Fibonacci Heaps

These lecture slides are adapted from CLRS, Chapter 20.

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.**

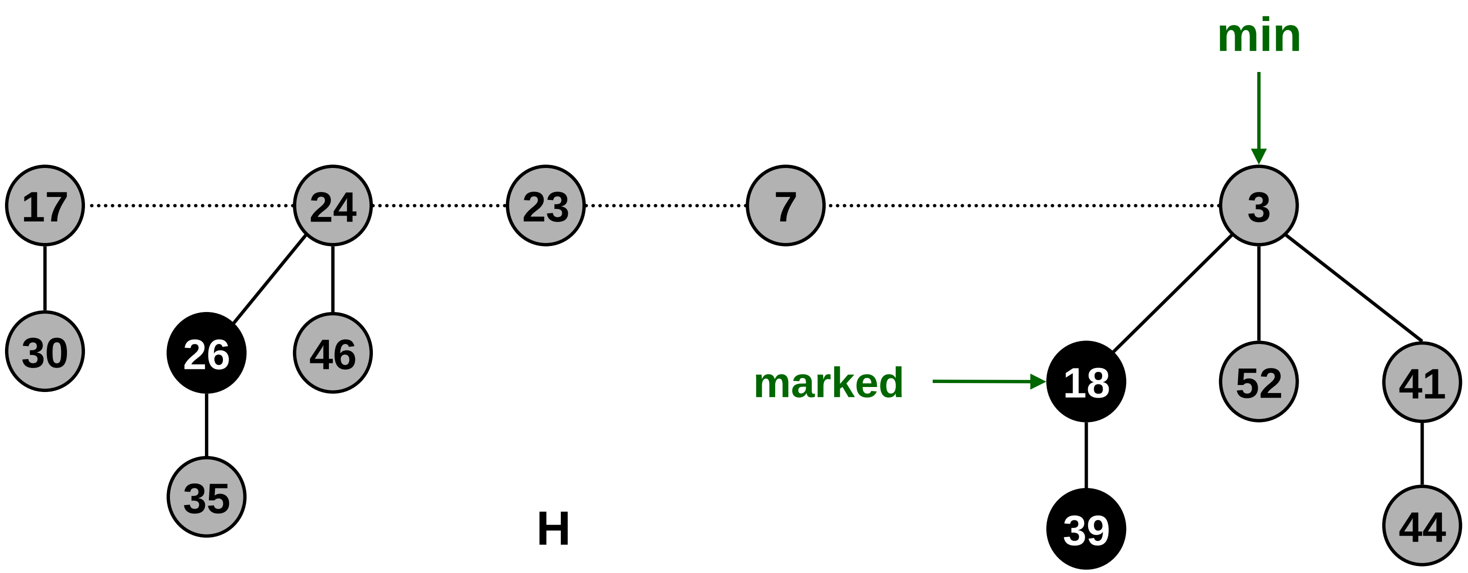
Fibonacci heap intuition.

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

Fibonacci Heaps: Structure

Fibonacci heap.

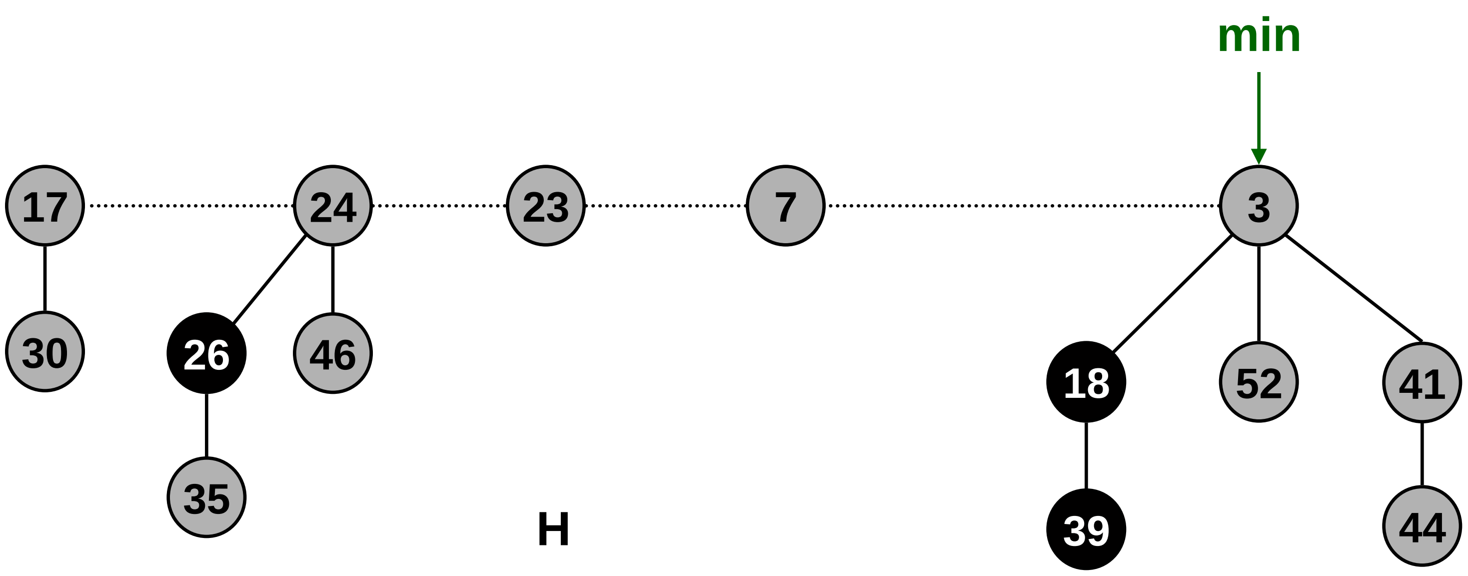
- **Set of min-heap ordered trees.**



Fibonacci Heaps: Implementation

Implementation.

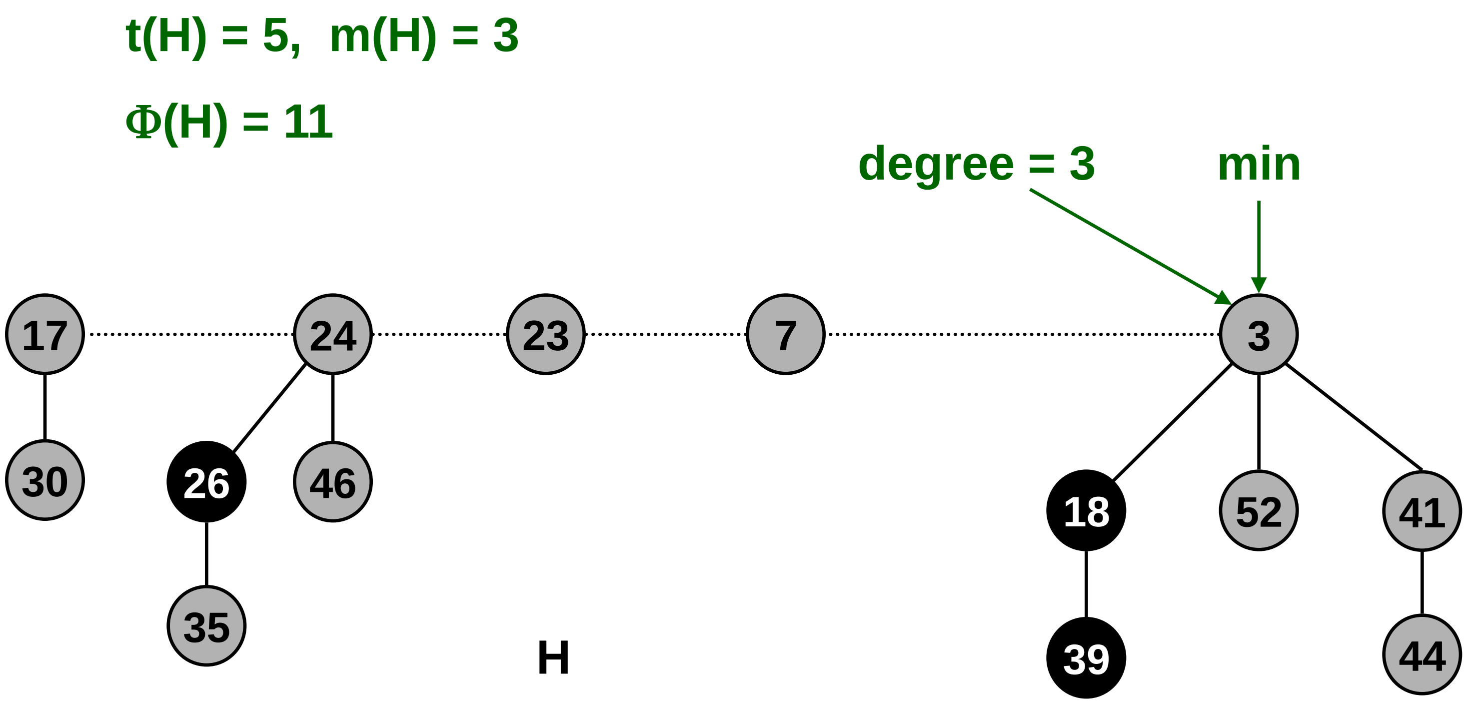
- Represent trees using left-child, right sibling pointers and circular, doubly linked list.
  – can quickly splice off subtrees
- Roots of trees connected with circular doubly linked list.
  – fast union
- Pointer to root of tree with min element.
  – fast find-min



Fibonacci Heaps: Potential Function

Key quantities.

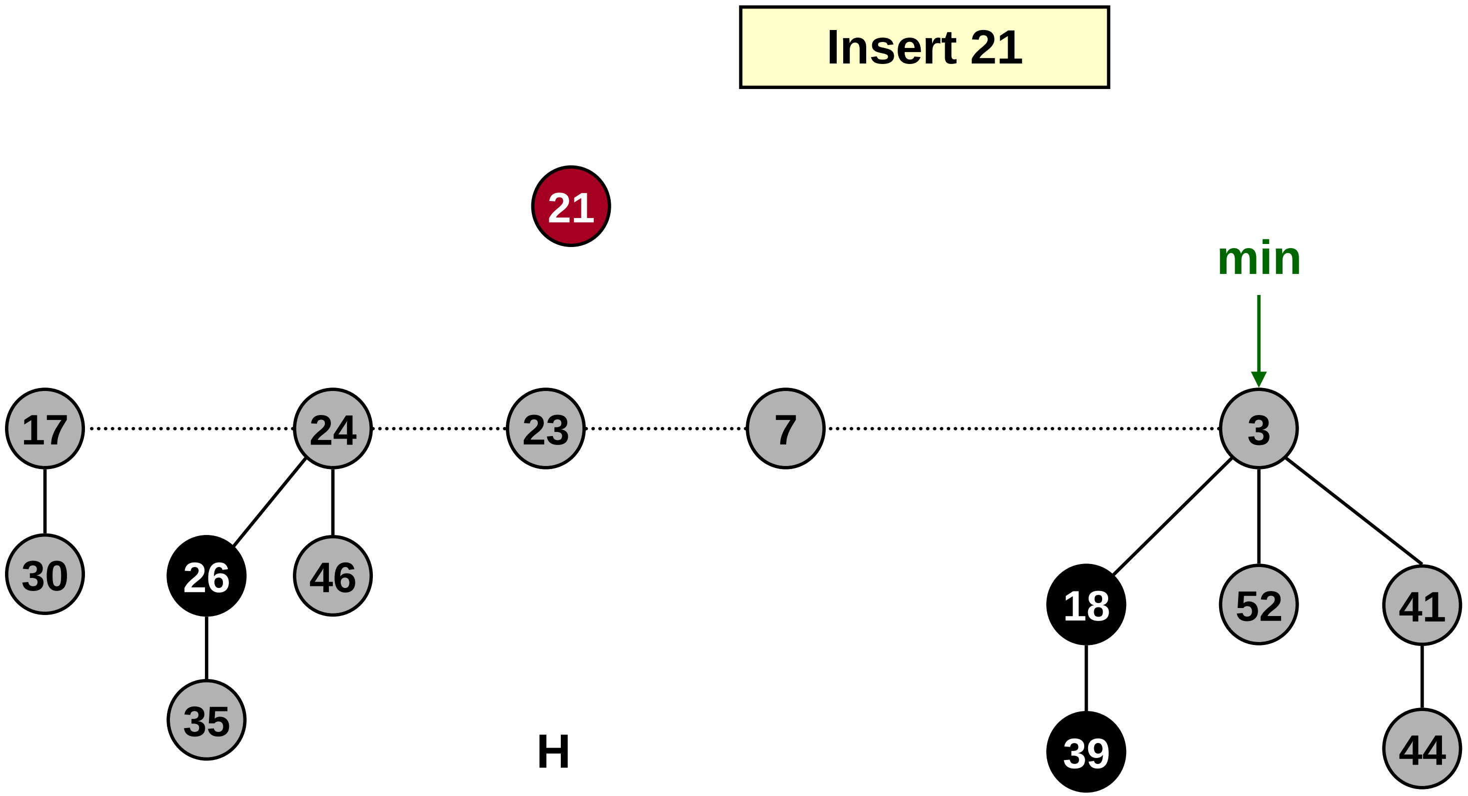
- Degree[x] = degree of node x.
- Mark[x] = mark of node x (black or gray).
- t(H) = # trees.
- m(H) = # marked nodes.
- $\Phi(H) = t(H) + 2m(H)$ = potential function.



Fibonacci Heaps: Insert

Insert.

