COMPSCI 330 Lecture 15 Disjoint sets

Monday, October 24, 2016 3:00 PM

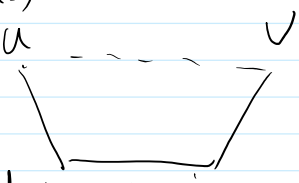
- Recall: Kruskal's algorithm
- Disjoint sets using Arrays
- Disjoint sets using Trees

- Kruskal

- two operations

1. union: merging two sets (connected components)

2. find. check which set an element is in.



find(u) = find(v) iff u and v are in same set.

- union-find using arrays
  - idea: for each set, maintain an array for its elements.
  - initially: have n sets, n arrays

[1], [2], [3], ..., [n]

union(3,5) [3,5]

union(3,6) [1,3,5] [2,6,7]

[1,3,5,2,6,7]

for each element, maintain a pointer to the array it is in

- running time: find: O(1) time
  union: O(n) time

union(1,2) union(2,3) union(3,4) ... union(n-1,n)

$$\text{total running time} = \sum_{k=1}^{n-1} (k+1) = \frac{n(n+1)}{2} - 1$$

[1,2,...,k] [k+1]

improve: should just append k+1 to the end of first array.

- new union algorithm

```
union (a, b)
    if size(find(a)) > size(find(b))
        append find(b) to find(a)
```

else append find(a) to find(b)

[2, 3, 5] [6, 8, 9, 11]

size 3 size = 4

[6, 8, 9, 11, 2, 3, 5]

- analyze running time for new algorithm

find : $O(1)$ time

union : time = size of the smaller set

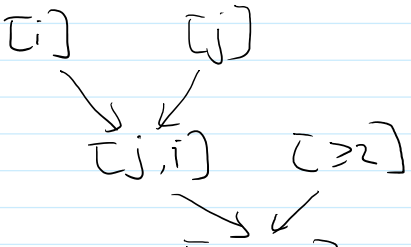
$\Theta(n)$ in the worst case $[1, 2, \cdots, \frac{n}{2}]$ $[\frac{n}{2}+1, \cdots, n]$

- amortized analysis: not all union operations can take $\Theta(n)$ time.

Claim: If element i has moved k times, then find(i) is a set of size at least 2^k.

Proof: by induction. clearly this is true when $k=0$.

if this is true for k, then at k+1-th time when i is moving



[i] [ ]

$\geq 2^k$ (by induction hypothesis) $\geq 2^k$ (by algorithm) □

Claim $\Rightarrow$ no element can move more than $\lceil \log_2 n \rceil$ times.

$\sum$ union cost $\leq \sum_{\text{elements } i}$ # times that i moved

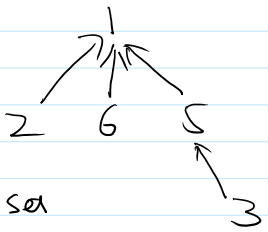
$\leq \sum_{i=1}^{n} \lceil \log_2 n \rceil = O(n \log n)$

- union find by trees

{1, 2, 3, 5, 6} {4}

- sets $\Leftrightarrow$ trees
- store parent for each element

	1	2	3	4	5	6
parent	1	1	5	4	1	1



- root of tree : index for the set

find(u) : return root of the tree

if parent(u) = u

return u

else return find(parent(u))

time = height of the tree.

else return Find(parent(u))

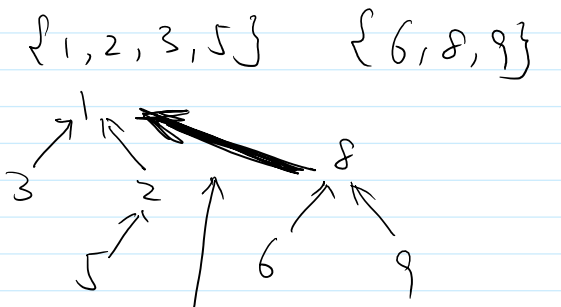
time = height of the tree.

