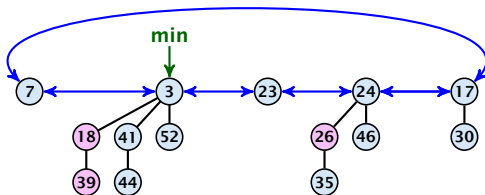


## 8.3 Fibonacci Heaps

Collection of trees that fulfill the heap property.

Structure is much more relaxed than binomial heaps.



## 8.3 Fibonacci Heaps

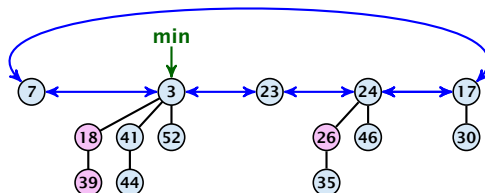
### Additional implementation details:

- ▶ Every node  $x$  stores its degree in a field  $x.degree$ . Note that this can be updated in constant time when adding a child to  $x$ .
- ▶ Every node stores a boolean value  $x.marked$  that specifies whether  $x$  is **marked** or not.

## 8.3 Fibonacci Heaps

### The potential function:

- ▶  $t(S)$  denotes the number of trees in the heap.
- ▶  $m(S)$  denotes the number of marked nodes.
- ▶ We use the potential function  $\Phi(S) = t(S) + 2m(S)$ .



The potential is  $\Phi(S) = 5 + 2 \cdot 3 = 11$ .

## 8.3 Fibonacci Heaps

We assume that one unit of potential can pay for a constant amount of work, where the constant is chosen “big enough” (to take care of the constants that occur).

To make this more explicit we use  $c$  to denote the amount of work that a unit of potential can pay for.

## 8.3 Fibonacci Heaps

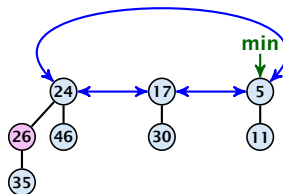
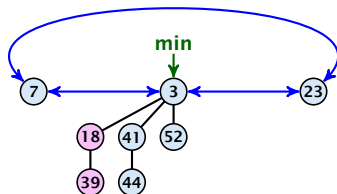
### **S. minimum()**

- ▶ Access through the min-pointer.
- ▶ Actual cost  $\mathcal{O}(1)$ .
- ▶ No change in potential.
- ▶ Amortized cost  $\mathcal{O}(1)$ .

## 8.3 Fibonacci Heaps

### $S.$ merge( $S'$ )

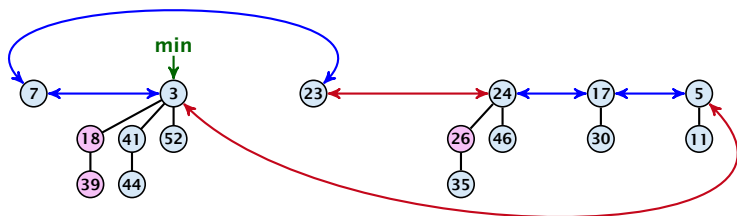
- ▶ Merge the root lists.
- ▶ Adjust the min-pointer



## 8.3 Fibonacci Heaps

### $S.$ merge( $S'$ )

- ▶ Merge the root lists.
- ▶ Adjust the min-pointer



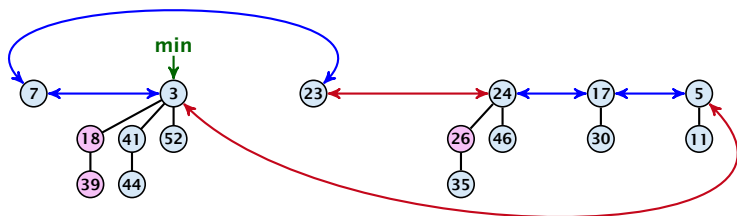
### Running time:

- ▶ Actual cost  $\mathcal{O}(1)$ .

## 8.3 Fibonacci Heaps

### $S.$ merge( $S'$ )

- ▶ Merge the root lists.
- ▶ Adjust the min-pointer



### Running time:

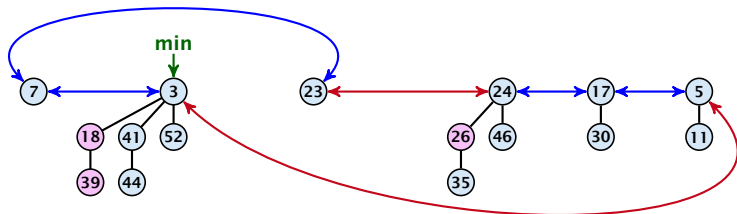
- ▶ Actual cost  $\mathcal{O}(1)$ .
- ▶ No change in potential.



## 8.3 Fibonacci Heaps

### $S.$ merge( $S'$ )

- ▶ Merge the root lists.
- ▶ Adjust the min-pointer



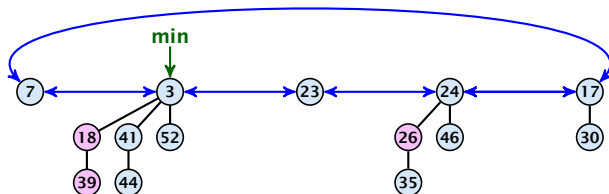
### Running time:

- ▶ Actual cost  $\mathcal{O}(1)$ .
- ▶ No change in potential.
- ▶ Hence, amortized cost is  $\mathcal{O}(1)$ .

## 8.3 Fibonacci Heaps

### $S.\text{insert}(x)$

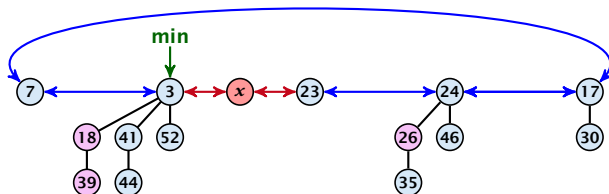
- ▶ Create a new tree containing  $x$ .
- ▶ Insert  $x$  into the root-list.
- ▶ Update min-pointer, if necessary.



## 8.3 Fibonacci Heaps

### $S.\text{insert}(x)$

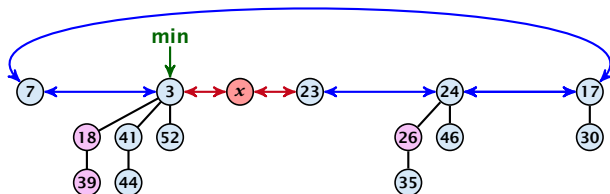
- ▶ Create a new tree containing  $x$ .
- ▶ Insert  $x$  into the root-list.
- ▶ Update min-pointer, if necessary.



## 8.3 Fibonacci Heaps

### $S.\text{insert}(x)$

- ▶ Create a new tree containing  $x$ .
- ▶ Insert  $x$  into the root-list.
- ▶ Update min-pointer, if necessary.

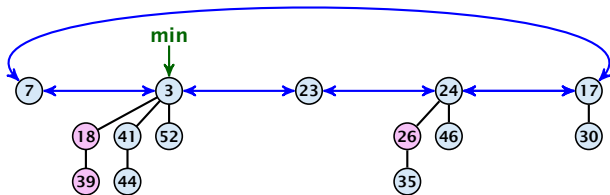


### Running time:

- ▶ Actual cost  $\mathcal{O}(1)$ .
- ▶ Change in potential is  $+1$ .
- ▶ Amortized cost is  $c + \mathcal{O}(1) = \mathcal{O}(1)$ .