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

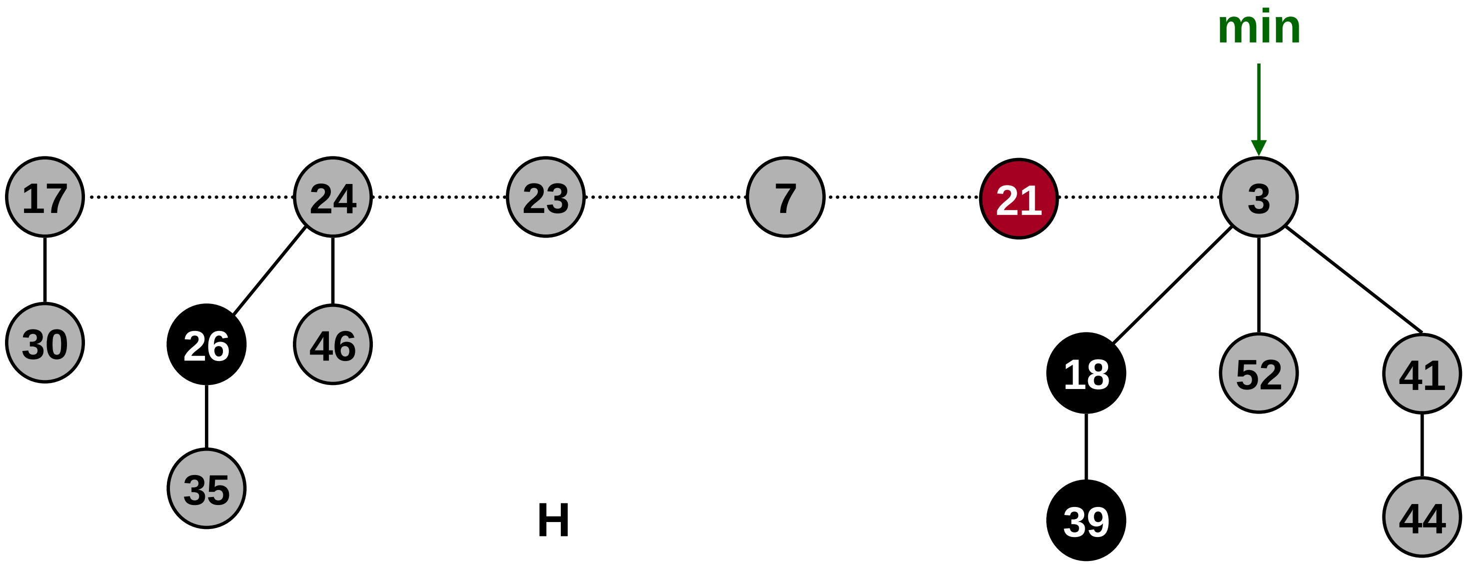


Fibonacci Heaps: Insert

Insert.

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

Insert 21



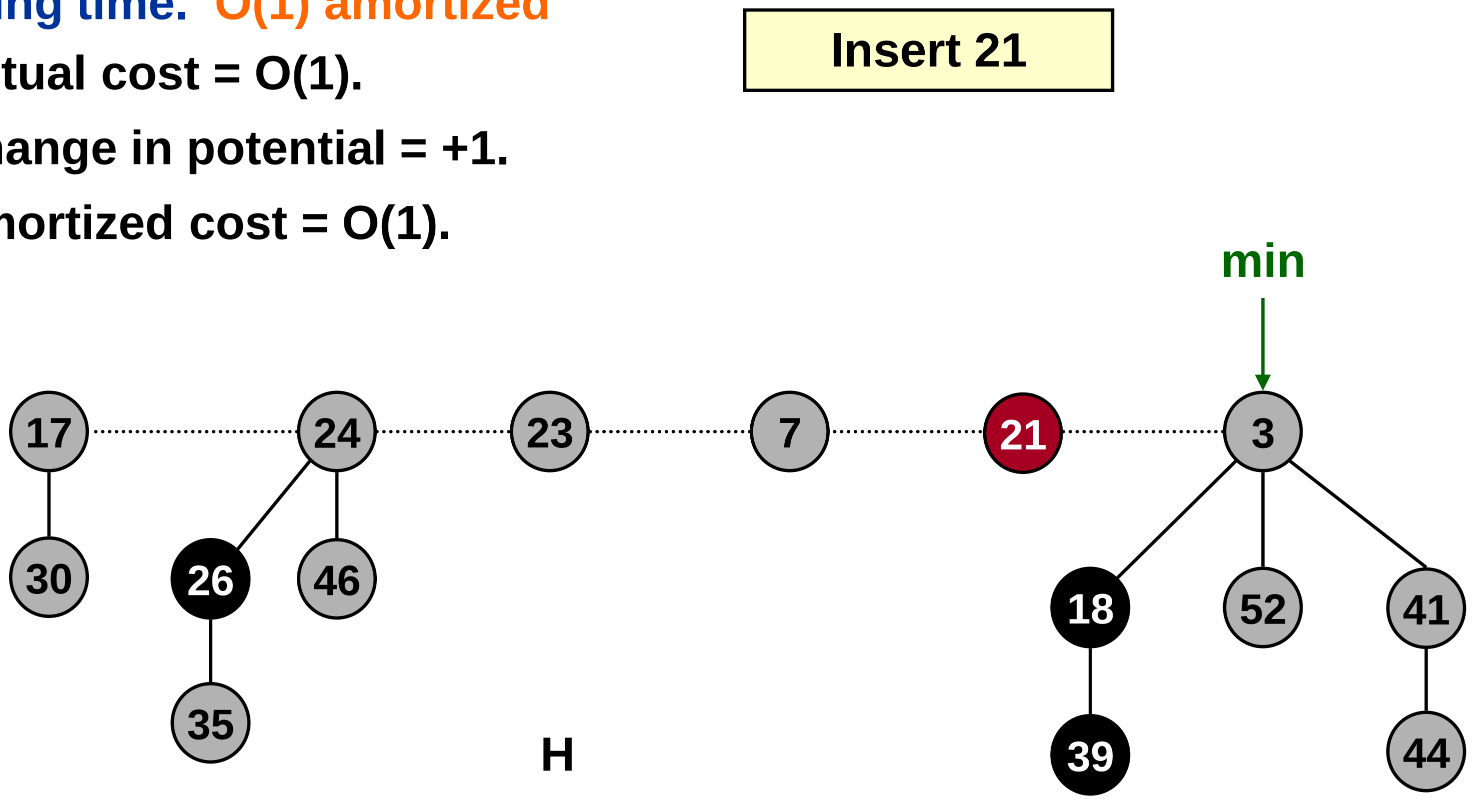
Fibonacci Heaps: Insert

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).



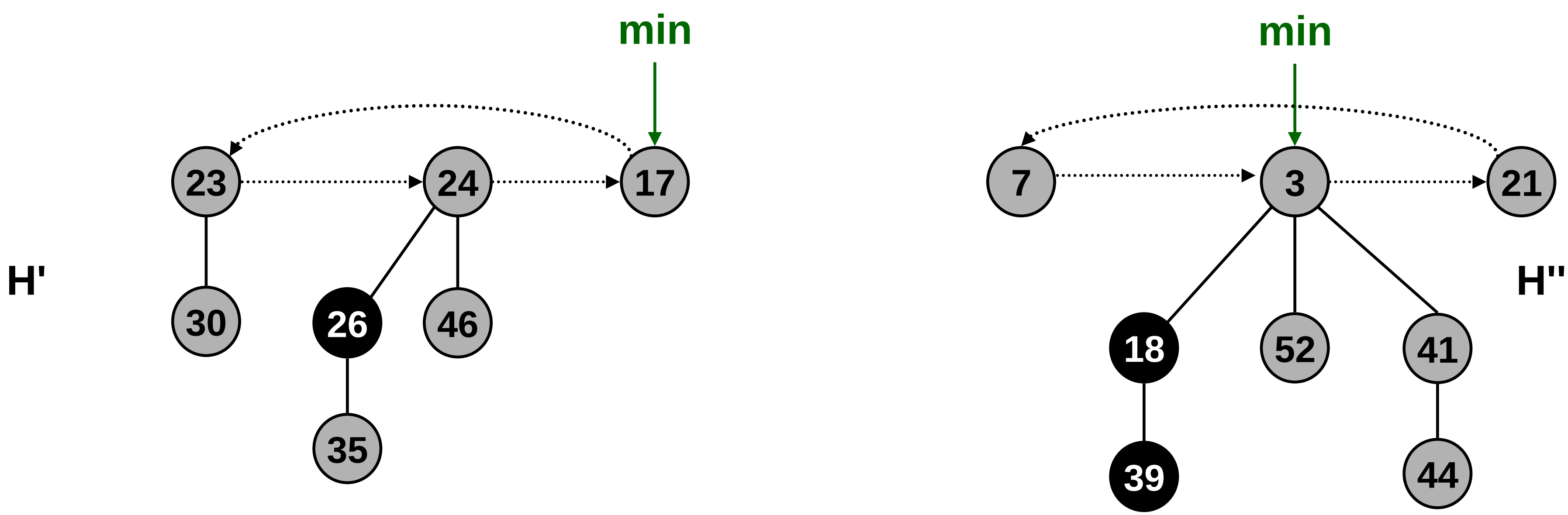
node has already been allocated and that $x.key$ has already

```
FIB-HEAP-INSERT(H, x)
x.degree = 0
x.p = NIL
x.child = NIL
x.mark = FALSE
if H.min == NIL
    create a root list for H containing just x
    H.min = x
else insert x into H's root list
    if x.key < H.min.key
        H.min = x
H.n = H.n + 1
```

Fibonacci Heaps: Union

Union.

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



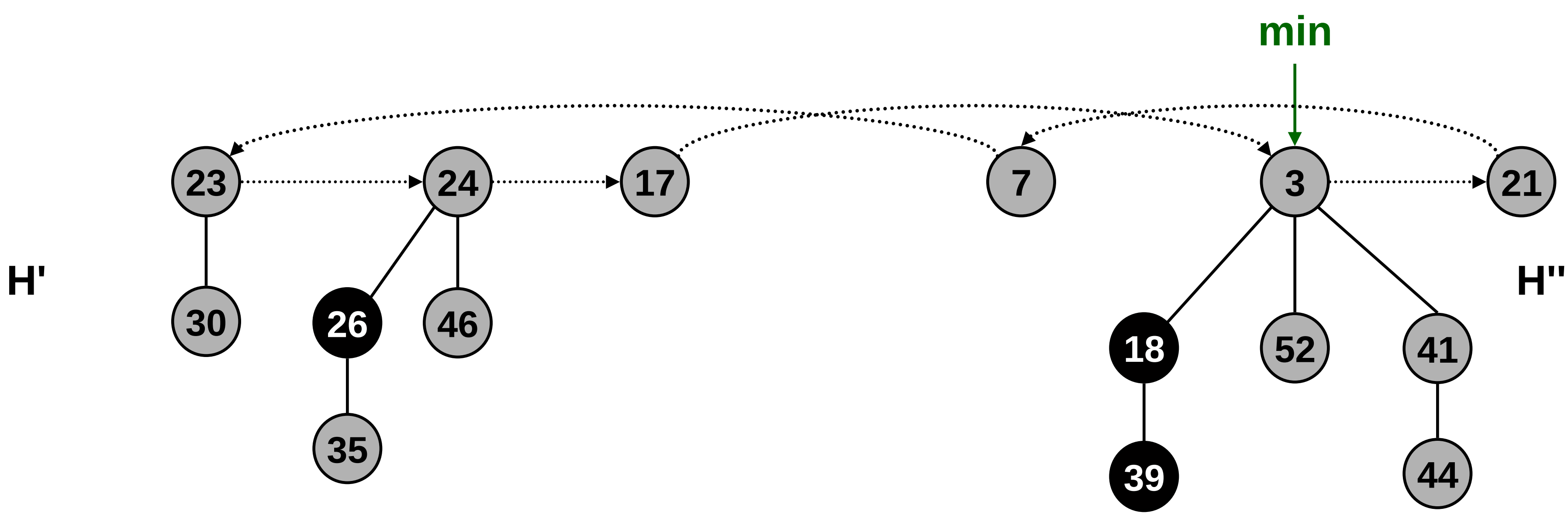
Fibonacci Heaps: Union

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).

