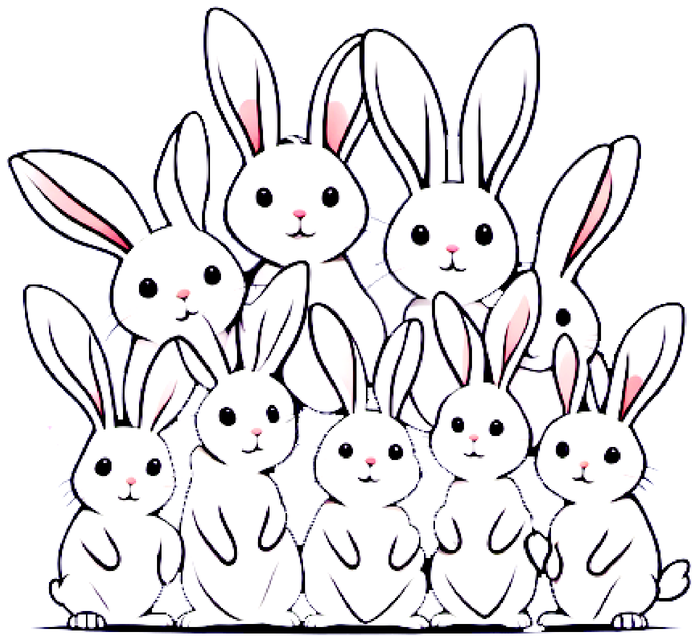
Fibonacci Heaps



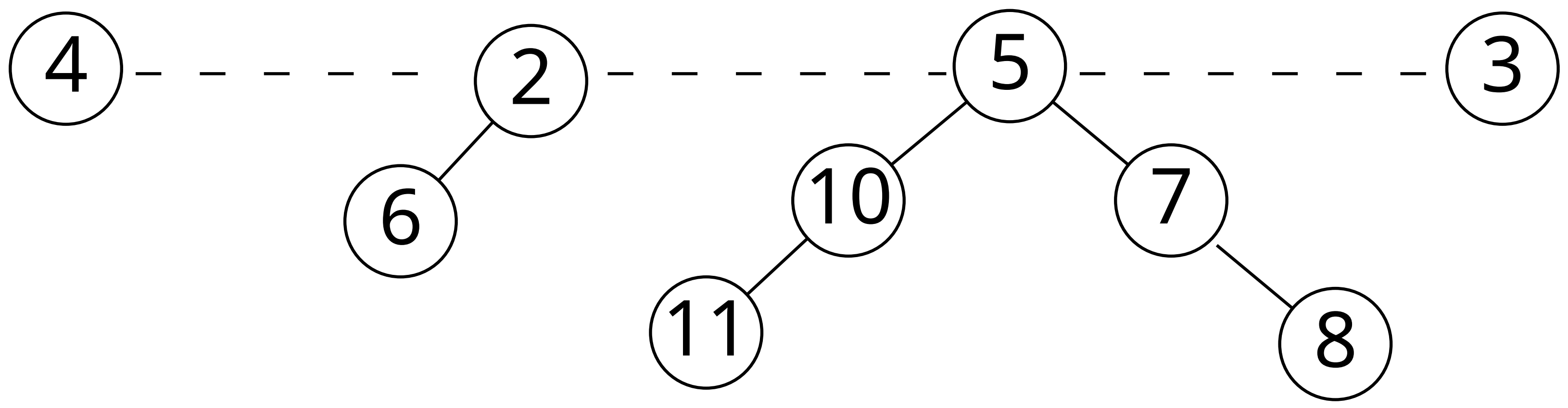
Fibonacci Heaps

Goal: Efficiently mergeable heaps with efficient decreaseKey

Fibonacci Heaps

Goal: Efficiently mergeable heaps with efficient decreaseKey

Fibonacci Heap is a set of trees with min-heap property,

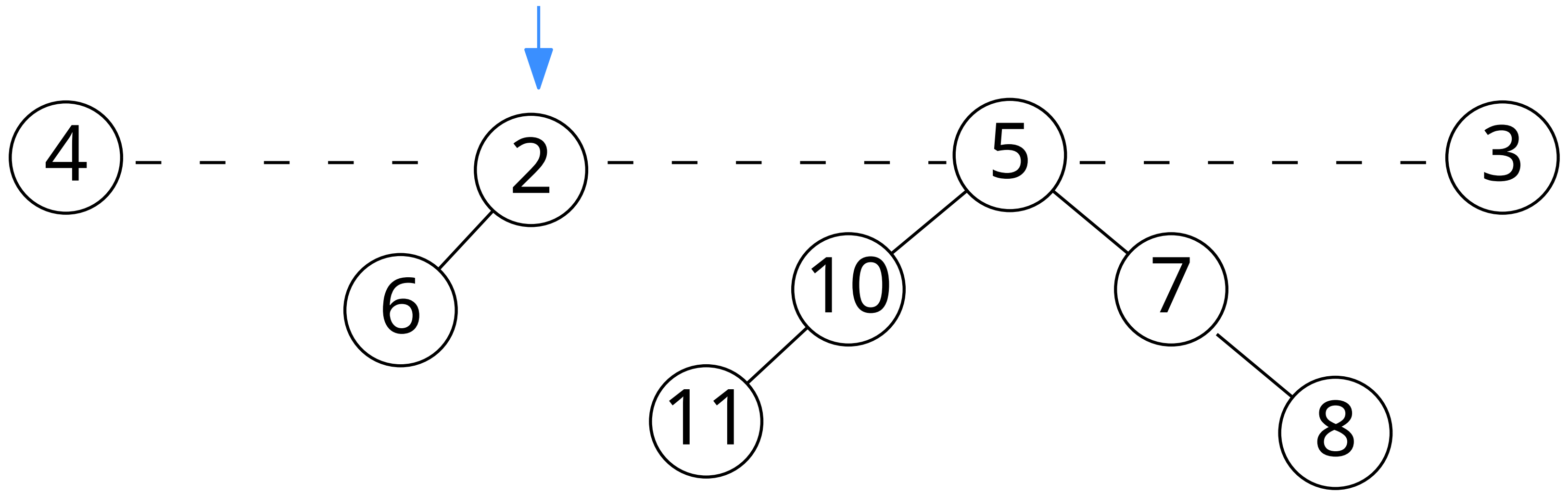


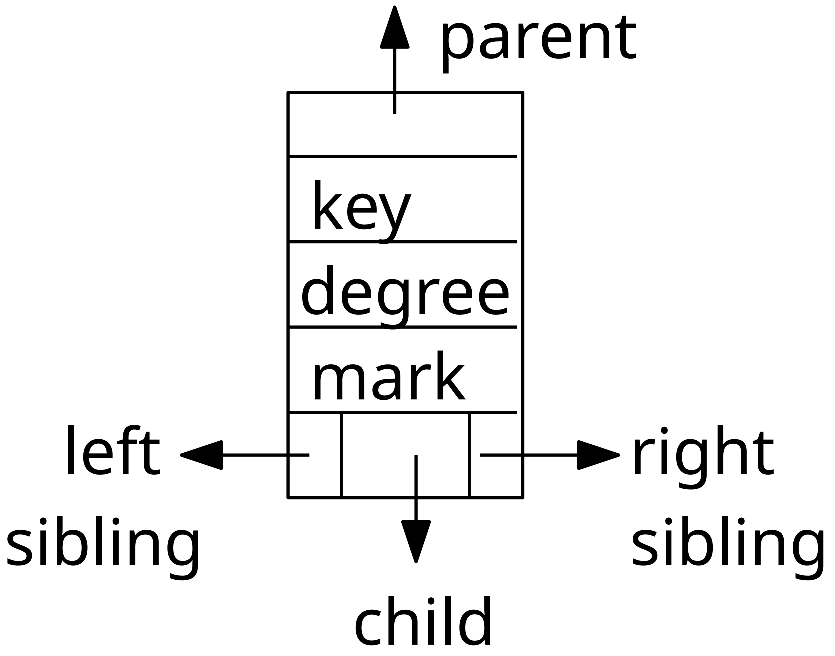
Fibonacci Heaps

Goal: Efficiently mergeable heaps with efficient decreaseKey

Fibonacci Heap is a set of trees with min-heap property, in it

- each node has four pointers, one each to parent, child, left sibling, right sibling as well as two attributes: degree and mark
- roots of all trees are stored in a doubly linked list
- pointer to minRoot in list





