

Fibonacci Heaps

COMP 160 - Algorithms - Tufts University

Original Slides from Kevin Wayne, Princeton
<http://www.cs.princeton.edu/courses/archive/spring07/cos423/lectures/fibonacci-heap.ppt>

Slightly adapted by Roni Khardon

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

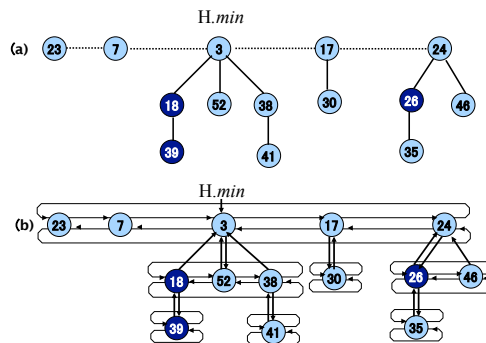
n = number of elements in priority queue † amortized

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.

Fibonacci Heaps

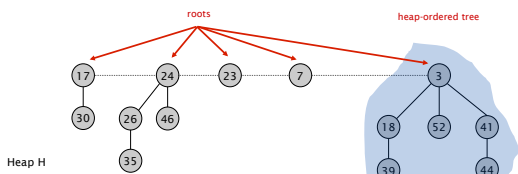
- History.** [Fredman and Tarjan, 1986]
- Ingenious data structure and analysis.
 - Original motivation: improve Dijkstra's shortest path algorithm
 - Repeat: extract-min
 - for all neighbors
 - if new value lower then decrease key
 - Complexity reduced from $O(E \log V)$ to $O(E + V \log V)$.

Our pictures vs. the detailed implementation



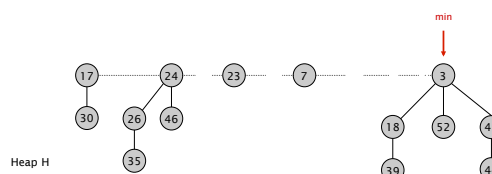
Fibonacci Heaps: Structure

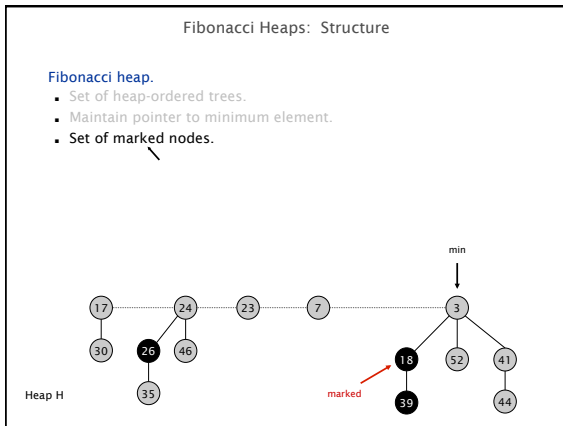
- Fibonacci heap.** each parent smaller than its children
- Set of **heap-ordered trees**.
 - Maintain pointer to minimum element.
 - Set of marked nodes.



Fibonacci Heaps: Structure

- Fibonacci heap.**
- Set of heap-ordered trees.
 - Maintain pointer to minimum element. find-min takes $O(1)$ time
 - Set of marked nodes.



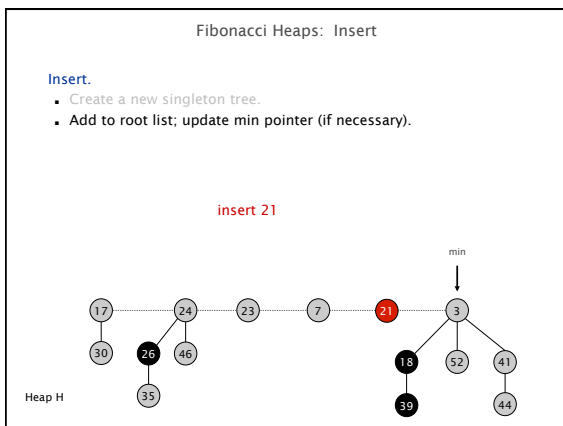
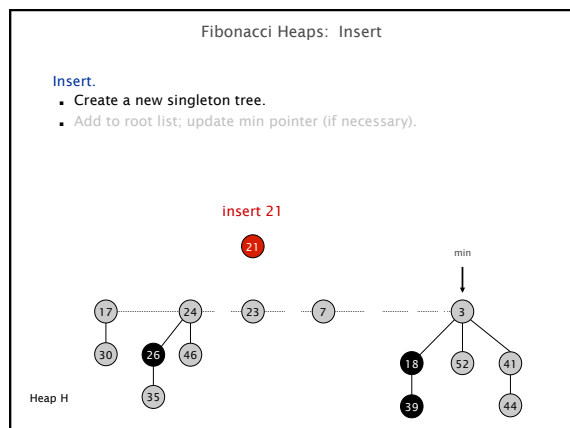