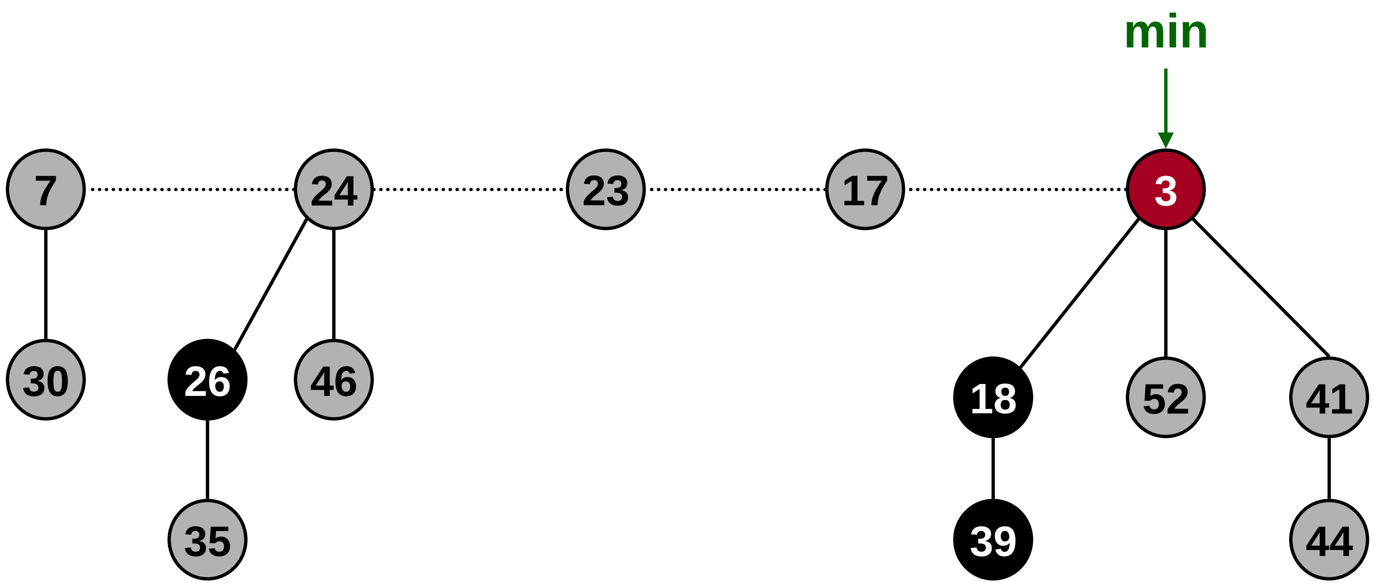


```
FIB-HEAP-UNION(H₁, H₂)
  H = MAKE-FIB-HEAP()
  H.min = H₁.min
  concatenate the root list of H₂ with the root list of H
  if (H₁.min == NIL) or (H₂.min ≠ NIL and H₂.min.key < H₁.min.key)
      H.min = H₂.min
  H.n = H₁.n + H₂.n
  return H
```

Fibonacci Heaps: Delete Min

Delete min.

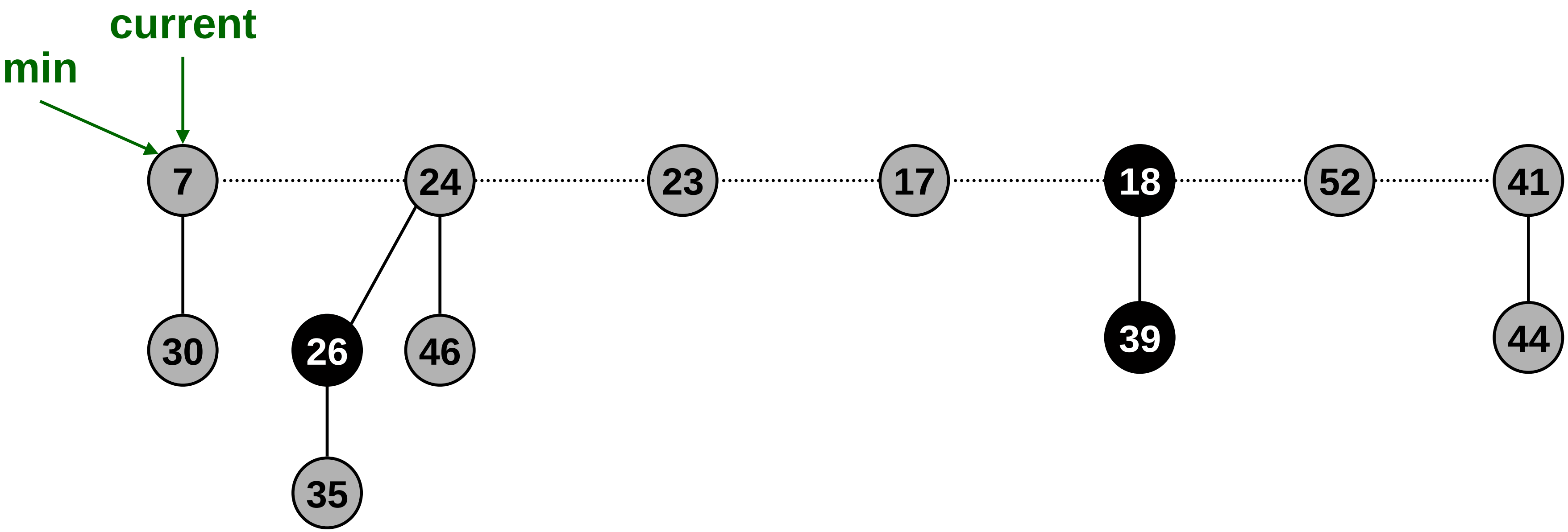
- **Delete min and concatenate its children into root list.**
- **Consolidate trees so that no two roots have same degree.**



Fibonacci Heaps: Delete Min

Delete min.

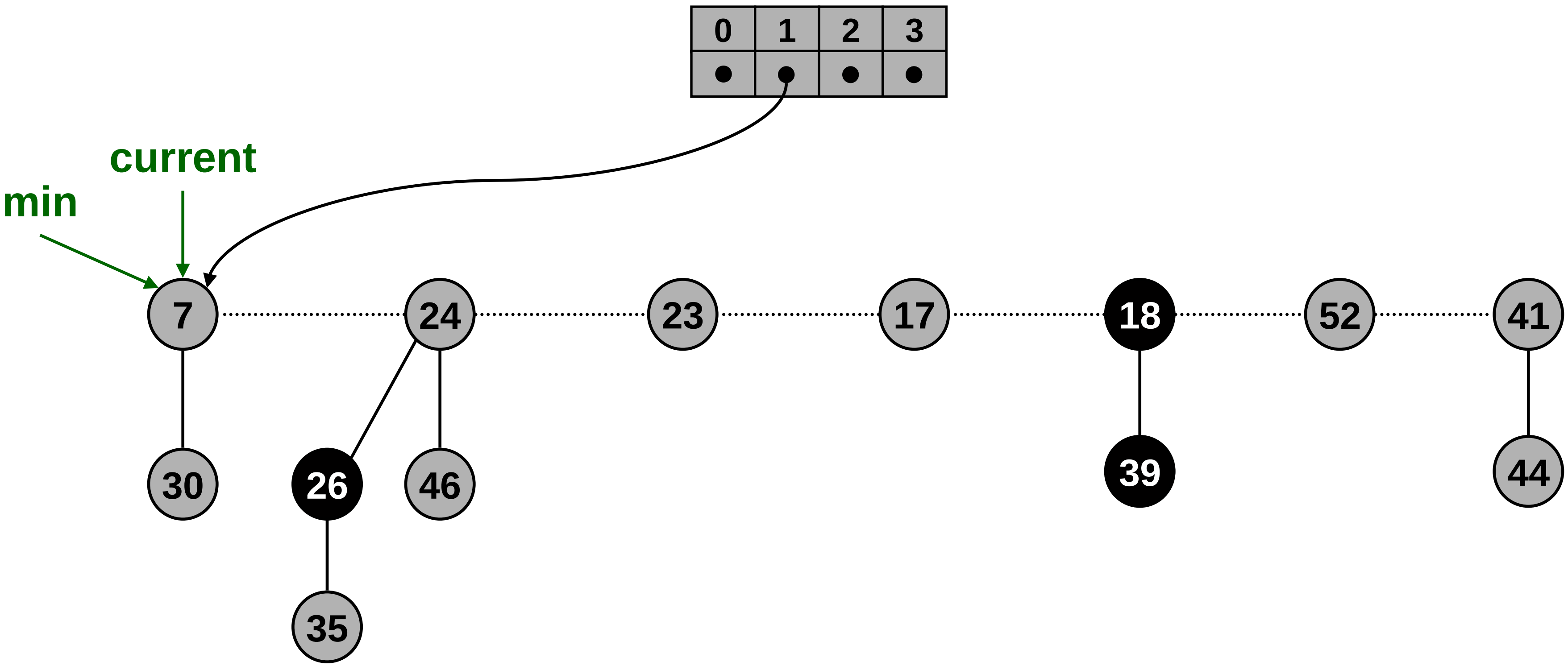
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Delete Min

Delete min.

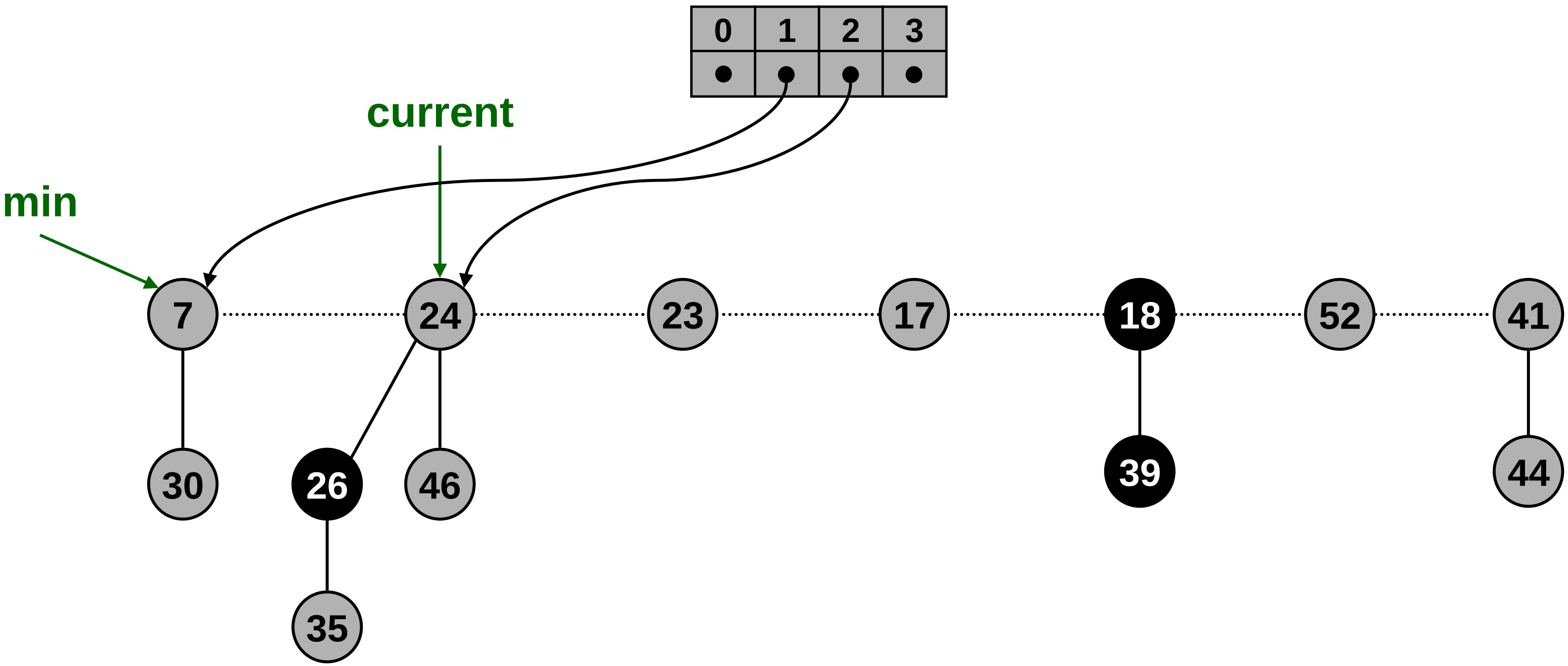
- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**



Fibonacci Heaps: Delete Min

Delete min.

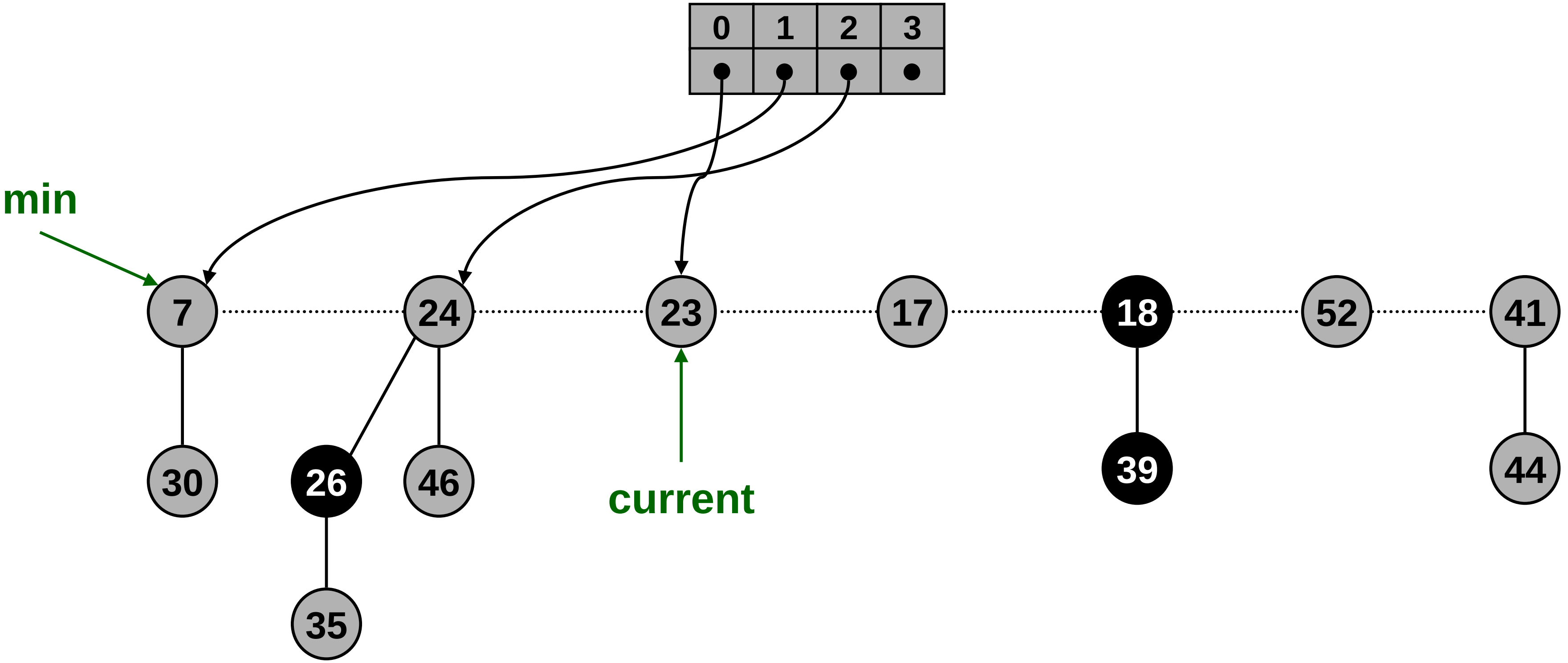
- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**



Fibonacci Heaps: Delete Min

Delete min.