Idea: like binomial heaps, but more flexible structure; link trees of equal degree

Fibonacci Heaps

Goal: Efficiently mergeable heaps with efficient decreaseKey

Fibonacci Heap is a set of trees with min-heap property,

We will use the potential method for the amortized analysis with potential function

$$\Phi(H) = \underbrace{t(H)}_{\#\text{ trees}} + 2\underbrace{m(H)}_{\#\text{ marks}}$$

Fibonacci Heaps

Goal: Efficiently mergeable heaps with efficient decreaseKey

Fibonacci Heap is a set of trees with min-heap property,

We will use the potential method for the amortized analysis with potential function

$$\Phi(H) = \underbrace{t(H)}_{\#\text{ trees}} + \boxed{\underbrace{2m(H)}_{\#\text{ marks}}}$$

ignore for now,
happens in
decreaseKey

Fibonacci Heaps

Goal: Efficiently mergeable heaps with efficient decreaseKey

Fibonacci Heap is a set of trees with min-heap property,

We will use the potential method for the amortized analysis with potential function

$$\Phi(H) = \underbrace{t(H)}_{\#\text{ trees}} \boxed{+ \underbrace{2m(H)}_{\#\text{ marks}}}$$

ignore for now, happens in decreaseKey

an empty fib-heap has $\Phi(H) = 0$ and any fib-heap has $\Phi(H) \geq 0$
$\Rightarrow$ amortised cost upper bound the actual costs

Fibonacci Heaps

Goal: Efficiently mergeable heaps with efficient decreaseKey

Fibonacci Heap is a set of trees with min-heap property,

We will use the potential method for the amortized analysis with potential function

$$\Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + \boxed{2\underbrace{m(H)}_{\text{\# marks}}}$$

ignore for now, happens in decreaseKey

an empty fib-heap has $\Phi(H) = 0$ and any fib-heap has $\Phi(H) \geq 0$
$\Rightarrow$ amortised cost upper bound the actual costs

We will analyze the amortized cost with respect to $D(n)$, which is the maximum degree of a node in a Fibonacci heap on n nodes. Afterwards we will bound

$$D(n) = O(\log n).$$

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\quad\quad \Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$

- make-0: generate empty heap

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

$$\Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$$

- make-0: generate empty heap $\quad \hat{c}_i = O(1) + 0 = O(1)$

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

$$\Phi(H) = \underbrace{t(H)}_{\#\text{ trees}} + 2\underbrace{m(H)}_{\#\text{ marks}}$$

- make-0: generate empty heap $\hat{c}_i = O(1) + 0 = O(1)$
- min: return key of minRoot

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = \underbrace{t(H)}_{\#\text{ trees}} + 2\underbrace{m(H)}_{\#\text{ marks}}$

- make-0: generate empty heap $\qquad \hat{c}_i = O(1) + 0 = O(1)$
- min: return key of minRoot $\qquad \hat{c}_i = O(1) + 0 = O(1)$

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = \underbrace{t(H)}_{\#\text{ trees}} + 2\underbrace{m(H)}_{\#\text{ marks}}$

- make-0: generate empty heap $\qquad \hat{c}_i = O(1) + 0 = O(1)$
- min: return key of minRoot $\qquad \hat{c}_i = O(1) + 0 = O(1)$
- union:
  - concatenate the two lists
  - update minRoot (by comparing the two minRoots before)

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = \underbrace{t(H)}_{\#\text{ trees}} + 2\underbrace{m(H)}_{\#\text{ marks}}$

- make-0: generate empty heap $\quad \hat{c}_i = O(1) + 0 = O(1)$
- min: return key of minRoot $\quad \hat{c}_i = O(1) + 0 = O(1)$
- union:
  - concatenate the two lists $\quad \hat{c}_i = O(1) + 0 = O(1)$
  - update minRoot (by comparing the two minRoots before)

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\quad \Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$

- make-0: generate empty heap $\quad \hat{c}_i = O(1) + 0 = O(1)$
- min: return key of minRoot $\quad \hat{c}_i = O(1) + 0 = O(1)$
- union:
  - concatenate the two lists $\quad \hat{c}_i = O(1) + 0 = O(1)$
  - update minRoot (by comparing the two minRoots before)
- insert:
  - make-1 and add to list
  - update minRoot

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = \underbrace{t(H)}_{\#\text{ trees}} + 2\underbrace{m(H)}_{\#\text{ marks}}$

- make-0: generate empty heap $\qquad \hat{c}_i = O(1) + 0 = O(1)$
- min: return key of minRoot $\qquad \hat{c}_i = O(1) + 0 = O(1)$
- union:
  - concatenate the two lists $\qquad \hat{c}_i = O(1) + 0 = O(1)$
  - update minRoot (by comparing the two minRoots before)
- insert:
  - make-1 and add to list $\qquad \hat{c}_i = O(1) + 1 = O(1)$
  - update minRoot

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$

- make-0: generate empty heap $\hat{c}_i = O(1) + 0 = O(1)$
- min: return key of minRoot $\hat{c}_i = O(1) + 0 = O(1)$
- union:
  - concatenate the two lists $\hat{c}_i = O(1) + 0 = O(1)$
  - update minRoot (by comparing the two minRoots before)
- insert:
  - make-1 and add to list $\hat{c}_i = O(1) + 1 = O(1)$
  - update minRoot

Q: what does a Fibonacci heap look like after n inserts?

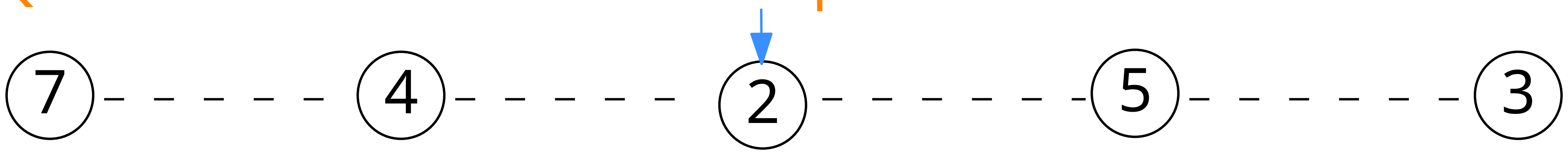
Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\quad\quad$ $\Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + \underbrace{2m(H)}_{\text{\# marks}}$

- make-0: generate empty heap $\quad \hat{c}_i = O(1) + 0 = O(1)$
- min: return key of minRoot $\quad \hat{c}_i = O(1) + 0 = O(1)$
- union:
  - concatenate the two lists $\quad \hat{c}_i = O(1) + 0 = O(1)$
  - update minRoot (by comparing the two minRoots before)
- insert:
  - make-1 and add to list $\quad \hat{c}_i = O(1) + 1 = O(1)$
  - update minRoot

Q: what does a Fibonacci heap look like after n inserts?



Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$

- deleteMin:
  - delete node minRoot
  - create array of size $D(n) + 1$
  - insert all roots (including children of deleted minRoot) in array, always linking roots of equal degree
  - create linked list of roots from array

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$

- deleteMin:
  - delete node minRoot
  - create array of size $D(n) + 1$
  - insert all roots (including children of deleted minRoot) in array, always linking roots of equal degree
  - create linked list of roots from array

actual costs: $O(D(n) + t(H))$

max # children of minRoot # trees before

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = \underbrace{t(H)}_{\#\text{ trees}} + 2\underbrace{m(H)}_{\#\text{ marks}}$

- deleteMin:
  - delete node minRoot
  - create array of size $D(n) + 1$
  - insert all roots (including children of deleted minRoot) in array, always linking roots of equal degree
  - create linked list of roots from array

actual costs: $O(D(n) + t(H))$

max # children of minRoot $\qquad$ # trees before

change in potential:

$$\leq \underbrace{(D(n) + 1 + 2m(H))}_{\text{potential after}} - \underbrace{(t(H) + 2m(H))}_{\text{potential before}} = D(n) + 1 - t(H)$$

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\quad\Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$

