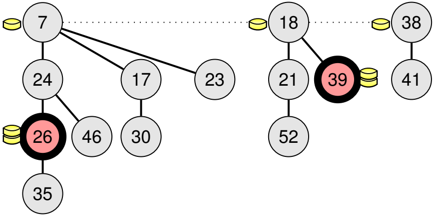


5.2 Fibonacci Heaps (Analysis)

Frank Stajano

Thomas Sauerwald

Lent 2016

UNIVERSITY OF CAMBRIDGE

Outline

Recap of INSERT, EXTRACT-MIN and DECREASE-KEY

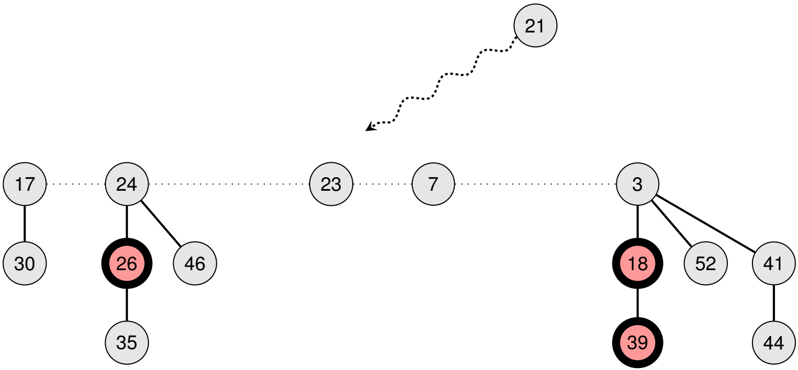
Glimpse at the Analysis

Amortized Analysis

Bounding the Maximum Degree

Fibonacci Heap: INSERT

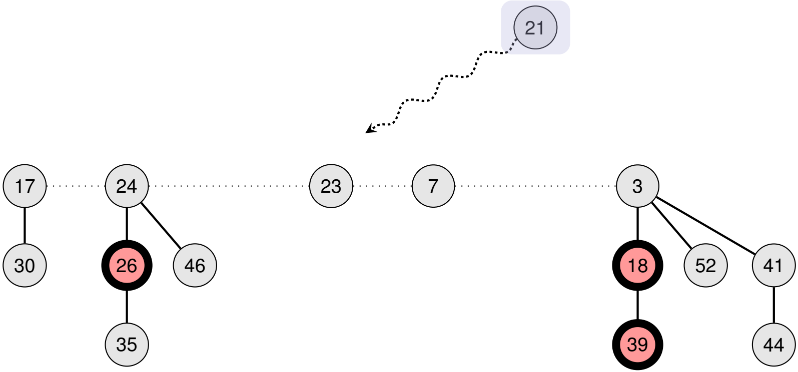
INSERT



Fibonacci Heap: INSERT

INSERT

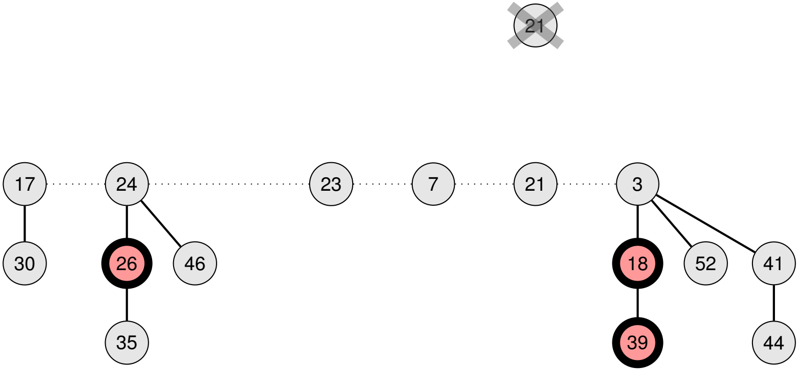
- Create a singleton tree



Fibonacci Heap: INSERT

INSERT

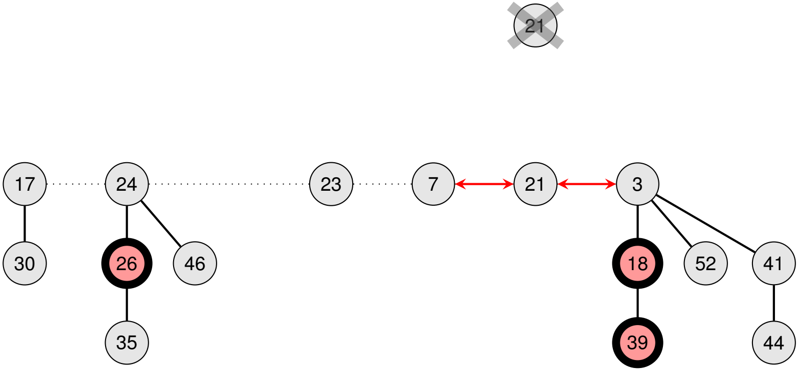
- Create a singleton tree
- Add to root list



Fibonacci Heap: INSERT

INSERT

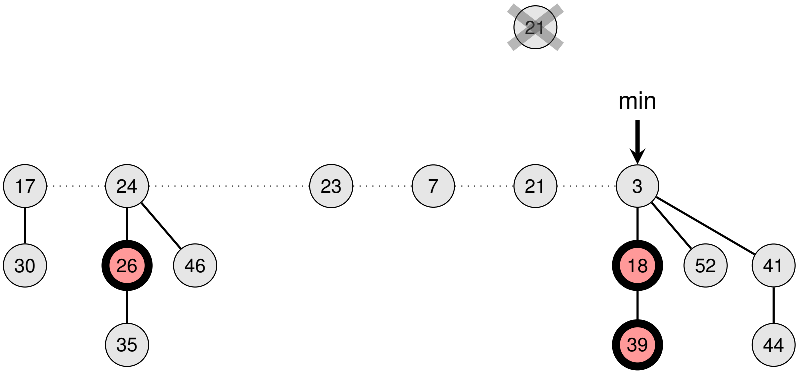
- Create a singleton tree
- Add to root list



Fibonacci Heap: INSERT

INSERT

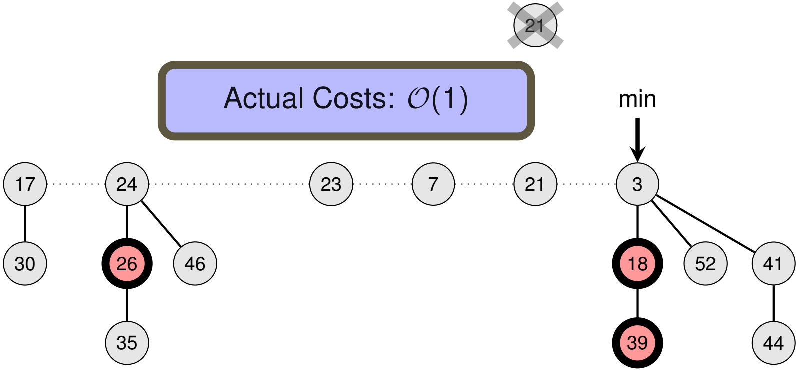
- Create a singleton tree
- Add to root list and update min-pointer (if necessary)



Fibonacci Heap: INSERT

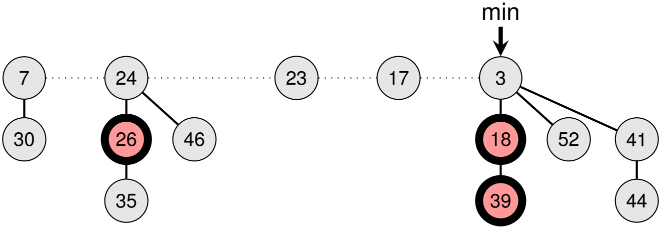
INSERT

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



Fibonacci Heap: EXTRACT-MIN

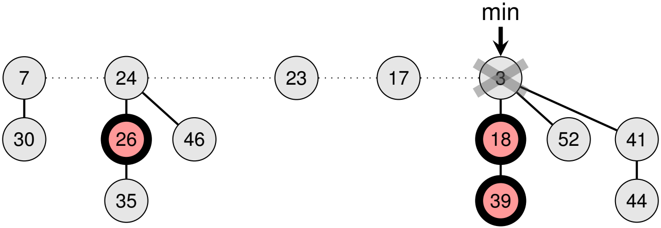
EXTRACT-MIN



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

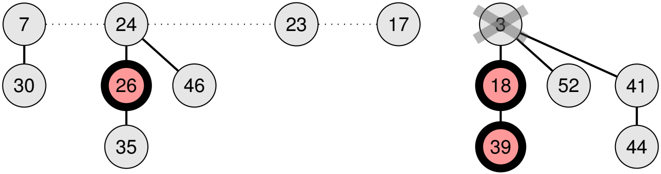
- Delete min



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

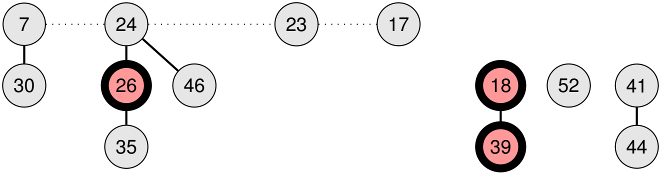
- Delete min ✓



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

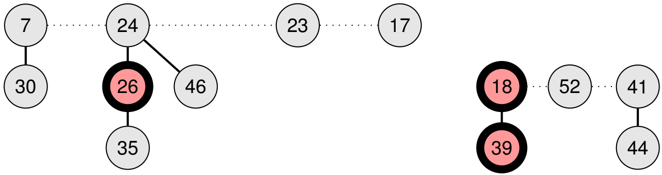
- Delete min ✓
- Meld childen into root list and unmark them



Fibonacci Heap: Extract-Min

Extract-Min

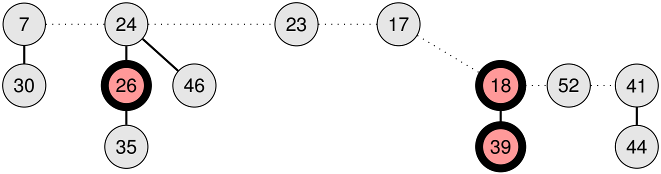
- Delete min ✓
- Meld childen into root list and unmark them



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

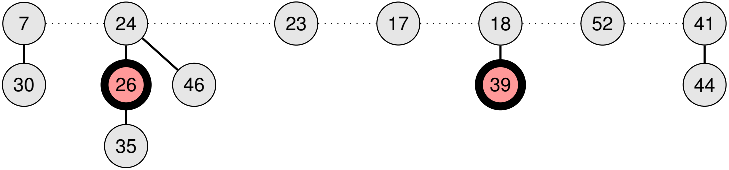
- Delete min ✓
- Meld childen into root list and unmark them



Fibonacci Heap: Extract-Min

Extract-Min

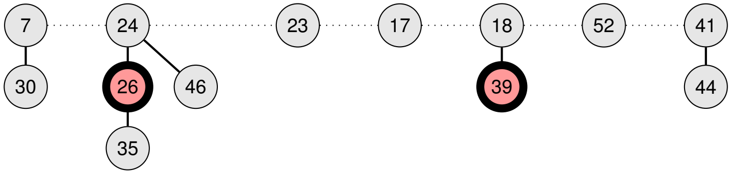
- Delete min ✓
- Meld childen into root list and unmark them ✓



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

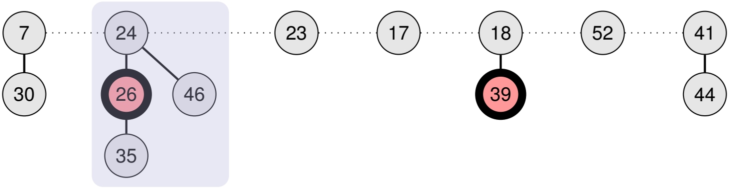
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

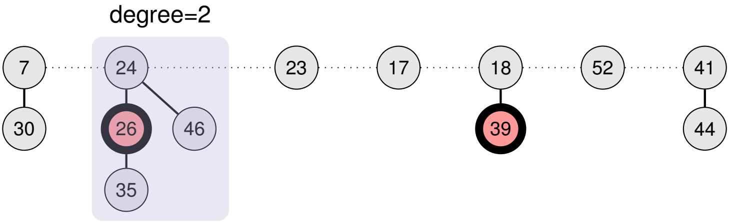
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

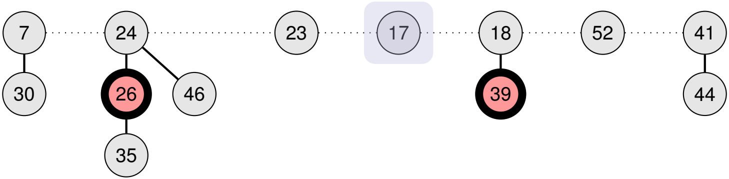
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: Extract-Min

Extract-Min

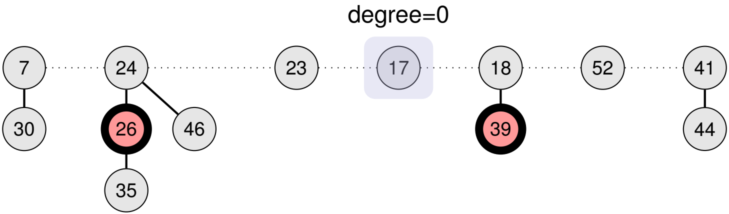
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

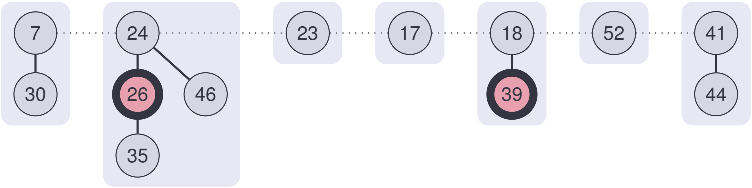
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

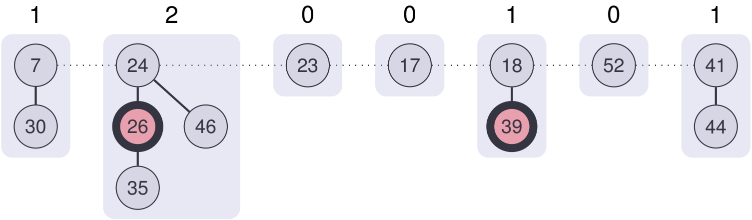
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: Extract-Min

Extract-Min

- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



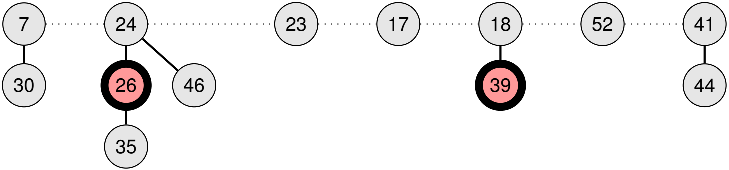
Fibonacci Heap: Extract-Min

Extract-Min

- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)

degree

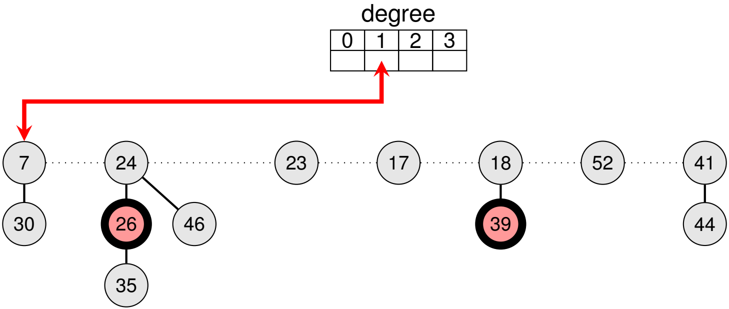
0	1	2	3



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

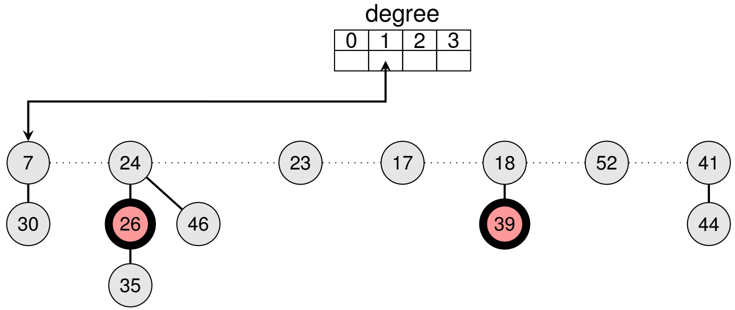
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