Part I: intuition: insert, extract-min, decrease-key

First we go through some ideas for the operations and from there for the potential function

Cost will turn out to depend on rank and we will make sure via extra operations that rank is bounded

Insert



Delete Min

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.

Fibonacci Heaps: Delete Min

Delete min.

- Delete min; meld its children into root list; update min.
- But now we have to find the new min element and this may be expensive → plan to pay for this step using potential function
- potential proportional to length of root list

Decrease Key

Fibonacci Heaps: Decrease Key

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 2a. [heap order violated]

- Decrease key of x .
- Cut tree rooted at x , meld into root list, and unmark.

Fibonacci Heaps: Decrease Key

Case 2a. [heap order violated]

- Decrease key of x .
- Cut tree rooted at x , meld into root list, and unmark.
- New Problem:** node with high rank but few descendants; this will interfere with our wish to maintain low rank
- Fix:** restructure more (and use marked nodes in potential function)

Fibonacci Heaps: Notation

Notation.

- n = number of nodes in heap.
- $\text{rank}(x)$ = number of children of node x .
- $\text{rank}(H)$ = max rank of any node in heap H .
- $\text{trees}(H)$ = number of trees in heap H .
- $\text{marks}(H)$ = number of marked nodes in heap H .

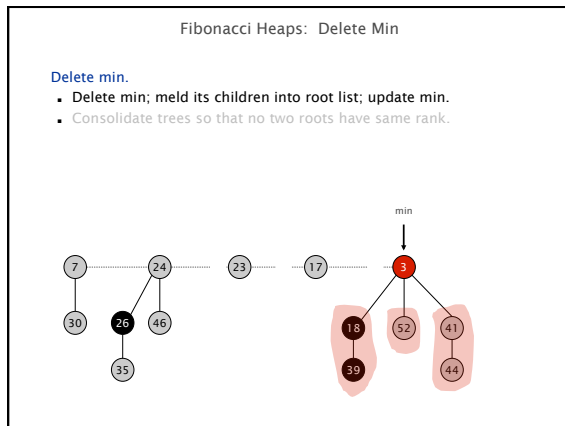
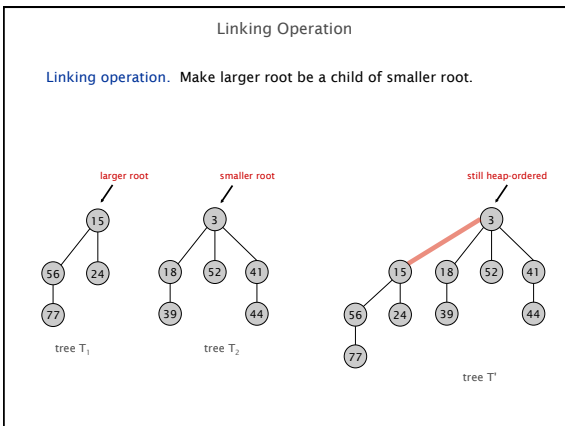
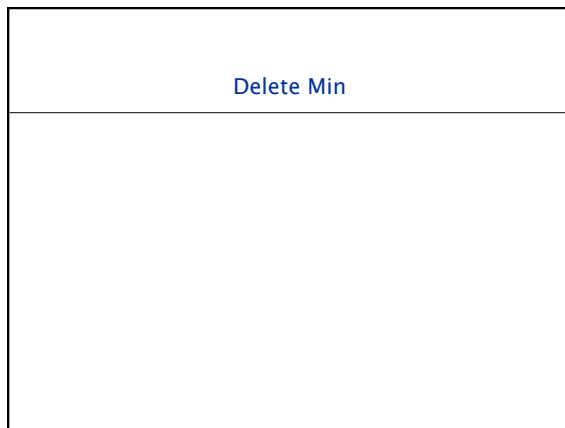
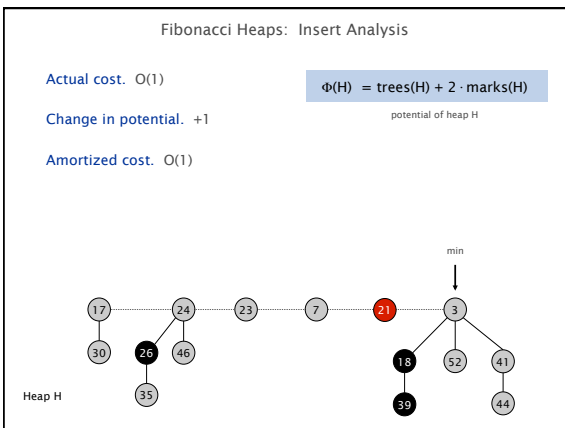
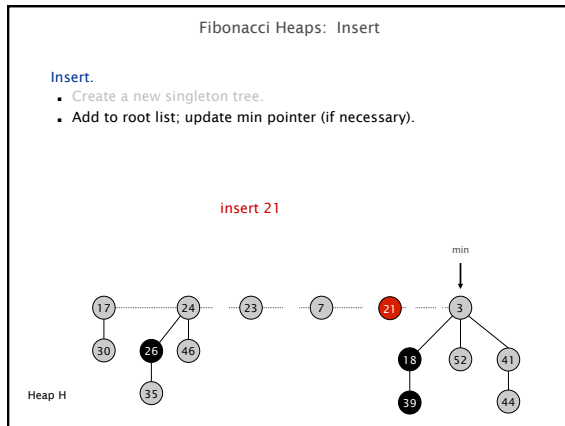
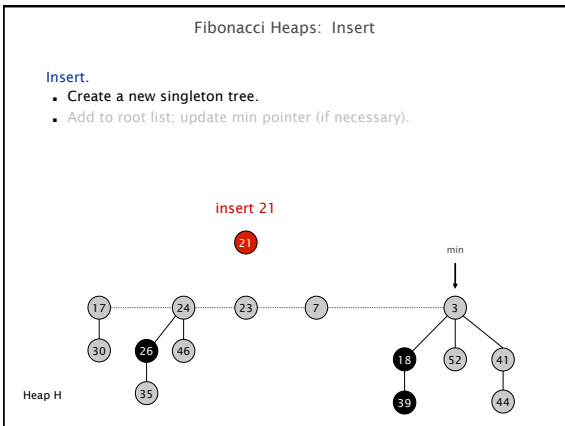
Fibonacci Heaps: Potential Function

