Fibonacci Heaps

Lecture slides adapted from:

- Chapter 20 of *Introduction to Algorithms* by Cormen, Leiserson, Rivest, and Stein.
- Chapter 9 of *The Design and Analysis of Algorithms* by Dexter Kozen.

Priority Queues Performance Cost Summary

Operation	Linked List	Binary Heap	Binomial Heap	Fibonacci Heap †	Relaxed Heap
make-heap	1	1	1	1	1
is-empty	1	1	1	1	1
insert	1	$\log n$	$\log n$	1	1
delete-min	n	$\log n$	$\log n$	$\log n$	$\log n$
decrease-key	n	$\log n$	$\log n$	1	1
delete	n	$\log n$	$\log n$	$\log n$	$\log n$
union	1	n	$\log n$	1	1
find-min	n	1	$\log n$	1	1

n = number of elements in priority queue

† amortized

Theorem. Starting from empty Fibonacci heap, any sequence of a_1 *insert*, a_2 *delete-min*, and a_3 *decrease-key* operations takes $O(a_1 + a_2 \log n + a_3)$ time.

Priority Queues Performance Cost Summary

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

n = number of elements in priority queue

† amortized

Hopeless challenge. O(1) *insert*, *delete-min* and *decrease-key*. Why?

Fibonacci Heaps

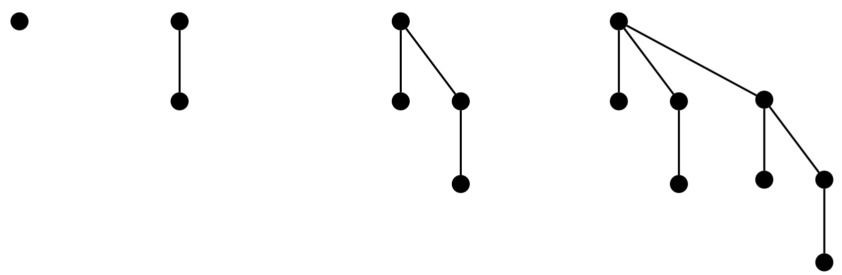
History. [Fredman and Tarjan, 1986]

- Ingenious data structure and analysis.
- Original motivation: improve Dijkstra's shortest path algorithm from $O(E \log V)$ to $O(E + V \log V)$.

V insert, V delete-min, E decrease-key

Basic idea.

- Similar to binomial heaps, but less rigid structure.
- Binomial heap: eagerly consolidate trees after each *insert*.



- Fibonacci heap: lazily defer consolidation until next *delete-min*.

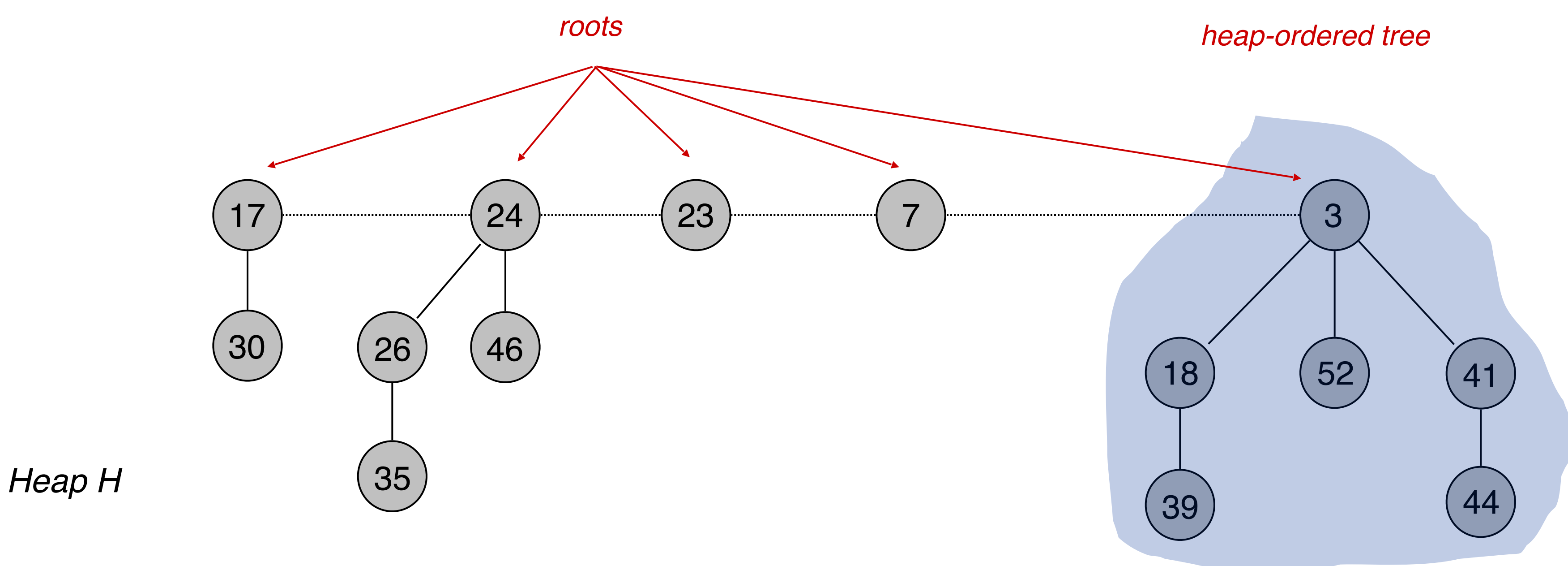
Fibonacci Heap Structure

Fibonacci Heaps: Structure

each parent larger than its children

Fibonacci heap.

- Set of heap-ordered trees.
- Maintain pointer to minimum element.
- Set of marked nodes.

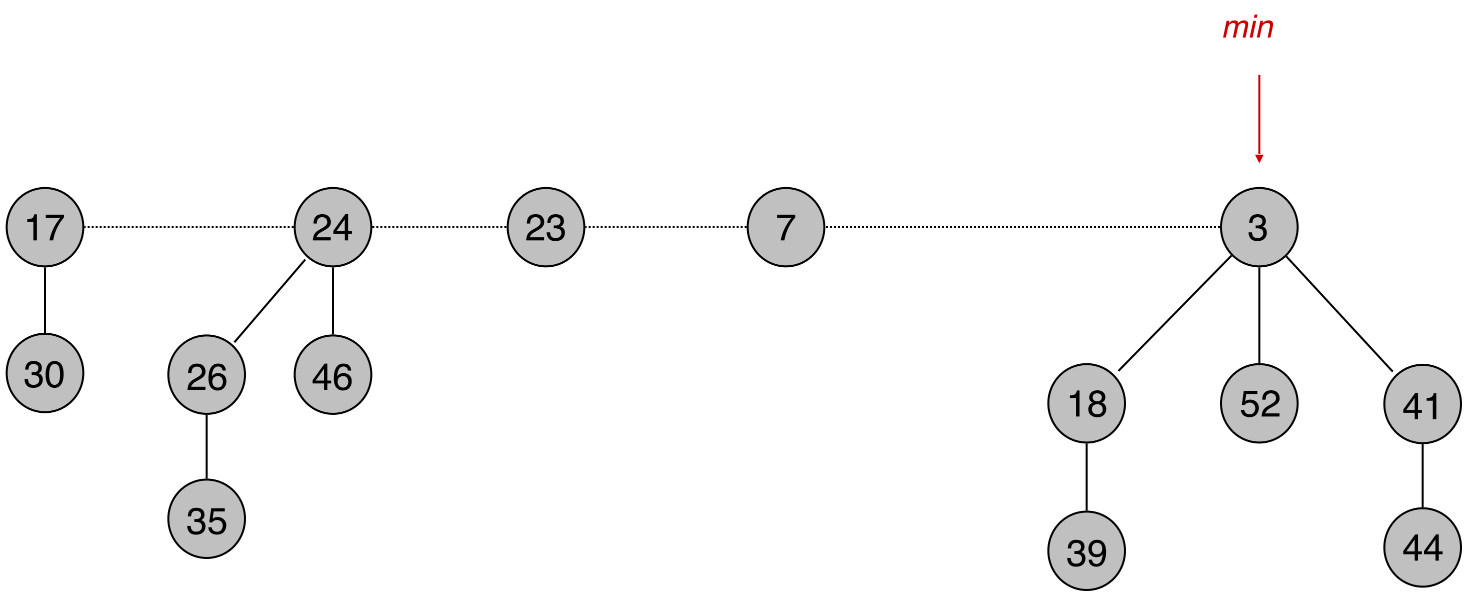


Fibonacci Heaps: Structure

Fibonacci heap.

- Set of heap-ordered trees.
- Maintain pointer to minimum element.
- Set of marked nodes.

find-min takes O(1) time



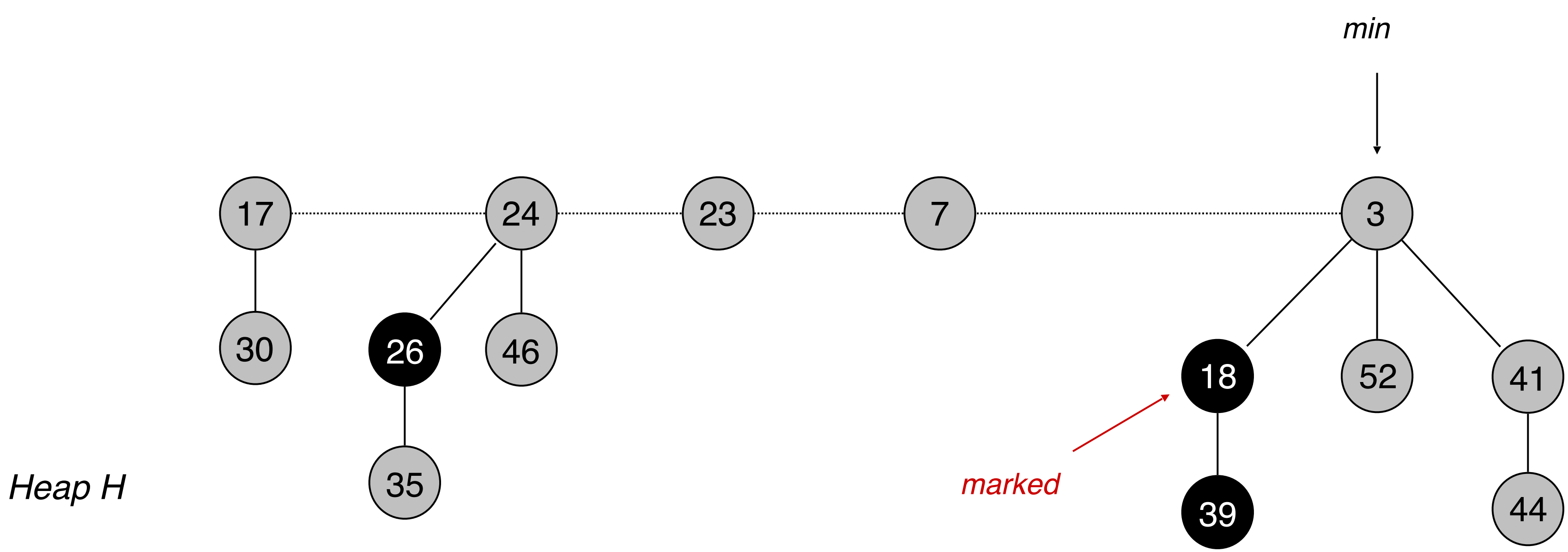
Heap H

Fibonacci Heaps: Structure

Fibonacci heap.

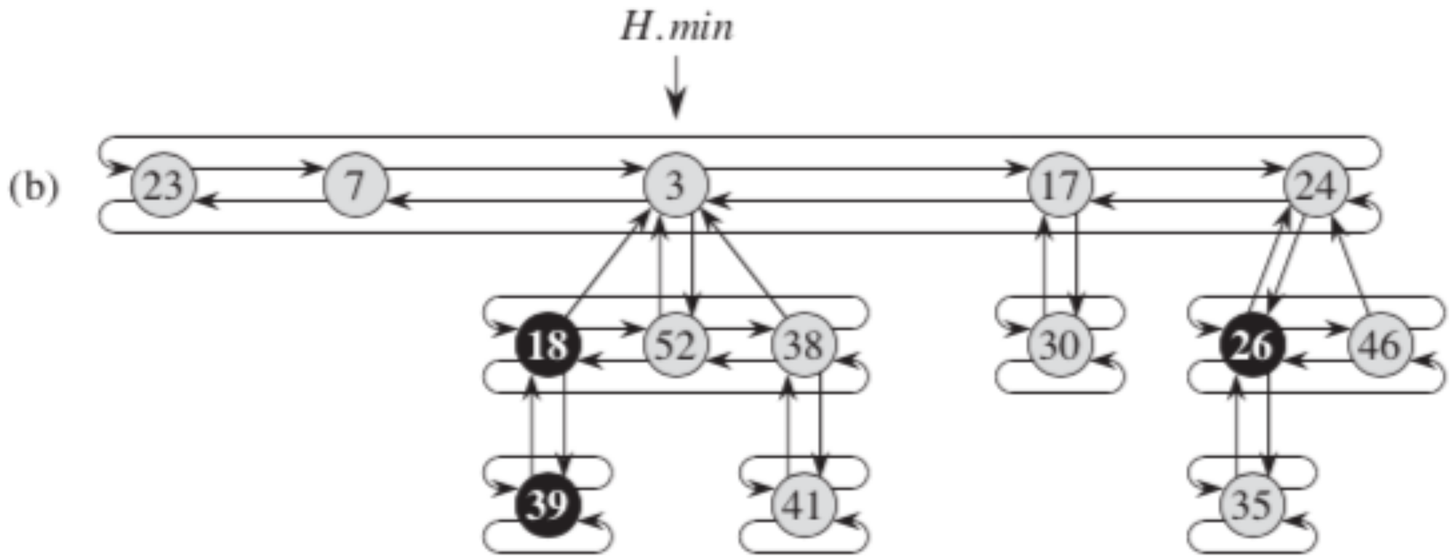
- Set of heap-ordered trees.
- Maintain pointer to minimum element.
- Set of marked nodes.

use to keep heaps flat (stay tuned)



Fibonacci heap.

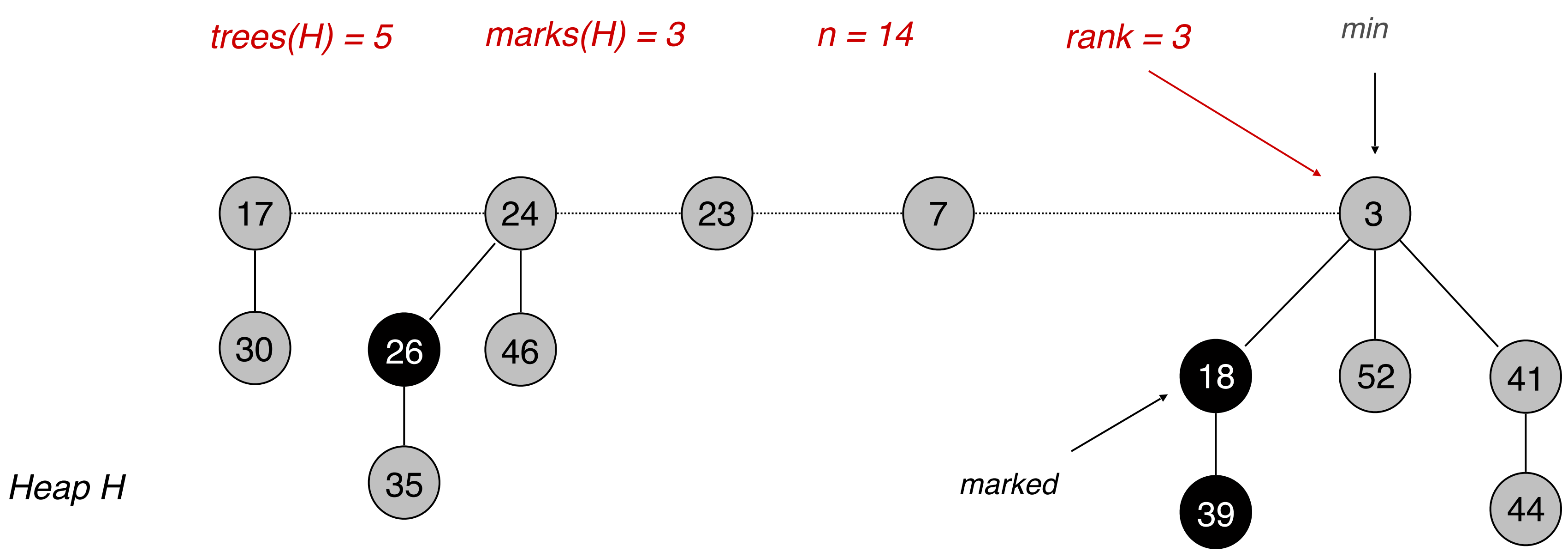
- Set of heap-ordered trees.
- Maintain pointer to minimum element.
- Set of marked nodes.
- Doubly linked list between children of any node



Fibonacci Heaps: Notation

Notation.

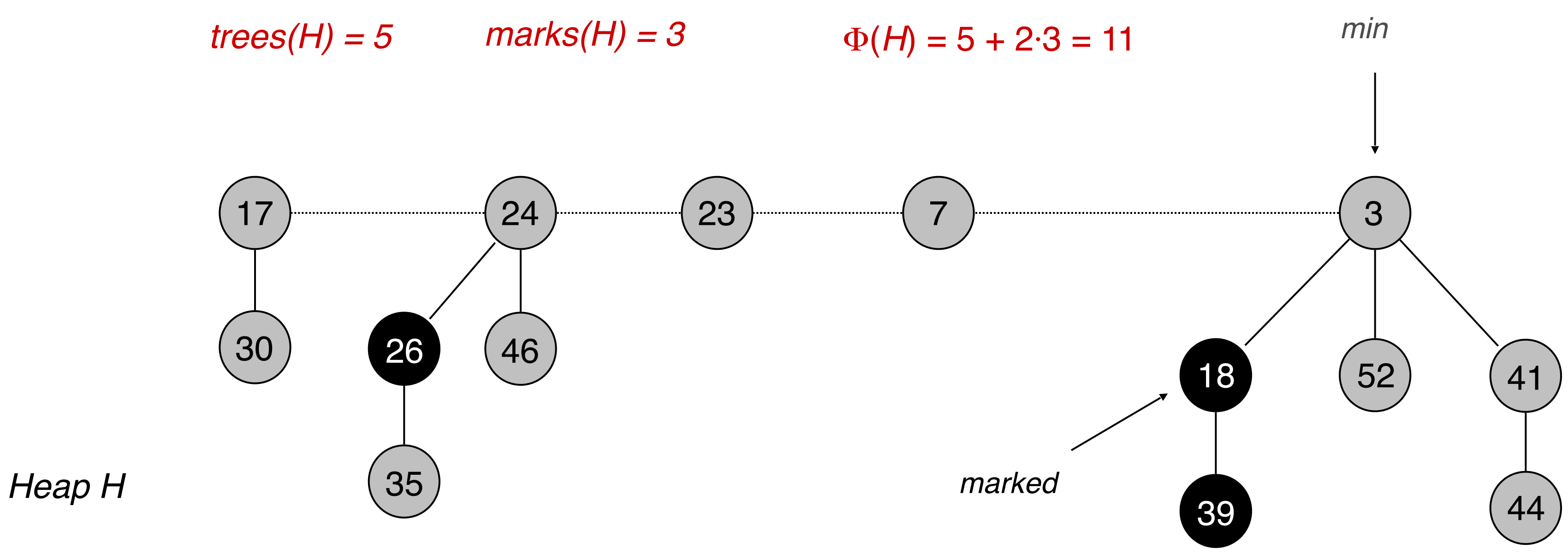
- *n* = number of nodes in heap.
- *rank(x)* = number of children of node *x*.
- *rank(H)* = max rank of any node in heap *H*.
- *trees(H)* = number of trees in heap *H*.
- *marks(H)* = number of marked nodes in heap *H*.