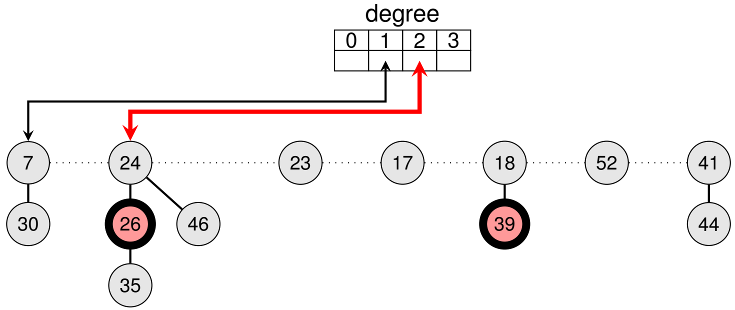
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

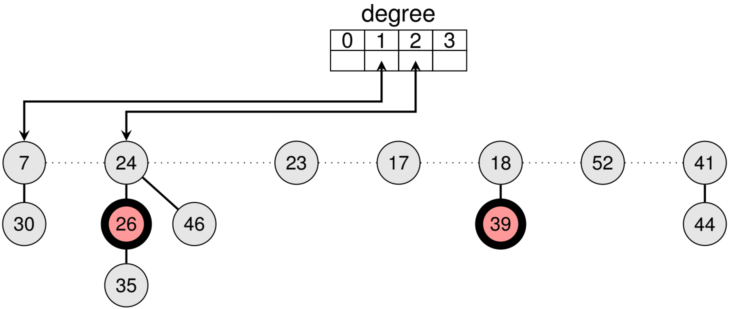
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: Extract-Min

Extract-Min

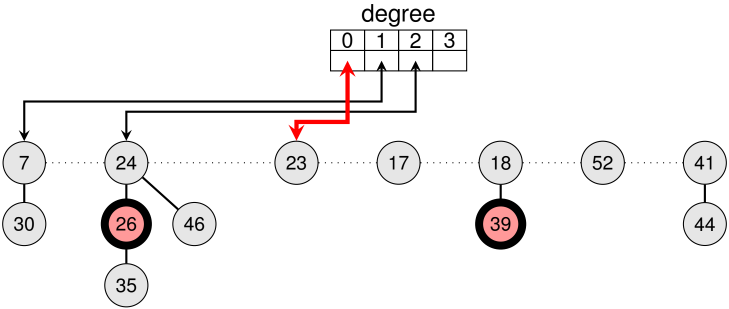
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

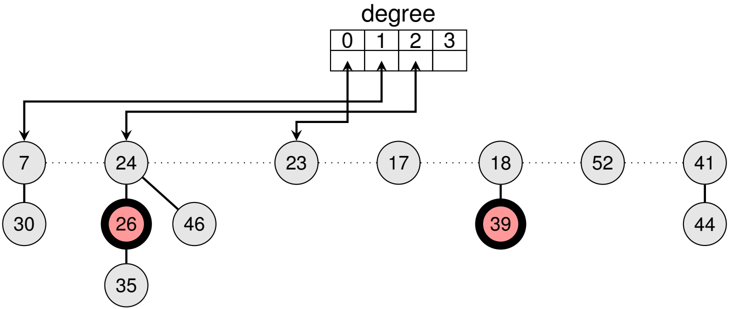
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

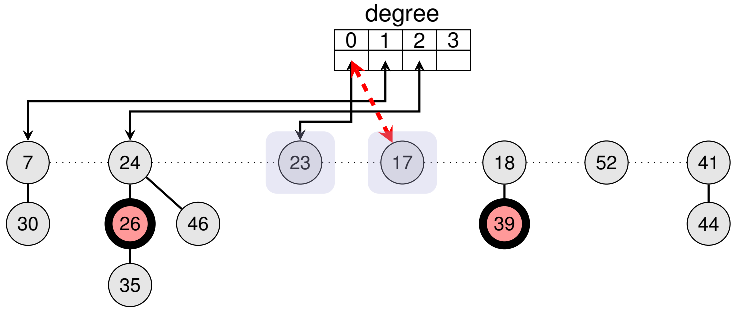
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

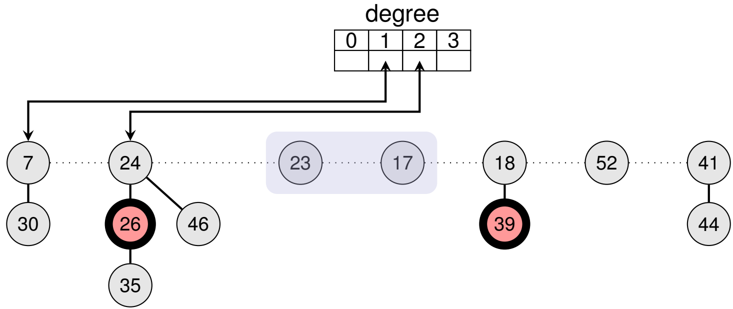
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

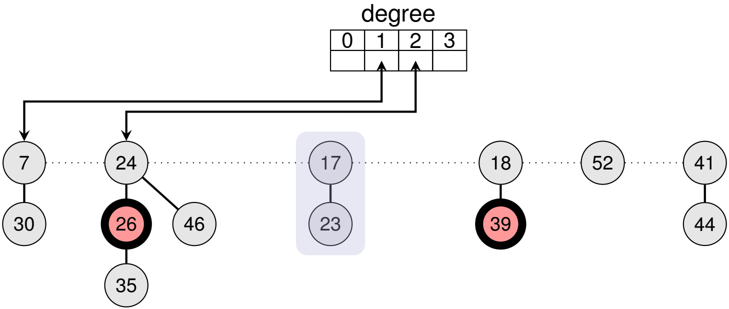
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

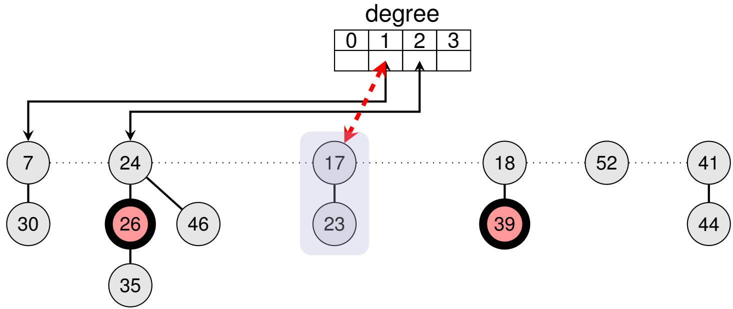
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: Extract-Min

Extract-Min

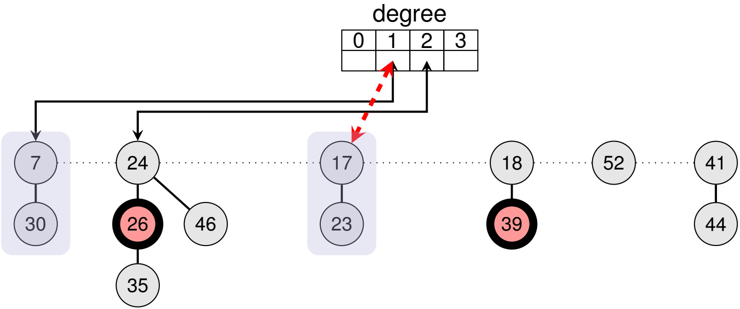
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: Extract-Min

Extract-Min

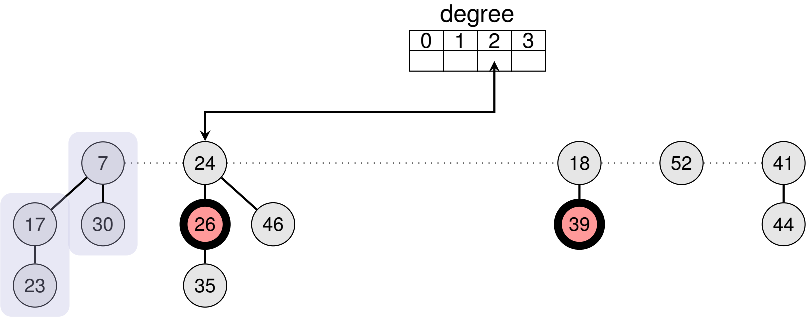
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: Extract-Min

Extract-Min

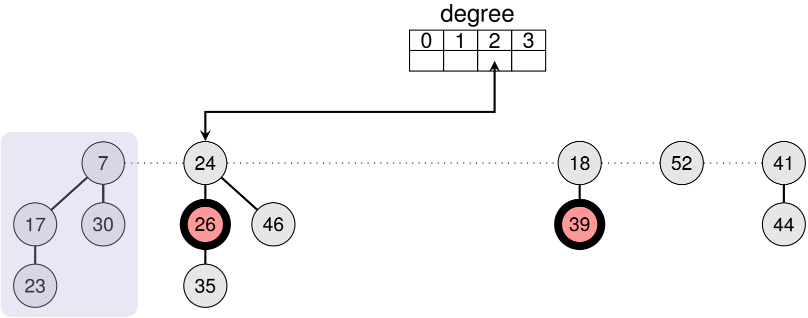
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: Extract-Min

Extract-Min

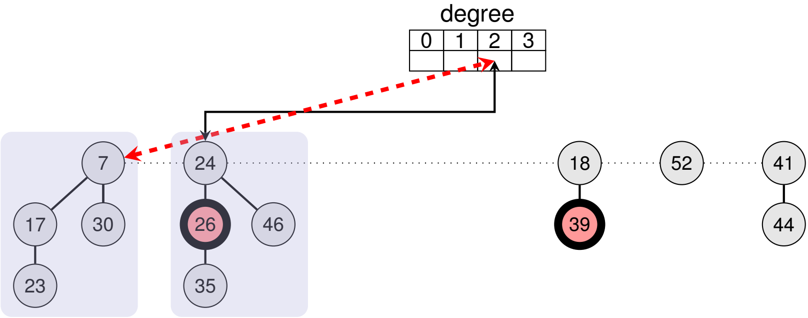
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



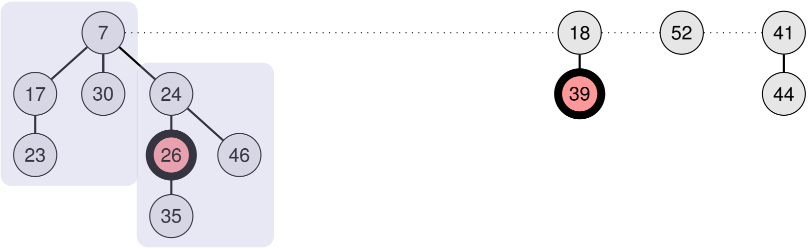
Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)

degree

0	1	2	3



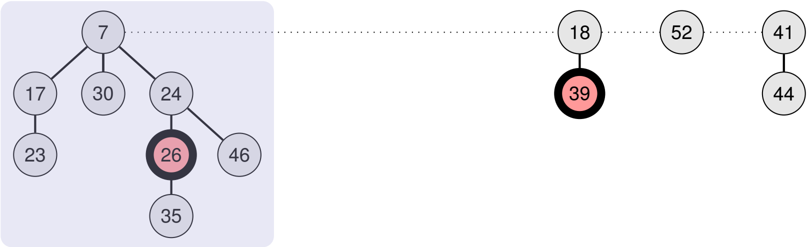
Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)

degree

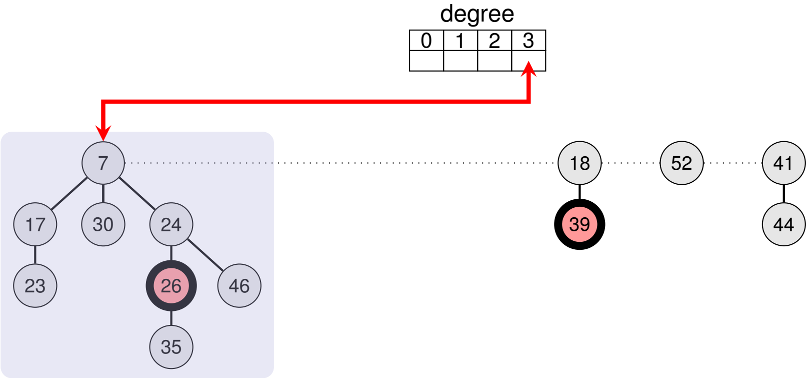
0	1	2	3



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

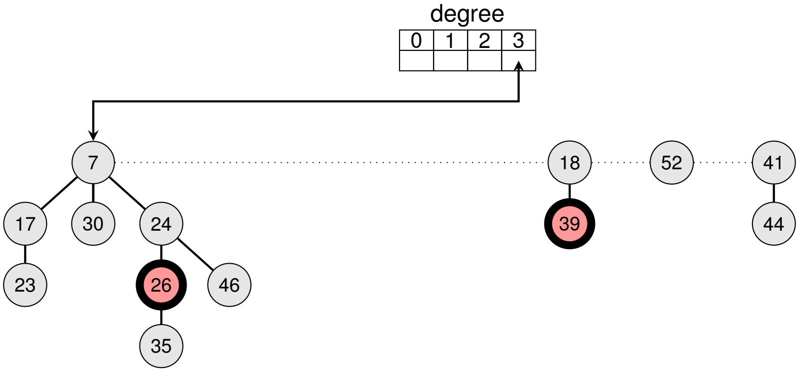
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

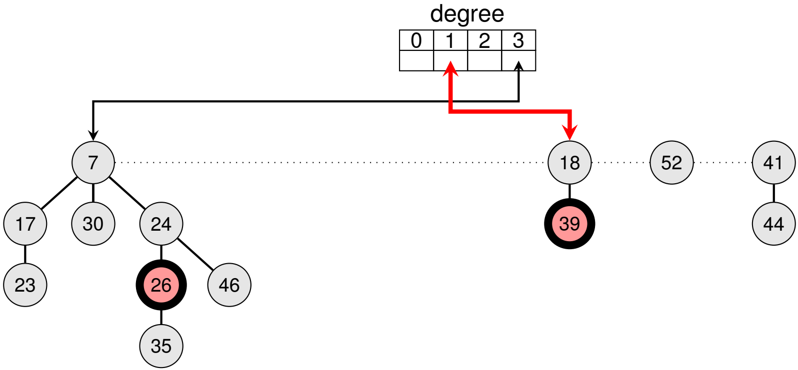
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

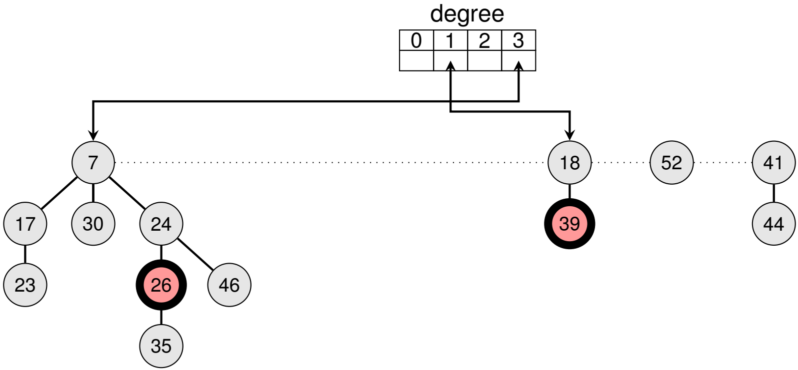
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: Extract-Min

Extract-Min

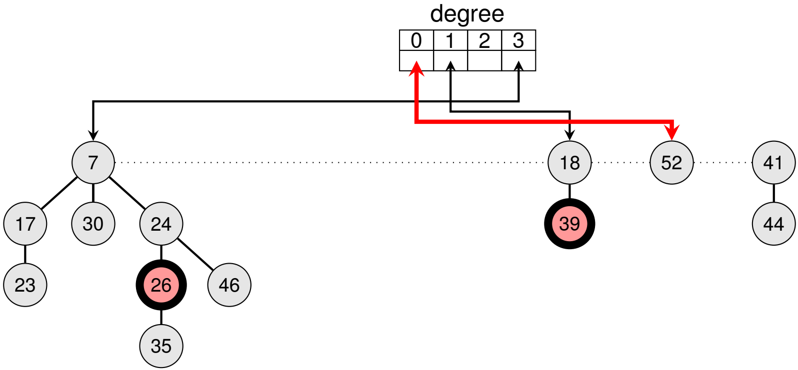
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