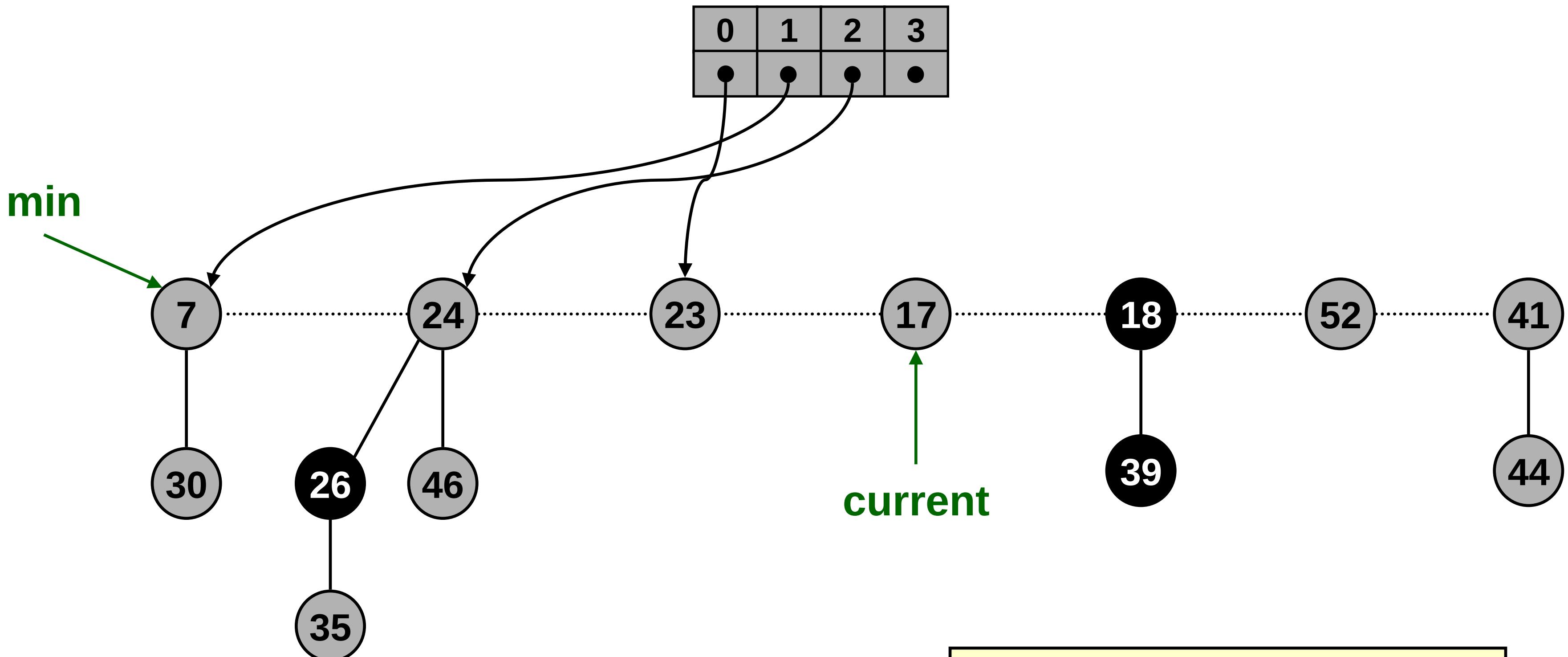
- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**



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.

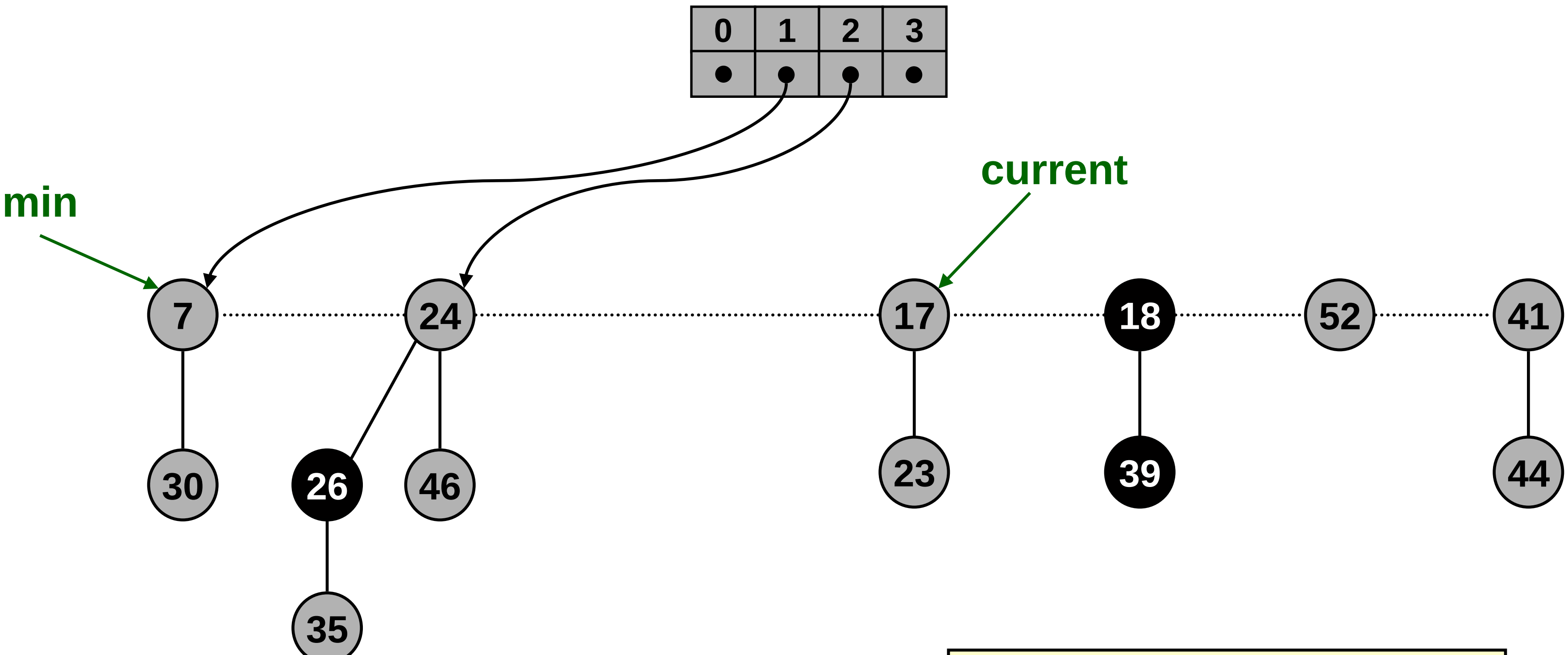


Merge 17 and 23 trees.

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.**

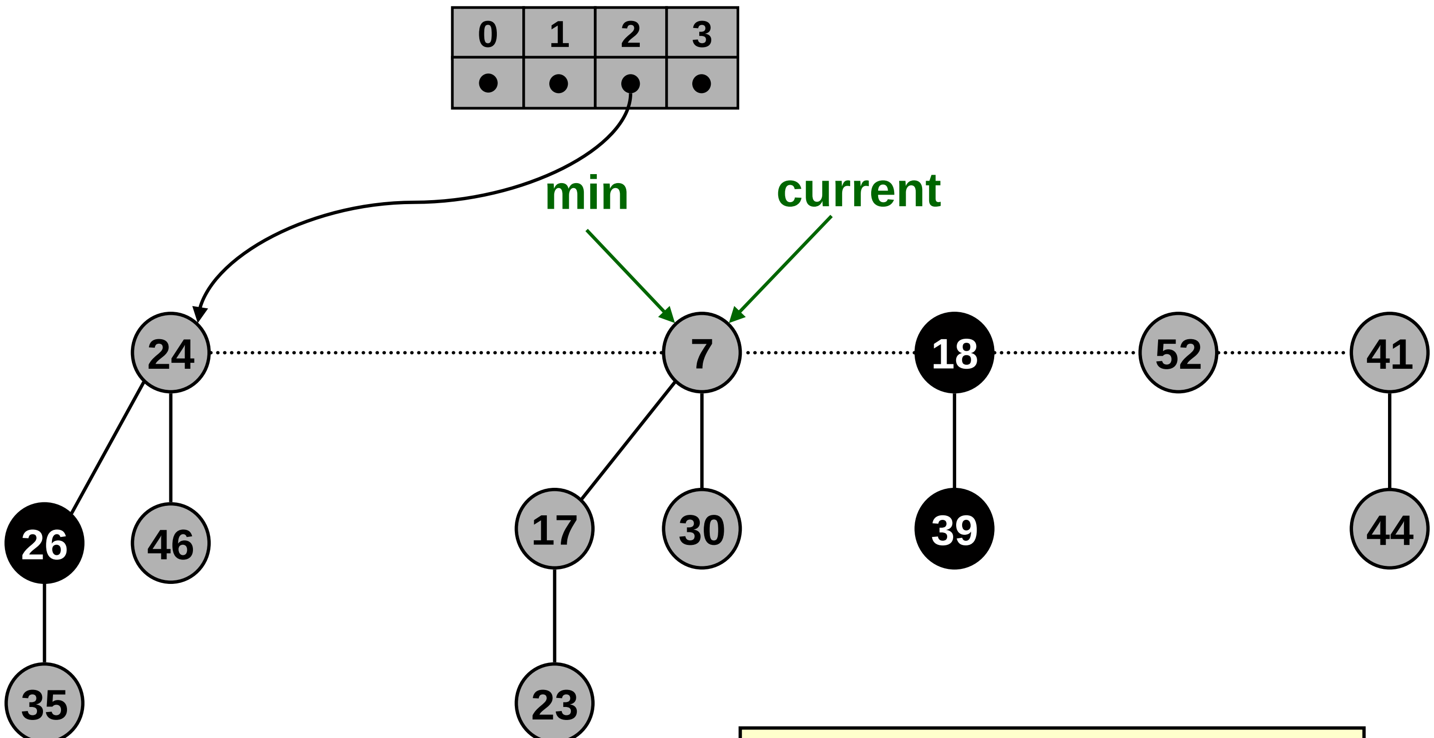


Merge 7 and 17 trees.

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.**

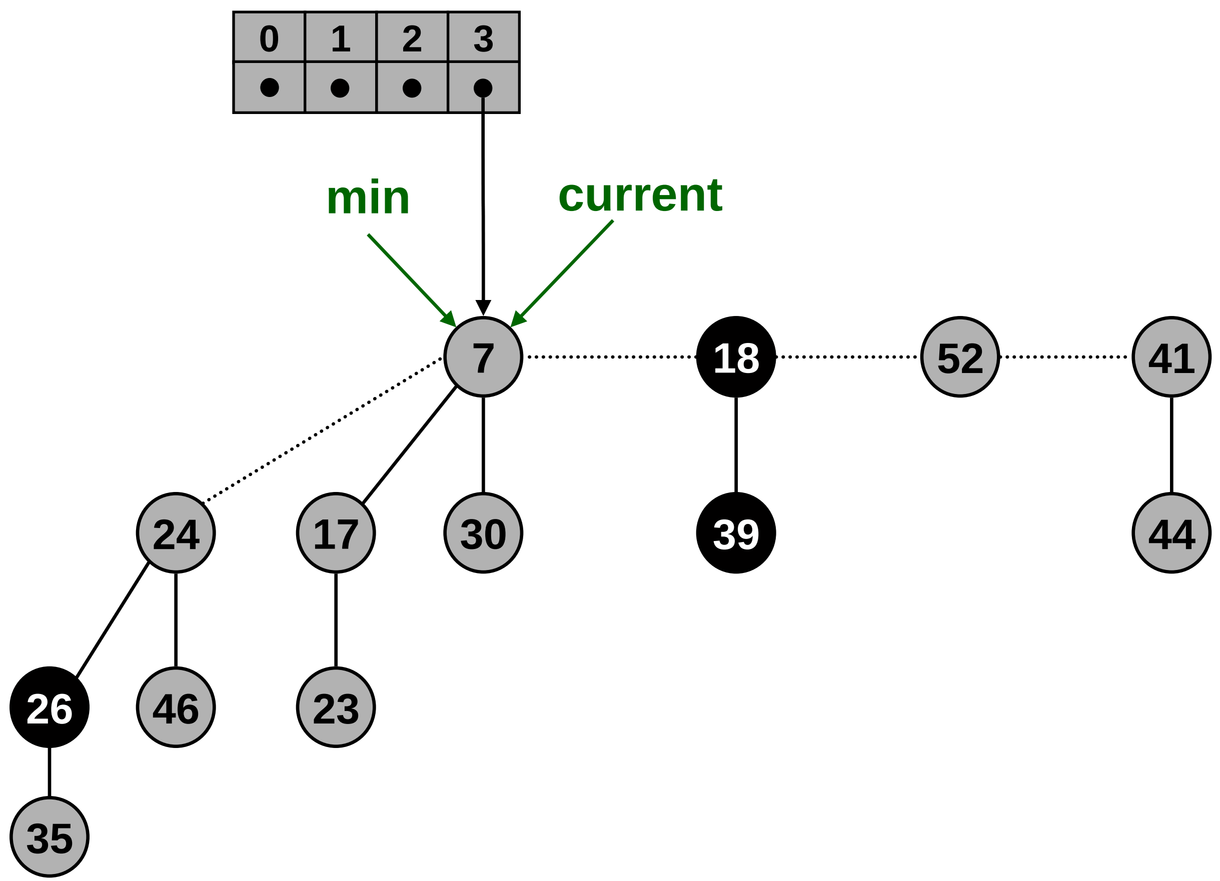


Merge 7 and 24 trees.

