Lecture 04 Union-Find and Amortization	L04.1 6.046 2/13/2020

Data Structure Analysis Technique

Reading: CLRS Chpts 17 & 21

Motivating Problem

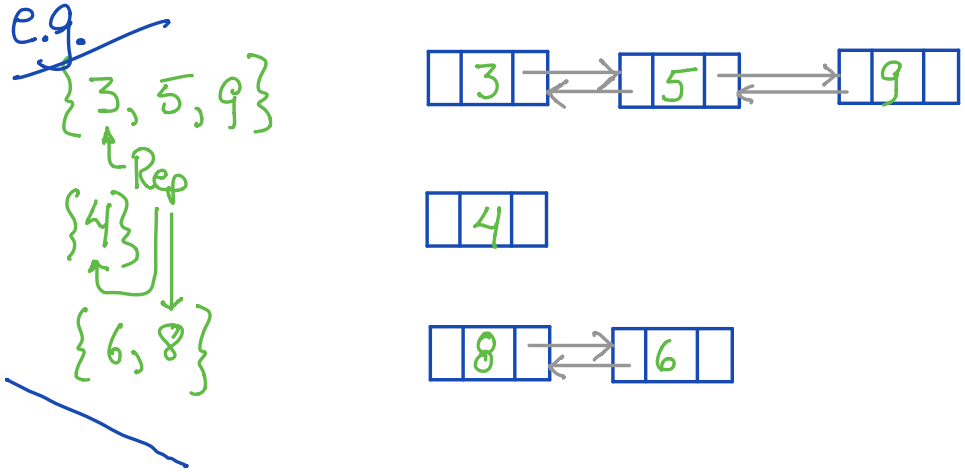
- maintain dynamic collection of pairwise disjoint sets $\mathcal{S} = \{S_1, S_2, \ldots, S_r\}$ with one (arbitrary) representative element per set, $Rep[S_i]$

- support operations
  - MAKE-SET(x): add set $\{x\}$ to collection with x as representative
  - FIND-SET(x): returns representative of set $S(x)$ containing element x, $Rep[S(x)]$
  - 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:

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

e.g.



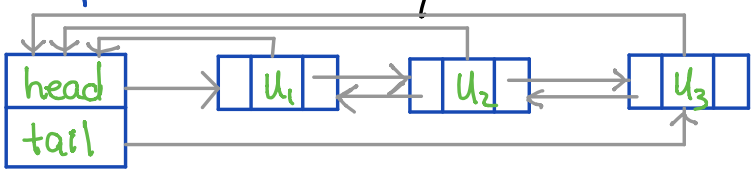
Running Time (Worst-case cost for n elements)

$O(1)$ MAKE-SET(x): create single node containing x

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

$O(n)$ UNION(x, y):
- walk to tail of list containing x and head of list containing y
- join lists with head-tail pointers
- note Rep[S(x)] is Rep of new set

Candidate Improvement: Easy access to list head&tail



Now O(1) FIND-SET(x): follow 2 pointers to head

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: MAKE-SET(0), MAKE-SET(1), ..., MAKE-SET(n-1),
UNION(1,0), UNION(2,0), ..., UNION(i,0), ..., 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 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.

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

Let n = total # of elements (MAKE-SET ops)
m = 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$[S(v)] \geq$ 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, length$[S(u)]$ at least doubles (can not exceed n and never decreases)
- Head pointer of u is updated $\leq \lg(n)$ times !!

So, total cost of all UNIONs is $O(n \log n)$ | L04.5

elements ↑ ↳ max cost per element

total cost overall is $O(m + n \log n)$

Because all m operations together take $O(m + n \log n)$ time, we say each is $O(\log n)$ in some average sense. ← recall $(m \geq n)$

We call this the **Amortized Cost**

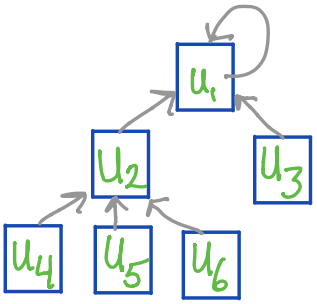
Amortized Analysis

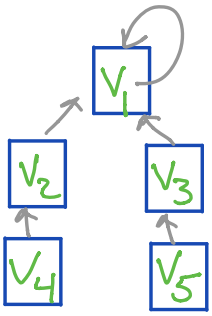
Def: An operation has amortized cost $T(n)$ iff any k operations have cost $\leq k \cdot T(n)$

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.





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

MAKE-SET(u): initialize single node — $O(1)$

FIND-SET(u): walk up tree & return root — $O(\text{height}[S(u)])$

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

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

Regrettably, we find the same worst-case behavior:

Per operation: $O(n)$

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

Consider: MAKE-SET(0), MAKE-SET(1), ..., MAKE-SET(n-1),
UNION(1,0), UNION(2,0), ..., UNION(i,0), ... UNION(n-1,0)

Can we *improve* this solution?