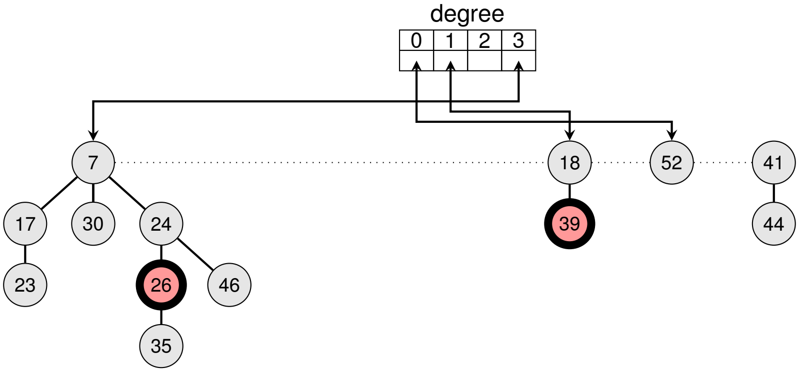
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: Extract-Min

Extract-Min

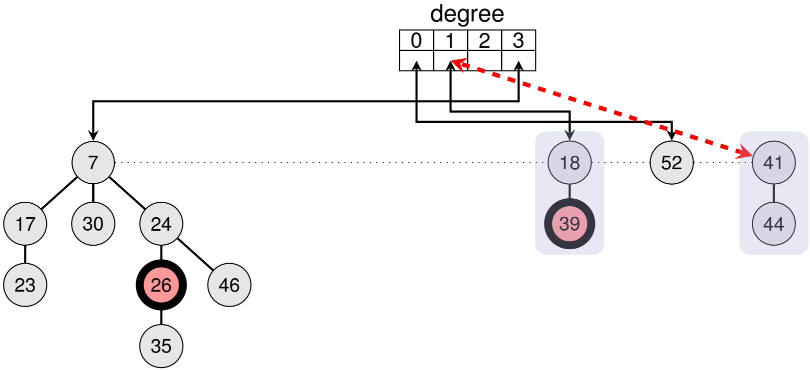
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

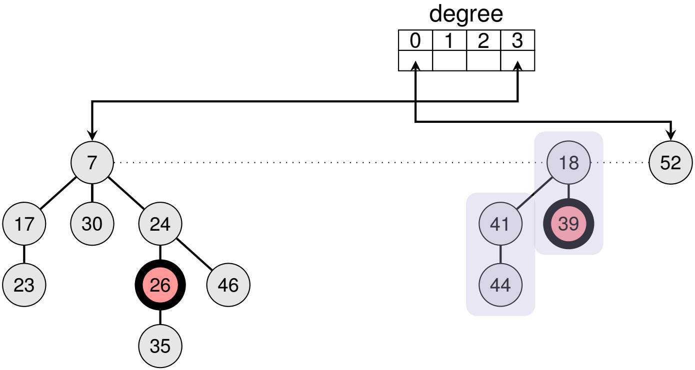
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

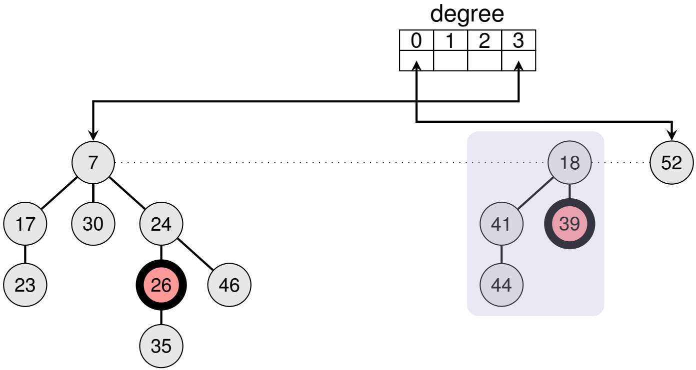
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

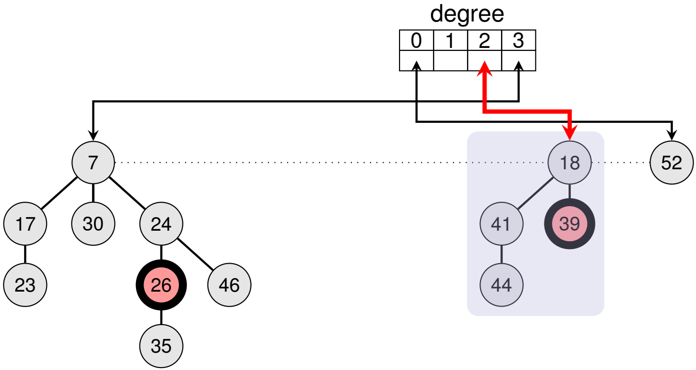
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

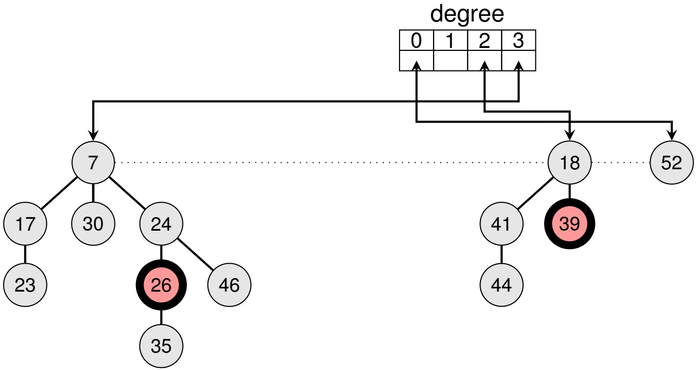
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children)



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

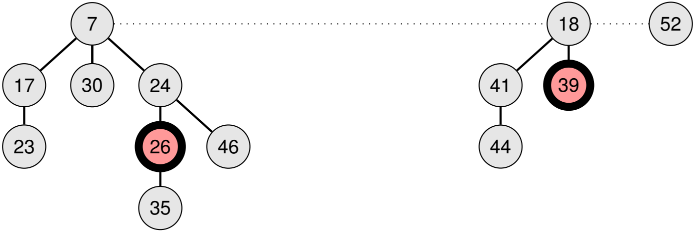
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children) ✓



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

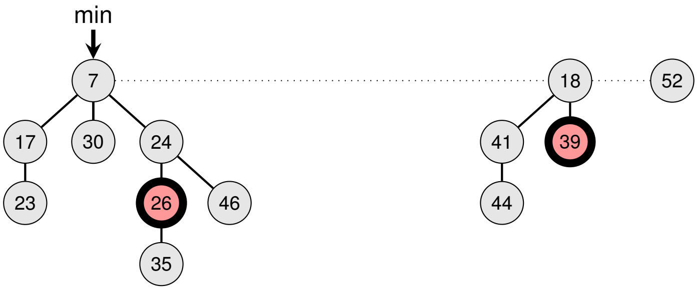
- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children) ✓
- Update minimum



Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children) ✓
- Update minimum ✓

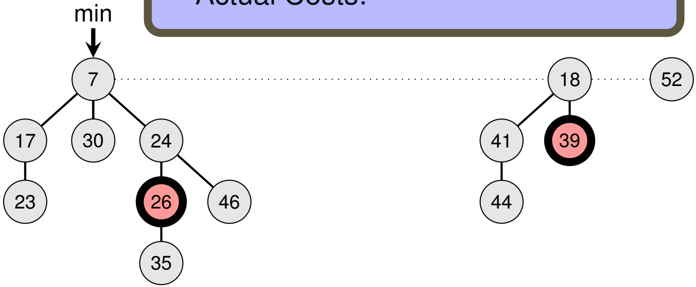


Fibonacci Heap: EXTRACT-MIN

EXTRACT-MIN

- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children) ✓
- Update minimum ✓

Actual Costs:



Fibonacci Heap: EXTRACT-MIN

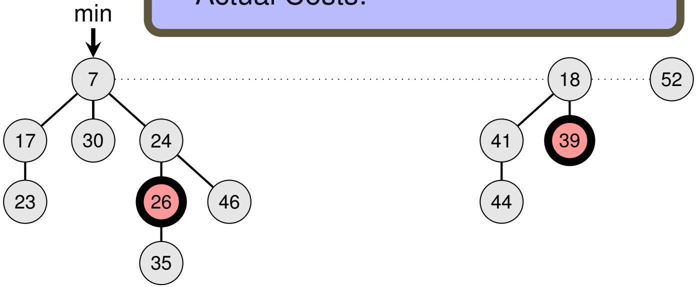
EXTRACT-MIN

- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children) ✓
- Update minimum ✓

Every root becomes child of another root at most once!

$d(n)$ is the maximum degree of a root in any Fibonacci heap of size n

Actual Costs:



Fibonacci Heap: EXTRACT-MIN

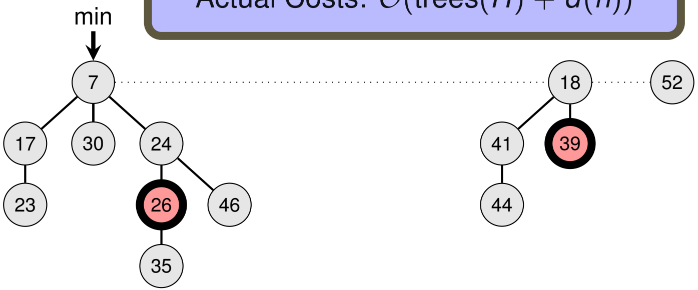
EXTRACT-MIN

- Delete min ✓
- Meld childen into root list and unmark them ✓
- Consolidate so that no roots have the same degree (# children) ✓
- Update minimum ✓

Every root becomes child of another root at most once!

$d(n)$ is the maximum degree of a root in any Fibonacci heap of size n

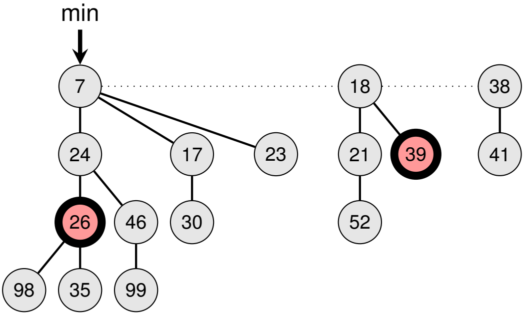
Actual Costs: $\mathcal{O}(\text{trees}(H) + d(n))$



Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

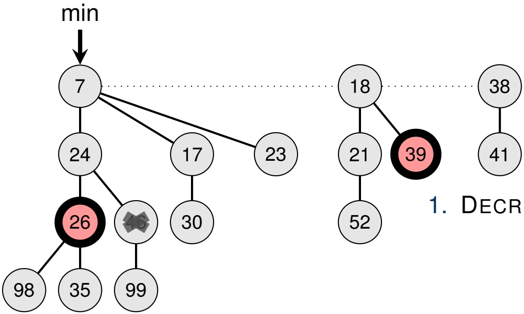
- Decrease the key of x (given by a pointer)



Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

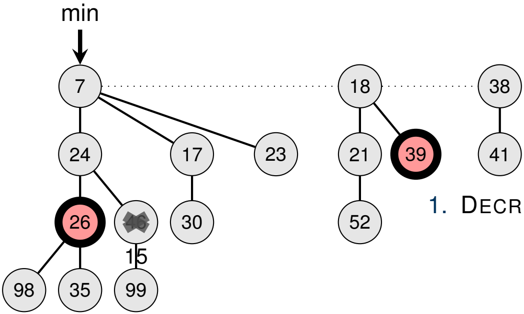


1. DECREASE-KEY 46 $\rightsquigarrow$ 15

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list



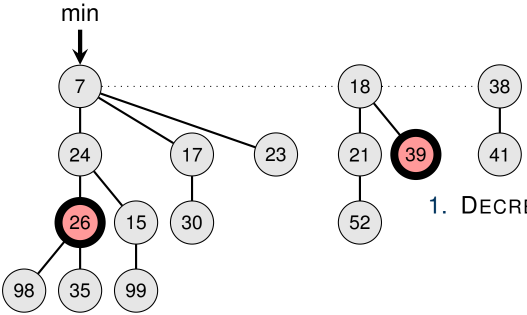
1. DECREASE-KEY 46 $\rightsquigarrow$ 15

Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node *x*

- Decrease the key of *x* (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at *x*, unmark *x*, meld into root list



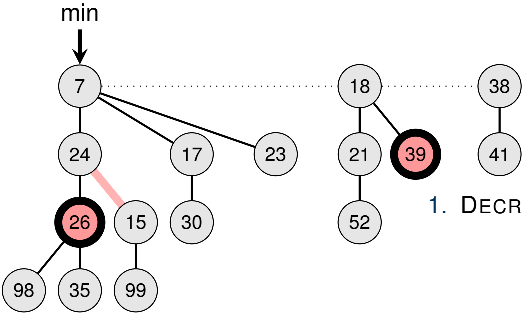
1. DECREASE-KEY 46 $\rightsquigarrow$ 15

Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list



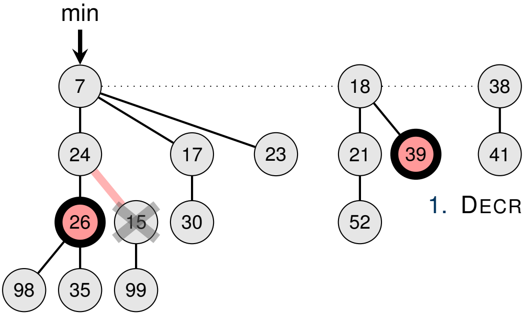
1. DECREASE-KEY 46 $\rightsquigarrow$ 15

Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list



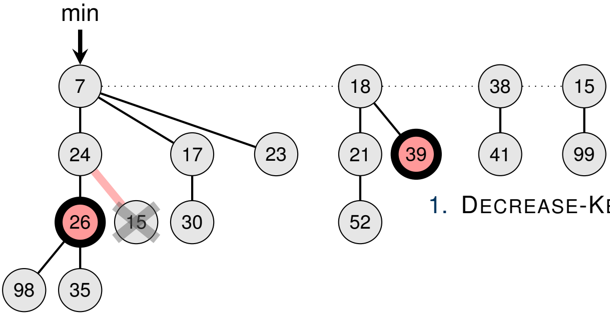
1. DECREASE-KEY 46 $\rightsquigarrow$ 15

Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:



1. DECREASE-KEY 46 $\rightsquigarrow$ 15

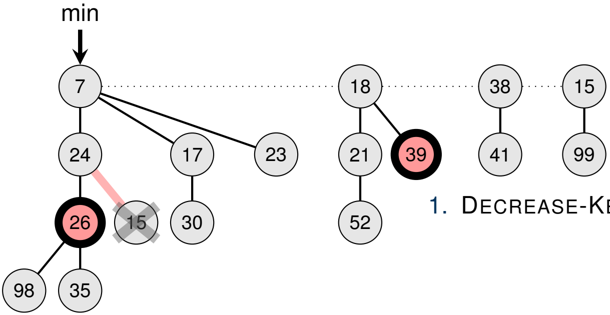
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked



1. DECREASE-KEY 46 $\rightsquigarrow$ 15

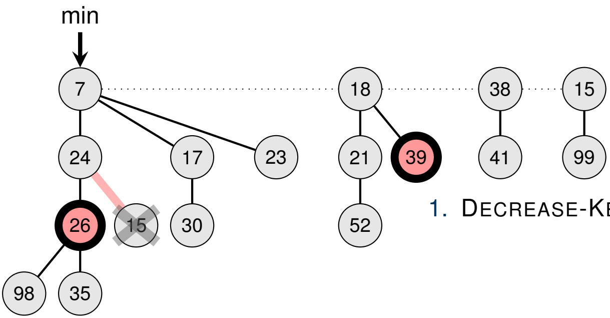
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15

