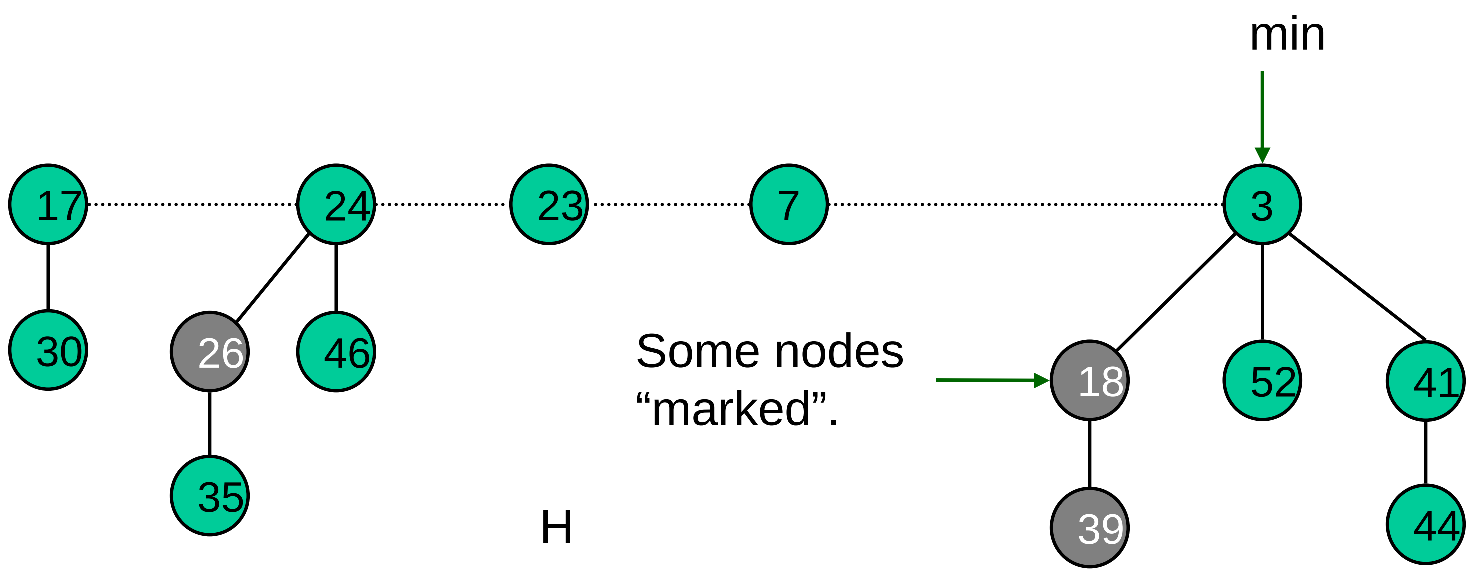
Fibonacci Heaps

Data Structures and Algorithms

Andrei Bulatov

Structure

Set of min-heap ordered trees



Implementation

Each node has 4 pointers: parent, 1st child, next & previous siblings.

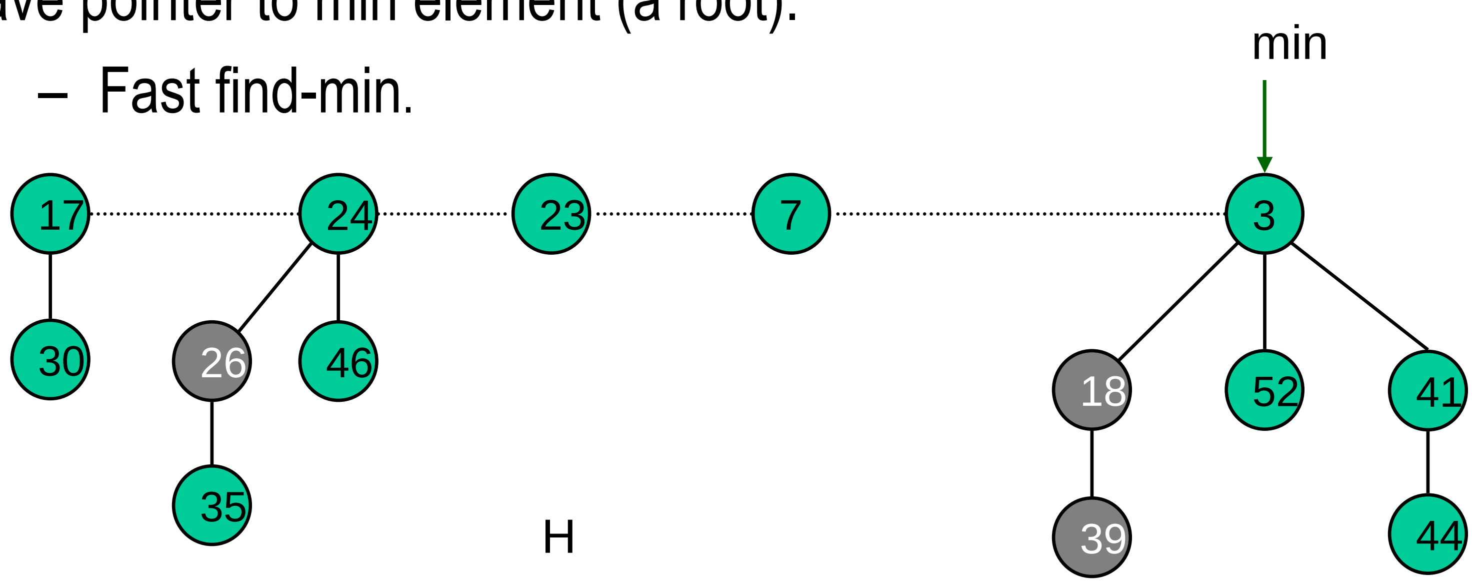
- Sibling pointers form circular, doubly-linked list.
- Can quickly splice off subtrees.

Roots in circular, doubly-linked list.

- Fast union.

Have pointer to min element (a root).

- Fast find-min.



Implementation

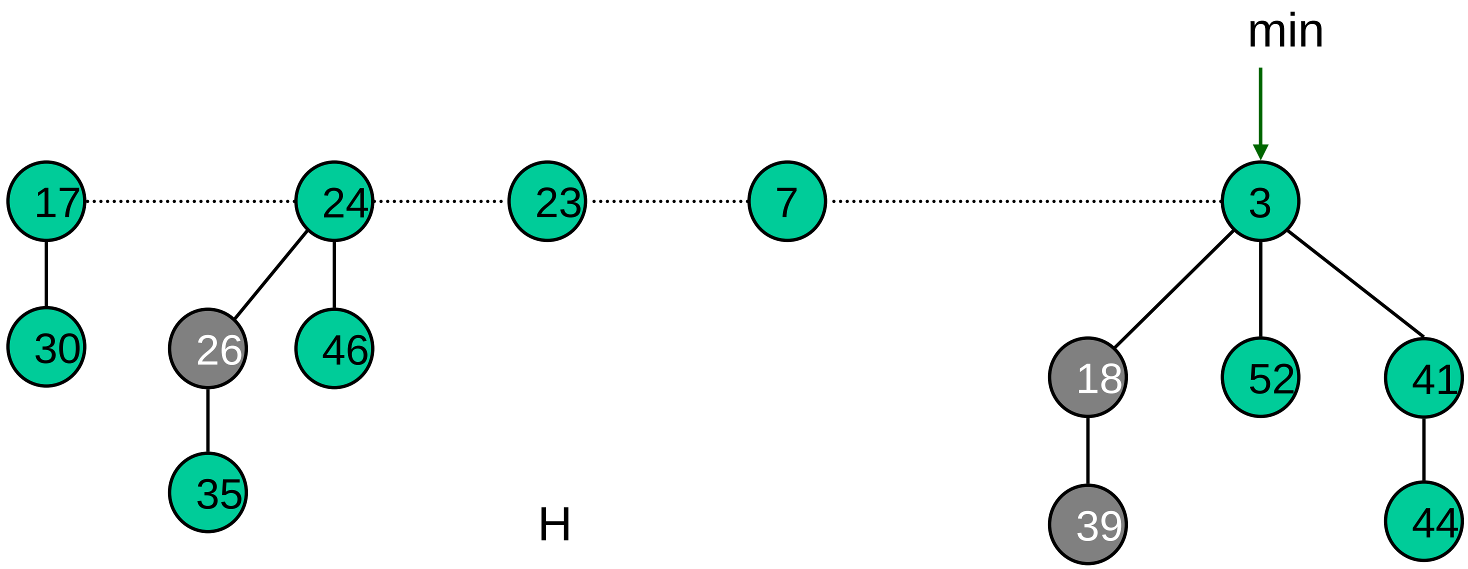
Key quantities:

- degree(x) = degree of node x.
- mark(x) = is node x marked?
- t(H) = # trees.
- m(H) = # marked nodes.
- Φ(H) = t(H) + 2m(H)

t(H) = ? 5

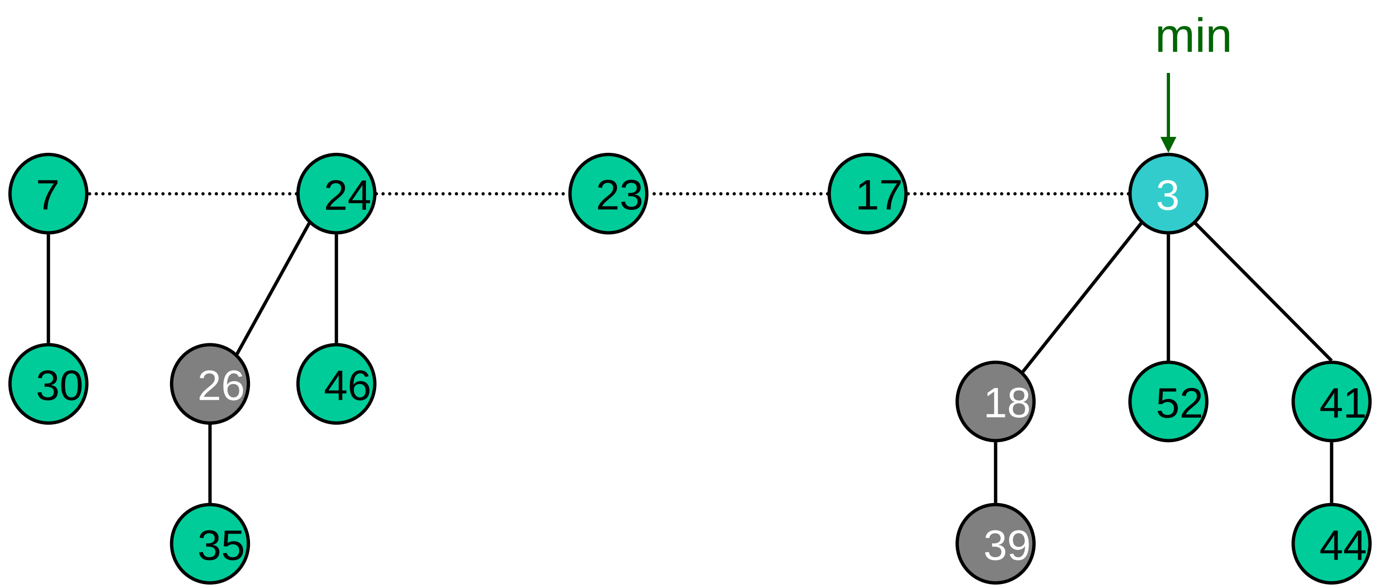
m(H) = ? 3

Φ(H) = ? 11



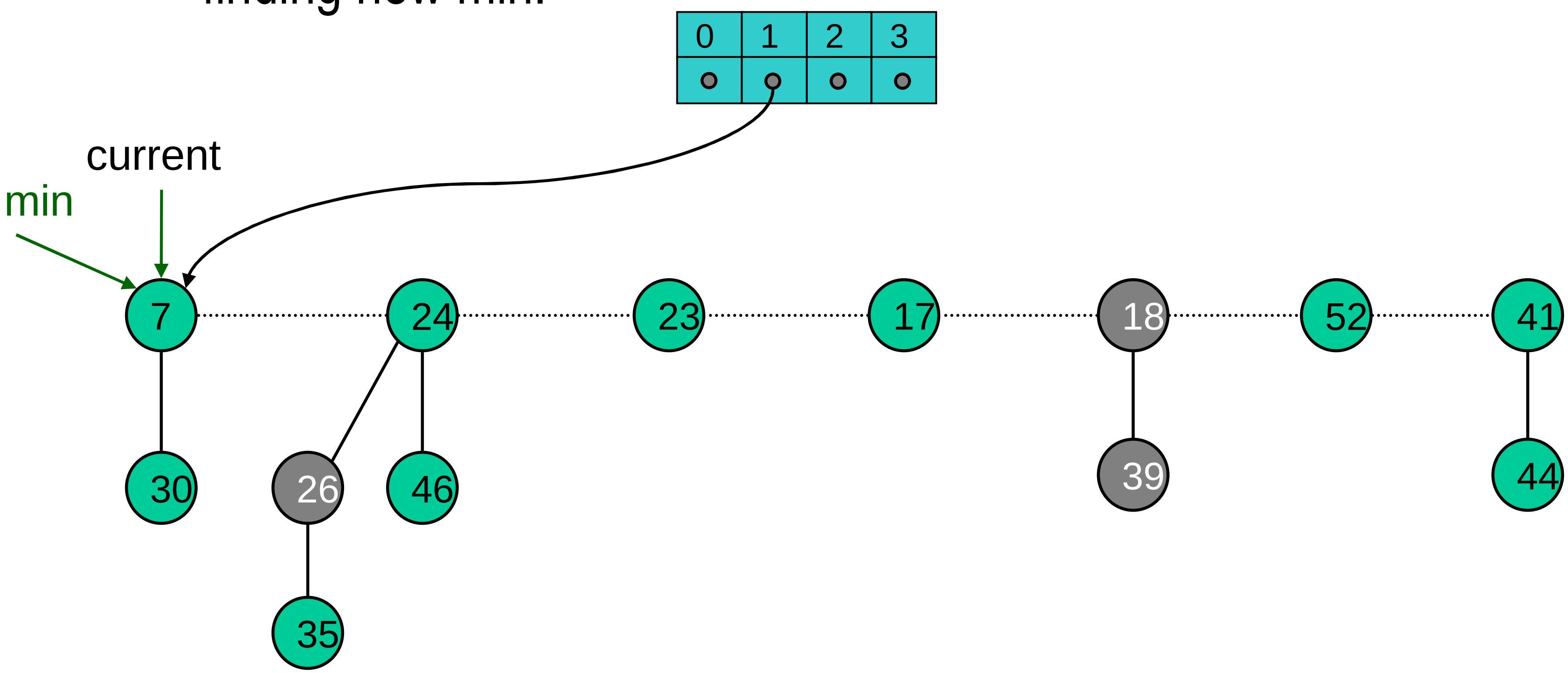
Delete Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



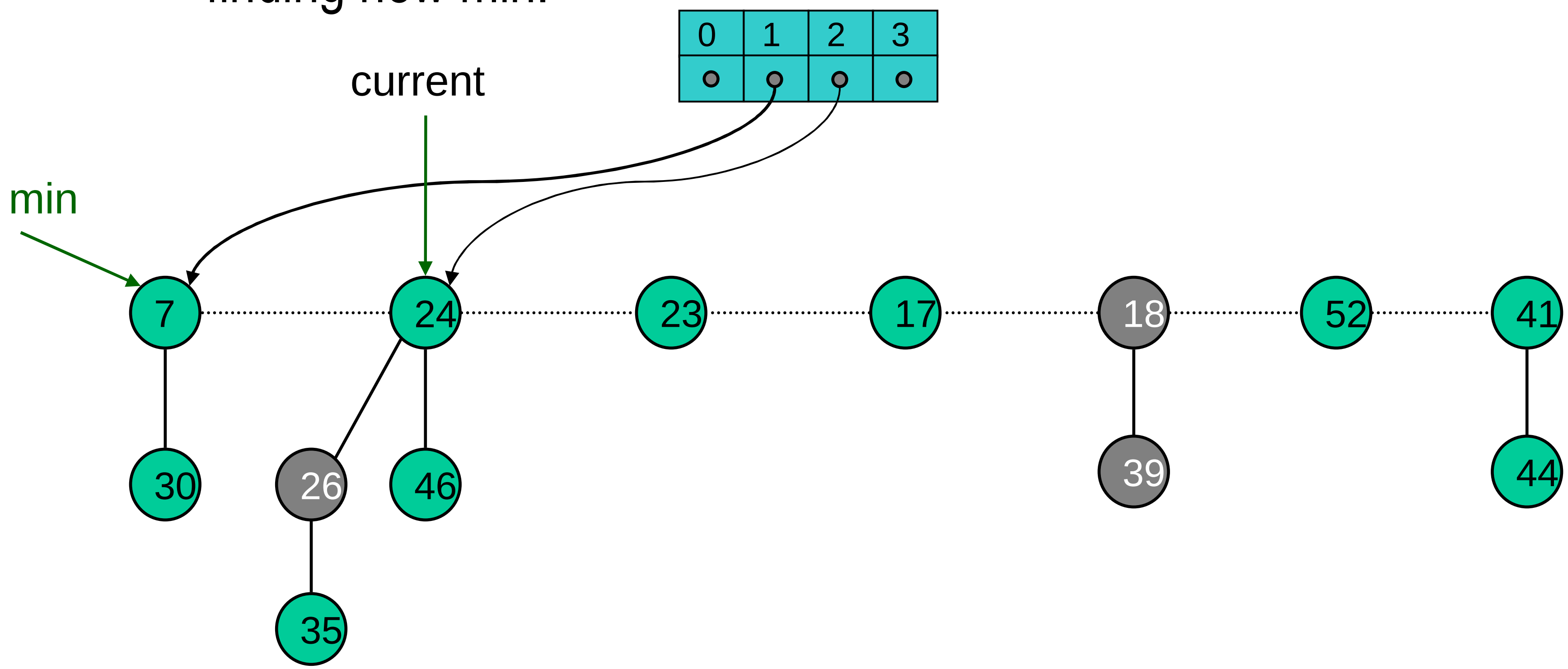
Delete Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