## 8.3 Fibonacci Heaps

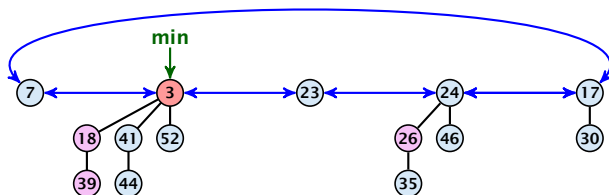
S. delete-min( $x$ )



## 8.3 Fibonacci Heaps

### S. delete-min( $x$ )

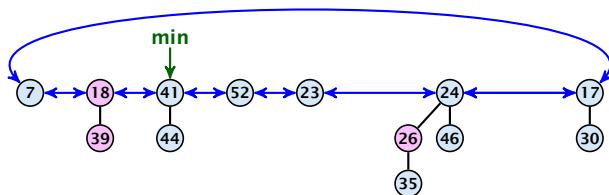
- ▶ Delete minimum; add child-trees to heap;  
time:  $D(\min) \cdot \mathcal{O}(1)$ .



## 8.3 Fibonacci Heaps

### S. delete-min( $x$ )

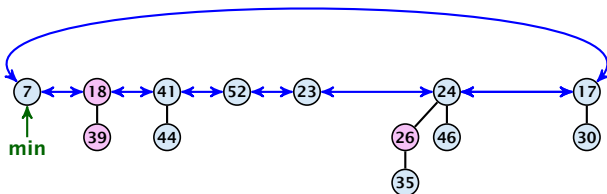
- ▶ Delete minimum; add child-trees to heap;  
time:  $D(\min) \cdot \mathcal{O}(1)$ .
- ▶ Update min-pointer; time:  $(t + D(\min)) \cdot \mathcal{O}(1)$ .



## 8.3 Fibonacci Heaps

### S. delete-min( $x$ )

- ▶ Delete minimum; add child-trees to heap;  
time:  $D(\min) \cdot \mathcal{O}(1)$ .
- ▶ Update min-pointer; time:  $(t + D(\min)) \cdot \mathcal{O}(1)$ .

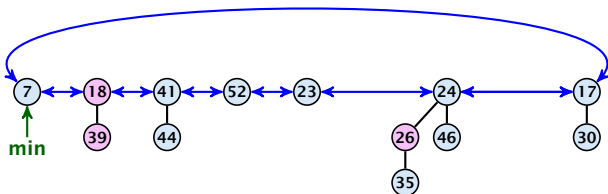




## 8.3 Fibonacci Heaps

### S. delete-min( $x$ )

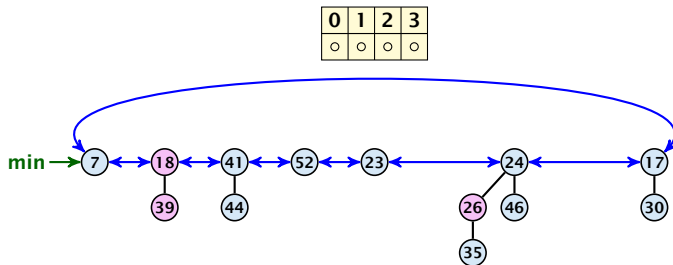
- ▶ Delete minimum; add child-trees to heap;  
time:  $D(\min) \cdot \mathcal{O}(1)$ .
- ▶ Update min-pointer; time:  $(t + D(\min)) \cdot \mathcal{O}(1)$ .



- ▶ Consolidate root-list so that no roots have the same degree.  
Time  $t \cdot \mathcal{O}(1)$  (see next slide).

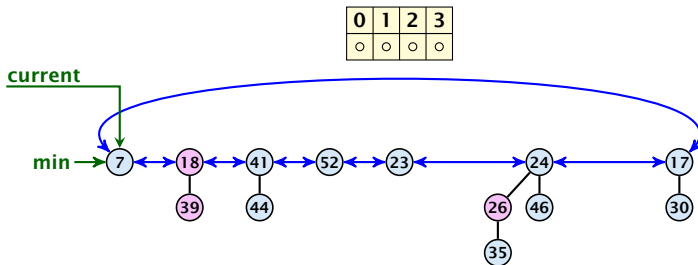
## 8.3 Fibonacci Heaps

Consolidate:



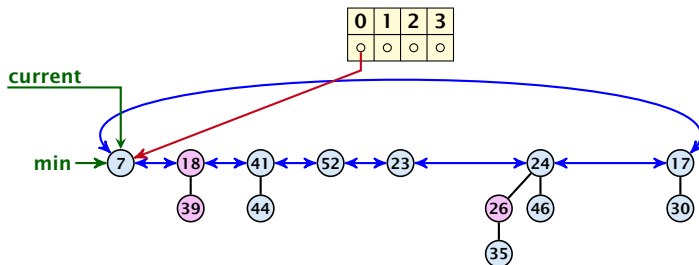
## 8.3 Fibonacci Heaps

Consolidate:



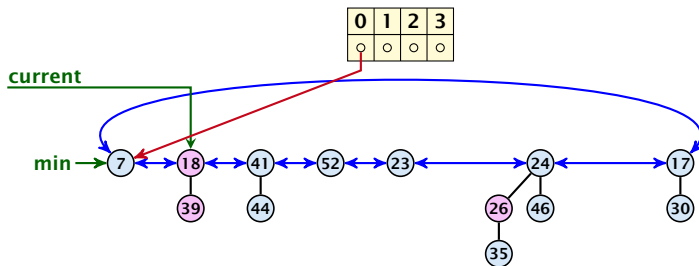
## 8.3 Fibonacci Heaps

Consolidate:



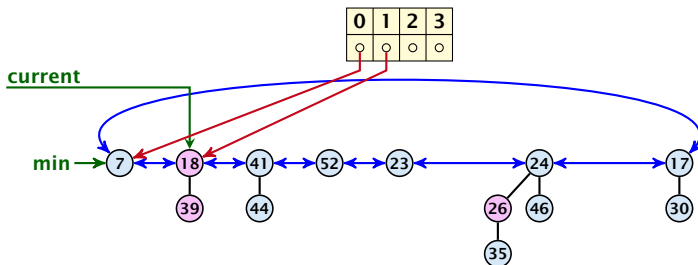
## 8.3 Fibonacci Heaps

Consolidate:



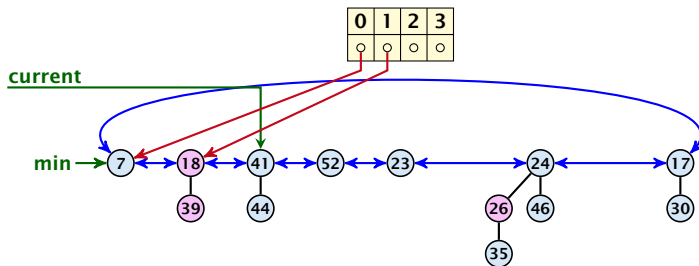
## 8.3 Fibonacci Heaps

Consolidate:



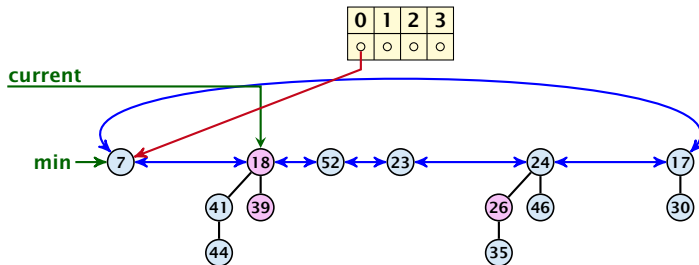
## 8.3 Fibonacci Heaps

Consolidate:



## 8.3 Fibonacci Heaps

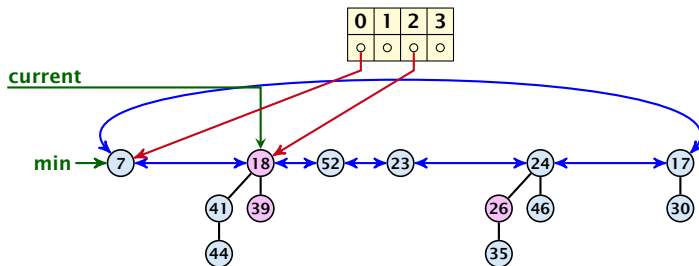
Consolidate:





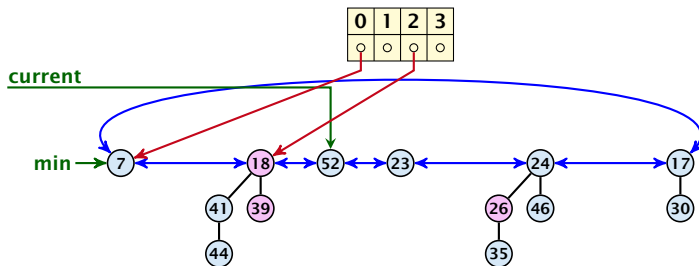
## 8.3 Fibonacci Heaps

Consolidate:



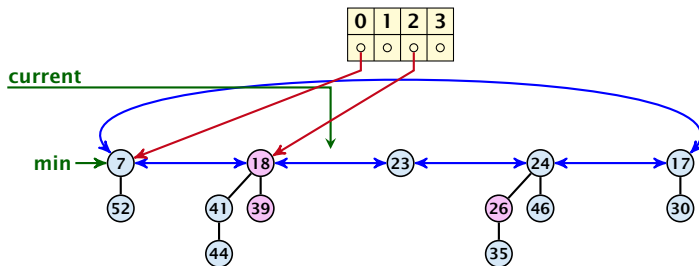
## 8.3 Fibonacci Heaps

Consolidate:



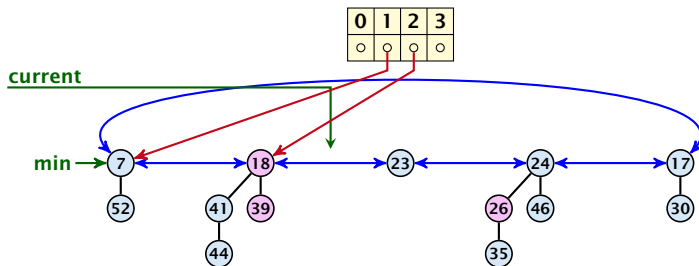
## 8.3 Fibonacci Heaps

Consolidate:



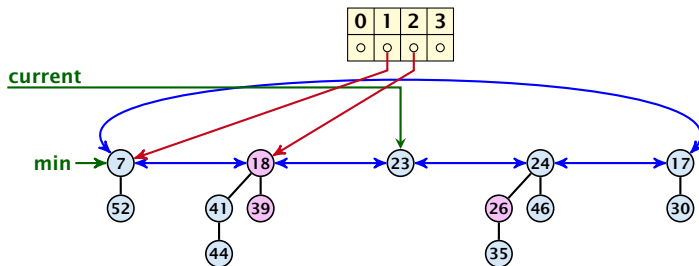
## 8.3 Fibonacci Heaps

Consolidate:



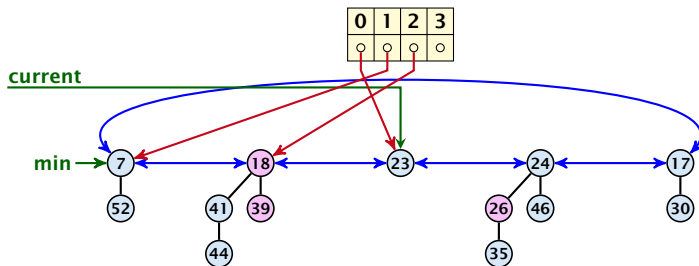
## 8.3 Fibonacci Heaps

Consolidate:



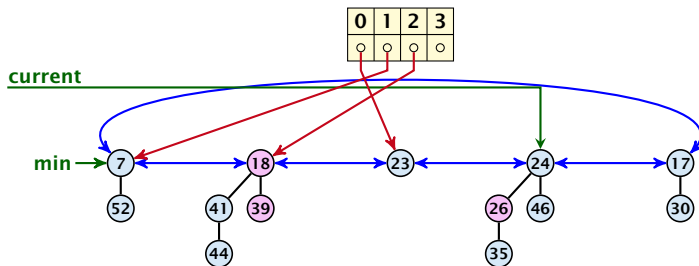
## 8.3 Fibonacci Heaps

Consolidate:



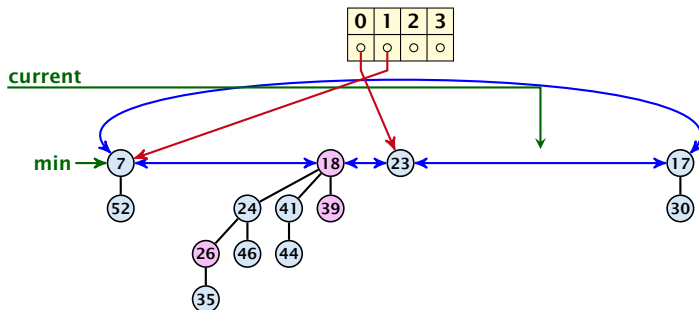
## 8.3 Fibonacci Heaps

Consolidate:



## 8.3 Fibonacci Heaps

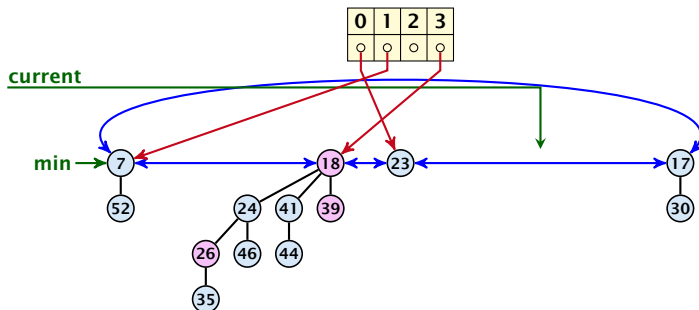
Consolidate:





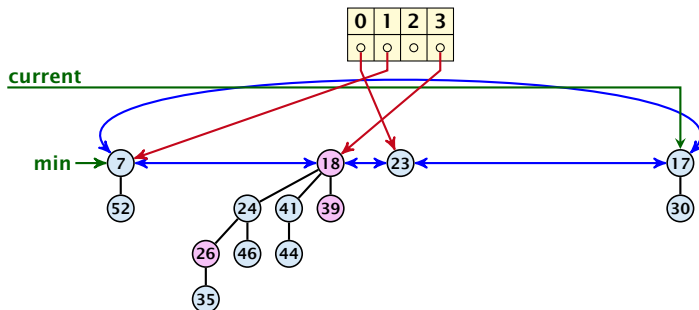
## 8.3 Fibonacci Heaps

Consolidate:



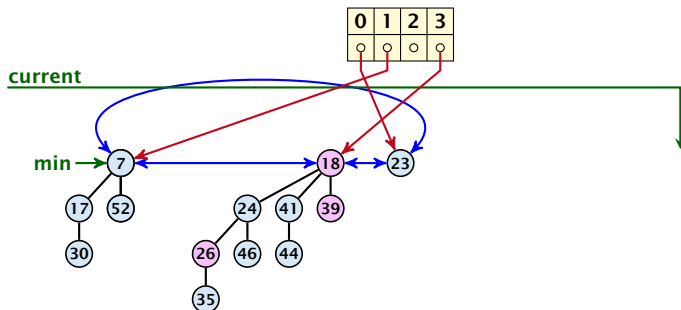
## 8.3 Fibonacci Heaps

Consolidate:



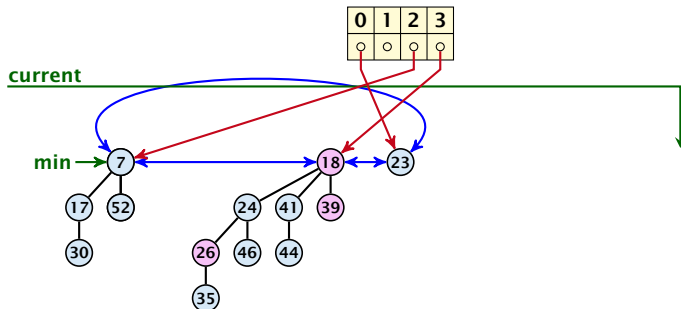
## 8.3 Fibonacci Heaps

Consolidate:



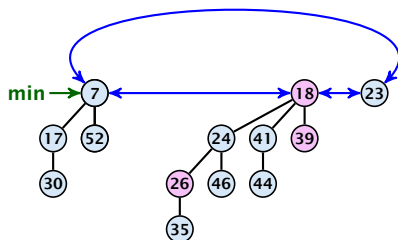
## 8.3 Fibonacci Heaps

Consolidate:



## 8.3 Fibonacci Heaps

**Consolidate:**



## 8.3 Fibonacci Heaps

### Actual cost for delete-min()

- ▶ At most  $D_n + t$  elements in root-list before consolidate.

