Fibonacci Heaps

Chapter 20 of CLRS Book.

Based on the slides by Kevin Wayne,

Princeton University

COS 423, Theory of Algorithms, Spring 2002

Priority Queues

		Heaps				
Operation	Linked List	Binary	Binomial	Fibonacci †	Relaxed
make-heap	1	1	1	1	1
insert	1	log N	log N	1	1
find-min	N	1	log N	1	1
delete-min	N	log N	log N	log N	log N
union	1	N	log N	1	1
decrease-key	1	log N	log N	1	1
delete	N	log N	log N	log N	log N
is-empty	1	1	1	1	1

† amortized

this time

Fibonacci Heaps

Fibonacci heap history. Fredman and Tarjan (1986)

- Ingenious data structure and analysis.
- Original motivation: O(m + n log n) shortest path algorithm.
  - also led to faster algorithms for MST, weighted bipartite matching
- Still ahead of its time.

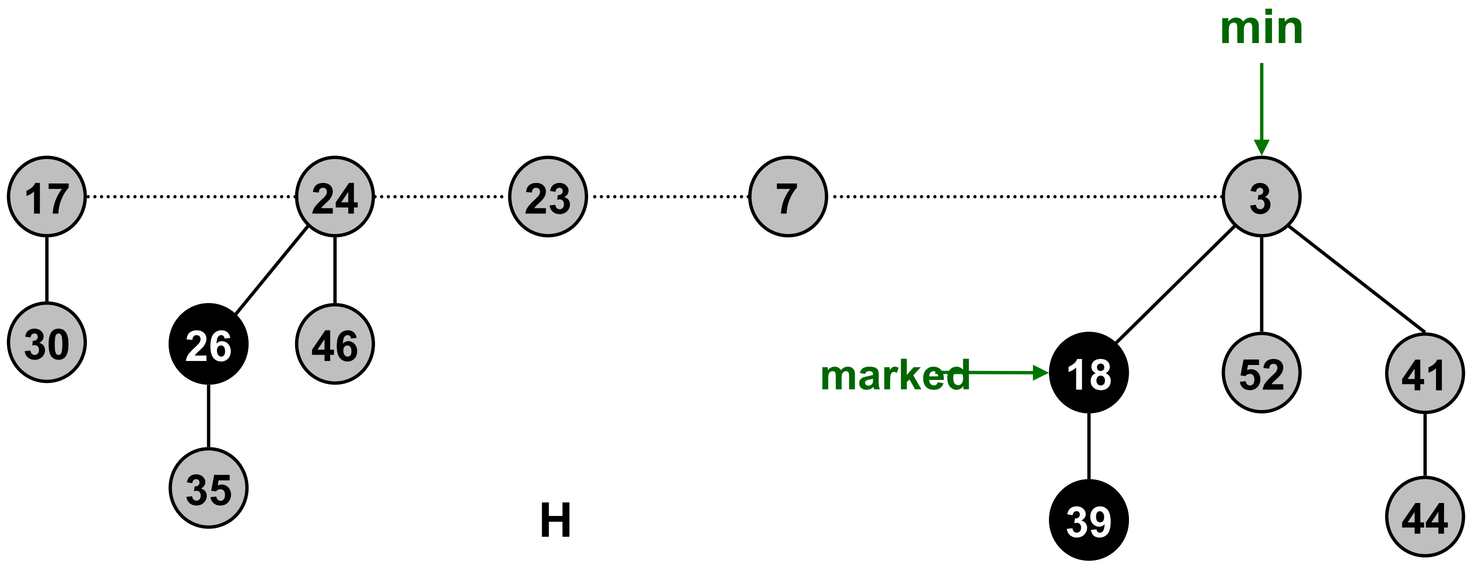
Fibonacci heap intuition.

- Similar to binomial heaps, but less structured.
- Decrease-key and union run in O(1) time.
- "Lazy" unions.

Fibonacci Heaps: Structure

Fibonacci heap.

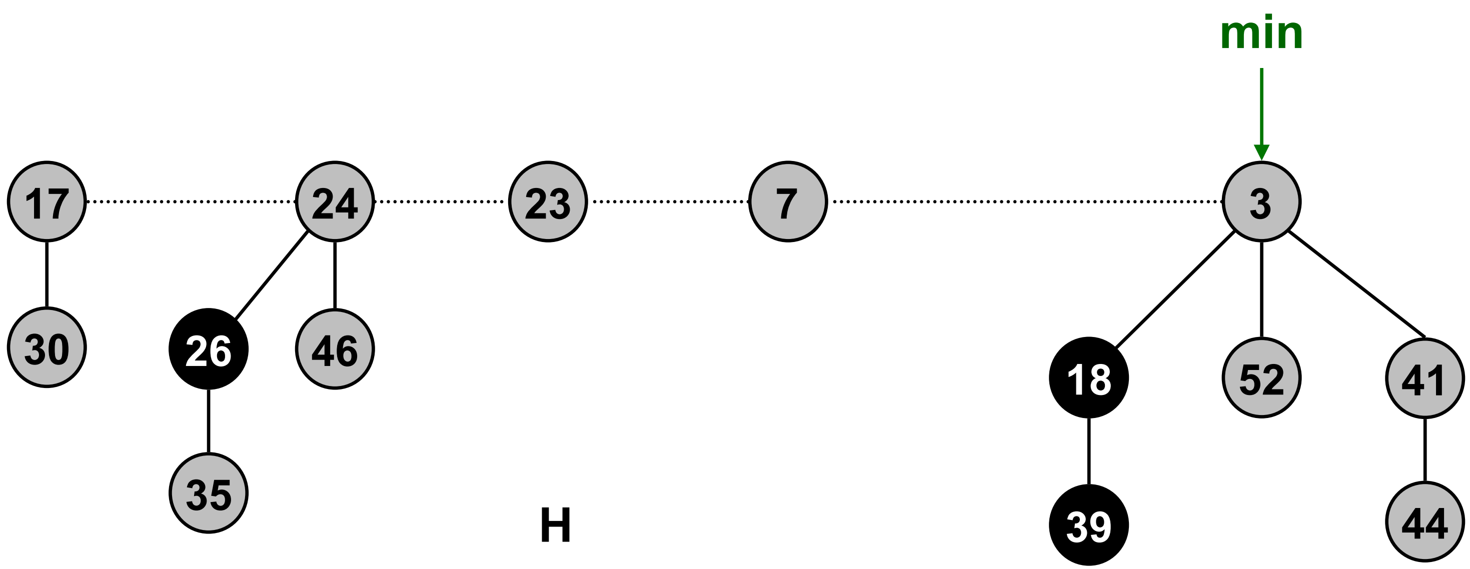
- **Set of min-heap ordered trees.**



Fibonacci Heaps: Implementation

Implementation.

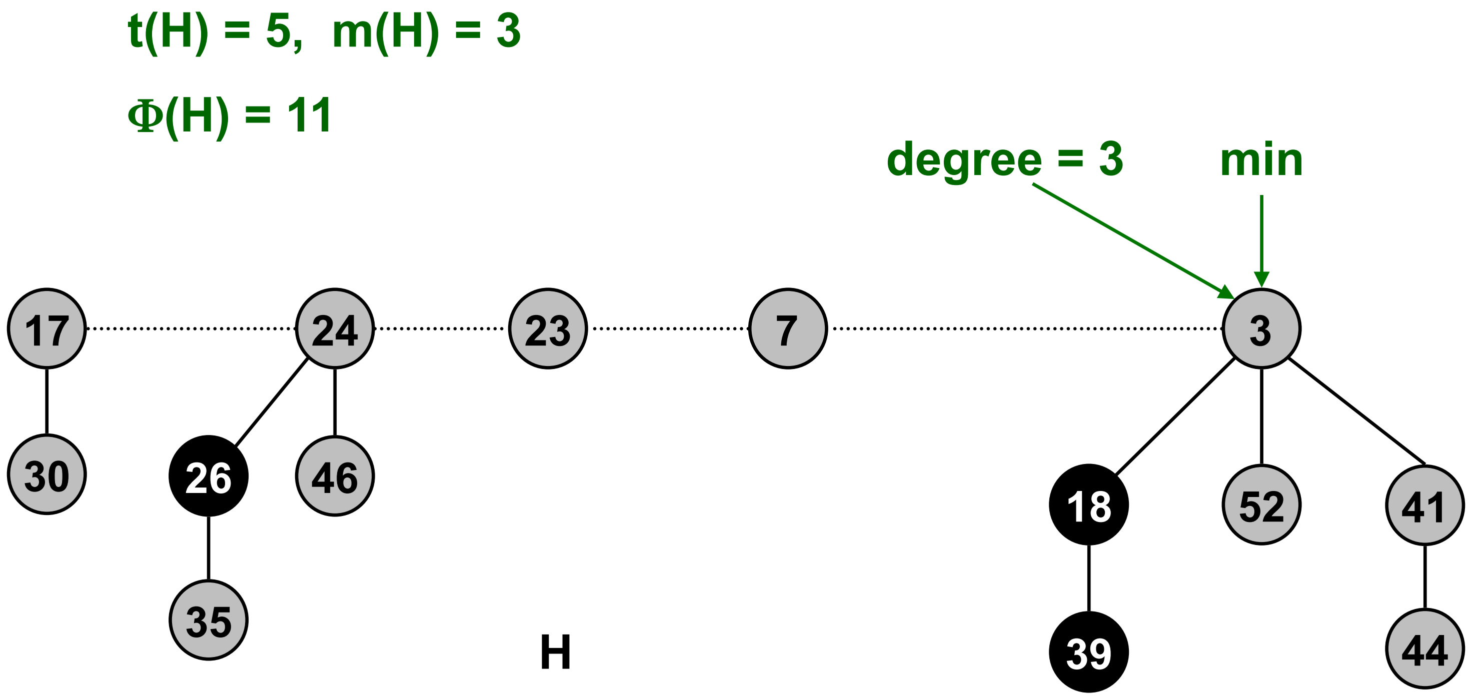
- Represent trees using left-child, right sibling pointers and circular, doubly linked list.
  - can quickly splice off subtrees
- Roots of trees connected with circular doubly linked list.
  - fast union
- Pointer to root of tree with min element.
  - fast find-min



Fibonacci Heaps: Potential Function

Key quantities.

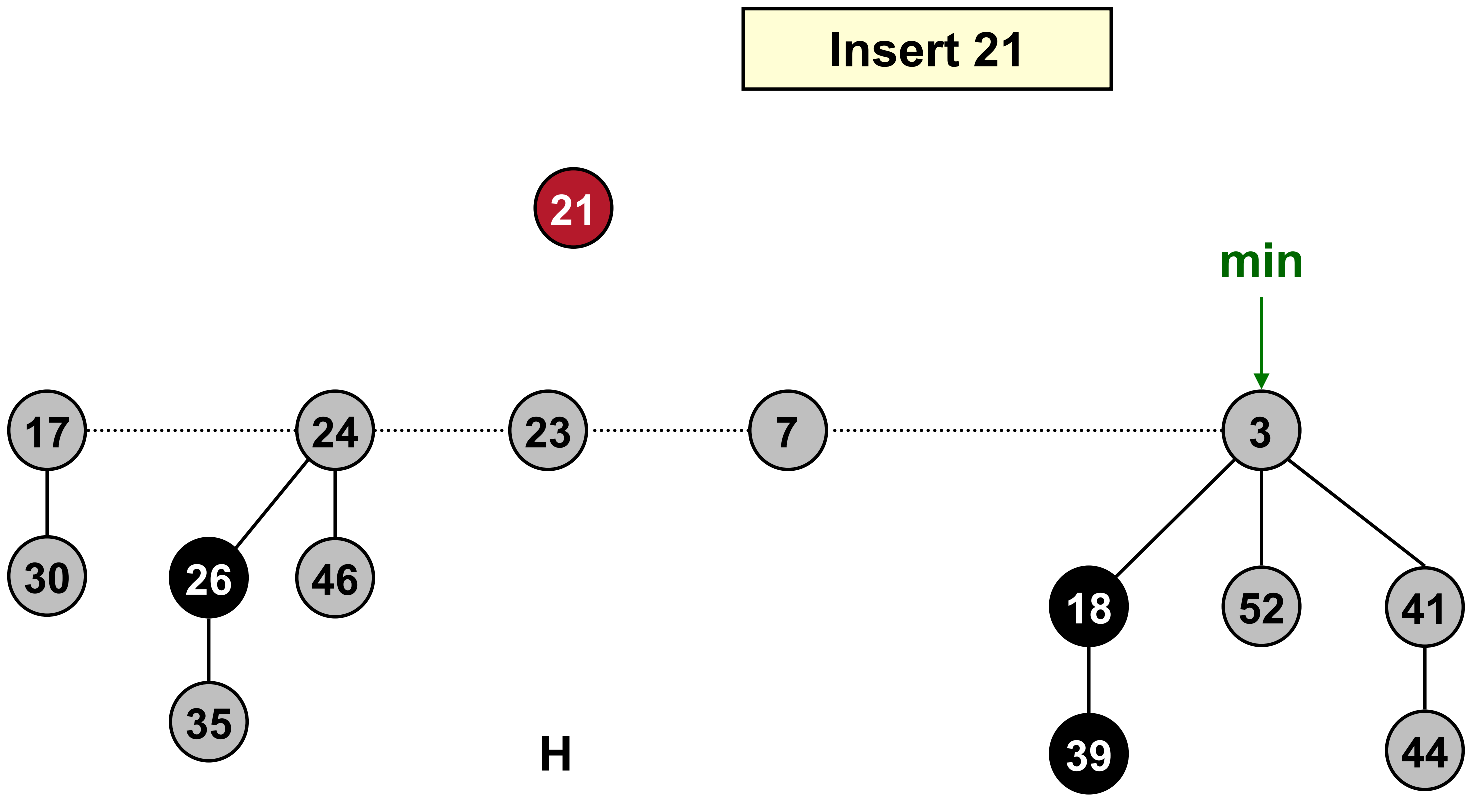
- Degree[x] = degree of node x.
- Mark[x] = mark of node x (black or gray).
- t(H) = # trees.
- m(H) = # marked nodes.
- Φ(H) = t(H) + 2m(H) = potential function. Nonzero at all times, used for Amortized Analysis.