- deleteMin:
  - delete node minRoot
  - create array of size $D(n) + 1$
  - insert all roots (including children of deleted minRoot) in array, always linking roots of equal degree
  - create linked list of roots from array

actual costs: $O(D(n) + t(H))$

max # children of minRoot $\quad$ # trees before

change in potential:

$$\leq (D(n) + 1 + 2m(H)) - (t(H) + 2m(H)) = D(n) + 1 - t(H)$$

hence amortised cost $\hat{c}_i = O(D(n) + t(H)) + D(n) + 1 - t(H)$

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = \underbrace{t(H)}_{\#\text{ trees}} + \underbrace{2m(H)}_{\#\text{ marks}}$

- deleteMin:
  - delete node minRoot
  - create array of size $D(n) + 1$
  - insert all roots (including children of deleted minRoot) in array, always linking roots of equal degree
  - create linked list of roots from array

actual costs: $O(D(n) + t(H))$

max $\#$ children of minRoot $\qquad$ # trees before

change in potential:

$$\leq (D(n) + 1 + 2m(H)) - (t(H) + 2m(H)) = D(n) + 1 - t(H)$$

"assuming we sufficently scale up the potential"

hence amortised cost $\hat{c}_i = O(D(n) \boxed{+ t(H)}) + D(n) + 1 \boxed{- t(H)} = O(D(n))$

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = c(\underbrace{t(H)}_{\#\text{ trees}} + 2\underbrace{m(H)}_{\#\text{ marks}})$

- deleteMin:
  - delete node minRoot
  - create array of size $D(n) + 1$
  - insert all roots (including children of deleted minRoot) in array, always linking roots of equal degree
  - create linked list of roots from array

actual costs: $O(\underbrace{D(n)}_{\text{max }\#\text{ children of minRoot}} + \underbrace{t(H)}_{\#\text{ trees before}})$

"assuming we sufficently scale up the potential"

change in potential:

$$\leq (D(n) + 1 + 2m(H)) - (t(H) + 2m(H)) = D(n) + 1 - t(H)$$

hence amortised cost $\hat{c}_i = O(D(n)\boxed{+\, t(H)}) + D(n) + 1 \boxed{-\, t(H)} = O(D(n))$

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$

- deleteMin:
  - delete node minRoot
  - create array of size $D(n) + 1$
  - insert all roots (including children of deleted minRoot) in array, always linking roots of equal degree
  - create linked list of roots from array

Note: so far, no nodes were marked

actual costs: $O(\underbrace{D(n)}_{\text{max \# children of minRoot}} + \underbrace{t(H)}_{\text{\# trees before}})$

change in potential:

$$\leq (D(n) + 1 + 2m(H)) - (t(H) + 2m(H)) = D(n) + 1 - t(H)$$

hence amortised cost $\hat{c}_i = O(D(n) + t(H)) + D(n) + 1 - t(H) = O(D(n))$

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\quad\quad \Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + \underbrace{2m(H)}_{\text{\# marks}}$

- deleteMin:
  - delete node minRoot
  - create array of size $D(n) + 1$
  - insert all roots (including children of deleted minRoot) in array, always linking roots of equal degree
  - create linked list of roots from array

Note: so far, no nodes were marked

Q: after n inserts, then one deleteMin, is a fib-heap a binomial heap?

short for Fibonacci heap

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$

- deleteMin:
  - delete node minRoot
  - create array of size $D(n) + 1$
  - insert all roots (including children of deleted minRoot) in array, always linking roots of equal degree
  - create linked list of roots from array

Note: so far, no nodes were marked

Q: after n inserts, then one deleteMin, is a fib-heap a binomial heap? Yes!

short for Fibonacci heap

Operations

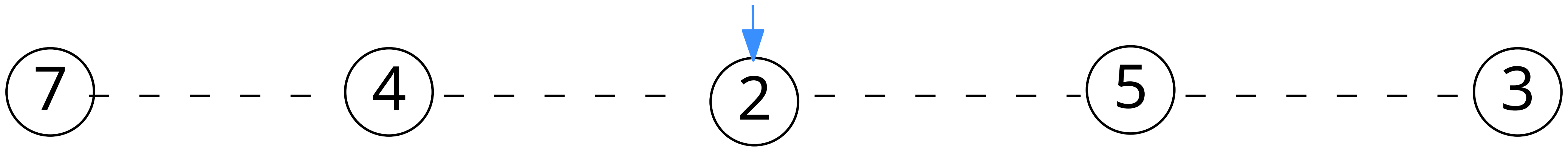
Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\quad \Phi(H) = \underbrace{t(H)}_{\#\,\text{trees}} + 2\underbrace{m(H)}_{\#\,\text{marks}}$

- deleteMin:
  - delete node minRoot
  - create array of size $D(n) + 1$
  - insert all roots (including children of deleted minRoot) in array, always linking roots of equal degree
  - create linked list of roots from array

Note: so far, no nodes were marked

Example:



Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

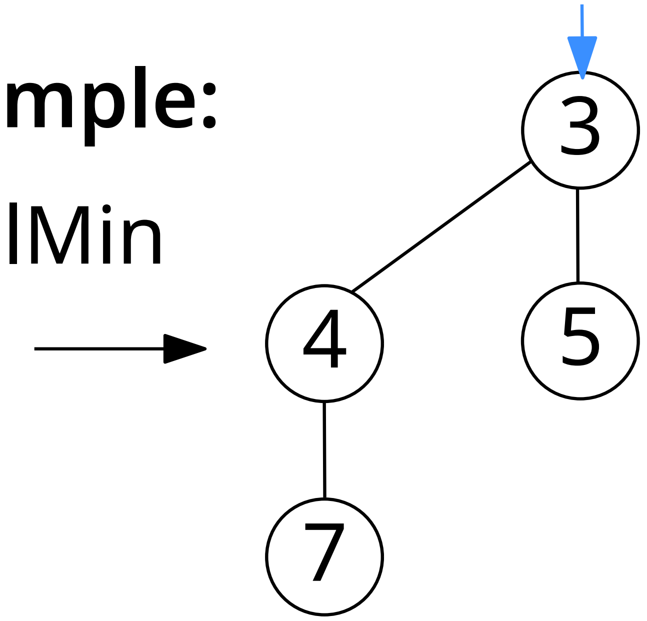
Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$

- deleteMin:
  - delete node minRoot
  - create array of size $D(n) + 1$
  - insert all roots (including children of deleted minRoot) in array, always linking roots of equal degree
  - create linked list of roots from array

Note: so far, no nodes were marked

Example:

delMin



Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

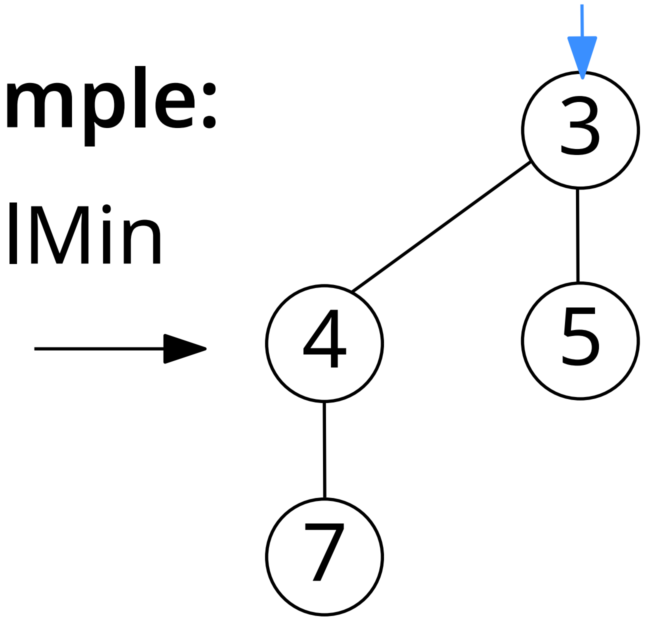
Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = \underbrace{t(H)}_{\#\text{ trees}} + 2\underbrace{m(H)}_{\#\text{ marks}}$

- deleteMin:
  - delete node minRoot
  - create array of size $D(n) + 1$
  - insert all roots (including children of deleted minRoot) in array, always linking roots of equal degree
  - create linked list of roots from array