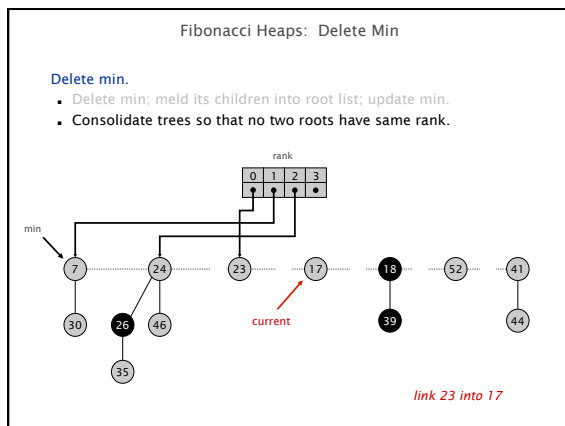
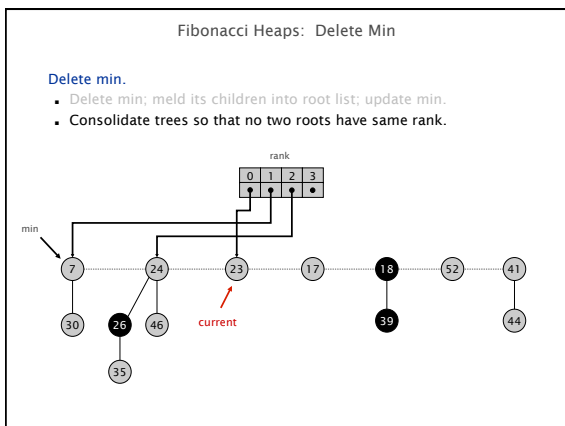
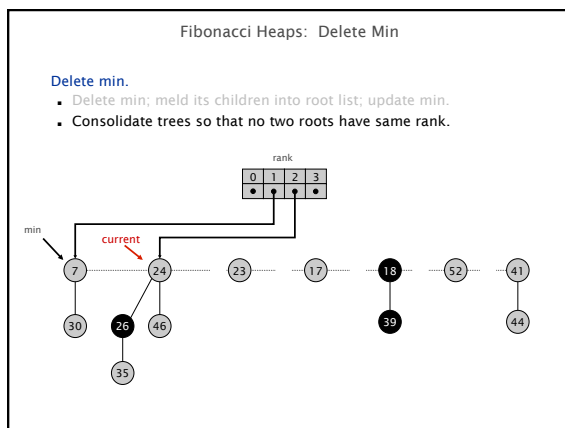
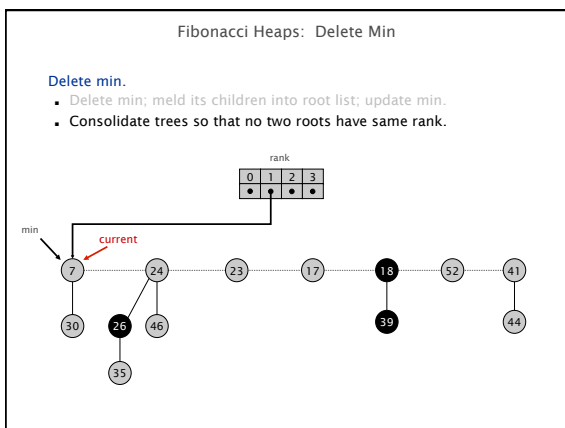
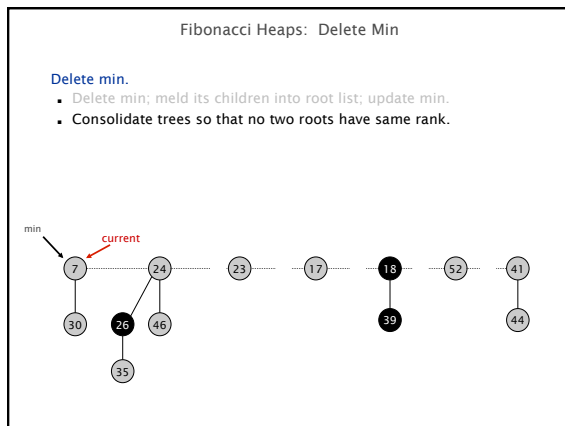
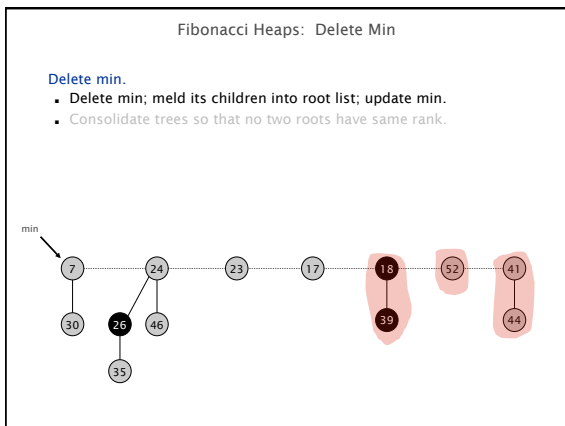
$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$