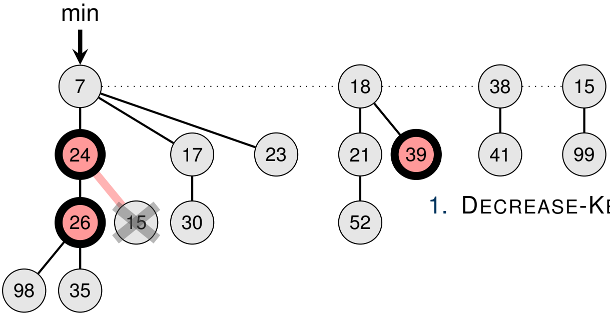
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15

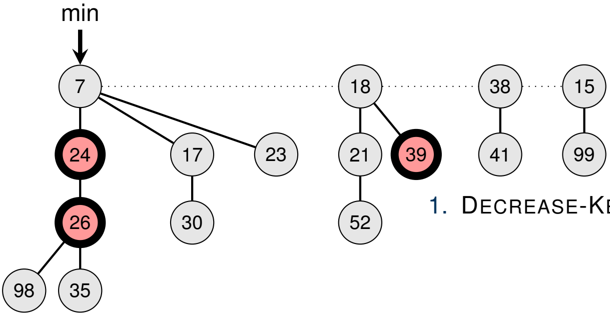
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓

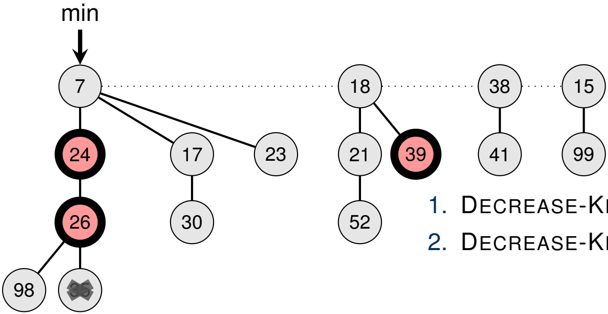
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

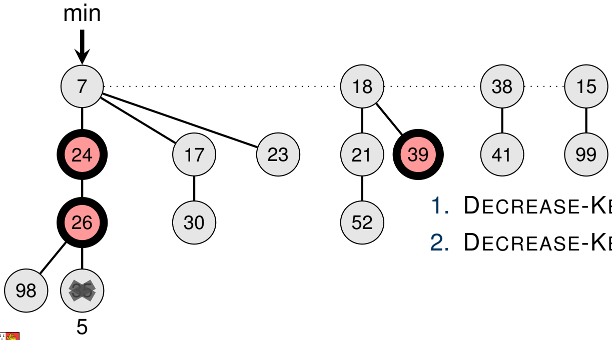
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

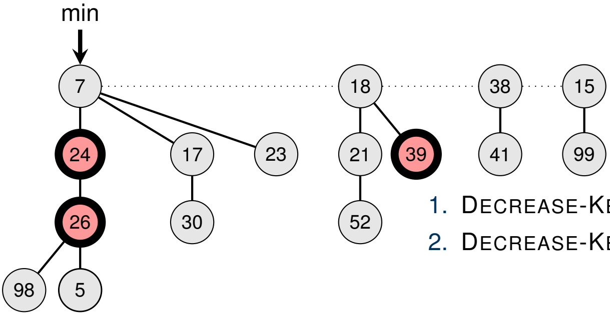
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

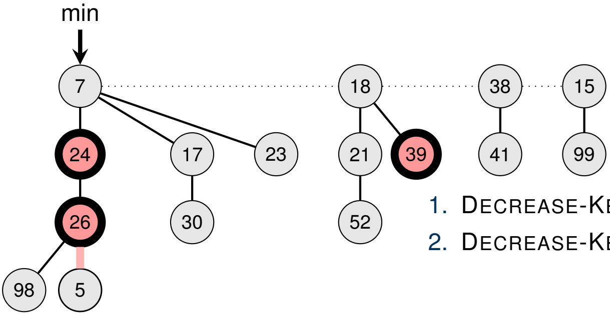
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

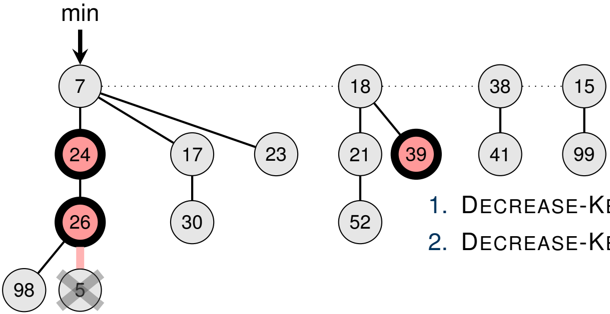
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

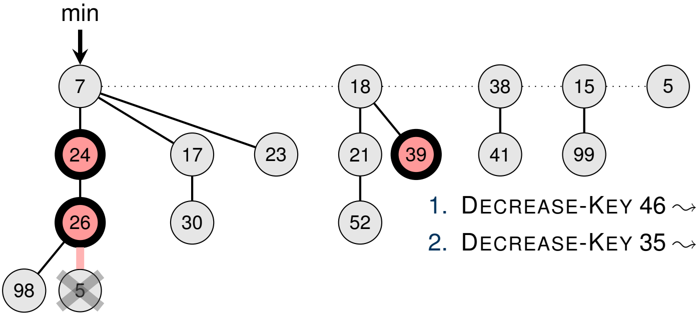
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)



1. DECREASE-KEY $46 \rightsquigarrow 15$ ✓
2. DECREASE-KEY $35 \rightsquigarrow 5$

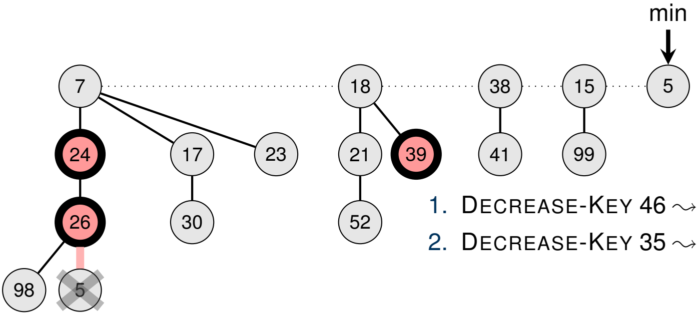
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node *x*

- Decrease the key of *x* (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at *x*, unmark *x*, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked,



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

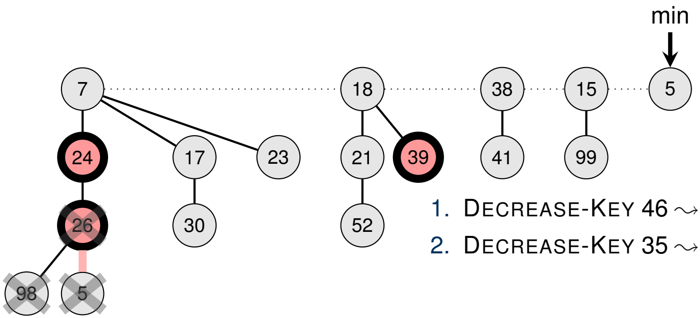
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

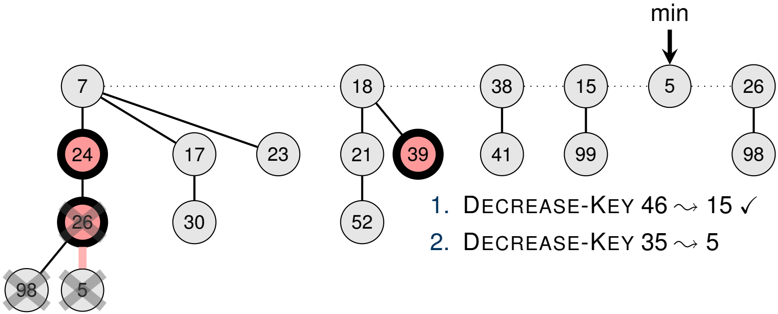
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

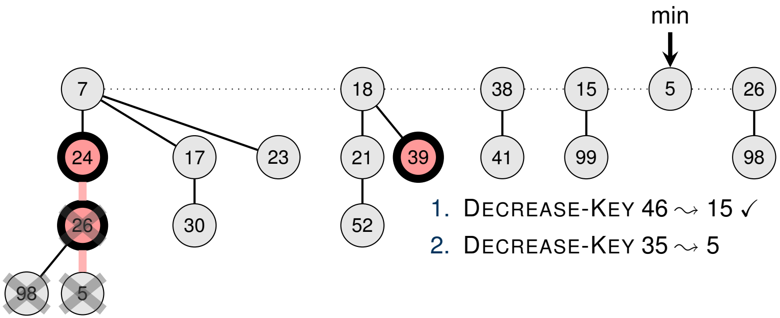
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

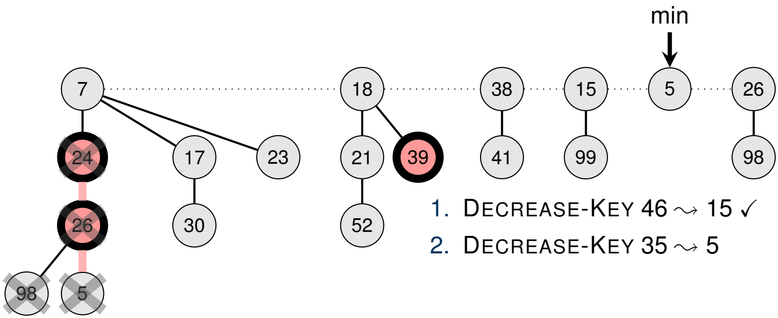
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

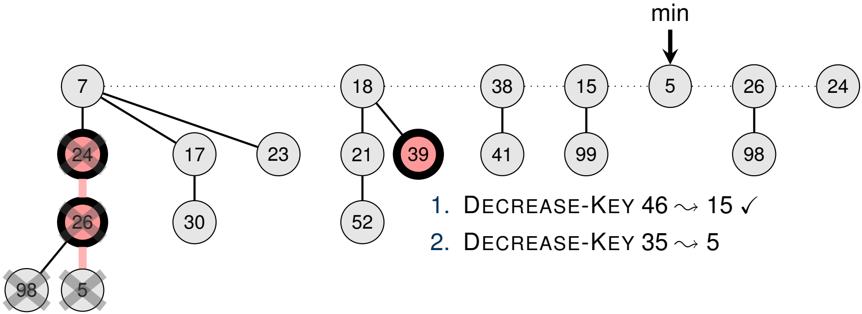
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



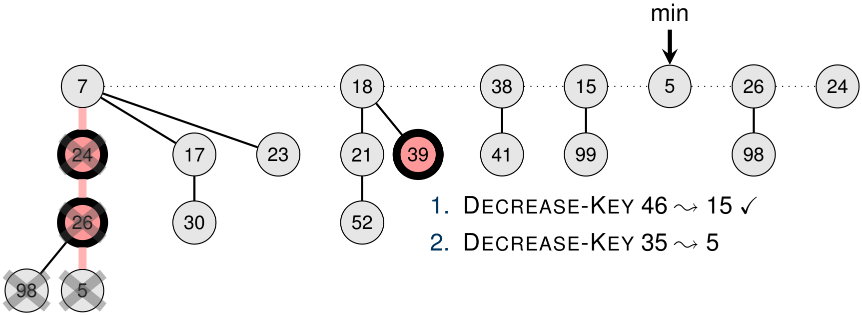
1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

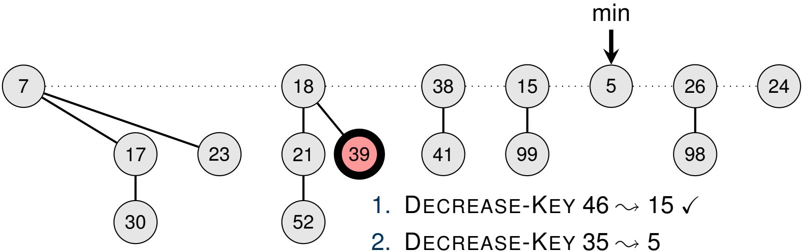
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5

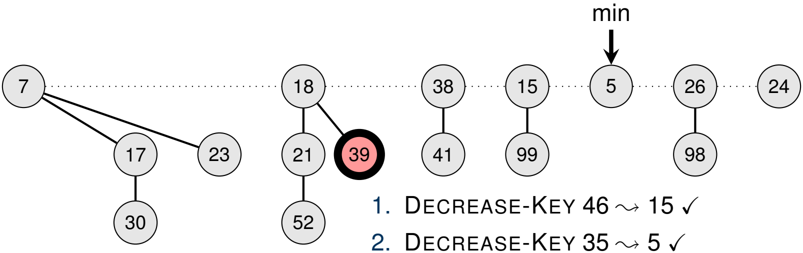
Fibonacci Heap: DECREASE-KEY

DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5 ✓

Fibonacci Heap: DECREASE-KEY

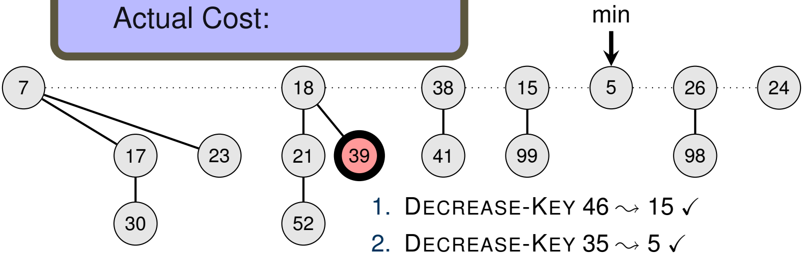
DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)

Actual Cost:



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5 ✓

Fibonacci Heap: DECREASE-KEY

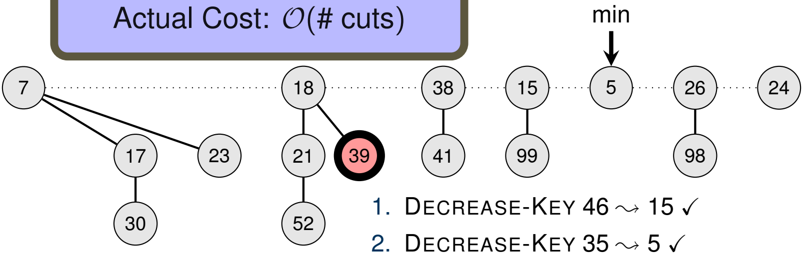
DECREASE-KEY of node x

- Decrease the key of x (given by a pointer)
- (Here we consider only cases where heap-order is violated)

$\Rightarrow$ Cut tree rooted at x, unmark x, meld into root list and:

- Check if parent node is marked
  - If unmarked, mark it (unless it is a root)
  - If marked, unmark and meld it into root list and recurse (Cascading Cut)

Actual Cost: $\mathcal{O}(\text{\# cuts})$



1. DECREASE-KEY 46 $\rightsquigarrow$ 15 ✓
2. DECREASE-KEY 35 $\rightsquigarrow$ 5 ✓

Outline

Recap of INSERT, EXTRACT-MIN and DECREASE-KEY

Glimpse at the Analysis

Amortized Analysis

Bounding the Maximum Degree

Amortized Analysis via Potential Method

- INSERT: actual $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$
- DECREASE-KEY: actual $\mathcal{O}(\text{\# cuts}) \leq \mathcal{O}(\text{marks}(H))$

Amortized Analysis via Potential Method

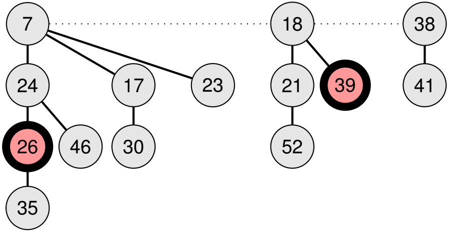
- INSERT: actual $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$
- DECREASE-KEY: actual $\mathcal{O}(\#\text{ cuts}) \leq \mathcal{O}(\text{marks}(H))$

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

Amortized Analysis via Potential Method

- INSERT: actual $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$
- DECREASE-KEY: actual $\mathcal{O}(\#\text{ cuts}) \leq \mathcal{O}(\text{marks}(H))$

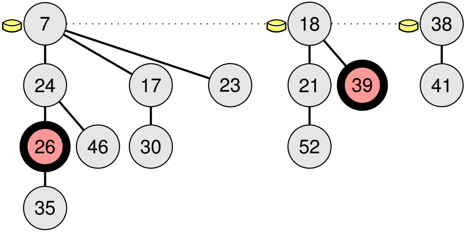
$$\boxed{\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)}$$