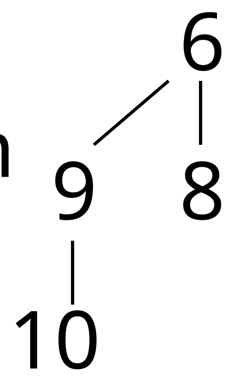
Note: so far, no nodes were marked

Example:

delMin



union



, insert(2), delMin

Operations

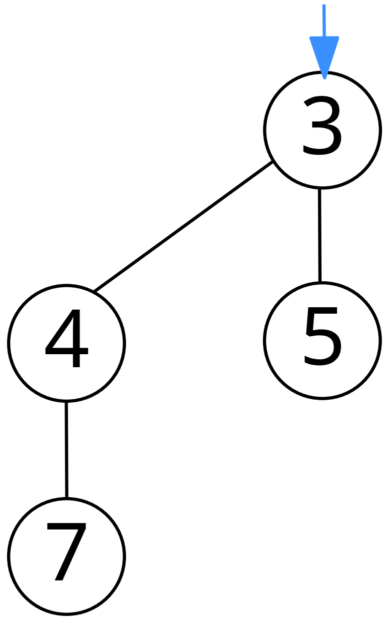
Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = \underbrace{t(H)}_{\#\text{ trees}} + 2\underbrace{m(H)}_{\#\text{ marks}}$

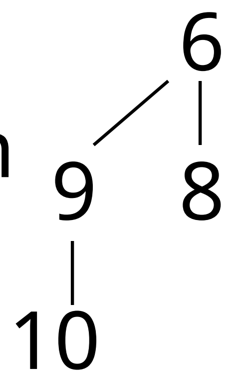
- deleteMin:
  - delete node minRoot
  - create array of size $D(n) + 1$
  - insert all roots (including children of deleted minRoot) in array, always linking roots of equal degree
  - create linked list of roots from array

Example:

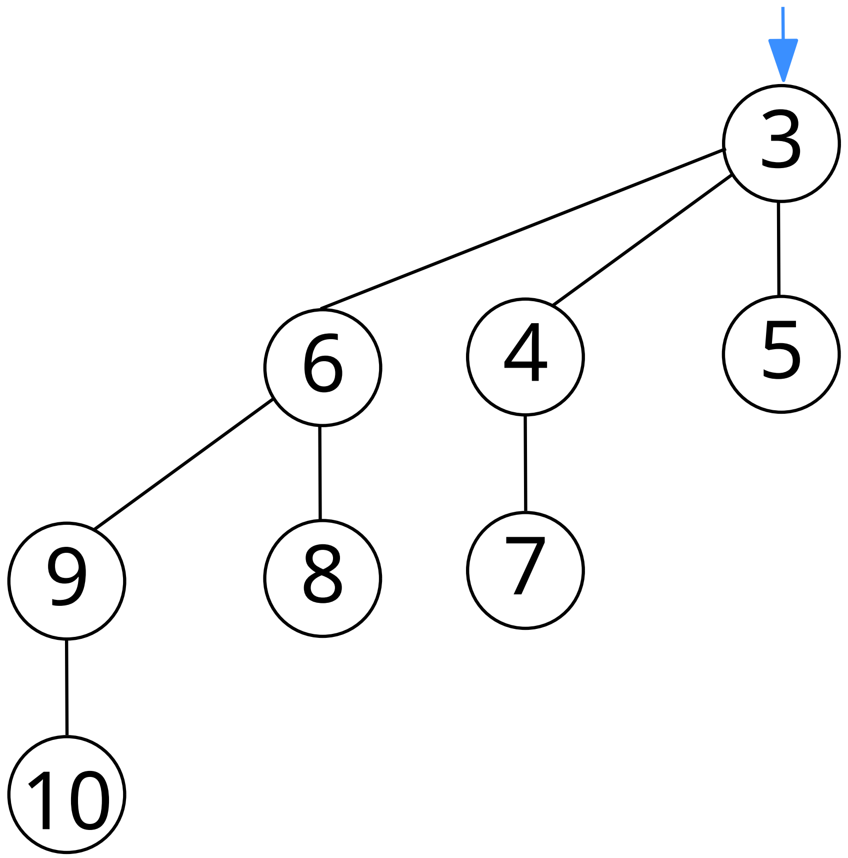
delMin $\longrightarrow$



union



, insert(2), delMin $\longrightarrow$



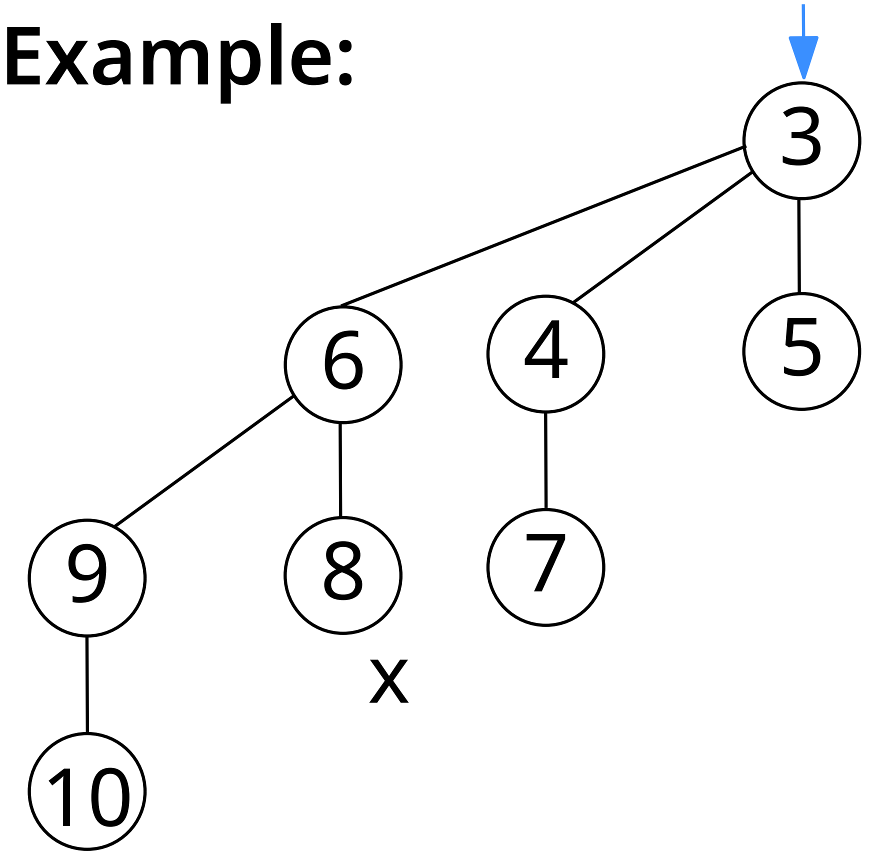
Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$

- decKey:
  - if key too small, cut subtree at node and add node to list of roots
  - if parent was unmarked, mark it (unless it is a root)

Example:



decKey(x,2)

$\longrightarrow$

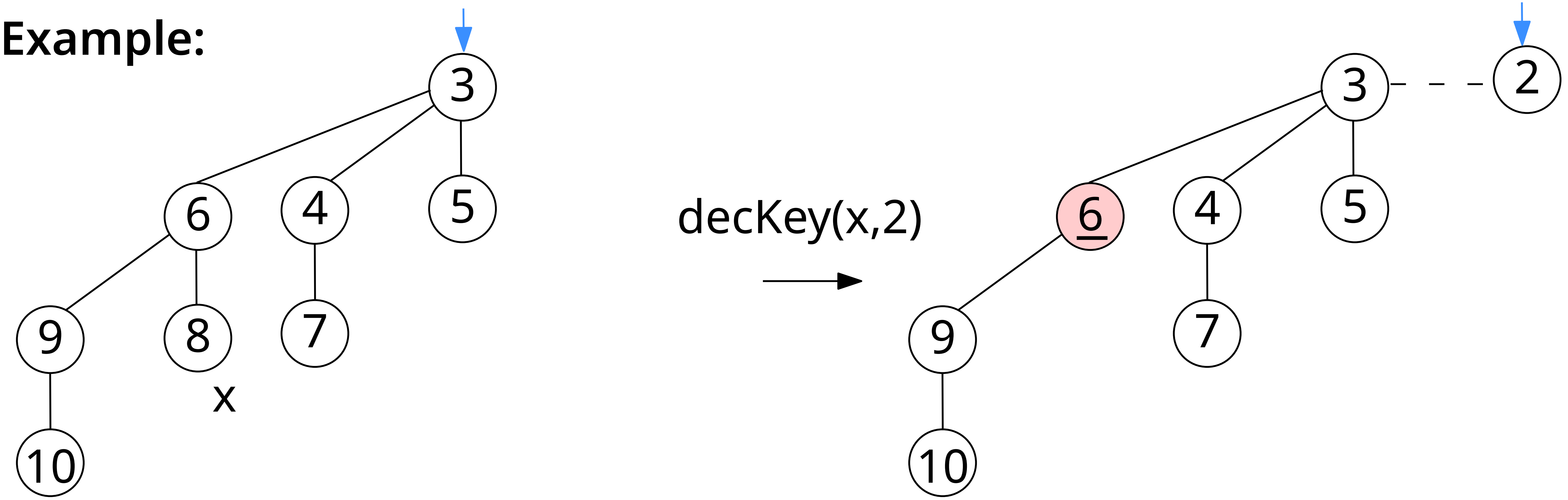
Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\quad\quad \Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$

