CS3383 Unit 2.4: Union Find Path Compression

David Bremner

February 20, 2024

Union Find

- Path Compression
- Path Compression Analysis

Motivation

Using union-find in Kruskal's Algorithm

- For unbounded edge weights, the sorting costs $\Omega(|E| \log |E|) = \Omega(|E| \log |V|)$
  - Naive union-find is fast enough.
- For small edge weights (e.g. weights bounded by $|E|$), sorting is no longer the bottleneck.

Amortized analysis

- It's hard to do find faster than $O(\log n)$ in the worst case
- We can make the *average* cost of all find operations in one run of a program almost constant
- This kind of average cost analysis is called amortized analysis
- Like with randomized algorithms, the algorithms are simple, but the analysis is a bit subtle.

[redacted]

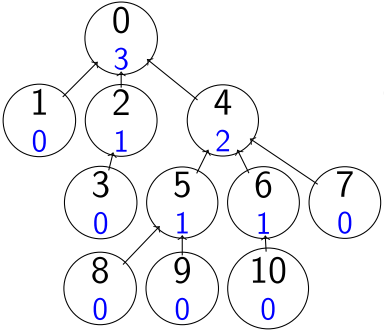
“Memoizing” the find routine

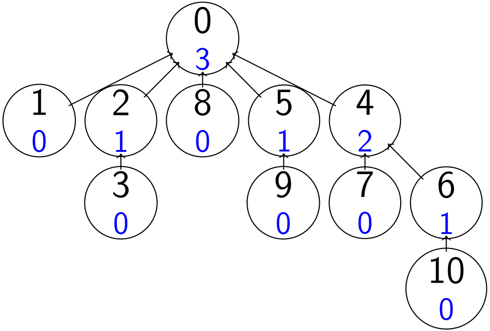
```
def find(P, key):
  while P.parent[key] != key:
    key = P.parent[key]
  return key
```

```
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

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

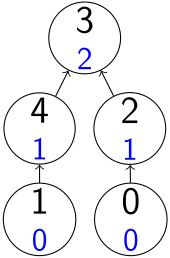


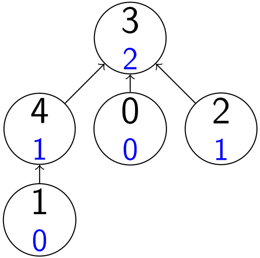


After find(8)

Strong Memoization

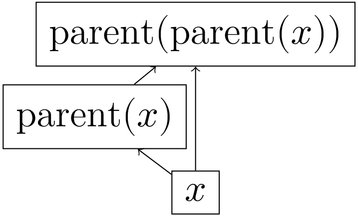
- not only only repeating the same query will be fast, but also any node on the path to the root.





- After $\text{find}(0)$
- Notice ranks look a bit odd, but still increase.

Rank ordering is maintained



Property 1

