Fibonacci Heap

Slides from COS 423 (Princeton)

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
extract-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.

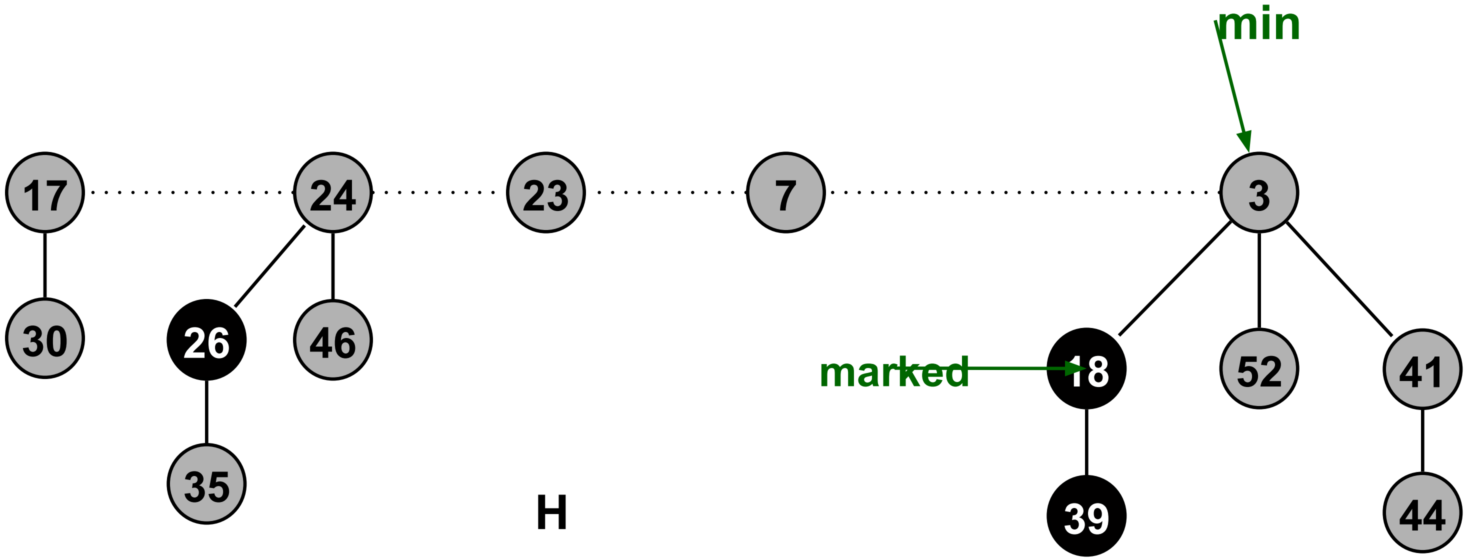
Fibonacci heap intuition.

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

Fibonacci Heaps: Structure

Fibonacci heap.

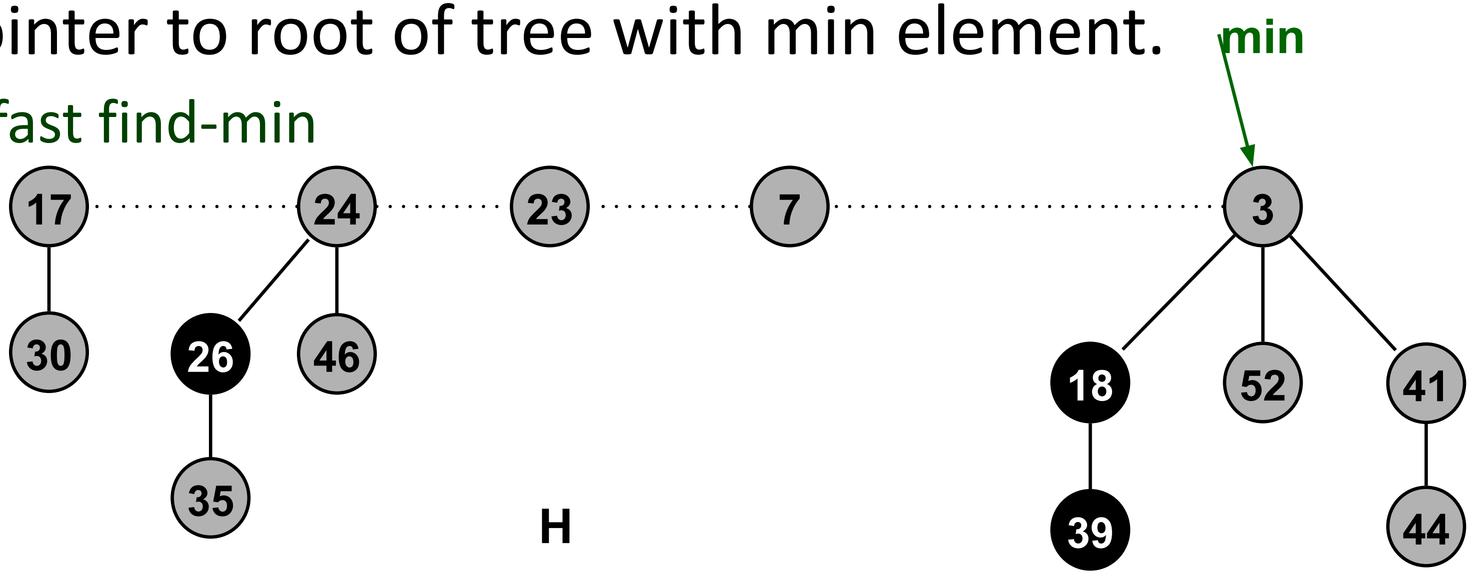
- Set of min-heap ordered trees.



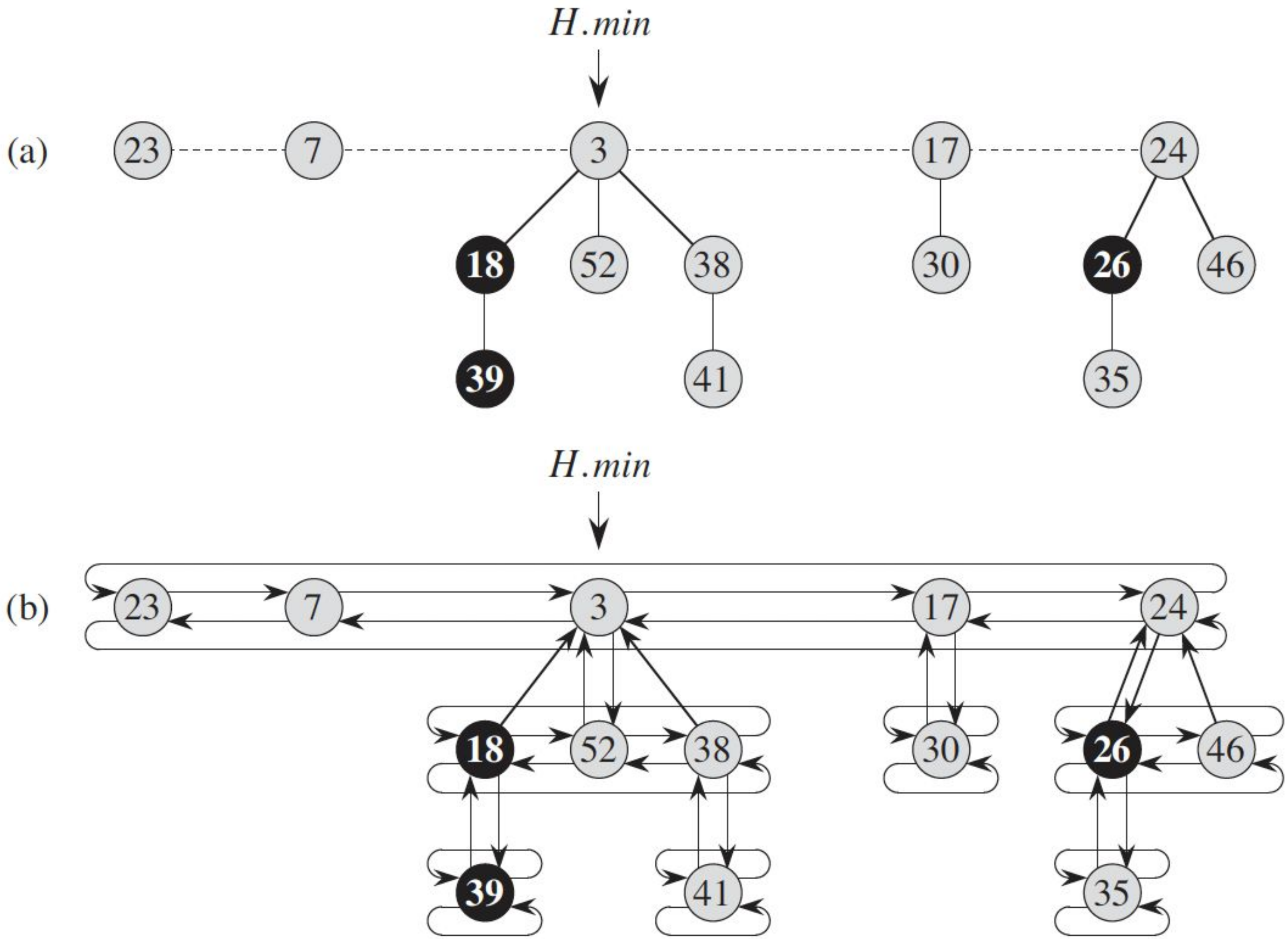
Fibonacci Heaps: Implementation

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: Implementation



Amortized Analysis (Potential Method)

- Potential method of amortized analysis represents the prepaid work as “potential energy,” or just “potential,”
  - which can be released to pay for future operations
- We associate the potential with the data structure as a whole

potential function Φ

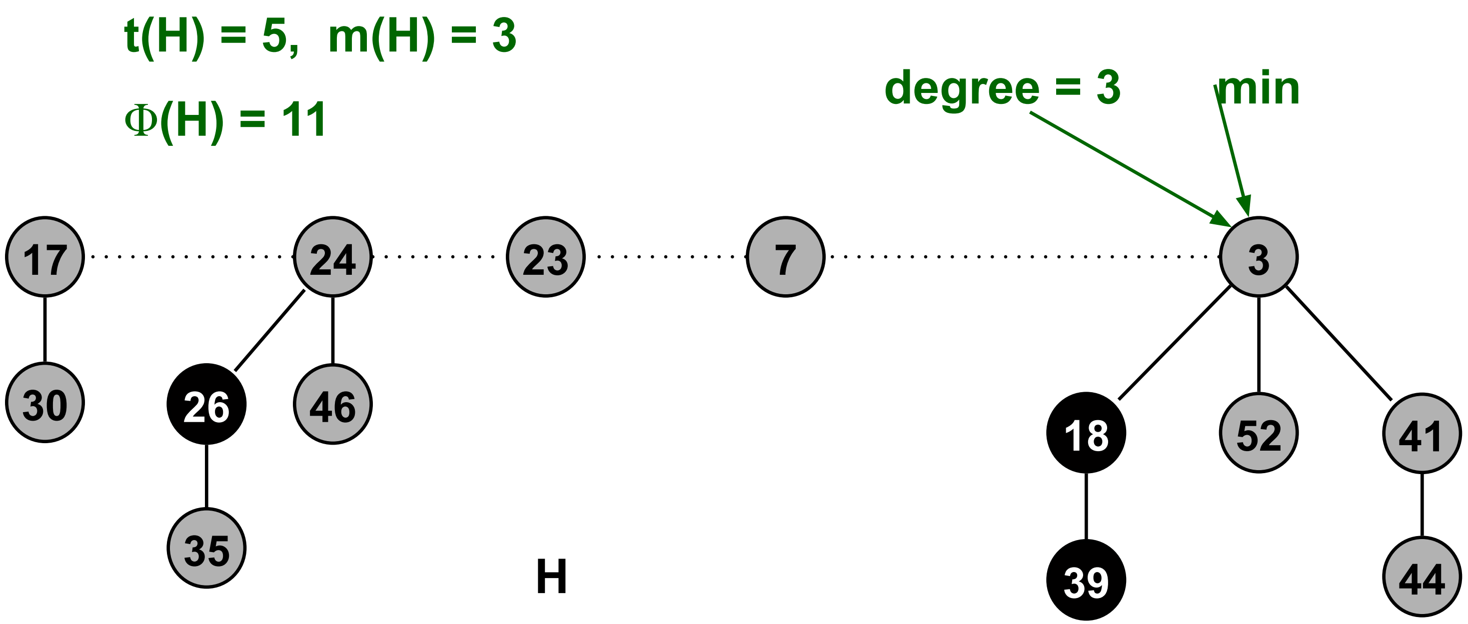
amortized cost $\widehat{c}_i$

$$\widehat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

Fibonacci Heaps: Potential Function

Key quantities.