- decKey:
  - if key too small, cut subtree at node and add node to list of roots
  - if parent was unmarked, mark it (unless it is a root)

Example:



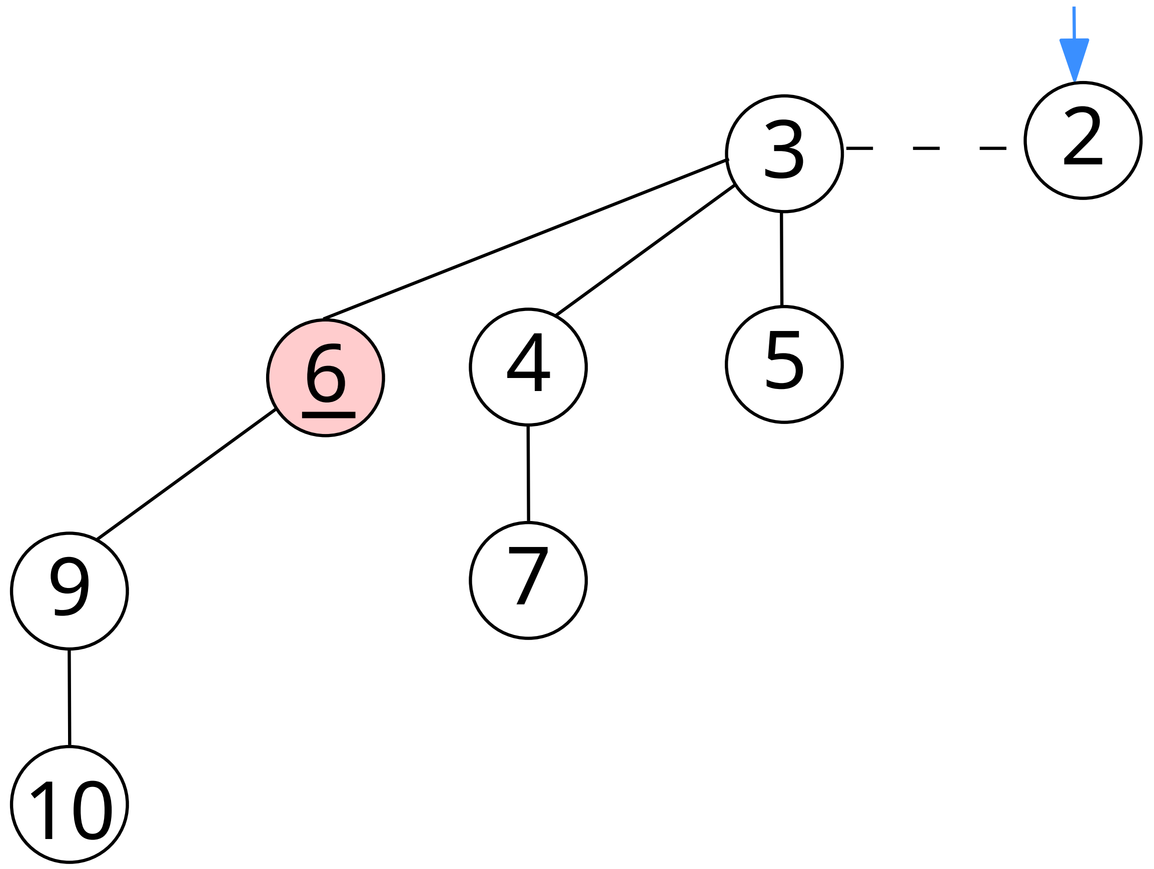
Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\Phi(H) = \underbrace{t(H)}_{\#\text{ trees}} + 2\underbrace{m(H)}_{\#\text{ marks}}$

- decKey:
  - if key too small, cut subtree at node and add node to list of roots
  - if parent was unmarked, mark it (unless it is a root)

Example:



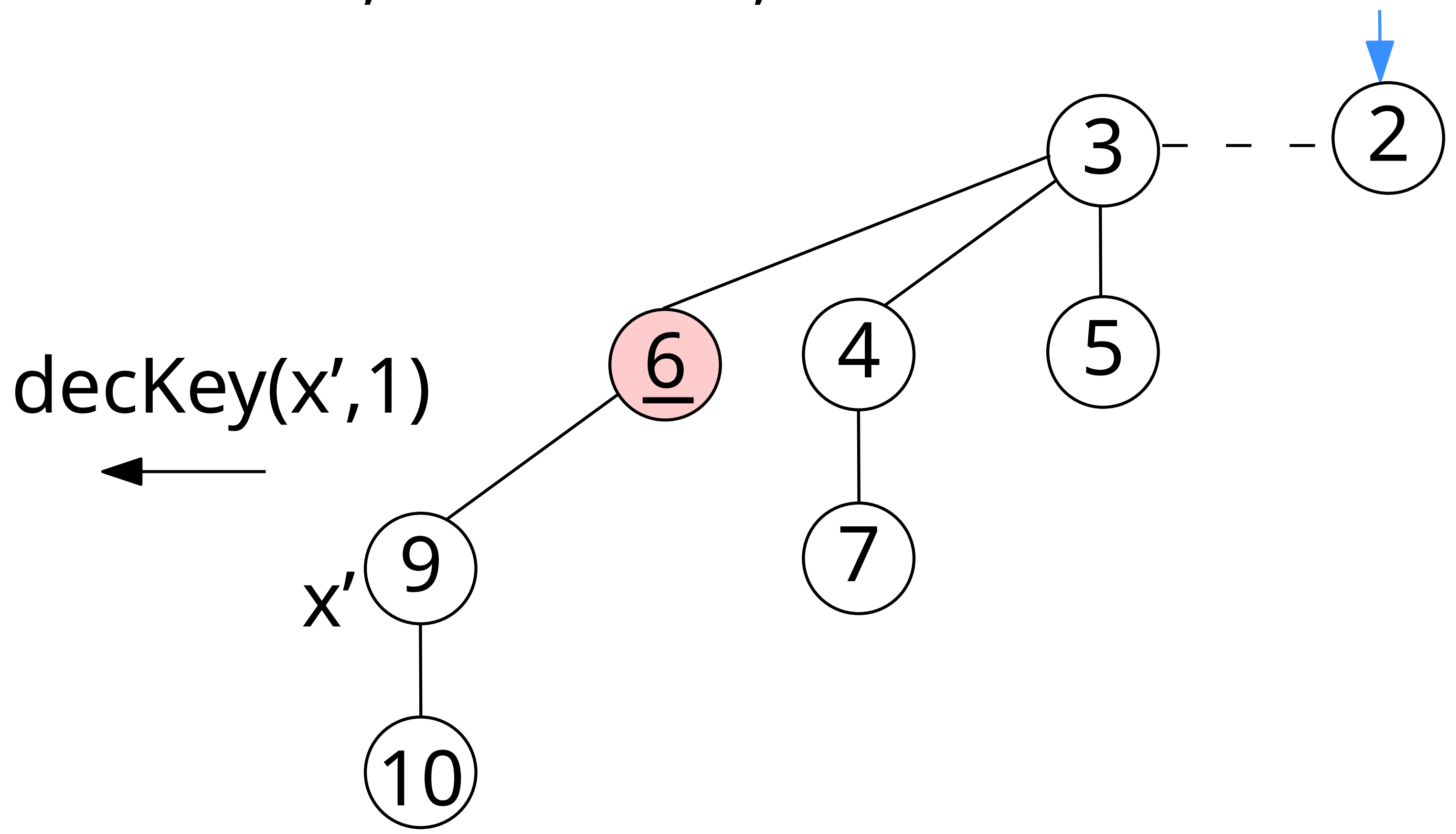
Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$