Fibonacci Heaps: Insert

Insert.

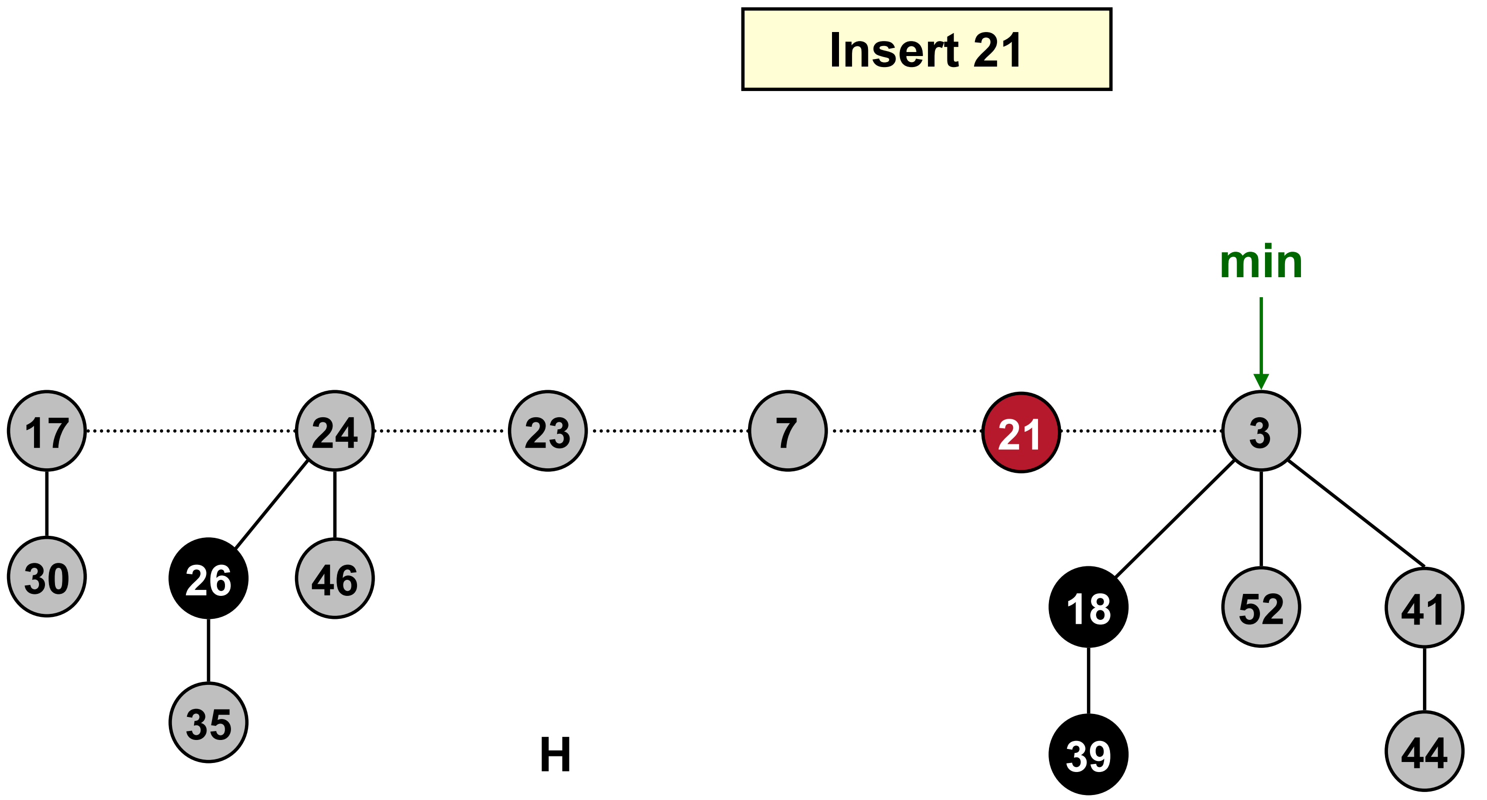
- Create a new singleton tree.
- Add to left of min pointer.
- Update min pointer.



Fibonacci Heaps: Insert

Insert.

- Create a new singleton tree.
- Add to left of min pointer.
- Update min pointer.



Fibonacci Heaps: Insert

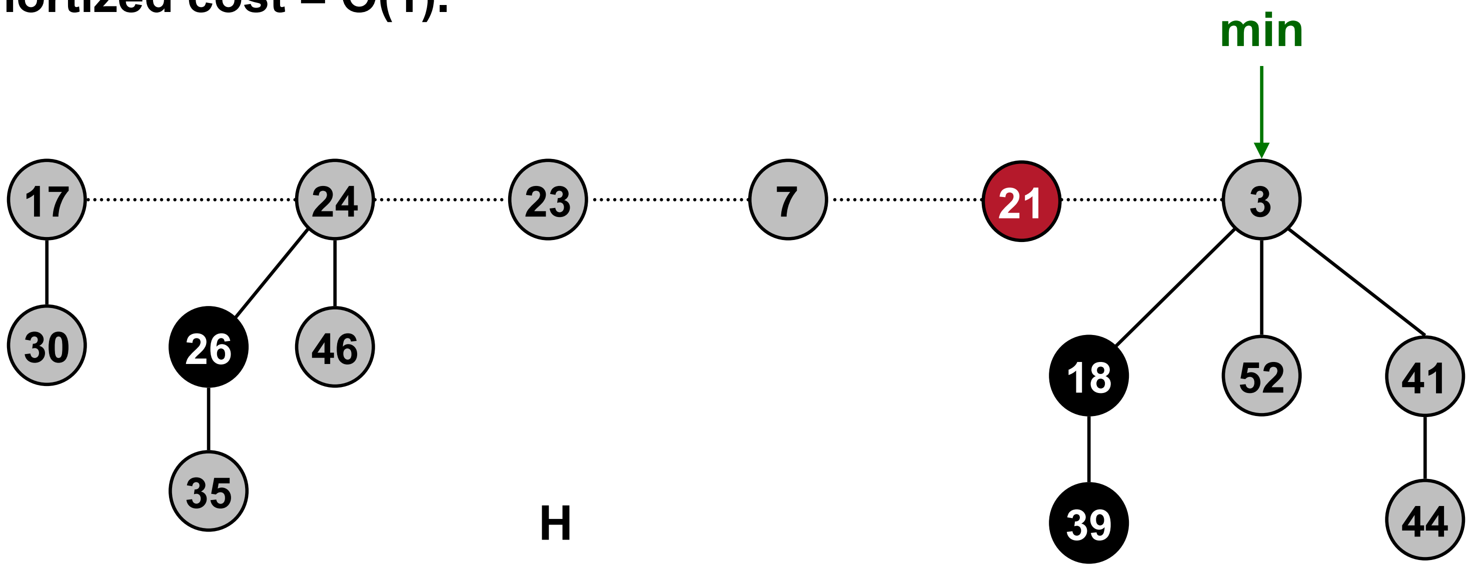
Insert.

- Create a new singleton tree.
- Add to left of min pointer.
- Update min pointer.

Running time. **O(1) amortized**

- Actual cost = O(1).
- Change in potential = +1.
- Amortized cost = O(1).

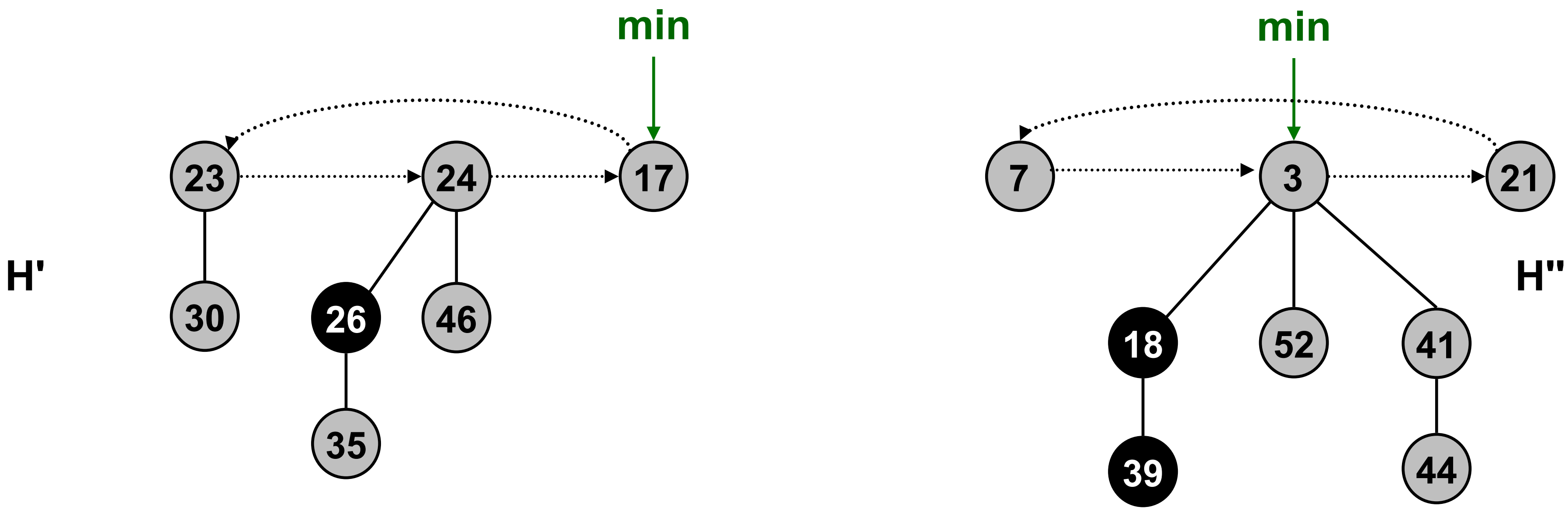
Insert 21



Fibonacci Heaps: Union

Union.

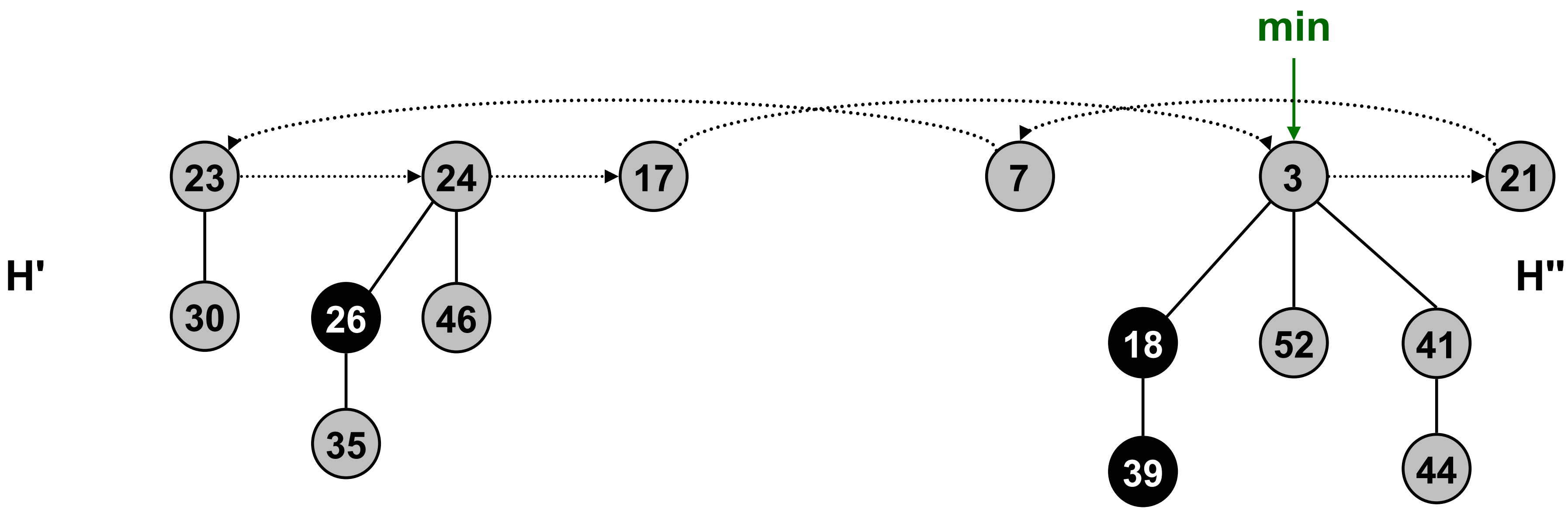
- Concatenate two Fibonacci heaps.
- Root lists are circular, doubly linked lists.



Fibonacci Heaps: Union

Union.
- Concatenate two Fibonacci heaps.
- Root lists are circular, doubly linked lists.