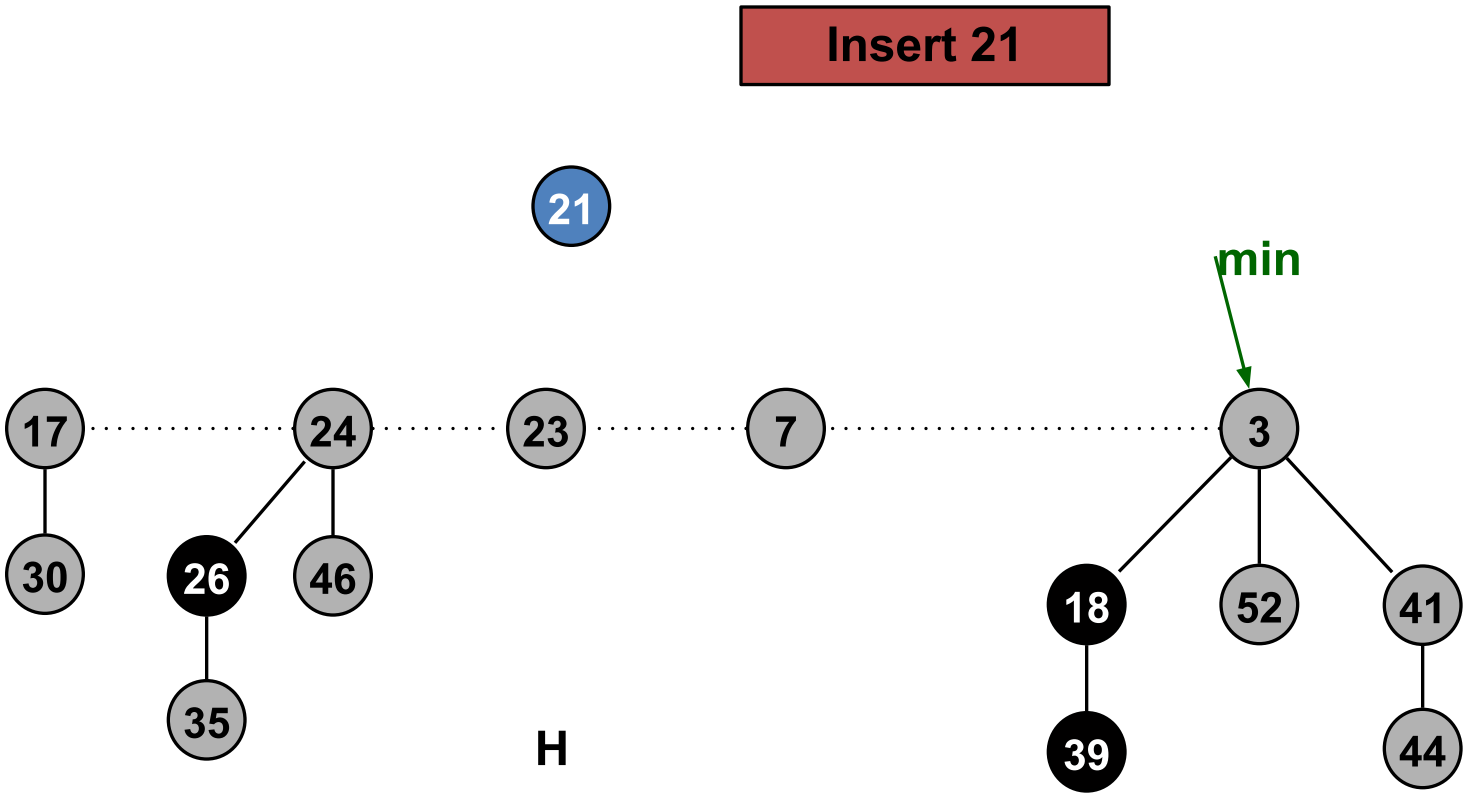
- Degree[x] = degree of node x.
- Mark[x] = mark of node x (black or gray).
- t(H)= # trees.
- m(H) = # marked nodes.
- Φ(H) = t(H) + 2m(H) = potential function.



Fibonacci Heaps: Insert

Insert.

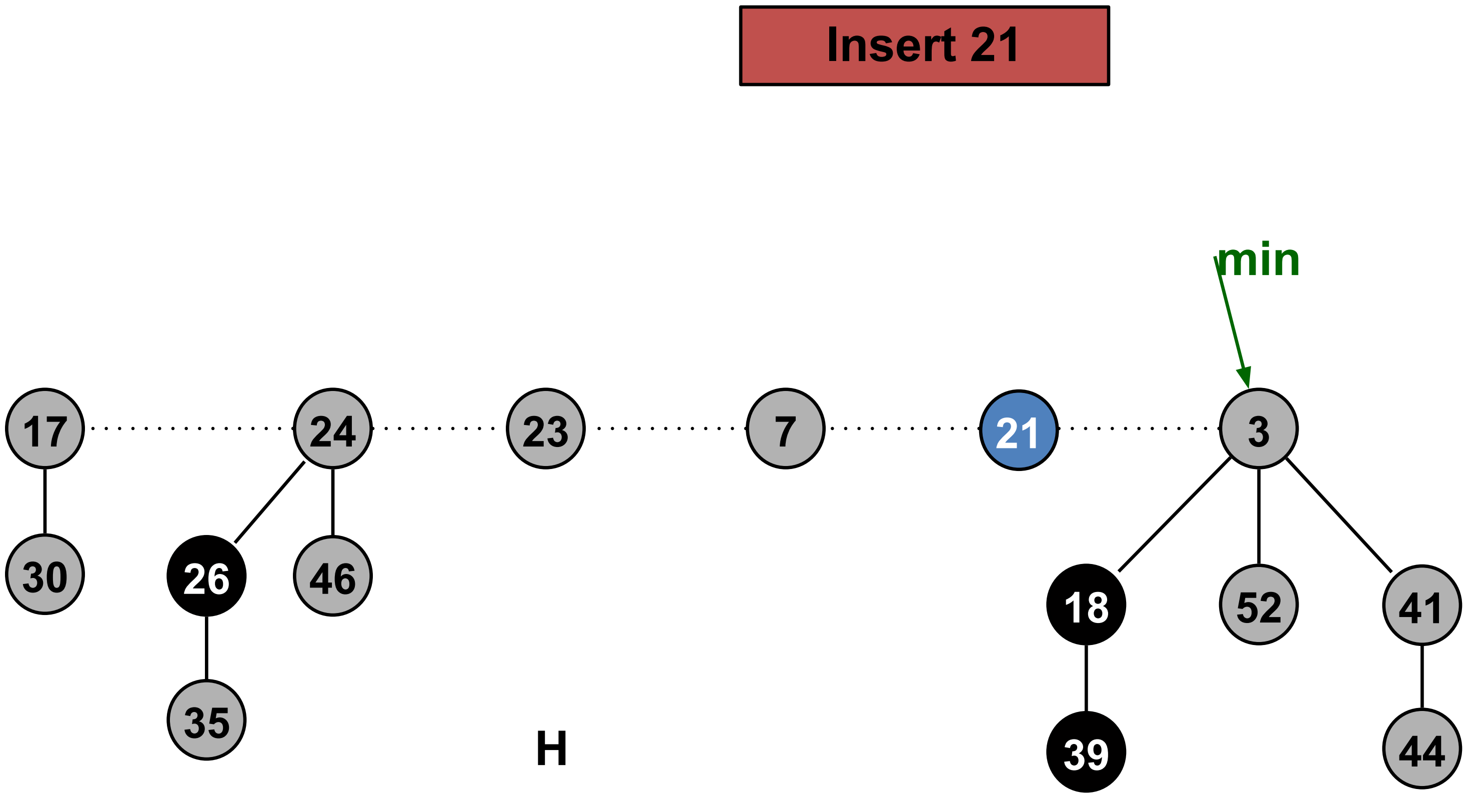
- Create a new singleton tree.
- Add to left of min pointer.
- Update min pointer.



Fibonacci Heaps: Insert

Insert.

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



Fibonacci Heaps: Insert

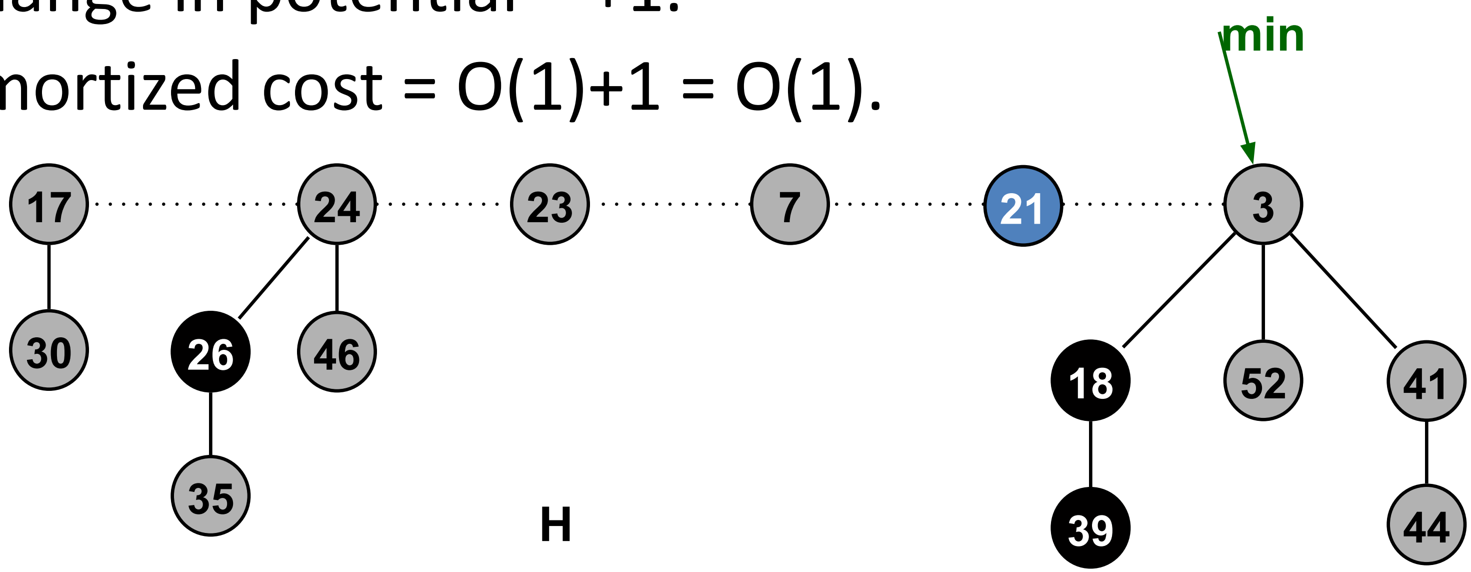
Insert.

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

Running time. O(1) amortized

Insert 21

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



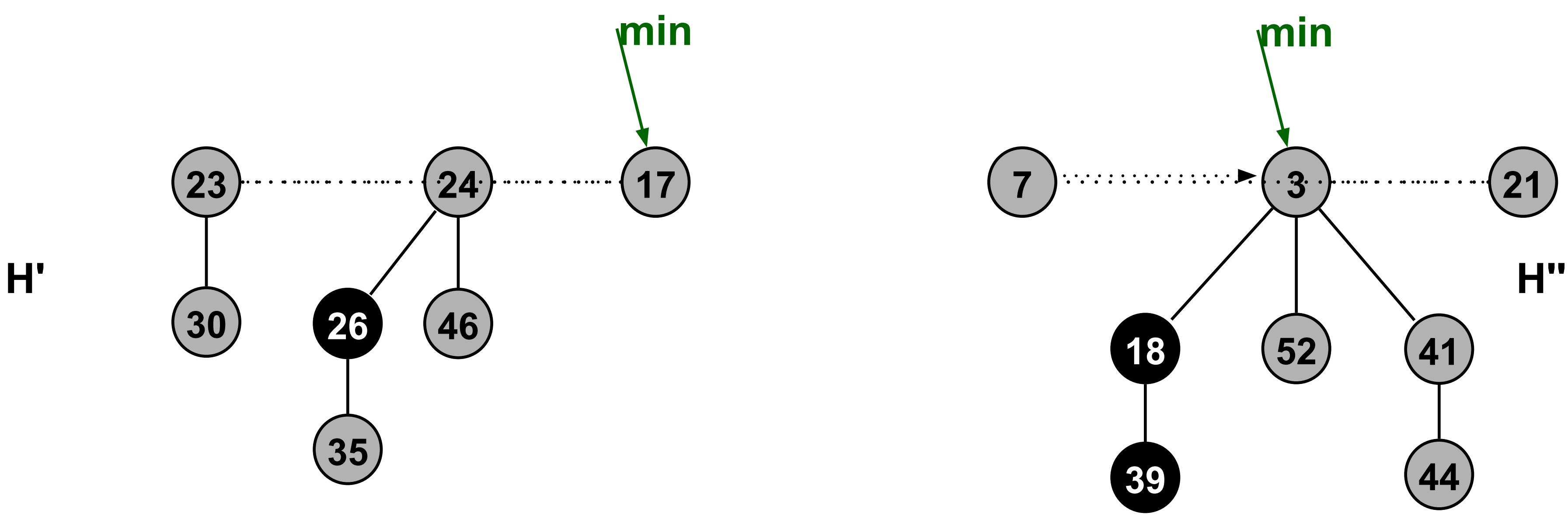
Fibonacci Heaps: Insert

```
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.



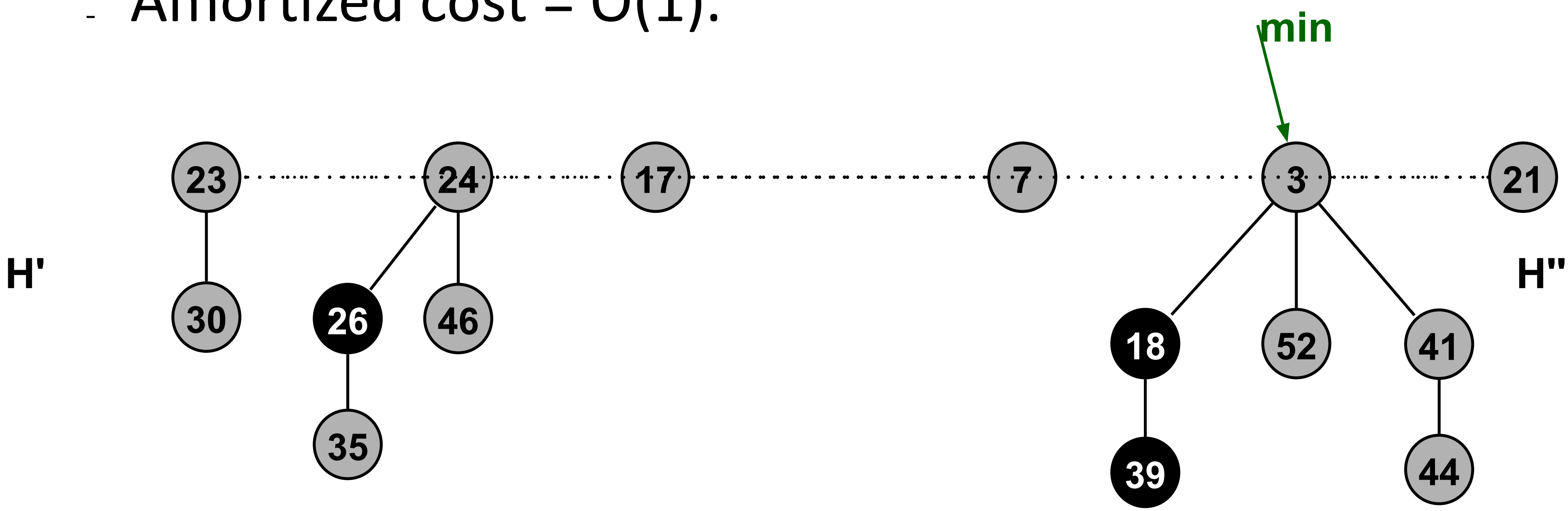
Fibonacci Heaps: Union

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).



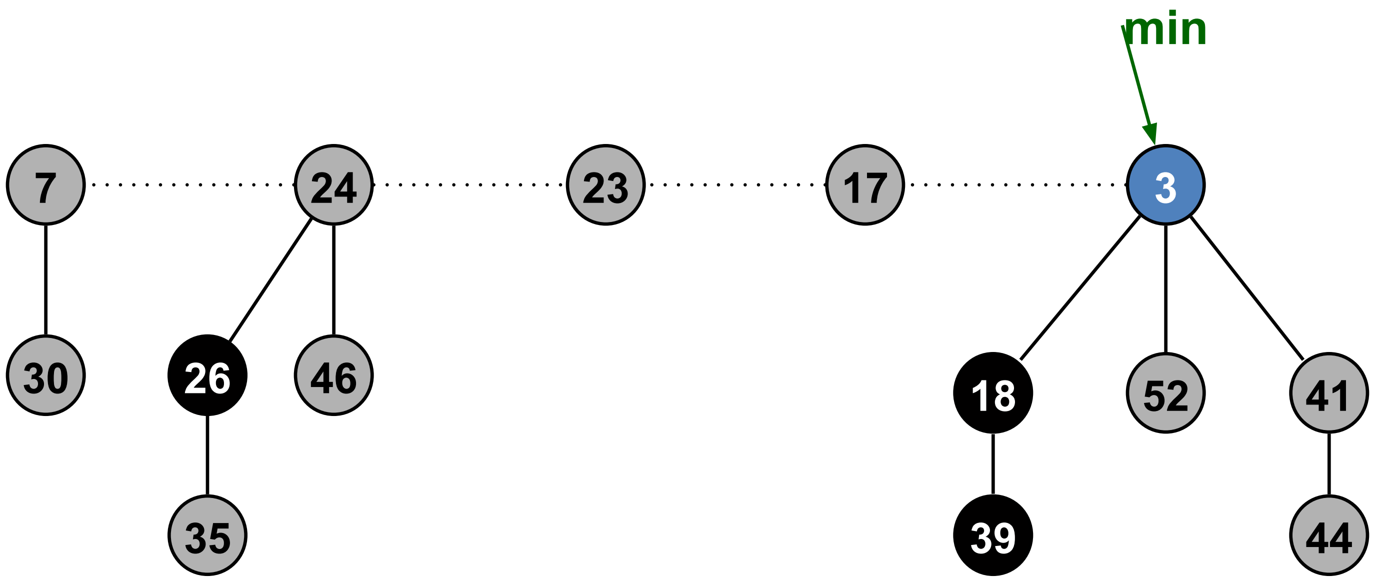
Fibonacci Heaps: Union