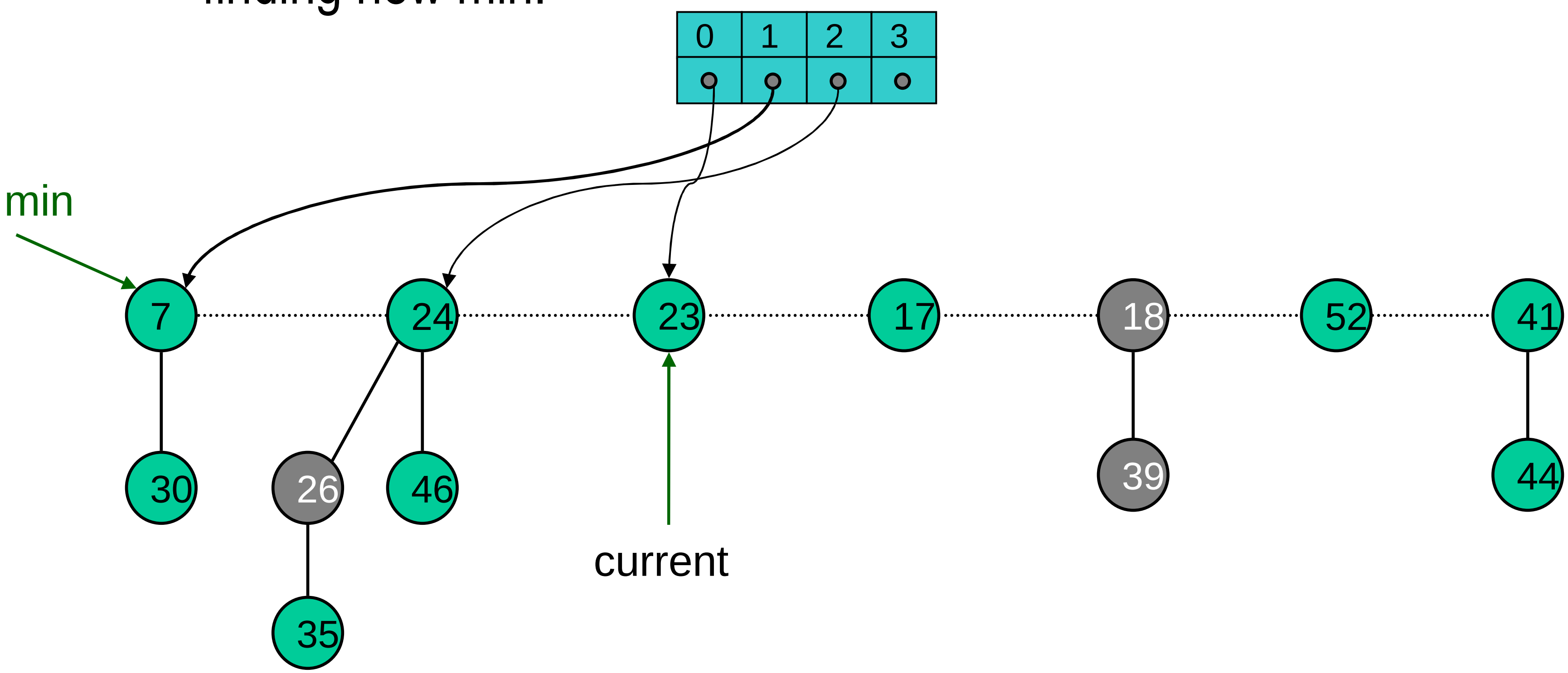
Delete Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



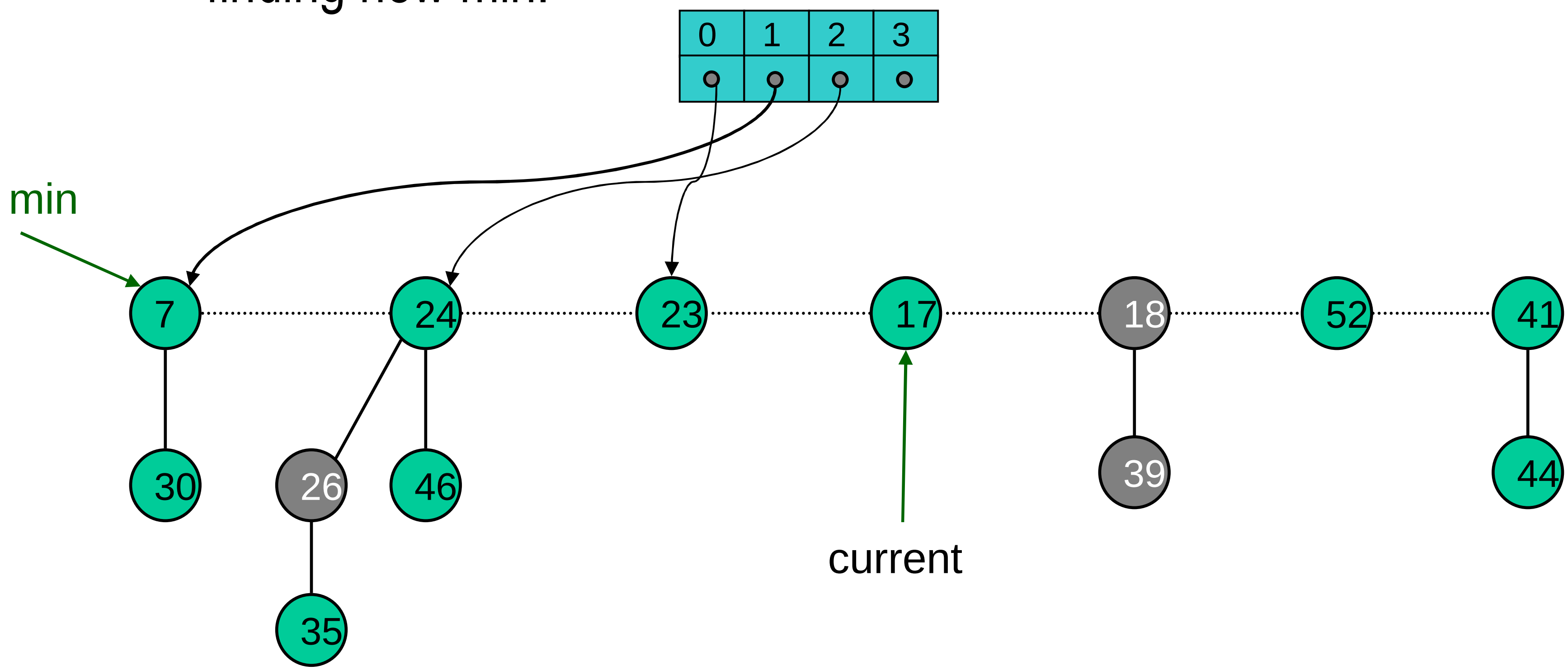
Delete Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Delete Min

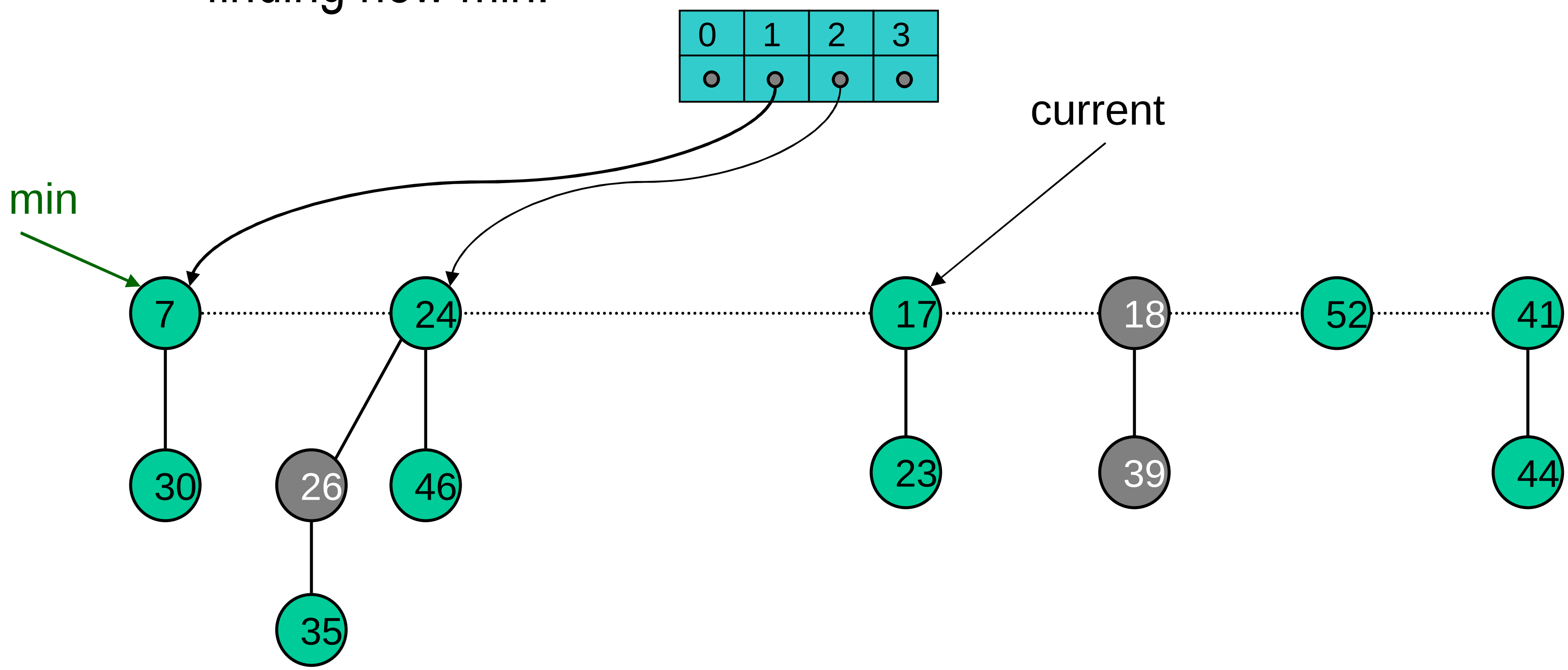
1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Merge 17 & 23 trees.

