CS3383 Unit 2.4: Union Find Path Compression

David Bremner

February 26, 2024

Outline

Union Find
- Path Compression
- Path Compression Analysis

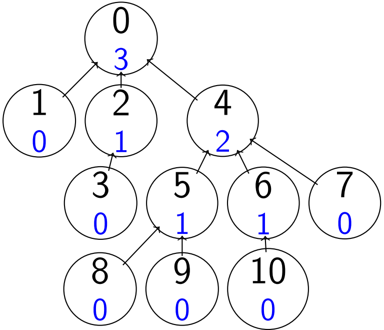
"Memoizing" the find routine

```
def find(P, key):
  while P.parent[key] != key:
    key = P.parent[key]
  return key
```

```
def find(P, key):
  if P.parent[key] != key:
    P.parent[key] = \
      P.find(P.parent[key])
  return P.parent[key]
```

"Memoizing" the find routine

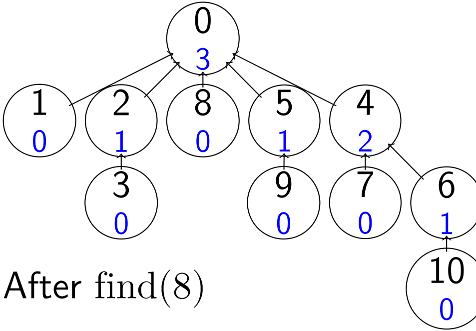
```
def find(P, key):
  if P.parent[key] != key:
    P.parent[key] = \
      P.find(P.parent[key])
  return P.parent[key]
```



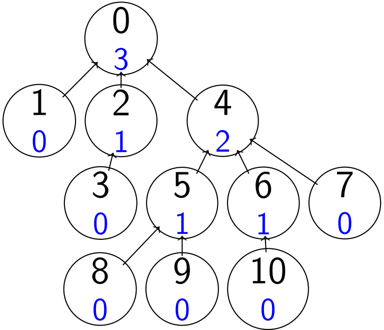
before

“Memoizing” the find routine

```
def find(P, key):
  if P.parent[key] != key:
    P.parent[key] = \
      P.find(P.parent[key])
  return P.parent[key]
```

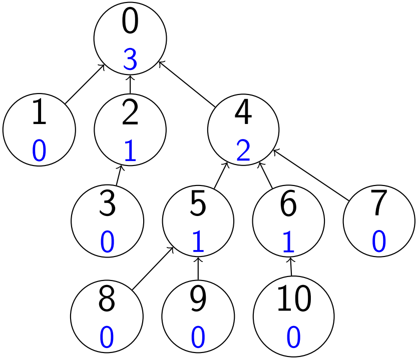


After find(8)