```
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: Extract Min

Delete min.

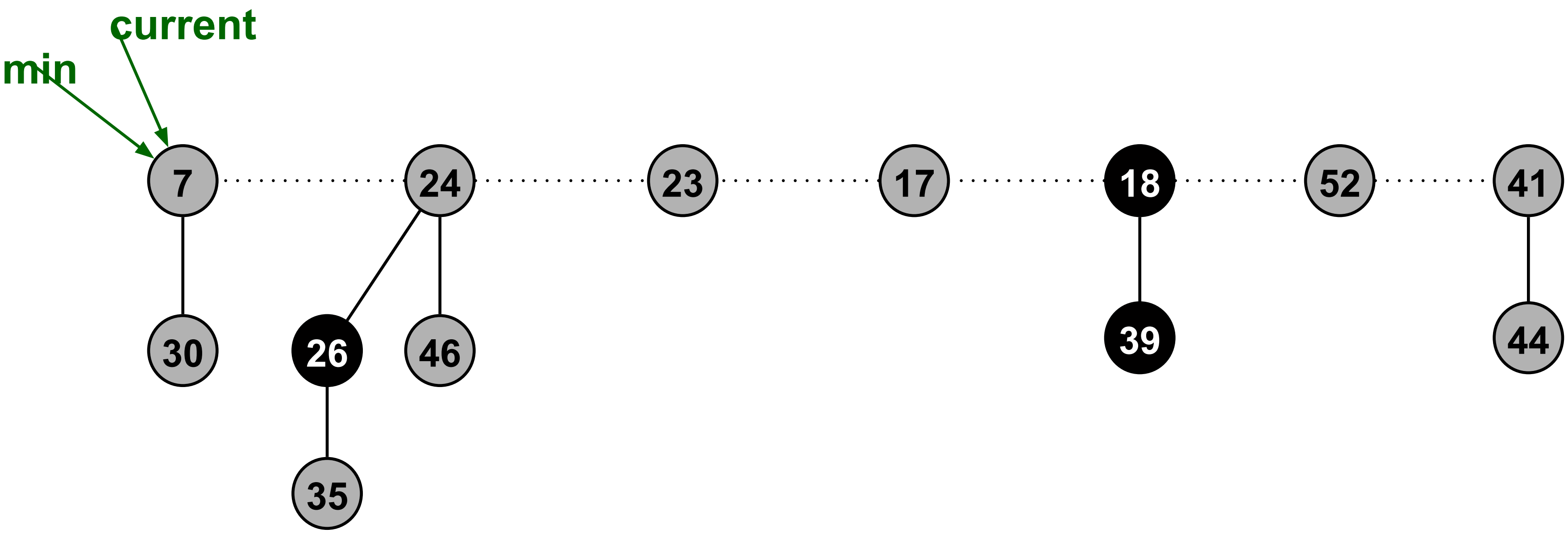
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract Min

Delete min.

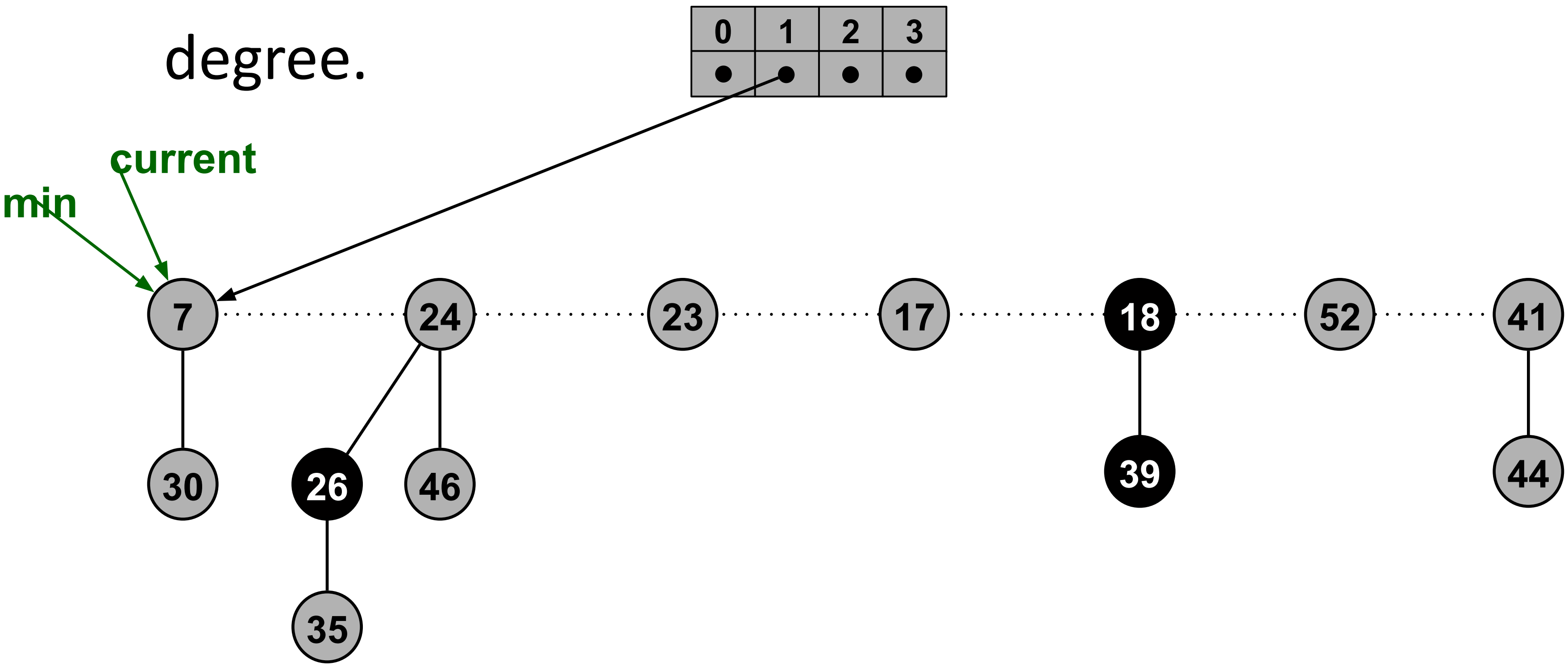
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract Min

Delete min.

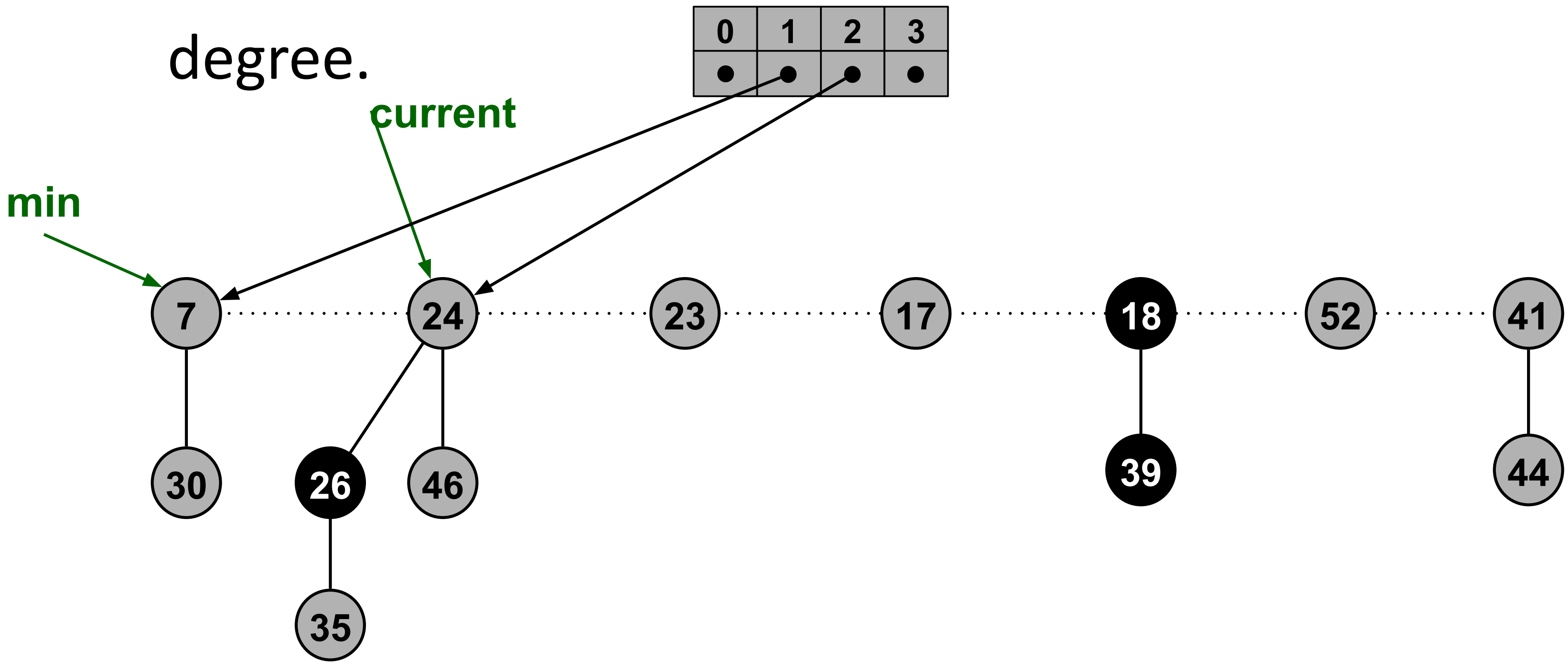
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract Min

Delete min.

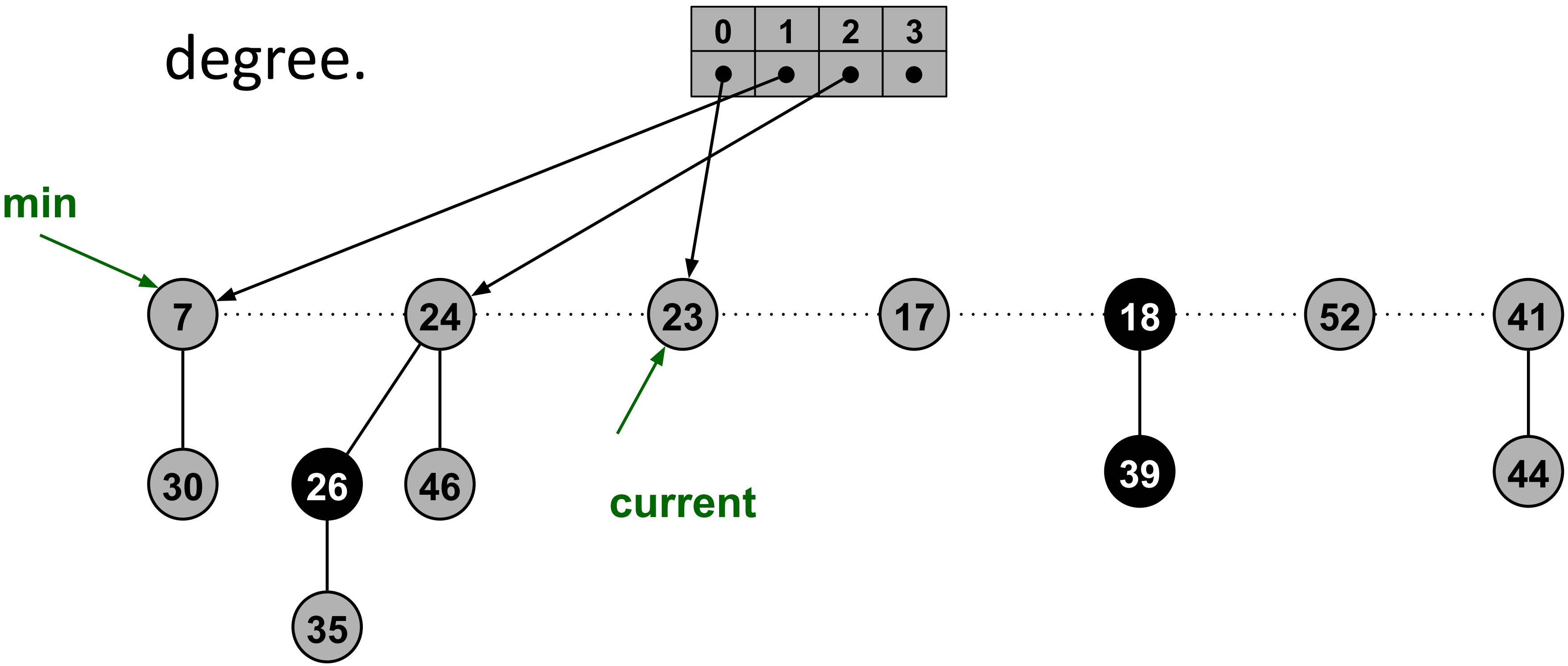
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract Min

Delete min.