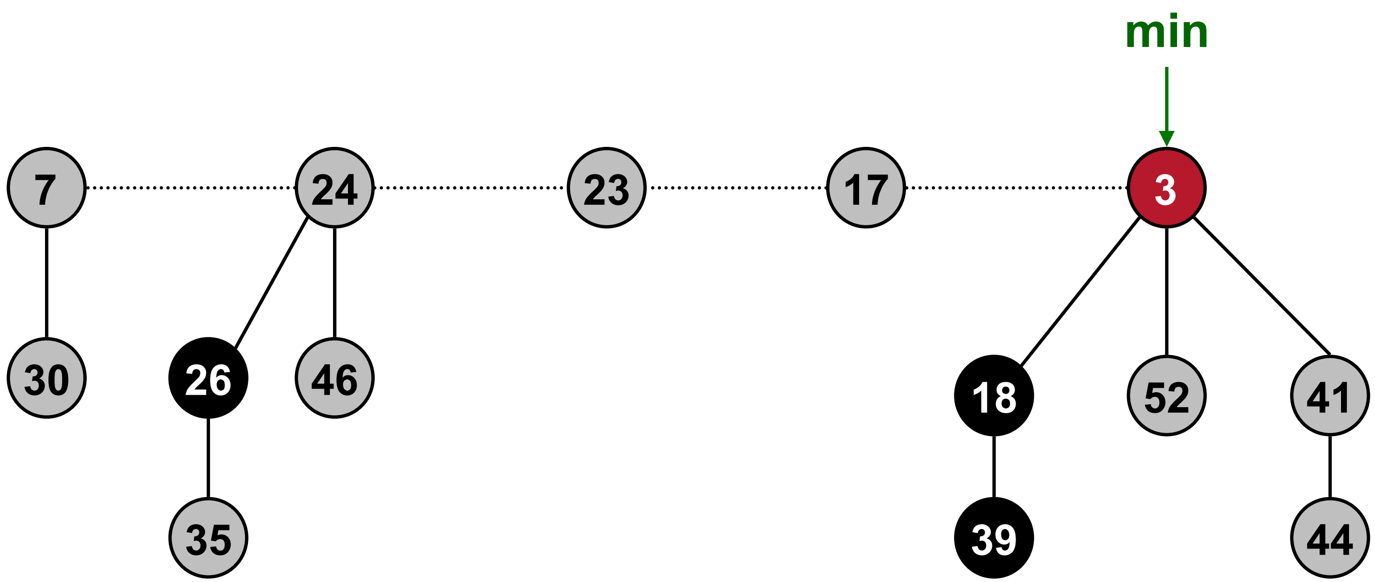
Running time. **O(1) amortized**
- Actual cost = O(1).
- Change in potential = 0.
- Amortized cost = O(1).



Fibonacci Heaps: Delete Min

Delete min.

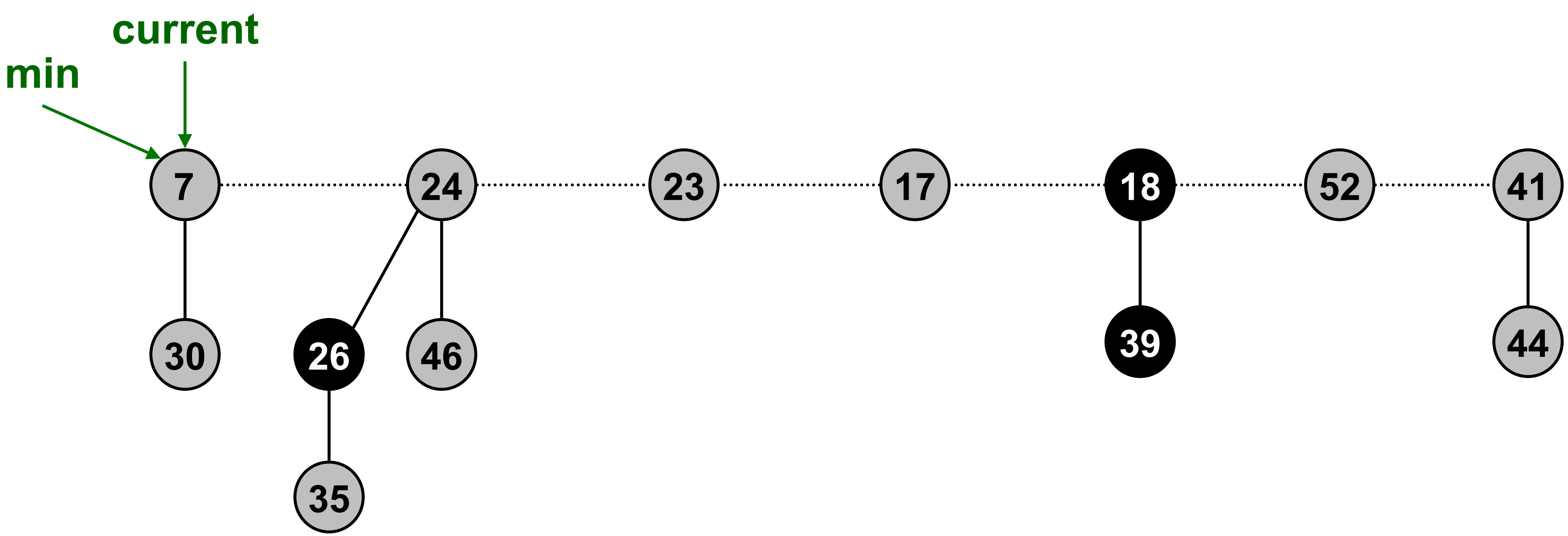
- **Delete min and concatenate its children into root list.**
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Delete Min

Delete min.

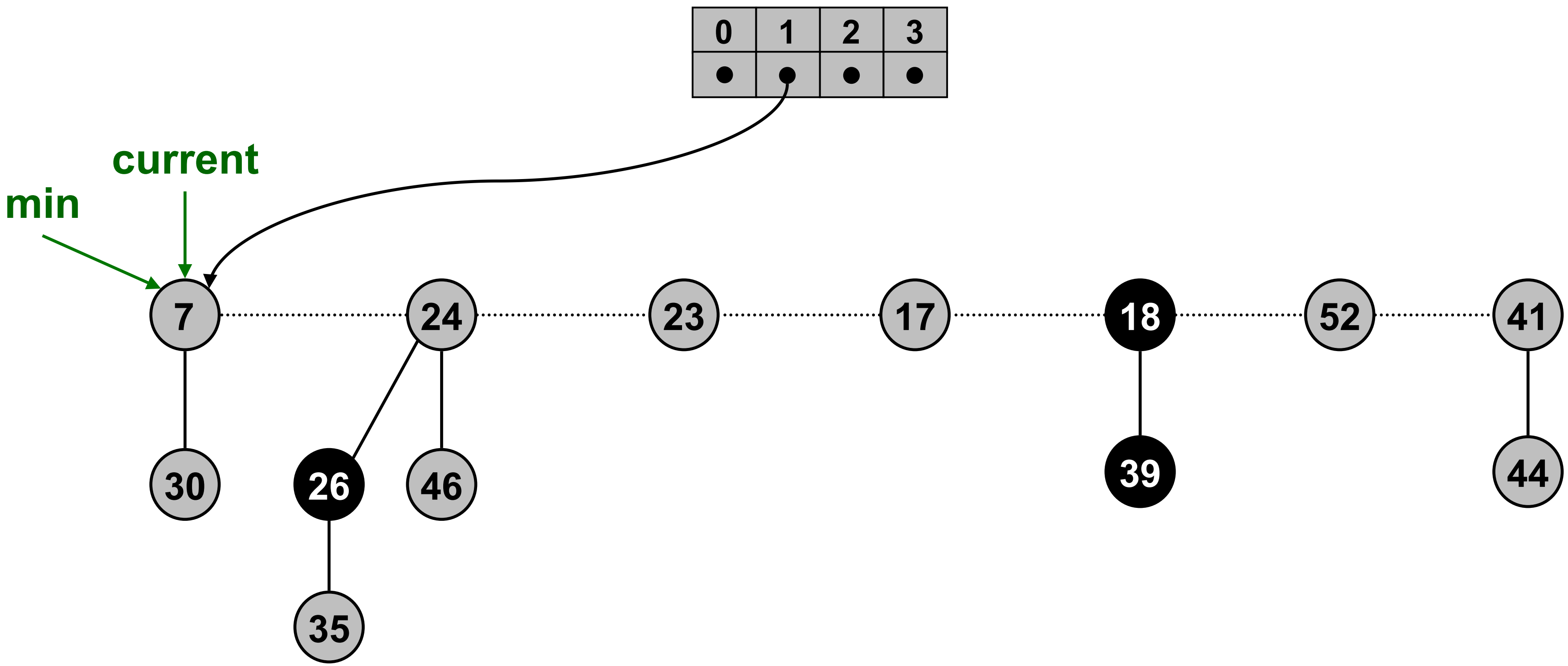
- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**



Fibonacci Heaps: Delete Min

Delete min.

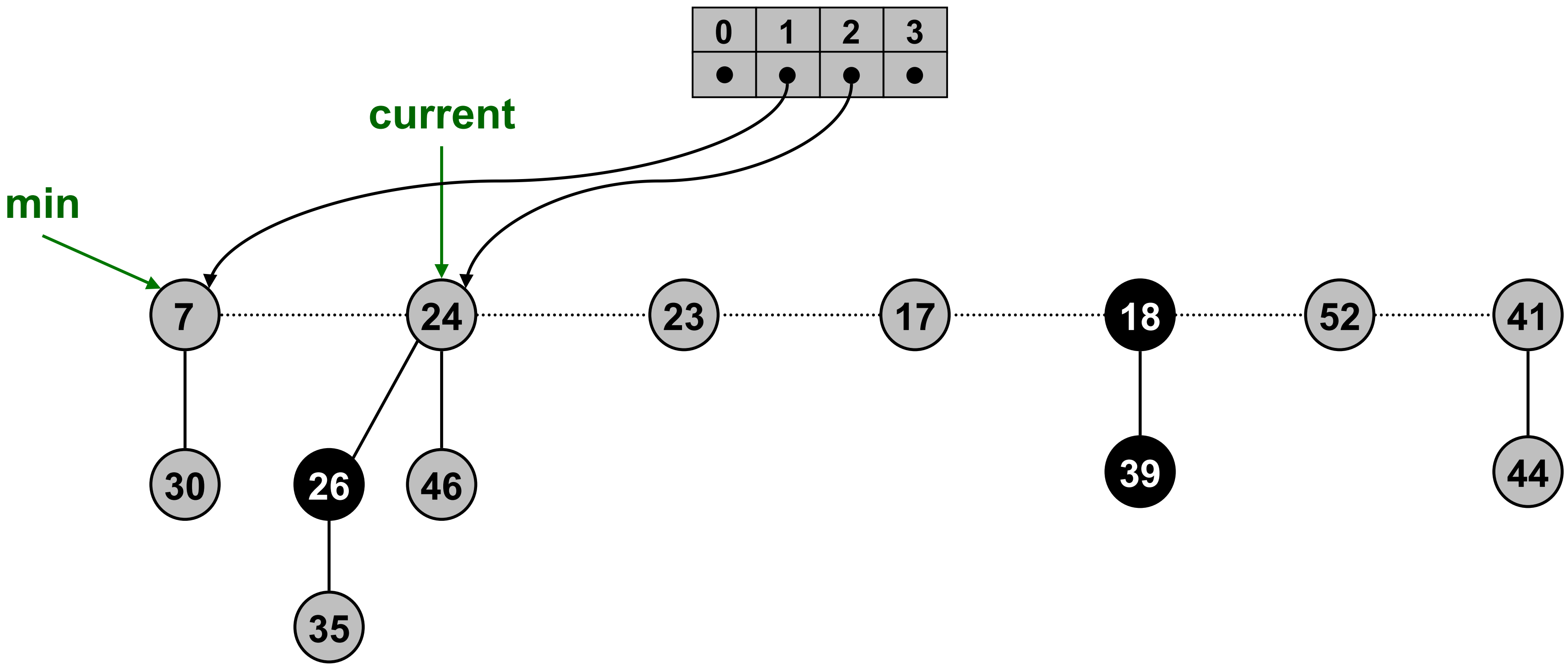
- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**



Fibonacci Heaps: Delete Min

Delete min.

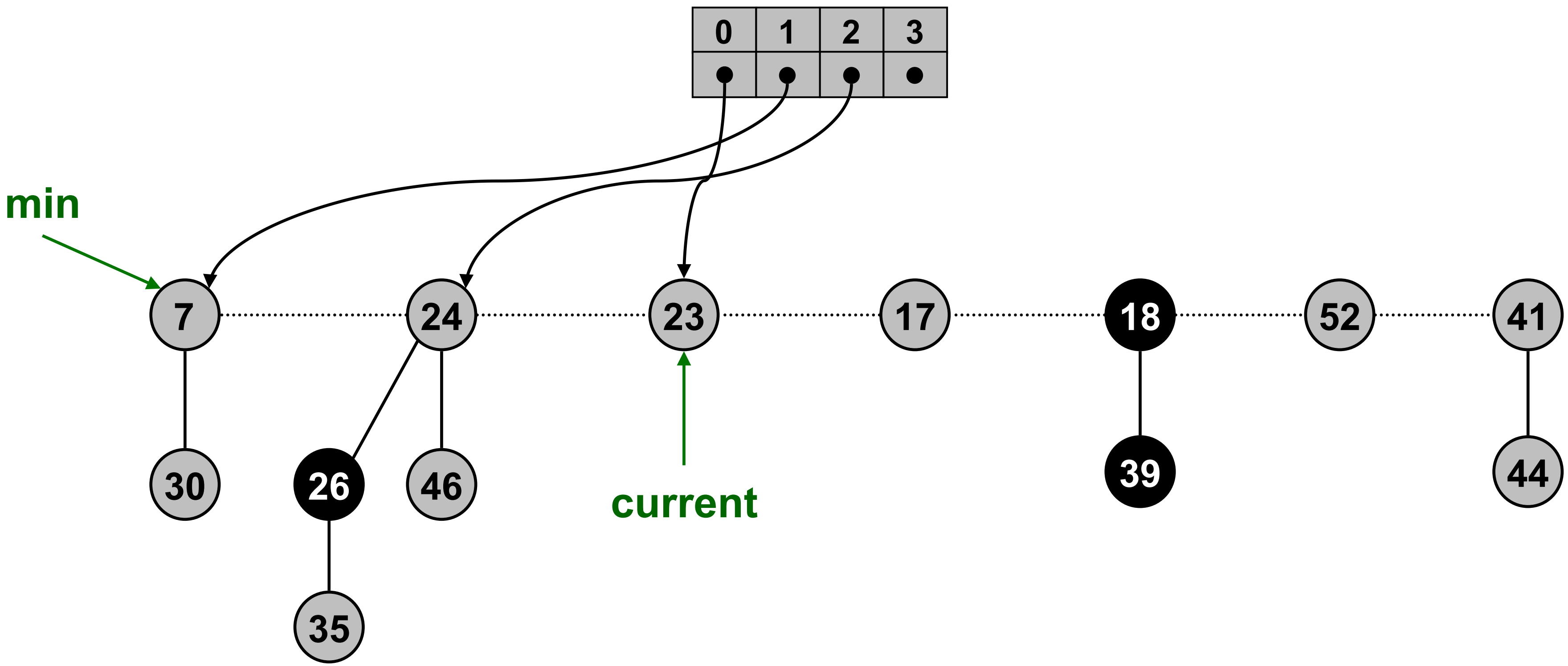
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Delete Min

Delete min.