- decKey:
  - if key too small, cut subtree at node and add node to list of roots
  - if parent was unmarked, mark it (unless it is a root)
  - if parent was marked, cut its subtree, unmark it, and recurse

Example:



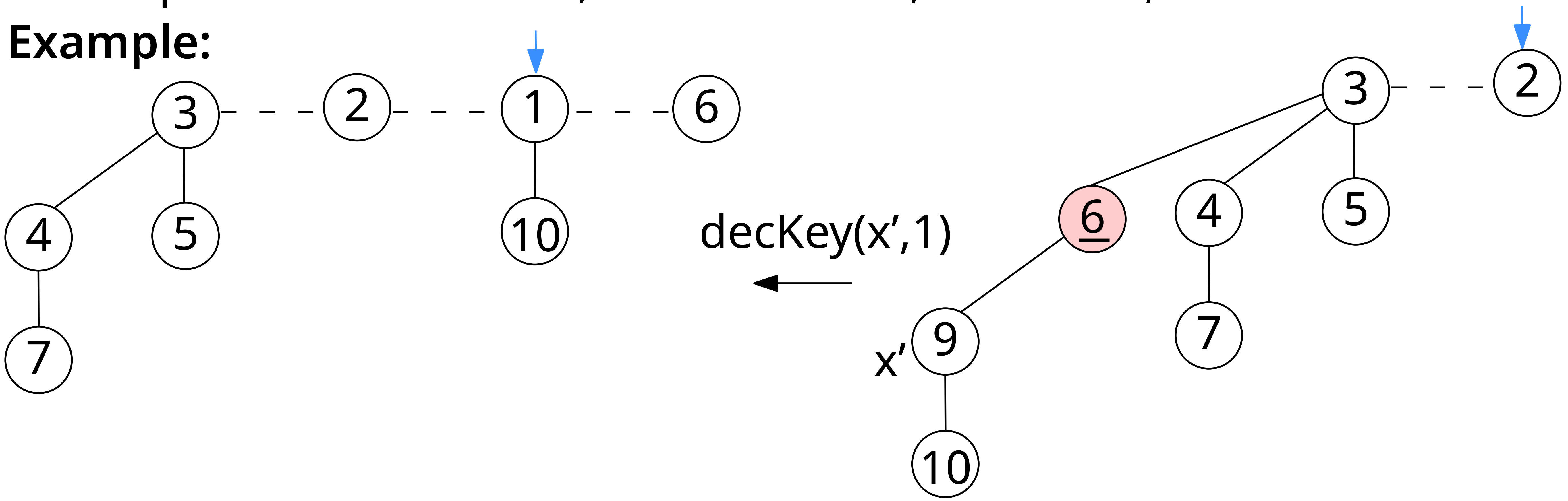
Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\quad\quad \Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$

- decKey:
  - if key too small, cut subtree at node and add node to list of roots
  - if parent was unmarked, mark it (unless it is a root)
  - if parent was marked, cut its subtree, unmark it, and recurse

Example:



Operations

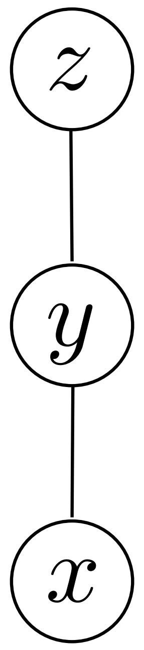
Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\quad\quad \Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$

- decKey:

decKey(x, k)
  x.key $= k$
  $y = x$.parent
  if $y \neq$ nil and x.key $< y$.key **then**
    cut(x)
    cascade-cut(y)
  if $k <$ min **then**
    min $= k$

cascade-cut(y)
  $z = y$.parent
  if $z \neq$ nil **then**
    if y.mark = false **then**
      y.mark = true
    else
      cut(y)
      cascade-cut(z)



Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = \underbrace{t(H)}_{\#\text{ trees}} + 2\underbrace{m(H)}_{\#\text{ marks}}$

- decKey:

actual cost: $O(1+k)$ for $k-1$ cascading cuts (= k cuts with first)

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = \underbrace{t(H)}_{\#\text{ trees}} + 2\underbrace{m(H)}_{\#\text{ marks}}$

- decKey:

actual cost: $O(1+k)$ for $k-1$ cascading cuts (= k cuts with first)

change in potential:

$$\leq \underbrace{t(H) + \underset{\substack{\uparrow\\ \text{new trees}}}{k} + 2(m(H) \underline{- k + 2})}_{\text{potential after}} - \underbrace{(t(H) + 2m(H))}_{\text{potential before}} = 4 - k$$

$-(k-1)$ for cascading cuts, $+1$ new mark

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\quad\Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$

- decKey:

actual cost: $O(1+k)$ for $k-1$ cascading cuts (= k cuts with first)

change in potential:

$$\leq \underbrace{t(H) + k + 2(m(H) - k + 2)}_{\text{potential after}} - \underbrace{(t(H) + 2m(H))}_{\text{potential before}} = 4 - k$$

hence amortised cost $\hat{c}_i = O(k) + 4 - k = O(1)$

again by appropriate scaling of the potential

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\quad \Phi(H) = \underbrace{t(H)}_{\#\text{ trees}} + 2\underbrace{m(H)}_{\#\text{ marks}}$

- decKey:

actual cost: $O(1+k)$ for $k-1$ cascading cuts (= k cuts with first)

change in potential:

$$\le \underbrace{t(H) + k + 2(m(H) - k + 2)}_{\text{potential after}} - \underbrace{(t(H) + 2m(H))}_{\text{potential before}} = 4 - k$$

hence amortised cost $\hat{c}_i = O(k) + 4 - k = O(1)$

again by appropriate scaling of the potential

Q: Why did we need the factor 2 for $m(H)$ in the potential?

Operations

Idea: Operations (except for decreaseKey) like Binomial Heap with lazy union

Amortised costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ $\qquad \Phi(H) = \underbrace{t(H)}_{\text{\# trees}} + 2\underbrace{m(H)}_{\text{\# marks}}$

- decKey:

actual cost: $O(1+k)$ for $k-1$ cascading cuts (= k cuts with first)

change in potential:

$$\leq \underbrace{t(H) + k + 2(m(H) - k + 2)}_{\text{potential after}} - \underbrace{(t(H) + 2m(H))}_{\text{potential before}} = 4 - k$$

hence amortised cost $\hat{c}_i = O(k) + 4 - k = O(1)$

again by appropriate scaling of the potential

Q: Why did we need the factor 2 for $m(H)$ in the potential?

one for paying for a cut, one for paying for the new tree

Runtimes

	amortised
make	$O(1)$
min	$O(1)$
insert	$O(1)$
union	$O(1)$
deleteMin	$O(D(n))$
decKey	$O(1)$

Runtimes

	amortised
make	$O(1)$
min	$O(1)$
insert	$O(1)$
union	$O(1)$
deleteMin	$O(D(n))$
decKey	$O(1)$

Can we bound $D(n)$?

Runtimes

	amortised
make	$O(1)$
min	$O(1)$
insert	$O(1)$
union	$O(1)$
deleteMin	$O(D(n))$
decKey	$O(1)$

Can we bound $D(n)$? yes!

Intuition

if we would never cut $\rightarrow$ Binomial heap

Runtimes

	amortised
make	$O(1)$
min	$O(1)$
insert	$O(1)$
union	$O(1)$
deleteMin	$O(D(n))$
decKey	$O(1)$

Can we bound $D(n)$? yes!

Intuition

if we would never cut $\rightarrow$ Binomial heap

then: for any node x of degree k in a binomial tree:

size(x) = 2^k (by induction).

size of subtree with x as root

Runtimes

	amortised
make	$O(1)$
min	$O(1)$
insert	$O(1)$
union	$O(1)$
deleteMin	$O(D(n))$
decKey	$O(1)$

Can we bound $D(n)$? yes!

Intuition

if we would never cut $\longrightarrow$ Binomial heap

then: for any node x of degree k in a binomial tree:

size(x) = 2^k (by induction).

size of subtree with x as root

for binomial heaps this showed: $D(n) = O(\log n)$

Runtimes

	amortised
make	$O(1)$
min	$O(1)$
insert	$O(1)$
union	$O(1)$
deleteMin	$O(D(n))$
decKey	$O(1)$

Can we bound $D(n)$? yes!

