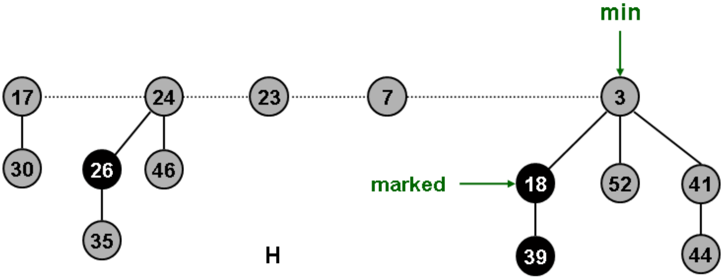
Fibonacci Heaps

Fibonacci Heaps

- Fibonacci heap history. Fredman and Tarjan (1986)
  - Ingenious data structure and analysis.
  - Original motivation: $O(m + nlogn)$ 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

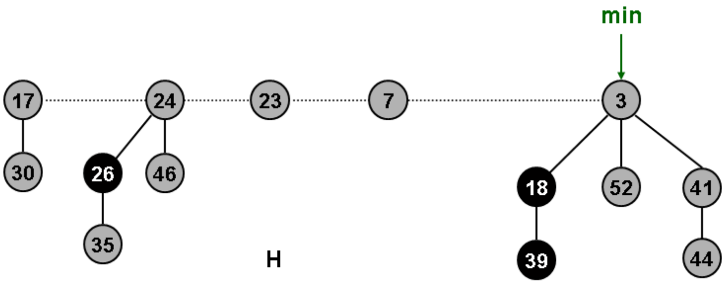
- Fibonacci heap.
  Set of min-heap ordered trees.



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



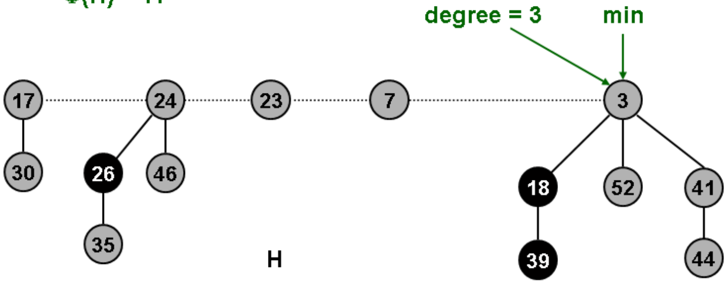
Fibonacci Heaps: Potential Function

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.

$t(H) = 5, \quad m(H) = 3$

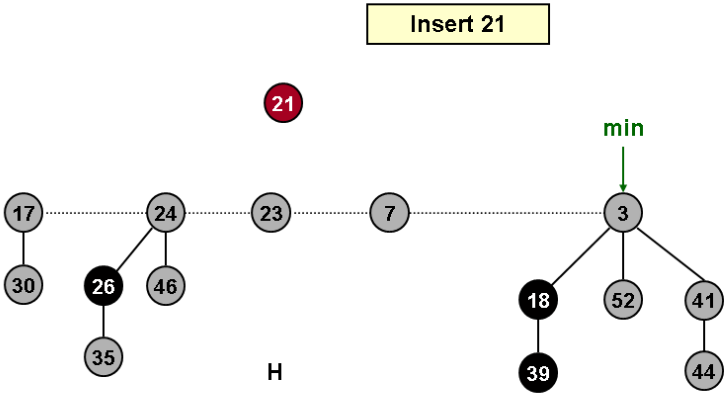
$\Phi(H) = 11$



Fibonacci Heaps: Insert

Insert.