## 8.3 Fibonacci Heaps

### Actual cost for delete-min()

- ▶ At most  $D_n + t$  elements in root-list before consolidate.
- ▶ Actual cost for a delete-min is at most  $\mathcal{O}(1) \cdot (D_n + t)$ .  
Hence, there exists  $c_1$  s.t. actual cost is at most  $c_1 \cdot (D_n + t)$ .

## 8.3 Fibonacci Heaps

### Actual cost for delete-min()

- ▶ At most  $D_n + t$  elements in root-list before consolidate.
- ▶ Actual cost for a delete-min is at most  $\mathcal{O}(1) \cdot (D_n + t)$ .  
Hence, there exists  $c_1$  s.t. actual cost is at most  $c_1 \cdot (D_n + t)$ .

### Amortized cost for delete-min()

- ▶  $t' \leq D_n + 1$  as degrees are different after consolidating.



## 8.3 Fibonacci Heaps

### Actual cost for delete-min()

- ▶ At most  $D_n + t$  elements in root-list before consolidate.
- ▶ Actual cost for a delete-min is at most  $\mathcal{O}(1) \cdot (D_n + t)$ .  
Hence, there exists  $c_1$  s.t. actual cost is at most  $c_1 \cdot (D_n + t)$ .

### Amortized cost for delete-min()

- ▶  $t' \leq D_n + 1$  as degrees are different after consolidating.
- ▶ Therefore  $\Delta\Phi \leq D_n + 1 - t$ ;

## 8.3 Fibonacci Heaps

### Actual cost for delete-min()

- ▶ At most  $D_n + t$  elements in root-list before consolidate.
- ▶ Actual cost for a delete-min is at most  $\mathcal{O}(1) \cdot (D_n + t)$ .  
Hence, there exists  $c_1$  s.t. actual cost is at most  $c_1 \cdot (D_n + t)$ .

### Amortized cost for delete-min()

- ▶  $t' \leq D_n + 1$  as degrees are different after consolidating.
- ▶ Therefore  $\Delta\Phi \leq D_n + 1 - t$ ;
- ▶ We can pay  $c \cdot (t - D_n - 1)$  from the potential decrease.

## 8.3 Fibonacci Heaps

### Actual cost for delete-min()

- ▶ At most  $D_n + t$  elements in root-list before consolidate.
- ▶ Actual cost for a delete-min is at most  $\mathcal{O}(1) \cdot (D_n + t)$ .  
Hence, there exists  $c_1$  s.t. actual cost is at most  $c_1 \cdot (D_n + t)$ .

### Amortized cost for delete-min()

- ▶  $t' \leq D_n + 1$  as degrees are different after consolidating.
- ▶ Therefore  $\Delta\Phi \leq D_n + 1 - t$ ;
- ▶ We can pay  $c \cdot (t - D_n - 1)$  from the potential decrease.
- ▶ The amortized cost is

## 8.3 Fibonacci Heaps

### Actual cost for delete-min()

- ▶ At most  $D_n + t$  elements in root-list before consolidate.
- ▶ Actual cost for a delete-min is at most  $\mathcal{O}(1) \cdot (D_n + t)$ .  
Hence, there exists  $c_1$  s.t. actual cost is at most  $c_1 \cdot (D_n + t)$ .

### Amortized cost for delete-min()

- ▶  $t' \leq D_n + 1$  as degrees are different after consolidating.
- ▶ Therefore  $\Delta\Phi \leq D_n + 1 - t$ ;
- ▶ We can pay  $c \cdot (t - D_n - 1)$  from the potential decrease.
- ▶ The amortized cost is

$$c_1 \cdot (D_n + t) - c \cdot (t - D_n - 1)$$

## 8.3 Fibonacci Heaps

### Actual cost for delete-min()

- ▶ At most  $D_n + t$  elements in root-list before consolidate.
- ▶ Actual cost for a delete-min is at most  $\mathcal{O}(1) \cdot (D_n + t)$ .  
Hence, there exists  $c_1$  s.t. actual cost is at most  $c_1 \cdot (D_n + t)$ .

### Amortized cost for delete-min()

- ▶  $t' \leq D_n + 1$  as degrees are different after consolidating.
- ▶ Therefore  $\Delta\Phi \leq D_n + 1 - t$ ;
- ▶ We can pay  $c \cdot (t - D_n - 1)$  from the potential decrease.
- ▶ The amortized cost is

$$\begin{aligned} c_1 \cdot (D_n + t) - c \cdot (t - D_n - 1) \\ \leq (c_1 + c)D_n + (c_1 - c)t + c \end{aligned}$$

## 8.3 Fibonacci Heaps

### Actual cost for delete-min()

- ▶ At most  $D_n + t$  elements in root-list before consolidate.
- ▶ Actual cost for a delete-min is at most  $\mathcal{O}(1) \cdot (D_n + t)$ .  
Hence, there exists  $c_1$  s.t. actual cost is at most  $c_1 \cdot (D_n + t)$ .

### Amortized cost for delete-min()

- ▶  $t' \leq D_n + 1$  as degrees are different after consolidating.
- ▶ Therefore  $\Delta\Phi \leq D_n + 1 - t$ ;
- ▶ We can pay  $c \cdot (t - D_n - 1)$  from the potential decrease.
- ▶ The amortized cost is

$$\begin{aligned} c_1 \cdot (D_n + t) - c \cdot (t - D_n - 1) \\ \leq (c_1 + c)D_n + (c_1 - c)t + c \leq 2c(D_n + 1) \end{aligned}$$

## 8.3 Fibonacci Heaps

### Actual cost for delete-min()

- ▶ At most  $D_n + t$  elements in root-list before consolidate.
- ▶ Actual cost for a delete-min is at most  $\mathcal{O}(1) \cdot (D_n + t)$ .  
Hence, there exists  $c_1$  s.t. actual cost is at most  $c_1 \cdot (D_n + t)$ .

### Amortized cost for delete-min()

- ▶  $t' \leq D_n + 1$  as degrees are different after consolidating.
- ▶ Therefore  $\Delta\Phi \leq D_n + 1 - t$ ;
- ▶ We can pay  $c \cdot (t - D_n - 1)$  from the potential decrease.
- ▶ The amortized cost is

$$\begin{aligned} c_1 \cdot (D_n + t) - c \cdot (t - D_n - 1) \\ \leq (c_1 + c)D_n + (c_1 - c)t + c \leq 2c(D_n + 1) \leq \mathcal{O}(D_n) \end{aligned}$$

## 8.3 Fibonacci Heaps

### Actual cost for delete-min()

- ▶ At most  $D_n + t$  elements in root-list before consolidate.
- ▶ Actual cost for a delete-min is at most  $\mathcal{O}(1) \cdot (D_n + t)$ .  
Hence, there exists  $c_1$  s.t. actual cost is at most  $c_1 \cdot (D_n + t)$ .