Delete Min

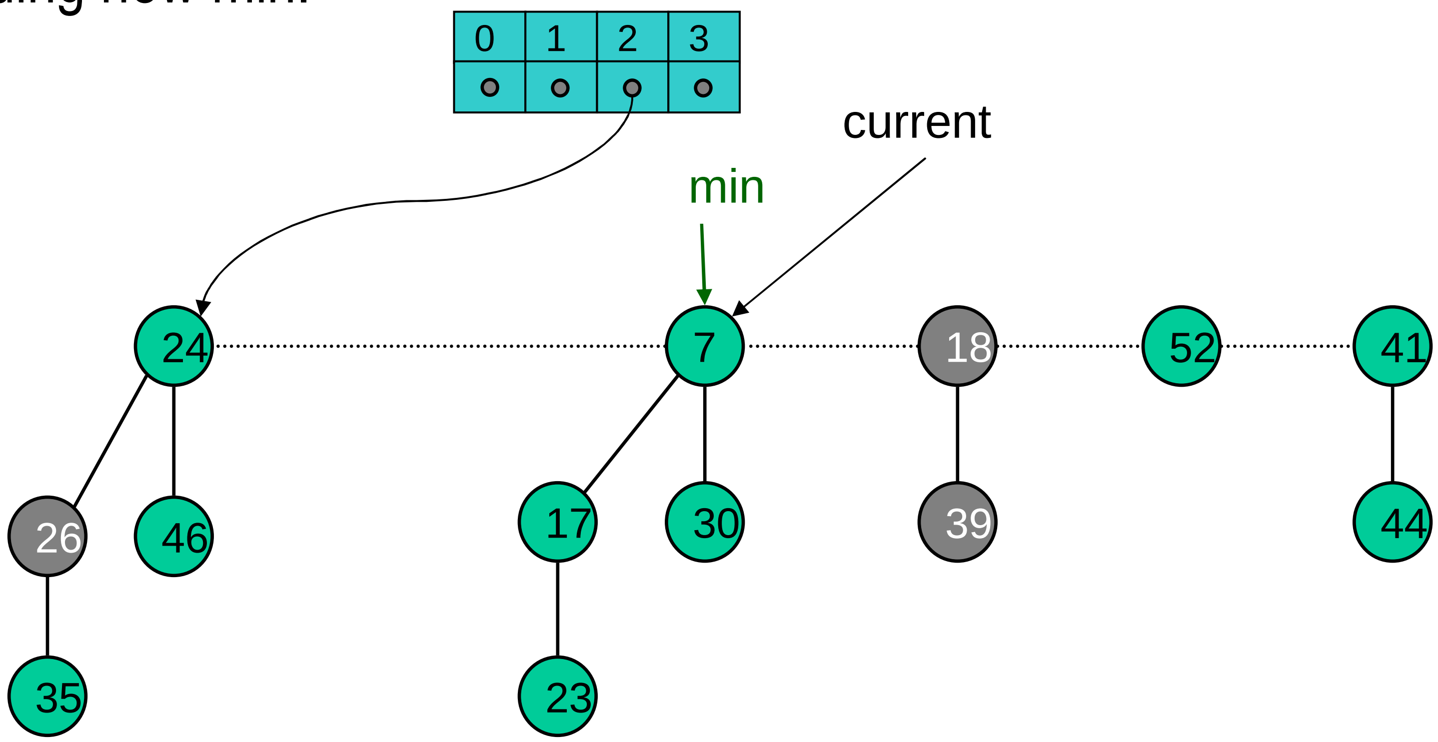
1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Merge 17 & 7 trees.

Delete Min

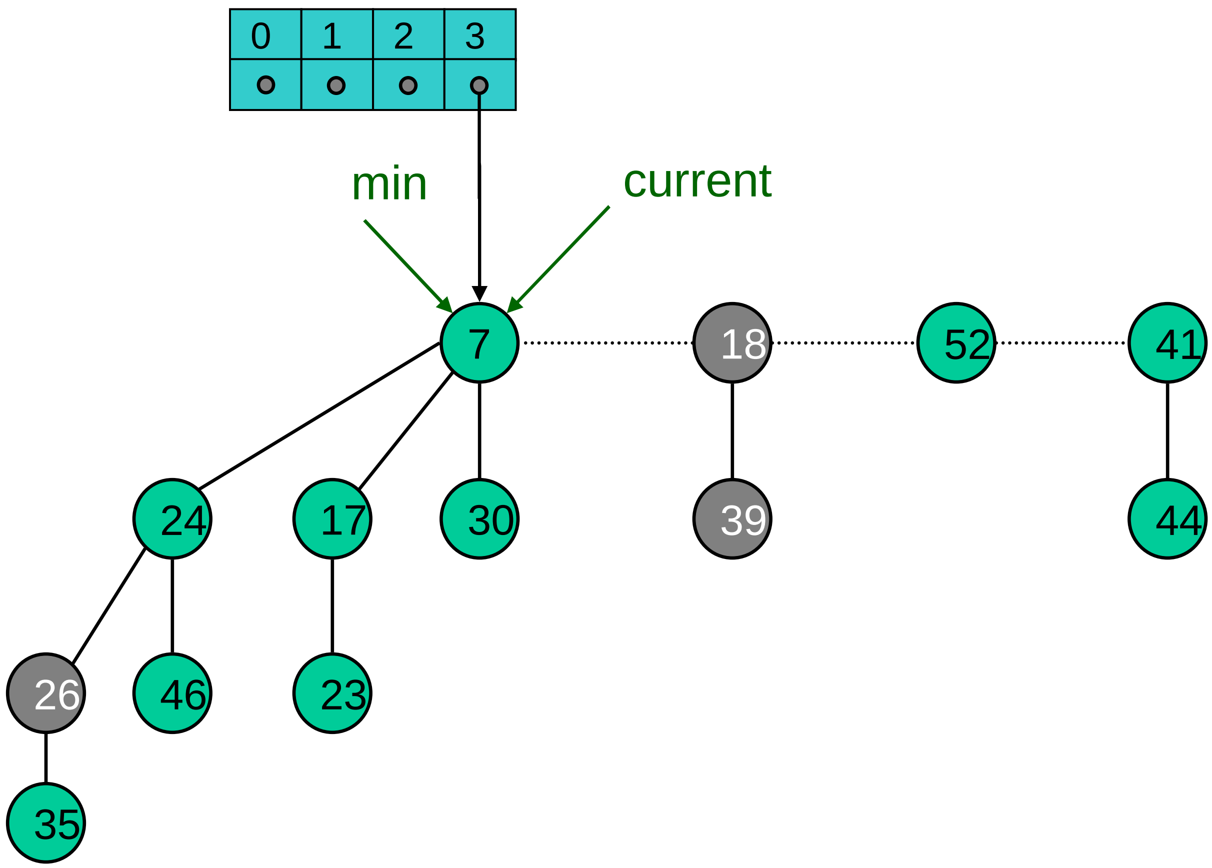
1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Merge 7 & 24 trees.