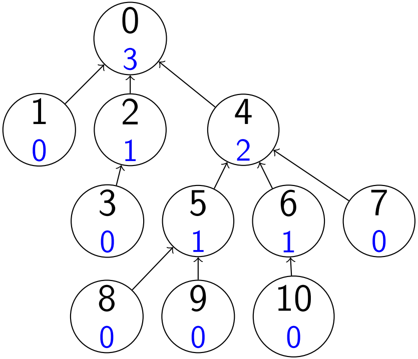


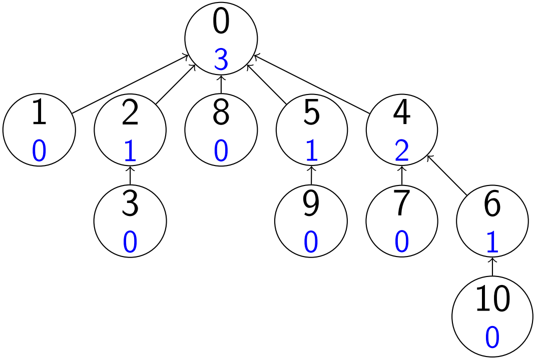
before

Find example



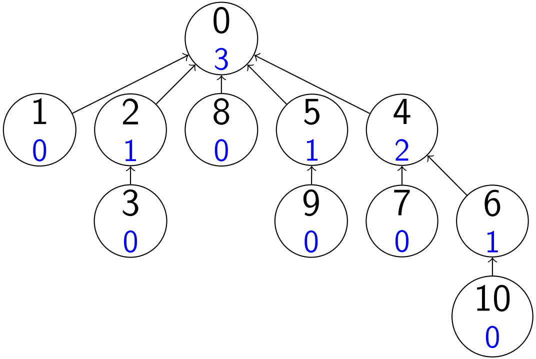
Find example



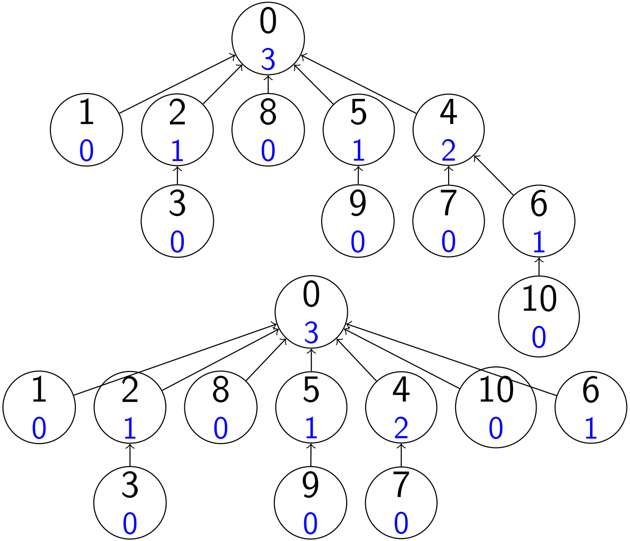


After find(8)

find(8), find(10)



find(8), find(10)



Rank ordering is maintained

Property 1

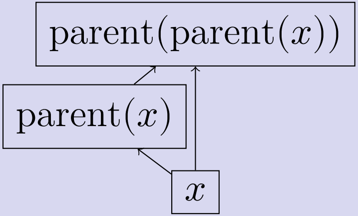
For any x such that $\mathrm{parent}(x) \neq x$,
$\mathrm{rank}(x) < \mathrm{rank}(\mathrm{parent}(x))$

Rank ordering is maintained

Property 1

For any x such that $\mathrm{parent}(x) \neq x$,
$\mathrm{rank}(x) < \mathrm{rank}(\mathrm{parent}(x))$

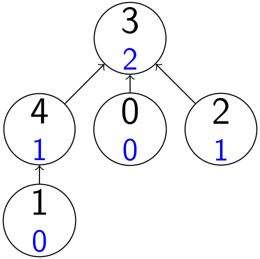
Shortcuts preserve order



Size of trees is preserved, but not subtrees.

Property 2'

Any root node of rank k has at least 2^k nodes in its subtree.



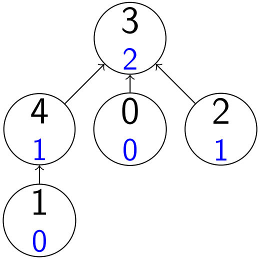
Size of trees is preserved, but not subtrees.

Property 2'

Any root node of rank k has at least 2^k nodes in its subtree.

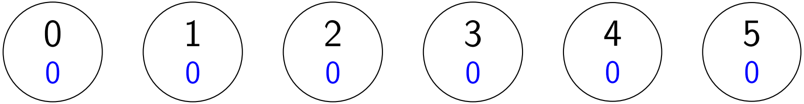
Proof of property 2'.

Induction: Base case is $k = 0$. Roots of rank k are made from two rank $k - 1$ roots. □



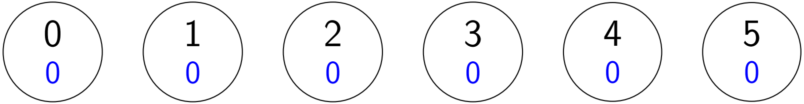
Union+Find Example 1/

- initial partition

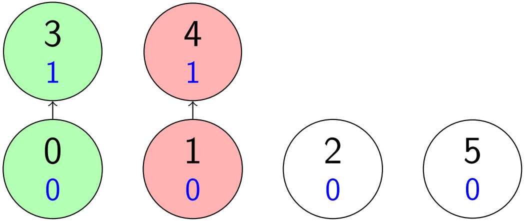


Union+Find Example 1/

- initial partition

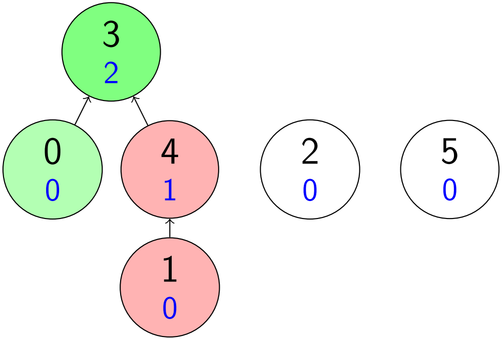


- after union(0,3), union(1,4)