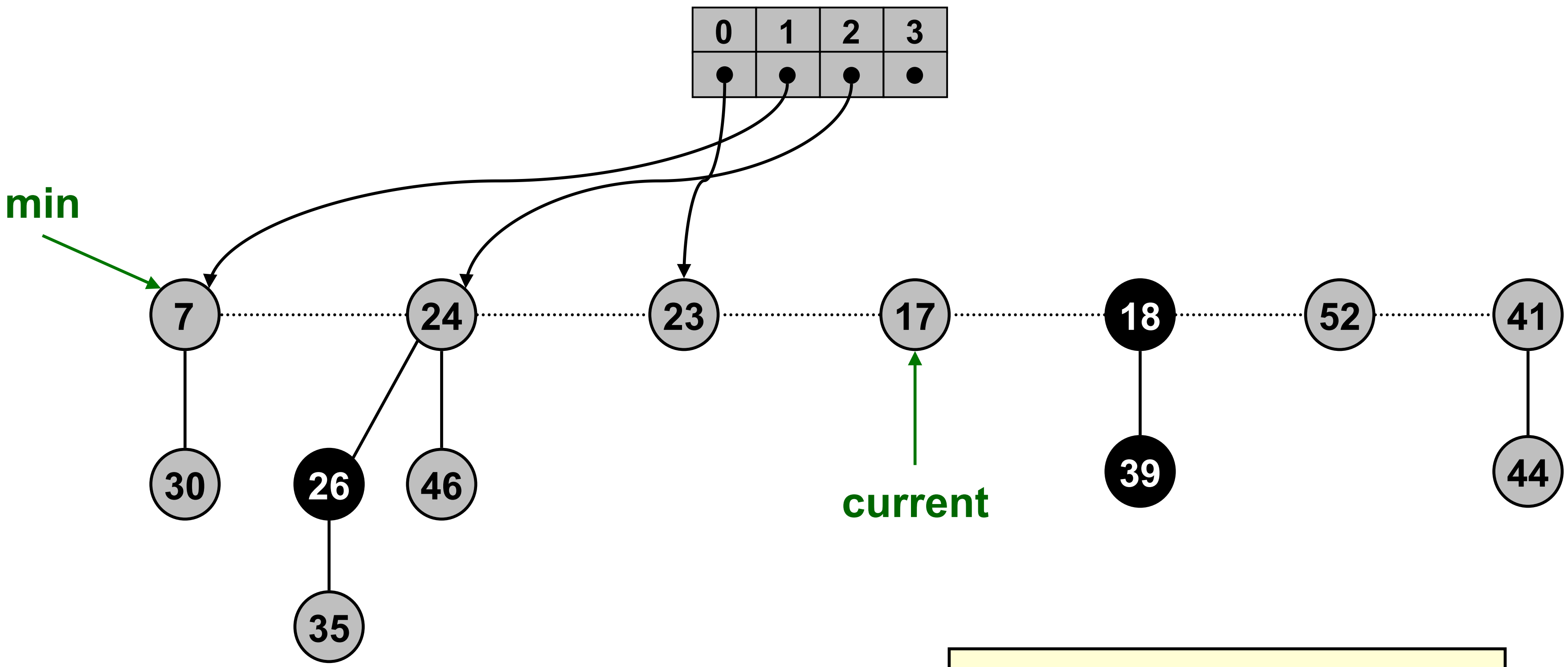
- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**



Fibonacci Heaps: Delete Min

Delete min.

- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**

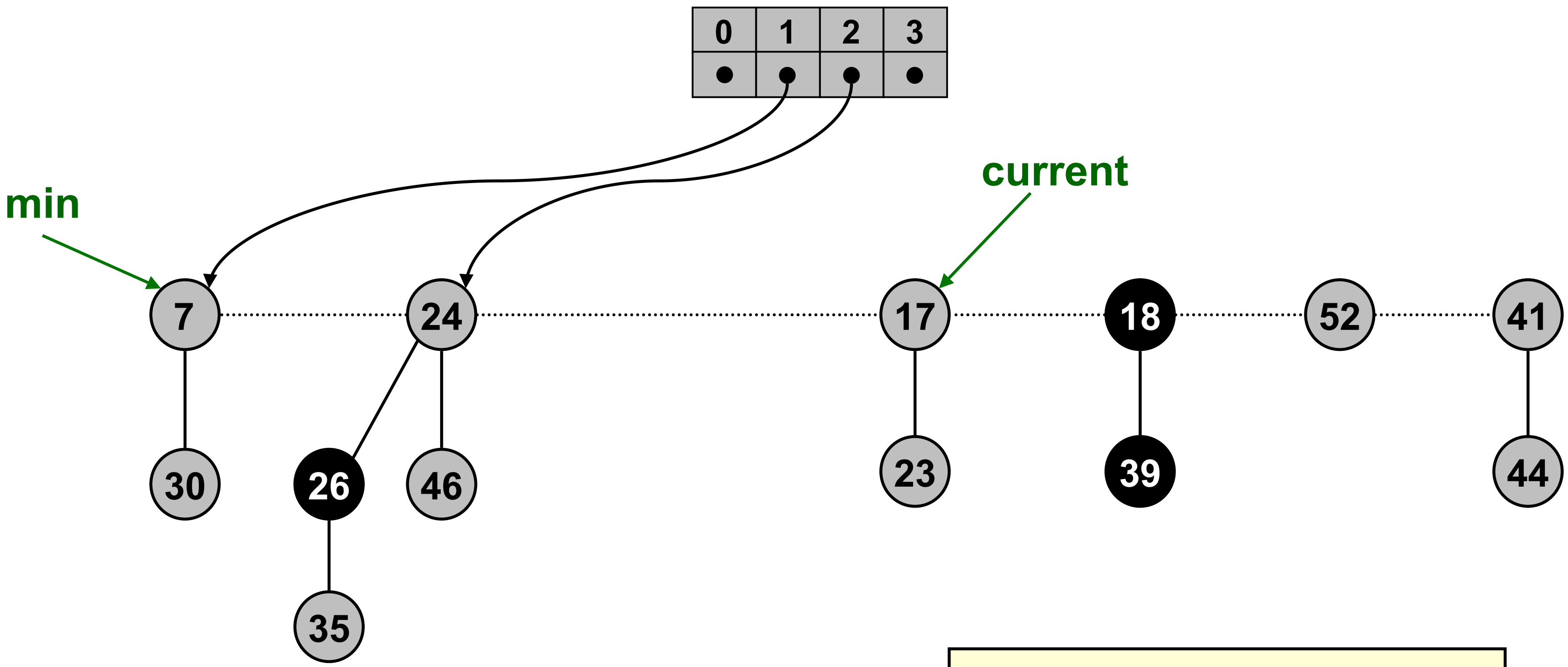


Merge 17 and 23 trees.

Fibonacci Heaps: Delete Min

Delete min.

- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**

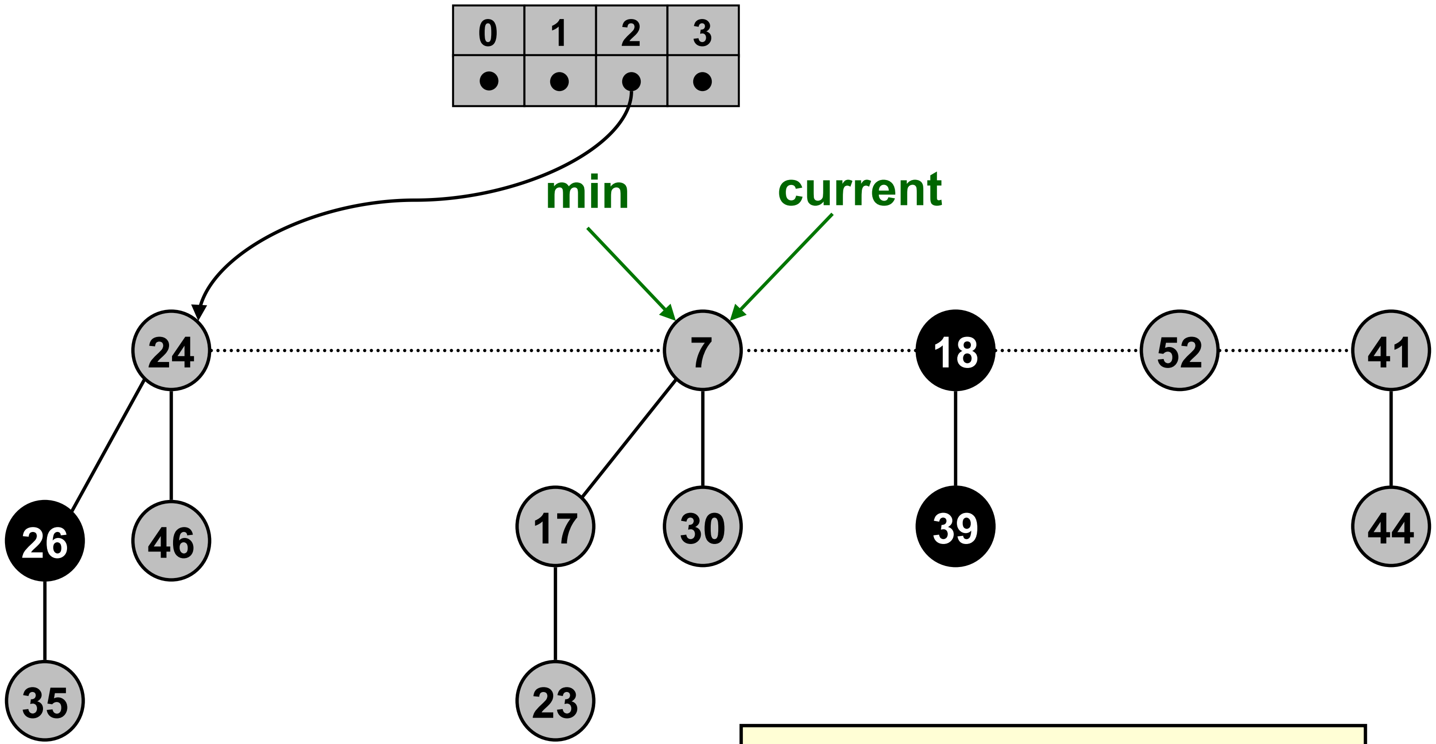


Merge 7 and 17 trees.

Fibonacci Heaps: Delete Min

Delete min.

- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.

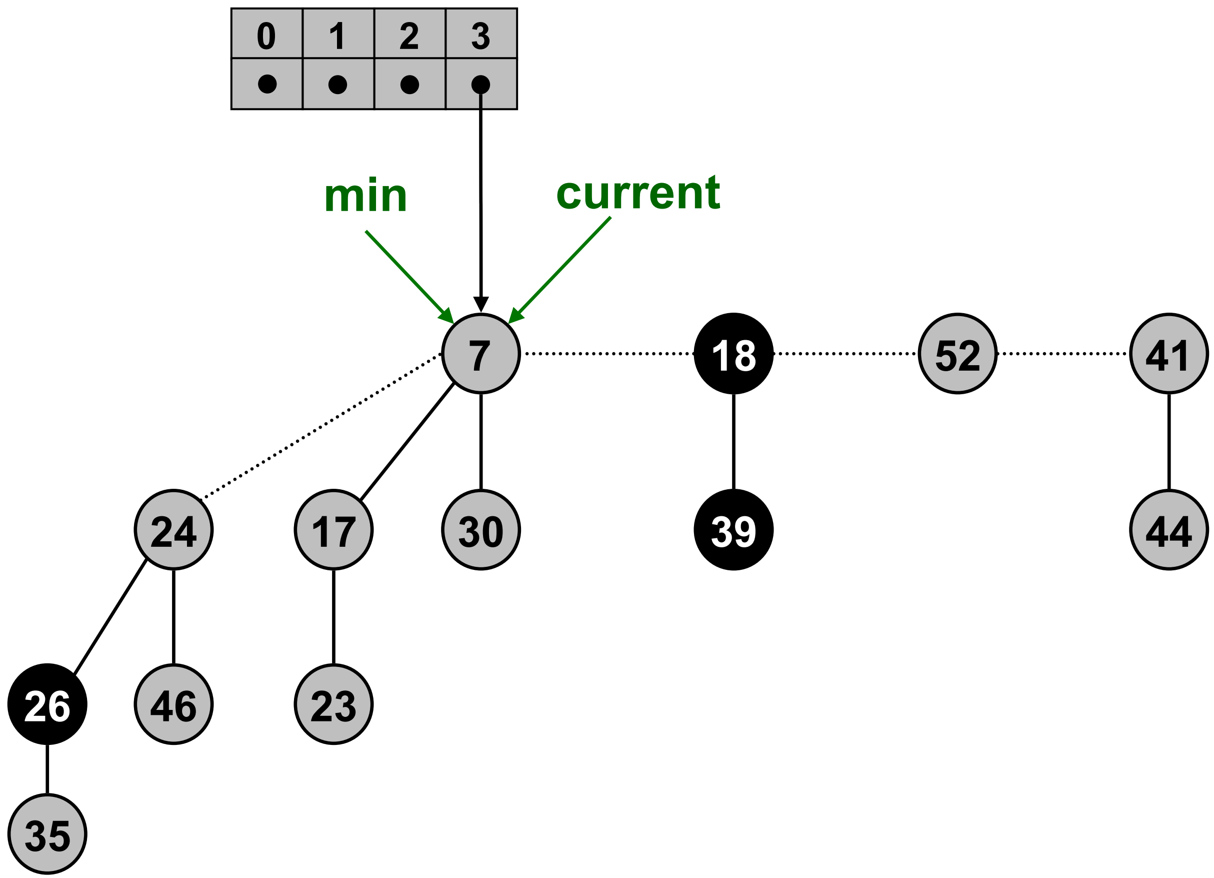


Merge 7 and 24 trees.