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

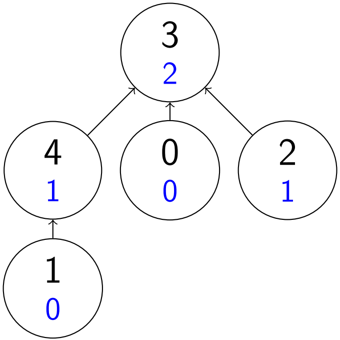
Size of trees is preserved, but not subtrees.

Property 2

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

Property 2'

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



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.

- 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.

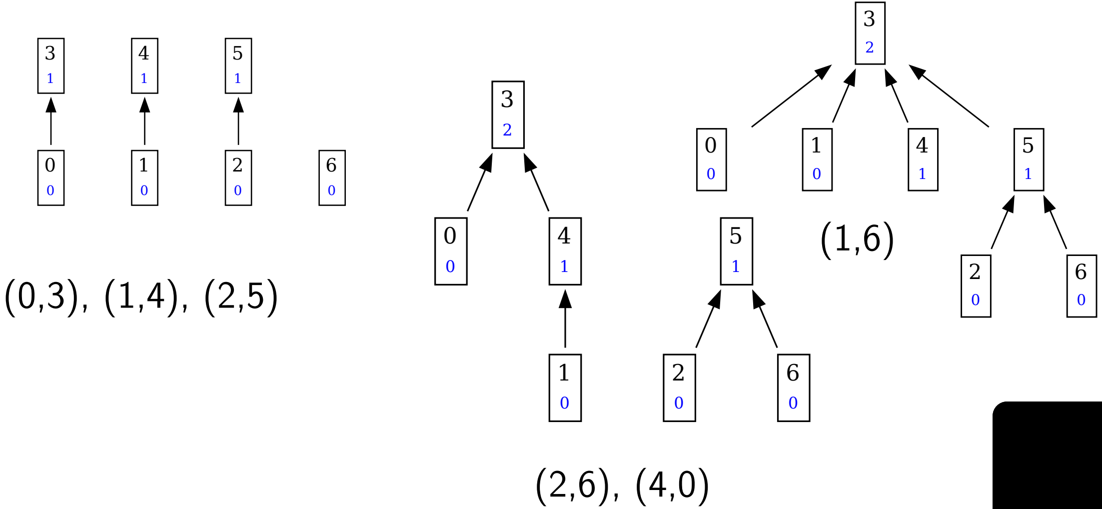
Not too many nodes of rank k

Property 3

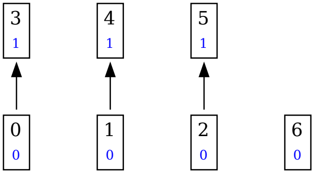
If there are n elements, there are at most $\lfloor n/2^k \rfloor$ nodes of rank k.

- 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.

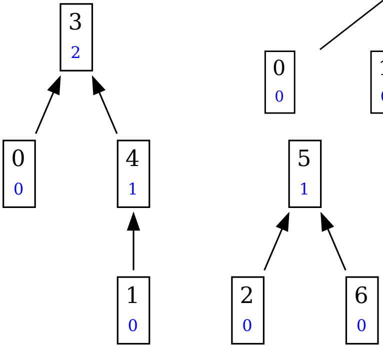
Path compression example



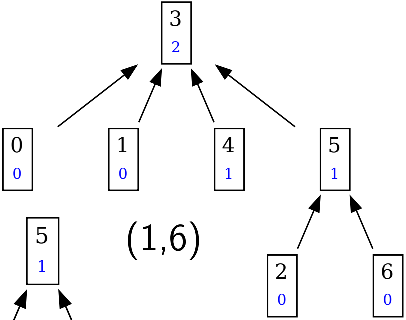
Path compression example



(0,3), (1,4), (2,5)



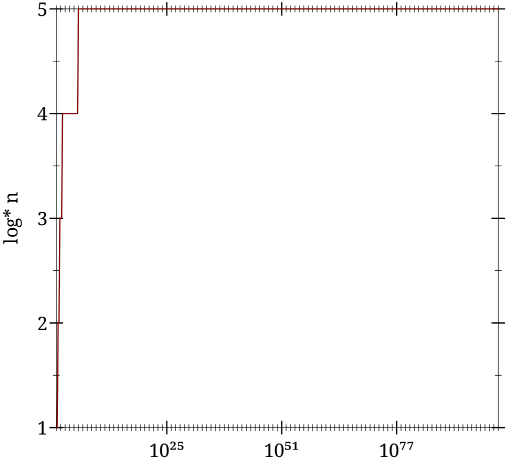
(2,6), (4,0)



(1,6)

$\log^* n$

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



Amortization

- We will keep track of (some) operations by counting them locally at every node.
- In order to "pay" for future operations, we give every node 2^k "dollars" if its max rank is in

$$[k+1, \ldots 2^k]$$

  for some $k = 2^j$.
- We will count the total amount of money passed out
- And argue that no node runs out of money.

Paying for find operations

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

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

Summing up

- 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.