Intuition

if we would never cut $\rightarrow$ Binomial heap

then: for any node x of degree k in a binomial tree:

size(x) = 2^k (by induction).

size of subtree with x as root

for binomial heaps this showed: $D(n) = O(\log n)$

if we cut, we can cut at most one child per node

Runtimes

	amortised
make	$O(1)$
min	$O(1)$
insert	$O(1)$
union	$O(1)$
deleteMin	$O(D(n))$
decKey	$O(1)$

Can we bound $D(n)$? yes!

Intuition

if we would never cut $\rightarrow$ Binomial heap

then: for any node x of degree k in a binomial tree:

size(x) = 2^k (by induction).

size of subtree with x as root

for binomial heaps this showed: $D(n) = O(\log n)$

if we cut, we can cut at most one child per node

then: degree of any node decreases by at most 1,

inductive argument still works for size(x) $\geq \phi^k$.

(golden ratio)

Runtimes

	amortised
make	$O(1)$
min	$O(1)$
insert	$O(1)$
union	$O(1)$
deleteMin	$O(D(n))$
decKey	$O(1)$

Can we bound $D(n)$? yes!

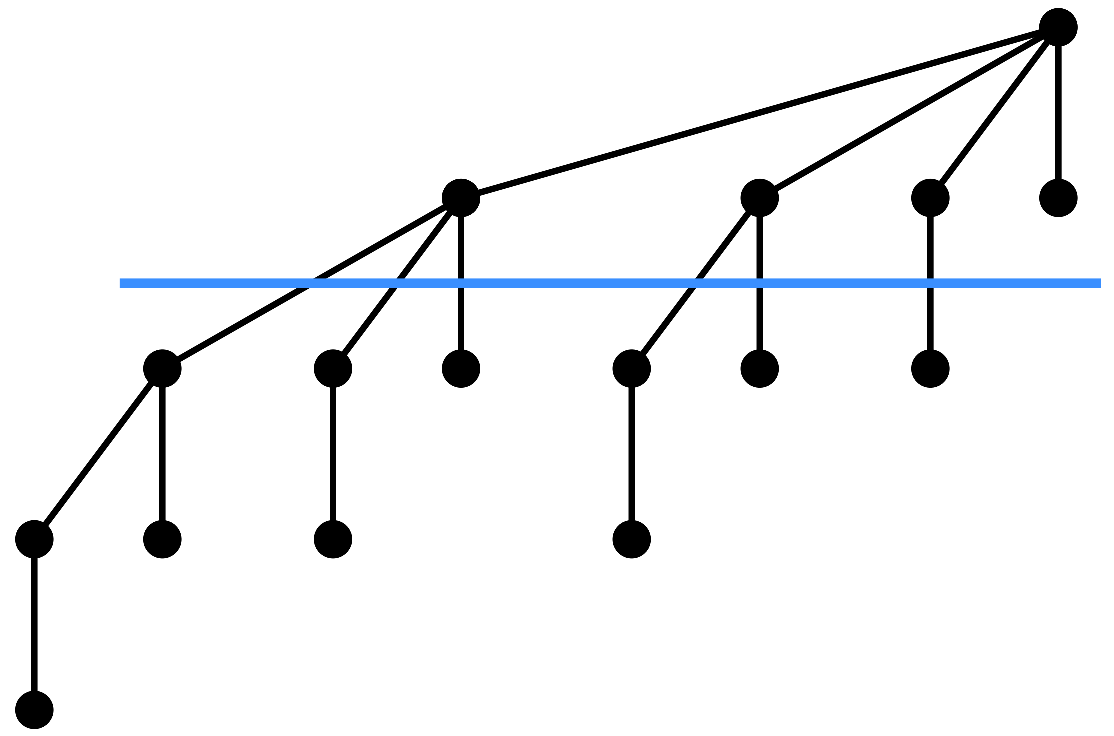
Q: Could we still bound $D(n)$ if we were allowed to cut more than 1 child per node?

Runtimes

	amortised
make	$O(1)$
min	$O(1)$
insert	$O(1)$
union	$O(1)$
deleteMin	$O(D(n))$
decKey	$O(1)$

Can we bound $D(n)$? yes!

Q: Could we still bound $D(n)$ if we were allowed to cut more than 1 child per node? no!



Bounding the maximal degree

Goal: show $D(n) \leq \lfloor \log_\phi n \rfloor$ where $\phi = \frac{1+\sqrt{5}}{2}$ the golden ratio.

Bounding the maximal degree

Goal: show $D(n) \leq \lfloor \log_\phi n \rfloor$ where $\phi = \frac{1+\sqrt{5}}{2}$ the golden ratio.

For this we show size$(x) \underset{\text{Lm 4}}{\geq} F_{k+2} \underset{\text{Lm 3}}{\geq} \phi^k$ for a node x in a Fib-heap of degree k.

Bounding the maximal degree

Goal: show $D(n) \leq \lfloor \log_\phi n \rfloor$ where $\phi = \frac{1+\sqrt{5}}{2}$ the golden ratio.

For this we show size$(x) \underset{\text{Lm 4}}{\geq} F_{k+2} \underset{\text{Lm 3}}{\geq} \phi^k$ for a node x in a Fib-heap of degree k.

($k+2$)-nd Fibonacci number

Recall: $F_0 = 0, F_1 = 1, F_2 = 1, F_3 = 2, \ldots F_{k+2} = F_k + F_{k+1}$

Bounding the maximal degree

Goal: show $D(n) \leq \lfloor \log_\phi n \rfloor$ where $\phi = \frac{1+\sqrt{5}}{2}$ the golden ratio.

For this we show size$(x) \underset{\text{Lm 4}}{\geq} F_{k+2} \underset{\text{Lm 3}}{\geq} \phi^k$ for a node x in a Fib-heap of degree k.

$(k+2)$-nd Fibonacci number

Recall: $F_0 = 0, F_1 = 1, F_2 = 1, F_3 = 2, \ldots F_{k+2} = F_k + F_{k+1}$

This implies $n \geq$ size$(x) \geq \phi^{D(n)}$ and thus $D(n) \leq \lfloor \log_\phi n \rfloor$.

Bounding the maximal degree

Goal: show $D(n) \leq \lfloor \log_\phi n \rfloor$ where $\phi = \frac{1+\sqrt{5}}{2}$ the golden ratio.

Lemma 1: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Let $y_1, \ldots, y_k$ be the children of x in order of age.
Then degree$[y_1] \geq 0$ and degree$[y_i] \geq i-2$ for $i = 2, \ldots, k$.

Bounding the maximal degree

Goal: show $D(n) \leq \lfloor \log_\phi n \rfloor$ where $\phi = \frac{1+\sqrt{5}}{2}$ the golden ratio.

Lemma 1: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Let $y_1, \ldots, y_k$ be the children of x in order of age.
Then degree$[y_1] \geq 0$ and degree$[y_i] \geq i-2$ for $i = 2, \ldots, k$.

time when y_i got linked to x

Bounding the maximal degree

Goal: show $D(n) \leq \lfloor \log_\phi n \rfloor$ where $\phi = \frac{1+\sqrt{5}}{2}$ the golden ratio.

Lemma 1: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Let $y_1, \ldots, y_k$ be the children of x in order of age. ← time when y_i got linked to x
Then degree$[y_1] \geq 0$ and degree$[y_i] \geq i - 2$ for $i = 2, \ldots, k$.

Proof: The first statement is trivial.
For the second, observe that when y_i was linked to x, it was degree$[y_i]$ = degree$[x] \geq i - 1$. Then at most one child was deleted.

Bounding the maximal degree

Goal: show $D(n) \leq \lfloor \log_\phi n \rfloor$ where $\phi = \frac{1+\sqrt{5}}{2}$ the golden ratio.

Lemma 1: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Let $y_1, \ldots, y_k$ be the children of x in order of age.
Then degree$[y_1] \geq 0$ and degree$[y_i] \geq i - 2$ for $i = 2, \ldots, k$.

Lemma 2: For $k \geq 0$ holds $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$.