Fibonacci Heaps: Delete Min

Delete min.

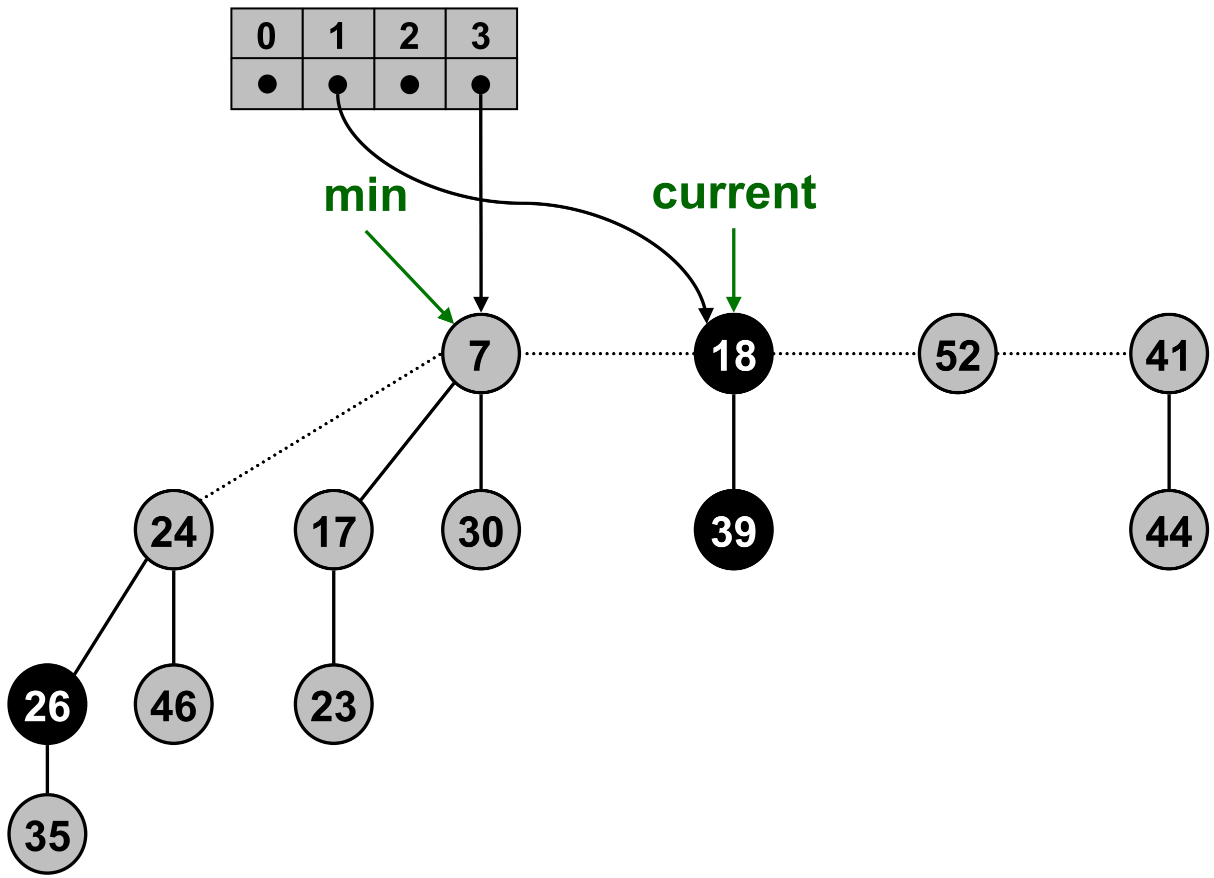
- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**



Fibonacci Heaps: Delete Min

Delete min.

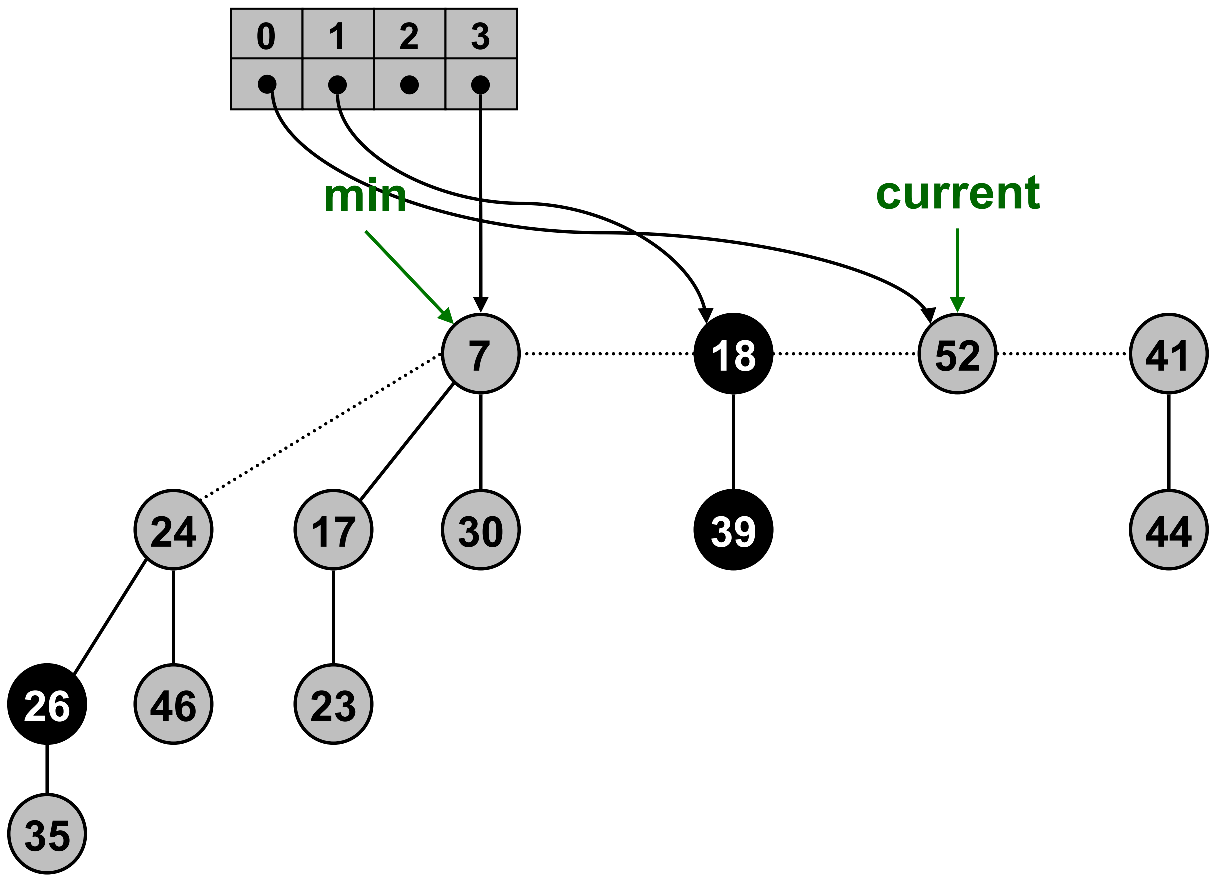
- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**



Fibonacci Heaps: Delete Min

Delete min.

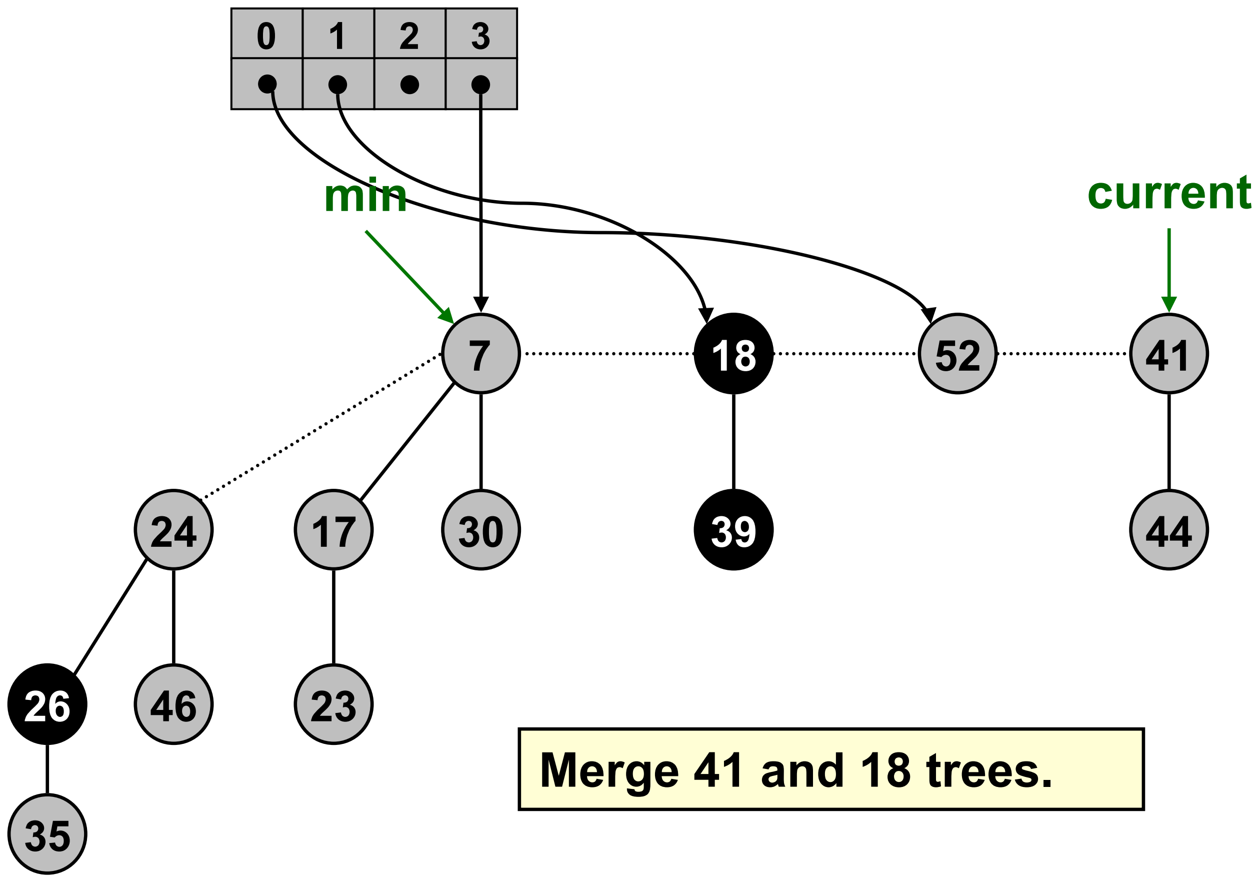
- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**



Fibonacci Heaps: Delete Min

Delete min.

- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**

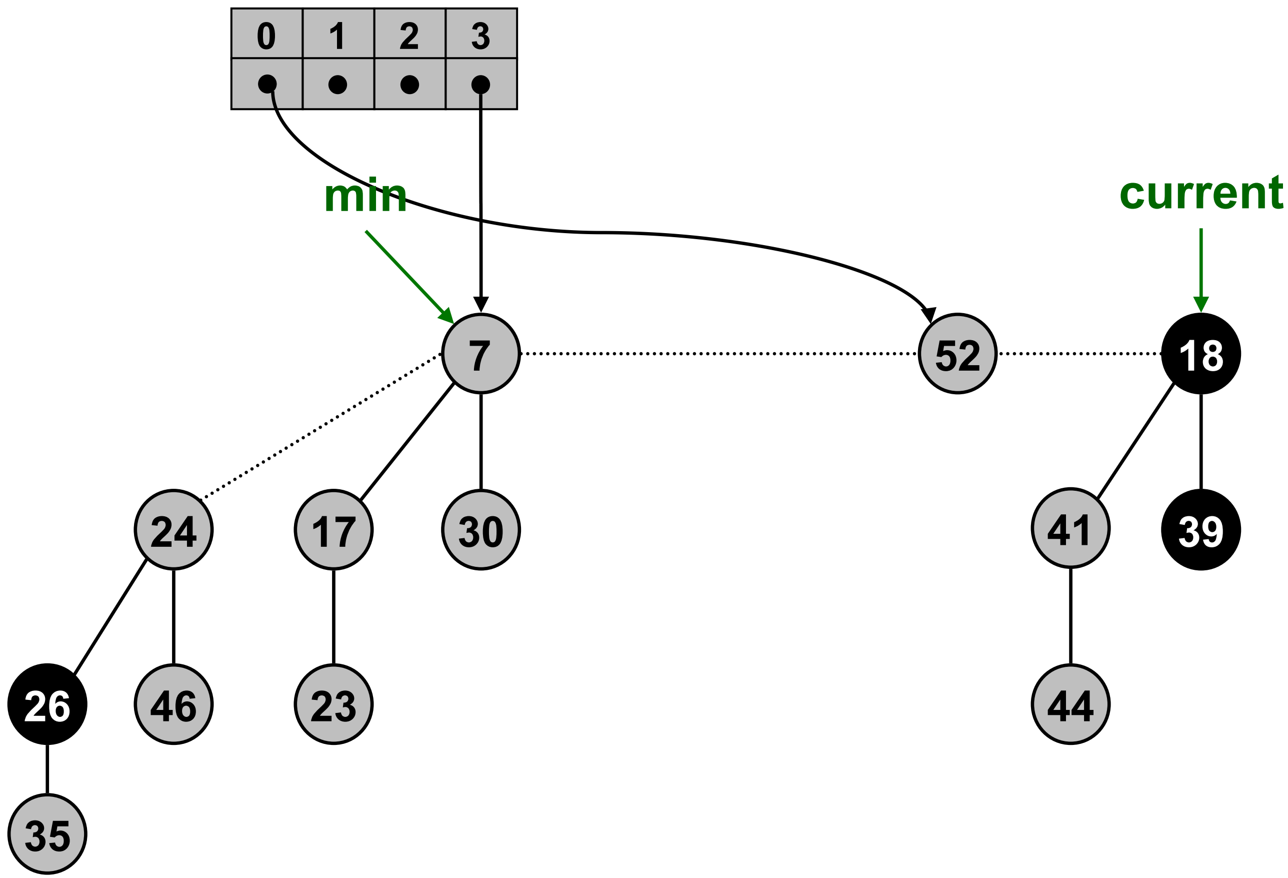


Merge 41 and 18 trees.