Amortized Analysis via Potential Method

- INSERT: actual $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$
- DECREASE-KEY: actual $\mathcal{O}(\#\text{ cuts}) \leq \mathcal{O}(\text{marks}(H))$

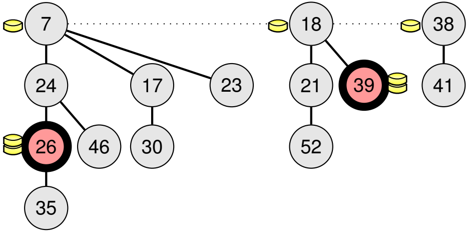
$$\boxed{\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)}$$



Amortized Analysis via Potential Method

- INSERT: actual $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$
- DECREASE-KEY: actual $\mathcal{O}(\#\text{ cuts}) \leq \mathcal{O}(\text{marks}(H))$

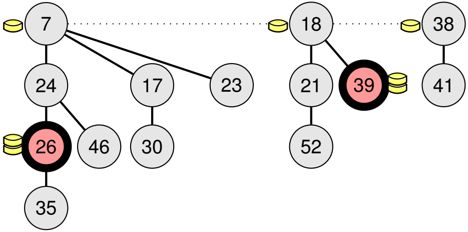
$$\Phi(H) = \text{trees}(H) + 2 \cdot \text{marks}(H)$$



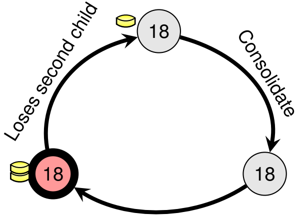
Amortized Analysis via Potential Method

- INSERT: actual $\mathcal{O}(1)$ amortized $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$ amortized $\mathcal{O}(d(n))$
- DECREASE-KEY: actual $\mathcal{O}(\#\text{ cuts}) \leq \mathcal{O}(\text{marks}(H))$ amortized $\mathcal{O}(1)$

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



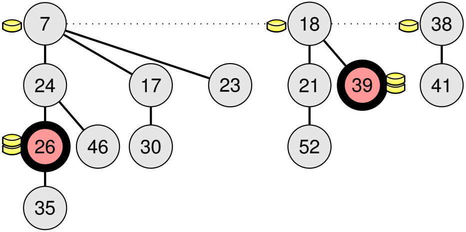
Lifecycle of a node



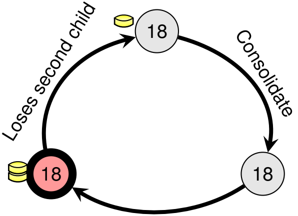
Amortized Analysis via Potential Method

- INSERT: actual $\mathcal{O}(1)$ amortized $\mathcal{O}(1)$ ✓
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$ amortized $\mathcal{O}(d(n))$?
- DECREASE-KEY: actual $\mathcal{O}(\#\text{ cuts}) \leq \mathcal{O}(\text{marks}(H))$ amortized $\mathcal{O}(1)$?

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



Lifecycle of a node



Outline

Recap of INSERT, EXTRACT-MIN and DECREASE-KEY

Glimpse at the Analysis

Amortized Analysis

Bounding the Maximum Degree

Amortized Analysis of DECREASE-KEY

Actual Cost

- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

Amortized Analysis of DECREASE-KEY

Actual Cost

- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

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

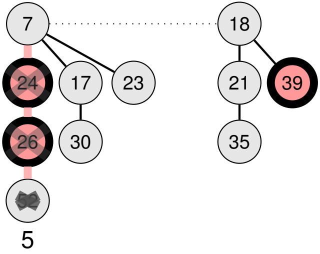
Amortized Analysis of DECREASE-KEY

Actual Cost

- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

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

Change in Potential



Amortized Analysis of DECREASE-KEY

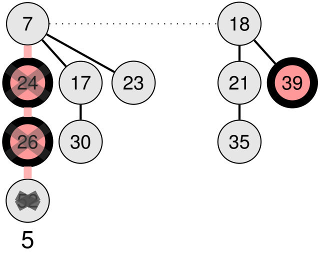
Actual Cost

- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

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

Change in Potential

- $\text{trees}(H') =$



Amortized Analysis of DECREASE-KEY

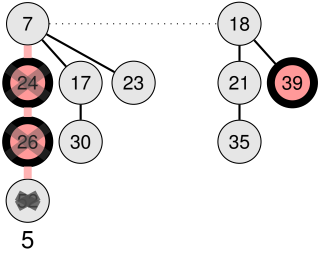
Actual Cost

- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

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

Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$



Amortized Analysis of DECREASE-KEY

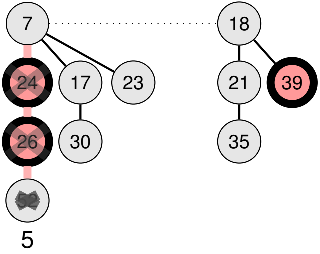
Actual Cost

- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

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

Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq$



Amortized Analysis of DECREASE-KEY

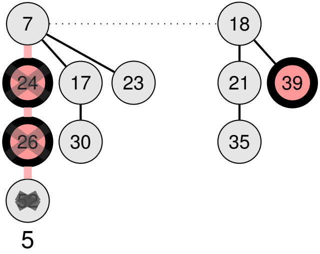
Actual Cost

- DECREASE-KEY: $\mathcal{O}(x + 1)$, where x is the number of cuts.

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

Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq \text{marks}(H) - x + 2$



Amortized Analysis of DECREASE-KEY

Actual Cost

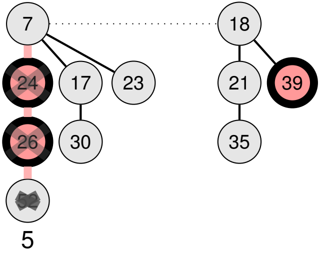
- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

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

Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq \text{marks}(H) - x + 2$

$\Rightarrow \Delta\Phi \leq x + 2 \cdot (-x + 2) = 4 - x.$



Amortized Analysis of DECREASE-KEY

Actual Cost

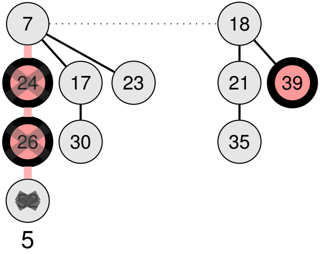
- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

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

Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq \text{marks}(H) - x + 2$

$\Rightarrow \Delta\Phi \leq x + 2 \cdot (-x+2) = 4 - x.$



Amortized Cost

$$\widetilde{c}_i = c_i + \Delta\Phi$$

Amortized Analysis of DECREASE-KEY

Actual Cost

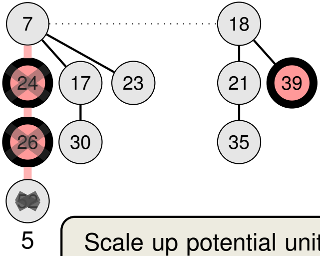
- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

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

Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq \text{marks}(H) - x + 2$

$\Rightarrow \Delta\Phi \leq x + 2 \cdot (-x + 2) = 4 - x.$



Scale up potential units

Amortized Cost

$$\widetilde{c}_i = c_i + \Delta\Phi \leq \mathcal{O}(x+1) + 4 - x$$

Amortized Analysis of DECREASE-KEY

Actual Cost

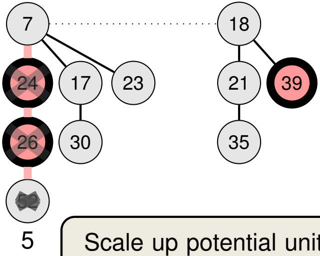
- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

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

Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq \text{marks}(H) - x + 2$

$\Rightarrow \Delta\Phi \leq x + 2 \cdot (-x + 2) = 4 - x.$



Scale up potential units

Amortized Cost

$$\widetilde{c}_i = c_i + \Delta\Phi \leq \mathcal{O}(x+1) + 4 - x = \mathcal{O}(1)$$

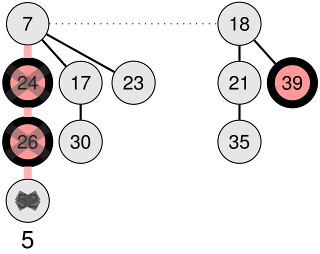
Amortized Analysis of DECREASE-KEY

Actual Cost

- DECREASE-KEY: $\mathcal{O}(x+1)$, where x is the number of cuts.

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

First Coin $\rightsquigarrow$ pays cut
Second Coin $\rightsquigarrow$ increase of trees(H)



Change in Potential

- $\text{trees}(H') = \text{trees}(H) + x$
- $\text{marks}(H') \leq \text{marks}(H) - x + 2$

$\Rightarrow \Delta\Phi \leq x + 2 \cdot (-x + 2) = 4 - x.$

Amortized Cost

$$\widetilde{c}_i = c_i + \Delta\Phi \leq \mathcal{O}(x+1) + 4 - x = \mathcal{O}(1)$$

Amortized Analysis of EXTRACT-MIN

Actual Cost

- EXTRACT-MIN: $\mathcal{O}(\text{trees}(H) + d(n))$

Amortized Analysis of EXTRACT-MIN

Actual Cost

- EXTRACT-MIN: $\mathcal{O}(\text{trees}(H) + d(n))$

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

Amortized Analysis of EXTRACT-MIN

Actual Cost

- EXTRACT-MIN: $\mathcal{O}(\text{trees}(H) + d(n))$

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

Change in Potential

Amortized Analysis of Extract-Min

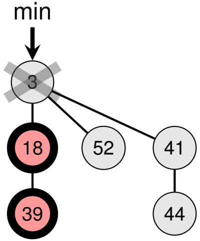
Actual Cost

- Extract-Min: $\mathcal{O}(\text{trees}(H) + d(n))$

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

Change in Potential

- $\text{marks}(H') \ ? \ \text{marks}(H)$



Amortized Analysis of EXTRACT-MIN

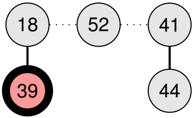
Actual Cost

- EXTRACT-MIN: $\mathcal{O}(\text{trees}(H) + d(n))$

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

Change in Potential

- $\text{marks}(H')$? $\text{marks}(H)$



Amortized Analysis of EXTRACT-MIN

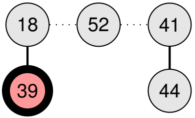
Actual Cost

- EXTRACT-MIN: $\mathcal{O}(\text{trees}(H) + d(n))$

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

Change in Potential

- $\text{marks}(H') \leq \text{marks}(H)$



Amortized Analysis of Extract-Min

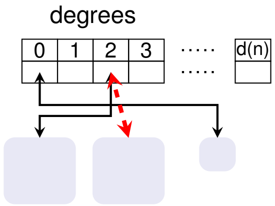
Actual Cost

- Extract-Min: $\mathcal{O}(\text{trees}(H) + d(n))$

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

Change in Potential

- $\text{marks}(H') \leq \text{marks}(H)$
- $\text{trees}(H') \leq$



Amortized Analysis of EXTRACT-MIN

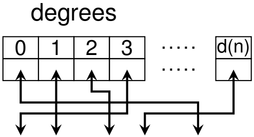
Actual Cost

- EXTRACT-MIN: $\mathcal{O}(\text{trees}(H) + d(n))$

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

Change in Potential

- $\text{marks}(H') \leq \text{marks}(H)$
- $\text{trees}(H') \leq$



Amortized Analysis of EXTRACT-MIN

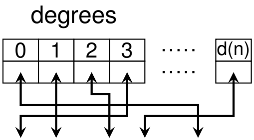
Actual Cost

- EXTRACT-MIN: $\mathcal{O}(\text{trees}(H) + d(n))$

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

Change in Potential

- $\text{marks}(H') \leq \text{marks}(H)$
- $\text{trees}(H') \leq d(n) + 1$



Amortized Analysis of EXTRACT-MIN

Actual Cost

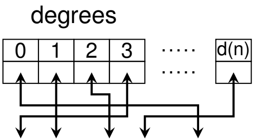
- EXTRACT-MIN: $\mathcal{O}(\text{trees}(H) + d(n))$

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

Change in Potential

- $\text{marks}(H') \leq \text{marks}(H)$
- $\text{trees}(H') \leq \ d(n) + 1$

$\Rightarrow$ $\Delta\Phi \leq d(n) + 1 - \text{trees}(H)$



Amortized Analysis of Extract-Min

Actual Cost

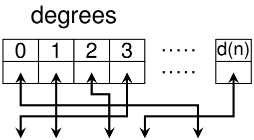
- Extract-Min: $\mathcal{O}(\text{trees}(H) + d(n))$

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

Change in Potential

- $\text{marks}(H') \leq \text{marks}(H)$
- $\text{trees}(H') \leq d(n) + 1$

$\Rightarrow \Delta\Phi \leq d(n) + 1 - \text{trees}(H)$



Amortized Cost

$$\widetilde{c_i} = c_i + \Delta\Phi$$

Amortized Analysis of EXTRACT-MIN

Actual Cost

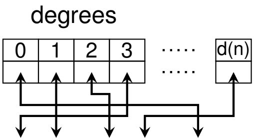
- EXTRACT-MIN: $\mathcal{O}(\text{trees}(H) + d(n))$

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

Change in Potential

- $\text{marks}(H') \leq \text{marks}(H)$
- $\text{trees}(H') \leq d(n) + 1$

$\Rightarrow$ $\Delta\Phi \leq d(n) + 1 - \text{trees}(H)$



Amortized Cost

$$\widetilde{c_i} = c_i + \Delta\Phi \leq \mathcal{O}(\text{trees(H)} + d(n)) + d(n) + 1 - \text{trees(H)}$$

Amortized Analysis of EXTRACT-MIN

Actual Cost

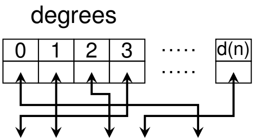
- EXTRACT-MIN: $\mathcal{O}(\text{trees}(H) + d(n))$

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

Change in Potential

- $\text{marks}(H') \le \text{marks}(H)$
- $\text{trees}(H') \le d(n) + 1$

$\Rightarrow \Delta\Phi \le d(n) + 1 - \text{trees}(H)$



Amortized Cost

$$\widetilde{c}_i = c_i + \Delta\Phi \le \mathcal{O}(\text{trees(H)} + d(n)) + d(n) + 1 - \text{trees(H)} = \mathcal{O}(d(n))$$

Amortized Analysis of EXTRACT-MIN

Actual Cost

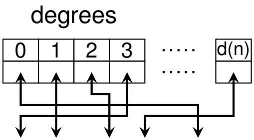
- EXTRACT-MIN: $\mathcal{O}(\text{trees}(H) + d(n))$

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

Change in Potential

- $\text{marks}(H') \leq \text{marks}(H)$
- $\text{trees}(H') \leq d(n) + 1$

$\Rightarrow \Delta\Phi \leq d(n) + 1 - \text{trees}(H)$



Amortized Cost

$$\widetilde{c}_i = c_i + \Delta\Phi \leq \mathcal{O}(\text{trees(H)} + d(n)) + d(n) + 1 - \text{trees(H)} = \mathcal{O}(d(n))$$

How to bound $d(n)$?

Outline

Recap of INSERT, EXTRACT-MIN and DECREASE-KEY

Glimpse at the Analysis

Amortized Analysis

Bounding the Maximum Degree

Bounding the Maximum Degree

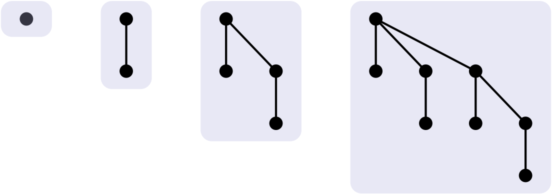
Binomial Heap

Every tree is a binomial tree $\Rightarrow d(n) \leq \log_2 n$.

Bounding the Maximum Degree

Binomial Heap

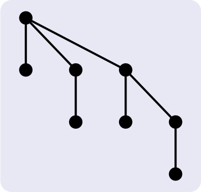
Every tree is a binomial tree $\Rightarrow d(n) \leq \log_2 n$.



