EECS 311: Data Structure and Data Management Lecture 25, 26
Union-Find

Union-Find (Cont.)

Analysis of union-by-rank with path-compression

Algorithm: find(i) with path compression

1. if array[i] ≤ 0, return i.
2. array[i] = find(array(i))
3. return array[i].

Def: the *rank* of a node, is its height without path-compression.

Note: store *negative rank* at root nodes.

Algorithm: union(i, j) by rank

1. i' = find(i).
2. j' = find(j).
3. if -array[i'] > -array[j'], array[j'] = i'.
4. if -array[i'] $\geq$ -array[j'], array[i'] = j'.
5. if -array[i'] = -array[j'], array[j']++.

Recall Def: $\log^* n$ = the number of logs you can take of n before you're ≤ 1.

Example:

$\log^* n$	0	1	2	3	4	5
n	1	2	3–4	5–16	17–65536	65537–2^{65536}

Claim: with path-compression and union-by-rank m union and find operations from creation costs $O(m \log^* n)$

Claim 0: # nodes with rank ≥ 1 (n') $\leq$ # of unions (m').

Proof: at most one node changes rank in union. induction.

Claim 1: node's rank $\leq$ parent's rank -1

Proof:

- without path compression,
  1. rank = height.
  2. node height $\leq$ parent height $- 1$.
- with path compression, node only gets higher ranked parent.

□

Claim 2: a rank r node has at least 2^r descendents.

Proof: (induction on rank)

base case (rank 0): true.

inductive hypothesis (rank r) : claim holds for r.

inductive step (rank $r + 1$):

- to get rank $r + 1$, union two rank r nodes.
- I.H. implies both have $\geq 2^r$ decendents.

- total decendents $\geq 2^{r+1}$.

□

Claim 3: at most $n'/2^r$ nodes of rank $r \geq 1$.

Proof:

- Claim 1
  $\Rightarrow$ ranks strictly increasing
  $\Rightarrow$ rank r nodes have no common descendents.
- Claim 0 & Claim 2
  $\Rightarrow$ at most $n'/2^r$ nodes of rank r.

□

Amortized Analysis of Union-Find

Def: rank-ceiling(r) = $\max\{r' : log^* r' = \log^* r\}$.

Note: rank-ceiling$(r) \leq 2^r$.

Accounting Method

- keep central account, and account at each non-root node.
- on find(i):
  - get paid $\$ \log^* n + 2$ (pocket).
  - must pay \$1 for each node j on path from i to root,
    - if j is root, j's parent is root, pay \$1 from pocket. (happens twice)
    - else if $\log^*(j$'s rank$) = \log^*(j$'s parent's rank), (*) withdraw \$1 from node acct to pay.
    - else if $\log^*(j$'s rank$) < \log^*(j$'s parent's rank), pay \$1 from pocket. (happens at most $\log^* n$ times)
- on union(i, j): (make j' child i')
  - get paid $3 \log^* n + 4$:
    - pay $\$2 \log^* n + 4$ find of i and j.
    - deposit $\$ \log^* n$ in central account.
  - (**) withdraw j's ceiling-rank(j's rank) from central account and deposit in j's account.

Example: if rank $j = 25$, deposit 35536 in j's account.

Analysis

"no overdrawn accounts"

Claim: no overdraw from node acct's at (*)

Proof:

- how many times can node of rank r be charged?
- each time a node is charged, it gets a parent of higher rank.
- can happen at most ceiling-rank(r) times.

□

Claim: no overdraw from central acct at (**)

Proof:

- let $k = \log^* n$,
  let $r_0, \ldots, r_k$ be {1,2,4,16,35536,...}
- total deposits: $m' \log^* n$.

- # nodes with rank $\geq r+1 \leq n'/2^r$ ($\leq \sum_{i=r+1}^{\infty} n'/2^i = n'/2^r$)
- withdrawals for ranks $\{r_i+1, \ldots, 2^r{}_j\} \leq$ ceiling-rank$(r)n'/2^r \leq n'$.
- total withdrawals: $\leq \sum_{j=0}^{k} n' = n'k \leq m' \log^* n$.

□