Fibonacci Heaps: Delete Min

Delete min.

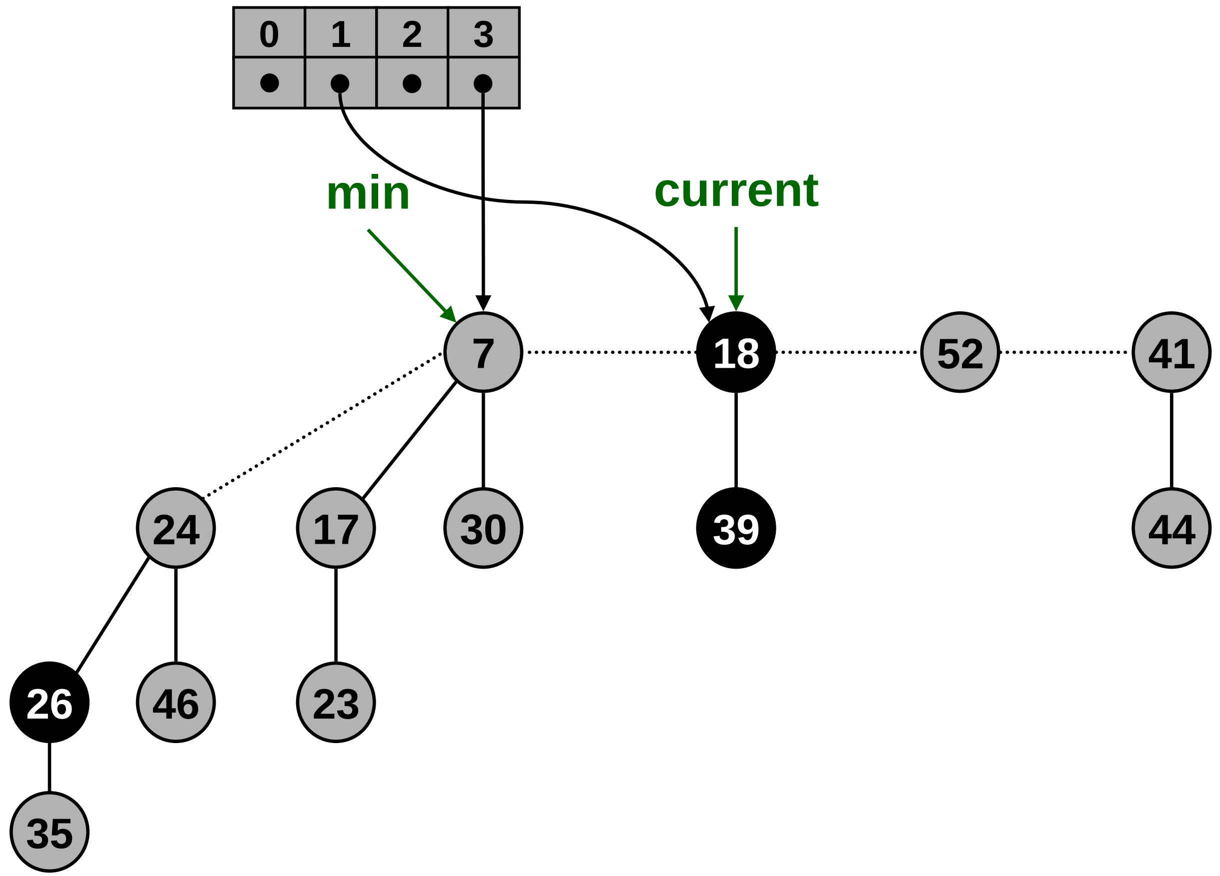
- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**



Fibonacci Heaps: Delete Min

Delete min.

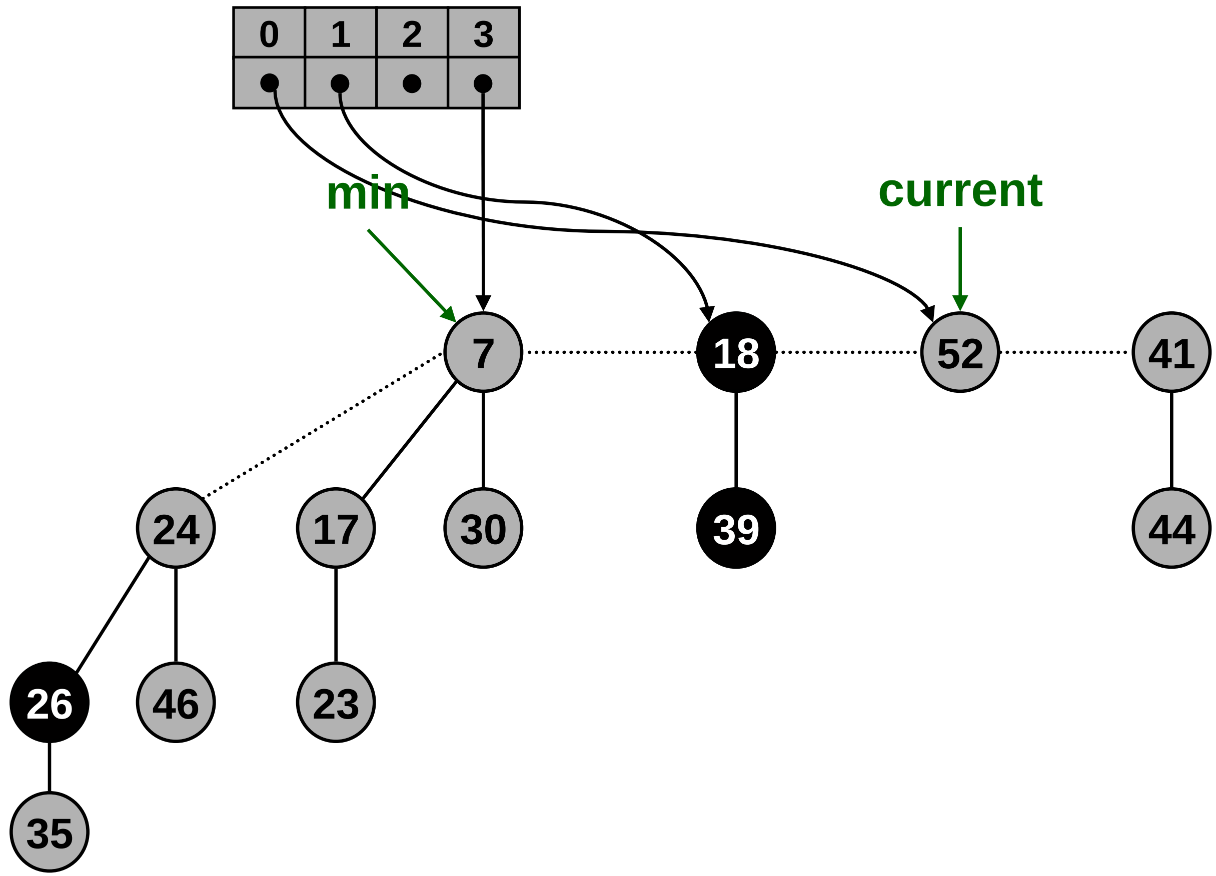
- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**



Fibonacci Heaps: Delete Min

Delete min.

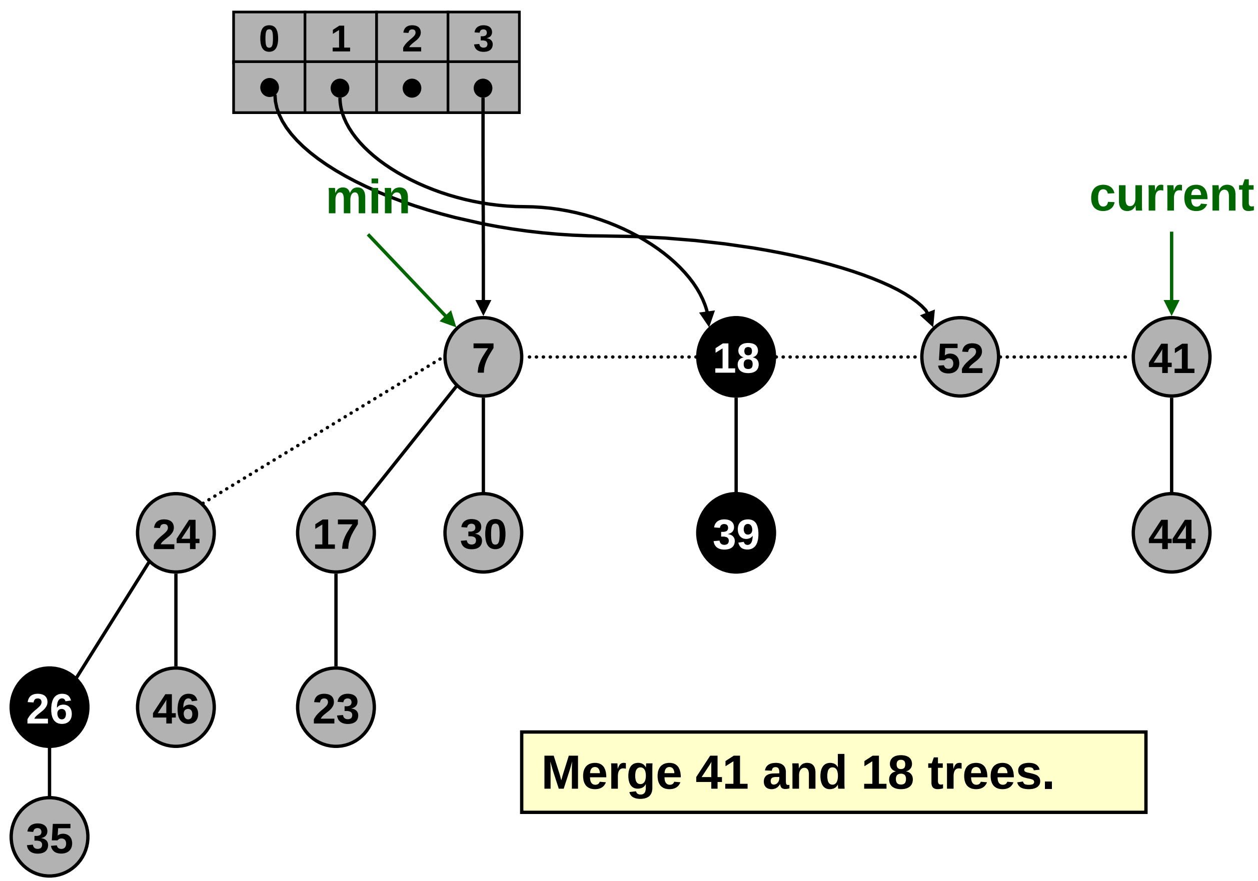
- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**



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.**

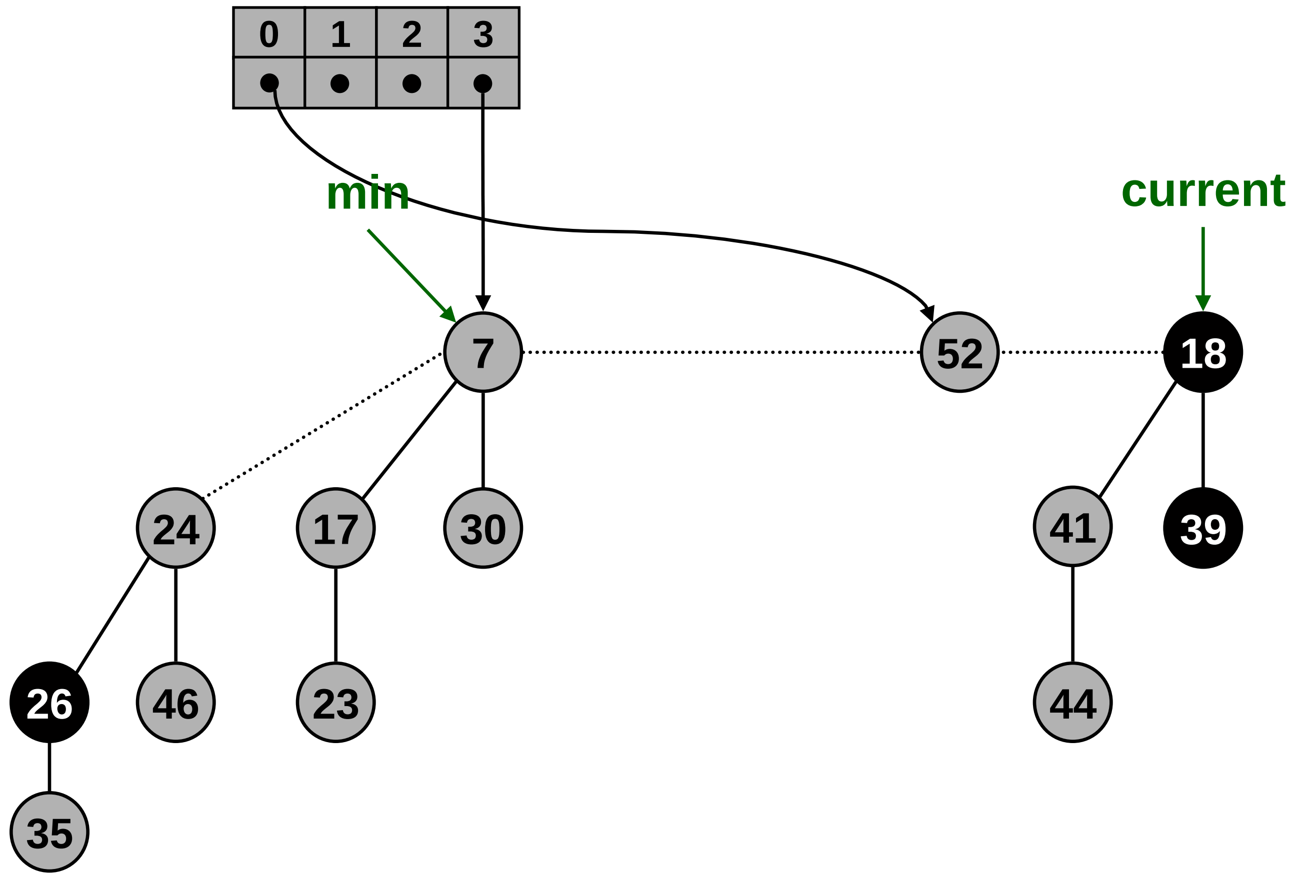


Merge 41 and 18 trees.

