m operations is
 $m O(\log n)$
 \Rightarrow Sum of true costs of m operations is
 $\leq m O(\log n) + \underbrace{\phi(\text{initial})}_{=0}$

So, IDEA 1 and IDEA 2, each applied alone
 produces $O(m \log n)$ algorithm

What happens, if we combine IDEA 1 and IDEA 2 ?

Claim: On a disjoint-set forest with union-by-rank
 and path compression, any sequence of m
 operations, n of which are MAKE-SET, has
 worst-case running time $O(m \alpha(n))$.

Inverse Ackerman function ↑
 (which grows extremely slowly: $\alpha(n < 10^{80}) \leq 4$)
 # atoms in the observable universe ↑

Proof: CLRS § 21.4

You are NOT responsible for this proof.