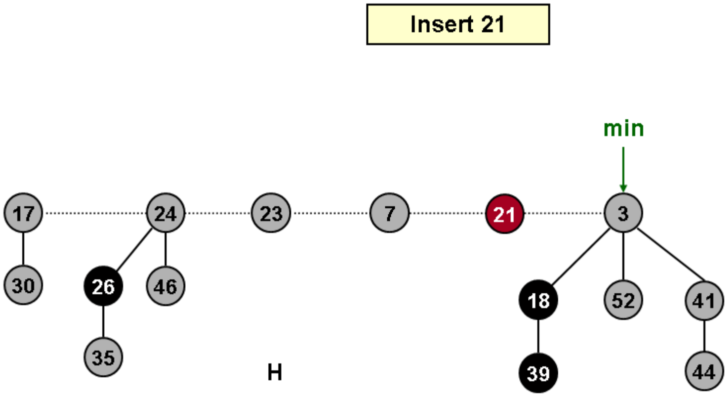
- Create a new singleton tree.
- Add to left of min pointer.
- Update min pointer.



Fibonacci Heaps: Insert

Insert.

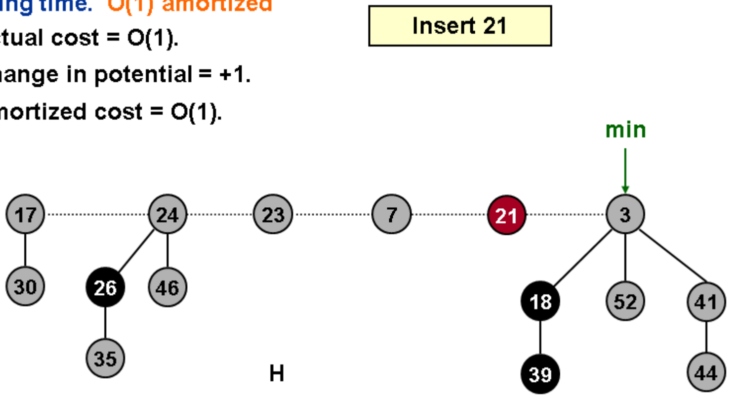
- Create a new singleton tree.
- Add to left of min pointer.
- Update min pointer.



Fibonacci Heaps: Insert

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

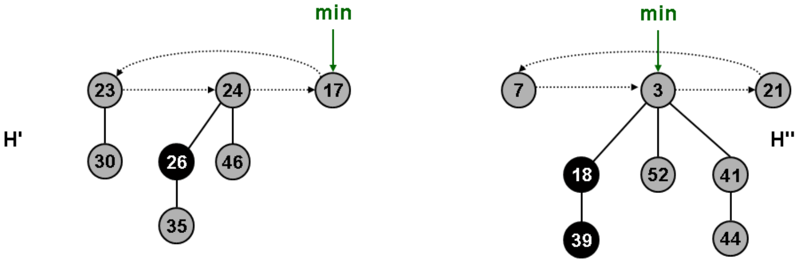
Running time. **O(1) amortized**
- Actual cost = O(1).
- Change in potential = +1.
- Amortized cost = O(1).



Binomial Heap: Union

Union.

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



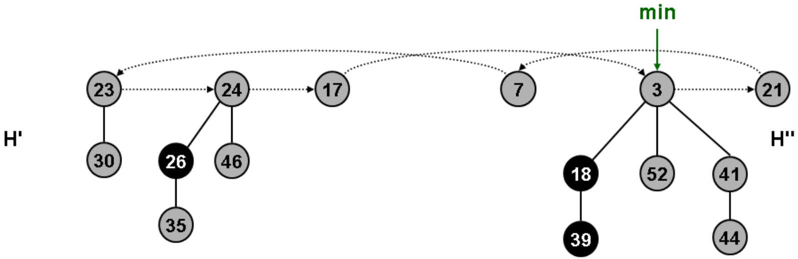
Fibonacci Heaps: Union

Union.

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

Running time. O(1) amortized

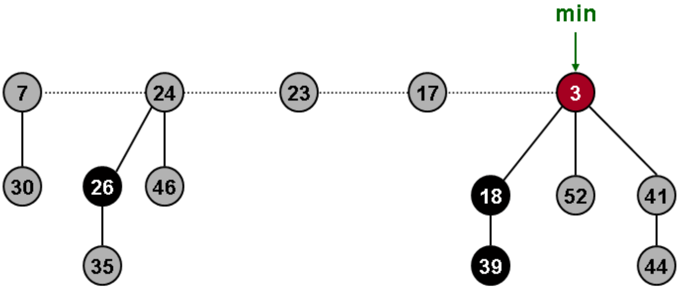
- Actual cost = O(1).
- Change in potential = 0.
- Amortized cost = O(1).