Fibonacci Heaps: Potential Function

$$\Phi(H) = trees(H) + 2 \cdot marks(H)$$

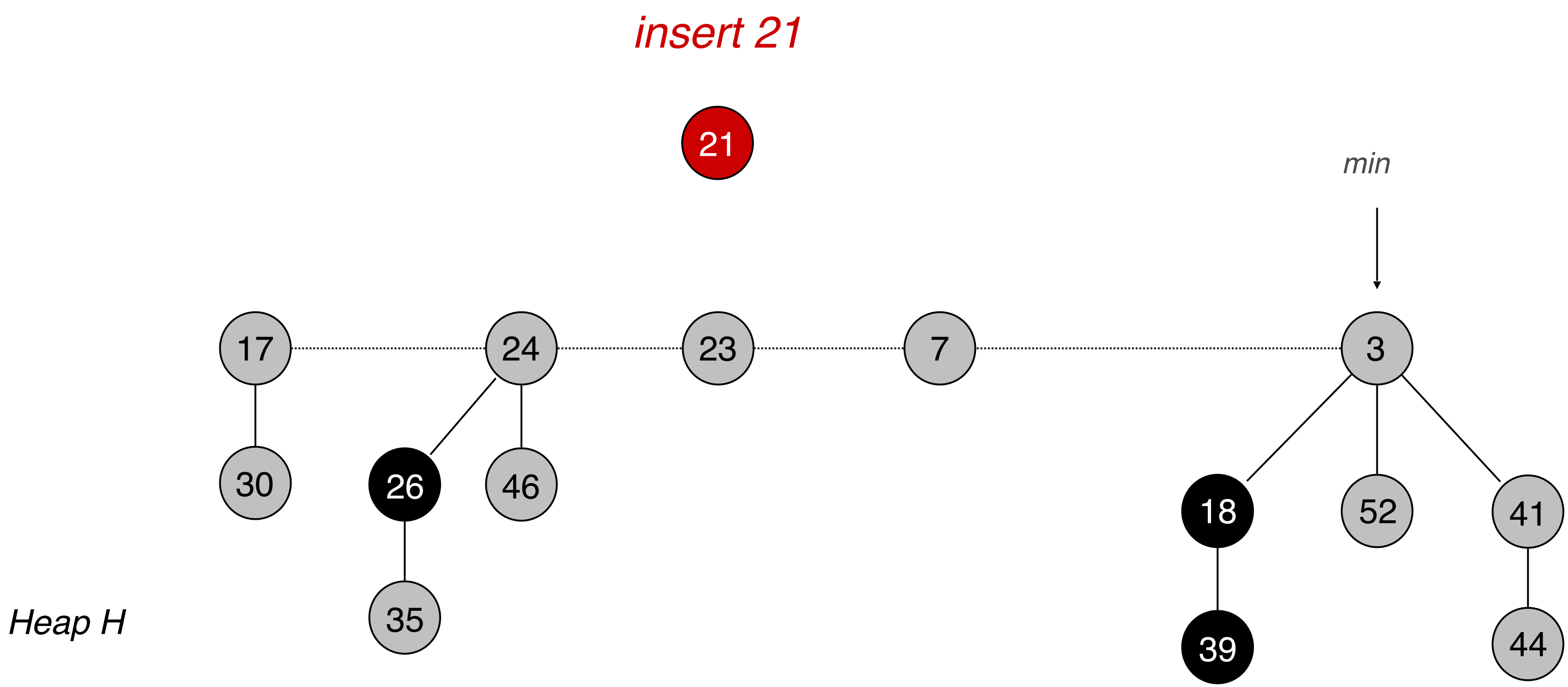
potential of heap H



Insert

Fibonacci Heaps: Insert

Insert.
- Create a new singleton tree.
- Add to root list; update min pointer (if necessary).

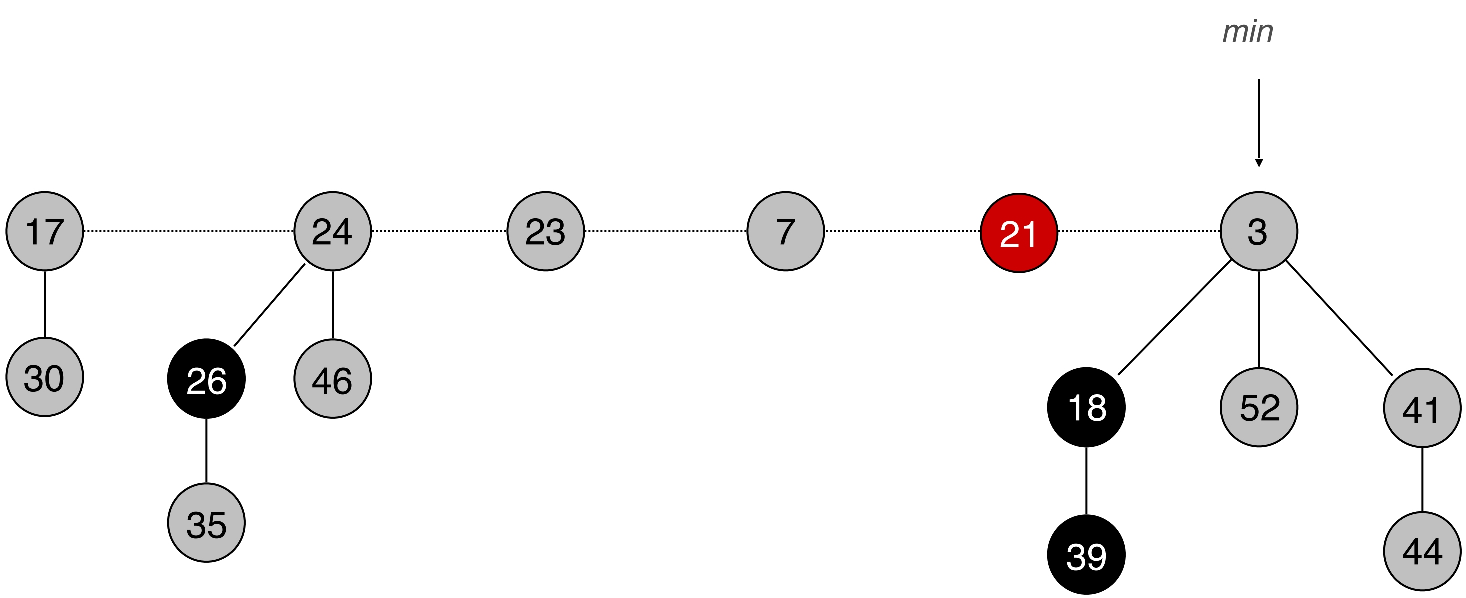


Fibonacci Heaps: Insert

Insert.

- Create a new singleton tree.
- Add to root list; update min pointer (if necessary).

insert 21



Heap H

Fibonacci Heaps: Insert Analysis

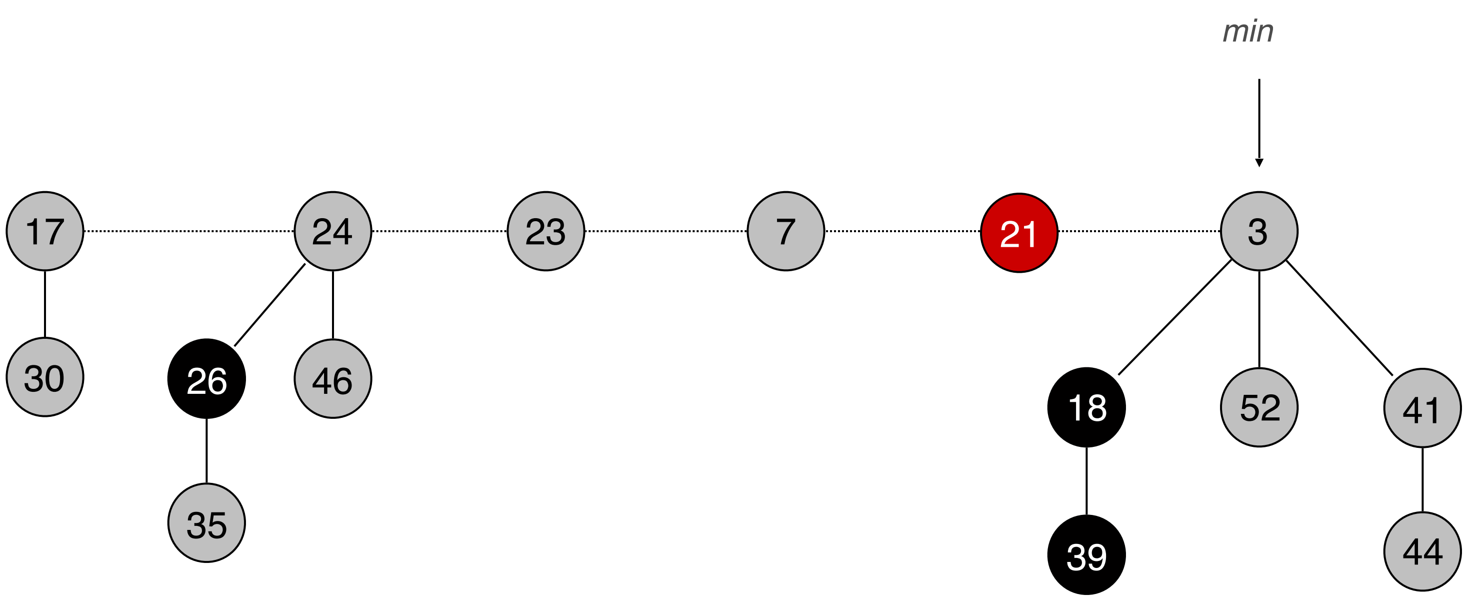
Actual cost. O(1)

Change in potential. +1

Amortized cost. O(1)

$$\Phi(H) = trees(H) + 2 \cdot marks(H)$$

potential of heap H



Heap H

Code

```
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
```

Heap Union

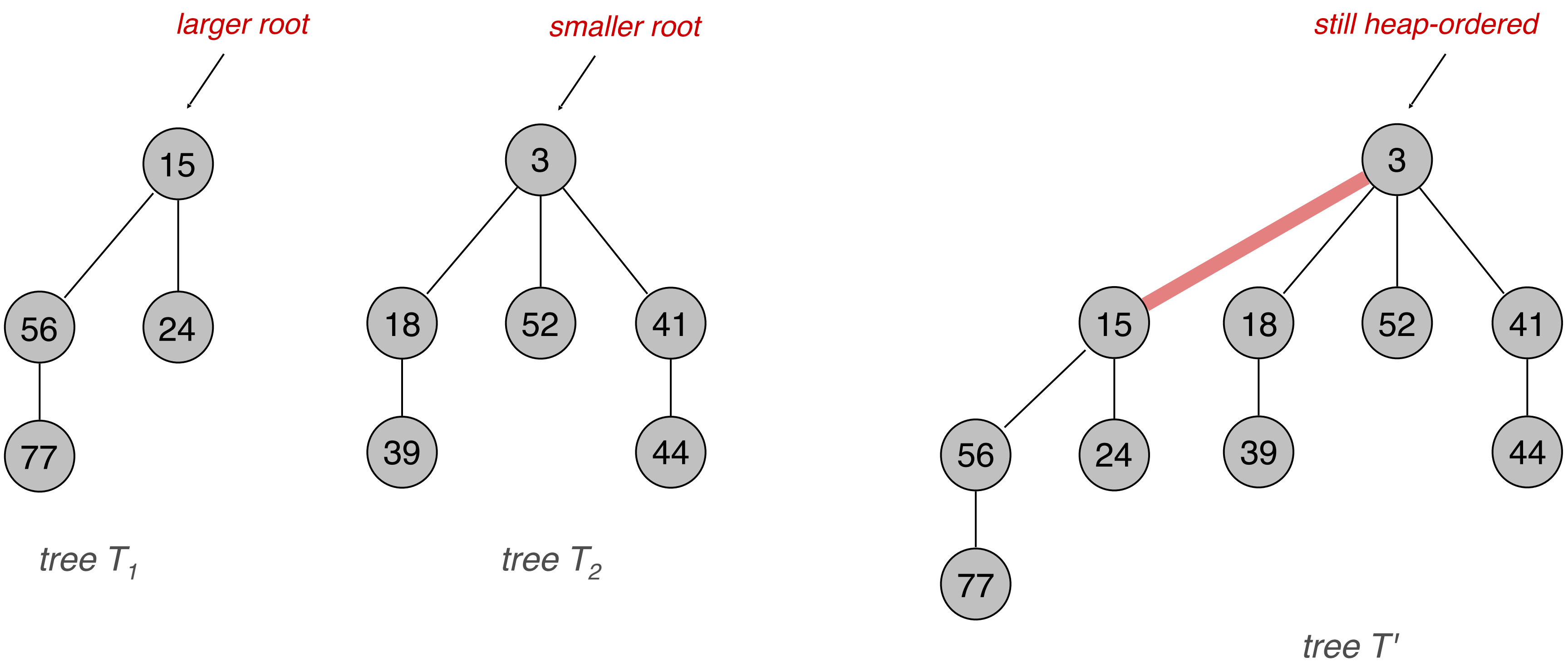
FIB-HEAP-UNION(H_1, H_2)

H = MAKE-FIB-HEAP()
$H.min = H_1.min$
concatenate the root list of H_2 with the root list of H
if $(H_1.min == \text{NIL})$ or $(H_2.min \neq \text{NIL}$ and $H_2.min.key < H_1.min.key)$
    $H.min = H_2.min$
$H.n = H_1.n + H_2.n$
return H

Delete Min

Linking Operation

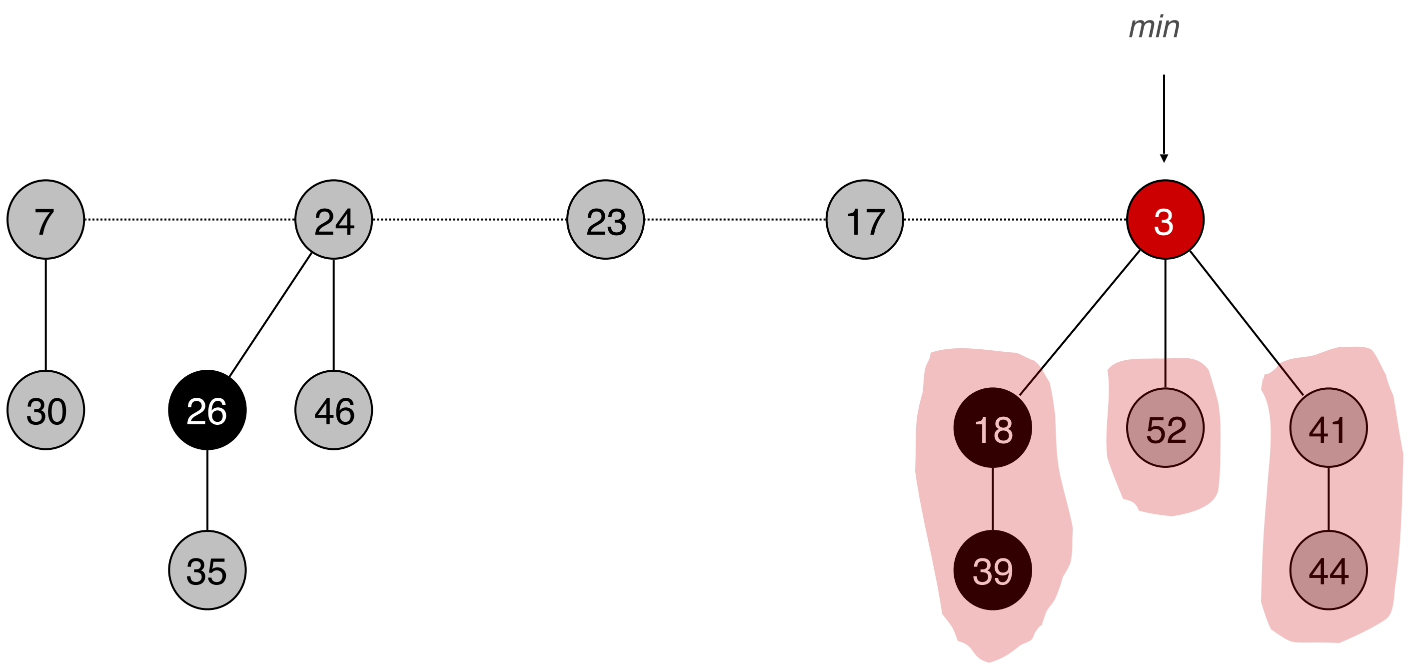
Linking operation. Make larger root be a child of smaller root.



Fibonacci Heaps: Delete Min

Delete min.