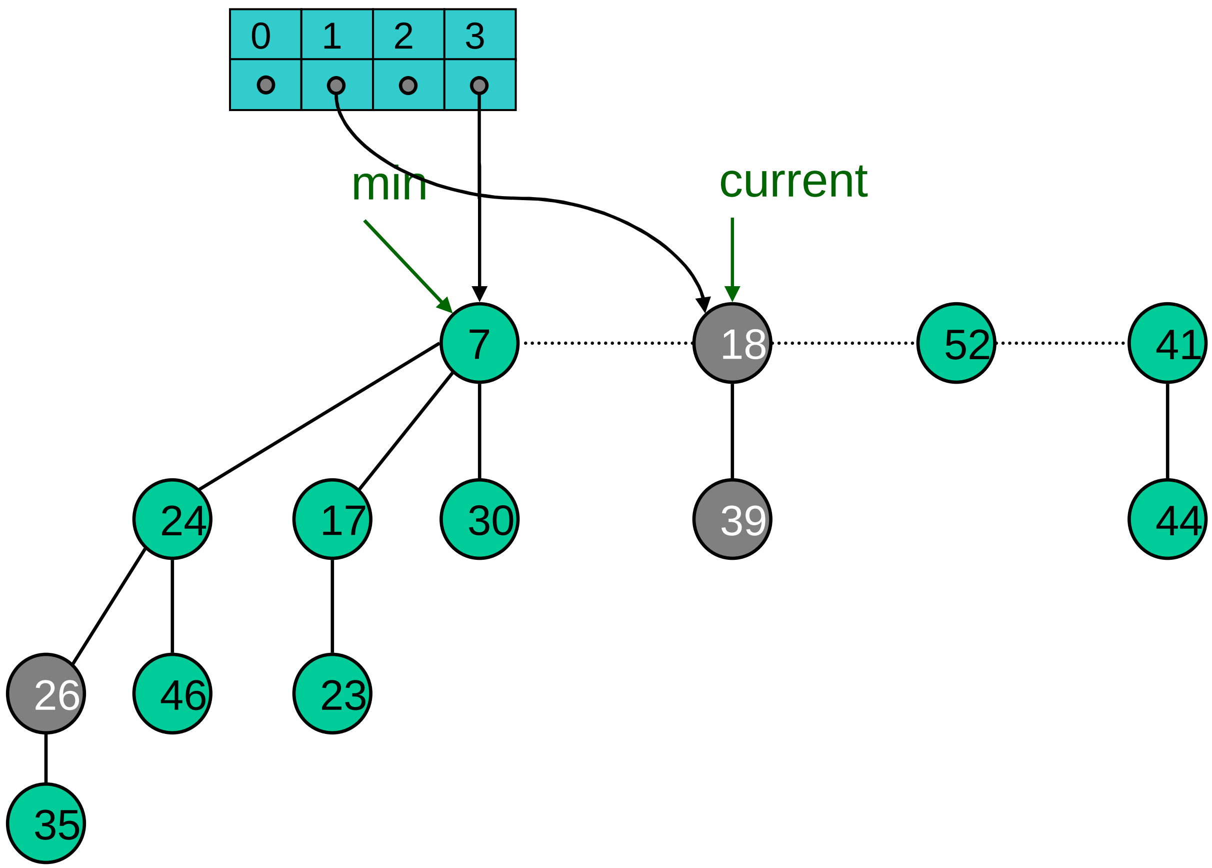
Delete Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



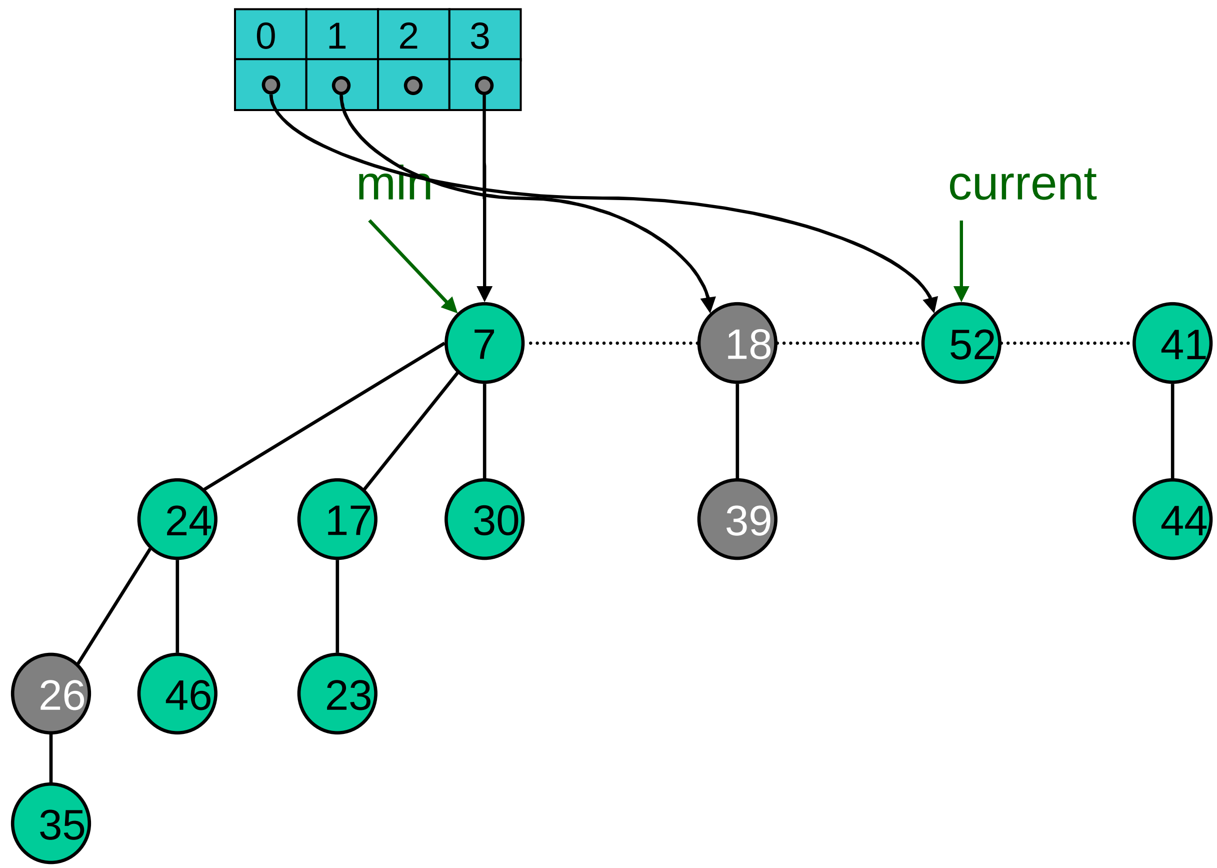
Delete Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



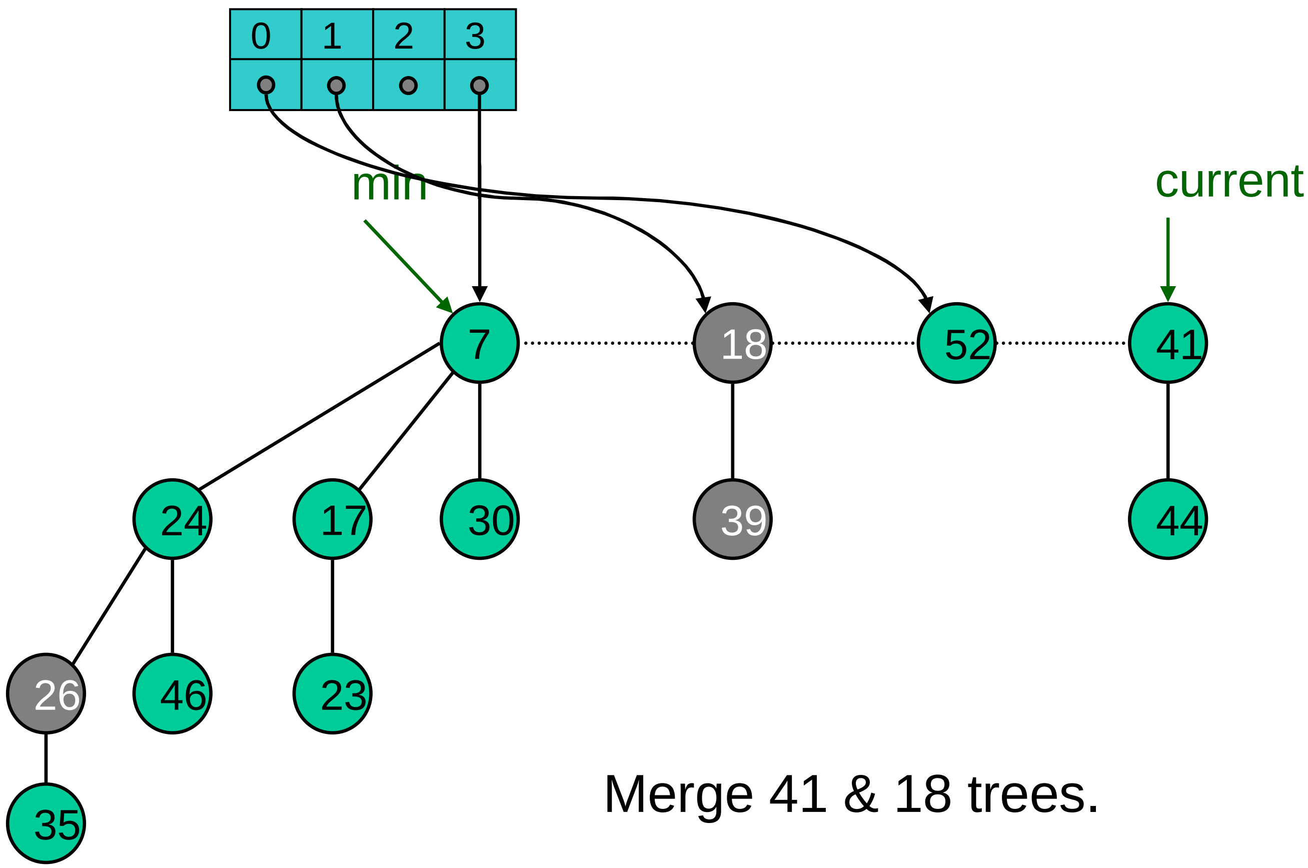
Delete Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Delete Min