Bounding the Maximum Degree

Binomial Heap

Every tree is a binomial tree $\Rightarrow d(n) \leq \log_2 n$.

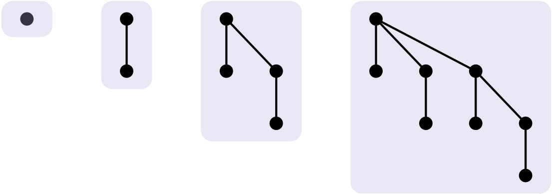


$d = 3, n = 2^3$

Bounding the Maximum Degree

Binomial Heap

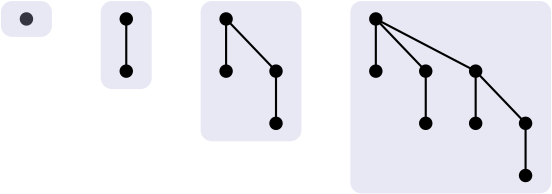
Every tree is a binomial tree $\Rightarrow d(n) \leq \log_2 n$.



Bounding the Maximum Degree

Binomial Heap

Every tree is a binomial tree $\Rightarrow d(n) \leq \log_2 n$.



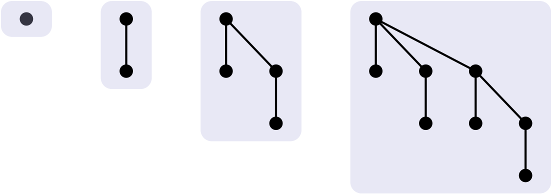
Fibonacci Heap

Not all trees are binomial trees, but still $d(n) \leq \log_\varphi n$, where $\varphi \approx 1.62$.

Bounding the Maximum Degree

Binomial Heap

Every tree is a binomial tree $\Rightarrow d(n) \leq \log_2 n$.



Fibonacci Heap

Not all trees are binomial trees, but still $d(n) \leq \log_\varphi n$, where $\varphi \approx 1.62$.

Skip Analysis

Lower Bounding Degrees of Children

$$d(n) \leq \log_{\varphi} n$$

Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_{\varphi} n$$

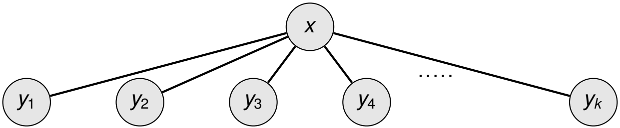
- Consider any node x of degree k (not necessarily a root) at the final state

Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment



Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

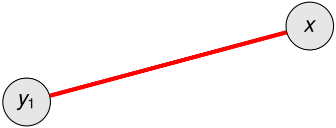
- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_{\varphi} n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

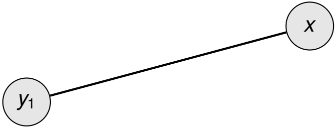


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_{\varphi} n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

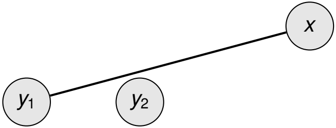


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

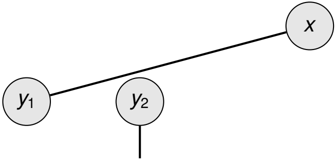


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_{\varphi} n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

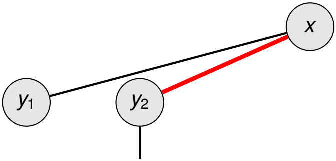


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_{\varphi} n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

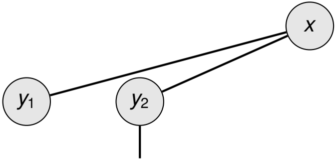


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

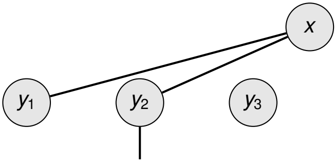


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_{\varphi} n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

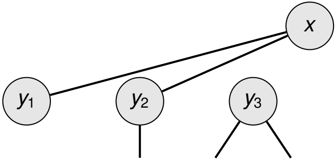


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

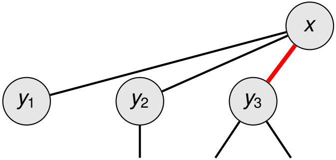


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_{\varphi} n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

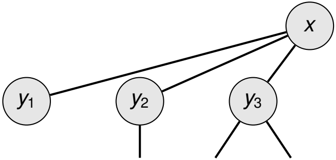


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

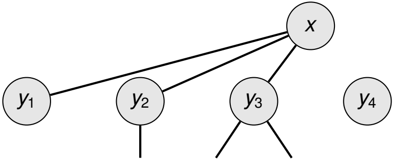


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_{\varphi} n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

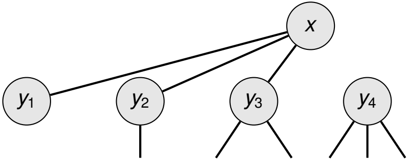


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

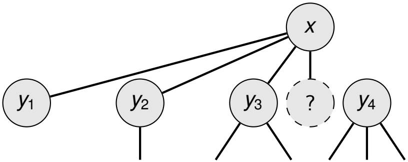


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

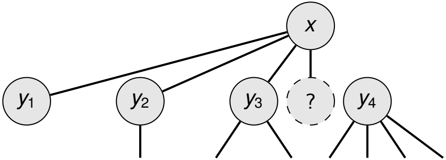


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

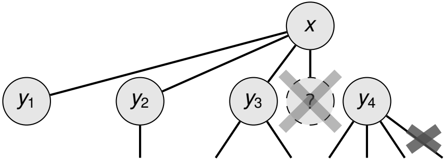


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_{\varphi} n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

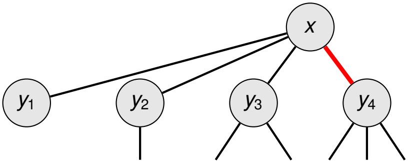


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

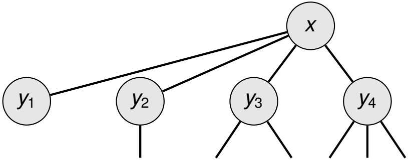


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_{\varphi} n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

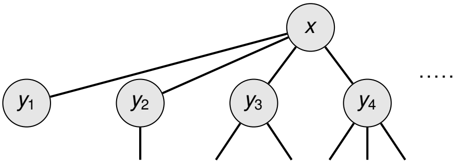


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

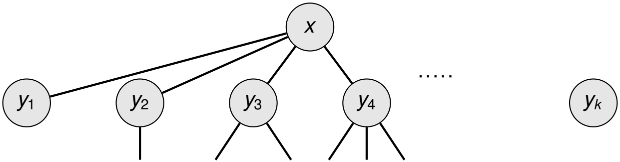


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

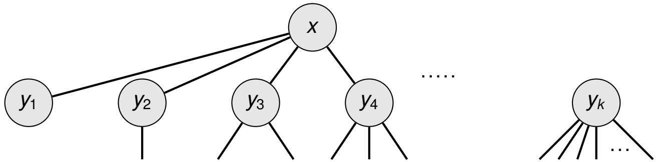


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

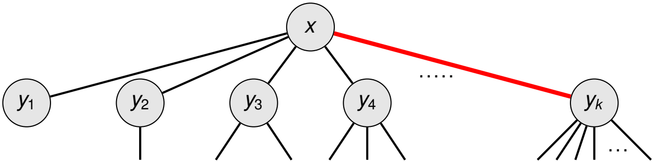


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

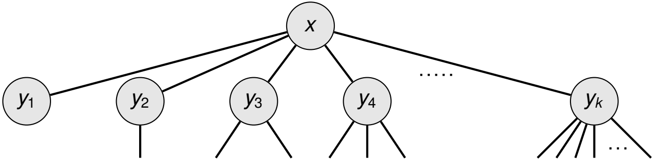


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_{\varphi} n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

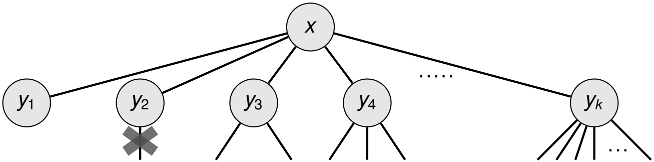


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

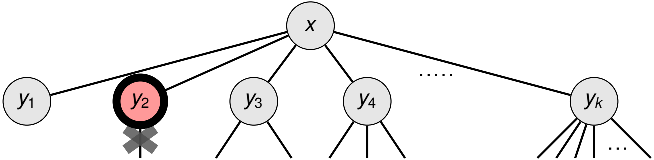


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_{\varphi} n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

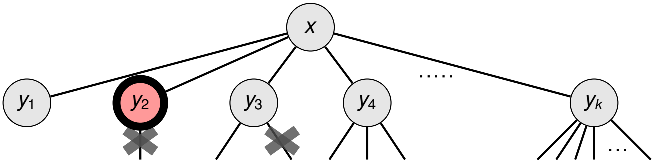


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_{\varphi} n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

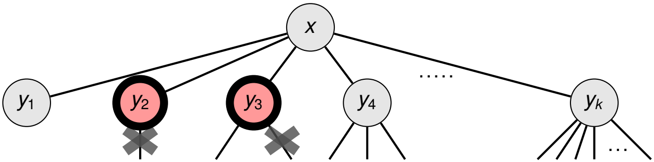


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

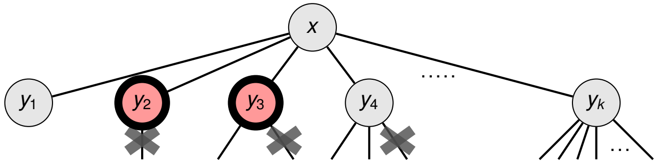


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

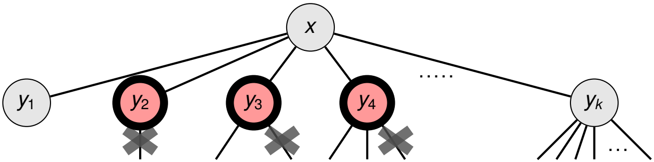


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

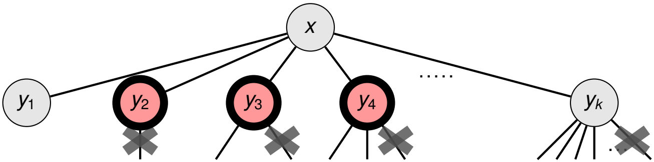


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_{\varphi} n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

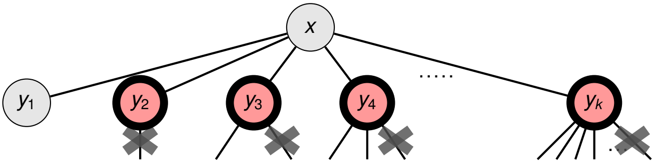


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_{\varphi} n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment

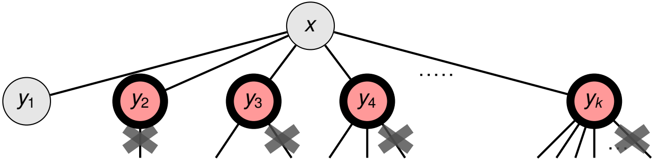


Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

$$d(n) \leq \log_\varphi n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment and $d_1, d_2, \ldots, d_k$ be their degrees



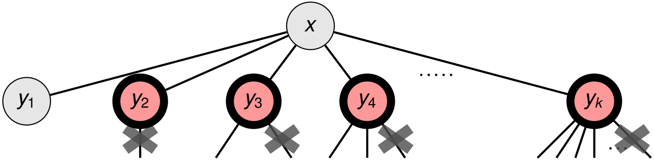
Lower Bounding Degrees of Children

We will prove a stronger statement:
Any tree with degree k contains at least φ^k nodes.

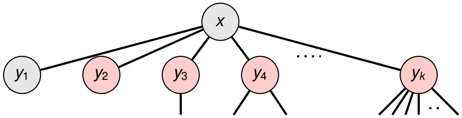
$$d(n) \leq \log_{\varphi} n$$

- Consider any node x of degree k (not necessarily a root) at the final state
- Let $y_1, y_2, \ldots, y_k$ be the children in the order of attachment and $d_1, d_2, \ldots, d_k$ be their degrees

$\Rightarrow$ $\boxed{\forall 1 \leq i \leq k: \quad d_i \geq i - 2}$

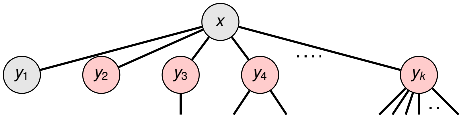


From Degrees to Minimum Subtree Sizes



$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

From Degrees to Minimum Subtree Sizes

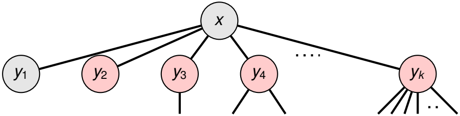


$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

From Degrees to Minimum Subtree Sizes



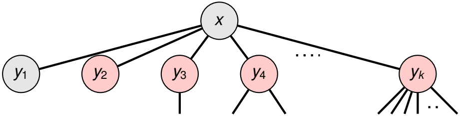
$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$N(0)$

From Degrees to Minimum Subtree Sizes



$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

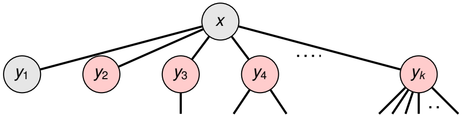
Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$N(0)$

- 0

From Degrees to Minimum Subtree Sizes



$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

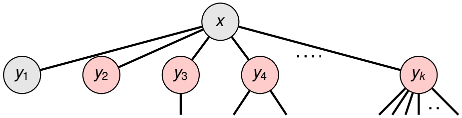
Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$N(0)$ $N(1)$

- 0

From Degrees to Minimum Subtree Sizes



$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

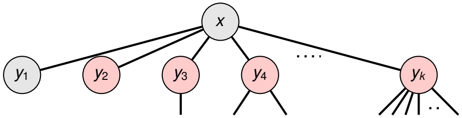
Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$N(0)$ $N(1)$

- 0
- 1

From Degrees to Minimum Subtree Sizes



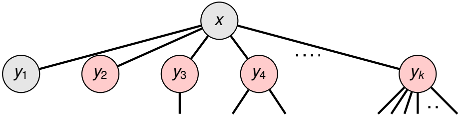
$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$N(0)$ $N(1)$

From Degrees to Minimum Subtree Sizes



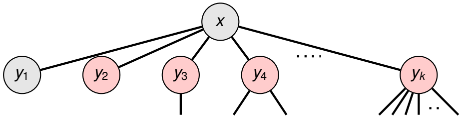
$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$N(0)$ $N(1)$ $N(2)$

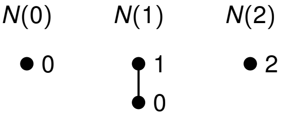
From Degrees to Minimum Subtree Sizes



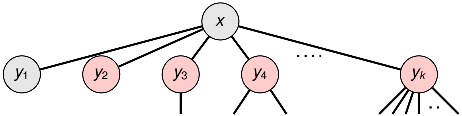
$$\forall 1 \leq i \leq k : \quad d_i \geq i - 2$$

Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.



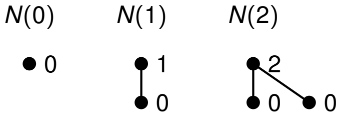
From Degrees to Minimum Subtree Sizes



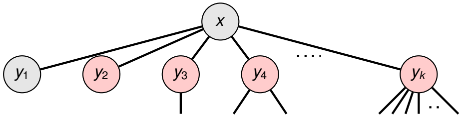
$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.



From Degrees to Minimum Subtree Sizes

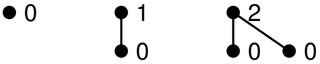


$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

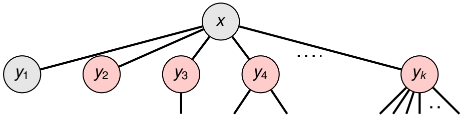
Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$N(0)$ $N(1)$ $N(2)$ $N(3)$



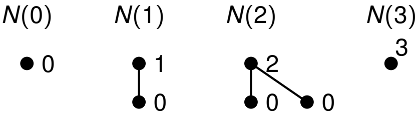
From Degrees to Minimum Subtree Sizes



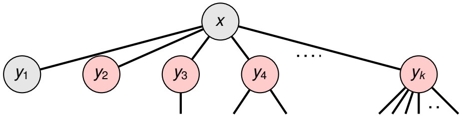
$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.