Fibonacci Heaps: Delete Min

Delete min.

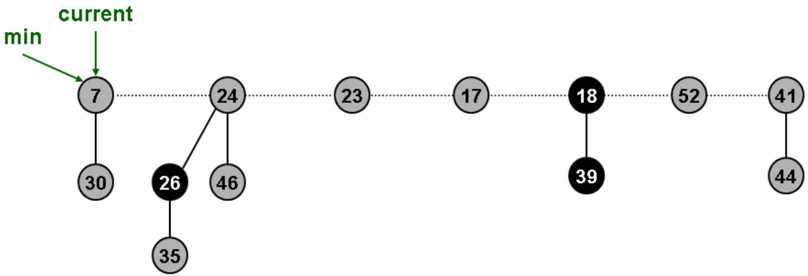
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Delete Min

Delete min.

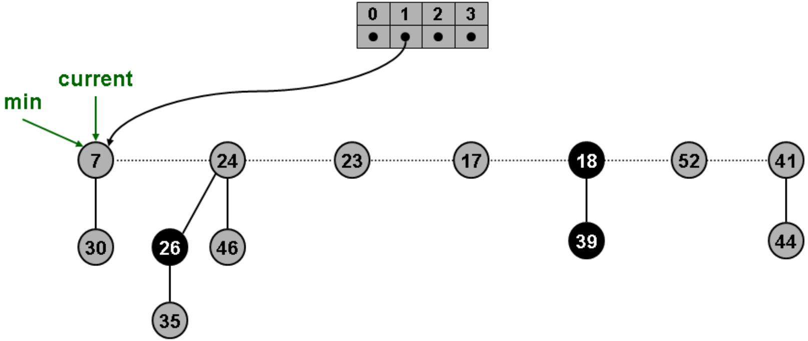
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Delete Min

Delete min.

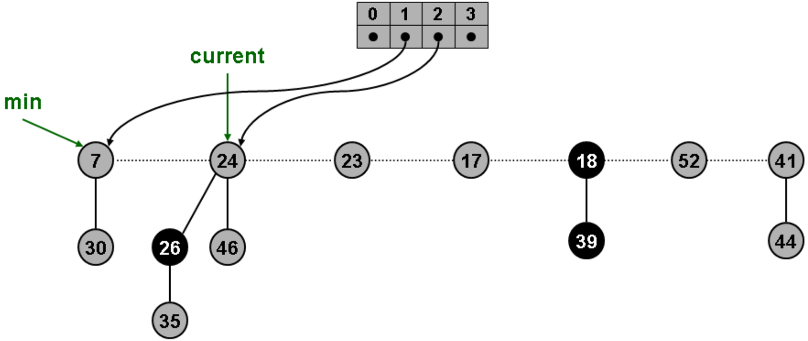
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Delete Min

Delete min.

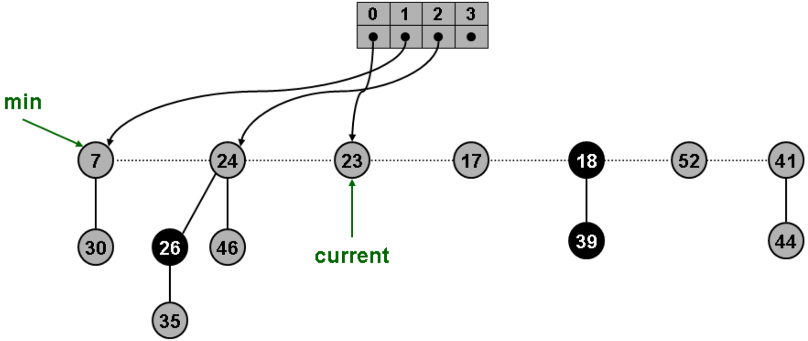
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Delete Min

Delete min.