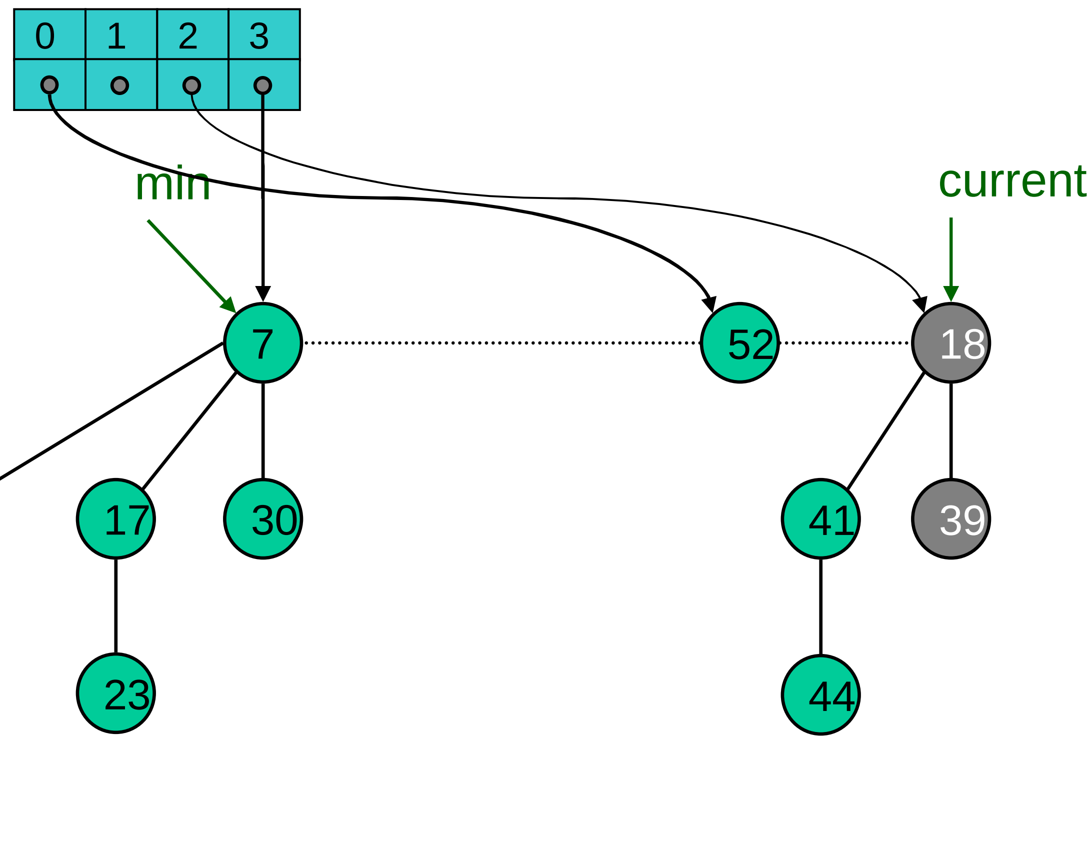
1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Merge 41 & 18 trees.

Delete Min

1. Delete min, concatenate its children into root list.
2. Consolidate trees so that no two roots have same degree, and finding new min.



Delete-Min Analysis

$\Phi(H) = t(H) + 2m(H)$

$D(n) =$ max degree of any node in Fibonacci heap with n nodes

Actual cost = $O(D(n) + t(H))$

- $O(1)$ work adding min's children into root list & updating min.
- $O(D(n) + t(H))$ work consolidating trees.
  - At most $D(n)$ children of min node.
  - $\leq D(n) + t(H) - 1$ trees at beginning of consolidation.
  - #trees decreases by one after each merging

Amortized cost = $O(D(n) + t(H)) + \Delta\Phi(H) = O(D(n))$

- $t(H') \leq D(n) + 1$, since no two trees have same degree.
- $m(H') \leq m(H)$
- $\Delta\Phi(H) \leq D(n) + 1 - t(H)$

Delete-Min Analysis

Theorem

The Delete-Min operation can be implemented to run in $O(D(n) + t(n))$ actual time, and $O(D(n))$ amortized time

Is amortized cost of $O(D(n))$ good?

Yes, if only Insert, Union, & Delete-min supported.

- In this case, Fibonacci heap contains only binomial trees, since we only merge trees of equal root degree.
- $D(n) \leq \lfloor \log_2 N \rfloor$

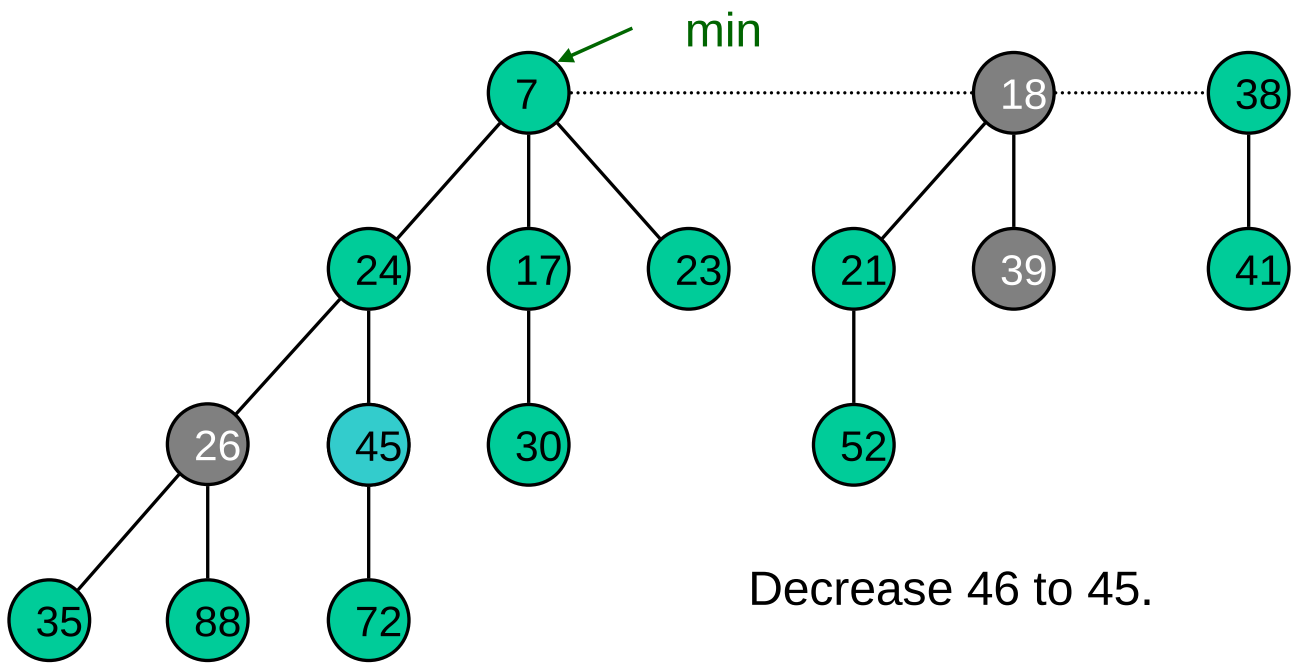
Yes, if we support Decrease-key cleverly.

- $D(n) \leq \lfloor \log_\phi N \rfloor$, where ϕ is golden ratio = 1.618…

Decrease Key

Case 0: min-heap property not violated.

1. Decrease key.
2. Change min pointer if necessary.

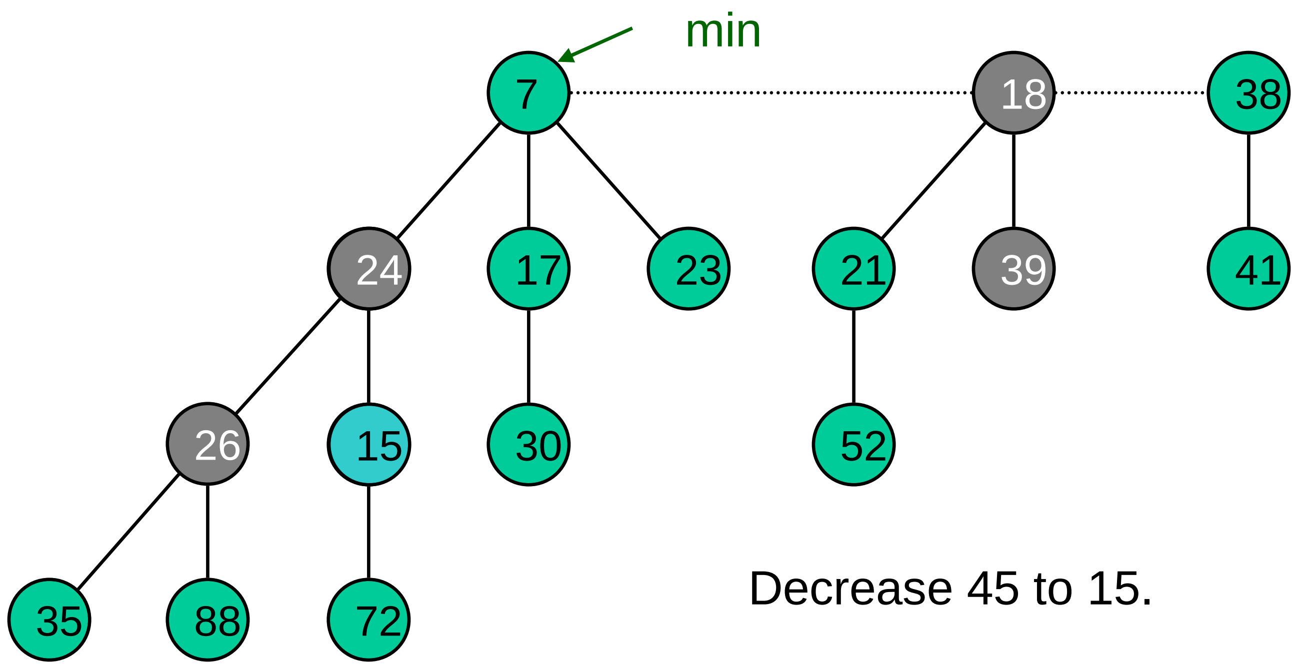


Decrease 46 to 45.