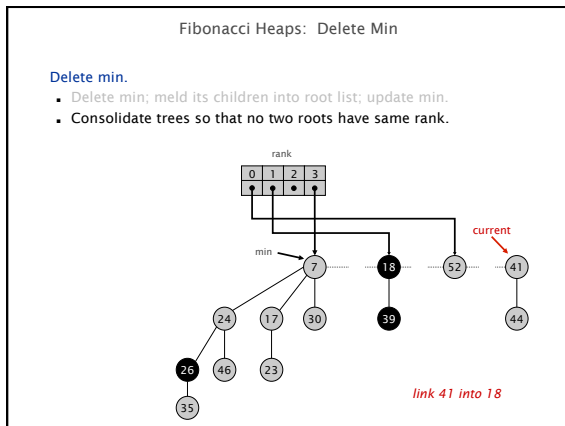
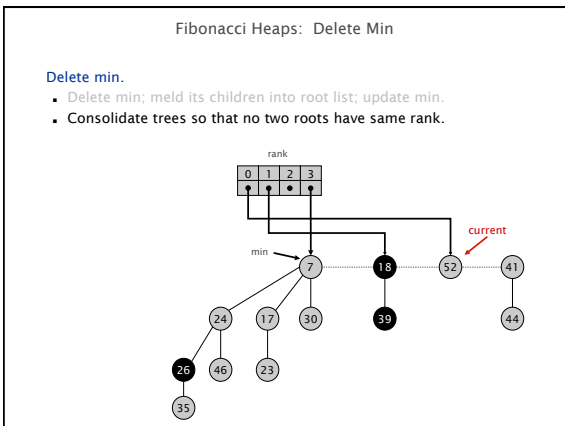
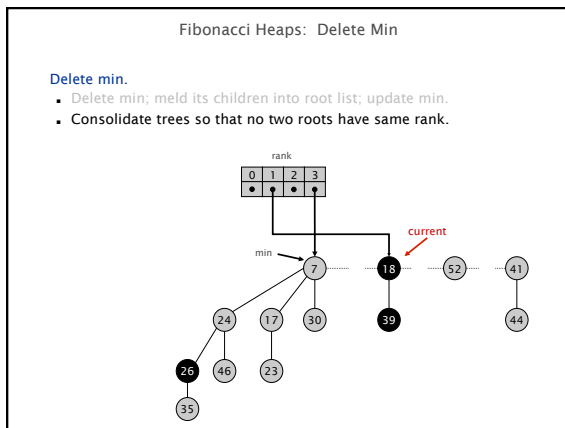
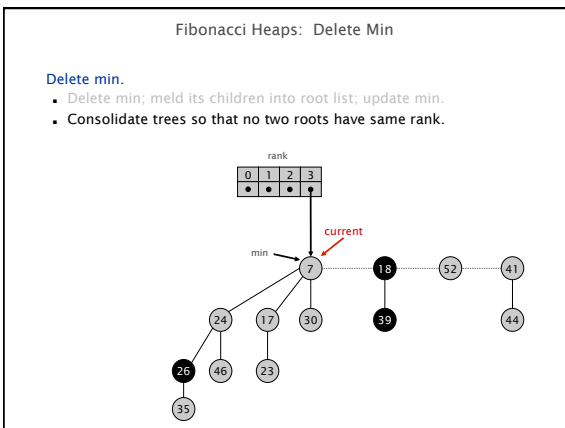
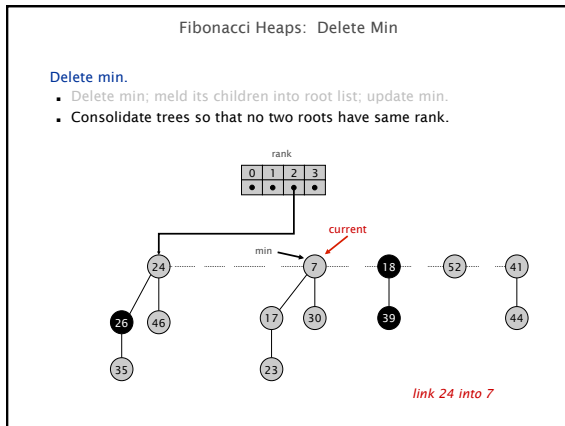
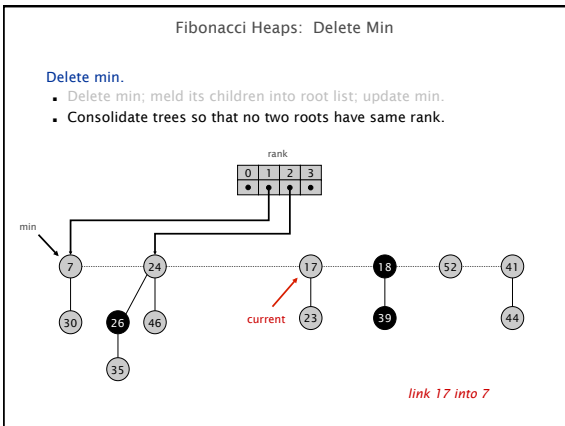
potential of heap H