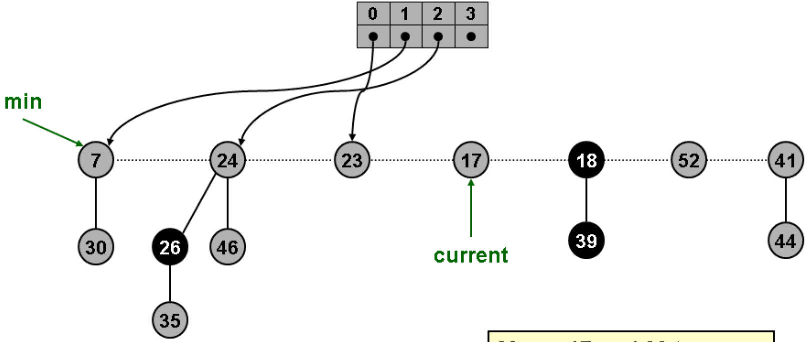
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



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.

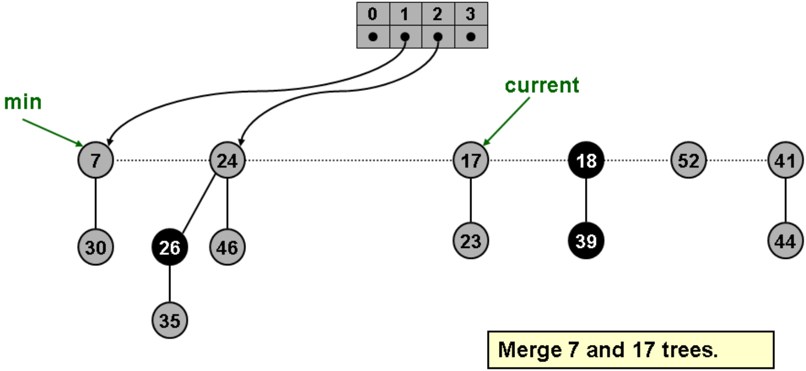


Merge 17 and 23 trees.

Fibonacci Heaps: Delete Min

Delete min.

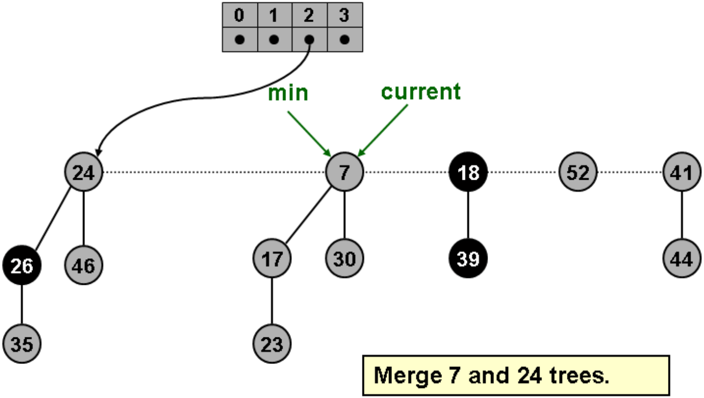
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Delete Min

Delete min.

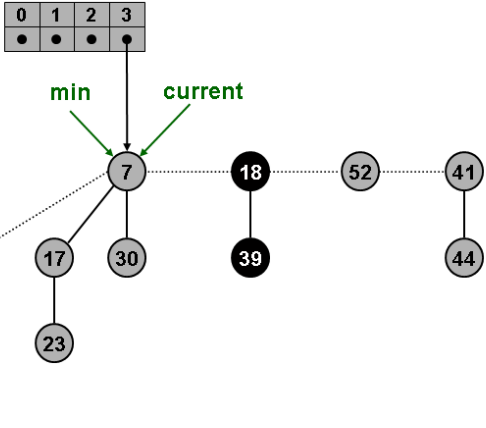
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Delete Min

Delete min.