Bounding the maximal degree

Goal: show $D(n) \leq \lfloor \log_\phi n \rfloor$ where $\phi = \frac{1+\sqrt{5}}{2}$ the golden ratio.

Lemma 1: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Let $y_1, \ldots, y_k$ be the children of x in order of age.
Then degree$[y_1] \geq 0$ and degree$[y_i] \geq i - 2$ for $i = 2, \ldots, k$.

Lemma 2: For $k \geq 0$ holds $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$.

Proof: by Induction(k)

$k = 0 : F_2 = 1 + 0 = 1 + F_0$

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

ind. hyp.

Bounding the maximal degree

Goal: show $D(n) \leq \lfloor \log_\phi n \rfloor$ where $\phi = \frac{1+\sqrt{5}}{2}$ the golden ratio.

Lemma 1: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Let $y_1, \ldots, y_k$ be the children of x in order of age.
Then degree$[y_1] \geq 0$ and degree$[y_i] \geq i-2$ for $i = 2, \ldots, k$.

Lemma 2: For $k \geq 0$ holds $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$.

Lemma 3: For $k \geq 0$ holds $F_{k+2} \geq \phi^k$.

Bounding the maximal degree

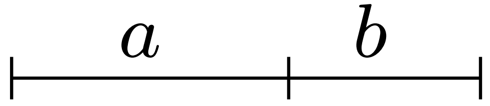
Goal: show $D(n) \leq \lfloor \log_\phi n \rfloor$ where $\phi = \frac{1+\sqrt{5}}{2}$ the golden ratio.

Lemma 1: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Let $y_1, \ldots, y_k$ be the children of x in order of age.
Then degree$[y_1] \geq 0$ and degree$[y_i] \geq i - 2$ for $i = 2, \ldots, k$.

Lemma 2: For $k \geq 0$ holds $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$.

Lemma 3: For $k \geq 0$ holds $F_{k+2} \geq \phi^k$.

Proof: by Induction(k)
$k = 0 : F_2 = 1 = \phi^0$, $k = 1 : F_3 = 2 > \phi^1$
$k > 2 : F_{k+2} = F_{k+1} + F_k \underset{\text{ind. hyp.}}{\geq} \phi^{k-1} + \phi^{k-2} = \phi^{k-2}\underbrace{(\phi + 1)}_{= \phi^2} = \phi^k$

Recall:


$\phi = \frac{a}{b} = \frac{a+b}{a} = 1 + \frac{1}{\phi}$
$\Rightarrow \phi^2 = 1 + \phi$

Bounding the maximal degree

Goal: show $D(n) \leq \lfloor \log_\phi n \rfloor$ where $\phi = \frac{1+\sqrt{5}}{2}$ the golden ratio.

Lemma 1: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Let $y_1, \ldots, y_k$ be the children of x in order of age.
Then degree$[y_1] \geq 0$ and degree$[y_i] \geq i - 2$ for $i = 2, \ldots, k$.

Lemma 2: For $k \geq 0$ holds $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$.

Lemma 3: For $k \geq 0$ holds $F_{k+2} \geq \phi^k$.

Lemma 4: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Then the subtree at x has size at least F_{k+2}.

Bounding the maximal degree

Lemma 4: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Then size(x), the size of the subtree at x, is at least F_{k+2}.

Proof: Let s_k be the minimal size of a node of degree k in a Fib-heap.

Bounding the maximal degree

Lemma 4: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Then size(x), the size of the subtree at x, is at least F_{k+2}.
Proof: Let s_k be the minimal size of a node of degree k in a Fib-heap.

Q: What is s_0 and s_1?

Bounding the maximal degree

Lemma 4: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Then size(x), the size of the subtree at x, is at least F_{k+2}.

Proof: Let s_k be the minimal size of a node of degree k in a Fib-heap.

Then $s_0 = 1$ and $s_1 = 2$ and s_k grows in k.

Bounding the maximal degree

Lemma 4: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Then size(x), the size of the subtree at x, is at least F_{k+2}.
Proof: Let s_k be the minimal size of a node of degree k in a Fib-heap.

Then $s_0 = 1$ and $s_1 = 2$ and s_k grows in k.

Now let z be a node of degree k and size s_k, and let $y_1, \ldots, y_k$ be the ordered children of z. Then size$(x) \geq s_k \underset{z,\, y_1}{\geq} 2 + \sum_{i=2}^{k} s_{deg(y_i)} \underset{\text{Lm 1}}{\geq} 2 + \sum_{i=2}^{k} s_{i-2}$.

Bounding the maximal degree

Lemma 4: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Then size(x), the size of the subtree at x, is at least F_{k+2}.
Proof: Let s_k be the minimal size of a node of degree k in a Fib-heap.

Then $s_0 = 1$ and $s_1 = 2$ and s_k grows in k.

Now let z be a node of degree k and size s_k, and let $y_1, \ldots, y_k$ be the ordered children of z. Then size$(x) \geq s_k \underset{z,\, y_1}{\geq} 2 + \sum_{i=2}^{k} s_{deg(y_i)} \underset{\text{Lm 1}}{\geq} 2 + \sum_{i=2}^{k} s_{i-2}$.

By induction(k) we show: $s_k \geq F_{k+2}$.

Bounding the maximal degree

Lemma 4: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Then size(x), the size of the subtree at x, is at least F_{k+2}.
Proof: Let s_k be the minimal size of a node of degree k in a Fib-heap.

Then $s_0 = 1$ and $s_1 = 2$ and s_k grows in k.

Now let z be a node of degree k and size s_k, and let $y_1, \ldots, y_k$ be the ordered children of z. Then size$(x) \geq s_k \underset{z,\, y_1}{\geq} 2 + \sum_{i=2}^{k} s_{deg(y_i)} \underset{\text{Lm 1}}{\geq} 2 + \sum_{i=2}^{k} s_{i-2}$.

By induction(k) we show: $s_k \geq F_{k+2}$.

$s_0 = 1 = F_2$ and $s_1 = 2 = F_3$

Bounding the maximal degree

Lemma 4: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Then size(x), the size of the subtree at x, is at least F_{k+2}.

Proof: Let s_k be the minimal size of a node of degree k in a Fib-heap.

Then $s_0 = 1$ and $s_1 = 2$ and s_k grows in k.

Now let z be a node of degree k and size s_k, and let $y_1, \ldots, y_k$ be the ordered children of z. Then size$(x) \geq s_k \underset{z,\, y_1}{\geq} 2 + \sum_{i=2}^{k} s_{deg(y_i)} \underset{\text{Lm 1}}{\geq} 2 + \sum_{i=2}^{k} s_{i-2}$.

By induction(k) we show: $s_k \geq F_{k+2}$.

$s_0 = 1 = F_2$ and $s_1 = 2 = F_3$

For $k \geq 2$ we have

$$s_k = 2 + \sum_{i=2}^{k} s_{i-2} \underset{\text{ind. hyp.}}{\geq} 2 + \sum_{i=2}^{k} F_i = 1 + \sum_{i=0}^{k} F_i \underset{\text{Lm 2}}{=} F_{k+2}$$

Bounding the maximal degree

Goal: show $D(n) \leq \lfloor \log_\phi n \rfloor$ where $\phi = \frac{1+\sqrt{5}}{2}$ the golden ratio.

Lemma 1: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Let $y_1, \ldots, y_k$ be the children of x in order of age.
Then degree$[y_1] \geq 0$ and degree$[y_i] \geq i - 2$ for $i = 2, \ldots, k$.

Lemma 2: For $k \geq 0$ holds $F_{k+2} = 1 + \sum_{i=0}^{k} F_i$.

Lemma 3: For $k \geq 0$ holds $F_{k+2} \geq \phi^k$.

Lemma 4: Let x be an arbitrary node of degree k in a Fibonacci Heap H.
Then the subtree at x has size at least F_{k+2}.

Corollary 5: $D(n) = O(\log n)$

Comparison of Runtimes

	Binary Heap	Binomial Heap lazy union	Fibonacci Heap
make	$O(1)$	$O(1)$	$O(1)$
min	$O(1)$	$O(1)$	$O(1)$
insert	$O(\log n)$	$O(1)$	$O(1)$
union	$O(n)$	$O(1)$	$O(1)$
deleteMin	$O(\log n)$	$O(\log n)^*$	$O(\log n)^*$
decKey	$O(\log n)$	$O(\log n)$	$O(1)^*$

* amortised