Fibonacci Heaps: Delete Min

Delete min.

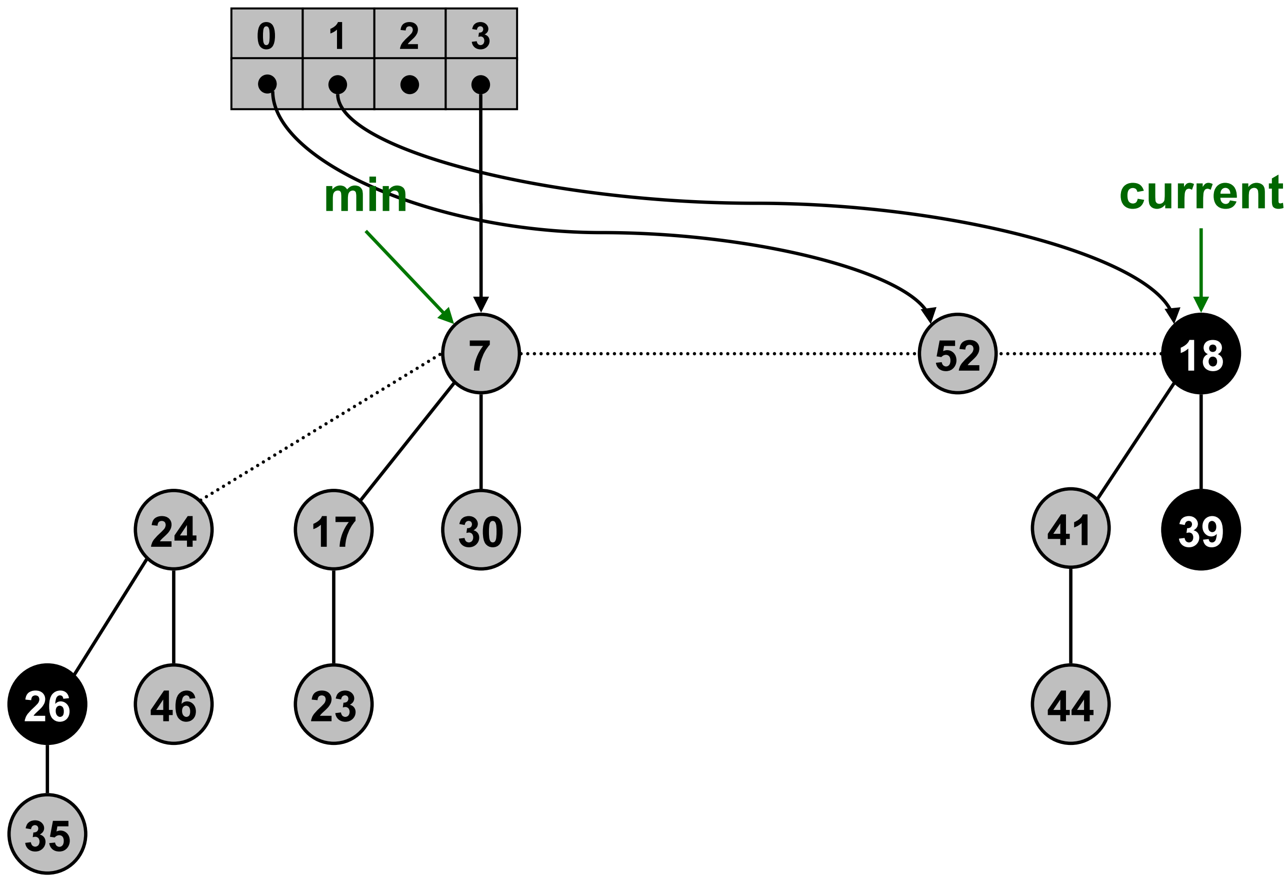
- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**



Fibonacci Heaps: Delete Min

Delete min.

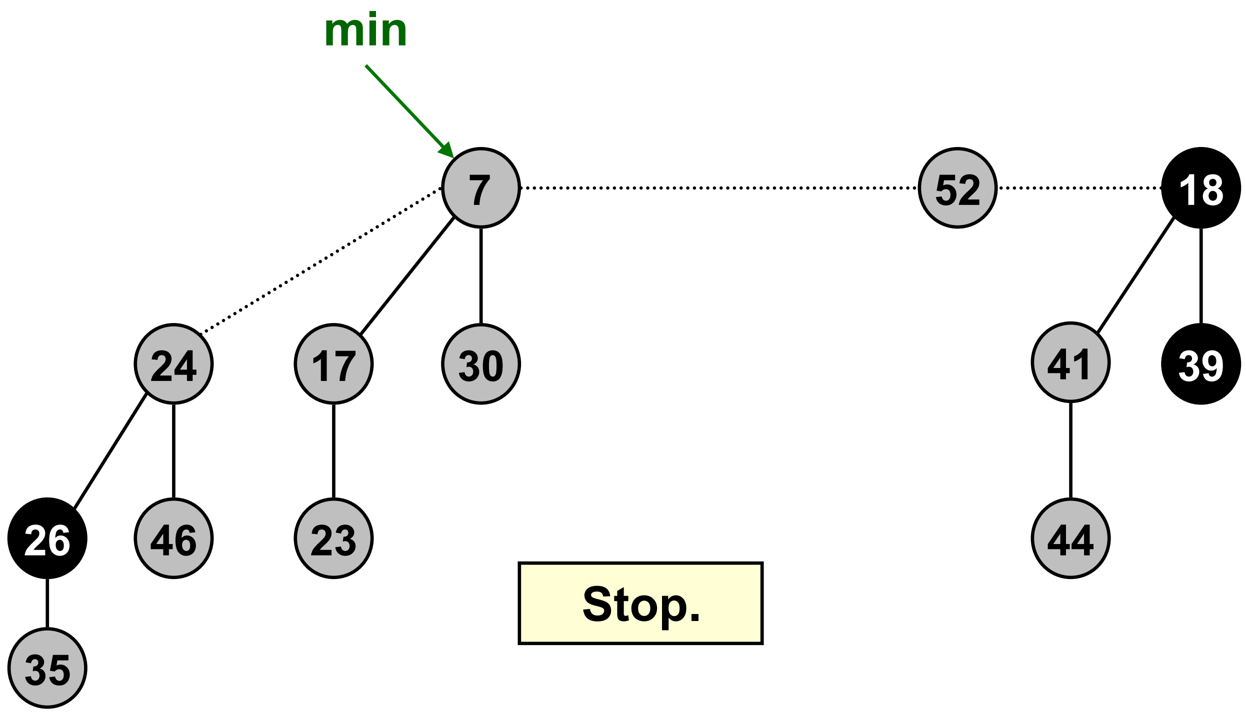
- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**



Fibonacci Heaps: Delete Min

Delete min.

- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Delete Min Analysis

Notation.

- $D(n)$ = max degree of any node in Fibonacci heap with n nodes.
- $t(H)$ = # trees in heap H.
- $\Phi(H) = t(H) + 2m(H)$.

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

- $O(D(n))$ work adding min's children into root list and updating min.
  - at most $D(n)$ children of min node
- $O(D(n) + t(H))$ work consolidating trees.
  - work is proportional to size of root list since number of roots decreases by one after each merging
  - $\leq D(n) + t(H) - 1$ root nodes at beginning of consolidation

Amortized cost. $O(D(n))$

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

Fibonacci Heaps: Delete Min Analysis

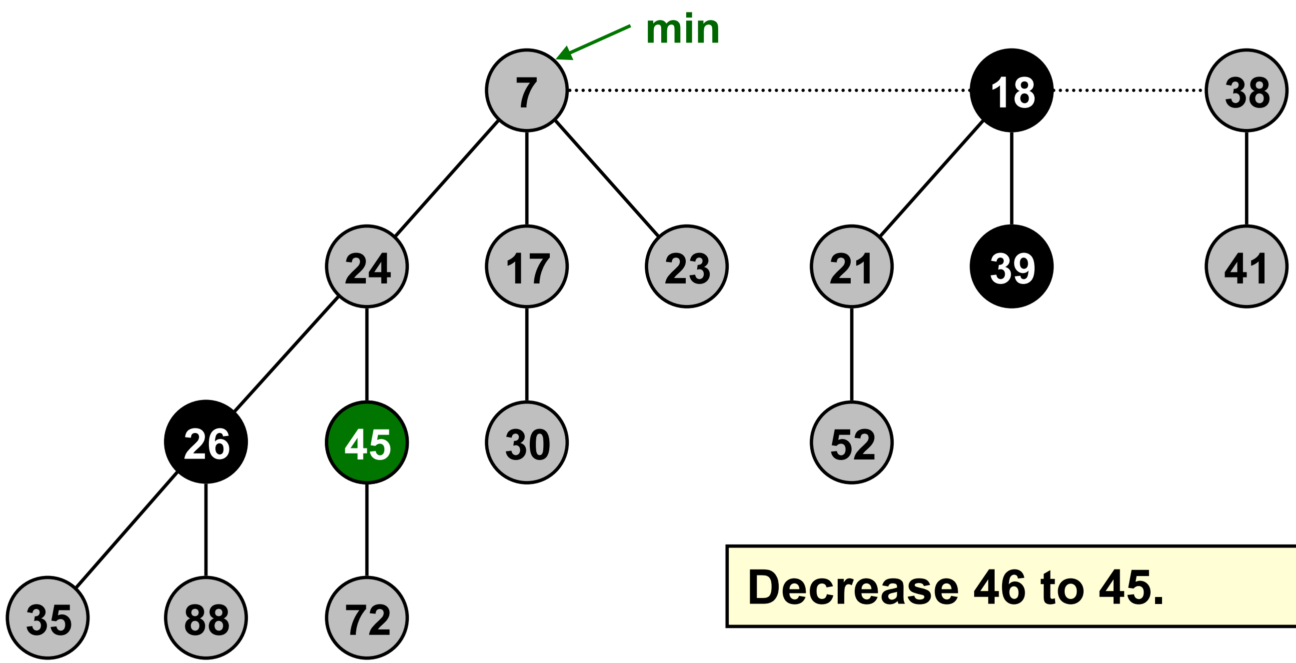
Is amortized cost of $O(D(n))$ good?

- **Yes, if only Insert, Delete-min, and Union operations supported.**
  - **in this case, Fibonacci heap contains only binomial trees since we only merge trees of equal root degree**
  - **this implies $D(n) \leq \lfloor \log_2 N \rfloor$**
- **Yes, if we support Decrease-key in clever way.**
  - **we'll show that $D(n) \leq \lfloor \log_\phi N \rfloor$, where ϕ is golden ratio**
  - **$\phi^2 = 1 + \phi$**
  - **$\phi = (1 + \sqrt{5}) / 2 = 1.618\ldots$**
  - **limiting ratio between successive Fibonacci numbers!**

Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

- **Case 0: min-heap property not violated.**
  - **decrease key of x to k**
  - **change heap min pointer if necessary**

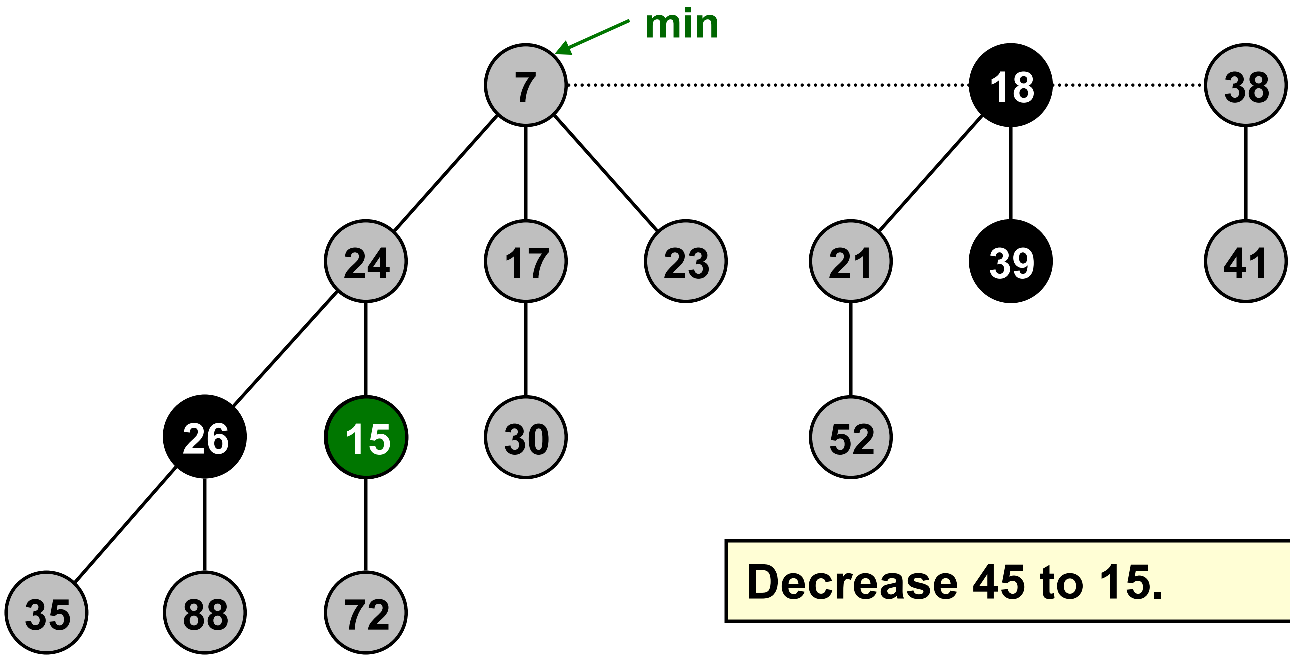


Decrease 46 to 45.

Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

- Case 1: parent of x is unmarked.
  - decrease key of x to k
  - cut off link between x and its parent
  - mark parent
  - add tree rooted at x to root list, updating heap min pointer

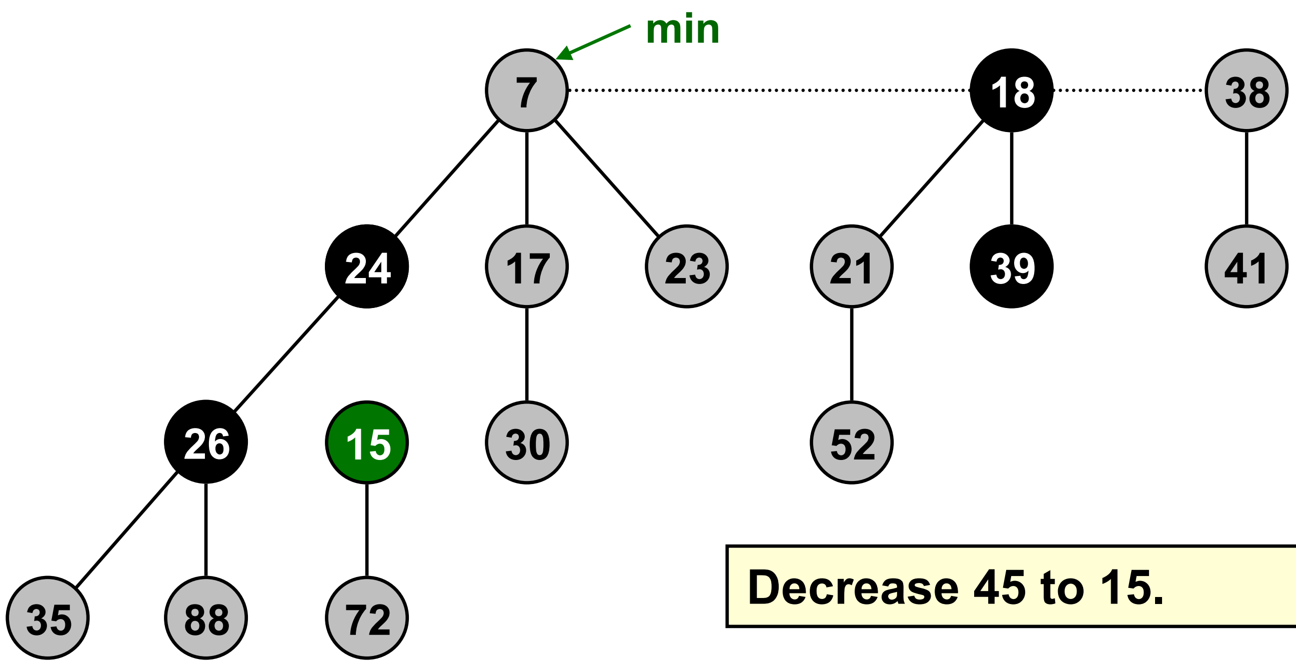


Decrease 45 to 15.

Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

- Case 1: parent of x is unmarked.
  - decrease key of x to k
  - cut off link between x and its parent
  - mark parent
  - add tree rooted at x to root list, updating heap min pointer

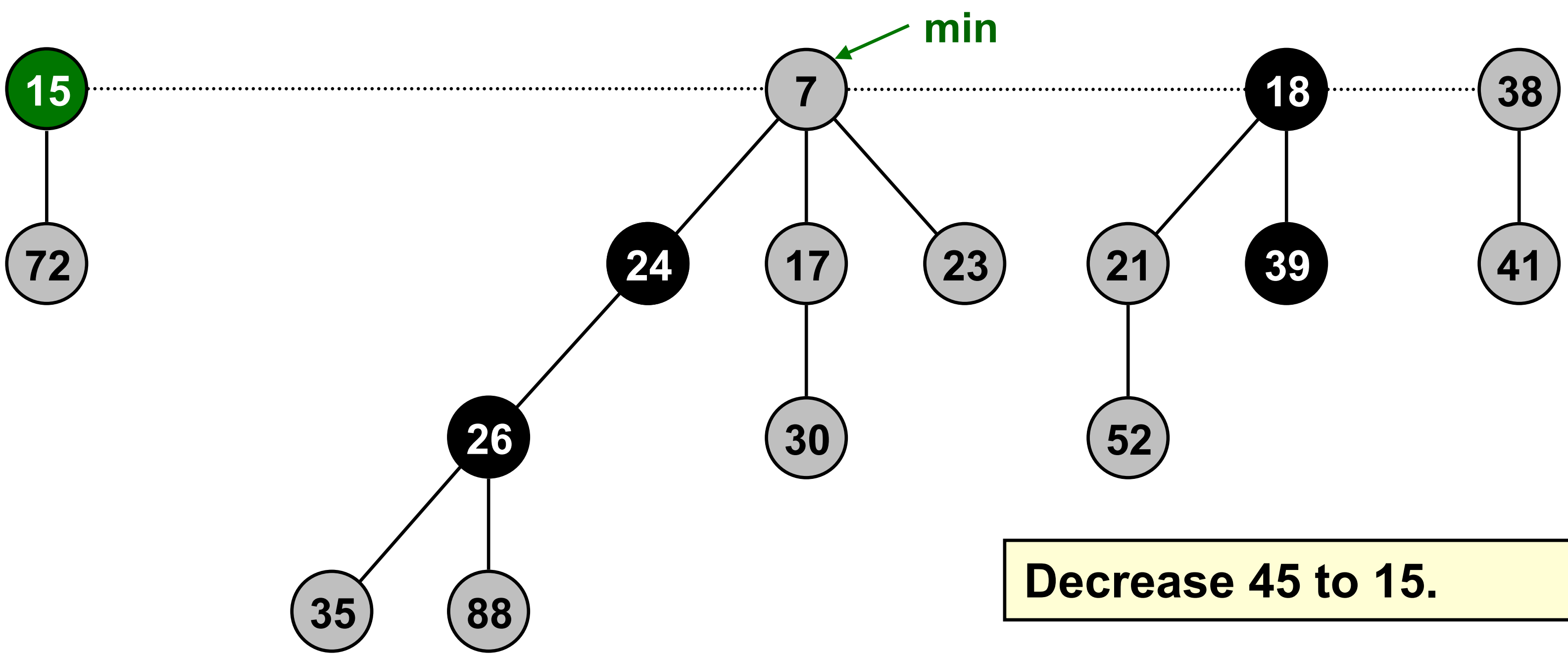


Decrease 45 to 15.

Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

- Case 1: parent of x is unmarked.
  - decrease key of x to k
  - cut off link between x and its parent
  - mark parent
  - add tree rooted at x to root list, updating heap min pointer

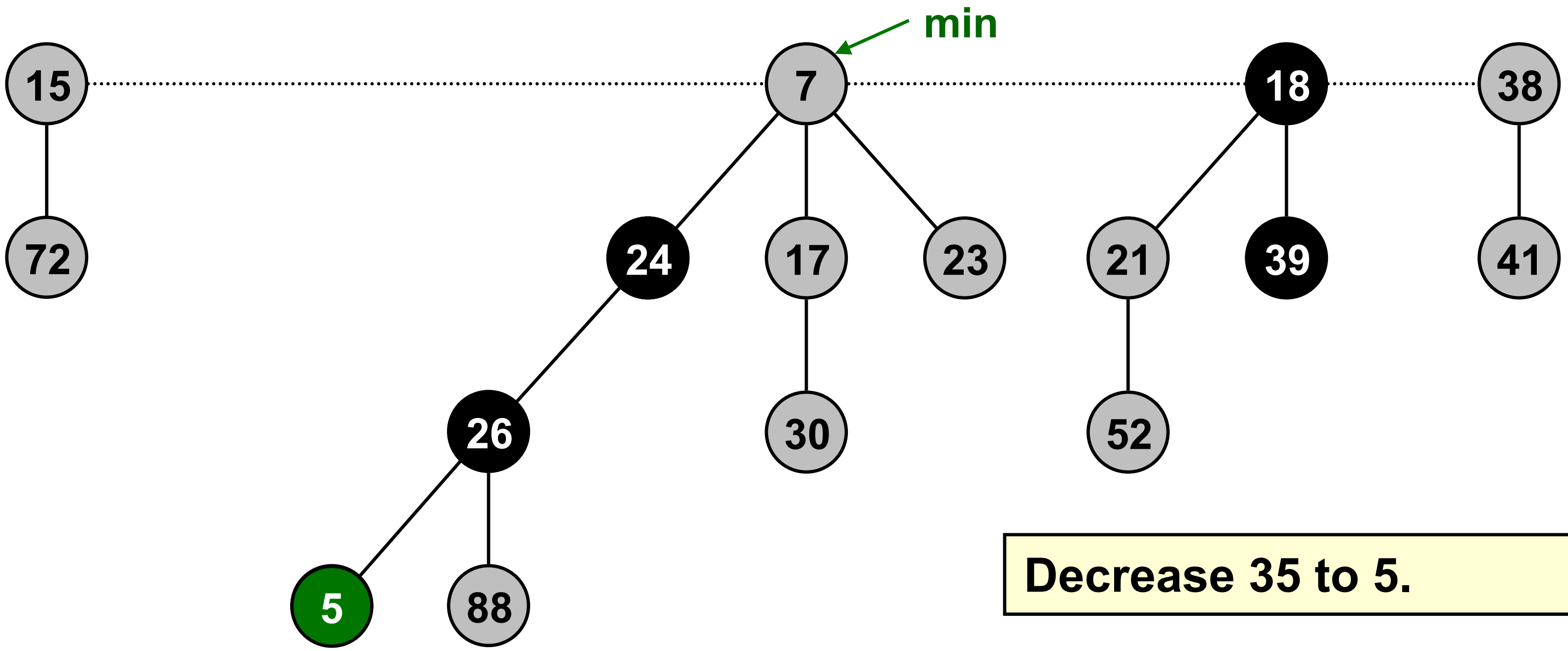


Decrease 45 to 15.

Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

- **Case 2: parent of x is marked.**
  - decrease key of x to k
  - cut off link between x and its parent p[x], and add x to root list
  - cut off link between p[x] and p[p[x]], add p[x] to root list
    - If p[p[x]] unmarked, then mark it.
    - If p[p[x]] marked, cut off p[p[x]], unmark, and repeat.

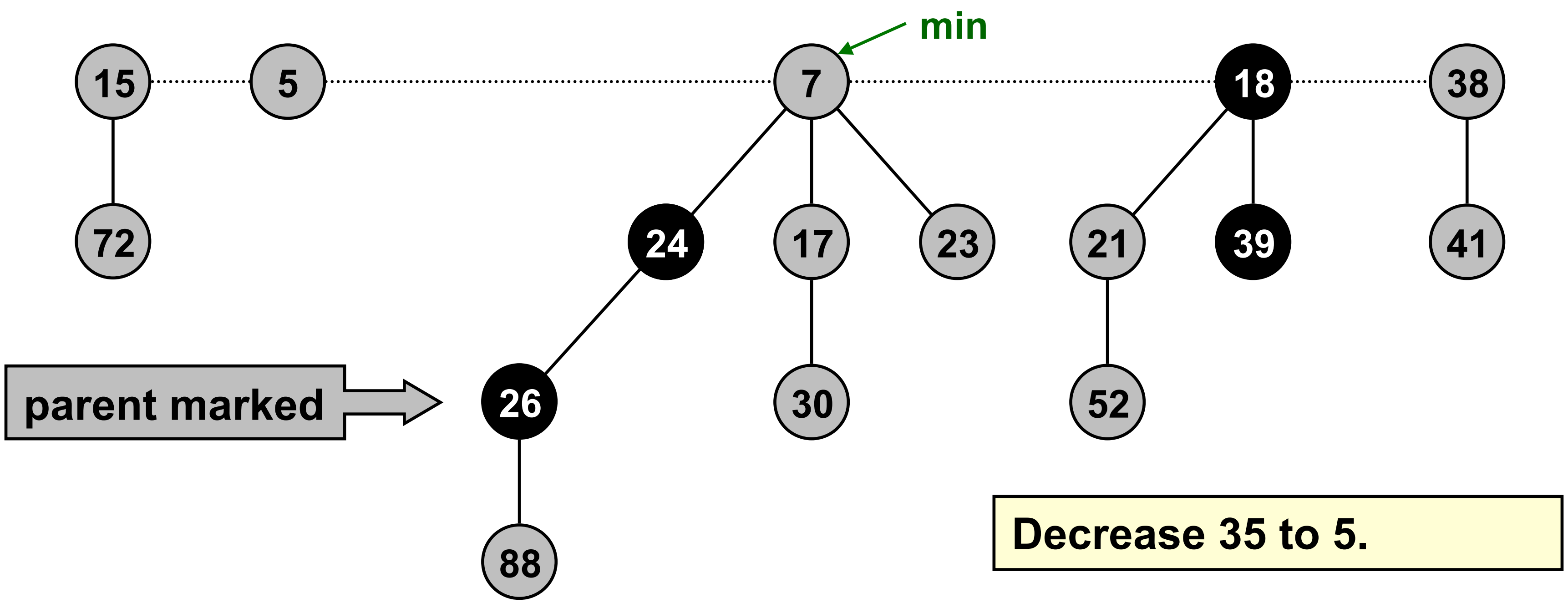


Decrease 35 to 5.

Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

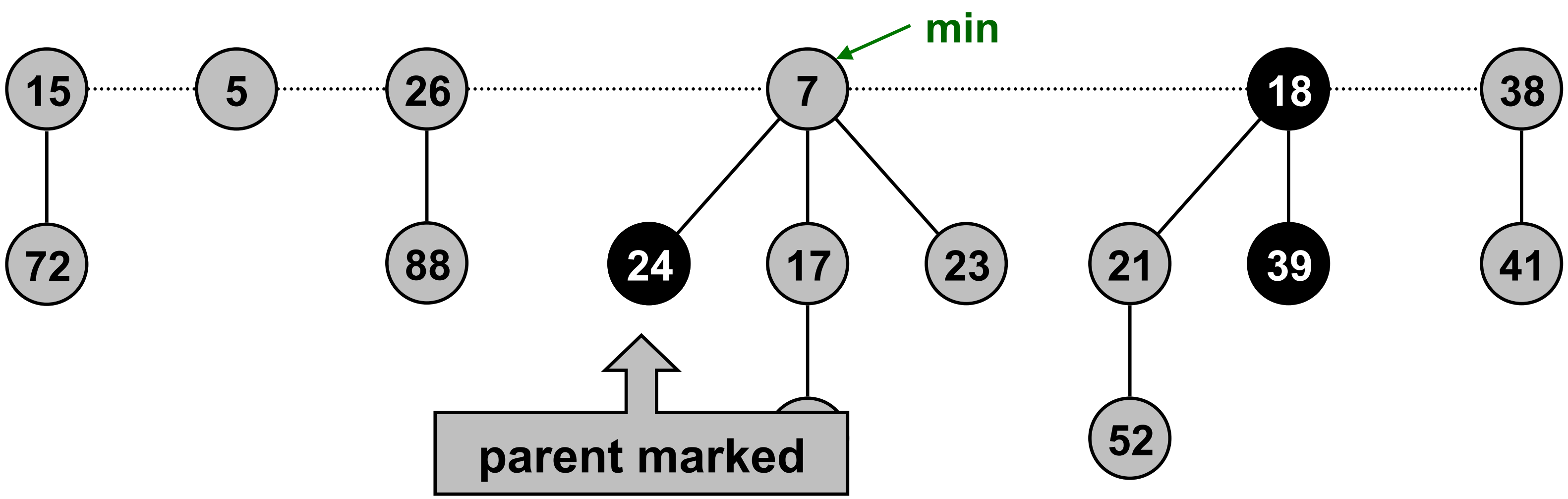
- Case 2: parent of x is marked.
  - decrease key of x to k
  - cut off link between x and its parent p[x], and add x to root list
  - cut off link between p[x] and p[p[x]], add p[x] to root list
    - If p[p[x]] unmarked, then mark it.
    - If p[p[x]] marked, cut off p[p[x]], unmark, and repeat.



Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

- Case 2: parent of x is marked.
  - decrease key of x to k
  - cut off link between x and its parent p[x], and add x to root list
  - cut off link between p[x] and p[p[x]], add p[x] to root list
    - If p[p[x]] unmarked, then mark it.
    - If p[p[x]] marked, cut off p[p[x]], unmark, and repeat.

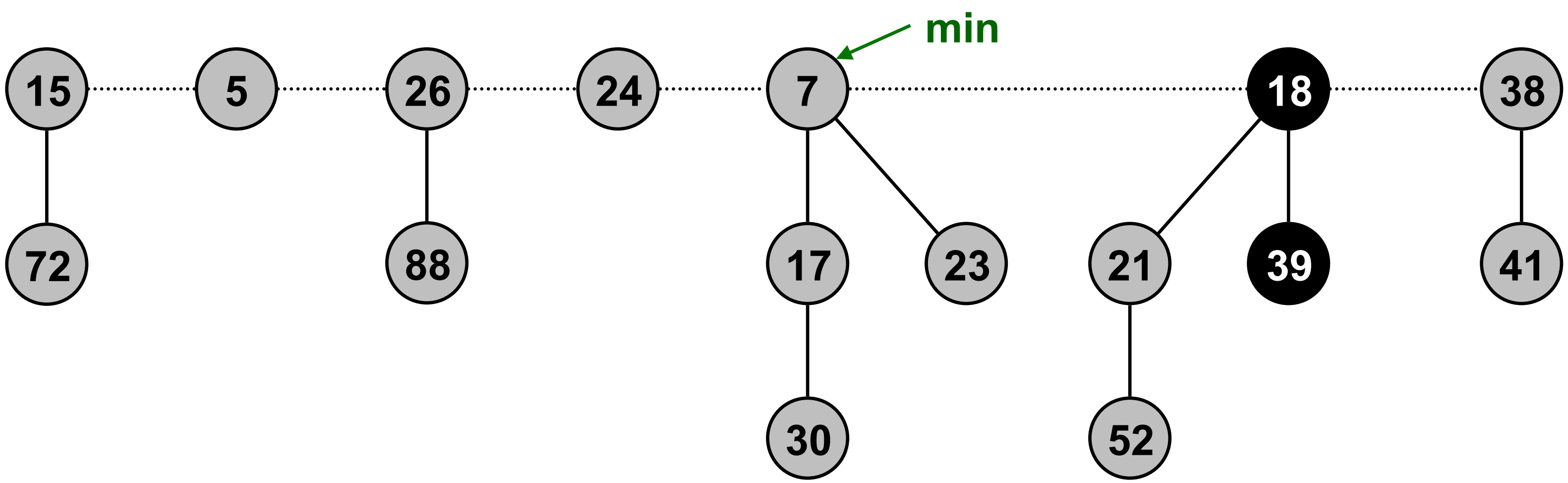


Decrease 35 to 5.

Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

- Case 2: parent of x is marked.
  - decrease key of x to k
  - cut off link between x and its parent p[x], and add x to root list
  - cut off link between p[x] and p[p[x]], add p[x] to root list
    - ✐ If p[p[x]] unmarked, then mark it.
    - ✐ If p[p[x]] marked, cut off p[p[x]], unmark, and repeat.



Decrease 35 to 5.

Fibonacci Heaps: Decrease Key Analysis

Notation.

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

Actual cost. O(c)

- O(1) time for decrease key.
- O(1) time for each of c cascading cuts, plus reinserting in root list.

Amortized cost. O(1)

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

Mark Field

Note: mark field is used as follows:

Mark is true if

1. at some time x was a root

2. Then x was linked to another node

And one child of x has been cut

If two children are removed, then x becomes a root itself.

Fibonacci Heaps: Delete

Delete node x.

- Decrease key of x to $-\infty$.
- 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.

Fibonacci Heaps: Bounding Max Degree

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

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

Corollary. Delete and Delete-min take O(log N) amortized time.

Lemma. Let x be a node with degree k, and let $y_1, \ldots, y_k$ denote the children of x in the order in which they were linked to x. Then:

$$\text{degree}(y_i) \ge \begin{cases} 0 & \text{if } i = 1 \\ i - 2 & \text{if } i \ge 1 \end{cases}$$

Proof.

- When y_i is linked to x, $y_1, \ldots, y_{i-1}$ already linked to x,
  $\Rightarrow$ degree(x) = i - 1
  $\Rightarrow$ degree(y_i) = i - 1 since we only link nodes of equal degree
- Since then, y_i has lost at most one child
  – otherwise it would have been cut from x
- Thus, degree(y_i) = i - 1 or i - 2

Fibonacci Heaps: Bounding Max Degree

Key lemma. In a Fibonacci heap with N nodes, the maximum degree of any node is at most $\log_\phi N$, where $\phi = (1 + \sqrt{5}) / 2$.

Proof of key lemma.

- For any node x, we show that $size(x) \geq \phi^{degree(x)}$.
  - size(x) = # node in subtree rooted at x
  - taking base ϕ logs, $degree(x) \leq \log_\phi (size(x)) \leq \log_\phi N$.
- Let s_k be min size of tree rooted at any degree k node.
  - trivial to see that $s_0 = 1$, $s_1 = 2$
  - s_k monotonically increases with k
- Let x* be a degree k node of size s_k, and let $y_1, \ldots, y_k$ be children in order that they were linked to x*.

Assume $k \geq 2$

$$
\begin{aligned}
s_k &= \text{size}(x^*) \\
&= 2 + \sum_{i=2}^{k} size(y_i) \\
&\geq 2 + \sum_{i=2}^{k} s_{\deg[y_i]} \\
&\geq 2 + \sum_{i=2}^{k} s_{i-2} \\
&= 2 + \sum_{i=0}^{k-2} s_i
\end{aligned}
$$

Fibonacci Facts

Definition. The Fibonacci sequence is:

$$F_k = \begin{cases} 1 & \text{if } k = 0 \\ 2 & \text{if } k = 1 \\ F_{k-1} + F_{k-2} & \text{if } k \geq 2 \end{cases}$$

- 1, 2, 3, 5, 8, 13, 21, . . .
- Slightly nonstandard definition.

Fact F1. $F_k \geq \phi^k$, where $\phi = (1 + \sqrt{5}) / 2 = 1.618\ldots$

Fact F2. For $k \geq 2$, $F_k = 2 + \sum_{i=0}^{k-2} F_i$

Consequence. $s_k \geq F_k \geq \phi^k$.

- This implies that size(x) $\geq \phi^{\text{degree(x)}}$ for all nodes x.

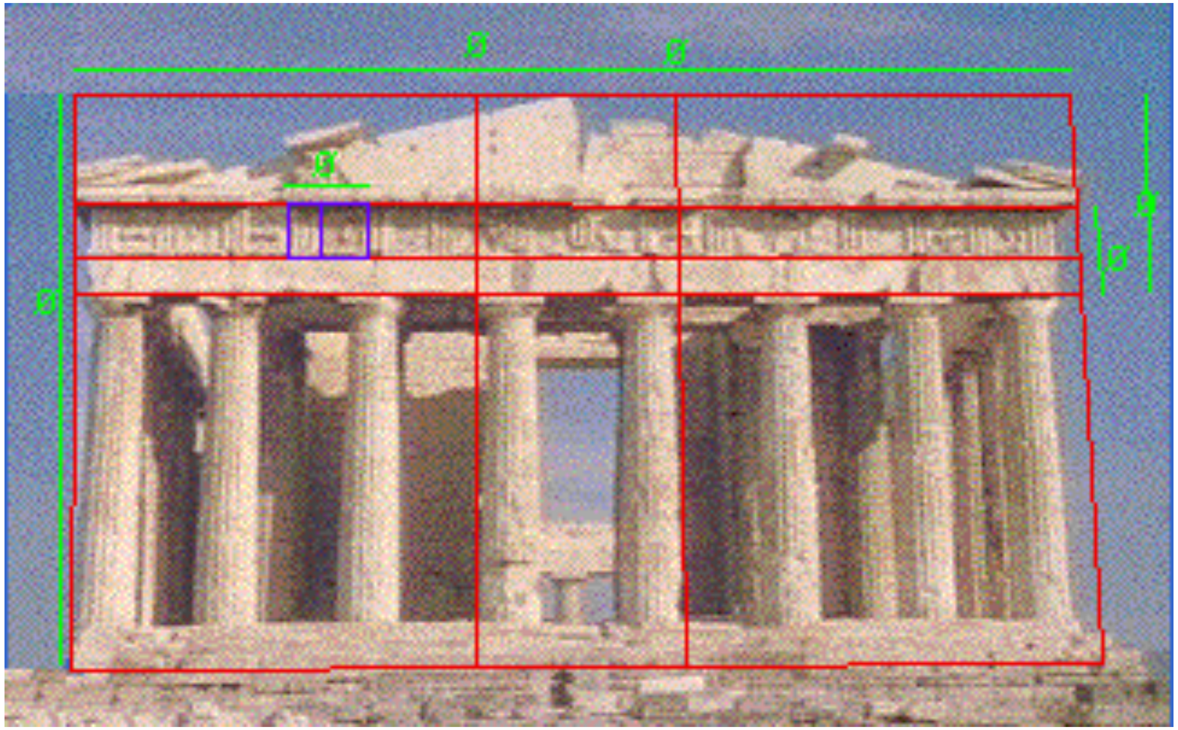
$$\begin{aligned} s_k &= \text{size}\,(x^*) \\ &= 2 + \sum_{i=2}^{k} size(y_i) \\ &\geq 2 + \sum_{i=2}^{k} s_{\deg[y_i]} \\ &\geq 2 + \sum_{i=2}^{k} s_{i-2} \\ &= 2 + \sum_{i=0}^{k-2} s_i \end{aligned}$$

Golden Ratio

Definition. **The Fibonacci sequence is: 1, 2, 3, 5, 8, 13, 21, . . .**

Definition. **The golden ratio $\phi = (1 + \sqrt{5}) / 2 = 1.618\ldots$**

- **Divide a rectangle into a square and smaller rectangle such that the smaller rectangle has the same ratio as original one.**



Parthenon, Athens Greece

Fibonacci Proofs

Fact F1. $F_k \geq \phi^k$.

Proof. (by induction on k)

- **Base cases:**
  - $F_0 = 1$, $F_1 = 2 \geq \phi$.
- **Inductive hypotheses:**
  - $F_k \geq \phi^k$ and $F_{k+1} \geq \phi^{k+1}$

$$
\begin{aligned}
F_{k+2} &= F_k + F_{k+1} \\
&\geq \varphi^k + \varphi^{k+1} \\
&= \varphi^k (1 + \varphi) \\
&= \varphi^k (\varphi^2) \\
&= \varphi^{k+2}
\end{aligned}
$$

$\phi^2 = \phi + 1$

Fact F2. For $k \geq 2$, $F_k = 2 + \sum_{i=0}^{k-2} F_i$

Proof. (by induction on k)

- **Base cases:**
  - $F_2 = 3$, $F_3 = 5$
- **Inductive hypotheses:**

$$F_k = 2 + \sum_{i=0}^{k-2} F_i$$

$$
\begin{aligned}
F_{k+2} &= F_k + F_{k+1} \\
&= 2 + \sum_{i=0}^{k-2} F_i + F_{k+1} \\
&= 2 + \sum_{i=0}^{k} F_k
\end{aligned}
$$