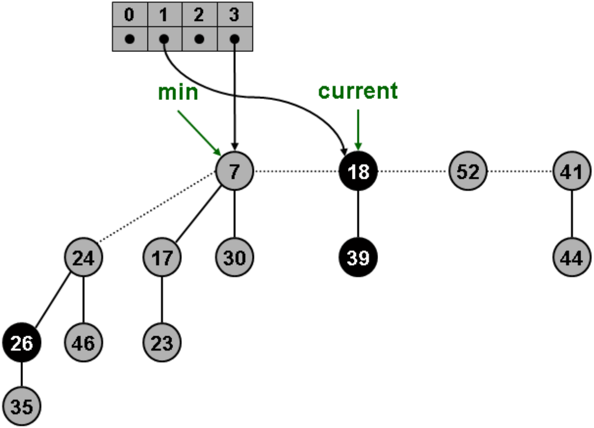
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Delete Min

Delete min.

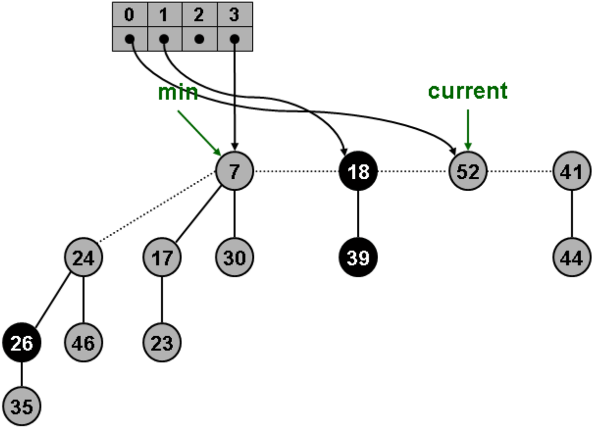
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Delete Min

Delete min.

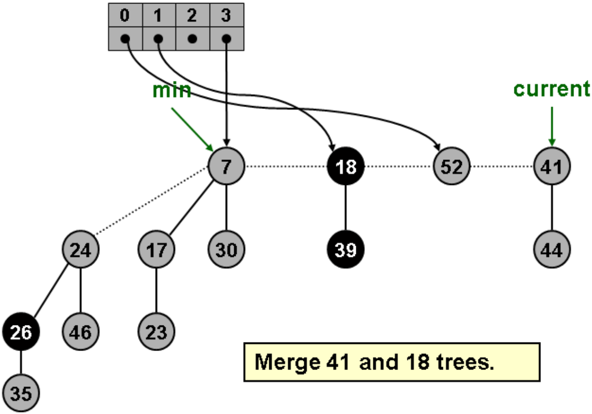
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



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.

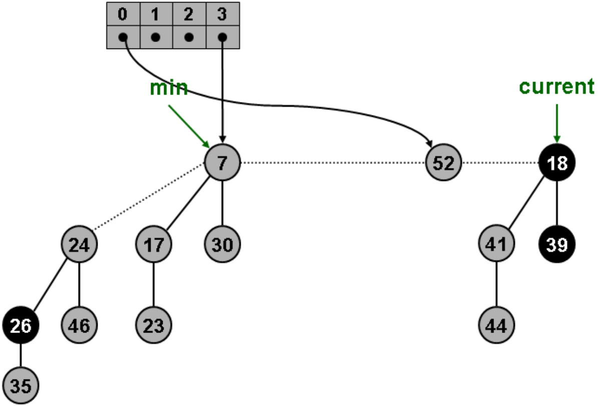


Merge 41 and 18 trees.

Fibonacci Heaps: Delete Min

Delete min.

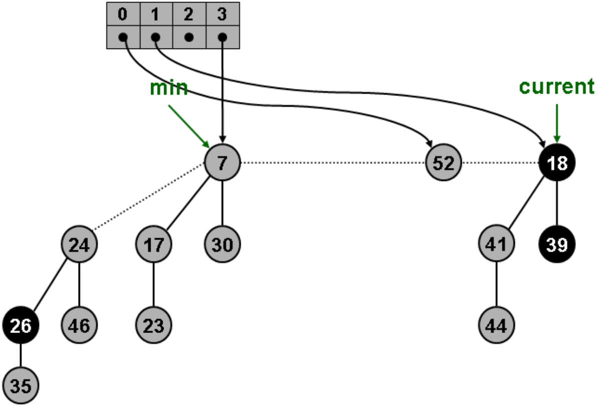
- Delete min and concatenate its children into root list.
- Consolidate trees so that no two roots have same degree.