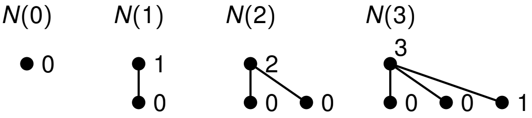
From Degrees to Minimum Subtree Sizes



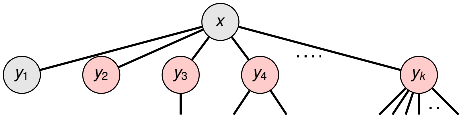
$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.



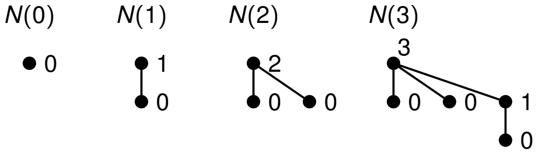
From Degrees to Minimum Subtree Sizes



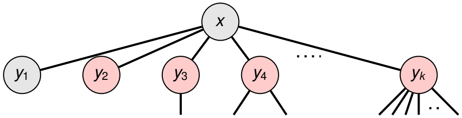
$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.



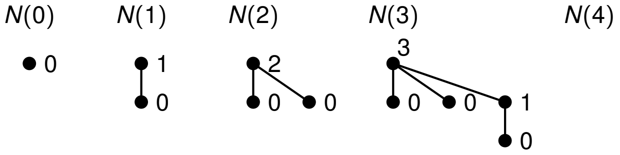
From Degrees to Minimum Subtree Sizes



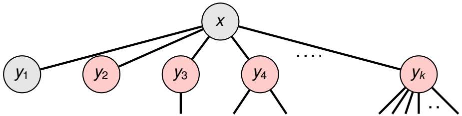
$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.



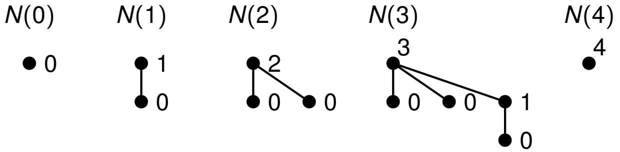
From Degrees to Minimum Subtree Sizes



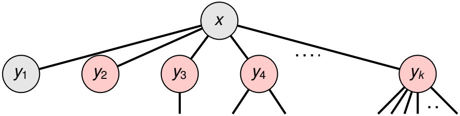
$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.



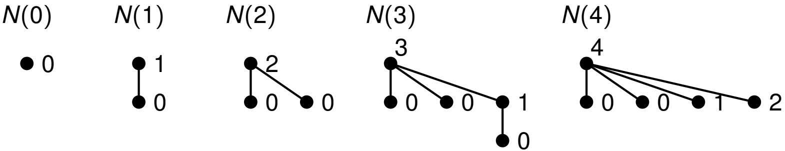
From Degrees to Minimum Subtree Sizes



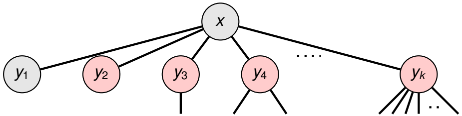
$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.



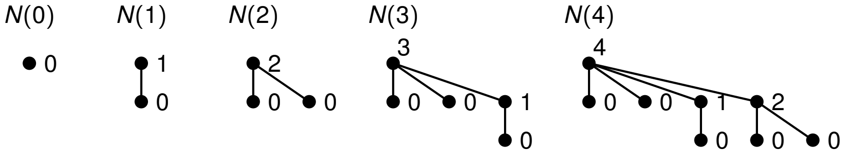
From Degrees to Minimum Subtree Sizes



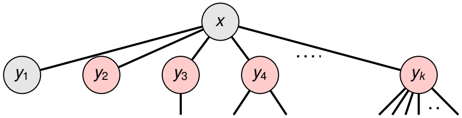
$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.



From Degrees to Minimum Subtree Sizes

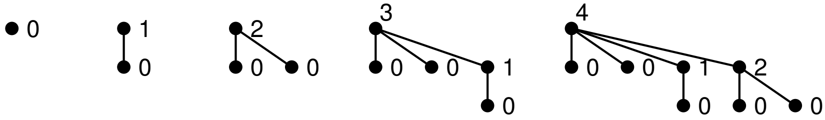


$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

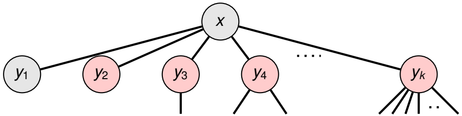
Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$N(0) = 1$ $\quad N(1) \quad N(2) \quad N(3) \quad N(4)$



From Degrees to Minimum Subtree Sizes

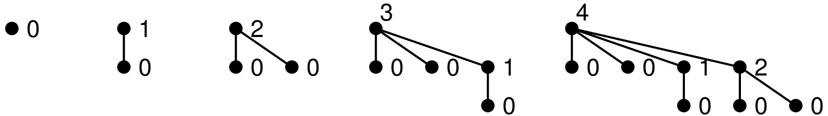


$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

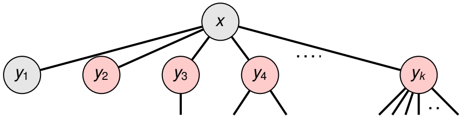
Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$N(0) = 1$ $N(1) = 2$ $N(2)$ $N(3)$ $N(4)$



From Degrees to Minimum Subtree Sizes

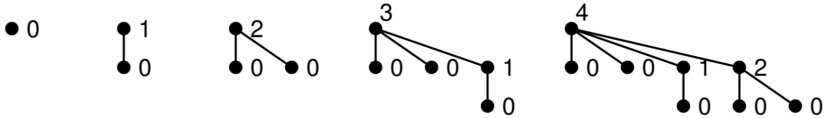


$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

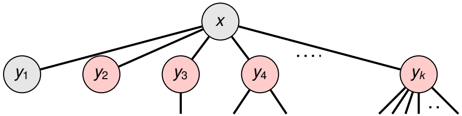
Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$N(0) = 1$ $\quad N(1) = 2$ $\quad N(2) = 3$ $\quad N(3)$ $\quad N(4)$



From Degrees to Minimum Subtree Sizes

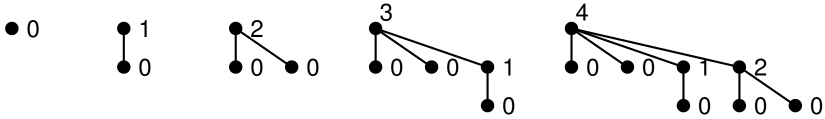


$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

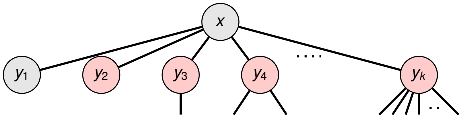
Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$N(0) = 1$ $\quad N(1) = 2$ $\quad N(2) = 3$ $\quad N(3) = 5$ $\quad N(4)$



From Degrees to Minimum Subtree Sizes

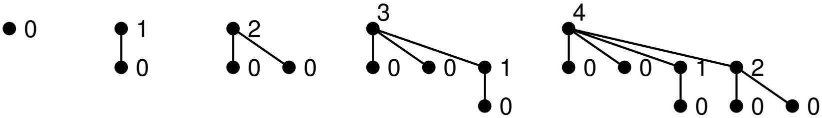


$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

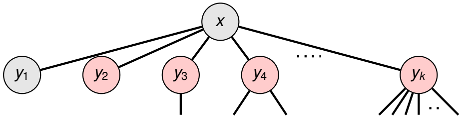
Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$N(0) = 1$ $N(1) = 2$ $N(2) = 3$ $N(3) = 5$ $N(4) = 8$



From Degrees to Minimum Subtree Sizes

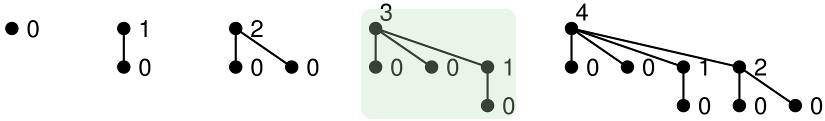


$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

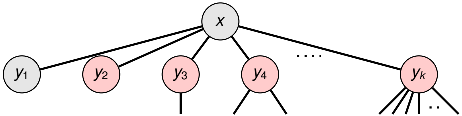
Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$N(0) = 1$ $N(1) = 2$ $N(2) = 3$ $N(3) = 5$ $N(4) = 8$



From Degrees to Minimum Subtree Sizes

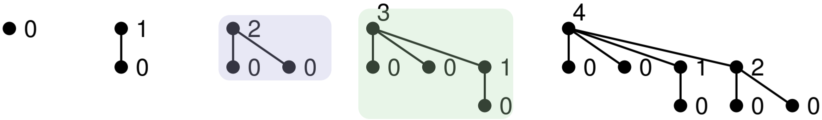


$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

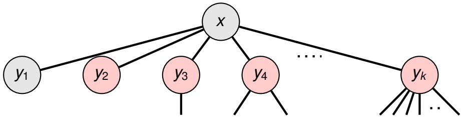
Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$N(0) = 1$ $N(1) = 2$ $N(2) = 3$ $N(3) = 5$ $N(4) = 8$



From Degrees to Minimum Subtree Sizes

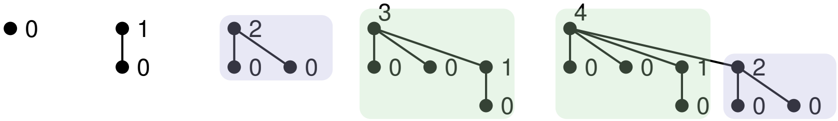


$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

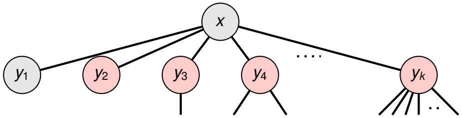
Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$N(0) = 1$ $N(1) = 2$ $N(2) = 3$ $N(3) = 5$ $N(4) = 8$



From Degrees to Minimum Subtree Sizes

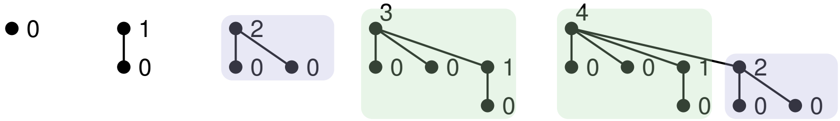


$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

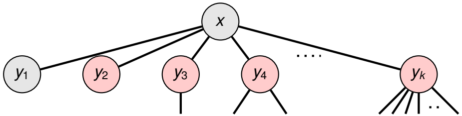
Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$N(0) = 1$ $N(1) = 2$ $N(2) = 3$ $N(3) = 5$ $N(4) = 8 = 5 + 3$



From Degrees to Minimum Subtree Sizes



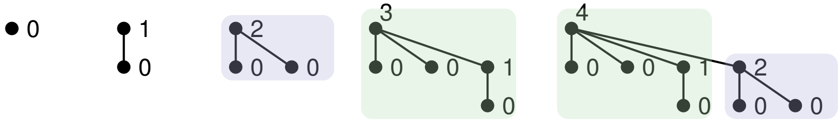
$$\forall 1 \le i \le k: \quad d_i \ge i - 2$$

Definition

Let $N(k)$ be the minimum possible number of nodes of a subtree rooted at a node of degree k.

$$N(k) \;=\; F(k+2)?$$

$N(0) = 1$ $N(1) = 2$ $N(2) = 3$ $N(3) = 5$ $N(4) = 8 = 5 + 3$



From Minimum Subtree Sizes to Fibonacci Numbers

$$\forall 1 \leq i \leq k: \quad d_i \geq i-2$$

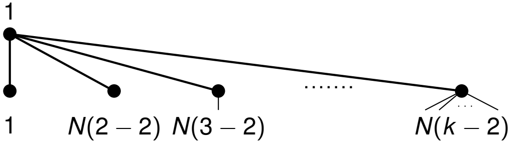
$$N(k) \; = \; F(k+2)?$$

From Minimum Subtree Sizes to Fibonacci Numbers

$$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$$

$$N(k) \;=\; F(k+2)?$$

$N(k) =$

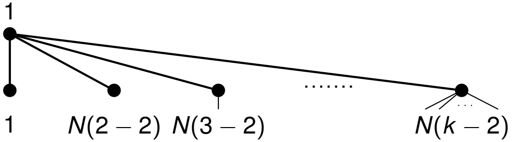


From Minimum Subtree Sizes to Fibonacci Numbers

$\forall 1 \leq i \leq k: \quad d_i \geq i - 2$

$N(k) \ = \ F(k+2)?$

$N(k) =$



$$
\begin{aligned}
N(k) &= 1 + 1 + N(2-2) + N(3-2) + \cdots + N(k-2) \\
&= 1 + 1 + \sum_{\ell=0}^{k-2} N(\ell) \\
&= 1 + 1 + \sum_{\ell=0}^{k-3} N(\ell) + N(k-2) \\
&= N(k-1) + N(k-2) \\
&= F(k+1) + F(k) = F(k+2) \qquad \square
\end{aligned}
$$

Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers $k \geq 0$, the $(k+2)$nd Fib. number satisfies $F(k+2) \geq \varphi^k$, where $\varphi = (1+\sqrt{5})/2 = 1.61803\ldots$.

Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers $k \geq 0$, the $(k+2)$nd Fib. number satisfies $F(k+2) \geq \varphi^k$, where $\varphi = (1+\sqrt{5})/2 = 1.61803\ldots$.

$\varphi^2 = \varphi + 1$

Fibonacci Numbers grow at least exponentially fast in k.

Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers $k \geq 0$, the $(k+2)$nd Fib. number satisfies $F(k+2) \geq \varphi^k$, where $\varphi = (1+\sqrt{5})/2 = 1.61803\ldots$.

$\varphi^2 = \varphi + 1$

Fibonacci Numbers grow at least exponentially fast in k.

Proof by induction on k:

Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers $k \geq 0$, the $(k+2)$nd Fib. number satisfies $F(k+2) \geq \varphi^k$, where $\varphi = (1+\sqrt{5})/2 = 1.61803\ldots$.

$\varphi^2 = \varphi + 1$

Fibonacci Numbers grow at least exponentially fast in k.

Proof by induction on k:

- Base $k = 0$: $F(2) = 1$ and $\varphi^0 = 1$

Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers $k \geq 0$, the $(k+2)$nd Fib. number satisfies $F(k+2) \geq \varphi^k$, where $\varphi = (1+\sqrt{5})/2 = 1.61803\ldots$.

$\varphi^2 = \varphi + 1$

Fibonacci Numbers grow at least exponentially fast in k.

Proof by induction on k:

- Base $k = 0$: $F(2) = 1$ and $\varphi^0 = 1$ ✓

Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers $k \geq 0$, the $(k+2)$nd Fib. number satisfies $F(k+2) \geq \varphi^k$, where $\varphi = (1+\sqrt{5})/2 = 1.61803\ldots$.

$\varphi^2 = \varphi + 1$

Fibonacci Numbers grow at least exponentially fast in k.

Proof by induction on k:

- Base $k = 0$: $F(2) = 1$ and $\varphi^0 = 1$ ✓
- Base $k = 1$: $F(3) = 2$ and $\varphi^1 \approx 1.619 < 2$

Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers $k \geq 0$, the $(k+2)$nd Fib. number satisfies $F(k+2) \geq \varphi^k$, where $\varphi = (1+\sqrt{5})/2 = 1.61803\ldots$.

$\varphi^2 = \varphi + 1$

Fibonacci Numbers grow at least exponentially fast in k.

Proof by induction on k:

- Base $k = 0$: $F(2) = 1$ and $\varphi^0 = 1$ ✓
- Base $k = 1$: $F(3) = 2$ and $\varphi^1 \approx 1.619 < 2$ ✓

Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers $k \geq 0$, the $(k+2)$nd Fib. number satisfies $F(k+2) \geq \varphi^k$, where $\varphi = (1+\sqrt{5})/2 = 1.61803\ldots$.

$\varphi^2 = \varphi + 1$

Fibonacci Numbers grow at least exponentially fast in k.