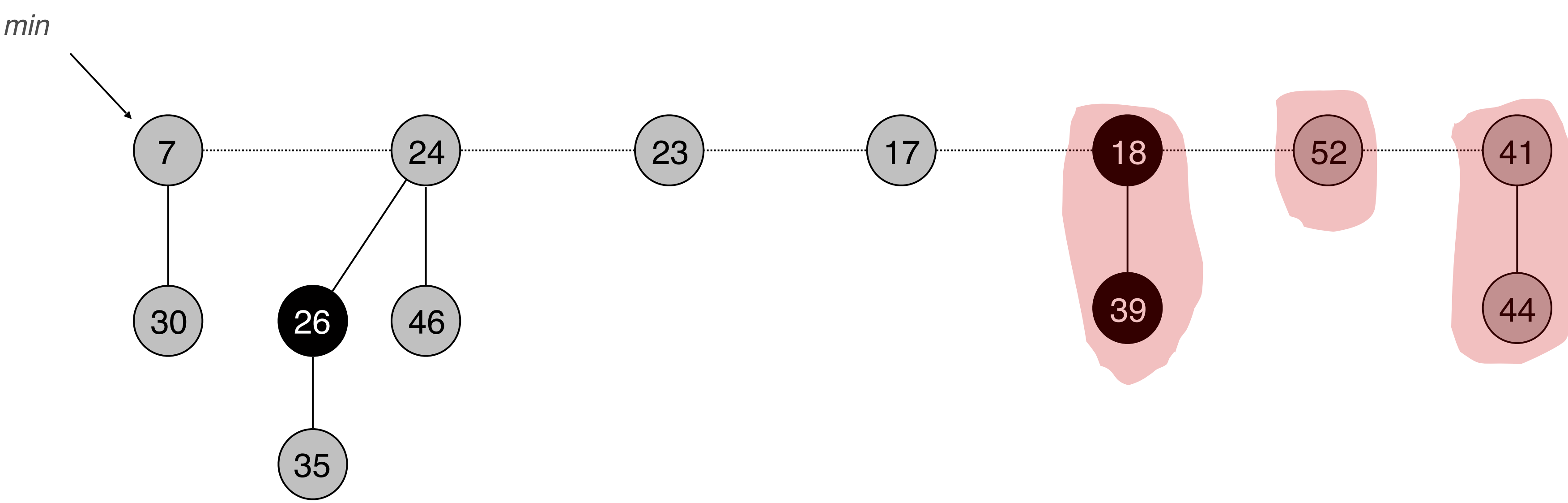
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