Fibonacci Heaps: Delete Min

Delete min.

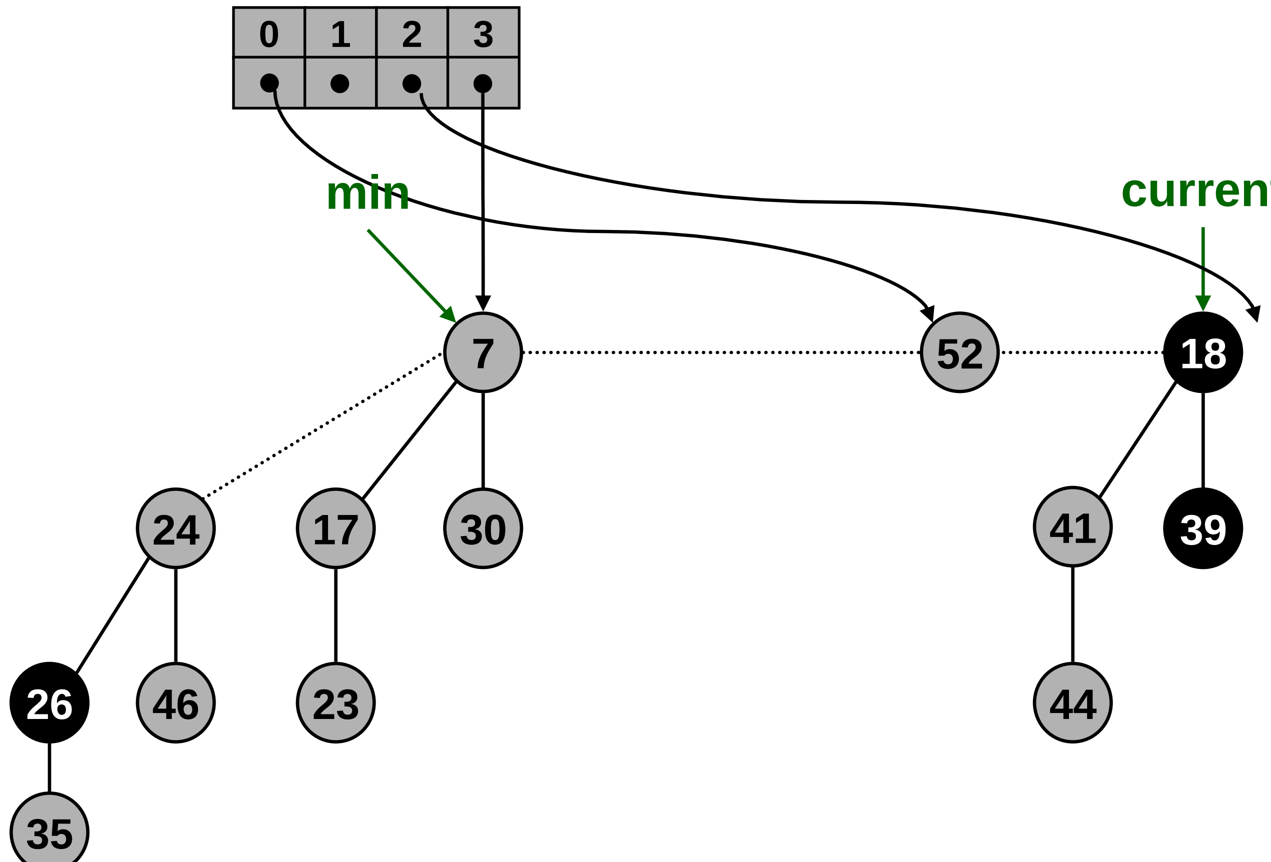
- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**



Fibonacci Heaps: Delete Min

Delete min.

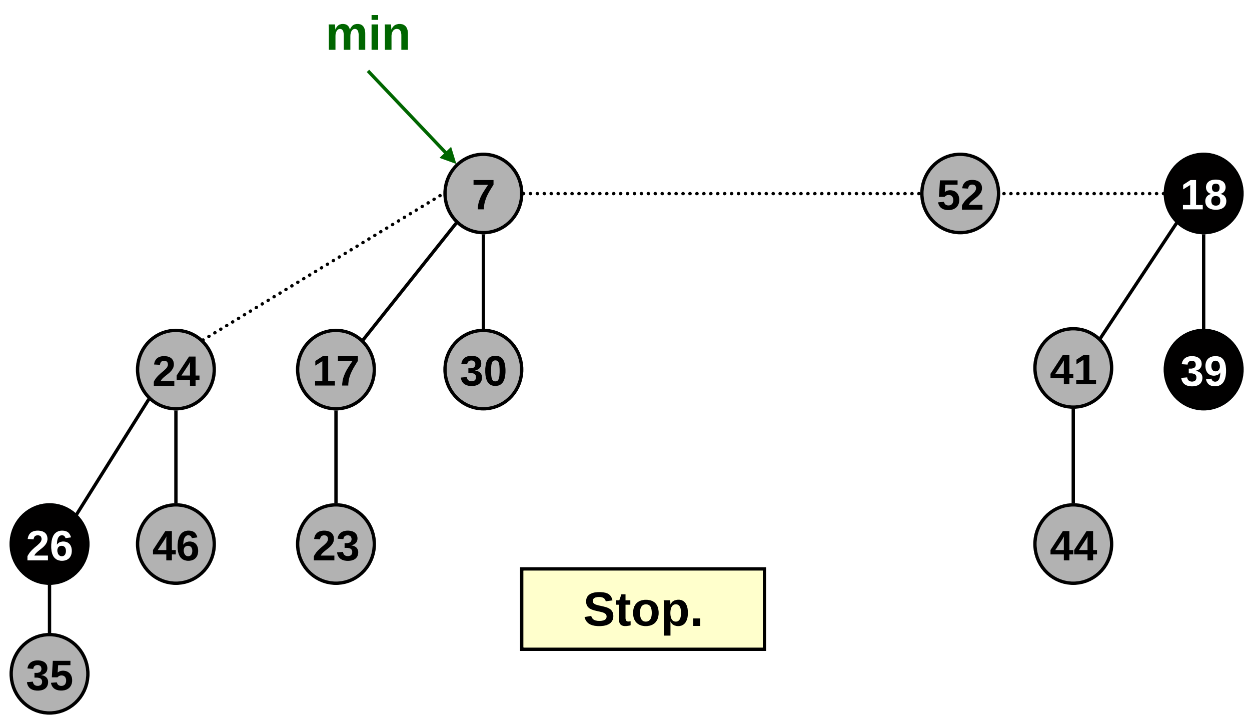
- Delete min and concatenate its children into root list.
- **Consolidate trees so that no two roots have same degree.**



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.



FIB-HEAP-EXTRACT-MIN(H)

```
z = H.min
if z ≠ NIL
      for each child x of z
            add x to the root list of H
            x.p = NIL
      remove z from the root list of H
      if z == z.right
            H.min = NIL
      else H.min = z.right
            CONSOLIDATE(H)
      H.n = H.n − 1
return z
```

EXTRACT-MI

```
CONSOLIDATE(H)
let A[0 . . D(H.n)] be a new array
for i = 0 to D(H.n)
    A[i] = NIL
for each node w in the root list of H
    x = w
    d = x.degree
    while A[d] ≠ NIL
        y = A[d]            // another node with the same degree as x
        if x.key > y.key
            exchange x with y
        FIB-HEAP-LINK(H, y, x)
        A[d] = NIL
        d = d + 1
    A[d] = x
H.min = NIL
for i = 0 to D(H.n)
    if A[i] ≠ NIL
        if H.min == NIL
            create a root list for H containing just A[i]
            H.min = A[i]
        else insert A[i] into H's root list
            if A[i].key < H.min.key
                H.min = A[i]
```

FIB-HEAP-LINK(H, y, x)

FIB-HEAP –LINK(H,y,x)

1. remove y from the root list of H

2. Make y a child of x, incrementing x.degree.

3. y.mark = FALSE

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

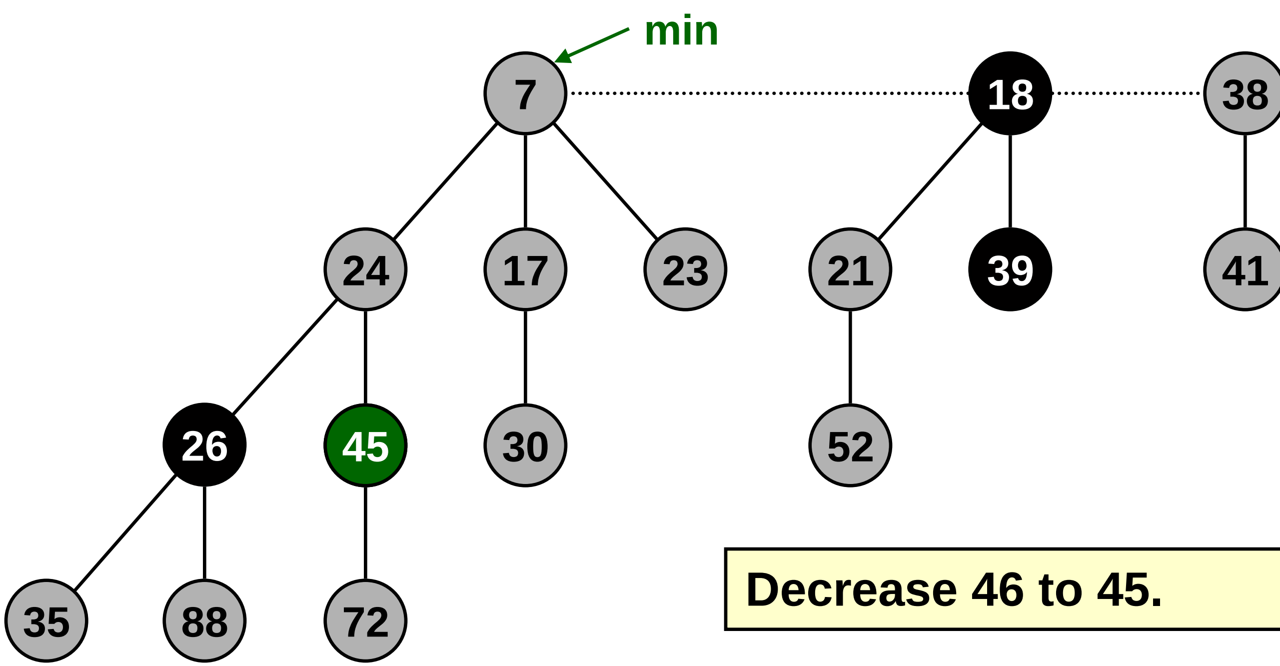
Is amortized cost of $O(D(n))$ good?