Fibonacci Heaps: Delete Min

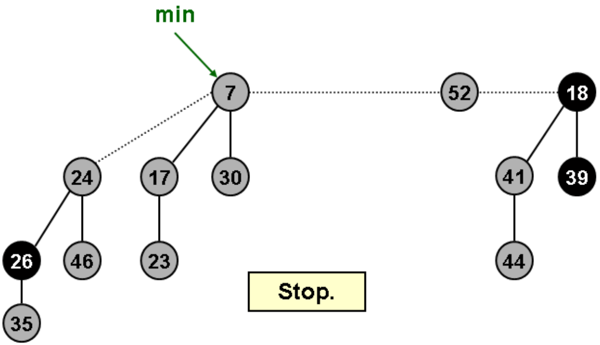
Delete min.

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



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

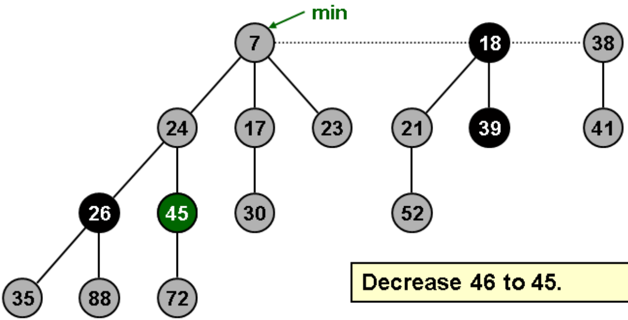
Is amortized cost of $O(D(n))$ good?

- 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 $D(n) \le \lfloor \log_2 N \rfloor$
- Yes, if we support Decrease-key in clever way.
  - we'll show that $D(n) \le \lfloor \log_\phi N \rfloor$, 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

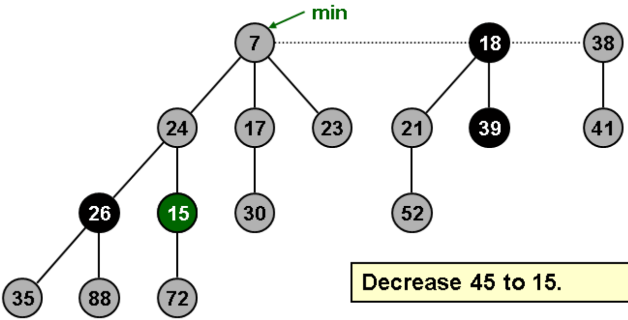


Decrease 46 to 45.

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

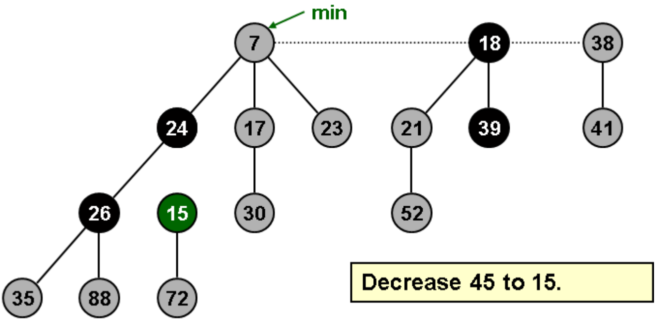


Decrease 45 to 15.

Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

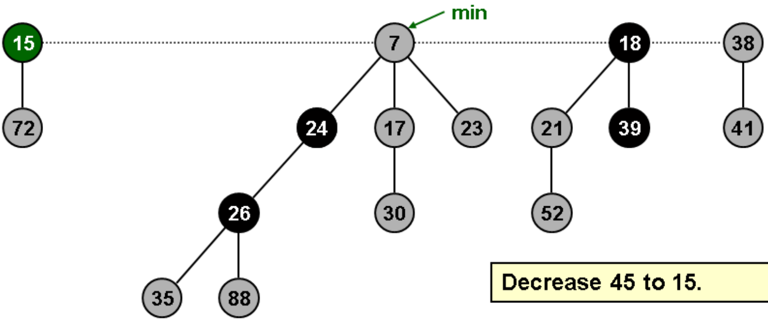
- Case 1: parent of x is unmarked.
  - decrease key of x to k
  - cut off link between x and its parent
  - mark parent
  - add tree rooted at x to root list, updating heap min pointer