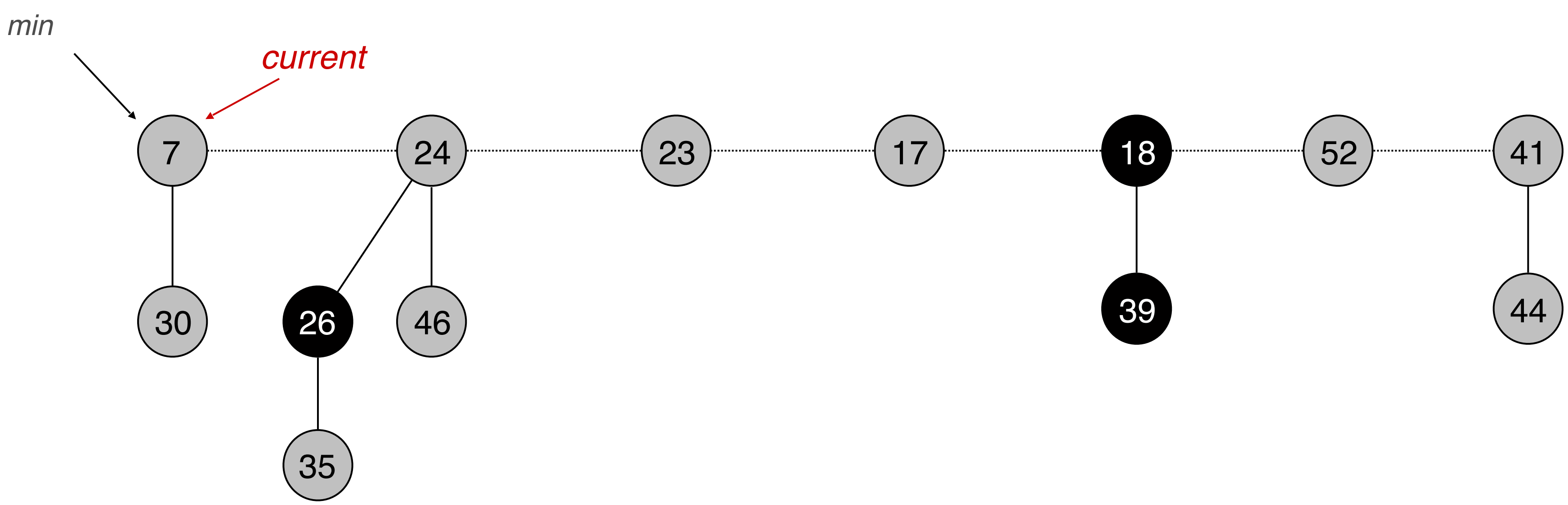
Delete Min Code

```
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: Delete Min

Delete min.

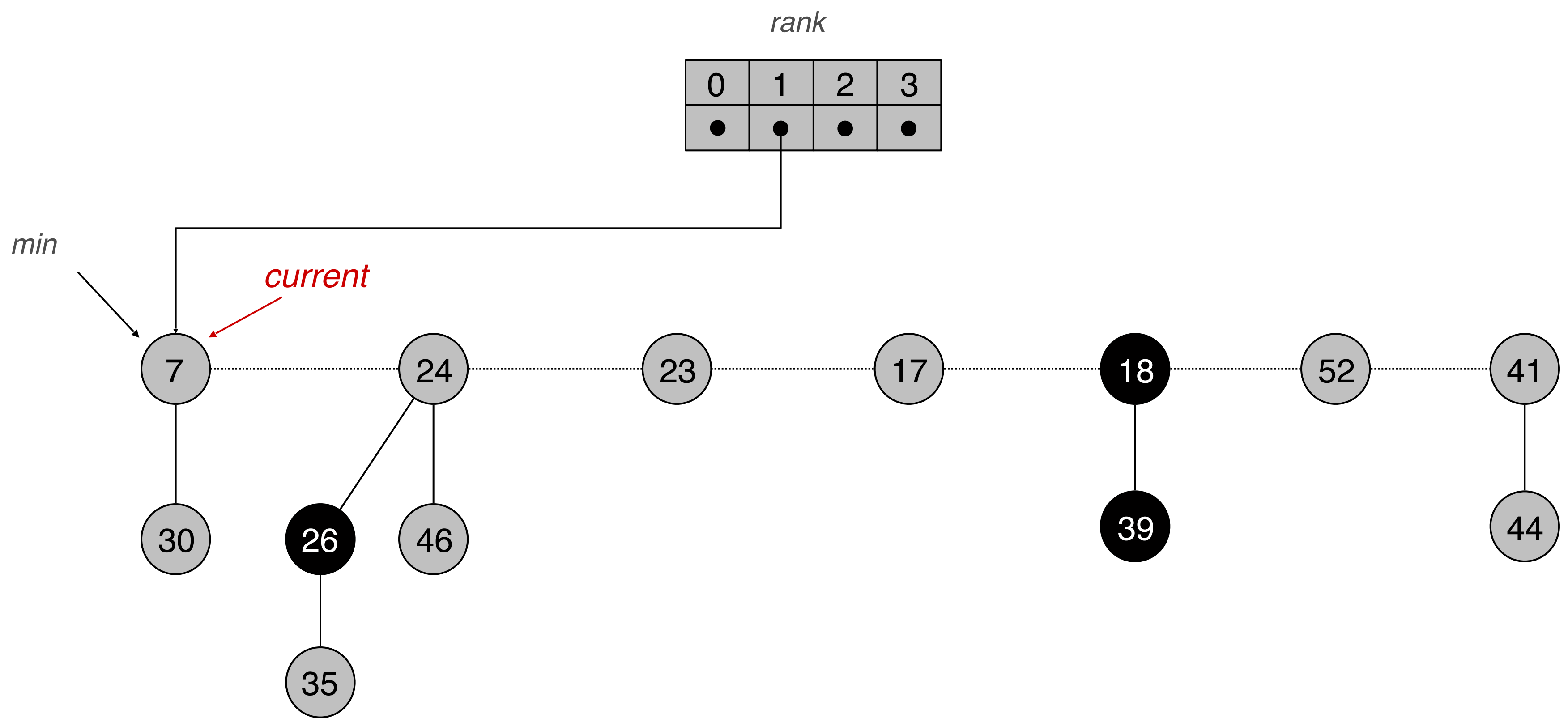
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

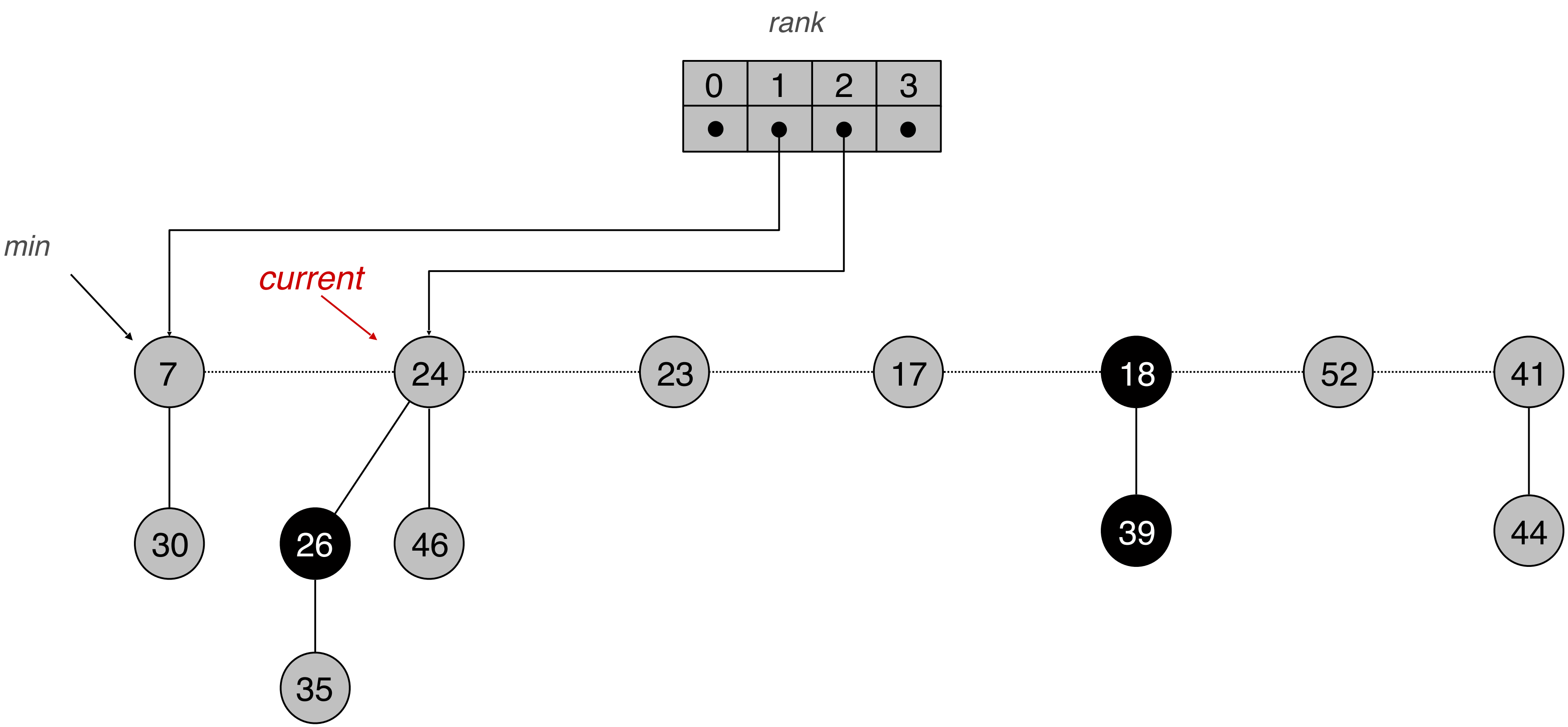
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

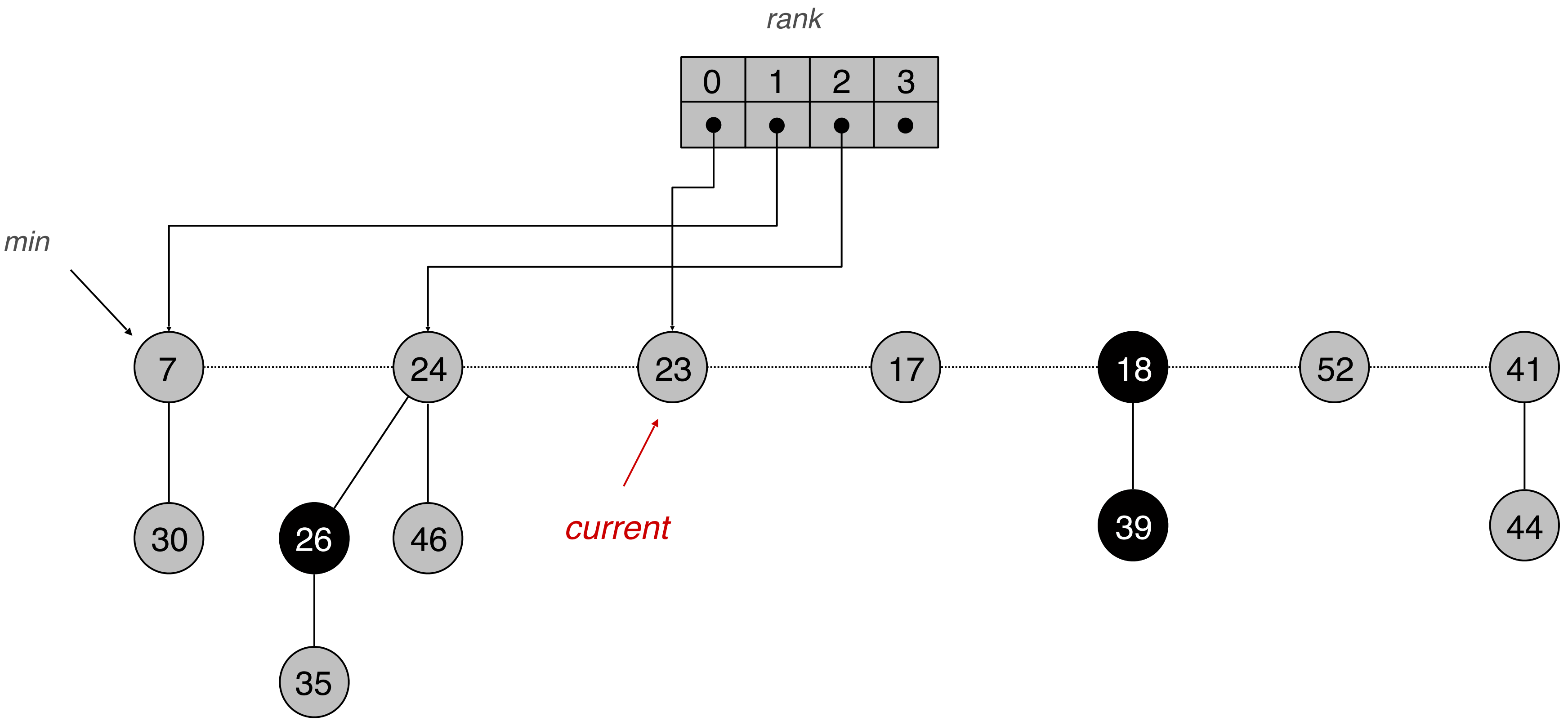
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

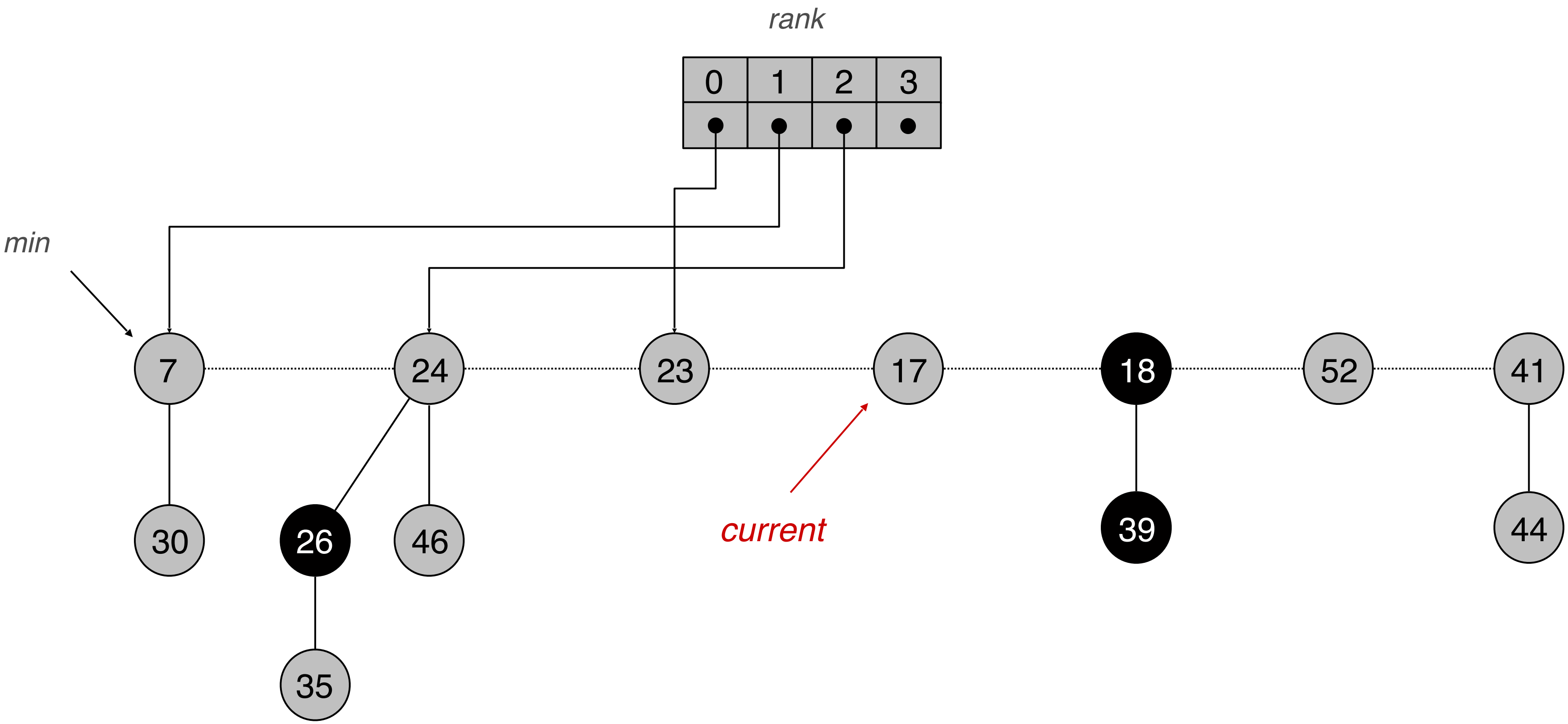
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.

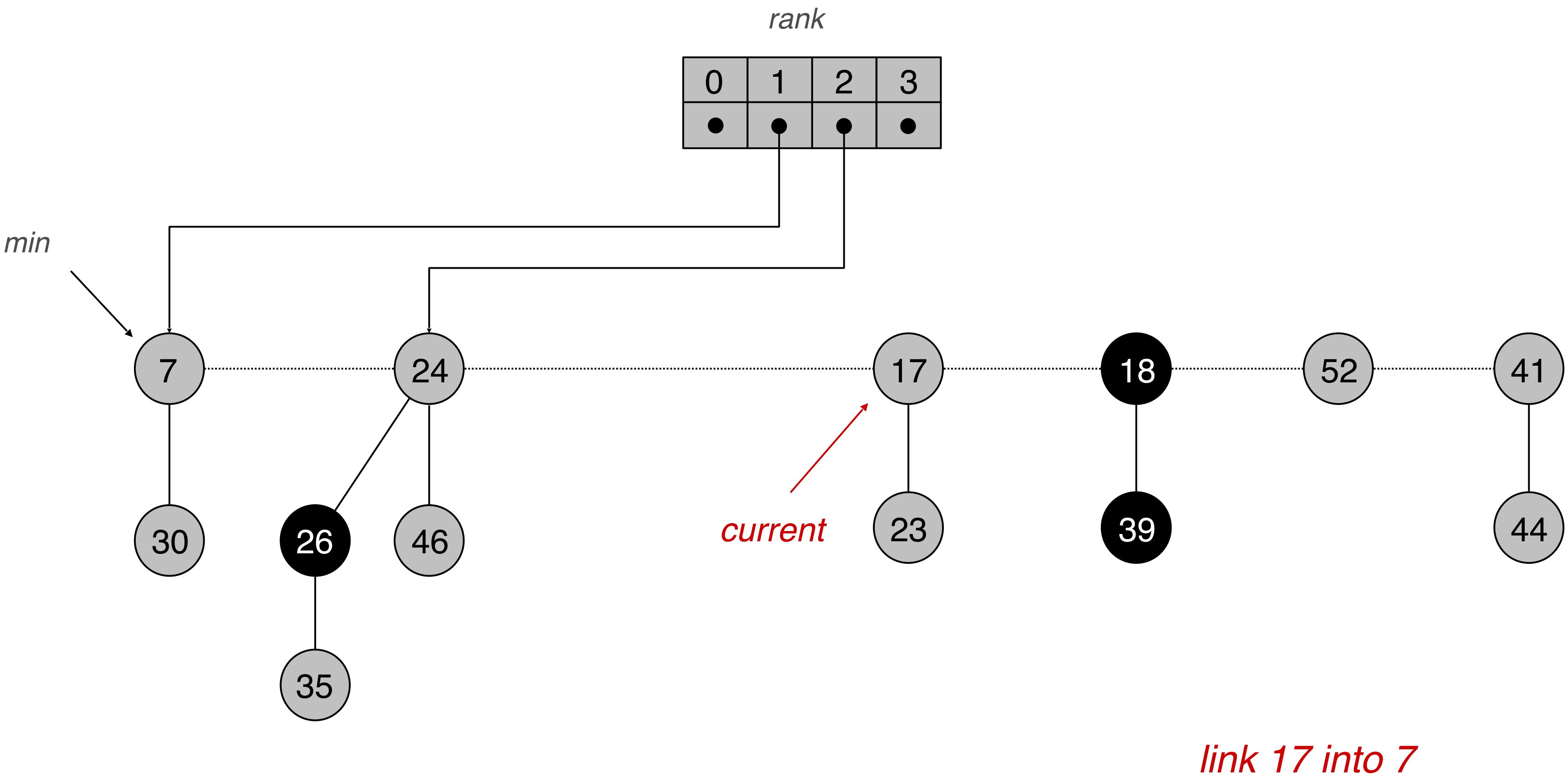


link 23 into 17

Fibonacci Heaps: Delete Min

Delete min.

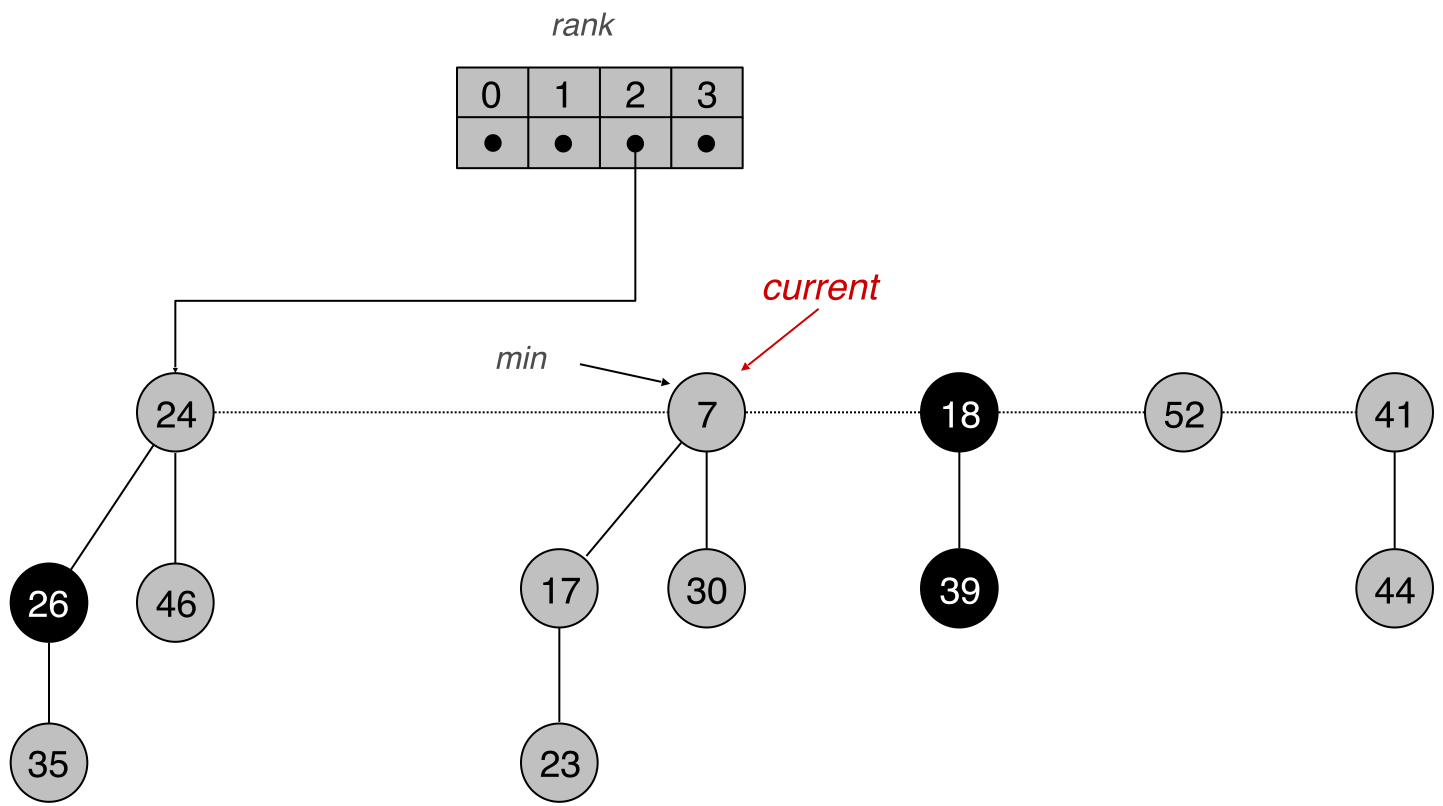
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.

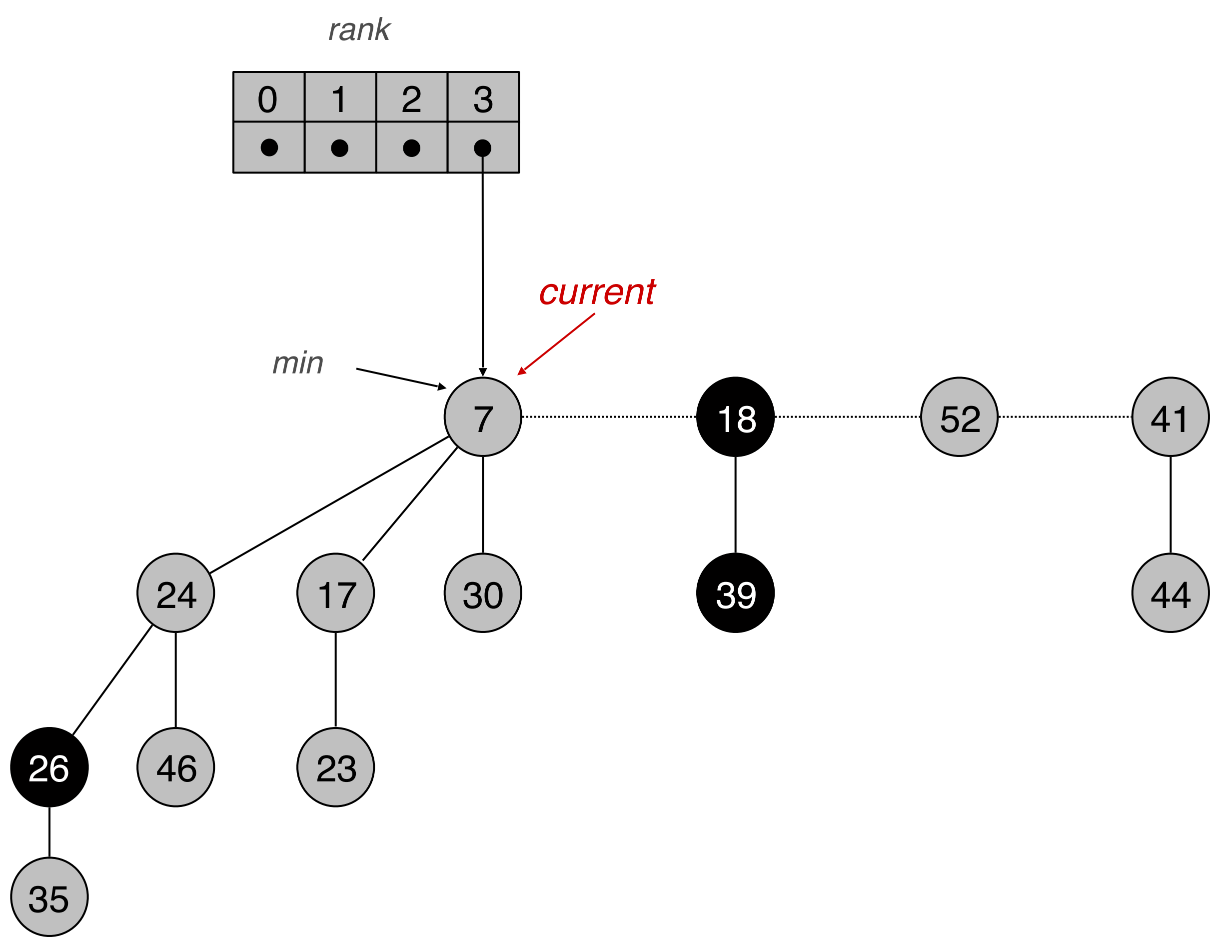


link 24 into 7