- **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

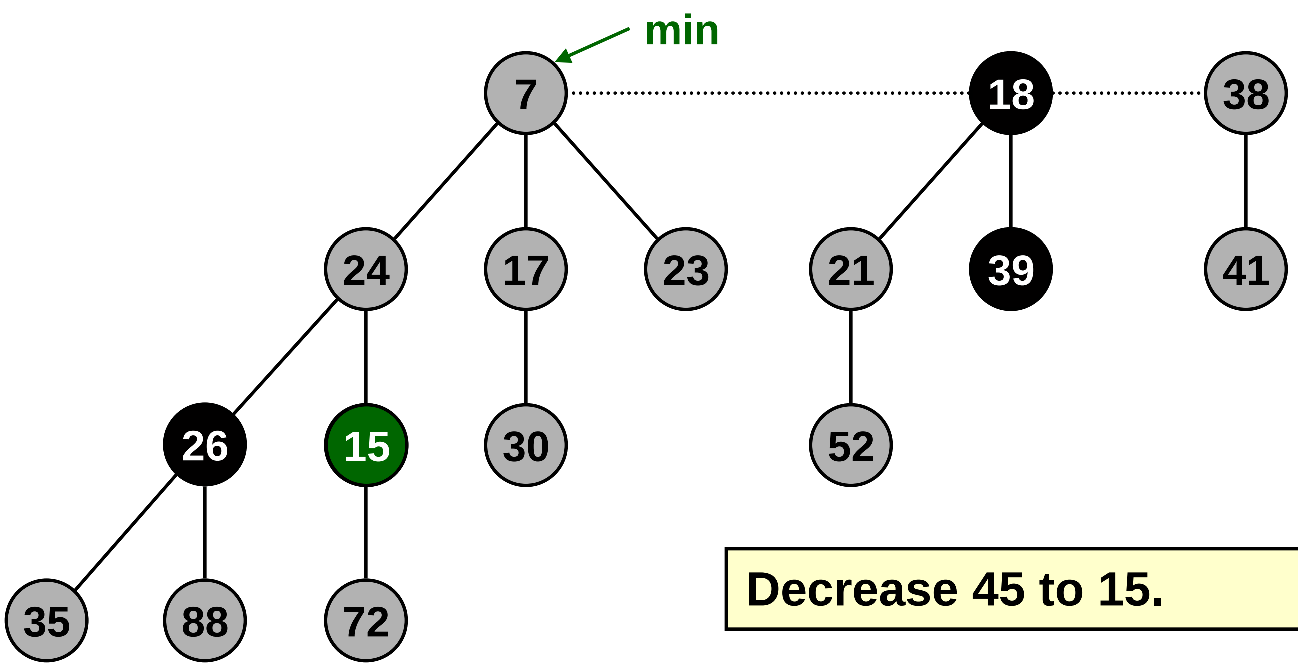


Decrease 46 to 45.

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

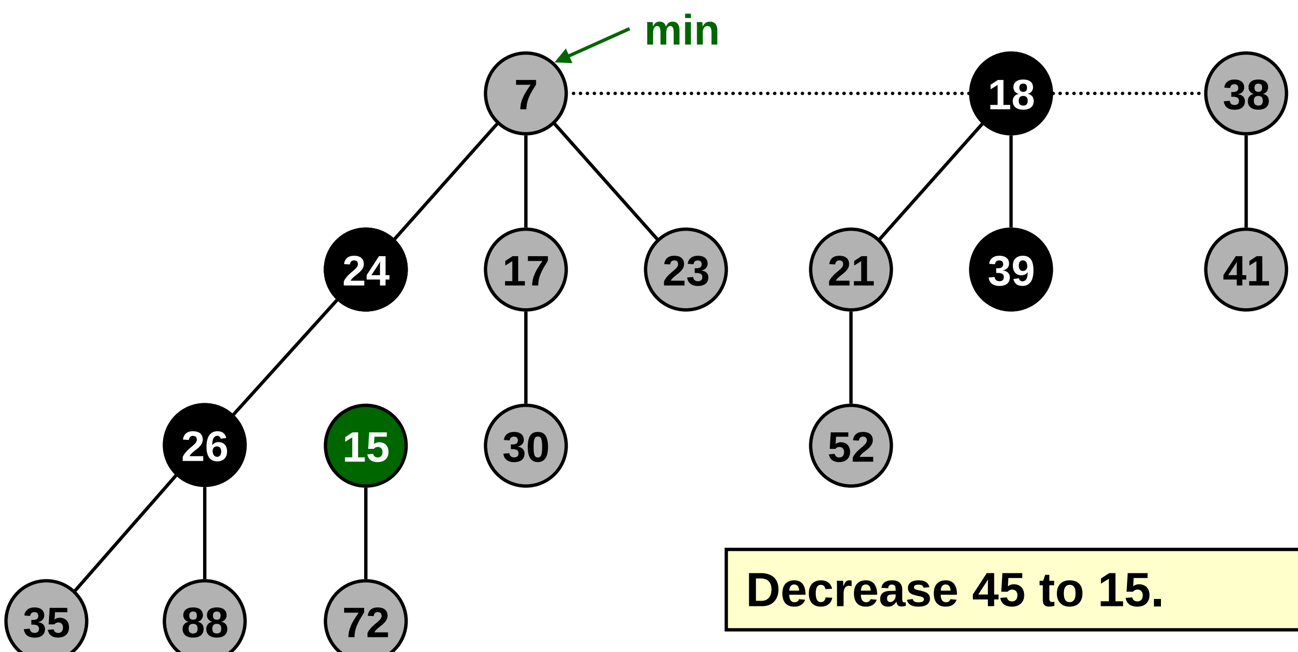


Decrease 45 to 15.

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

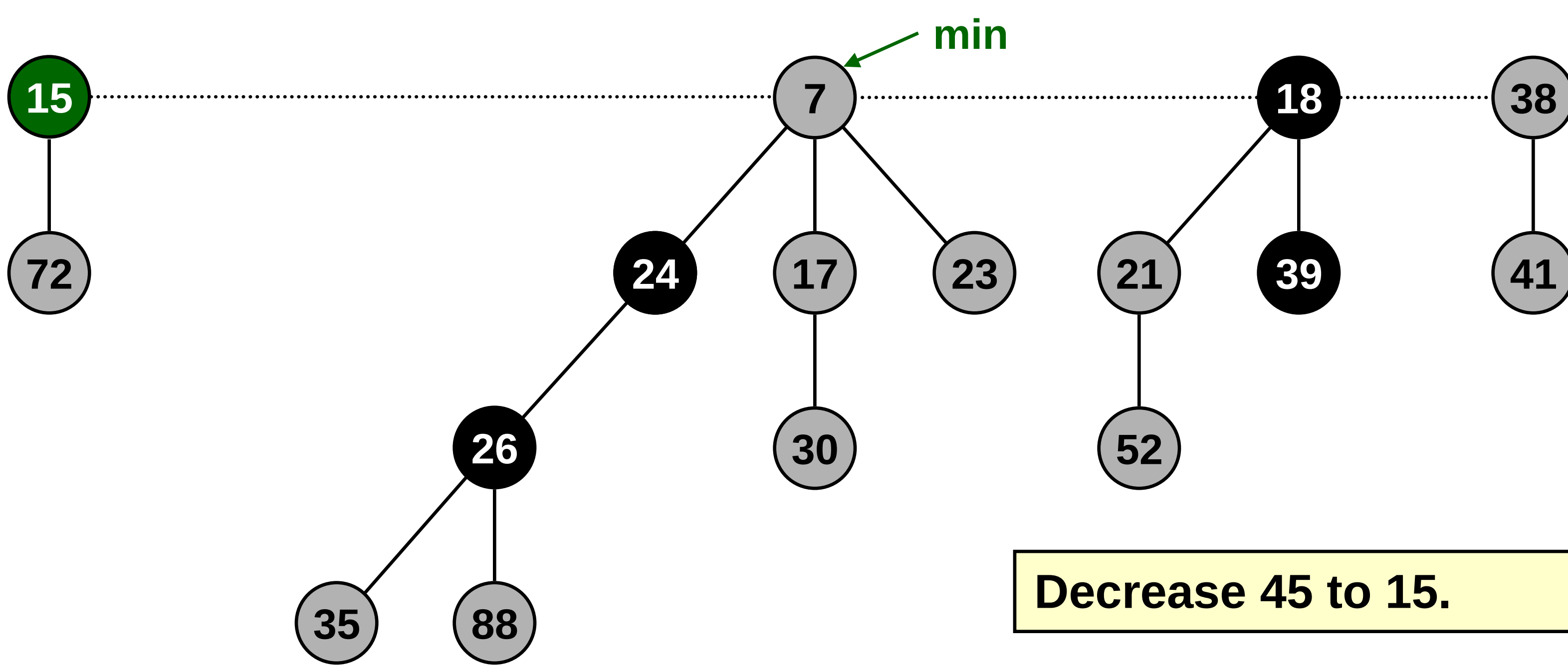


Decrease 45 to 15.

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

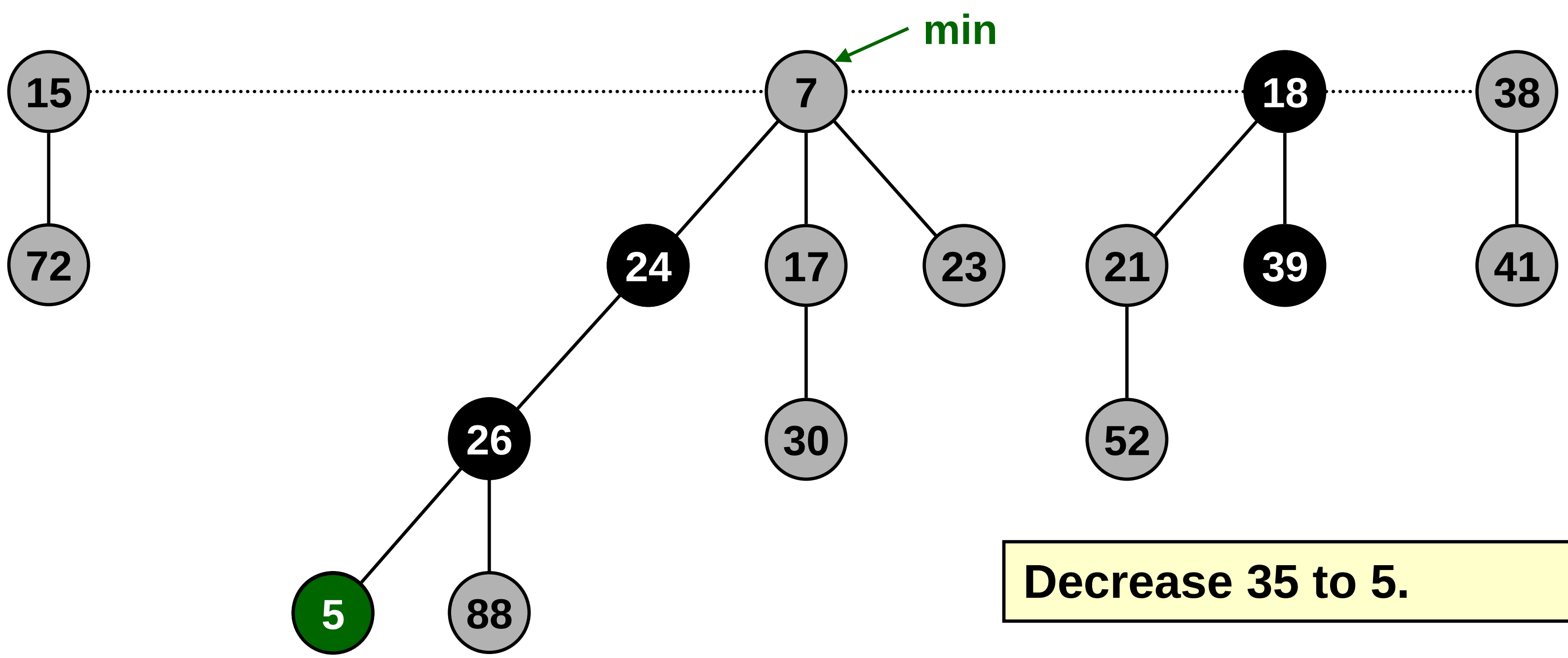


Decrease 45 to 15.

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.

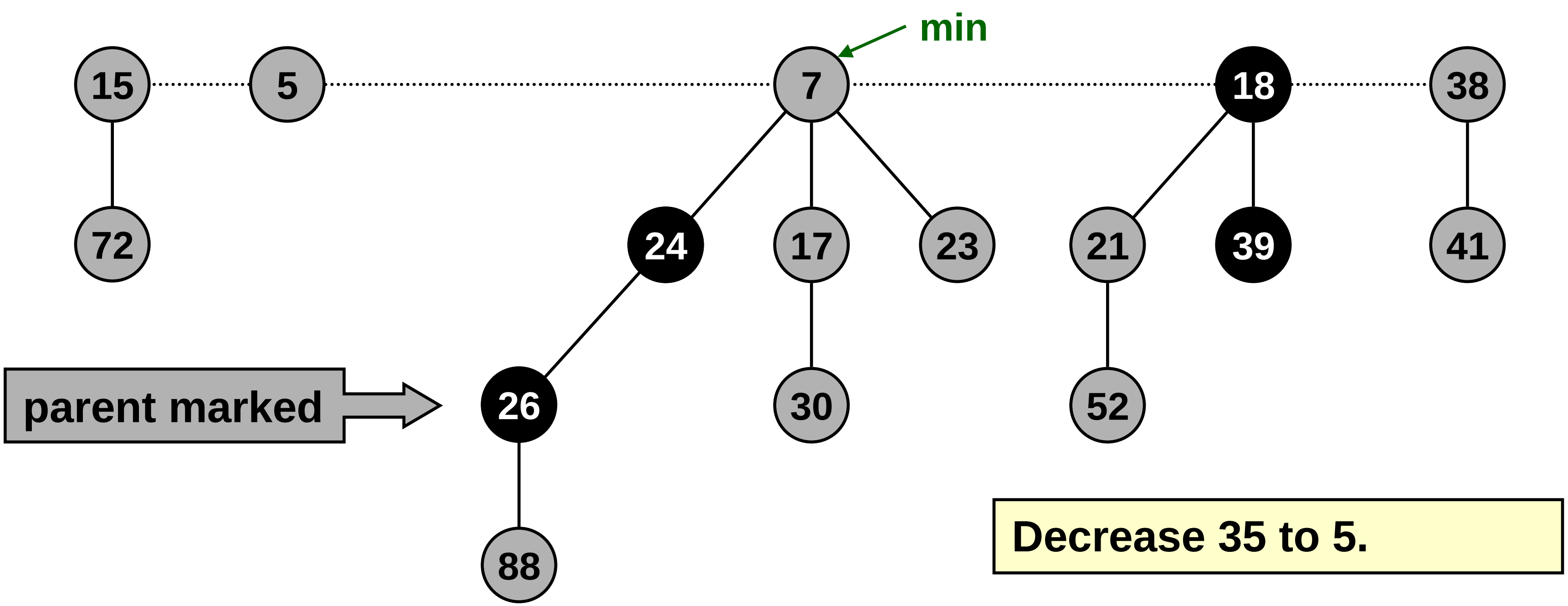


Decrease 35 to 5.

Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

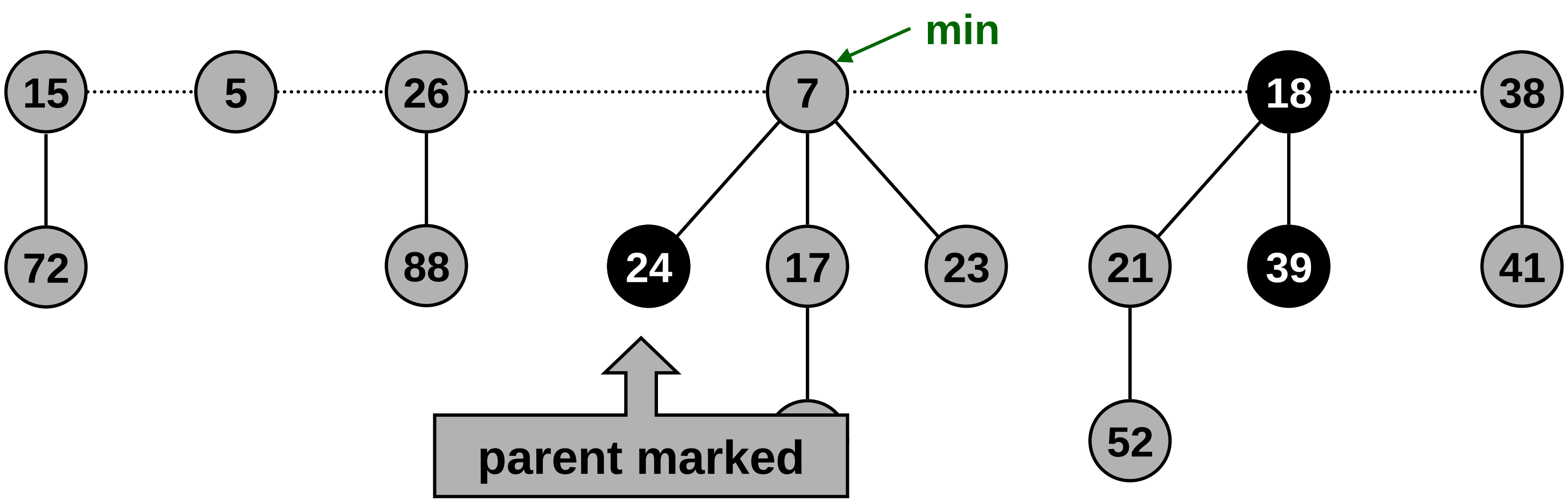
- Case 2: parent of x is marked.
  - decrease key of x to k
  - cut off link between x and its parent p[x], and add x to root list
  - cut off link between p[x] and p[p[x]], add p[x] to root list
    - ✐ If p[p[x]] unmarked, then mark it.
    - ✐ If p[p[x]] marked, cut off p[p[x]], unmark, and repeat.



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.

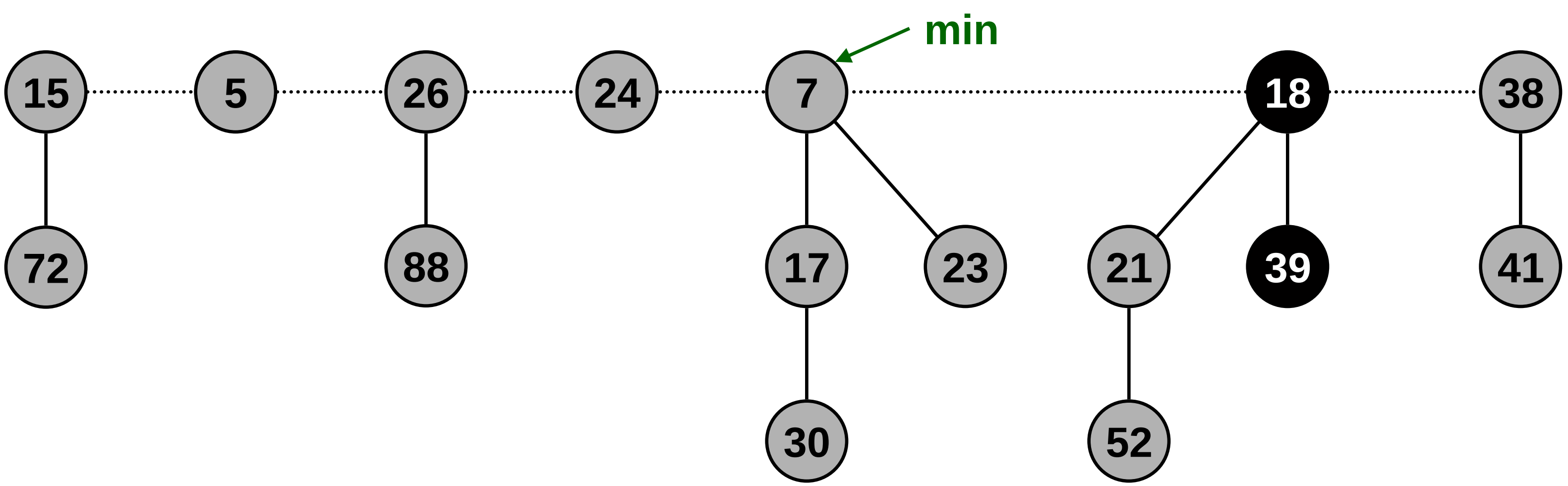


Decrease 35 to 5.

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.

```
FIB-HEAP-DECREASE-KEY(H, x, k)
if k > x.key
    error "new key is greater than current key"
x.key = k
y = x.p
if y ≠ NIL and x.key < y.key
    CUT(H, x, y)
    CASCADING-CUT(H, y)
if x.key < H.min.key
    H.min = x
```

```
CUT(H, x, y)
remove x from the child list of y, decrementing y.degree
add x to the root list of H
x.p = NIL
x.mark = FALSE
```

```
CASCADING-CUT(H, y)
z = y.p
if z ≠ NIL
    if y.mark == FALSE
        y.mark = TRUE
    else CUT(H, y, z)
        CASCADING-CUT(H, z)
```

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
- $\Delta\Phi \leq c + 2(-c + 2) = 4 - c$.

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. $D(N) \leq \log_\phi N$, 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:

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

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 $\text{size}(x) \geq \phi^{\text{degree}(x)}$.
  - size(x) = # node in subtree rooted at x
  - taking base ϕ logs, $\text{degree}(x) \leq \log_\phi (\text{size}(x)) \leq \log_\phi N$.
- 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. $$\text{For } k \geq 2,\ F_k = 2 + \sum_{i=0}^{k-2} F_i$$

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

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

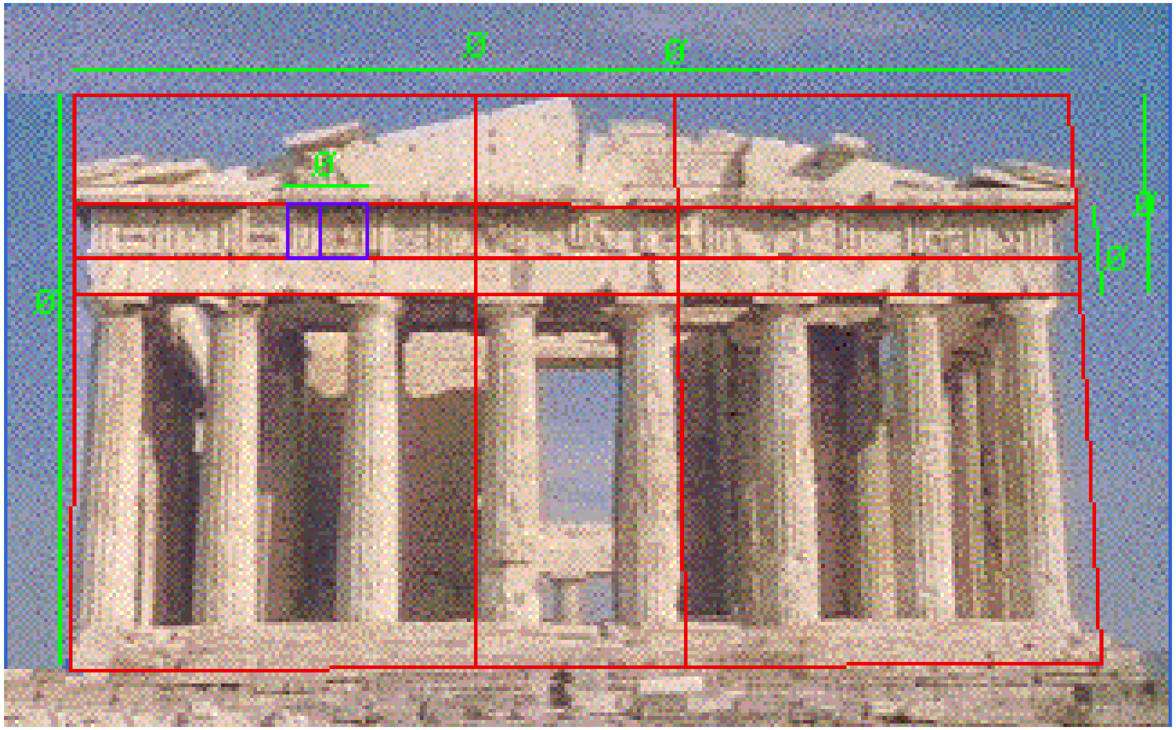
- **This implies that** $\text{size}(x) \geq \phi^{\text{degree}(x)}$ **for all nodes x.**

Golden Ratio

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

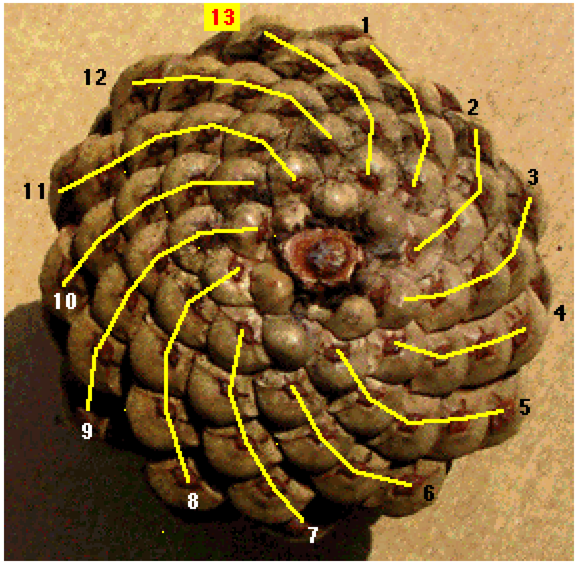
Definition. **The golden ratio $\phi = (1 + \sqrt{5}) / 2 = 1.618\ldots$**

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

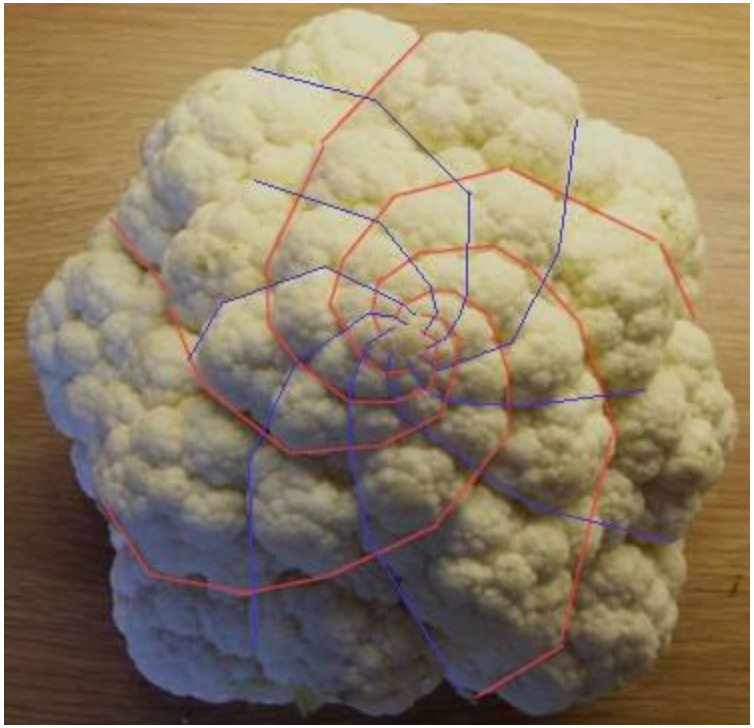


Parthenon, Athens Greece

Fibonacci Numbers and Nature



Pinecone



Cauliflower

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:

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

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

On Complicated Algorithms

"Once you succeed in writing the programs for [these] complicated algorithms, they usually run extremely fast. The computer doesn't need to understand the algorithm, its task is only to run the programs."

R. E. Tarjan