### Amortized cost for delete-min()

- ▶  $t' \leq D_n + 1$  as degrees are different after consolidating.
- ▶ Therefore  $\Delta\Phi \leq D_n + 1 - t$ ;
- ▶ We can pay  $c \cdot (t - D_n - 1)$  from the potential decrease.
- ▶ The amortized cost is

$$\begin{aligned} c_1 \cdot (D_n + t) - c \cdot (t - D_n - 1) \\ \leq (c_1 + c)D_n + (c_1 - c)t + c \leq 2c(D_n + 1) \leq \mathcal{O}(D_n) \end{aligned}$$

for  $c \geq c_1$  .



## 8.3 Fibonacci Heaps

If the input trees of the consolidation procedure are binomial trees (for example only singleton vertices) then the output will be a set of distinct binomial trees, and, hence, the Fibonacci heap will be (more or less) a Binomial heap right after the consolidation.

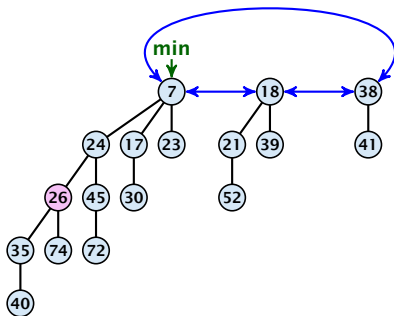
If we do not have delete or decrease-key operations then  
 $D_n \leq \log n$ .

## 8.3 Fibonacci Heaps

If the input trees of the consolidation procedure are binomial trees (for example only singleton vertices) then the output will be a set of distinct binomial trees, and, hence, the Fibonacci heap will be (more or less) a Binomial heap right after the consolidation.

If we do not have delete or decrease-key operations then  
 $D_n \leq \log n$ .

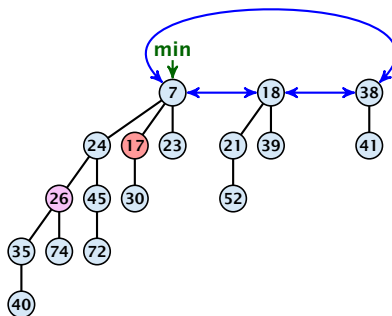
## Fibonacci Heaps: decrease-key(handle $h, v$ )



### Case 1: decrease-key does not violate heap-property

- ▶ Just decrease the key-value of element referenced by  $h$ . Nothing else to do.

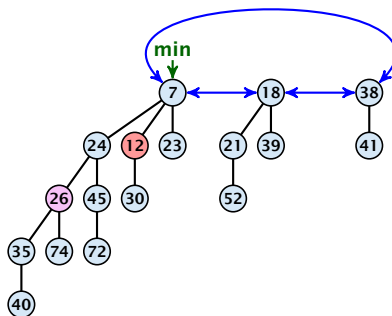
# Fibonacci Heaps: decrease-key(handle $h, v$ )



## Case 1: decrease-key does not violate heap-property

- ▶ Just decrease the key-value of element referenced by  $h$ . Nothing else to do.

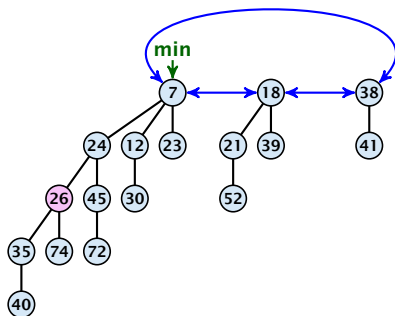
# Fibonacci Heaps: decrease-key(handle $h, v$ )



## Case 1: decrease-key does not violate heap-property

- ▶ Just decrease the key-value of element referenced by  $h$ . Nothing else to do.

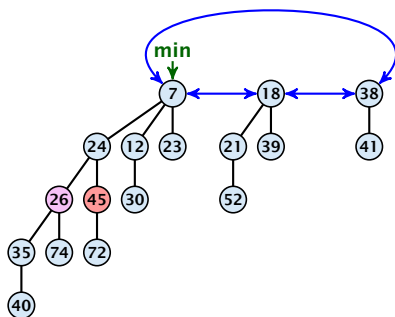
# Fibonacci Heaps: decrease-key(handle $h, v$ )



## Case 1: decrease-key does not violate heap-property

- ▶ Just decrease the key-value of element referenced by  $h$ . Nothing else to do.

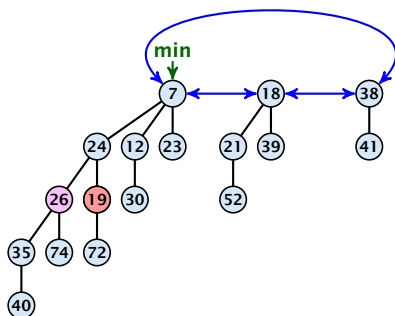
# Fibonacci Heaps: decrease-key(handle $h, v$ )



## Case 2: heap-property is violated, but parent is not marked

- ▶ Decrease key-value of element  $x$  reference by  $h$ .
- ▶ If the heap-property is violated, cut the parent edge of  $x$ , and make  $x$  into a root.
- ▶ Adjust min-pointers, if necessary.
- ▶ Mark the (previous) parent of  $x$  (unless it's a root).

# Fibonacci Heaps: decrease-key(handle $h, v$ )

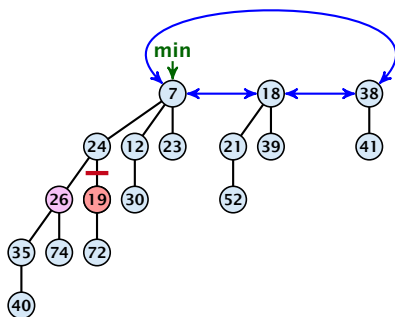


## Case 2: heap-property is violated, but parent is not marked

- ▶ Decrease key-value of element  $x$  reference by  $h$ .
- ▶ If the heap-property is violated, cut the parent edge of  $x$ , and make  $x$  into a root.
- ▶ Adjust min-pointers, if necessary.
- ▶ Mark the (previous) parent of  $x$  (unless it's a root).



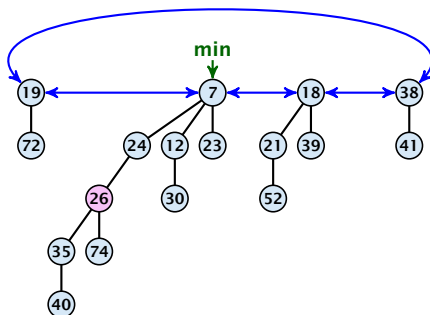
# Fibonacci Heaps: decrease-key(handle $h, v$ )



## Case 2: heap-property is violated, but parent is not marked

- ▶ Decrease key-value of element  $x$  reference by  $h$ .
- ▶ If the heap-property is violated, cut the parent edge of  $x$ , and make  $x$  into a root.
- ▶ Adjust min-pointers, if necessary.
- ▶ Mark the (previous) parent of  $x$  (unless it's a root).

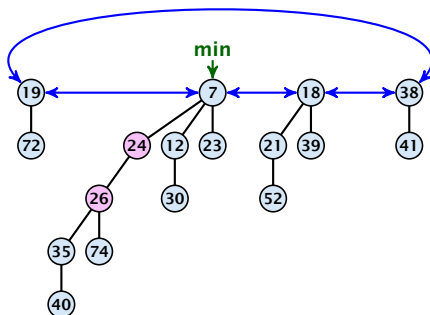
# Fibonacci Heaps: decrease-key(handle $h, v$ )



## Case 2: heap-property is violated, but parent is not marked

- ▶ Decrease key-value of element  $x$  reference by  $h$ .
- ▶ If the heap-property is violated, cut the parent edge of  $x$ , and make  $x$  into a root.
- ▶ Adjust min-pointers, if necessary.
- ▶ Mark the (previous) parent of  $x$  (unless it's a root).