Fibonacci Heaps: Delete Min

Delete min.

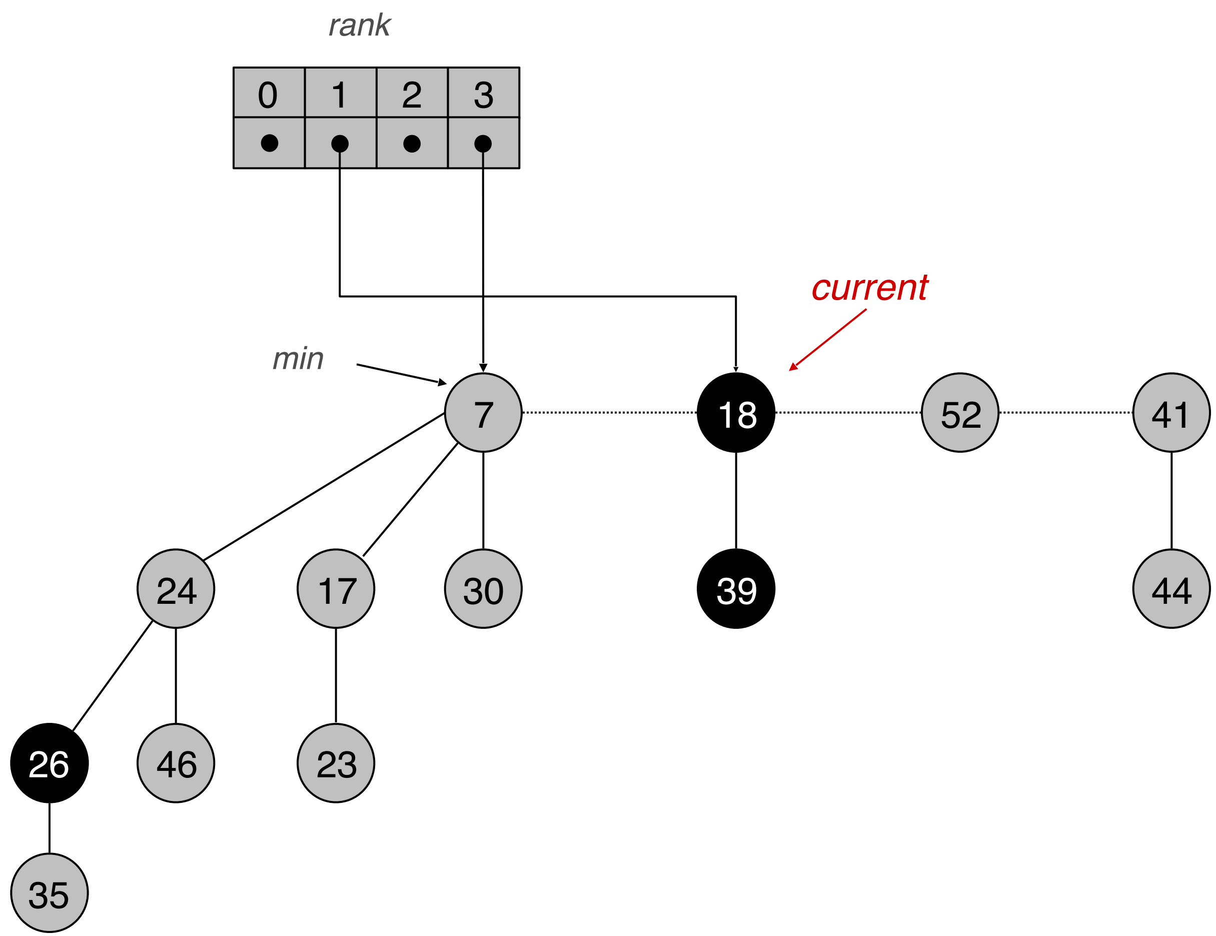
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

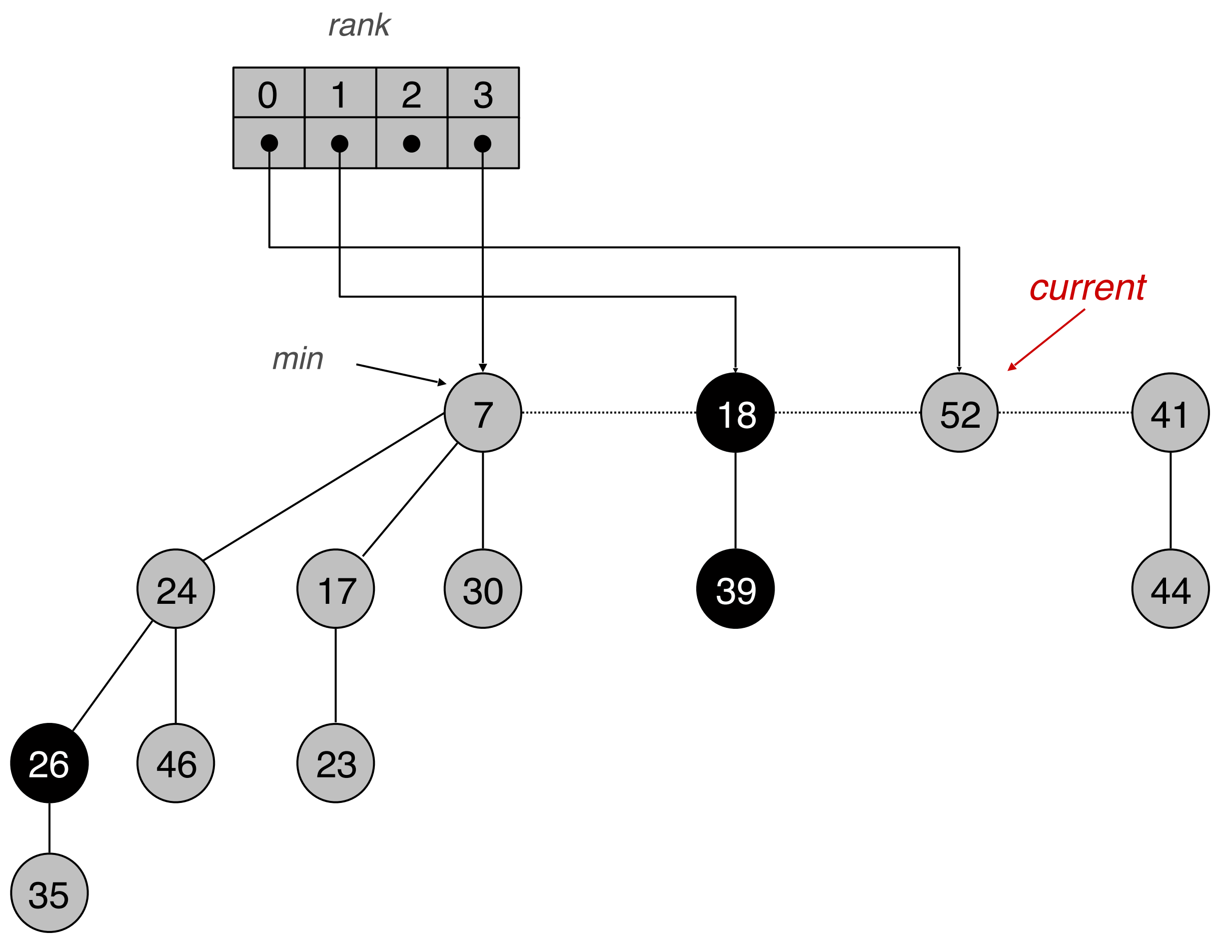
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

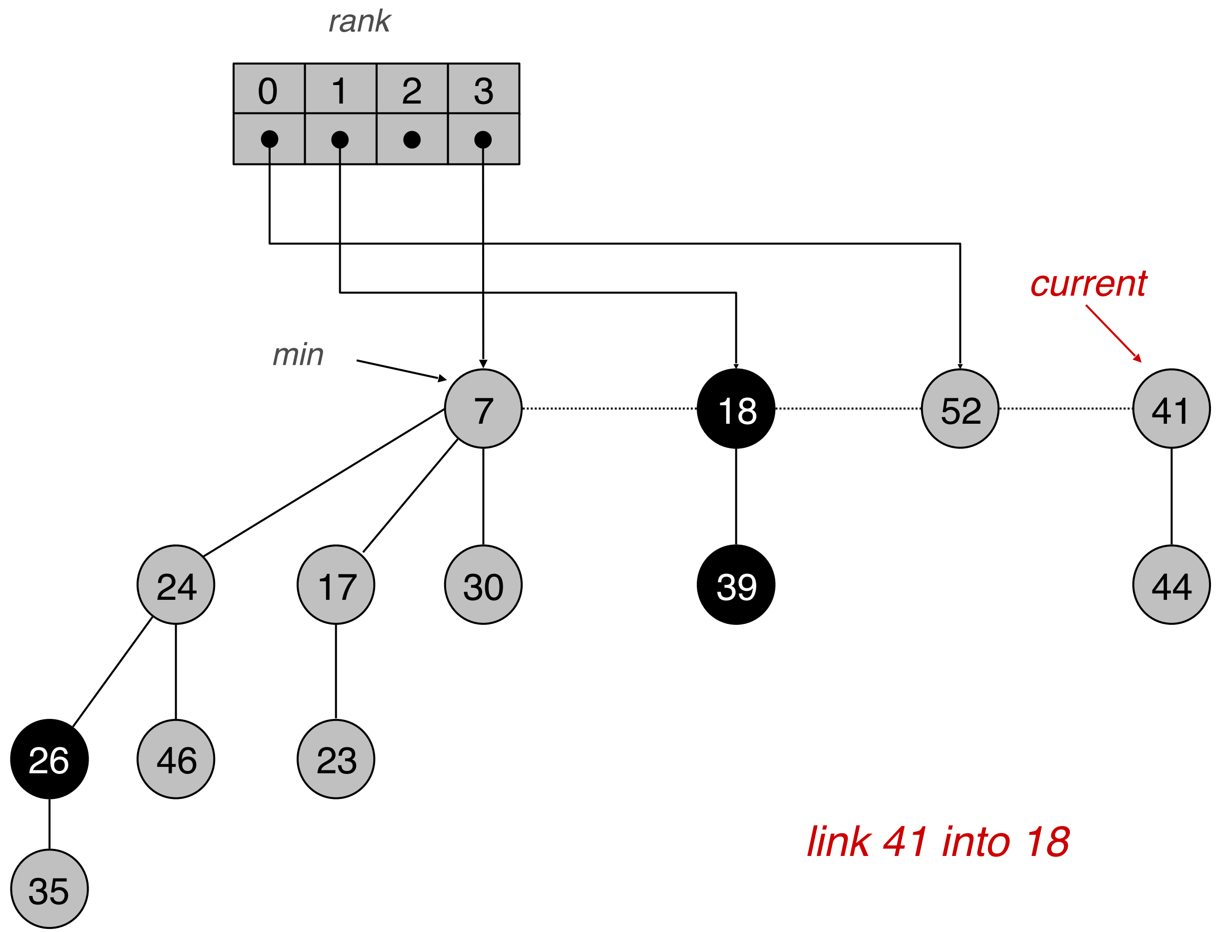
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

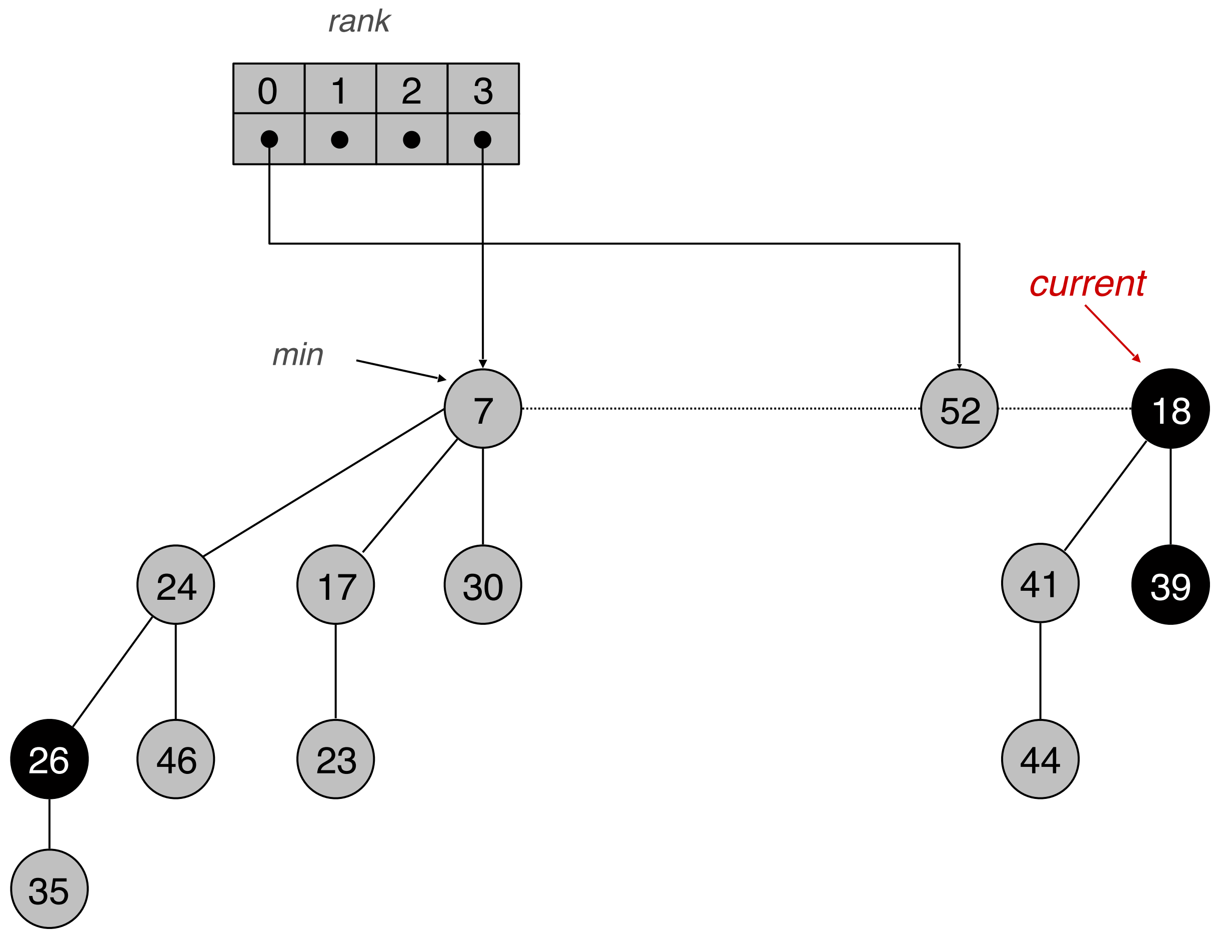
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

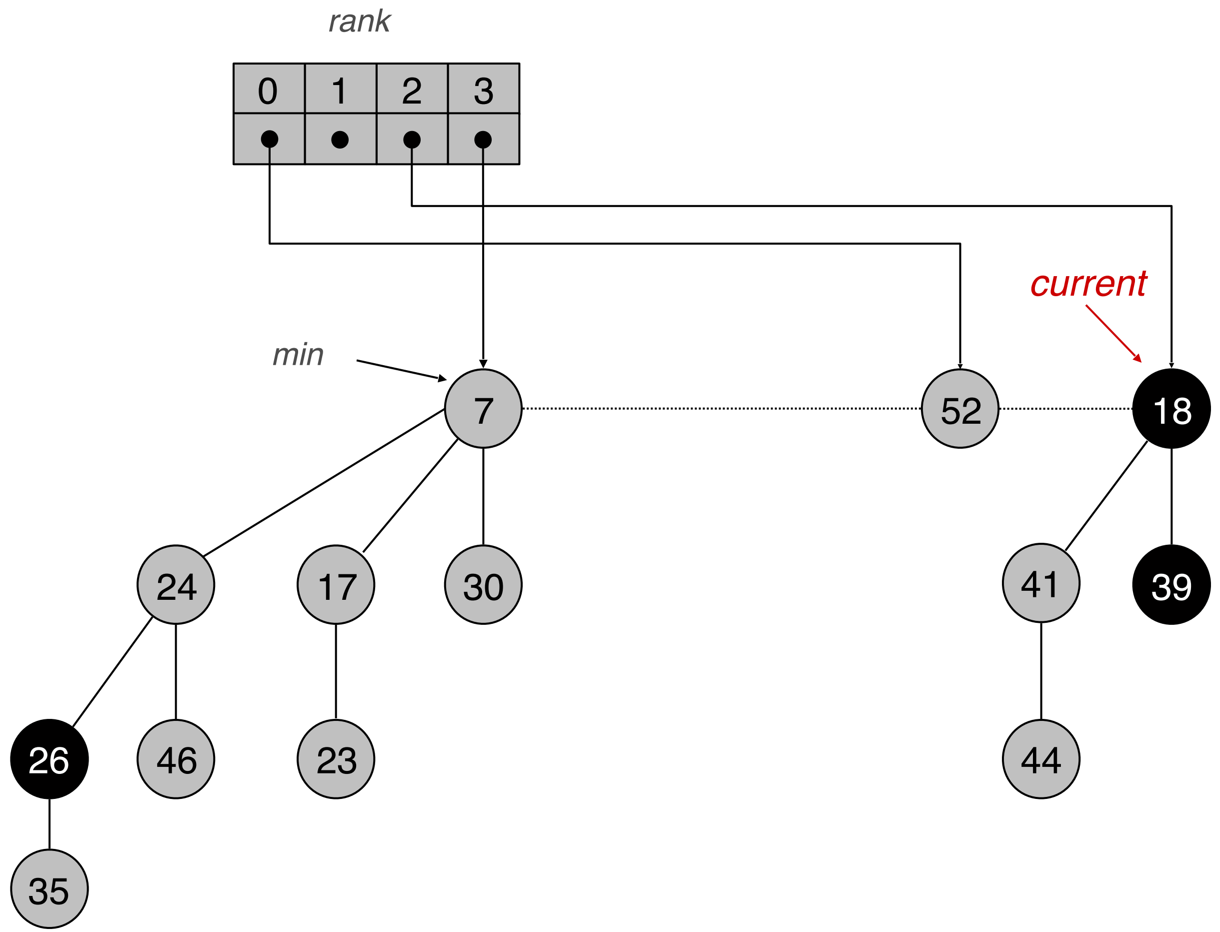
Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Delete min.

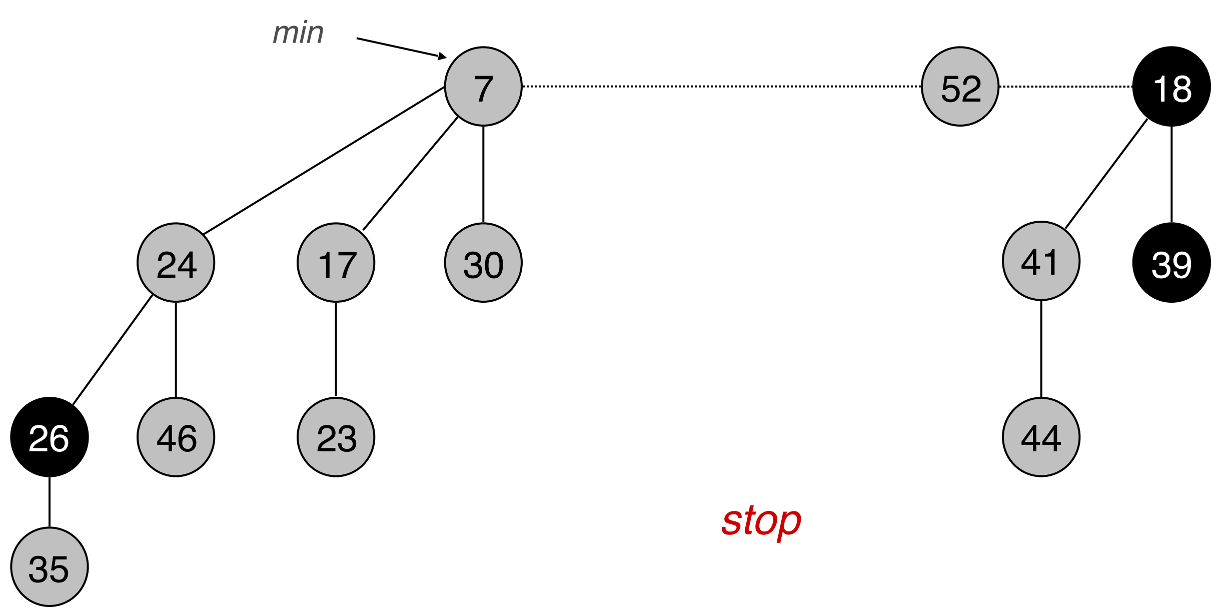
- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



Fibonacci Heaps: Delete Min

Delete min.

- Delete min; meld its children into root list; update min.
- Consolidate trees so that no two roots have same rank.