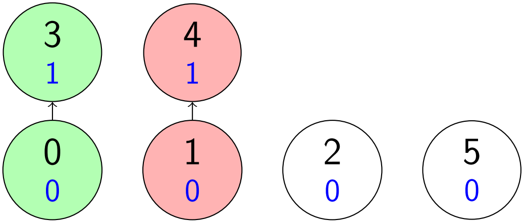


Union+Find Example 2/

after union(4,0)

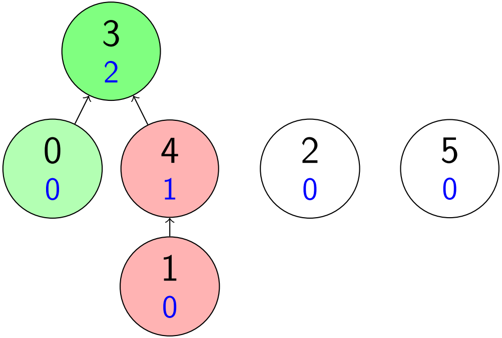


after union(0,3), union(1,4)

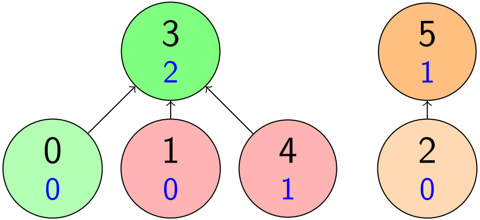


Union+Find Example 3/

after union(4,0)

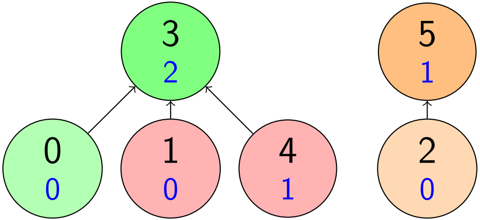


after union(4,0), find(1), union(2,5)

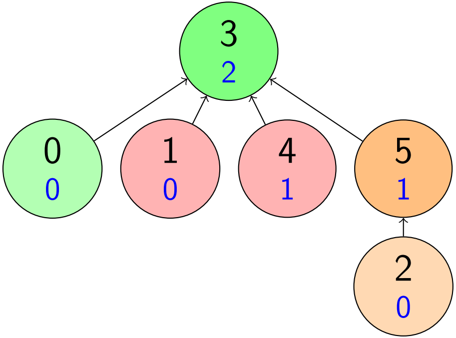


Union+Find Example 4/

after union(4,0), find(1), union(2,5)

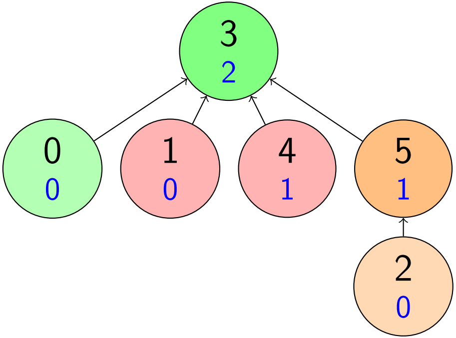


after union(5,0)

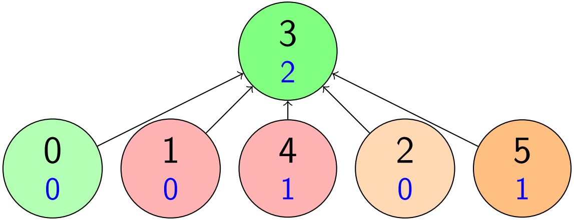


Union+Find Example 5/

after union(5,0)



after union(5,0), find(2)



Not too many nodes of rank k

Property 3

If there are n elements, there are at most $\lfloor n/2^k \rfloor$ nodes of rank k.