Idea 1: Merge shorter trees into taller ones
- maintain height of tree at root

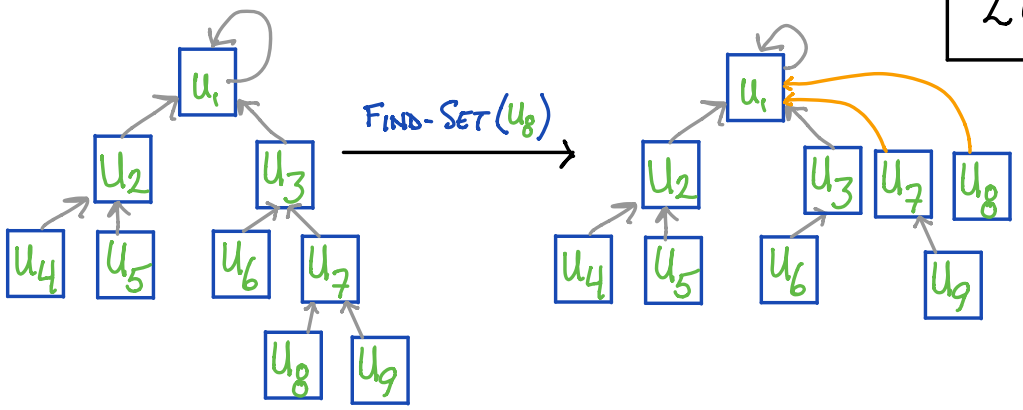
```
UNION(u,v):  ū ← FIND-SET(u)
             v̄ ← FIND-SET(v)
             if ū.rank = v̄.rank
                ū.rank ← ū.rank + 1
                v̄.parent ← ū
             else if ū.rank > v̄.rank
                v̄.parent ← ū
             else
                ū.parent ← v̄
```

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"



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

Proof: Using potential function method

- Define potential function ϕ mapping data-structure configuration $\rightarrow$ non-negative integer ← corresponds to "potential energy"

$$\hat{c} \equiv c + \Delta\phi$$

- Define make-believe-cost $\equiv$ true cost + $\Delta\phi$

$\Delta\phi$ = ϕ(DS after op) − ϕ(DS before)

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

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

Check our conditions:

MAKE-SET: $\hat{c} = 1 + 0$

1 = true cost; 0 = change in potential

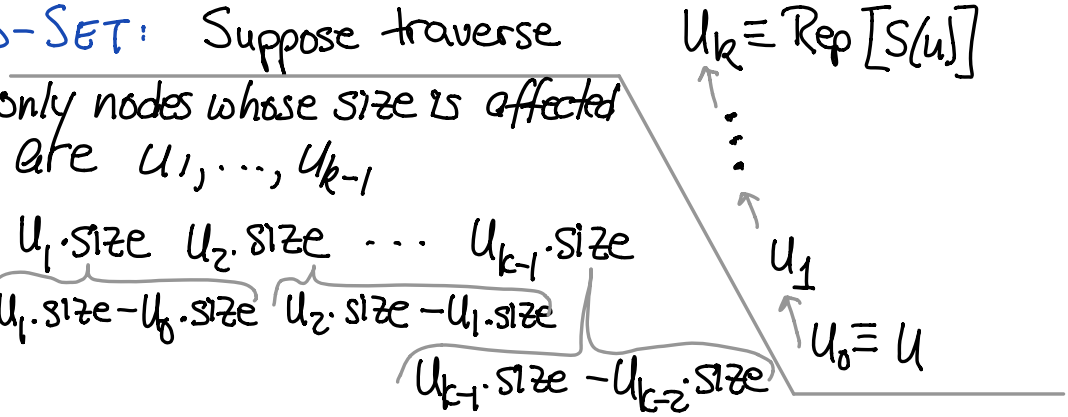
UNION: Cost of 2x FIND-SETs + Cost of linking trees

$\hat{C}_{linking} = 1 + \lg(Rep[S(u)].size + Rep[S(v)].size) - \lg(Rep[S(u)].size)$

only Rep[S(u)].size changed

$\leq O(\log n)$

FIND-SET: Suppose traverse



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

Before: $u_1.size$ $u_2.size$ $\cdots$ $u_{k-1}.size$

After: $u_1.size - u_0.size$ $u_2.size - u_1.size$ $u_{k-1}.size - u_{k-2}.size$

So, $\hat{c} = k + \phi(DS\ after) - \phi(DS\ before)$

$$= k + \sum_{i=1}^{k-1} \lg(s_i - s_{i-1}) - \sum_{i=1}^{k-1} \lg(s_i)$$

$$= 1 + \sum_{i=1}^{k-1} \left[1 + \lg\left(\frac{s_i - s_{i-1}}{s_i}\right)\right]$$

Lemma: The function $\left[1 + \lg \frac{s_i - s_{i-1}}{s_i}\right]$ is 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 $\hat{c} = O(\log n)$

$\Rightarrow$ 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.