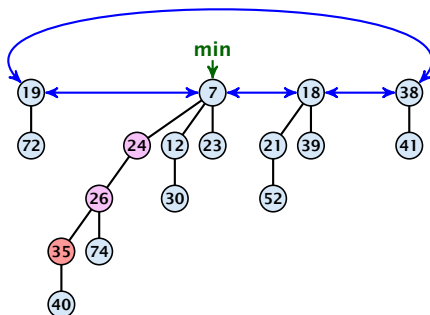
# Fibonacci Heaps: decrease-key(handle $h, v$ )



## Case 2: heap-property is violated, but parent is not marked

- ▶ Decrease key-value of element  $x$  reference by  $h$ .
- ▶ If the heap-property is violated, cut the parent edge of  $x$ , and make  $x$  into a root.
- ▶ Adjust min-pointers, if necessary.
- ▶ Mark the (previous) parent of  $x$  (unless it's a root).

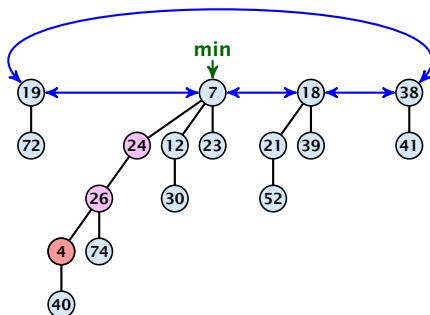
# Fibonacci Heaps: decrease-key(handle $h, v$ )



## Case 3: heap-property is violated, and parent is marked

- ▶ Decrease key-value of element  $x$  reference by  $h$ .
- ▶ Cut the parent edge of  $x$ , and make  $x$  into a root.
- ▶ Adjust min-pointers, if necessary.
- ▶ Continue cutting the parent until you arrive at an unmarked node.

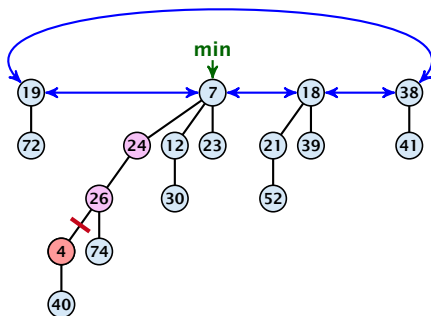
# Fibonacci Heaps: decrease-key(handle $h, v$ )



## Case 3: heap-property is violated, and parent is marked

- ▶ Decrease key-value of element  $x$  reference by  $h$ .
- ▶ Cut the parent edge of  $x$ , and make  $x$  into a root.
- ▶ Adjust min-pointers, if necessary.
- ▶ Continue cutting the parent until you arrive at an unmarked node.

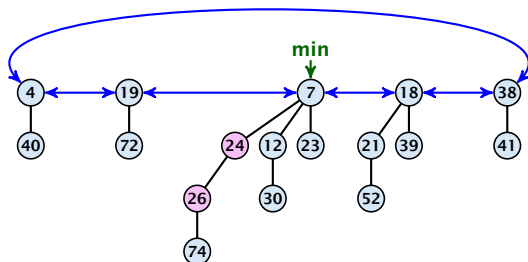
# Fibonacci Heaps: decrease-key(handle $h, v$ )



## Case 3: heap-property is violated, and parent is marked

- ▶ Decrease key-value of element  $x$  reference by  $h$ .
- ▶ Cut the parent edge of  $x$ , and make  $x$  into a root.
- ▶ Adjust min-pointers, if necessary.
- ▶ Continue cutting the parent until you arrive at an unmarked node.

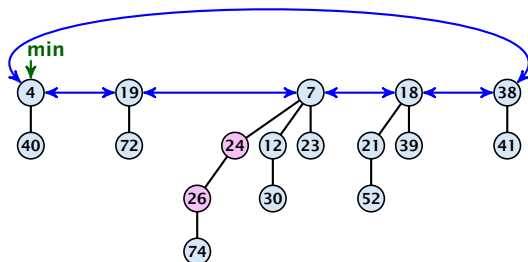
# Fibonacci Heaps: decrease-key(handle $h, v$ )



## Case 3: heap-property is violated, and parent is marked

- ▶ Decrease key-value of element  $x$  reference by  $h$ .
- ▶ Cut the parent edge of  $x$ , and make  $x$  into a root.
- ▶ Adjust min-pointers, if necessary.
- ▶ Continue cutting the parent until you arrive at an unmarked node.

# Fibonacci Heaps: decrease-key(handle $h, v$ )

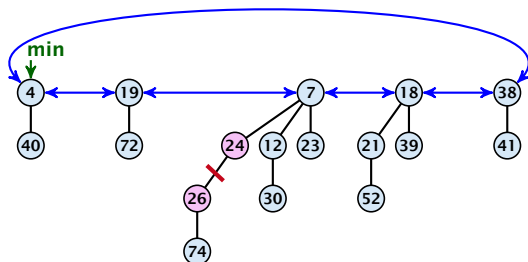


## Case 3: heap-property is violated, and parent is marked

- ▶ Decrease key-value of element  $x$  reference by  $h$ .
- ▶ Cut the parent edge of  $x$ , and make  $x$  into a root.
- ▶ Adjust min-pointers, if necessary.
- ▶ Continue cutting the parent until you arrive at an unmarked node.



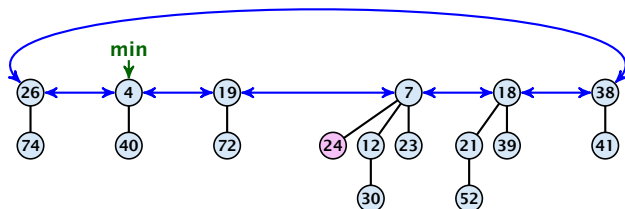
# Fibonacci Heaps: decrease-key(handle $h, v$ )



## Case 3: heap-property is violated, and parent is marked

- ▶ Decrease key-value of element  $x$  reference by  $h$ .
- ▶ Cut the parent edge of  $x$ , and make  $x$  into a root.
- ▶ Adjust min-pointers, if necessary.
- ▶ Continue cutting the parent until you arrive at an unmarked node.

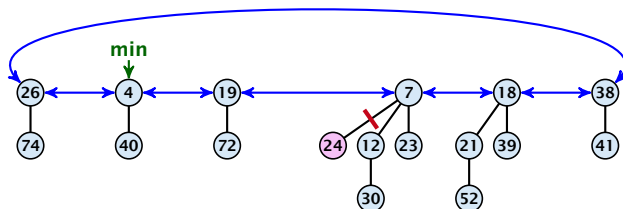
# Fibonacci Heaps: decrease-key(handle $h, v$ )



## Case 3: heap-property is violated, and parent is marked

- ▶ Decrease key-value of element  $x$  reference by  $h$ .
- ▶ Cut the parent edge of  $x$ , and make  $x$  into a root.
- ▶ Adjust min-pointers, if necessary.
- ▶ Continue cutting the parent until you arrive at an unmarked node.

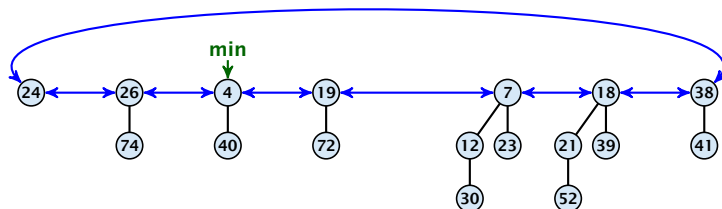
# Fibonacci Heaps: decrease-key(handle $h, v$ )



## Case 3: heap-property is violated, and parent is marked

- ▶ Decrease key-value of element  $x$  reference by  $h$ .
- ▶ Cut the parent edge of  $x$ , and make  $x$  into a root.
- ▶ Adjust min-pointers, if necessary.
- ▶ Continue cutting the parent until you arrive at an unmarked node.