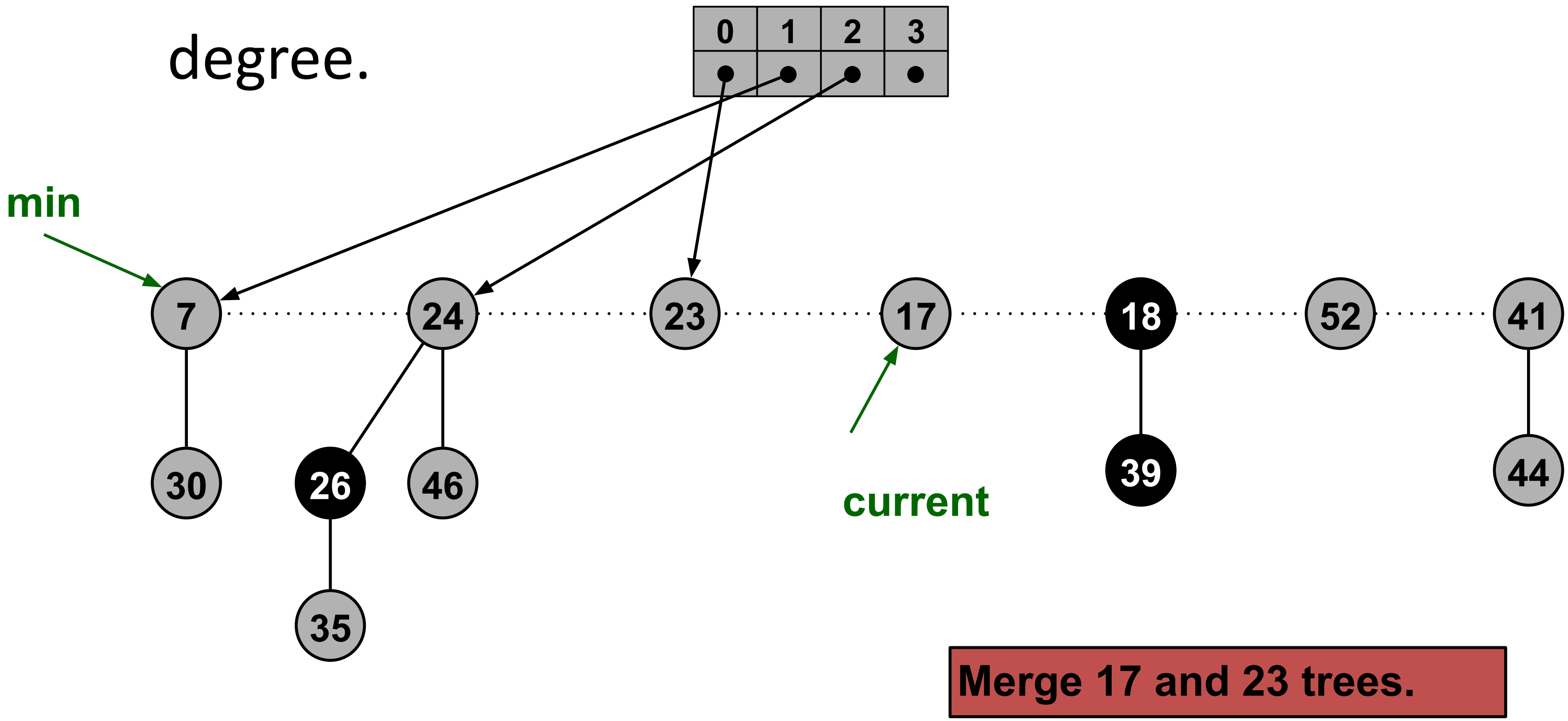
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract Min

Delete min.

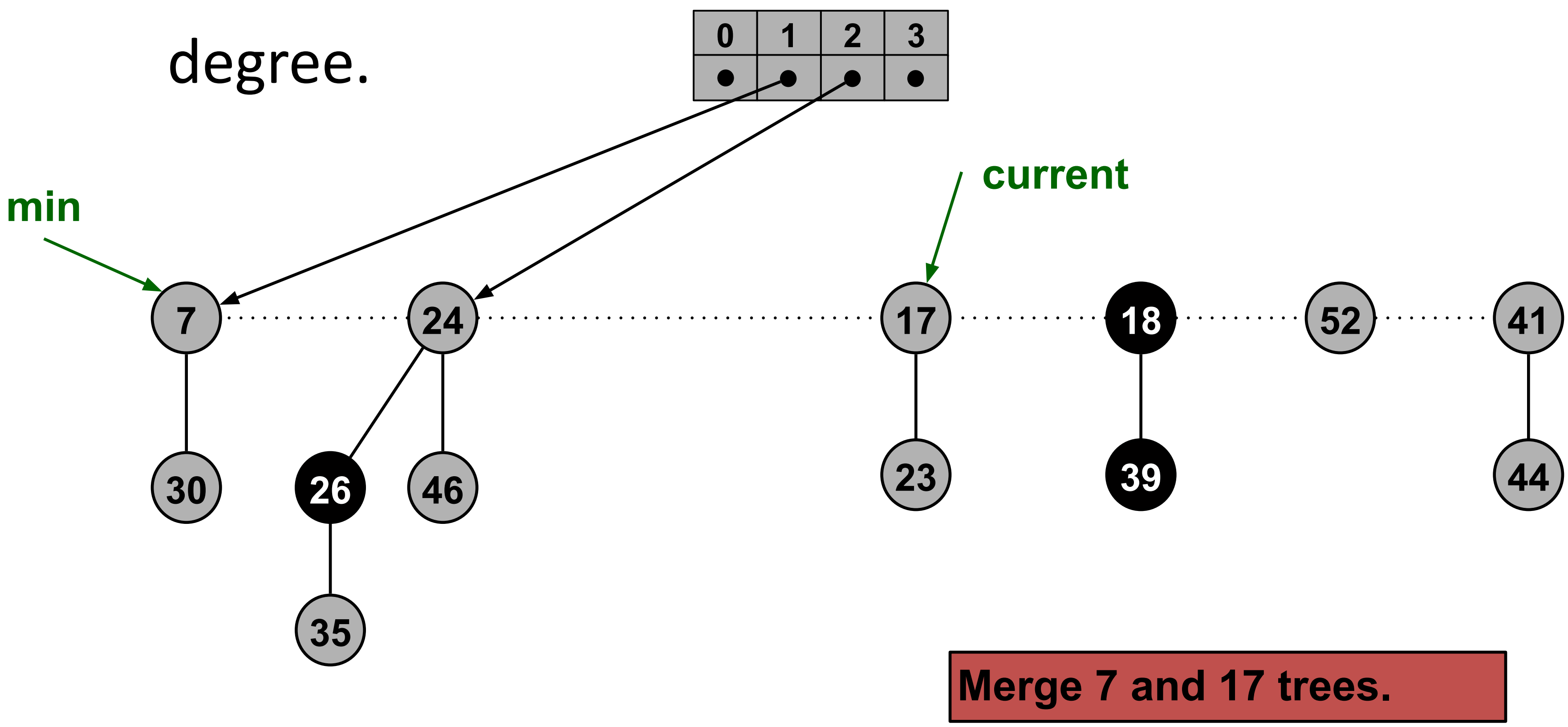
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract Min

Delete min.

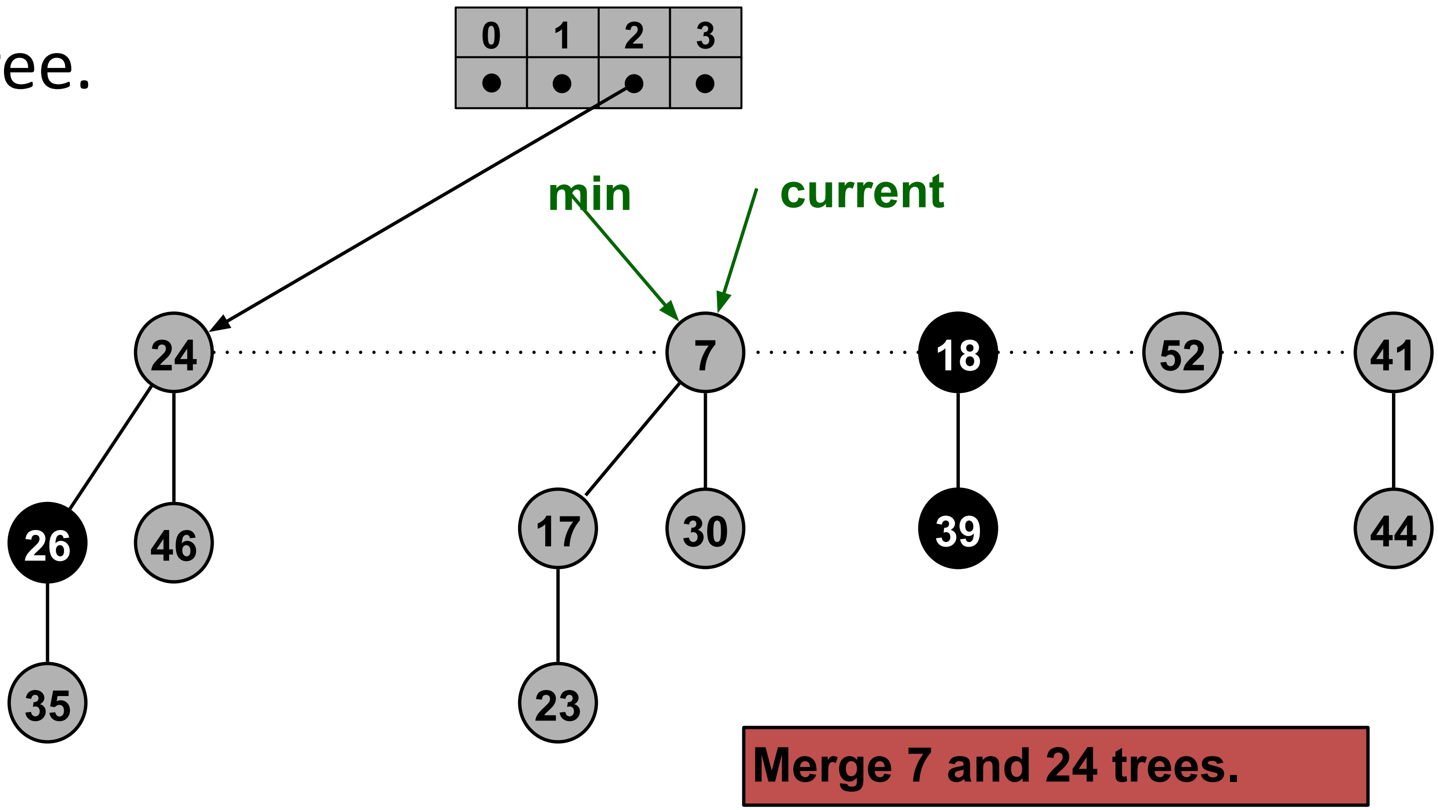
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract Min

Delete min.

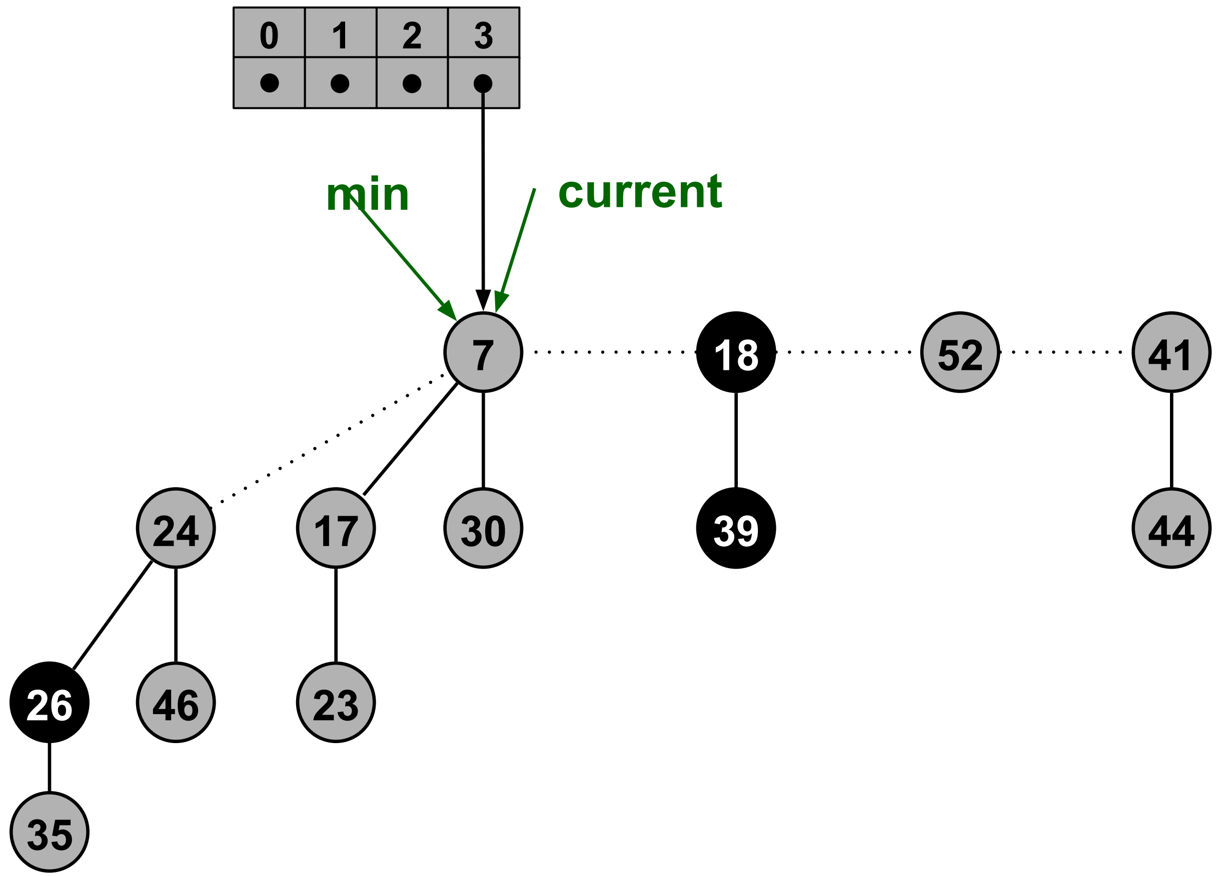
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract Min

Delete min.