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

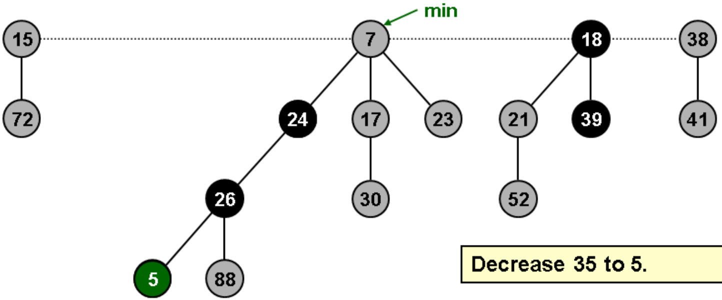


Decrease 45 to 15.

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.



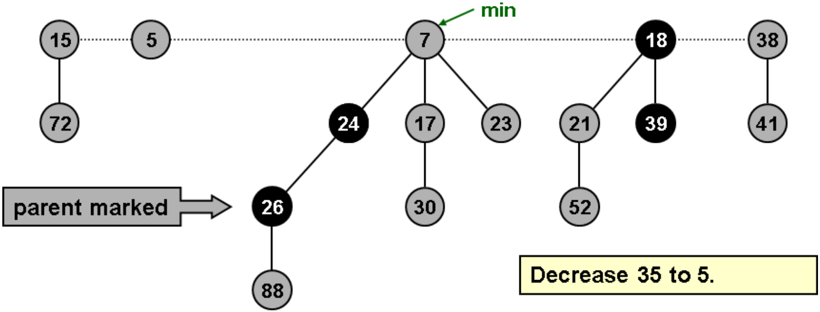
Decrease 35 to 5.

33

Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

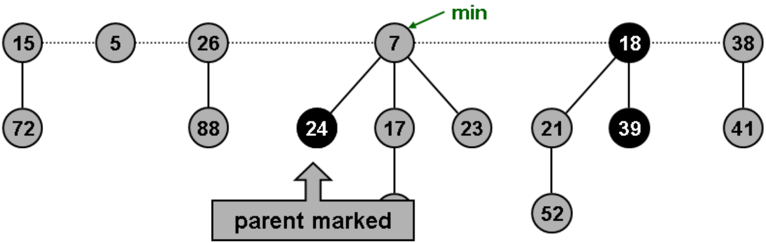
- Case 2: parent of x is marked.
  - decrease key of x to k
  - cut off link between x and its parent p[x], and add x to root list
  - cut off link between p[x] and p[p[x]], add p[x] to root list
    - If p[p[x]] unmarked, then mark it.
    - If p[p[x]] marked, cut off p[p[x]], unmark, and repeat.



Fibonacci Heaps: Decrease Key

Decrease key of element x to k.

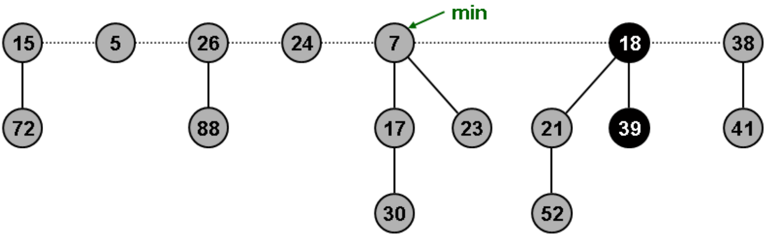
- Case 2: parent of x is marked.
  - decrease key of x to k
  - cut off link between x and its parent p[x], and add x to root list
  - cut off link between p[x] and p[p[x]], add p[x] to root list
    - If p[p[x]] unmarked, then mark it.
    - If p[p[x]] marked, cut off p[p[x]], unmark, and repeat.



Decrease 35 to 5.

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

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.