$\text{trees}(H) = 5$ $\text{marks}(H) = 3$ $\Phi(H) = 5 + 2 \cdot 3 = 11$

Insert







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 Analysis

Delete min.

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

potential function

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

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

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

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

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

Decrease Key

Fibonacci Heaps: Decrease Key

Intuition for decreasing 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).

decrease-key of x from 46 to 29

Fibonacci Heaps: Decrease Key

Case 1. [heap order not violated]

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

decrease-key of x from 46 to 29

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).

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).

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).

decrease-key of x from 29 to 15

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).

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 Analysis

Decrease-key. $\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$
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. $O(1) - c$

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

Amortized cost. $O(1)$

Analysis

Analysis Summary

Insert. $O(1)$
Delete-min. $O(\text{rank}(H))^\dagger$
Decrease-key. $O(1)^\dagger$
† amortized

Key lemma. $\text{rank}(H) = O(\log n)$.
number of nodes is exponential in rank

Fibonacci Heaps: Bounding the Rank

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

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

Pf.

- When y_i was linked into x , x had at least $i-1$ children y_1, \dots, y_{i-1} .
- Since only trees of equal rank are linked, at that time $\text{rank}(y_i) = \text{rank}(x) \geq i-1$.
- Since then, y_i has lost at most one child.
- Thus, right now $\text{rank}(y_i) \geq i-2$. 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, \dots, y_k denote its children in the order in which they were linked to x . Then:

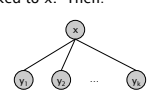
$$\text{rank}(y_i) \geq \begin{cases} 0 & \text{if } i=1 \\ i-2 & \text{if } i \geq 1 \end{cases}$$

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

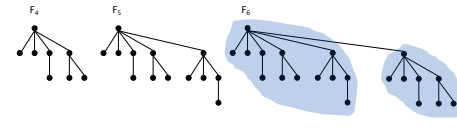
1 2 3 5 8 13

Fibonacci Heaps: Bounding the Rank

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

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


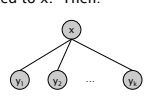
Def. Let F_k be smallest possible tree of rank k satisfying property.



8 13 8 + 13 = 21

Fibonacci Heaps: Bounding the Rank

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

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


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

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

Corollary. $\text{rank}(H) \leq \log_{\phi} n$. golden ratio

Fibonacci Numbers

Fibonacci Numbers: Exponential Growth

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

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

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

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 $F_{k+1} \geq \phi^{k+1}$

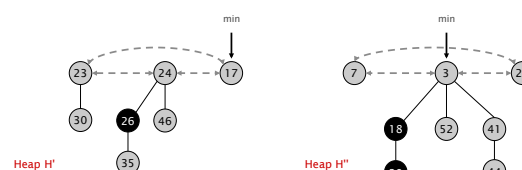
$$\begin{aligned} F_{k+2} &= F_k + F_{k+1} && \text{(definition)} \\ &\geq \phi^k + \phi^{k+1} && \text{(inductive hypothesis)} \\ &= \phi^k (1 + \phi) && \text{(algebra)} \\ &= \phi^k (\phi^2) && (\phi^2 = \phi + 1) \\ &= \phi^{k+2} && \text{(algebra)} \end{aligned}$$

Union

Fibonacci Heaps: Union

Union. Combine two Fibonacci heaps.

Representation. Root lists are circular, doubly linked lists.



Heap H' Heap H''

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. $O(1)$

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

potential function

Delete

Fibonacci Heaps: Delete

Delete node x.

- decrease-key of x to $-\infty$.
- delete-min element in heap.

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

potential function

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

- $O(1)$ amortized for decrease-key.
- $O(\text{rank}(H))$ 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