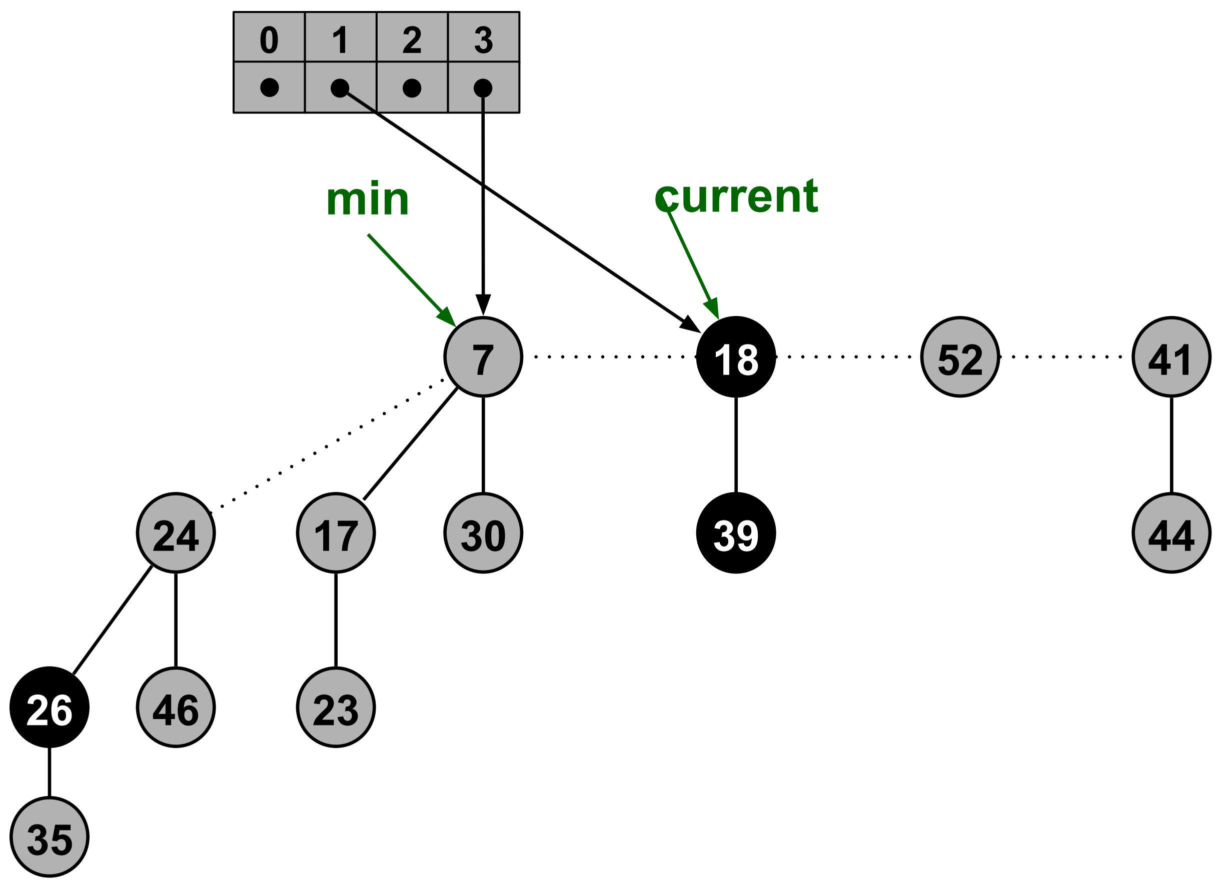
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract Min

Delete min.

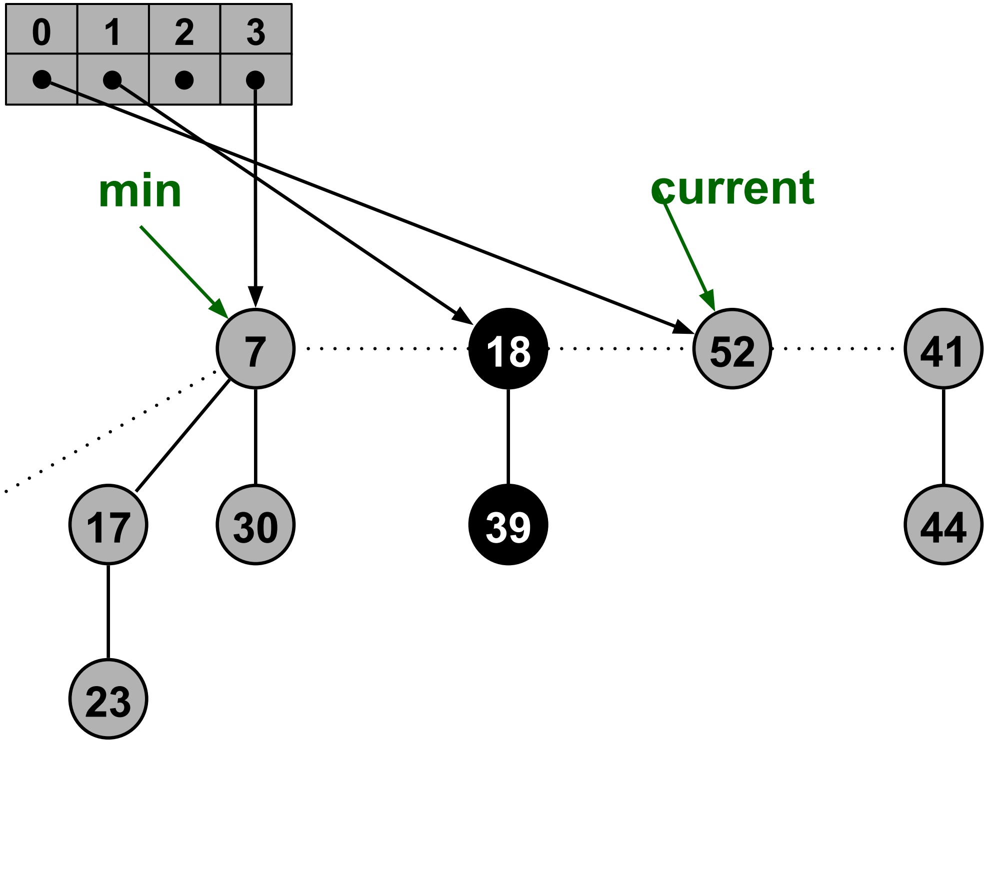
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract Min

Delete min.

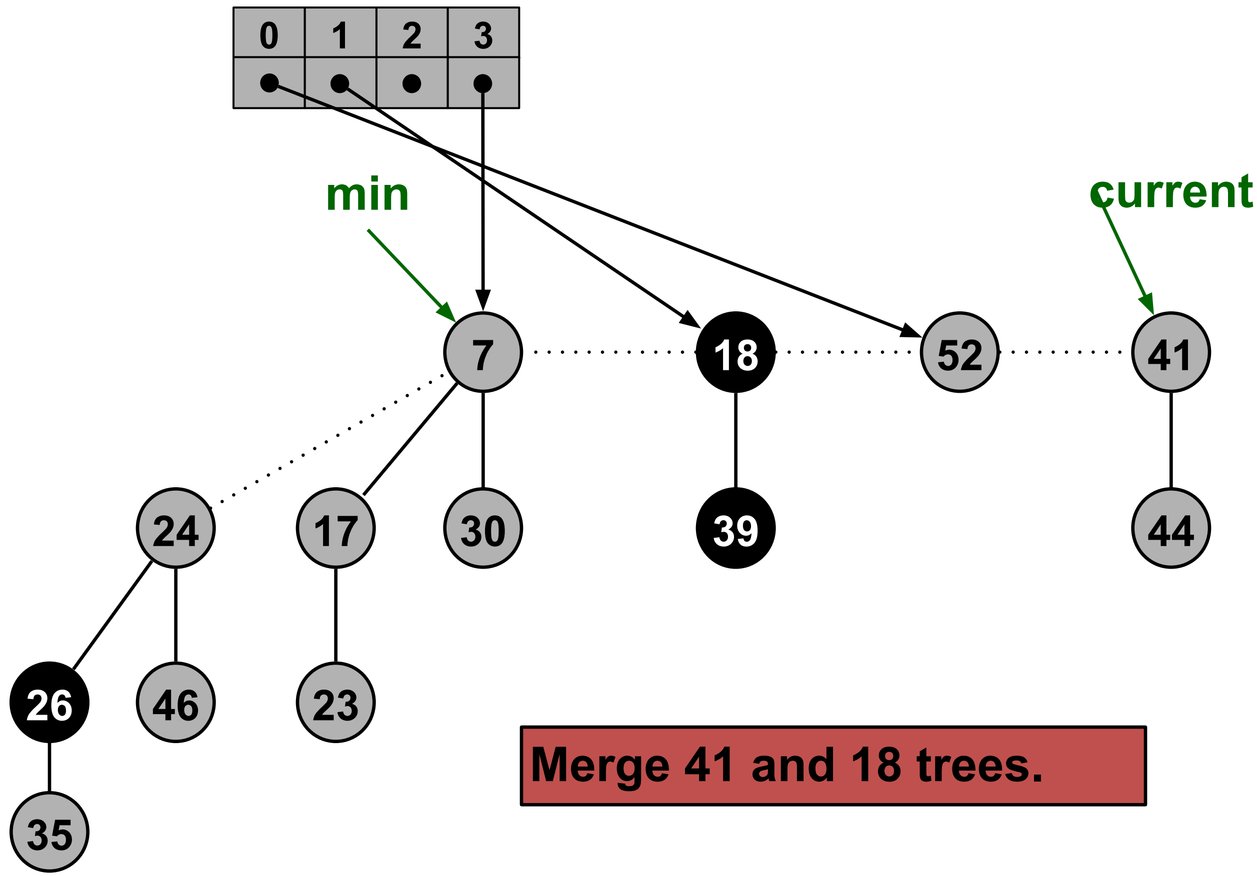
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract Min

Delete min.

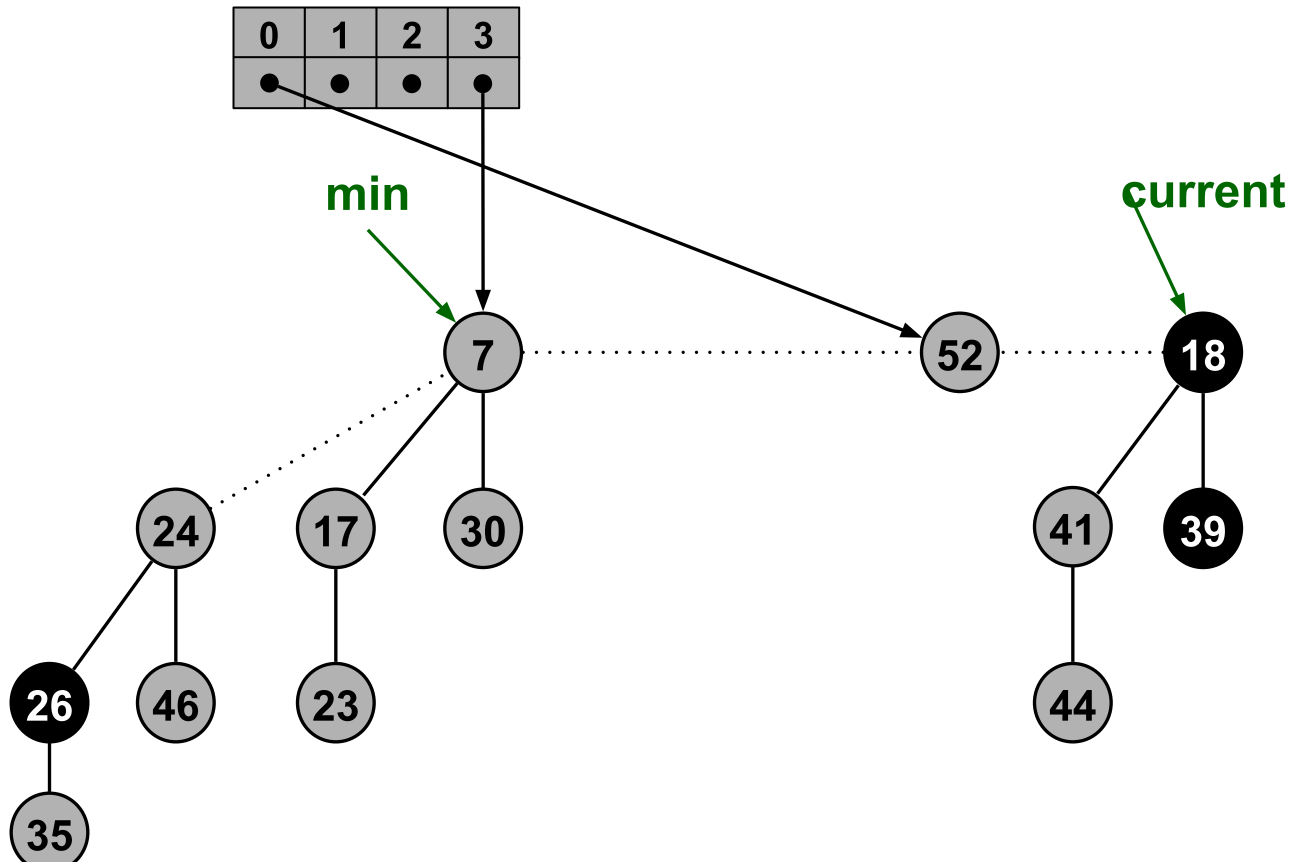
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract Min

Delete min.