# Fibonacci Heaps: decrease-key(handle $h, v$ )



## Case 3: heap-property is violated, and parent is marked

- ▶ Decrease key-value of element  $x$  reference by  $h$ .
- ▶ Cut the parent edge of  $x$ , and make  $x$  into a root.
- ▶ Adjust min-pointers, if necessary.
- ▶ Continue cutting the parent until you arrive at an unmarked node.

# Fibonacci Heaps: decrease-key(handle $h, v$ )

## Case 3: heap-property is violated, and parent is marked

- ▶ Decrease key-value of element  $x$  reference by  $h$ .
- ▶ Cut the parent edge of  $x$ , and make  $x$  into a root.
- ▶ Adjust min-pointers, if necessary.
- ▶ Execute the following:

```
 $p \leftarrow \text{parent}[x];$   
while ( $p$  is marked)  
     $pp \leftarrow \text{parent}[p];$   
    cut of  $p$ ; make it into a root; unmark it;  
     $p \leftarrow pp;$   
if  $p$  is unmarked and not a root mark it;
```

# Fibonacci Heaps: decrease-key(handle $h, v$ )

## Actual cost:

- ▶ Constant cost for decreasing the value.
- ▶ Constant cost for each of  $\ell$  cuts.
- ▶ Hence, cost is at most  $c_2 \cdot (\ell + 1)$ , for some constant  $c_2$ .

## Amortized cost:

- ▶  $\ell$  is at most  $\log n$ , as every cut creates one new root.
- ▶  $\ell$  is at most  $\log n$  since  $\ell$  is at most  $\log n$ , since all but the first cut marks a node; the last cut may mark a node.

- ▶ Amortized cost is at most

$$O(\log n)$$

# Fibonacci Heaps: decrease-key(handle $h, v$ )

## Actual cost:

- ▶ Constant cost for decreasing the value.
- ▶ Constant cost for each of  $\ell$  cuts.
- ▶ Hence, cost is at most  $c_2 \cdot (\ell + 1)$ , for some constant  $c_2$ .

## Amortized cost:

- ▶ Every cut creates one new root.
- ▶ Every node is marked at most once, since all but the first cut on a node marks a node; the last cut may mark a node.
- ▶ Hence, the amortized cost is at most  $c_1 + c_2$ .

# Fibonacci Heaps: decrease-key(handle $h, v$ )

## Actual cost:

- ▶ Constant cost for decreasing the value.
- ▶ Constant cost for each of  $\ell$  cuts.
- ▶ Hence, cost is at most  $c_2 \cdot (\ell + 1)$ , for some constant  $c_2$ .

## Amortized cost:

- ▶ Every cut creates one new root.
- ▶ Every root has at most  $\Delta$  children, since all but the first cut child are marked.
- ▶ Every marked node has at most one child.
- ▶ Every root has at most  $\Delta$  unmarked children.
- ▶ Amortized cost is at most  $\Delta + 1$ .



## Fibonacci Heaps: decrease-key(handle $h$ , $v$ )

**Actual cost:**

- ▶ Constant cost for decreasing the value.
- ▶ Constant cost for each of  $\ell$  cuts.
- ▶ Hence, cost is at most  $c_2 \cdot (\ell + 1)$ , for some constant  $c_2$ .

# Fibonacci Heaps: decrease-key(handle $h, v$ )

## Actual cost:

- ▶ Constant cost for decreasing the value.
- ▶ Constant cost for each of  $\ell$  cuts.
- ▶ Hence, cost is at most  $c_2 \cdot (\ell + 1)$ , for some constant  $c_2$ .

## Amortized cost:

- ▶  $t' = t + \ell$ , as every cut creates one new root.
- ▶  $m' \leq m - (\ell - 1) + 1 = m - \ell + 2$ , since all but the first cut unmarks a node; the last cut may mark a node.
- ▶  $\Delta\Phi \leq \ell + 2(-\ell + 2) = 4 - \ell$
- ▶ Amortized cost is at most

# Fibonacci Heaps: decrease-key(handle $h, v$ )

## Actual cost:

- ▶ Constant cost for decreasing the value.
- ▶ Constant cost for each of  $\ell$  cuts.
- ▶ Hence, cost is at most  $c_2 \cdot (\ell + 1)$ , for some constant  $c_2$ .

## Amortized cost:

- ▶  $t' = t + \ell$ , as every cut creates one new root.
- ▶  $m' \leq m - (\ell - 1) + 1 = m - \ell + 2$ , since all but the first cut unmarks a node; the last cut may mark a node.
- ▶  $\Delta\Phi \leq \ell + 2(-\ell + 2) = 4 - \ell$
- ▶ Amortized cost is at most

# Fibonacci Heaps: decrease-key(handle $h, v$ )

## Actual cost:

- ▶ Constant cost for decreasing the value.
- ▶ Constant cost for each of  $\ell$  cuts.
- ▶ Hence, cost is at most  $c_2 \cdot (\ell + 1)$ , for some constant  $c_2$ .

## Amortized cost:

- ▶  $t' = t + \ell$ , as every cut creates one new root.
- ▶  $m' \leq m - (\ell - 1) + 1 = m - \ell + 2$ , since all but the first cut unmarks a node; the last cut may mark a node.
- ▶  $\Delta\Phi \leq \ell + 2(-\ell + 2) = 4 - \ell$
- ▶ Amortized cost is at most

# Fibonacci Heaps: decrease-key(handle $h, v$ )

## Actual cost:

- ▶ Constant cost for decreasing the value.
- ▶ Constant cost for each of  $\ell$  cuts.
- ▶ Hence, cost is at most  $c_2 \cdot (\ell + 1)$ , for some constant  $c_2$ .

## Amortized cost:

- ▶  $t' = t + \ell$ , as every cut creates one new root.
- ▶  $m' \leq m - (\ell - 1) + 1 = m - \ell + 2$ , since all but the first cut unmarks a node; the last cut may mark a node.
- ▶  $\Delta\Phi \leq \ell + 2(-\ell + 2) = 4 - \ell$
- ▶ Amortized cost is at most

$$c_2(\ell + 1) + c(4 - \ell) \leq (c_2 - c)\ell + 4c + c_2 = \mathcal{O}(1),$$

if  $c \geq c_2$ .

# Fibonacci Heaps: decrease-key(handle $h, v$ )

## Actual cost:

- ▶ Constant cost for decreasing the value.
- ▶ Constant cost for each of  $\ell$  cuts.
- ▶ Hence, cost is at most  $c_2 \cdot (\ell + 1)$ , for some constant  $c_2$ .

## Amortized cost:

- ▶  $t' = t + \ell$ , as every cut creates one new root.
- ▶  $m' \leq m - (\ell - 1) + 1 = m - \ell + 2$ , since all but the first cut unmarks a node; the last cut may mark a node.
- ▶  $\Delta\Phi \leq \ell + 2(-\ell + 2) = 4 - \ell$
- ▶ Amortized cost is at most

$$c_2(\ell + 1) + c(4 - \ell) \leq (c_2 - c)\ell + 4c + c_2 = \mathcal{O}(1),$$

if  $c \geq c_2$ .

# Fibonacci Heaps: decrease-key(handle $h, v$ )

## Actual cost:

- ▶ Constant cost for decreasing the value.
- ▶ Constant cost for each of  $\ell$  cuts.
- ▶ Hence, cost is at most  $c_2 \cdot (\ell + 1)$ , for some constant  $c_2$ .

## Amortized cost:

- ▶  $t' = t + \ell$ , as every cut creates one new root.
- ▶  $m' \leq m - (\ell - 1) + 1 = m - \ell + 2$ , since all but the first cut unmarks a node; the last cut may mark a node.
- ▶  $\Delta\Phi \leq \ell + 2(-\ell + 2) = 4 - \ell$
- ▶ Amortized cost is at most

$$c_2(\ell + 1) + c(4 - \ell) \leq (c_2 - c)\ell + 4c + c_2 = \mathcal{O}(1),$$

if  $c \geq c_2$ .