```
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)
remove y from the root list of H
make y a child of x, incrementing x.degree
y.mark = FALSE
```

Fibonacci Heaps: Delete Min Analysis

Delete min.

$$\Phi(H) = \mathit{trees}(H) + 2 \cdot \mathit{marks}(H)$$

potential function

Actual cost. $O(\mathit{rank}(H)) + O(\mathit{trees}(H))$

- $O(\mathit{rank}(H))$ to meld min's children into root list.
- $O(\mathit{rank}(H)) + O(\mathit{trees}(H))$ to update min.
- $O(\mathit{rank}(H)) + O(\mathit{trees}(H))$ to consolidate trees.

Change in potential. $O(\mathit{rank}(H)) - \mathit{trees}(H)$

- $\mathit{trees}(H') \le \mathit{rank}(H) + 1$ since no two trees have same rank.
- $\Delta\Phi(H) \le \mathit{rank}(H) + 1 - \mathit{trees}(H)$.

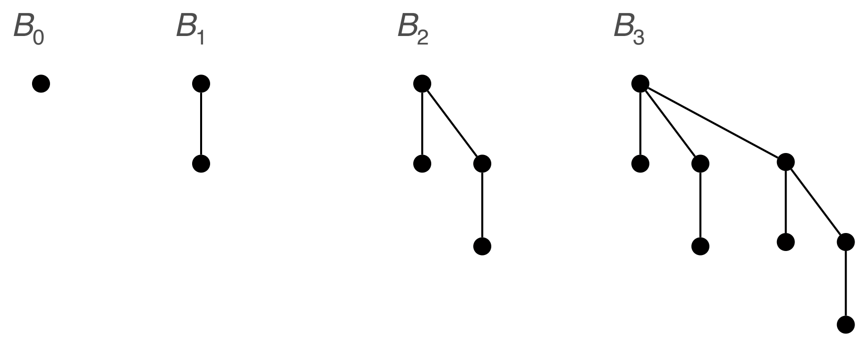
Amortized cost. $O(\mathit{rank}(H))$

Fibonacci Heaps: Delete Min Analysis

Q. Is amortized cost of O(*rank*(*H*)) good?

A. Yes, if only *insert* and *delete-min* operations.
- In this case, all trees are binomial trees.
- This implies *rank*(*H*) $\leq$ lg *n*.

we only link trees of equal rank



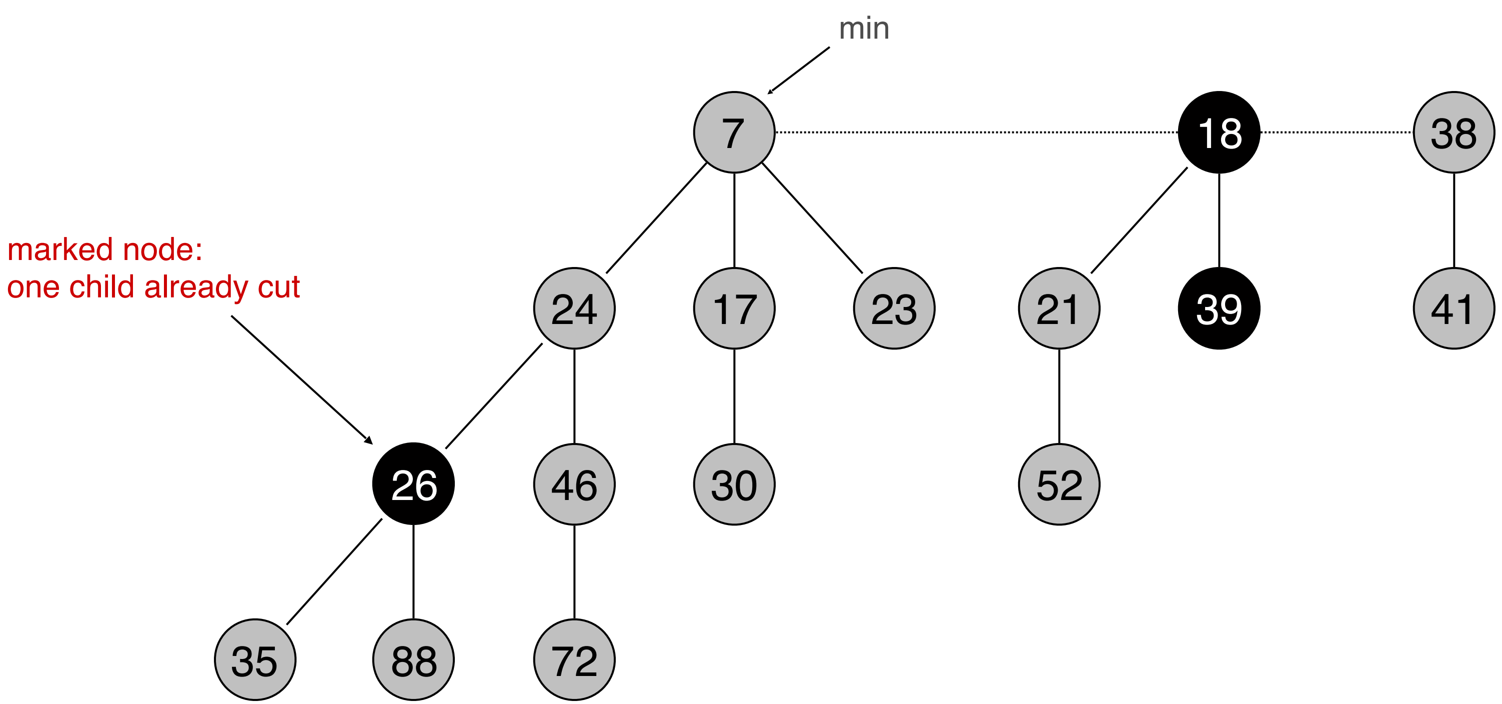
A. Yes, we'll implement *decrease-key* so that *rank*(*H*) = O(log *n*).

Decrease Key

Fibonacci Heaps: Decrease Key

Intuition for deceasing the key of node *x*.

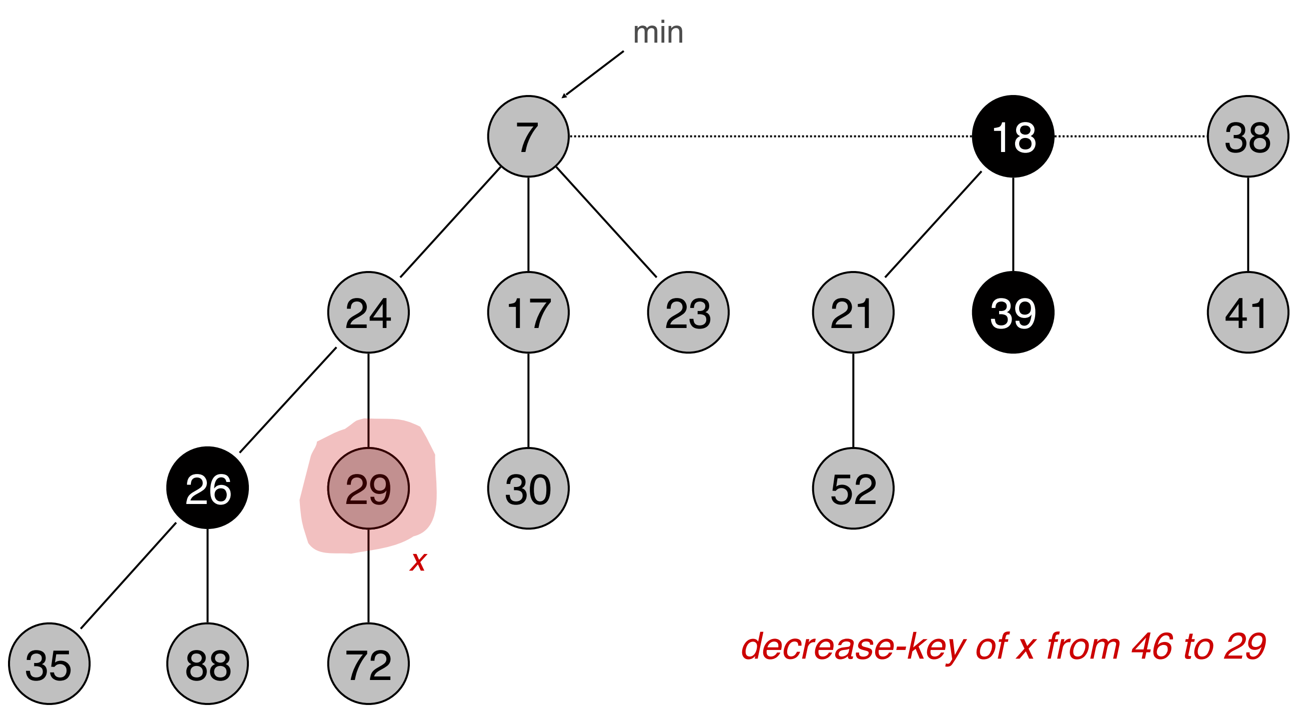
- If heap-order is not violated, just decrease the key of *x*.
- Otherwise, cut tree rooted at *x* and meld into root list.
- To keep trees flat: as soon as a node has its second child cut, cut it off and meld into root list (and unmark it).



Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

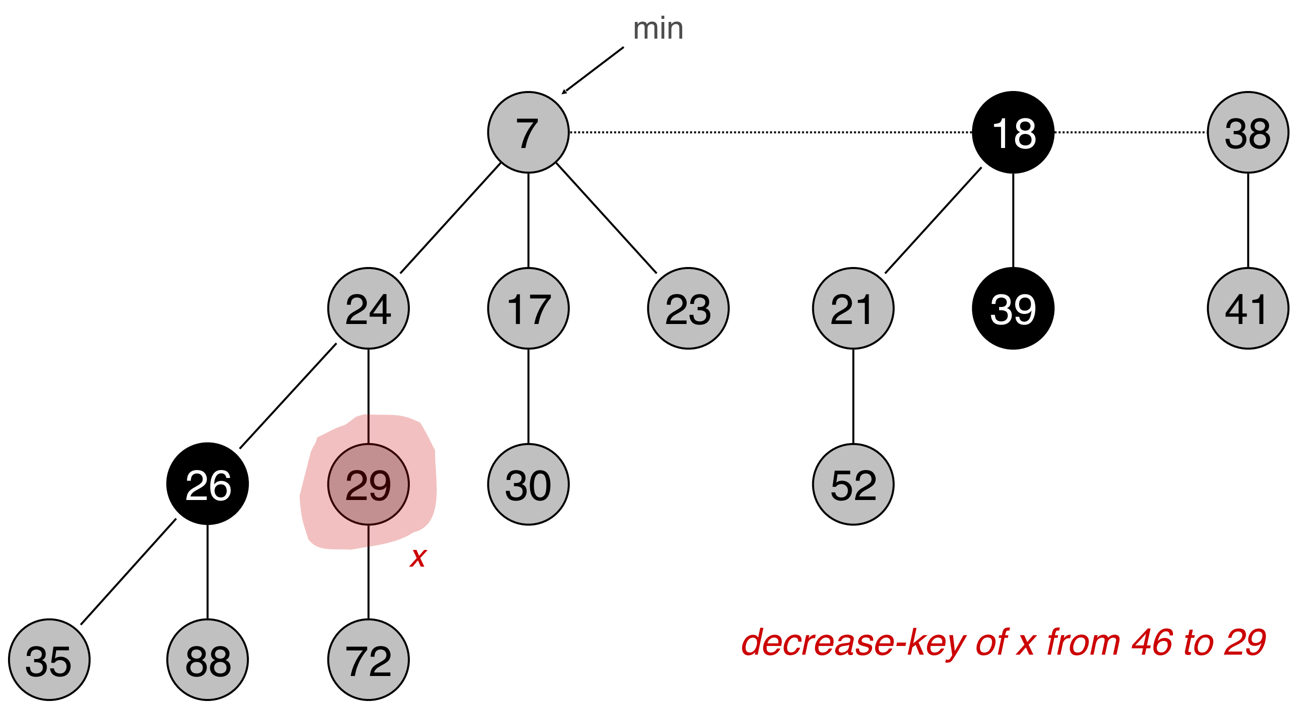
- Decrease key of *x*.
- Change heap min pointer (if necessary).



Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

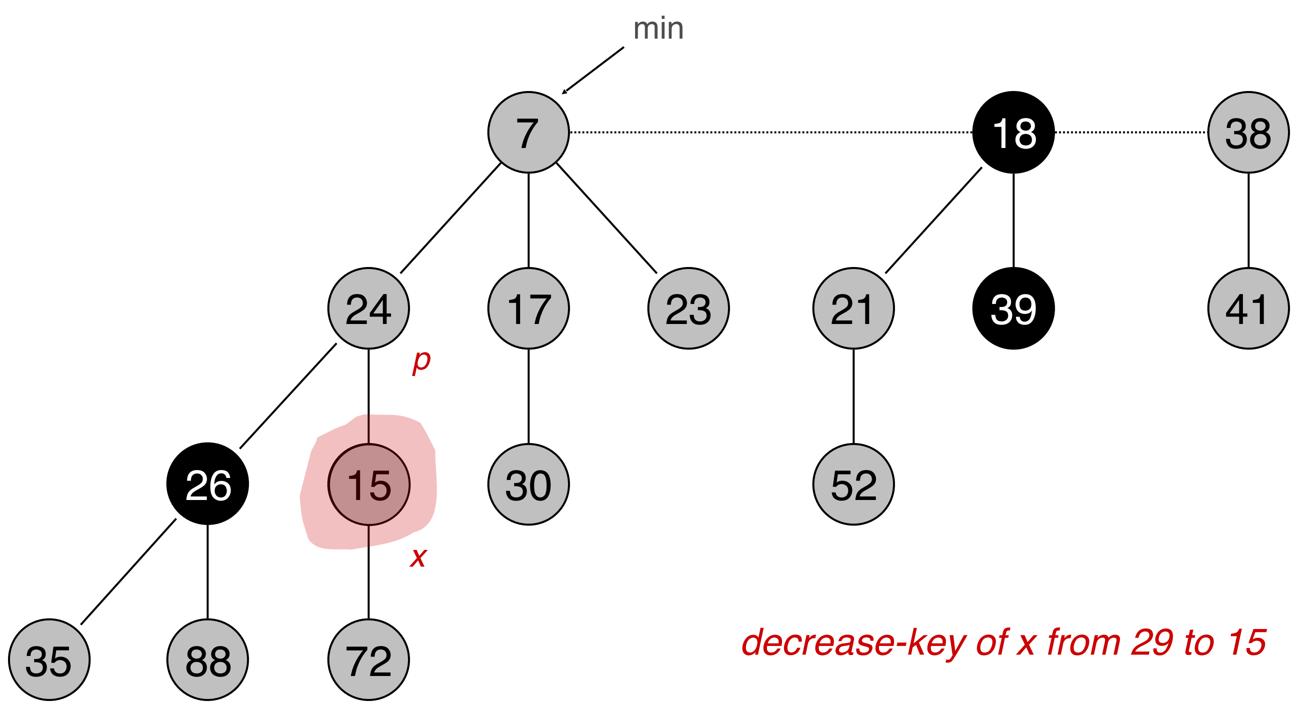
- Decrease key of *x*.
- Change heap min pointer (if necessary).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

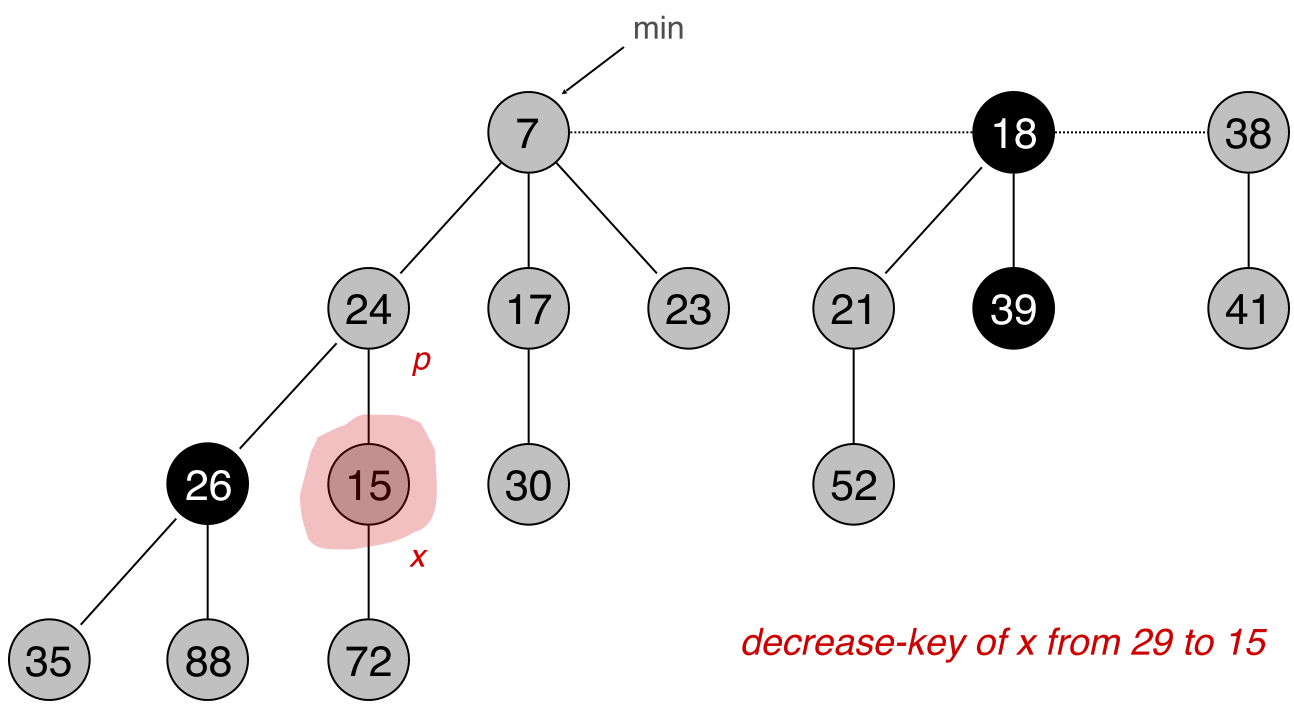
- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

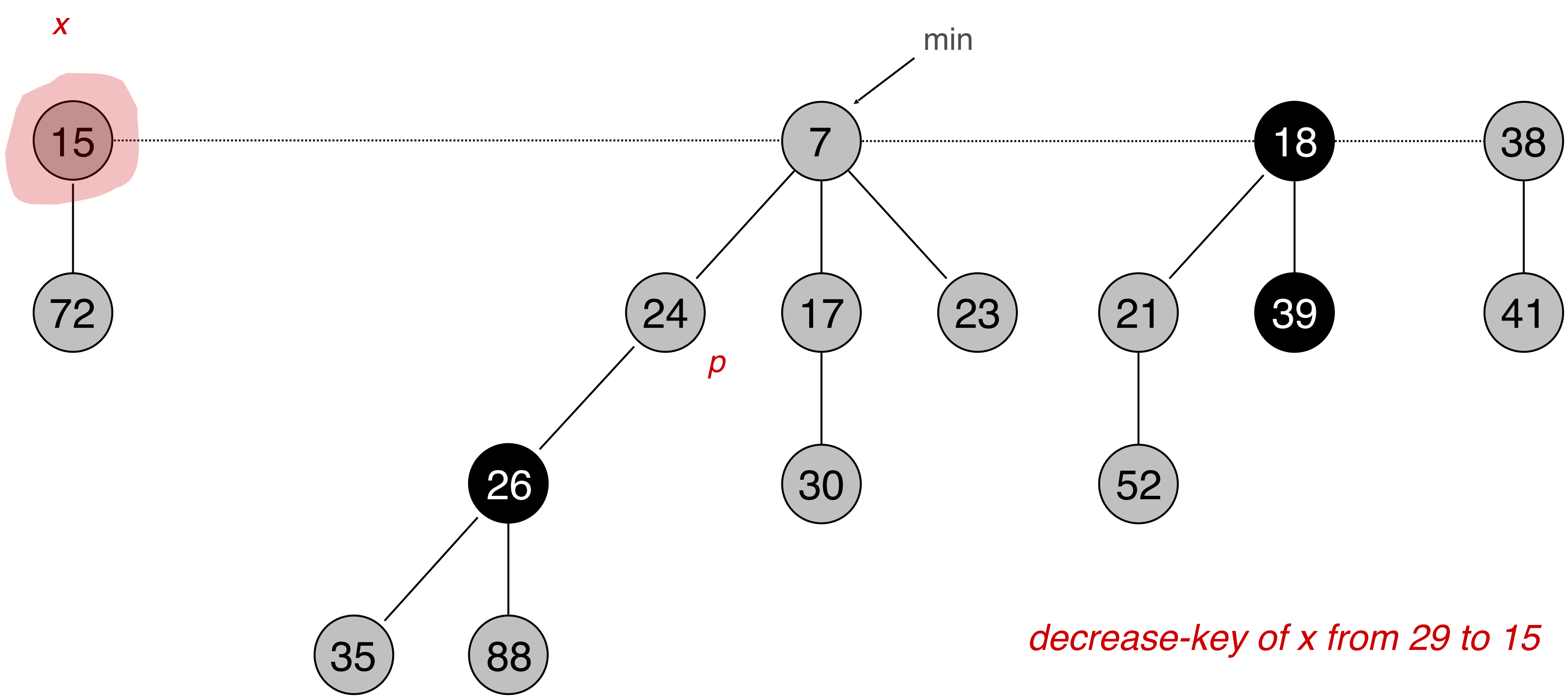
- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

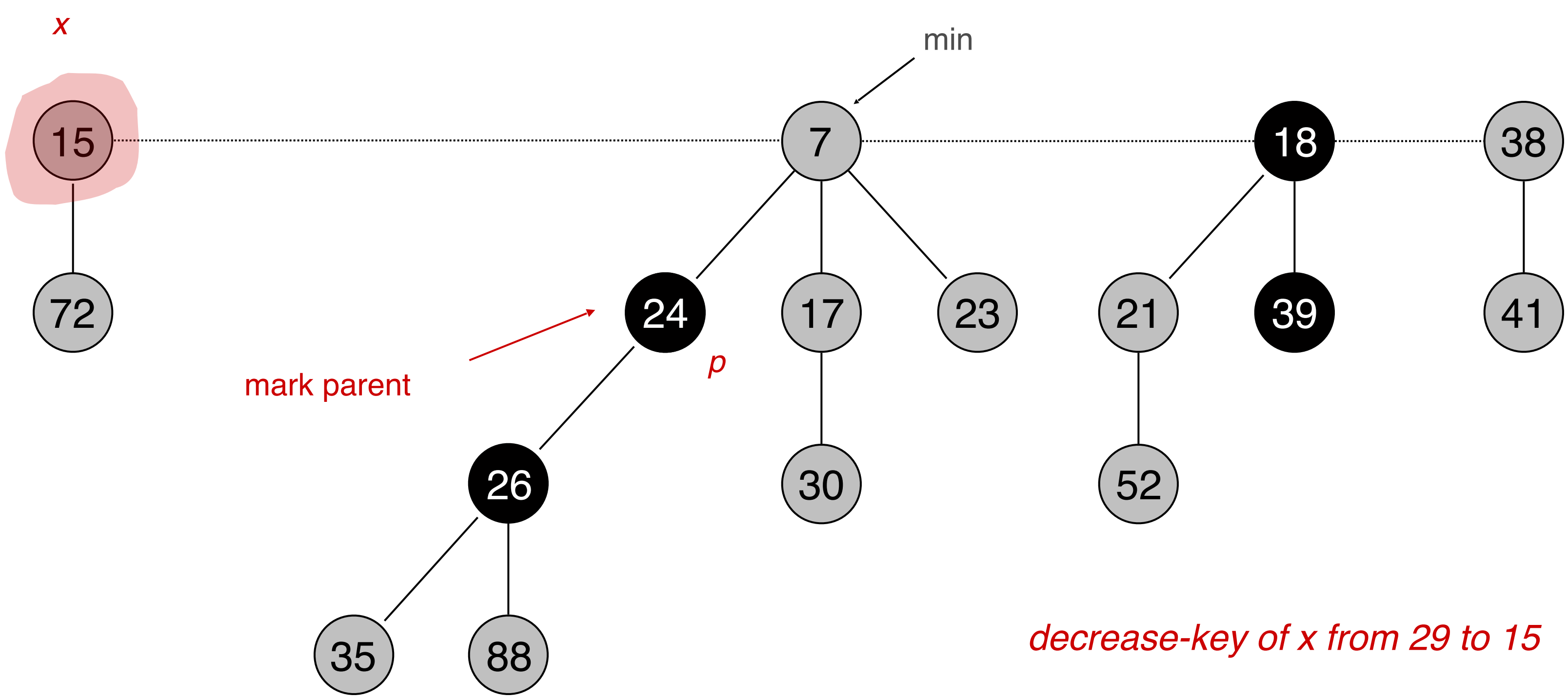


decrease-key of x from 29 to 15

Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

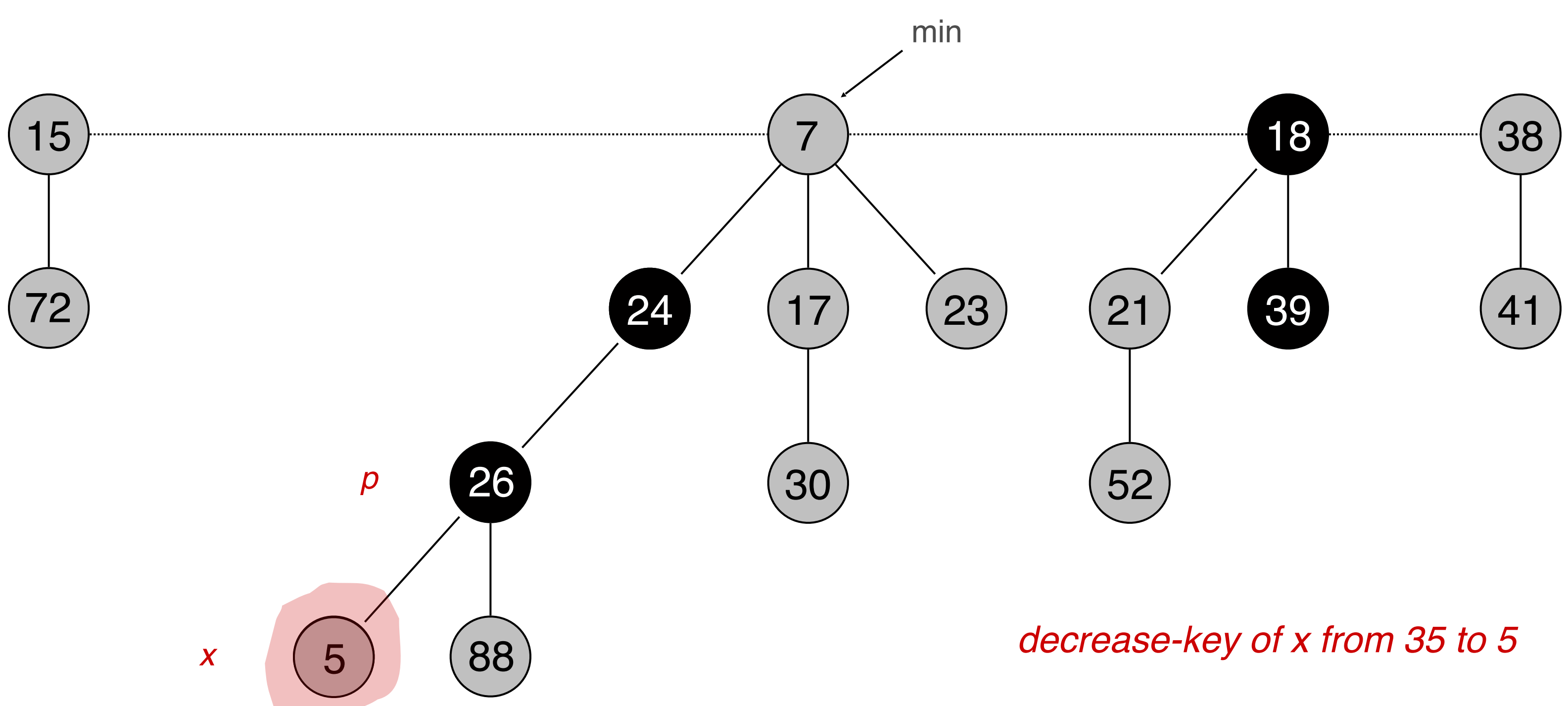
- Decrease key of *x*.
- Cut tree rooted at *x,* meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p,* meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

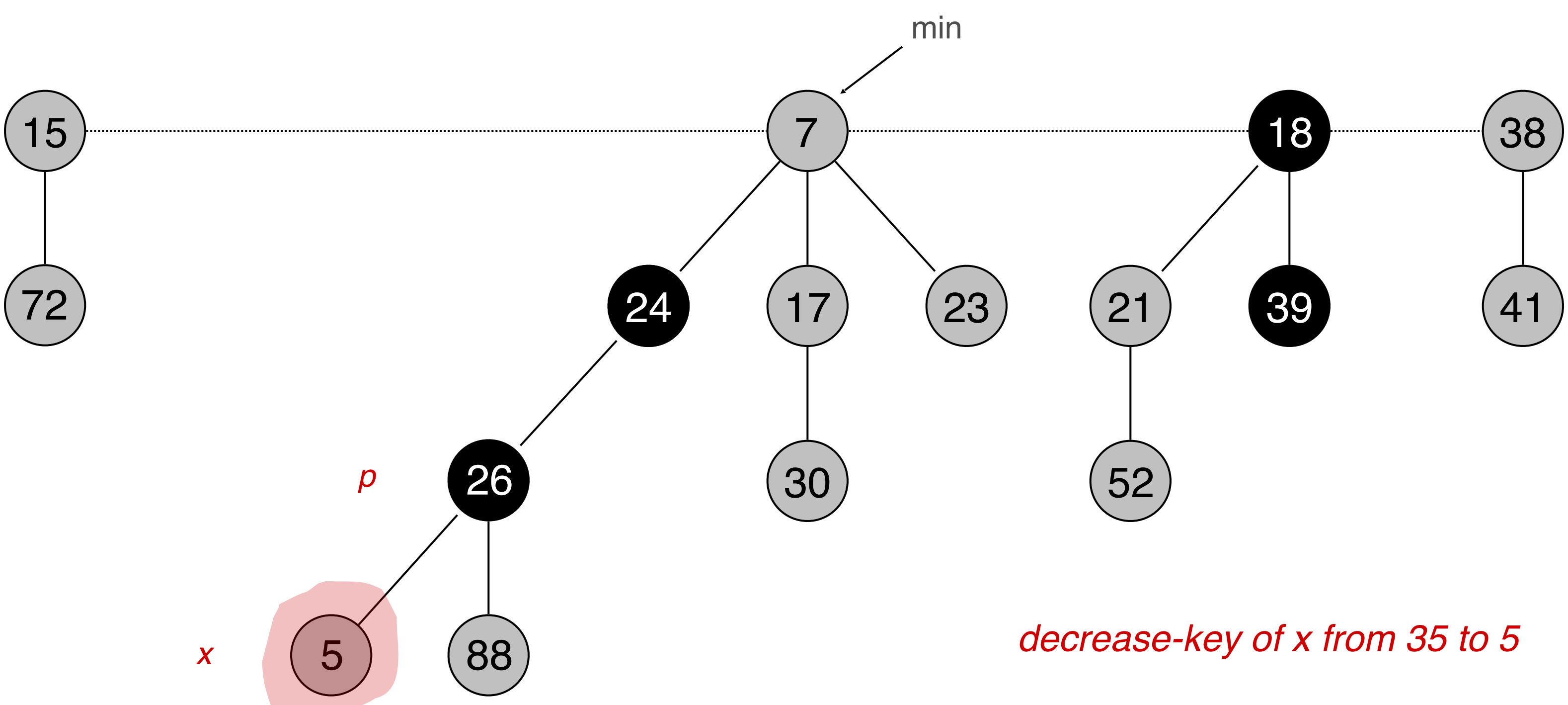
- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

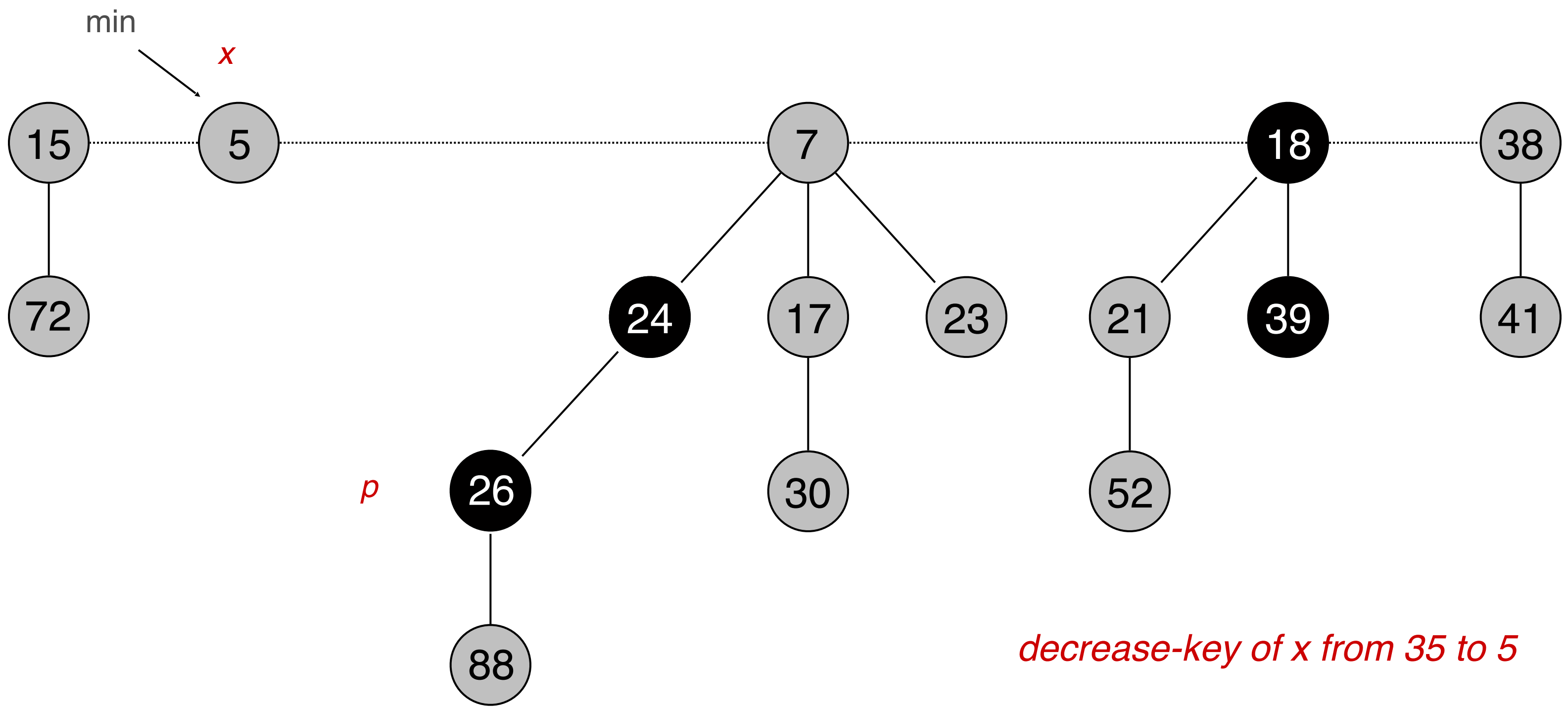
- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

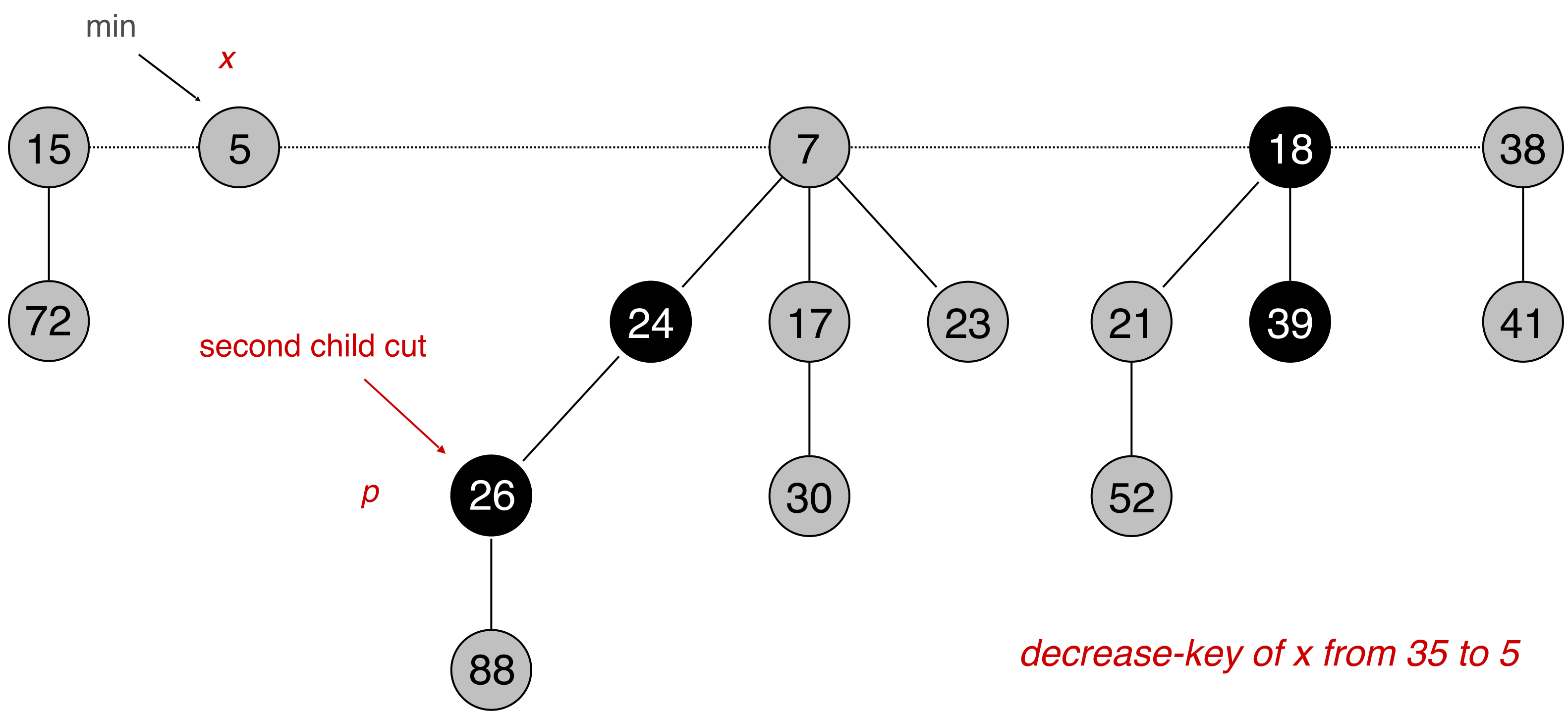
- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

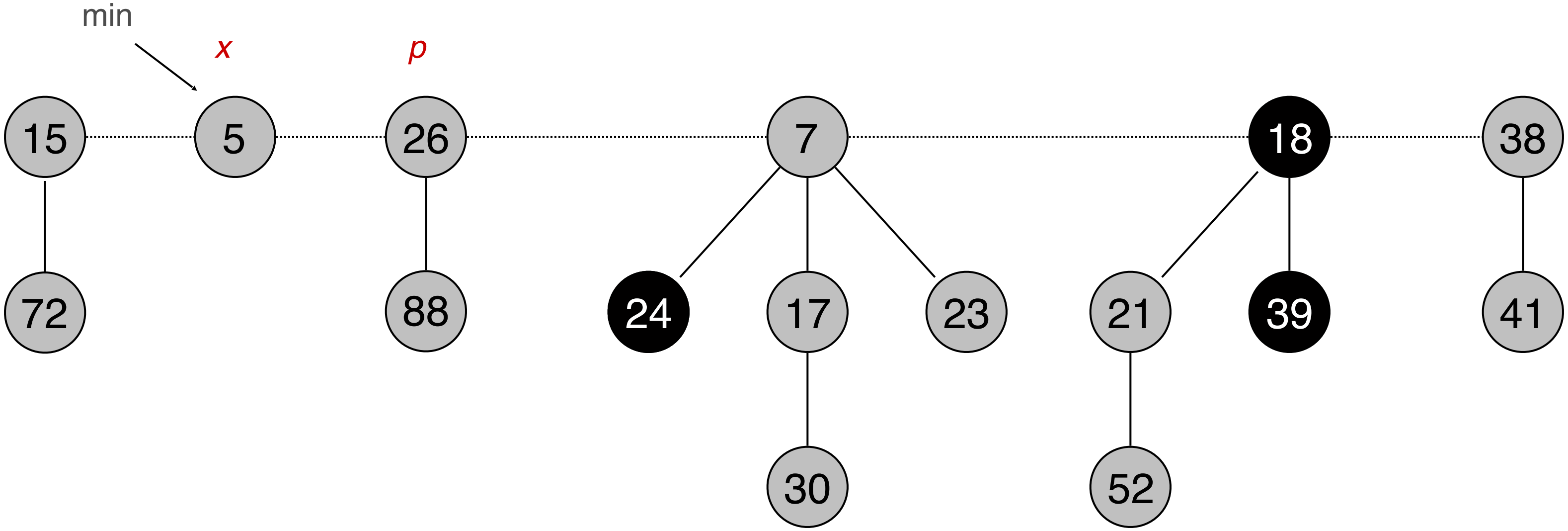
- Decrease key of *x*.
- Cut tree rooted at *x,* meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p,* meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

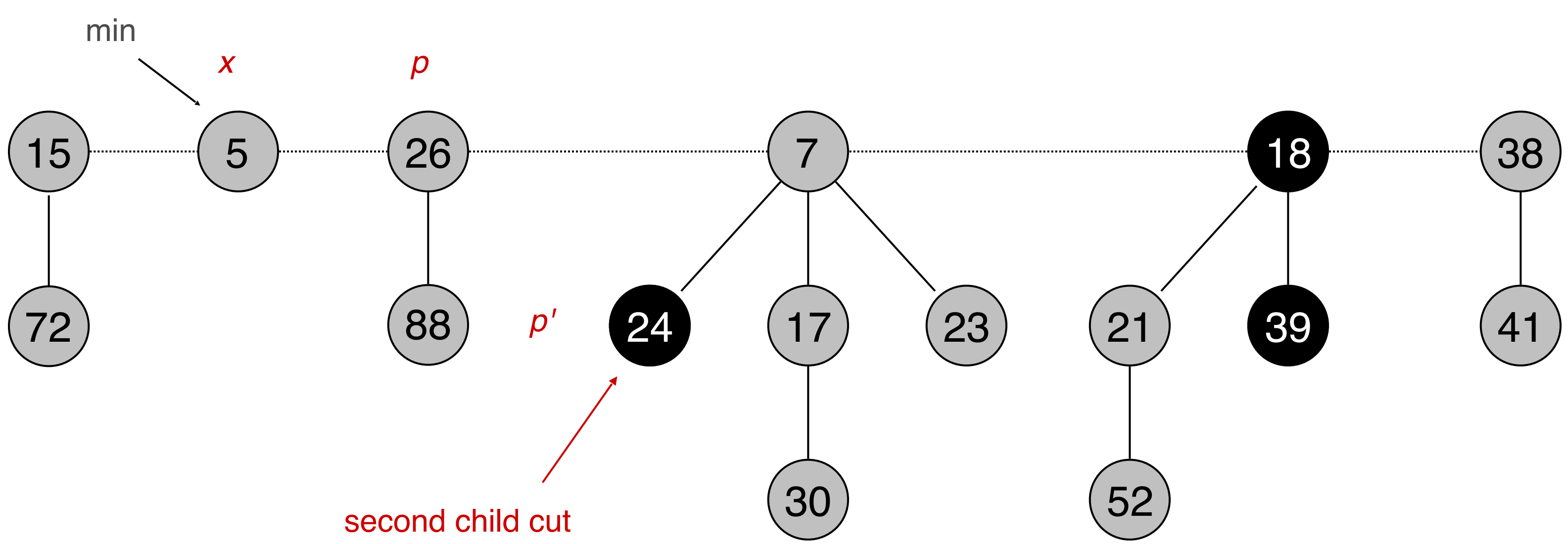


decrease-key of x from 35 to 5

Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).

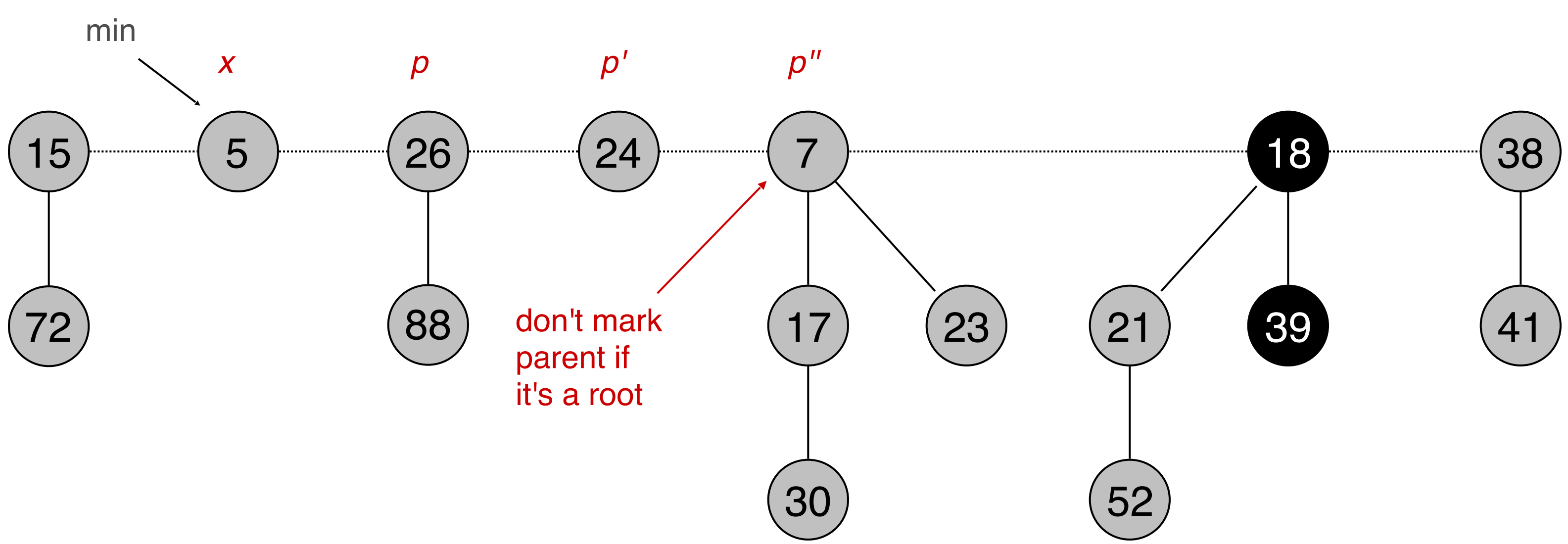


decrease-key of x from 35 to 5

Fibonacci Heaps: Decrease Key

Case 2b. [heap order violated]

- Decrease key of *x*.
- Cut tree rooted at *x*, meld into root list, and unmark.
- If parent *p* of *x* is unmarked (hasn't yet lost a child), mark it;
  Otherwise, cut *p*, meld into root list, and unmark
  (and do so recursively for all ancestors that lose a second child).



decrease-key of x from 35 to 5

Fibonacci Heaps: Decrease Key Analysis

Decrease-key.

$$\Phi(H) = trees(H) + 2 \cdot marks(H)$$

potential function

Actual cost. $O(c)$

- O(1) time for changing the key.
- O(1) time for each of c cuts, plus melding into root list.

Change in potential. $O(1) - c$

- $trees(H') = trees(H) + c.$
- $marks(H') \leq marks(H) - c + 2.$
- $\Delta\Phi \leq c + 2 \cdot (-c + 2) = 4 - c.$

Amortized cost. $O(1)$