Not too many nodes of rank k

Property 3

If there are n elements, there are at most $\lfloor n/2^k \rfloor$ nodes of rank k.

- From property 1, descendents of a given rank k node are distinct.

Not too many nodes of rank k

Property 3

If there are n elements, there are at most $\lfloor n/2^k \rfloor$ nodes of rank k.

- From property 1, descendents of a given rank k node are distinct.
- When a node gets rank $k > 0$, it is a root, and has 2^k descendents.

Not too many nodes of rank k

Property 3

If there are n elements, there are at most $\lfloor n/2^k \rfloor$ nodes of rank k.

- From property 1, descendents of a given rank k node are distinct.
- When a node gets rank $k > 0$, it is a root, and has 2^k descendents.
- Those descendents are never used to make another node rank k. (non-roots stay non-roots).

Rank intervals

- We divide the numbers $[1, n]$ into $[k+1, 2^k]$

$$[1,1],[2,2],[3,4],[5,16],\ldots,[k+1,2^k]$$

Rank intervals

- We divide the numbers $[1, n]$ into $[k+1, 2^k]$

$$[1,1],[2,2],[3,4],[5,16],\ldots,[k+1,2^k]$$

- The first p intervals cover

$$2^{2^{2^{\cdots 2}}}\Big\}p-1 \text{ times}$$

Rank intervals

- We divide the numbers $[1, n]$ into $[k+1, 2^k]$

$$[1,1],[2,2],[3,4],[5,16],\ldots,[k+1,2^k]$$

- The first p intervals cover

$$2^{2^{2^{\cdots 2}}}\Big\}p-1 \text{ times}$$

- $\log^*(n)+1$ intervals cover n

$$\log^*(n) = \begin{cases} 1 & \text{if } \log(n) \leq 1 \\ 1+\log^*(\log(n)) & \text{otherwise} \end{cases}$$

Bounding disbursements 1/2

- Each node in an interval ending in 2^k gets 2^k dollars.

Bounding disbursements 1/2

- Each node in an interval ending in 2^k gets 2^k dollars.
- By property 3, the total number of nodes in such an interval is at most

$$\frac{n}{2^{k+1}} + \frac{n}{2^{k+2}} + \frac{n}{2^{k+3}} + \dots \frac{n}{2^{2^k}}$$

Bounding disbursements 1/2

- Each node in an interval ending in 2^k gets 2^k dollars.
- By property 3, the total number of nodes in such an interval is at most
$$\frac{n}{2^{k+1}} + \frac{n}{2^{k+2}} + \frac{n}{2^{k+3}} + \dots \frac{n}{2^{2^k}}$$
- We need to bound
$$\sum_{i=k+1}^{2^k} 2^{-i}$$

Bounding disbursements 1/2

- Each node in an interval ending in 2^k gets 2^k dollars.
- By property 3, the total number of nodes in such an interval is at most
$$\frac{n}{2^{k+1}} + \frac{n}{2^{k+2}} + \frac{n}{2^{k+3}} + \dots \frac{n}{2^{2^k}}$$
- We need to bound
$$\sum_{i=k+1}^{2^k} 2^{-i} = \frac{1}{2^{k+1}} \sum_{i=0}^{2^k-k-1} 2^{-i}$$

Bounding disbursements 1/2

- Each node in an interval ending in 2^k gets 2^k dollars.
- By property 3, the total number of nodes in such an interval is at most

$$\frac{n}{2^{k+1}} + \frac{n}{2^{k+2}} + \frac{n}{2^{k+3}} + \dots \frac{n}{2^{2^k}}$$

- We need to bound

$$\sum_{i=k+1}^{2^k} 2^{-i} = \frac{1}{2^{k+1}} \sum_{i=0}^{2^k-k-1} 2^{-i}$$

$$\leq \frac{1}{2^{k+1}} \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i$$

Bounding disbursements 2/2

- We need to bound

$$\sum_{i=k+1}^{2^k} 2^{-i} = \frac{1}{2^{k+1}} \sum_{i=0}^{2^k-k-1} 2^{-i}$$
$$\leq \frac{1}{2^{k+1}} \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i$$

- each interval get at most n dollars in total
- $n(\log^* n + 1)$ dollars over all intervals.