Proof by induction on k:

- Base $k = 0$: $F(2) = 1$ and $\varphi^0 = 1$ ✓
- Base $k = 1$: $F(3) = 2$ and $\varphi^1 \approx 1.619 < 2$ ✓
- Inductive Step ($k \geq 2$):

$$F(k+2) =$$

Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers $k \geq 0$, the $(k+2)$nd Fib. number satisfies $F(k+2) \geq \varphi^k$, where $\varphi = (1+\sqrt{5})/2 = 1.61803\ldots$.

$\varphi^2 = \varphi + 1$

Fibonacci Numbers grow at least exponentially fast in k.

Proof by induction on k:

- Base $k = 0$: $F(2) = 1$ and $\varphi^0 = 1$ ✓
- Base $k = 1$: $F(3) = 2$ and $\varphi^1 \approx 1.619 < 2$ ✓
- Inductive Step $(k \geq 2)$:

$$F(k+2) = F(k+1) + F(k)$$

Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers $k \geq 0$, the $(k+2)$nd Fib. number satisfies $F(k+2) \geq \varphi^k$, where $\varphi = (1+\sqrt{5})/2 = 1.61803\ldots$.

$\varphi^2 = \varphi + 1$

Fibonacci Numbers grow at least exponentially fast in k.

Proof by induction on k:

- Base $k = 0$: $F(2) = 1$ and $\varphi^0 = 1$ ✓
- Base $k = 1$: $F(3) = 2$ and $\varphi^1 \approx 1.619 < 2$ ✓
- Inductive Step $(k \geq 2)$:

$$\begin{aligned} F(k+2) &= F(k+1) + F(k) \\ &\geq \varphi^{k-1} + \varphi^{k-2} \qquad \text{(by the inductive hypothesis)} \end{aligned}$$

Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers $k \geq 0$, the $(k+2)$nd Fib. number satisfies $F(k+2) \geq \varphi^k$, where $\varphi = (1+\sqrt{5})/2 = 1.61803\ldots$.

$\varphi^2 = \varphi + 1$

Fibonacci Numbers grow at least exponentially fast in k.

Proof by induction on k:

- Base $k = 0$: $F(2) = 1$ and $\varphi^0 = 1$ ✓
- Base $k = 1$: $F(3) = 2$ and $\varphi^1 \approx 1.619 < 2$ ✓
- Inductive Step $(k \geq 2)$:

$$\begin{aligned} F(k+2) &= F(k+1) + F(k) \\ &\geq \varphi^{k-1} + \varphi^{k-2} && \text{(by the inductive hypothesis)} \\ &= \varphi^{k-2} \cdot (\varphi + 1) \end{aligned}$$

Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers $k \geq 0$, the $(k+2)$nd Fib. number satisfies $F(k+2) \geq \varphi^k$, where $\varphi = (1+\sqrt{5})/2 = 1.61803\ldots$.

$\varphi^2 = \varphi + 1$

Fibonacci Numbers grow at least exponentially fast in k.

Proof by induction on k:

- Base $k = 0$: $F(2) = 1$ and $\varphi^0 = 1$ ✓
- Base $k = 1$: $F(3) = 2$ and $\varphi^1 \approx 1.619 < 2$ ✓
- Inductive Step $(k \geq 2)$:

$$\begin{aligned}
F(k+2) &= F(k+1) + F(k) \\
&\geq \varphi^{k-1} + \varphi^{k-2} && \text{(by the inductive hypothesis)} \\
&= \varphi^{k-2} \cdot (\varphi + 1) \\
&= \varphi^{k-2} \cdot \varphi^2 && (\varphi^2 = \varphi + 1)
\end{aligned}$$

Exponential Growth of Fibonacci Numbers

Lemma 19.3

For all integers $k \geq 0$, the $(k+2)$nd Fib. number satisfies $F(k+2) \geq \varphi^k$, where $\varphi = (1+\sqrt{5})/2 = 1.61803\ldots$.

$\varphi^2 = \varphi + 1$

Fibonacci Numbers grow at least exponentially fast in k.

Proof by induction on k:

- Base $k = 0$: $F(2) = 1$ and $\varphi^0 = 1$ ✓
- Base $k = 1$: $F(3) = 2$ and $\varphi^1 \approx 1.619 < 2$ ✓
- Inductive Step ($k \geq 2$):

$$\begin{aligned}
F(k+2) &= F(k+1) + F(k) \\
&\geq \varphi^{k-1} + \varphi^{k-2} && \text{(by the inductive hypothesis)} \\
&= \varphi^{k-2} \cdot (\varphi + 1) \\
&= \varphi^{k-2} \cdot \varphi^2 && (\varphi^2 = \varphi + 1) \\
&= \varphi^k && \square
\end{aligned}$$

Putting the Pieces Together

Amortized Analysis

- INSERT: amortized cost $\mathcal{O}(1)$
- EXTRACT-MIN: amortized cost $\mathcal{O}(d(n))$
- DECREASE-KEY: amortized cost $\mathcal{O}(1)$

Putting the Pieces Together

Amortized Analysis

- INSERT: amortized cost $\mathcal{O}(1)$
- EXTRACT-MIN: amortized cost $\mathcal{O}(d(n))$
- DECREASE-KEY: amortized cost $\mathcal{O}(1)$

$N(k)$

Putting the Pieces Together

Amortized Analysis

- INSERT: amortized cost $\mathcal{O}(1)$
- EXTRACT-MIN: amortized cost $\mathcal{O}(d(n))$
- DECREASE-KEY: amortized cost $\mathcal{O}(1)$

$$N(k) = F(k+2)$$

Putting the Pieces Together

Amortized Analysis

- INSERT: amortized cost $\mathcal{O}(1)$
- EXTRACT-MIN: amortized cost $\mathcal{O}(d(n))$
- DECREASE-KEY: amortized cost $\mathcal{O}(1)$

$$N(k) = F(k+2) \geq \varphi^k$$

Putting the Pieces Together

Amortized Analysis

- INSERT: amortized cost $\mathcal{O}(1)$
- EXTRACT-MIN: amortized cost $\mathcal{O}(d(n))$
- DECREASE-KEY: amortized cost $\mathcal{O}(1)$

$$n \geq N(k) = F(k+2) \geq \varphi^k$$

Putting the Pieces Together

Amortized Analysis

- INSERT: amortized cost $\mathcal{O}(1)$
- EXTRACT-MIN: amortized cost $\mathcal{O}(d(n))$
- DECREASE-KEY: amortized cost $\mathcal{O}(1)$

$$
\begin{aligned}
& n \geq N(k) = F(k+2) \geq \varphi^k \\
\Rightarrow \quad & \log_\varphi n \geq k
\end{aligned}
$$

Putting the Pieces Together

Amortized Analysis

- INSERT: amortized cost $\mathcal{O}(1)$
- EXTRACT-MIN: amortized cost $\cancel{\mathcal{O}(d(n))}$ $\mathcal{O}(\log n)$
- DECREASE-KEY: amortized cost $\mathcal{O}(1)$

$$
\begin{aligned}
& n \geq N(k) = F(k+2) \geq \varphi^k \\
\Rightarrow \quad & \log_\varphi n \geq k
\end{aligned}
$$

What if we don't have marked nodes?

- INSERT: actual $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$
- DECREASE-KEY: actual $\mathcal{O}(1)$

What if we don't have marked nodes?

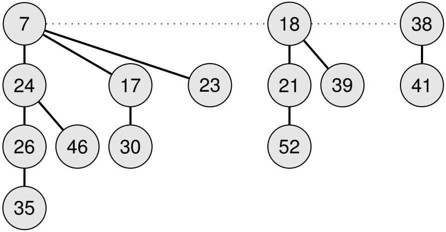
- INSERT: actual $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$
- DECREASE-KEY: actual $\mathcal{O}(1)$

$$\Phi(H) = \text{trees}(H)$$

What if we don’t have marked nodes?

- INSERT: actual $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$
- DECREASE-KEY: actual $\mathcal{O}(1)$

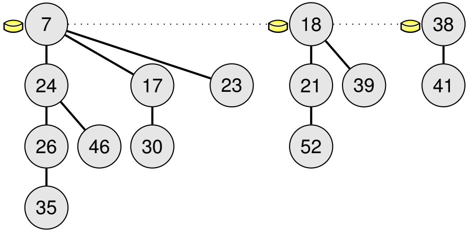
$$\Phi(H) = \text{trees}(H)$$



What if we don't have marked nodes?

- INSERT: actual $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$
- DECREASE-KEY: actual $\mathcal{O}(1)$

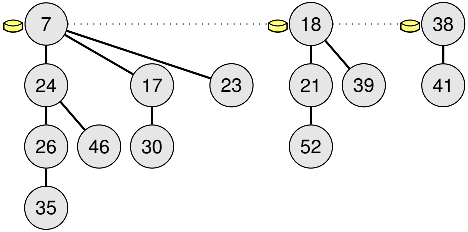
$$\Phi(H) = \text{trees}(H)$$



What if we don't have marked nodes?

- INSERT: actual $\mathcal{O}(1)$ amortized $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$ amortized $\mathcal{O}(d(n))$
- DECREASE-KEY: actual $\mathcal{O}(1)$ amortized $\mathcal{O}(1)$

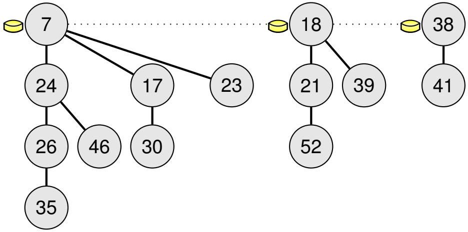
$$\Phi(H) = \text{trees}(H)$$



What if we don't have marked nodes?

- INSERT: actual $\mathcal{O}(1)$ amortized $\mathcal{O}(1)$
- EXTRACT-MIN: actual $\mathcal{O}(\text{trees}(H) + d(n))$ amortized $\mathcal{O}(d(n)) \neq \mathcal{O}(\log n)$
- DECREASE-KEY: actual $\mathcal{O}(1)$ amortized $\mathcal{O}(1)$

$$\Phi(H) = \text{trees}(H)$$



Summary

Operation	Linked list	Binary heap	Binomial heap	Fibon. heap
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
<u>INSERT</u>	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
<u>EXTRACT-MIN</u>	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
UNION	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
<u>DECREASE-KEY</u>	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Summary

Can we perform EXTRACT-MIN in $o(\log n)$?

Operation	Linked list	Binary heap	Binomial heap	Fibon. heap
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
UNION	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Summary

If this was possible, then there would be a sorting algorithm with runtime $o(n \log n)$!

Can we perform EXTRACT-MIN in $o(\log n)$?

Operation	Linked list	Binary heap	Binomial heap	Fibon. heap
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
UNION	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Summary

Operation	Linked list	Binary heap	Binomial heap	Fibon. heap
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
UNION	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Summary

Operation	Linked list	Binary heap	Binomial heap	Fibon. heap
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
UNION	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

DELETE = DECREASE-KEY + EXTRACT-MIN

Summary

Operation	Linked list	Binary heap	Binomial heap	Fibon. heap
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
UNION	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

DELETE = DECREASE-KEY + EXTRACT-MIN

EXTRACT-MIN = MIN + DELETE

Summary

Operation	Linked list	Binary heap	Binomial heap	Fibon. heap
MAKE-HEAP	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
INSERT	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
MINIMUM	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
UNION	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DECREASE-KEY	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
DELETE	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Crucial for many applications including shortest paths and minimum spanning trees!

Recent Studies

- Fibonacci Numbers were discovered >800 years ago
- Fibonacci Heaps were developed by Fredman and Tarjan in 1984

Recent Studies

- Fibonacci Numbers were discovered >800 years ago
- Fibonacci Heaps were developed by Fredman and Tarjan in 1984

Brodal, Lagogiannis, Tarjan: Strict Fibonacci Heap (STOC'12)

Strict Fibonacci Heap:

- pointer-based heap implementation similar to Fibonacci Heaps
- achieves the same cost as Fibonacci Heaps, but actual costs!

Recent Studies

- Fibonacci Numbers were discovered >800 years ago
- Fibonacci Heaps were developed by Fredman and Tarjan in 1984

Brodal, Lagogiannis, Tarjan: Strict Fibonacci Heap (STOC'12)

Strict Fibonacci Heap:

- pointer-based heap implementation similar to Fibonacci Heaps
- achieves the same cost as Fibonacci Heaps, but actual costs!

Li, Peebles: Replacing Mark Bits with Randomness in Fibonacci Heap (ICALP'15)

- Queries to marked bits are intercepted and responded with a random bit

Recent Studies

- Fibonacci Numbers were discovered >800 years ago
- Fibonacci Heaps were developed by Fredman and Tarjan in 1984

Brodal, Lagogiannis, Tarjan: Strict Fibonacci Heap (STOC'12)

Strict Fibonacci Heap:

- pointer-based heap implementation similar to Fibonacci Heaps
- achieves the same cost as Fibonacci Heaps, but actual costs!

Li, Peebles: Replacing Mark Bits with Randomness in Fibonacci Heap (ICALP'15)

- Queries to marked bits are intercepted and responded with a random bit
- several lower bounds on the amortized cost in terms of the size of the heap **and** the number of operations

Recent Studies

- Fibonacci Numbers were discovered >800 years ago
- Fibonacci Heaps were developed by Fredman and Tarjan in 1984

Brodal, Lagogiannis, Tarjan: Strict Fibonacci Heap (STOC'12)

Strict Fibonacci Heap:

- pointer-based heap implementation similar to Fibonacci Heaps
- achieves the same cost as Fibonacci Heaps, but actual costs!

Li, Peebles: Replacing Mark Bits with Randomness in Fibonacci Heap (ICALP'15)

- Queries to marked bits are intercepted and responded with a random bit
- several lower bounds on the amortized cost in terms of the size of the heap **and** the number of operations

$\Rightarrow$ less efficient than the original Fibonacci heap

Recent Studies

- Fibonacci Numbers were discovered >800 years ago
- Fibonacci Heaps were developed by Fredman and Tarjan in 1984

Brodal, Lagogiannis, Tarjan: Strict Fibonacci Heap (STOC'12)

Strict Fibonacci Heap:

- pointer-based heap implementation similar to Fibonacci Heaps
- achieves the same cost as Fibonacci Heaps, but actual costs!

Li, Peebles: Replacing Mark Bits with Randomness in Fibonacci Heap (ICALP'15)

- Queries to marked bits are intercepted and responded with a random bit
- several lower bounds on the amortized cost in terms of the size of the heap **and** the number of operations

$\Rightarrow$ less efficient than the original Fibonacci heap

$\Rightarrow$ marked bit is not redundant!

Outlook: A More Efficient Priority Queue for fixed Universe

Operation	Fibonacci heap amortized cost	Van Emde Boas Tree actual cost
INSERT	$\mathcal{O}(1)$	$\mathcal{O}(\log\log u)$
MINIMUM	$\mathcal{O}(1)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(\log n)$	$\mathcal{O}(\log\log u)$
MERGE/UNION	$\mathcal{O}(1)$	-
DECREASE-KEY	$\mathcal{O}(1)$	$\mathcal{O}(\log\log u)$
DELETE	$\mathcal{O}(\log n)$	$\mathcal{O}(\log\log u)$
SUCC	-	$\mathcal{O}(\log\log u)$
PRED	-	$\mathcal{O}(\log\log u)$
MAXIMUM	-	$\mathcal{O}(1)$

Outlook: A More Efficient Priority Queue for fixed Universe

Operation	Fibonacci heap amortized cost	Van Emde Boas Tree actual cost
INSERT	$\mathcal{O}(1)$	$\mathcal{O}(\log\log u)$
MINIMUM	$\mathcal{O}(1)$	$\mathcal{O}(1)$
EXTRACT-MIN	$\mathcal{O}(\log n)$	$\mathcal{O}(\log\log u)$
MERGE/UNION	$\mathcal{O}(1)$	-
DECREASE-KEY	$\mathcal{O}(1)$	$\mathcal{O}(\log\log u)$
DELETE	$\mathcal{O}(\log n)$	$\mathcal{O}(\log\log u)$
SUCC	-	$\mathcal{O}(\log\log u)$
PRED	-	$\mathcal{O}(\log\log u)$
MAXIMUM	-	$\mathcal{O}(1)$

all this requires key values to be in a universe of size u!