Decrease Key

Case 1: parent of x is unmarked.

1. Decrease key.
2. Remove link to parent.
3. Mark parent.
4. Add x's tree to root list, updating heap min pointer.

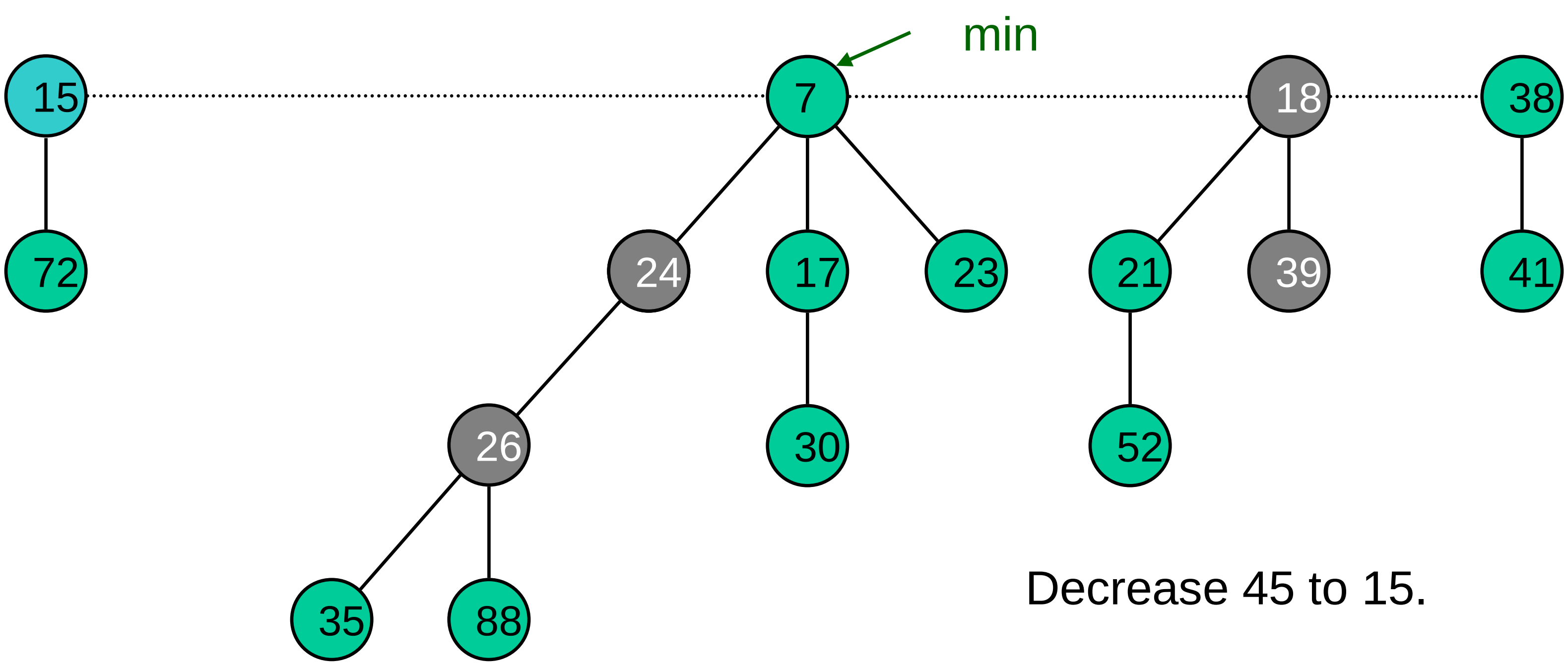


Decrease 45 to 15.

Decrease Key

Case 1: parent of x is unmarked.

1. Decrease key.
2. Remove link to parent.
3. Mark parent.
4. Add x's tree to root list, updating heap min pointer.

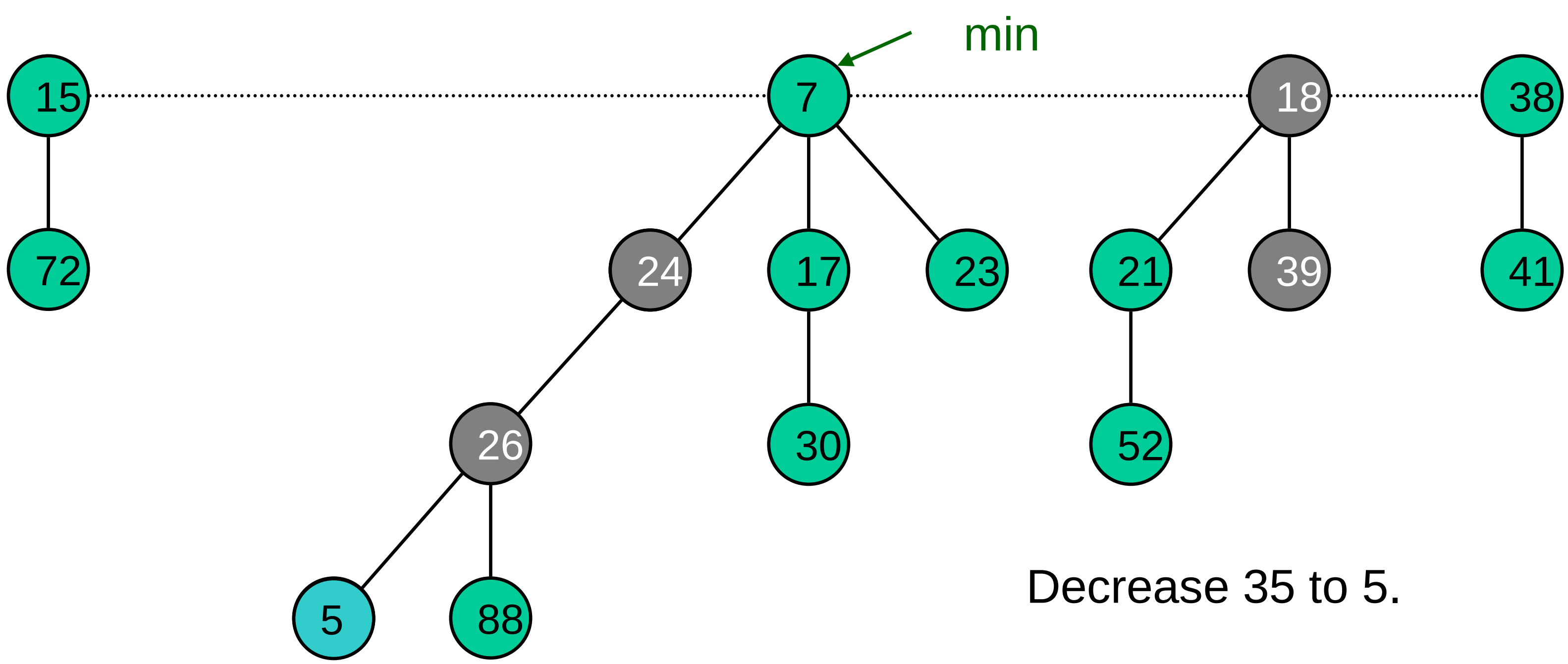


Decrease 45 to 15.

Decrease Key

Case 2: parent of x is marked.

1. Decrease key.
2. Move node to root list, updating heap min pointer.
3. Move chain of marked ancestors to root list, unmarking.
4. Mark first unmarked non-root ancestor.