- union (5, 9)

  find(5): 1

  find(9): 8

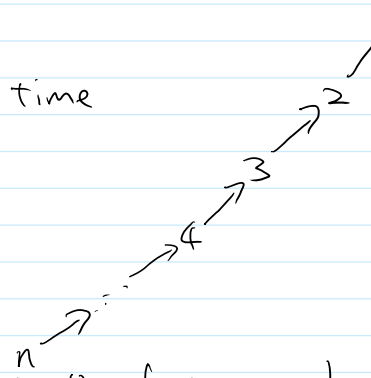
  parent(8) = 1



only need to add this edge.

- running time (worst case)

  find(n) takes $\Theta(n)$ time



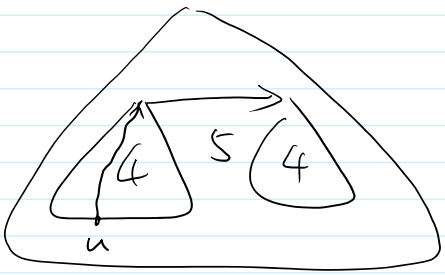
- ideal: want to minimize depth of tree when merging

  "union-by-rank"

  maintain rank for each tree

  when merging, always connect root with lower rank to root with higher rank.

  rank of new tree = { max rank if ranks were different; rank+1 if ranks were the same



intuition: rank = depth of the tree

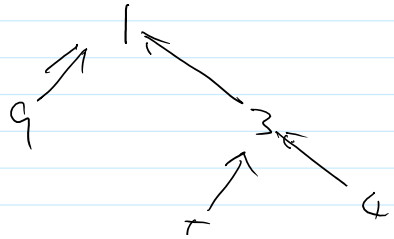
Claim: A tree of rank k has at least 2^k nodes.

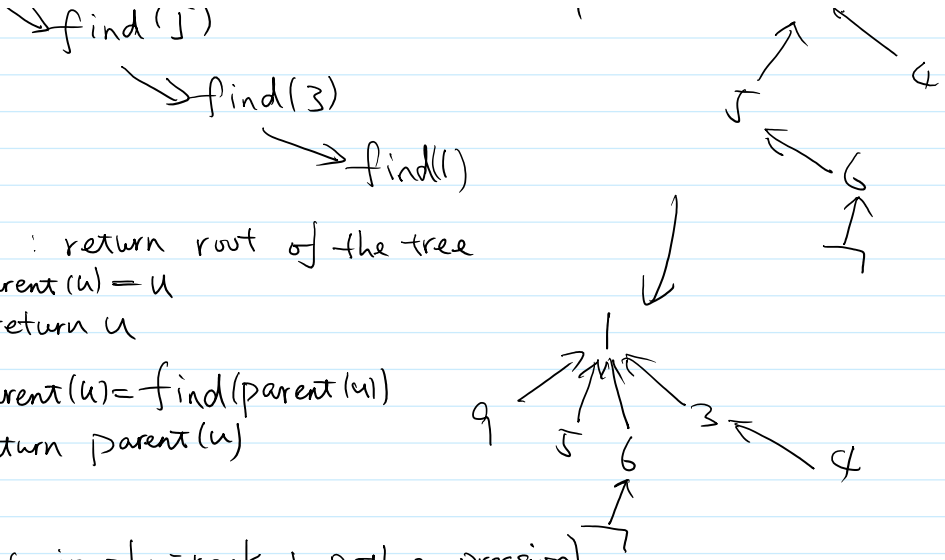
- idea 2: path compression

  find(6)

  → find(5)

  → find(2)





find(u) : return root of the tree
if parent(u) = u
return u
else
parent(u) = find(parent(u))
return parent(u)

- running time (union-by-rank + path compression)

for t union/find operations

running time = $O(t\,\alpha(n))$

- what is $\alpha(n)$? α: inverse Ackerman function

$\alpha(n) = 0$ if $\log^* n$

$\log^* \underbrace{2^{2^{2^{\cdot^{\cdot^{\cdot}}}}}}_{k \text{ layers}} = k$

for any reasonable number $\alpha(n) \leq 5$