```
FIB-HEAP-DECREASE-KEY(H, x, k)
if k > x.key
    error "new key is greater than cur
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)
```

Analysis Max Degree of Nodes

Analysis Summary

Insert. $O(1)$

Delete-min. $O(\mathit{rank}(H))$ †

Decrease-key. $O(1)$ †

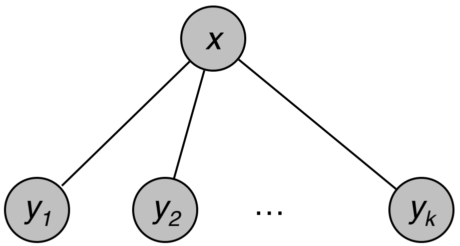
† amortized

Key lemma. $\mathit{rank}(H) = O(\log n)$.

number of nodes is exponential in rank

Lemma. Fix a point in time. Let x be a node, and let $y_1, \ldots, y_k$ denote its children in the order in which they were linked to x. Then:

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



Pf.

- When y_i was linked into x, x had at least i-1 children $y_1, \ldots, y_{i-1}$.
- Since only trees of equal rank are linked, at that time $rank(y_i) = rank(x_i) \geq i - 1$.
- Since then, y_i has lost at most one child.
- Thus, right now $rank(y_i) \geq i - 2$.

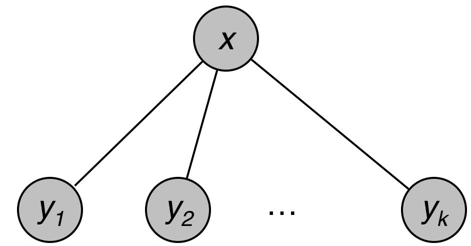
■

or y_i would have been cut

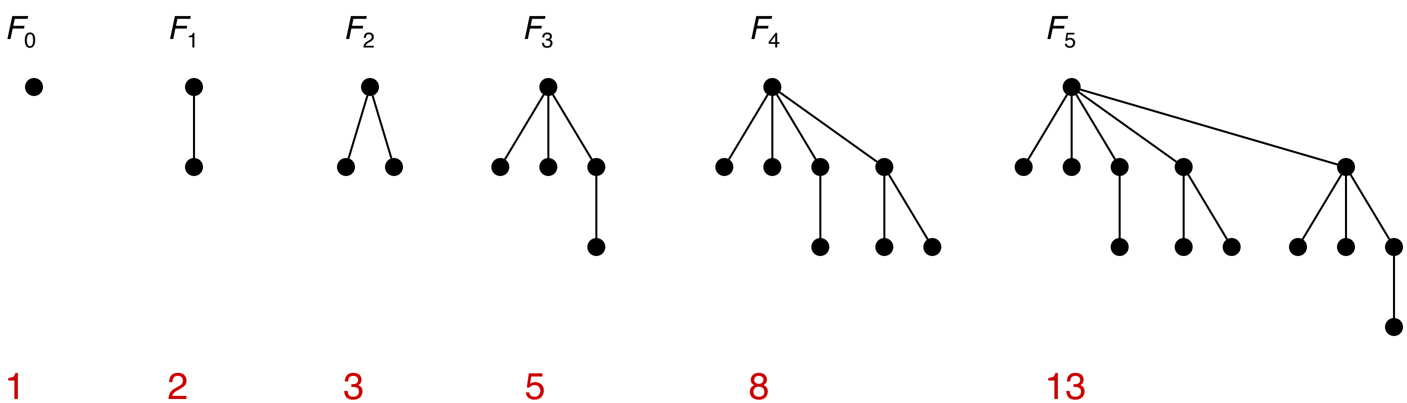
Fibonacci Heaps: Bounding the Rank

Lemma. Fix a point in time. Let x be a node, and let $y_1, \ldots, y_k$ denote its children in the order in which they were linked to x. Then:

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



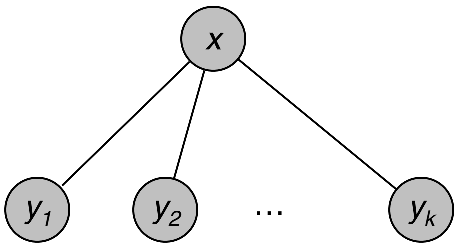
Def. Let F_k be smallest possible tree of rank k satisfying property.



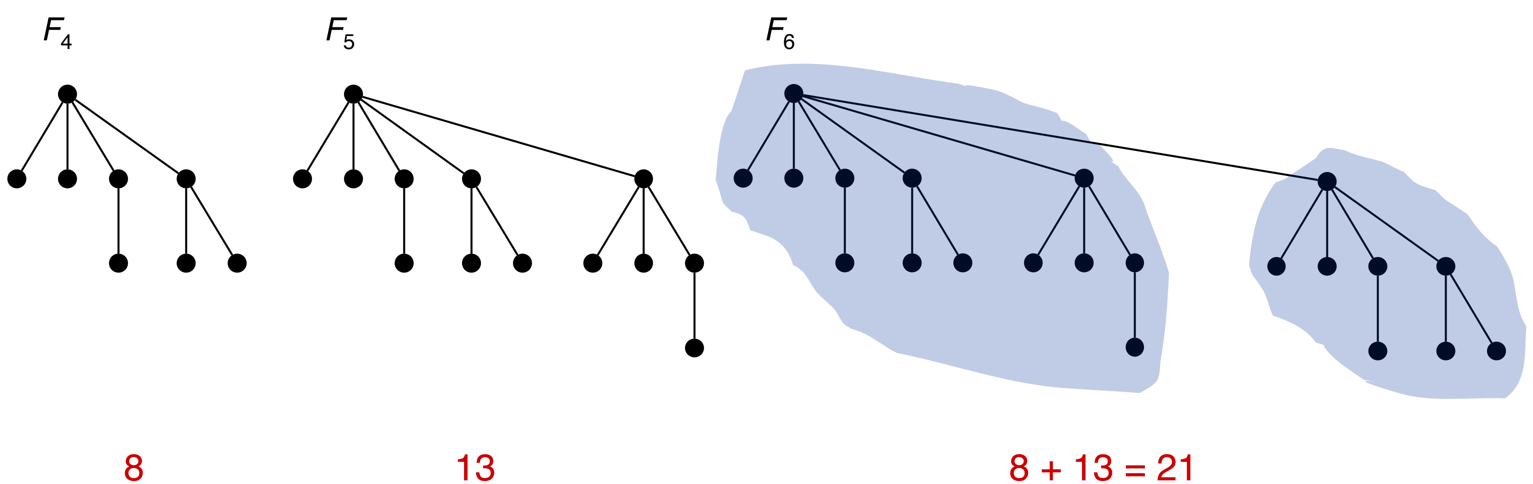
Fibonacci Heaps: Bounding the Rank

Lemma. Fix a point in time. Let x be a node, and let $y_1, \ldots, y_k$ denote its children in the order in which they were linked to x. Then:

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



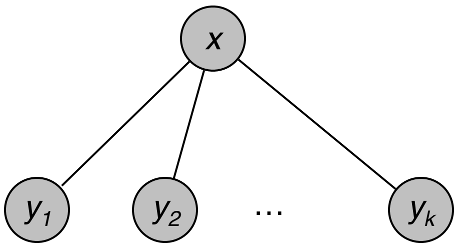
Def. Let F_k be smallest possible tree of rank k satisfying property.



Fibonacci Heaps: Bounding the Rank

Lemma. Fix a point in time. Let x be a node, and let $y_1, \ldots, y_k$ denote its children in the order in which they were linked to x. Then:

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



Def. Let F_k be smallest possible tree of rank k satisfying property.

Fibonacci fact. $F_k \geq \phi^k$, where $\phi = (1 + \sqrt{5}) / 2 \approx 1.618$.

Corollary. $rank(H) \leq \log_\phi n$.

golden ratio

Fibonacci Numbers

Fibonacci Numbers: Exponential Growth

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

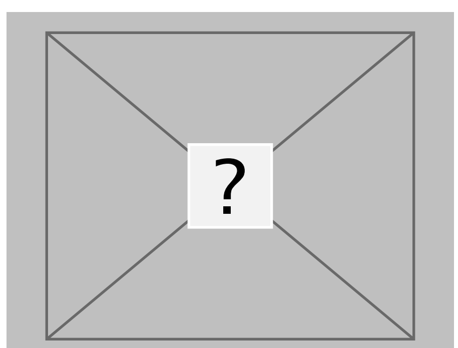
$$F_n = \begin{cases} 1 & n = 1, \\ 2 & n = 2, \\ F_{n-1} + F_{n-2} & o.w \end{cases}$$

slightly non-standard definition

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

Pf. [by induction on k]

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



(definition)

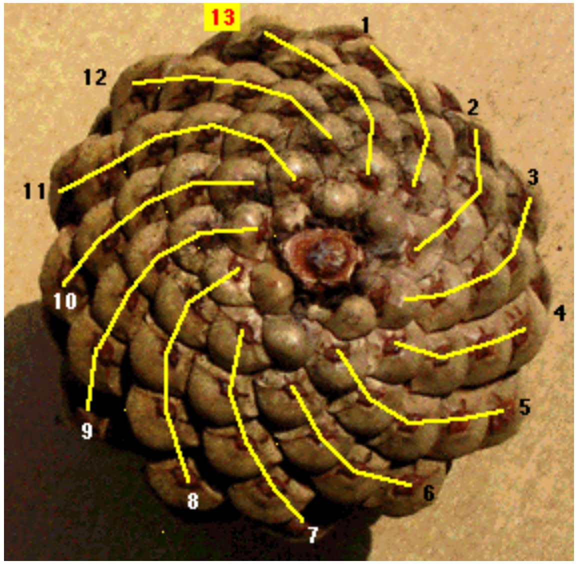
(inductive hypothesis)

(algebra)

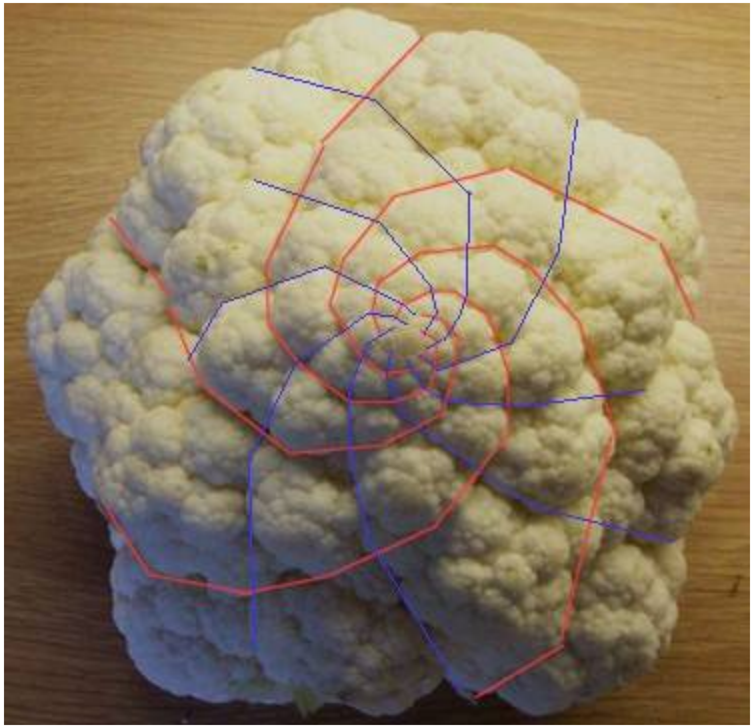
($\phi^2 = \phi + 1$)

(algebra)

Fibonacci Numbers and Nature



pinecone



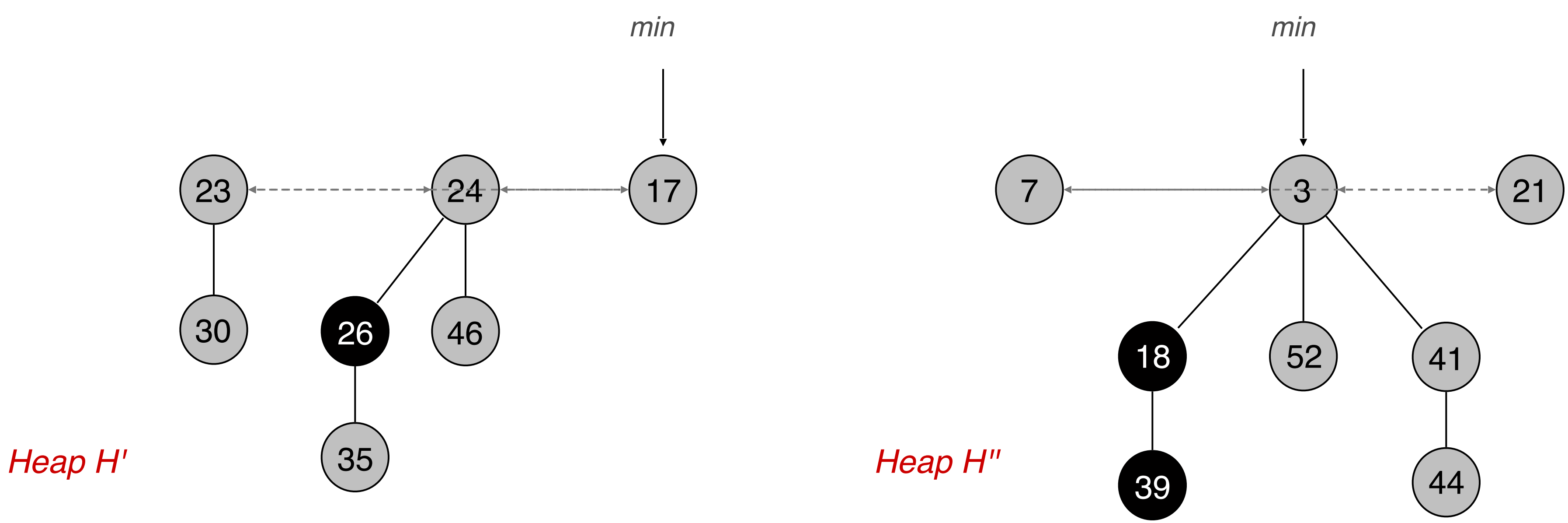
cauliflower

Union

Fibonacci Heaps: Union

Union. Combine two Fibonacci heaps.

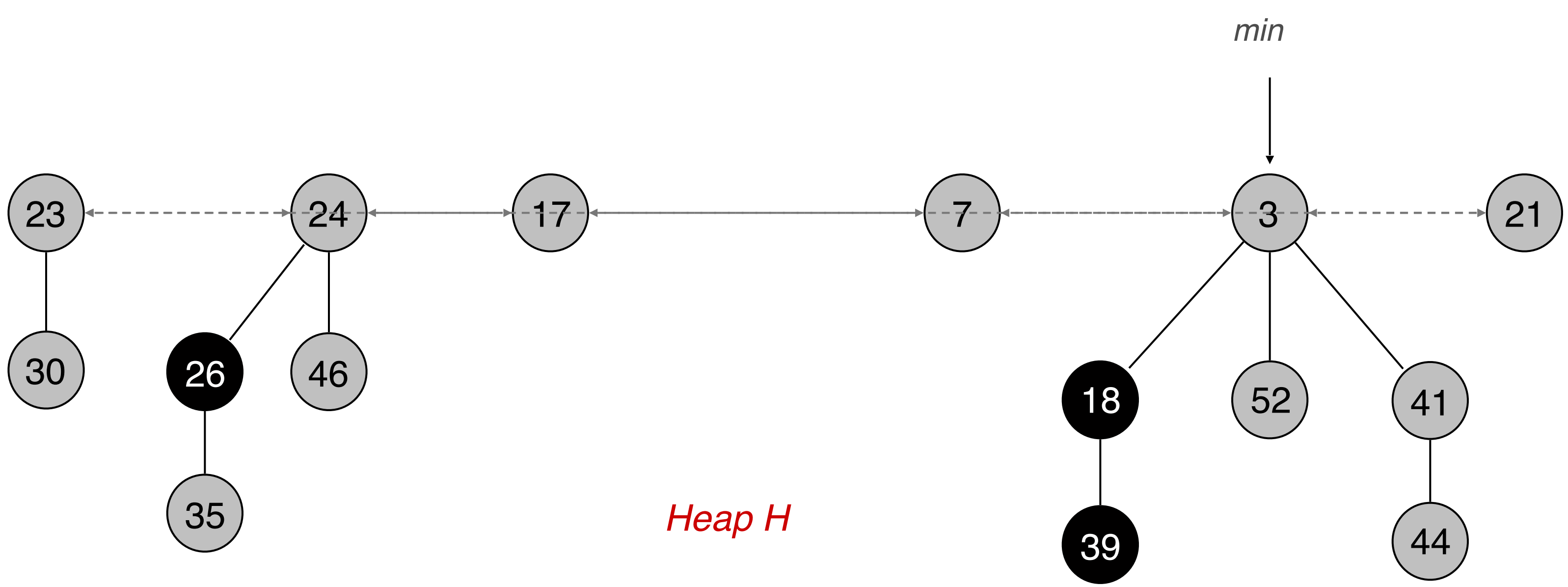
Representation. Root lists are circular, doubly linked lists.



Fibonacci Heaps: Union

Union. Combine two Fibonacci heaps.

Representation. Root lists are circular, doubly linked lists.



Fibonacci Heaps: Union

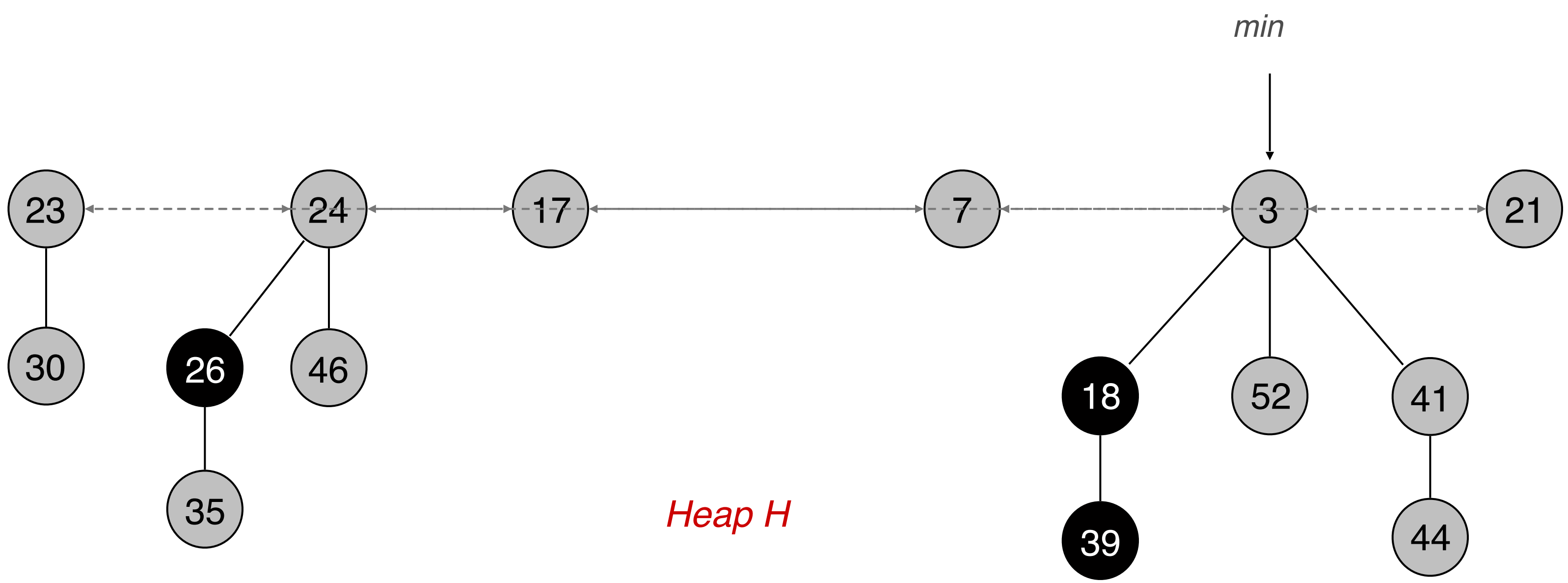
Actual cost. O(1)

Change in potential. 0

Amortized cost. O(1)

$$\Phi(H) = trees(H) + 2 \cdot marks(H)$$

potential function



Delete

Fibonacci Heaps: Delete

Delete node *x*.
- *decrease-key* of *x* to -∞.
- *delete-min* element in heap.

$$\Phi(H) = trees(H) + 2 \cdot marks(H)$$

potential function

Amortized cost. O(*rank*(*H*))
- O(1) amortized for *decrease-key*.
- O(*rank*(*H*)) amortized for *delete-min*.

Priority Queues Performance Cost Summary

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

n = number of elements in priority queue

† amortized