# Fibonacci Heaps: decrease-key(handle $h, v$ )

## Actual cost:

- ▶ Constant cost for decreasing the value.
- ▶ Constant cost for each of  $\ell$  cuts.
- ▶ Hence, cost is at most  $c_2 \cdot (\ell + 1)$ , for some constant  $c_2$ .

## Amortized cost:

- ▶  $t' = t + \ell$ , as every cut creates one new root.
- ▶  $m' \leq m - (\ell - 1) + 1 = m - \ell + 2$ , since all but the first cut unmarks a node; the last cut may mark a node.
- ▶  $\Delta\Phi \leq \ell + 2(-\ell + 2) = 4 - \ell$
- ▶ Amortized cost is at most
$$c_2(\ell + 1) + c(4 - \ell) \leq (c_2 - c)\ell + 4c + c_2 = \mathcal{O}(1),$$
if  $c \geq c_2$ .



# Delete node

***H.delete( $x$ ):***

- ▶ decrease value of  $x$  to  $-\infty$ .
- ▶ delete-min.

**Amortized cost:  $\mathcal{O}(D_n)$**

- ▶  $\mathcal{O}(1)$  for decrease-key.
- ▶  $\mathcal{O}(D_n)$  for delete-min.

## 8.3 Fibonacci Heaps

### Lemma 1

Let  $x$  be a node with degree  $k$  and let  $y_1, \dots, y_k$  denote the children of  $x$  in the order that they were linked to  $x$ . Then

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

## 8.3 Fibonacci Heaps

### Proof

- ▶ When  $y_i$  was linked to  $x$ , at least  $y_1, \dots, y_{i-1}$  were already linked to  $x$ .
- ▶ Hence, at this time  $\text{degree}(x) \geq i - 1$ , and therefore also  $\text{degree}(y_i) \geq i - 1$  as the algorithm links nodes of equal degree only.
- ▶ Since, then  $y_i$  has lost at most one child.
- ▶ Therefore,  $\text{degree}(y_i) \geq i - 2$ .

## 8.3 Fibonacci Heaps

### Proof

- ▶ When  $y_i$  was linked to  $x$ , at least  $y_1, \dots, y_{i-1}$  were already linked to  $x$ .
- ▶ Hence, at this time  $\text{degree}(x) \geq i - 1$ , and therefore also  $\text{degree}(y_i) \geq i - 1$  as the algorithm links nodes of equal degree only.
- ▶ Since, then  $y_i$  has lost at most one child.
- ▶ Therefore,  $\text{degree}(y_i) \geq i - 2$ .

## 8.3 Fibonacci Heaps

### Proof

- ▶ When  $y_i$  was linked to  $x$ , at least  $y_1, \dots, y_{i-1}$  were already linked to  $x$ .
- ▶ Hence, at this time  $\text{degree}(x) \geq i - 1$ , and therefore also  $\text{degree}(y_i) \geq i - 1$  as the algorithm links nodes of equal degree only.
- ▶ Since, then  $y_i$  has lost at most one child.
- ▶ Therefore,  $\text{degree}(y_i) \geq i - 2$ .

## 8.3 Fibonacci Heaps

### Proof

- ▶ When  $y_i$  was linked to  $x$ , at least  $y_1, \dots, y_{i-1}$  were already linked to  $x$ .
- ▶ Hence, at this time  $\text{degree}(x) \geq i - 1$ , and therefore also  $\text{degree}(y_i) \geq i - 1$  as the algorithm links nodes of equal degree only.
- ▶ Since, then  $y_i$  has lost at most one child.
- ▶ Therefore,  $\text{degree}(y_i) \geq i - 2$ .

## 8.3 Fibonacci Heaps

- ▶ Let  $s_k$  be the minimum possible size of a sub-tree rooted at a node of degree  $k$  that can occur in a Fibonacci heap.

## 8.3 Fibonacci Heaps

- ▶ Let  $s_k$  be the minimum possible size of a sub-tree rooted at a node of degree  $k$  that can occur in a Fibonacci heap.
- ▶  $s_k$  monotonically increases with  $k$



## 8.3 Fibonacci Heaps

- ▶ Let  $s_k$  be the minimum possible size of a sub-tree rooted at a node of degree  $k$  that can occur in a Fibonacci heap.
- ▶  $s_k$  monotonically increases with  $k$
- ▶  $s_0 = 1$  and  $s_1 = 2$ .

## 8.3 Fibonacci Heaps

- ▶ Let  $s_k$  be the minimum possible size of a sub-tree rooted at a node of degree  $k$  that can occur in a Fibonacci heap.
- ▶  $s_k$  monotonically increases with  $k$
- ▶  $s_0 = 1$  and  $s_1 = 2$ .

Let  $x$  be a degree  $k$  node of size  $s_k$  and let  $y_1, \dots, y_k$  be its children.

$$s_k = 2 + \sum_{i=2}^k \text{size}(y_i)$$

## 8.3 Fibonacci Heaps

- ▶ Let  $s_k$  be the minimum possible size of a sub-tree rooted at a node of degree  $k$  that can occur in a Fibonacci heap.
- ▶  $s_k$  monotonically increases with  $k$
- ▶  $s_0 = 1$  and  $s_1 = 2$ .

Let  $x$  be a degree  $k$  node of size  $s_k$  and let  $y_1, \dots, y_k$  be its children.

$$\begin{aligned}s_k &= 2 + \sum_{i=2}^k \text{size}(y_i) \\ &\geq 2 + \sum_{i=2}^k s_{i-2}\end{aligned}$$

## 8.3 Fibonacci Heaps

- ▶ Let  $s_k$  be the minimum possible size of a sub-tree rooted at a node of degree  $k$  that can occur in a Fibonacci heap.
- ▶  $s_k$  monotonically increases with  $k$
- ▶  $s_0 = 1$  and  $s_1 = 2$ .

Let  $x$  be a degree  $k$  node of size  $s_k$  and let  $y_1, \dots, y_k$  be its children.

$$\begin{aligned}s_k &= 2 + \sum_{i=2}^k \text{size}(y_i) \\ &\geq 2 + \sum_{i=2}^k s_{i-2} \\ &= 2 + \sum_{i=0}^{k-2} s_i\end{aligned}$$

## 8.3 Fibonacci Heaps

$\phi = \frac{1}{2}(1 + \sqrt{5})$  denotes the *golden ratio*.  
Note that  $\phi^2 = 1 + \phi$ .

### Definition 2

Consider the following non-standard Fibonacci type sequence:

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

### Facts:

1.  $F_k \geq \phi^k$ .
2. For  $k \geq 2$ :  $F_k = 2 + \sum_{i=0}^{k-2} F_i$ .

The above facts can be easily proved by induction. From this it follows that  $s_k \geq F_k \geq \phi^k$ , which gives that the maximum degree in a Fibonacci heap is logarithmic.

$$k=0: \quad 1 = F_0 \geq \Phi^0 = 1$$

$$k=1: \quad 2 = F_1 \geq \Phi^1 \approx 1.61$$

$$k-2, k-1 \rightarrow k: \quad F_k = F_{k-1} + F_{k-2} \geq \Phi^{k-1} + \Phi^{k-2} = \Phi^{k-2} \underbrace{(\Phi + 1)}_{\Phi^2} = \Phi^k$$

$$k=2: \quad 3 = F_2 = 2 + 1 = 2 + F_0$$

$$k-1 \rightarrow k: \quad F_k = F_{k-1} + F_{k-2} = 2 + \sum_{i=0}^{k-3} F_i + F_{k-2} = 2 + \sum_{i=0}^{k-2} F_i$$