Bounding disbursements 2/2

- We need to bound

$$\sum_{i=k+1}^{2^k} 2^{-i} = \frac{1}{2^{k+1}} \sum_{i=0}^{2^k-k-1} 2^{-i}$$
$$\leq \frac{1}{2^{k+1}} \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i$$
$$= \frac{1}{2^{k+1}} \frac{1}{1-1/2} \qquad \text{G.S.}$$

- each interval get at most n dollars in total
- $n(\log^* n + 1)$ dollars over all intervals.

Paying for find operations 1/2

```
def find(P, key):
  if P.parent[key] != key:
    P.parent[key] = \
        P.find(P.parent[key])
  return P.parent[key]
```

- Either $\mathrm{rank}(\mathrm{parent}\,[\text{key}])$ is in a later interval than $\mathrm{rank}[\text{key}]$ or not.

- every call does an update
- work $O(\#\text{updates}$

Paying for find operations 1/2

```
def find(P, key):
  if P.parent[key] != key:
    P.parent[key] = \
        P.find(P.parent[key])
  return P.parent[key]
```

- Either $\mathrm{rank}(\mathrm{parent}\,[\text{key}])$ is in a later interval than $\mathrm{rank}[\text{key}]$ or not.
- Increasing intervals can happen at most $\log^* n$ times.

- every call does an update
- work $O(\#\text{update}$

Paying for find operations 1/2

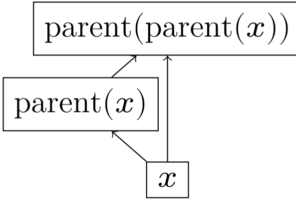
```
def find(P, key):
  if P.parent[key] != key:
    P.parent[key] = \
        P.find(P.parent[key])
  return P.parent[key]
```

- Either $\mathrm{rank}(\mathrm{parent}\,[\text{key}])$ is in a later interval than $\mathrm{rank}[\text{key}]$ or not.
- Increasing intervals can happen at most $\log^* n$ times.
- If in the same interval, we say key pays a dollar back.

- every call does an update
- work $O(\#\text{updates}$

Paying for find operations 2/2

If $\mathrm{rank}(\mathrm{parent}\,[\mathsf{key}])$ is in the interval as $\mathrm{rank}(\mathsf{key})$, we say key pays a dollar back.

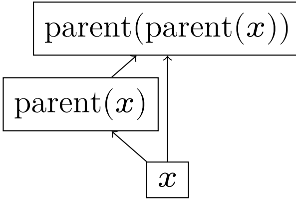


No node goes broke

- Each time x pays a dollar, it increases the rank of its parent.

Paying for find operations 2/2

If $\mathrm{rank}(\mathrm{parent}\,[\text{key}])$ is in the interval as $\mathrm{rank}(\text{key})$, we say key pays a dollar back.

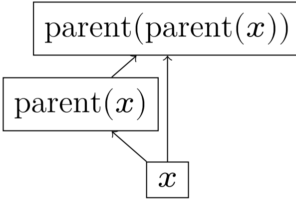


No node goes broke

- Each time x pays a dollar, it increases the rank of its parent.
- If $\mathrm{rank}(x) \in [k+1 \ldots 2^k]$, that can repeat less than 2^k times before its parent is in a higher interval.

Paying for find operations 2/2

If $\mathrm{rank}(\mathrm{parent}\,[\text{key}])$ is in the interval as $\mathrm{rank}(\text{key})$, we say key pays a dollar back.



No node goes broke

- Each time x pays a dollar, it increases the rank of its parent.
- If $\mathrm{rank}(x) \in [k+1 \ldots 2^k]$, that can repeat less than 2^k times before its parent is in a higher interval.
- Once that happens, payments stop.

Summing up

- We can think about the analysis as classifying all of the updates to a given key as "near" or "far", and bounding those in two different ways.
- Total cost for n operations
  - $\leq n \log^* n$ total steps where parent is in next interval
  - $\leq n \log^* n$ total steps where parent is in same interval
- Amortized cost in $O(\log^* n)$ per operation.