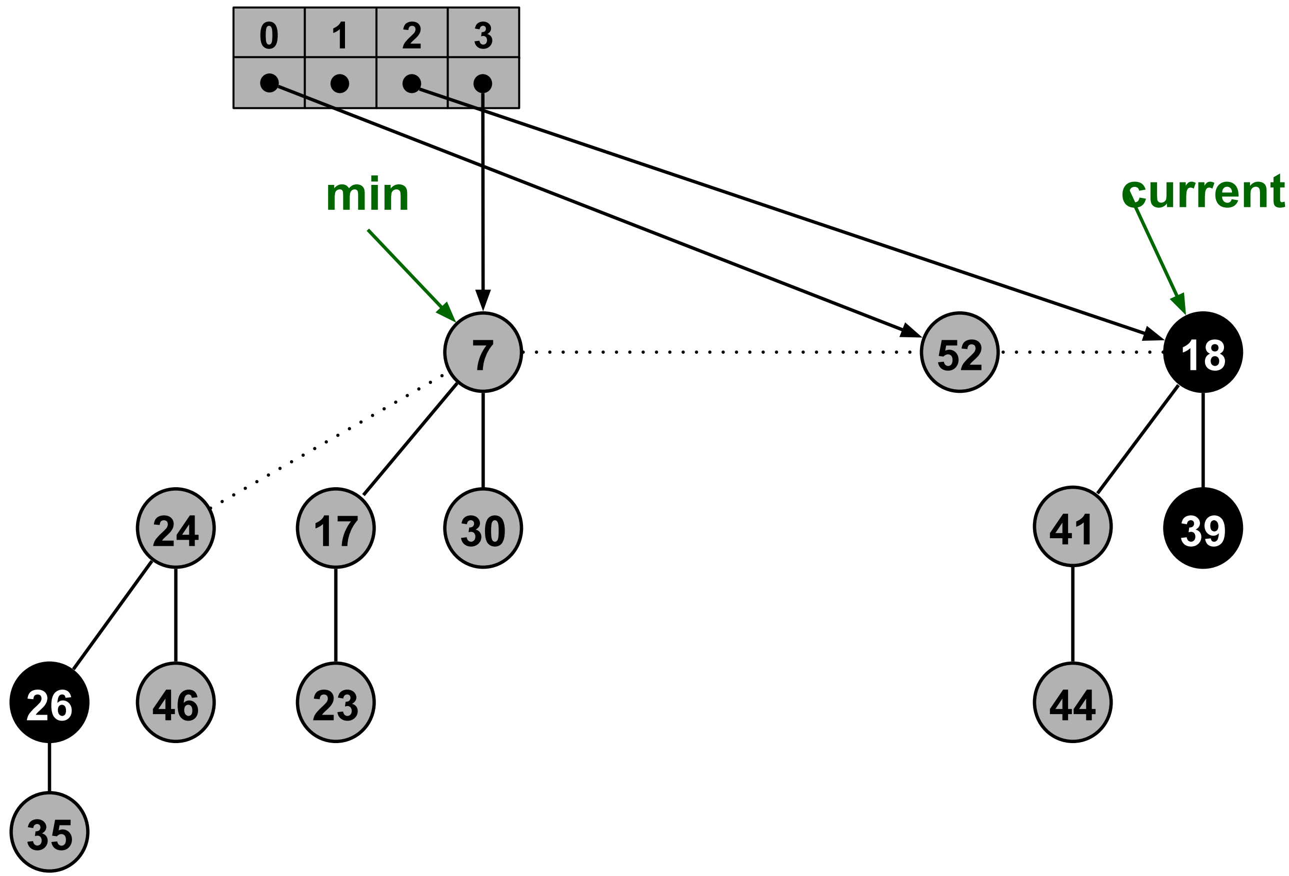
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract Min

Delete min.

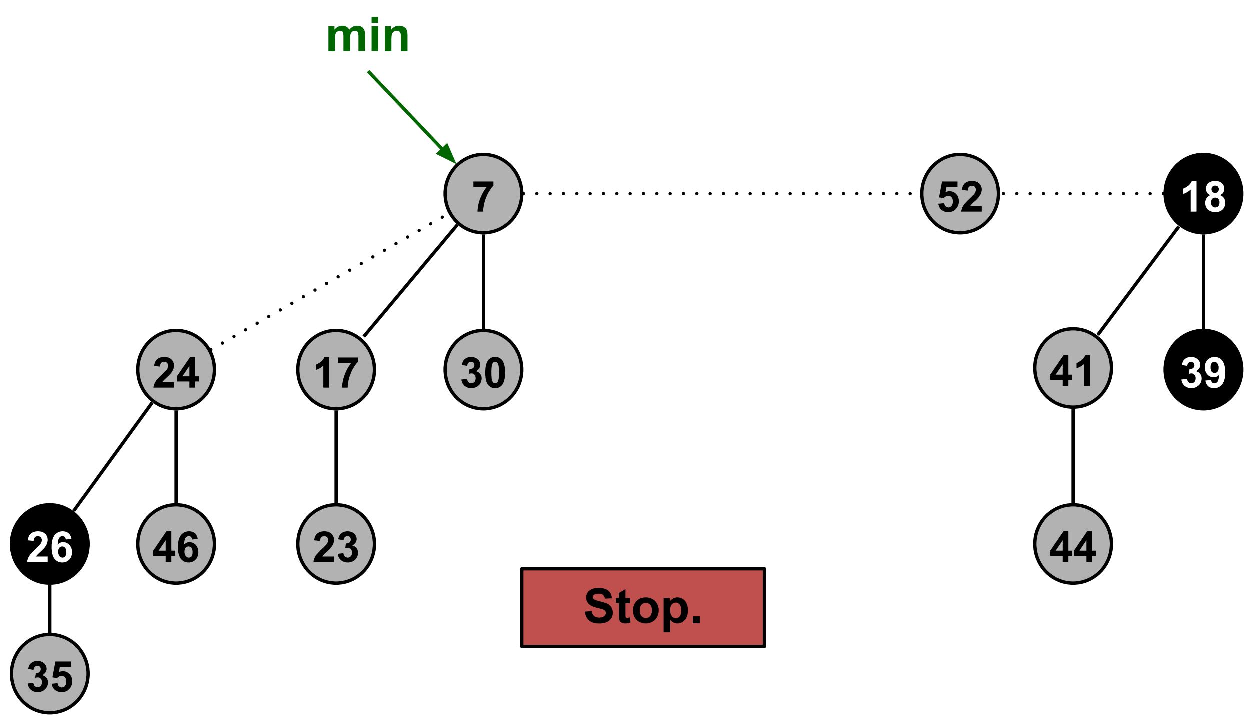
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Extract Min

Delete min.

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



Fibonacci Heaps: Extract Min

```
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
```

Fibonacci Heaps: Extract Min

```
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]
```

Fibonacci Heaps: Extract Min

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

remove y from the root list of H
make y a child of x, incrementing $x.degree$
$y.mark = \text{FALSE}$

Fibonacci Heaps: Extract 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: Extract Min Analysis

- The potential before extracting the minimum node is t(H) + 2m(H)
- The potential afterward is at most D(n)+1+ 2m(H), since
  - At most D(n)+1 roots remain (no two nodes of same degree)
  - No nodes become marked during the operation
- The amortized cost is thus at most

$$\begin{aligned} & O(D(n) + t(H)) + ((D(n) + 1) + 2m(H)) - (t(H) + 2m(H)) \\ & \quad = O(D(n)) + O(t(H)) - t(H) \\ & \quad = O(D(n)) , \end{aligned}$$

Fibonacci Heaps: Extract Min Analysis

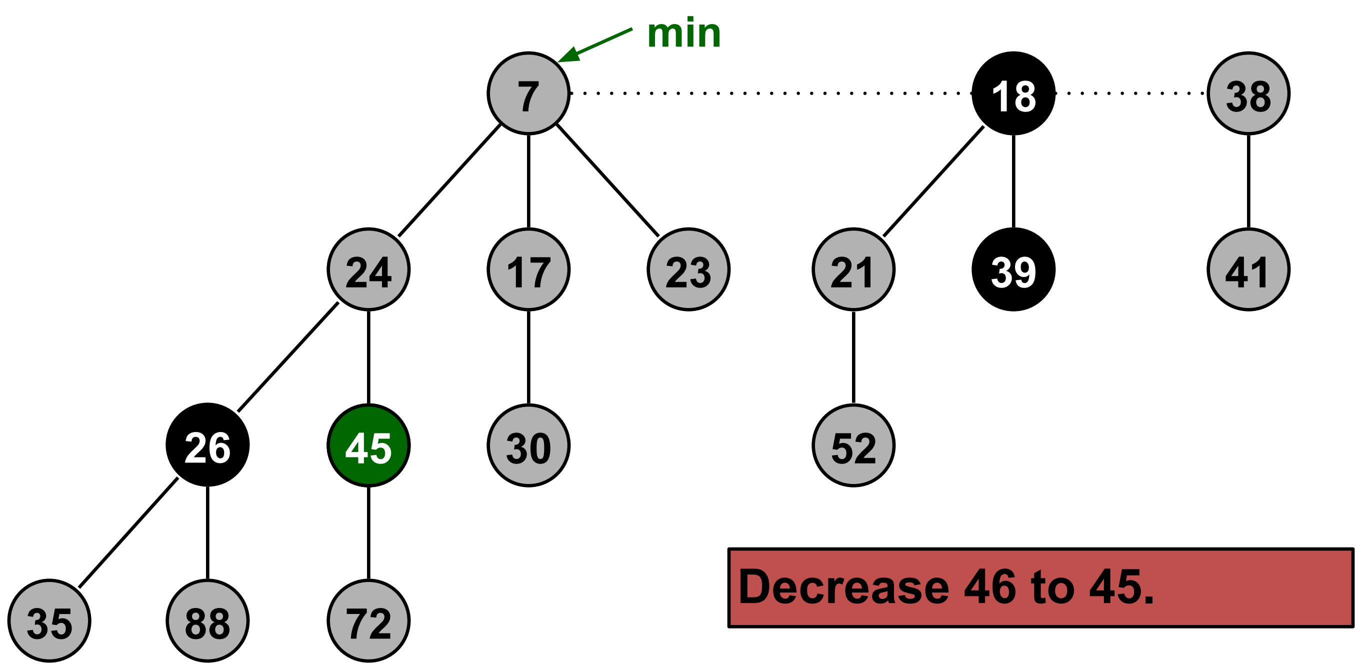
Is amortized cost of O(D(n)) good?

- 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 $D(n) \leq \lfloor \log_\varphi N \rfloor$, where φ is golden ratio
  - $\varphi^2 = 1 + \varphi$
  - $\varphi = (1 + \sqrt{5}) / 2 = 1.618\ldots$
  - limiting ratio between successive Fibonacci numbers!

Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

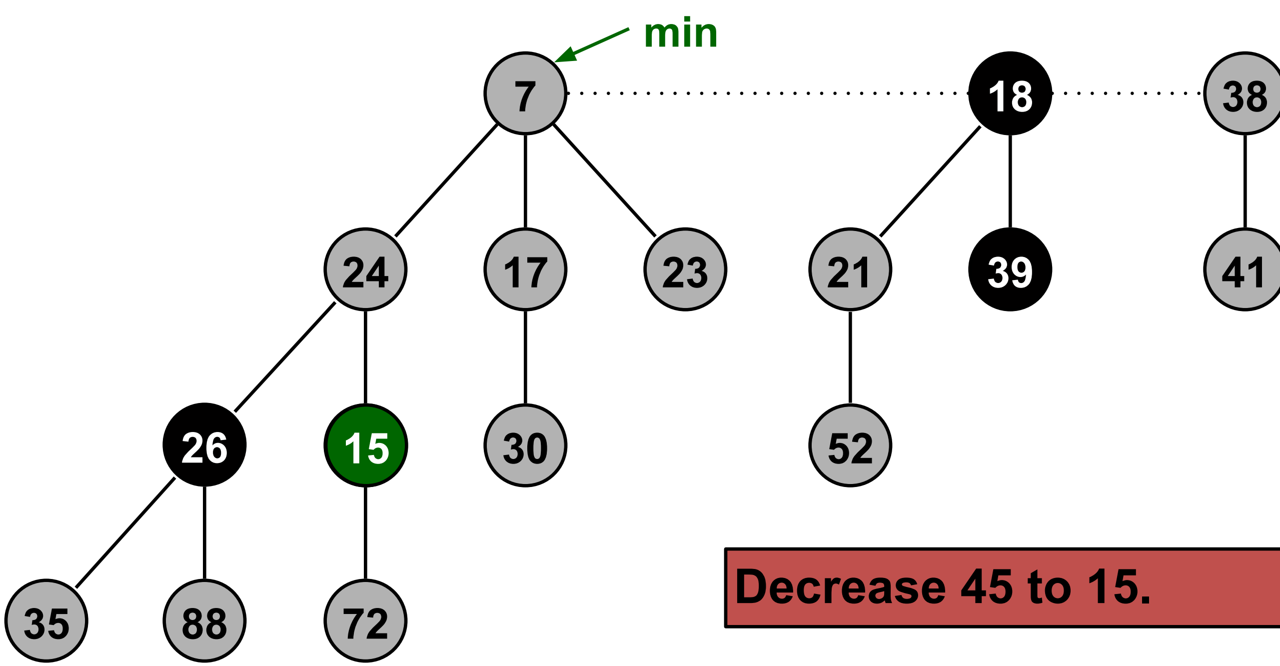
- Case 0: min-heap property not violated.
  - decrease key of x to k
  - change heap min pointer if necessary