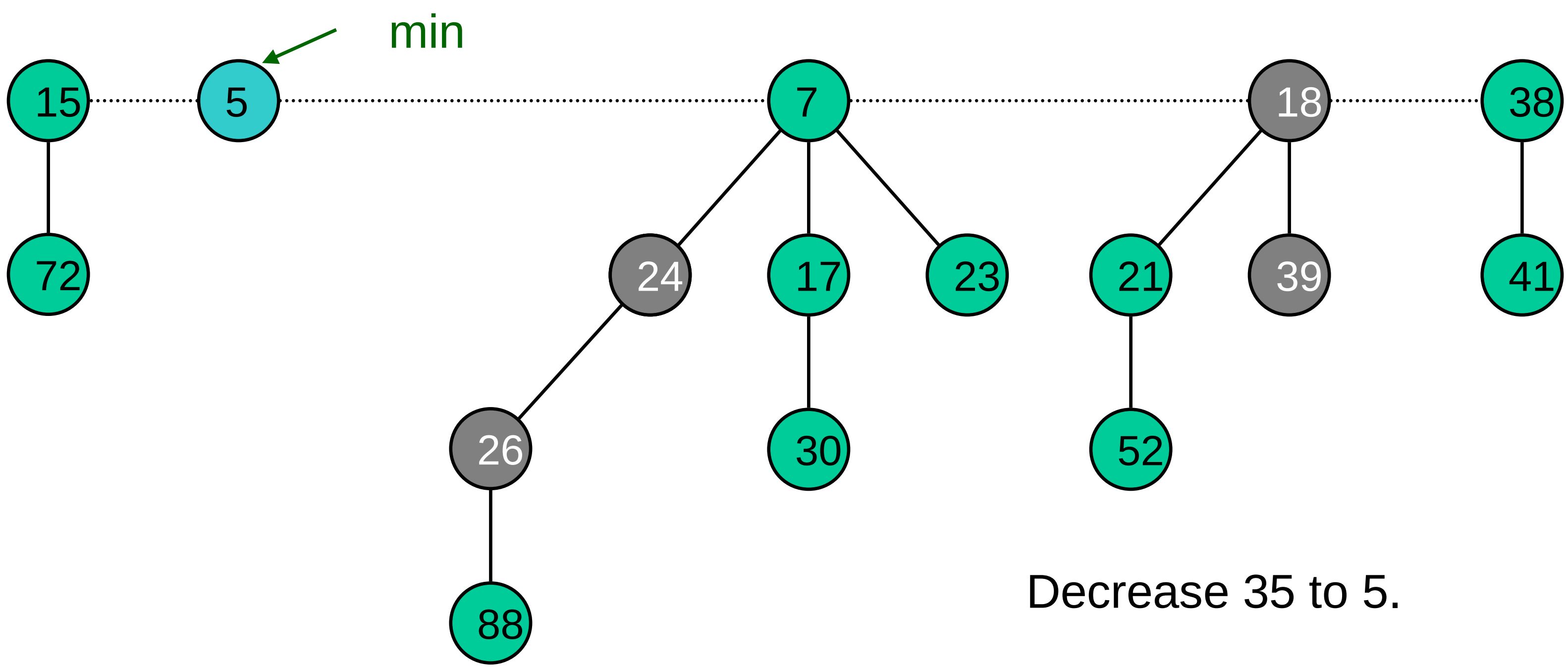


Decrease 35 to 5.

Decrease Key

Case 2: parent of x is marked.

1. Decrease key.
2. Move node to root list, updating heap min pointer.
3. Move chain of marked ancestors to root list, unmarking.
4. Mark first unmarked non-root ancestor.

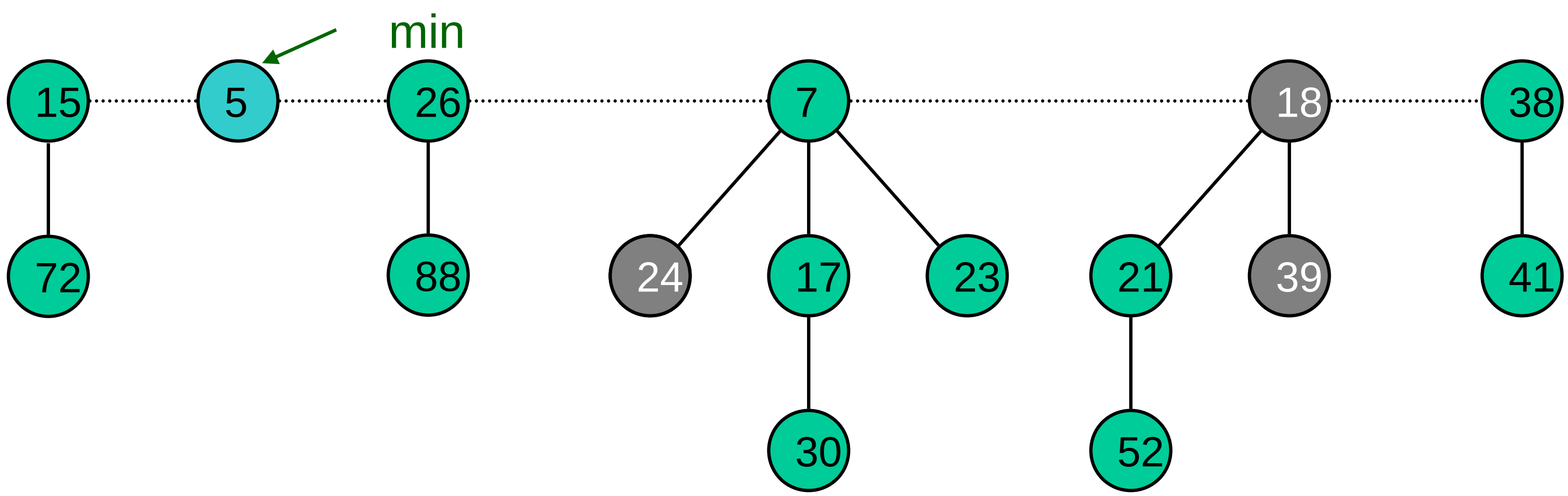


Decrease 35 to 5.

Decrease Key

Case 2: parent of x is marked.

1. Decrease key.
2. Move node to root list, updating heap min pointer.
3. Move chain of marked ancestors to root list, unmarking.
4. Mark first unmarked non-root ancestor.

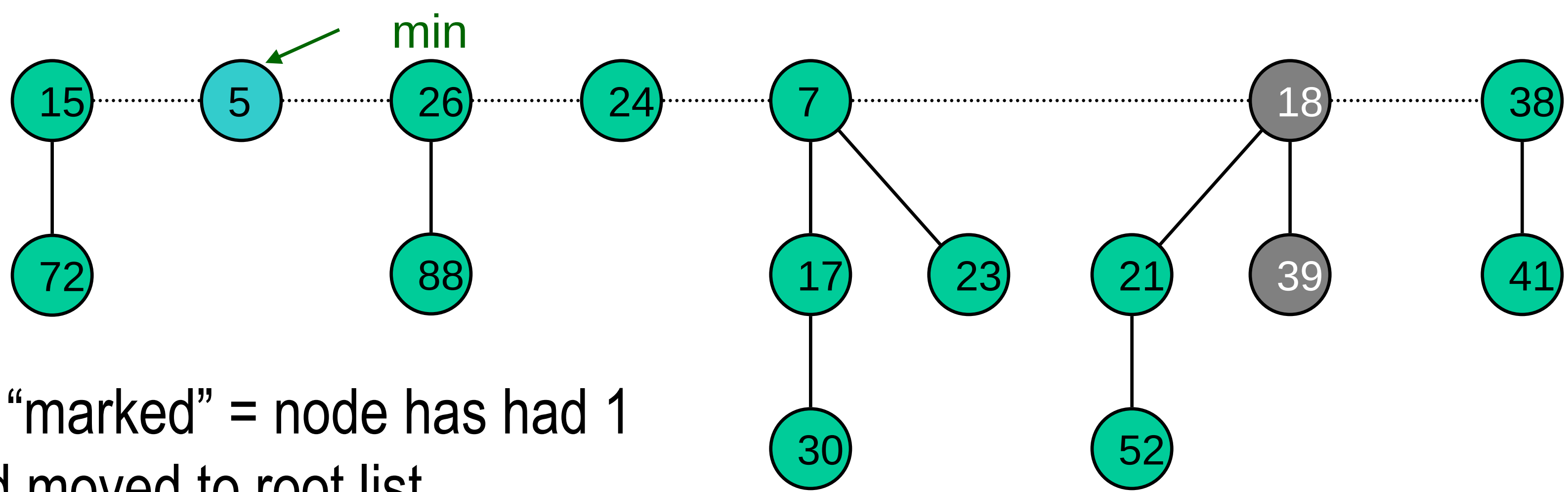


Decrease 35 to 5.

Decrease Key

Case 2: parent of x is marked.

1. Decrease key.
2. Move node to root list, updating heap min pointer.
3. Move chain of marked ancestors to root list, unmarking.
4. Mark first unmarked non-root ancestor.



I.e., "marked" = node has had 1 child moved to root list.
Once we move a 2nd child of node, we also move the node.

Decrease 35 to 5.

Decrease-Key Analysis

- $t(H)$ = # trees in heap H.
- $m(H)$ = # marked nodes in heap H.
- $\Phi(H)$ = $t(H) + 2m(H)$.

Actual cost = O(c), where c = # of nodes "cut"

- O(1) time for decrease key.
- O(1) time **for each** cut.

Amortized cost = $O(c) + \Delta\Phi(H) = O(1)$

- $t(H') = t(H) + c$
- $m(H') \leq m(H) - c + 2$
  - Each cut unmarks a node.
  - Last cut could potentially mark a node.
- $\Delta\Phi(H) \leq c + 2(-c + 2) = 4 - c$.

Decrease-Key Analysis

Theorem

The Decrease-Key operation can be implemented to run in $O(c)$ actual time, where c is the number of nodes "cut", and in $O(1)$ amortized time

Delete

1. Decrease key of x to $-\infty$.
2. Delete min element in heap.

Amortized cost = O(D(n))

- O(1) for decrease-key.
- O(D(n)) for delete-min.
- D(n) = max degree of any node in Fibonacci heap.

Bounding D(n)

D(n) = max degree in Fibonacci heap with n nodes.

Theorem

$D(n) \le \log_\phi n$, where $\phi = (1 + \sqrt{5}) / 2$.

Thus, Delete & Delete-min take O(log n) amortized time.

Proof is somewhat tedious & explained well in [CLRS].

Key Lemma

Let size(x) = #nodes in the subtree rooted at x.

Then, $\phi^{\text{degree}(x)} \le \text{size}(x)$.