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

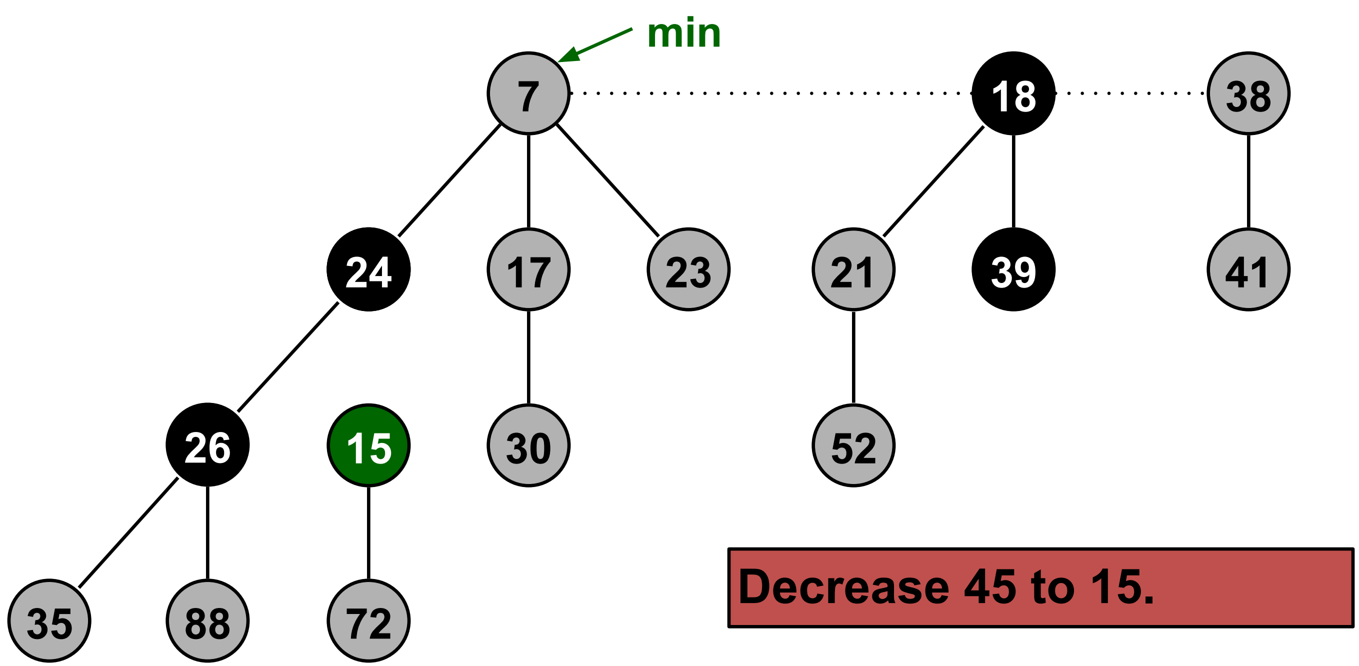


Decrease 45 to 15.

Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

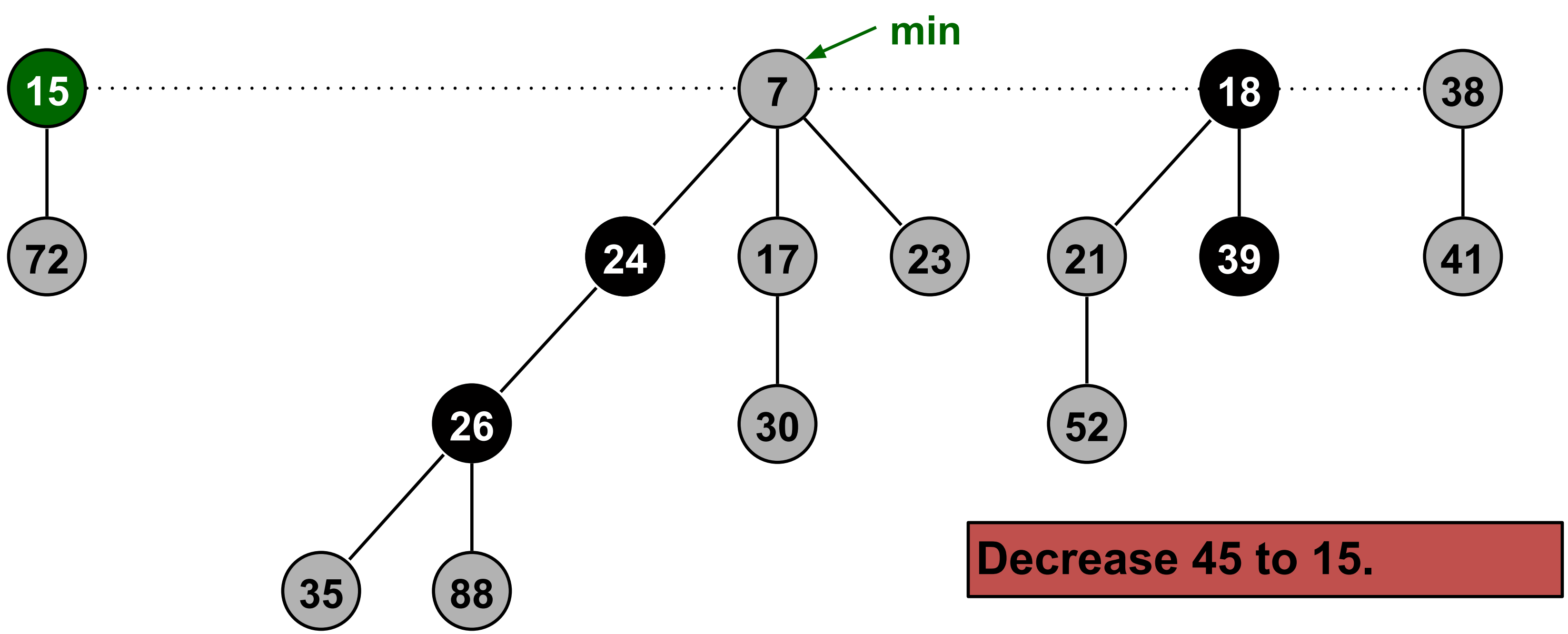
- Case 1: parent of x is unmarked.
  - decrease key of x to k
  - cut off link between x and its parent
  - mark parent
  - add tree rooted at x to root list, updating heap min pointer



Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

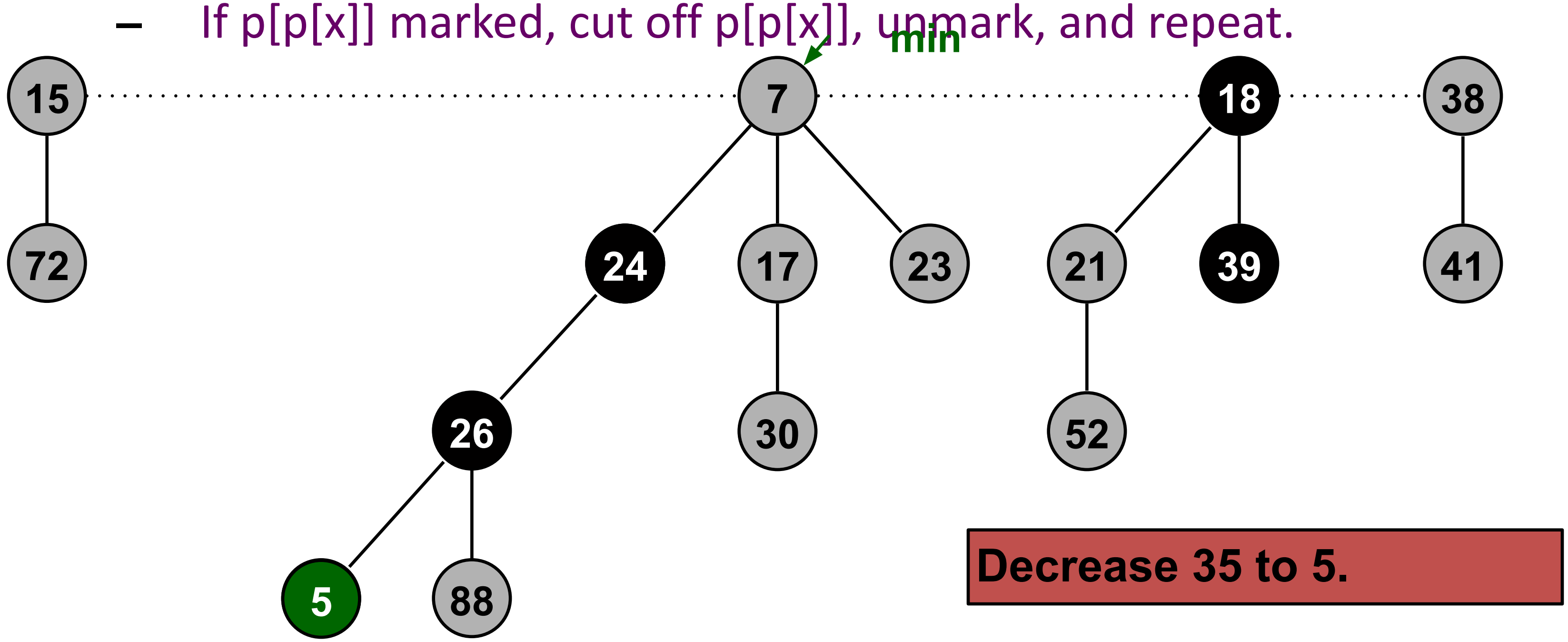
- Case 1: parent of x is unmarked.
  - decrease key of x to k
  - cut off link between x and its parent
  - mark parent
  - add tree rooted at x to root list, updating heap min pointer



Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

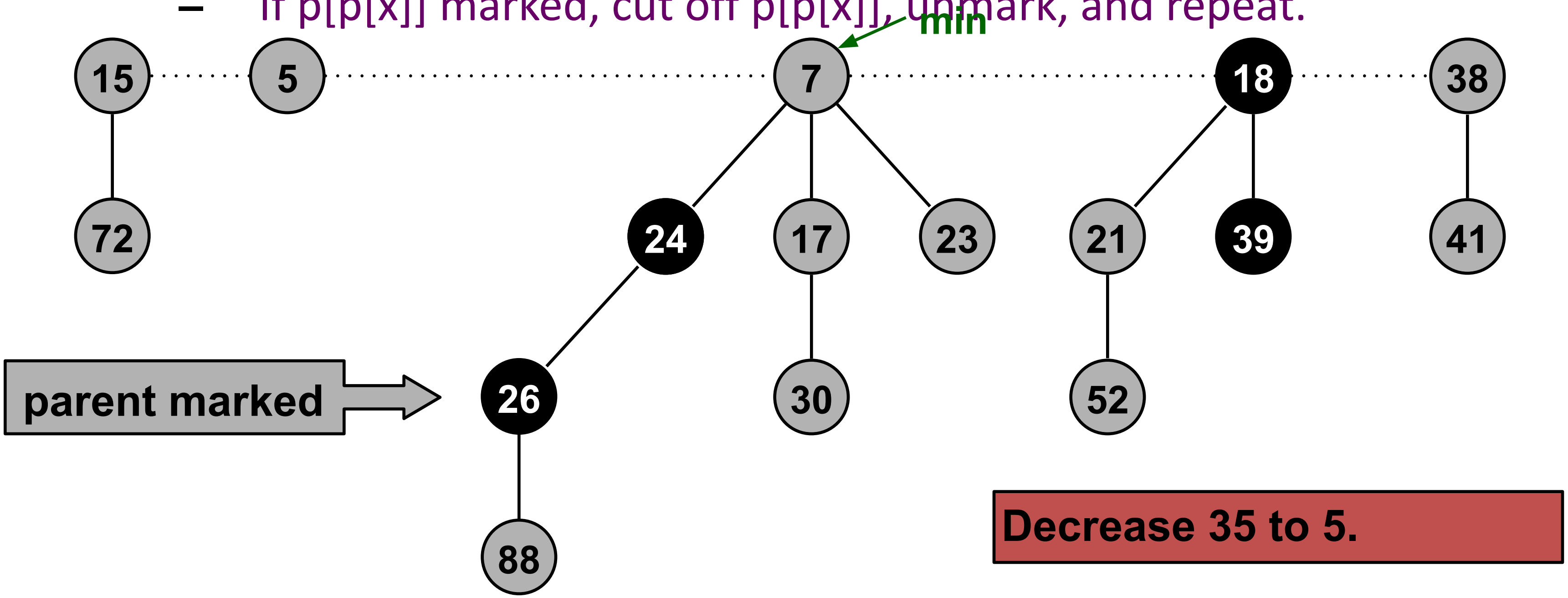
- Case 2: parent of x is marked.
  - decrease key of x to k
  - cut off link between x and its parent p[x], and add x to root list
  - cut off link between p[x] and p[p[x]], add p[x] to root list
    - If p[p[x]] unmarked, then mark it.
    - If p[p[x]] marked, cut off p[p[x]], unmark, and repeat.



Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

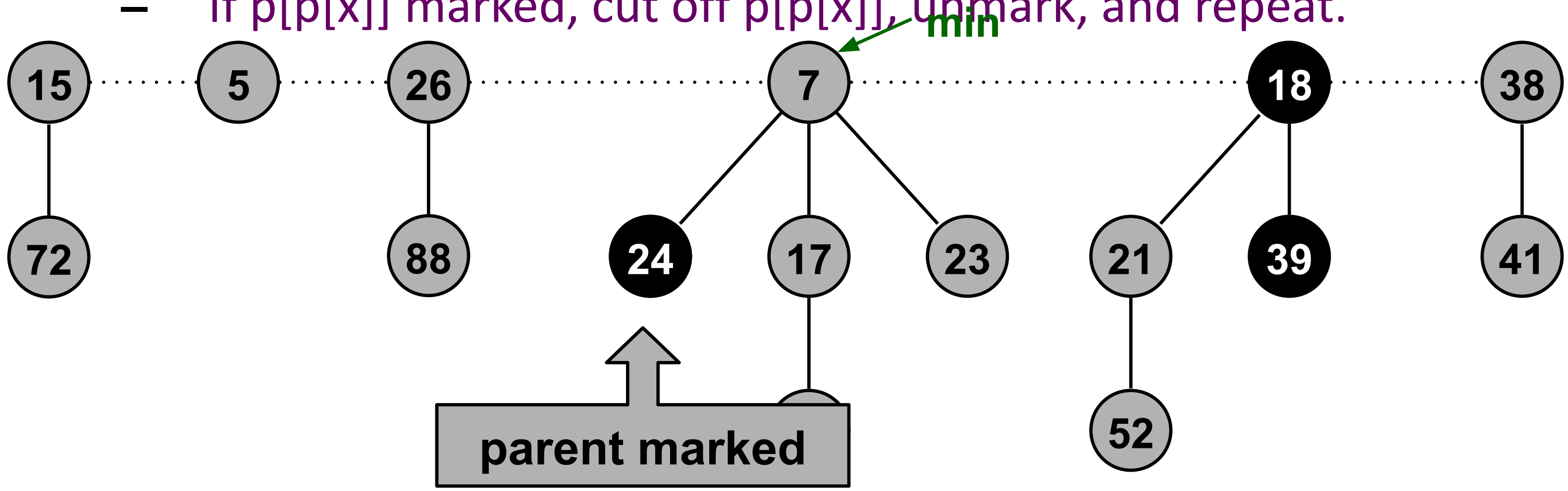
- Case 2: parent of x is marked.
  - decrease key of x to k
  - cut off link between x and its parent p[x], and add x to root list
  - cut off link between p[x] and p[p[x]], add p[x] to root list
    - If p[p[x]] unmarked, then mark it.
    - If p[p[x]] marked, cut off p[p[x]], unmark, and repeat.



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.

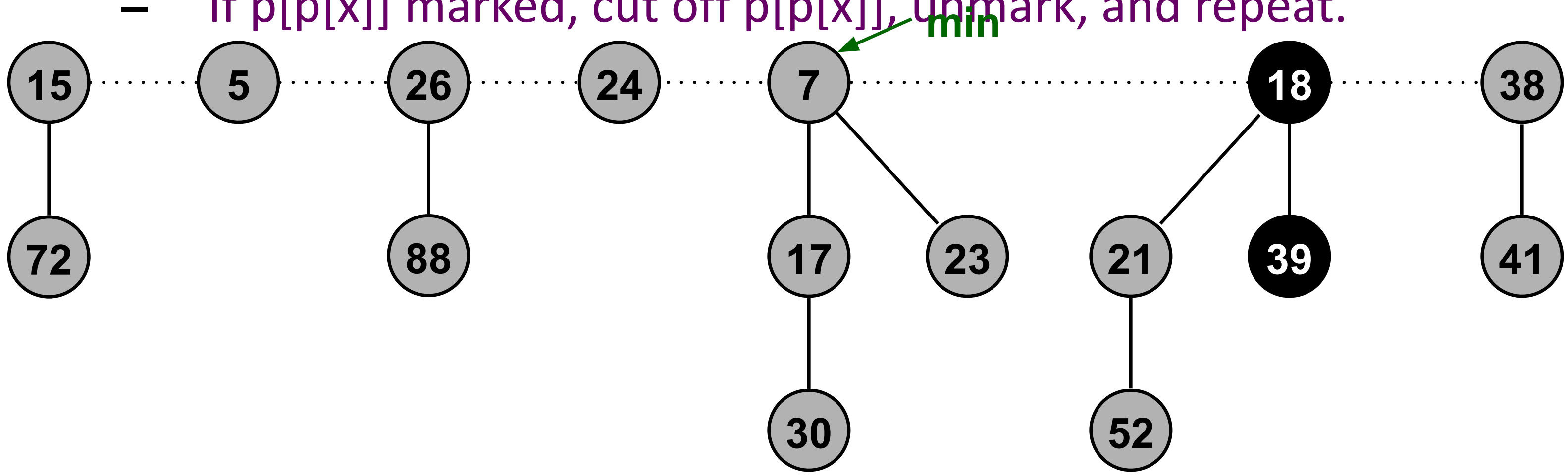


Decrease 35 to 5.

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

```
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
```

Marking nodes

- We use the mark attributes to obtain the desired time bounds.
- They record history of each node.
- Suppose that the following events have happened to node x:
    1. at some time, x was a root
    2. then x was linked to (made the child of) another node
    3. then two children of x were removed by cuts
- As soon as the second child has been lost, we cut x from its parent, making it a new root

Fibonacci Heaps: Decrease Key

```
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. $O(1)$

- $t(H') = t(H) + c$
- $m(H') \leq m(H) - (c-1) + 1 = m(H) - c + 2$
  - each cascading cut unmarks a node
  - last cascading cut could potentially mark a node
- $\Delta\Phi \leq c + 2(-c + 2) = 4 - c$.
- Thus, the amortized cost of FIB-HEAP-DECREASE-KEY is at most

  $O(c) + 4 - c = O(1)$

Fibonacci Heaps: Delete

Delete node x.

- Decrease key of x to -∞.
- Delete min element in heap.

$\text{FIB-HEAP-DELETE}(H, x)$

1 $\text{FIB-HEAP-DECREASE-KEY}(H, x, -\infty)$
2 $\text{FIB-HEAP-EXTRACT-MIN}(H)$

Amortized cost. O(D(n))

- O(1) for decrease-key.
- O(D(n)) for extract-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. $D(N) \le \log_\varphi N$, where $\varphi = (1 + \sqrt{5}) / 2$.
Corollary. Delete and Extract-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 $i - 2$

Lemma 19.2
For all integers $k \geq 0$,

$$F_{k+2} = 1 + \sum_{i=0}^{k} F_i \ .$$

Proof The proof is by induction on k. When $k = 0$,

$$\begin{aligned} 1 + \sum_{i=0}^{0} F_i &= 1 + F_0 \\ &= 1 + 0 \\ &= F_2 \ . \end{aligned}$$

We now assume the inductive hypothesis that $F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$, and we have

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

■

Lemma 19.3
For all integers $k \geq 0$, the $(k+2)$nd Fibonacci number satisfies $F_{k+2} \geq \phi^k$.

Proof The proof is by induction on k. The base cases are for $k = 0$ and $k = 1$. When $k = 0$ we have $F_2 = 1 = \phi^0$, and when $k = 1$ we have $F_3 = 2 > 1.619 > \phi^1$. The inductive step is for $k \geq 2$, and we assume that $F_{i+2} > \phi^i$ for $i = 0, 1, \ldots, k-1$. Recall that ϕ is the positive root of equation (3.23), $x^2 = x + 1$. Thus, we have

$$\begin{aligned}
F_{k+2} &= F_{k+1} + F_k \\
&\geq \phi^{k-1} + \phi^{k-2} && \text{(by the inductive hypothesis)} \\
&= \phi^{k-2}(\phi + 1) \\
&= \phi^{k-2} \cdot \phi^2 && \text{(by equation (3.23))} \\
&= \phi^k .
\end{aligned}$$

∎

Fibonacci Heaps: Bounding Max Degree

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

Proof of key lemma.

- For any node x, we show that size(x) $\geq \varphi^{\text{degree(x)}}$.
  - size(x) = # node in subtree rooted at x
  - taking base φ logs, degree(x) $\leq \log_\varphi$ (size(x)) $\leq \log_\varphi N$.
- 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 at least size s_k, and let $y_1, \ldots, y_k$ be children in order that they were linked to x.

$$
\begin{aligned}
\text{size}(x) &\geq s_k \\
&\geq 2 + \sum_{i=2}^{k} s_{y_i.degree} \\
&\geq 2 + \sum_{i=2}^{k} s_{i-2} ,
\end{aligned}
$$

We now show by induction on k that $s_k \geq F_{k+2}$ for all nonnegative integers k. The bases, for $k = 0$ and $k = 1$, are trivial. For the inductive step, we assume that $k \geq 2$ and that $s_i \geq F_{i+2}$ for $i = 0, 1, \ldots, k-1$. We have

$$
\begin{aligned}
s_k &\geq 2 + \sum_{i=2}^{k} s_{i-2} \\
&\geq 2 + \sum_{i=2}^{k} F_i \\
&= 1 + \sum_{i=0}^{k} F_i \\
&= F_{k+2} && \text{(by Lemma 19.2)} \\
&\geq \phi^k && \text{(by Lemma 19.3)}\ .
\end{aligned}
$$

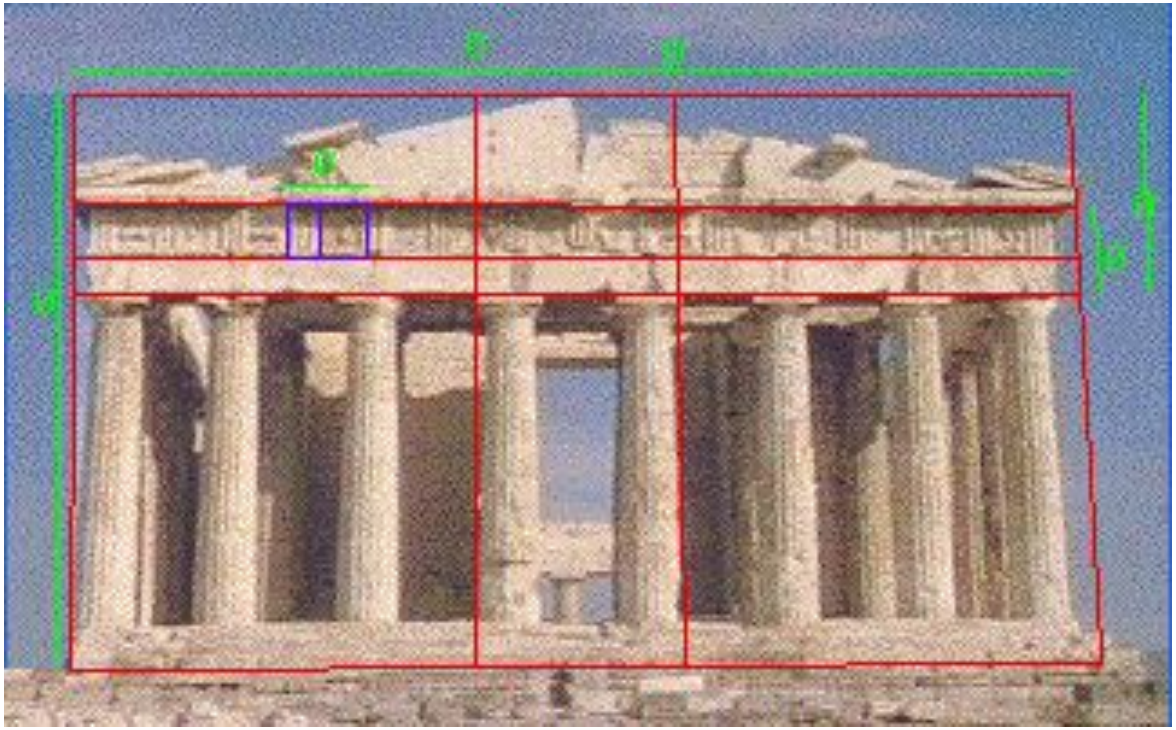
Thus, we have shown that $\text{size}(x) \geq s_k \geq F_{k+2} \geq \phi^k$. ■

Golden Ratio

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

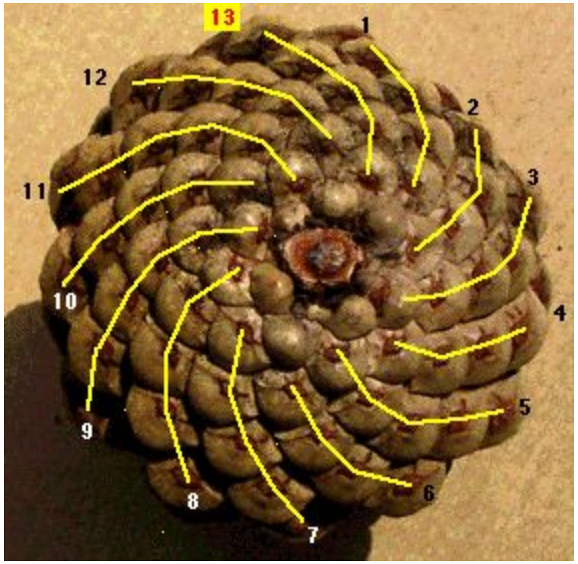
Definition. The golden ratio $\varphi = (1 + \sqrt{5}) / 2 = 1.618\ldots$

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

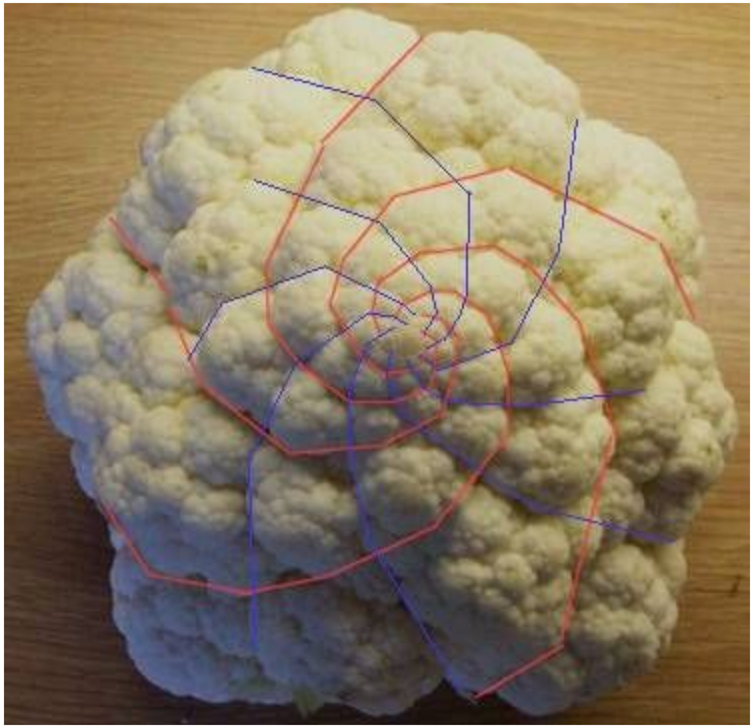


Parthenon, Athens Greece

Fibonacci Numbers and Nature



Pinecone



Cauliflower

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