Fibonacci Heaps

- Collection of ordered trees:

  the key of each node is less than or equal to the keys of all its children.

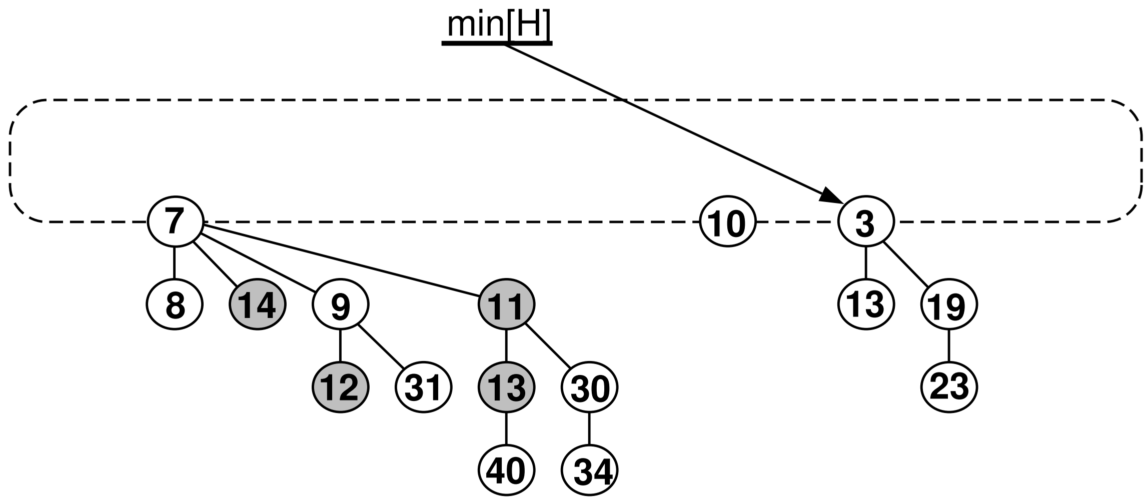
- The roots of these trees are contained in a (doubly linked) cyclic list (*root list*).

- At any time the pointer $\min[H]$ points to the node with the smallest key in the root list (and thus to the node with the smallest key in the entire heap).

- $n[H]$ denotes the number of nodes currently contained in the heap.

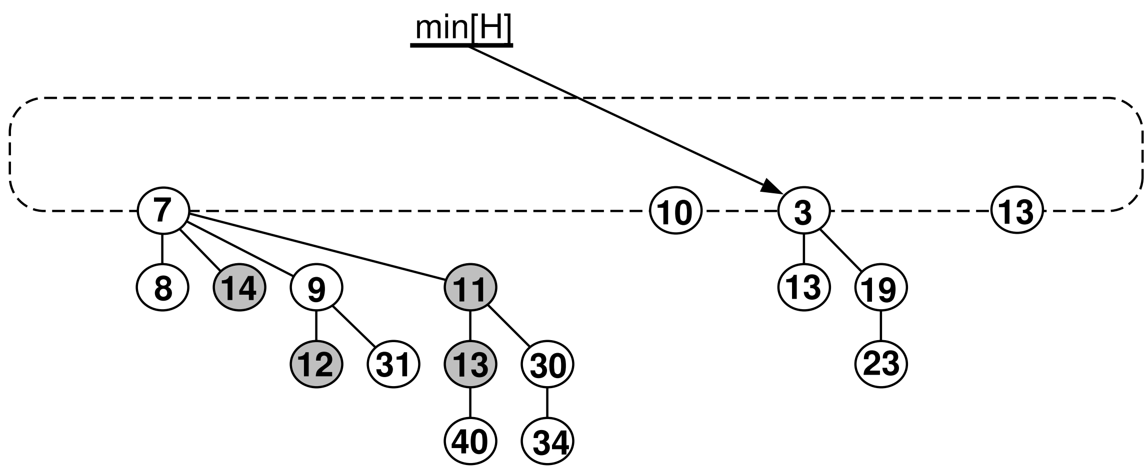
- To each node x the following fields and pointers are attached:

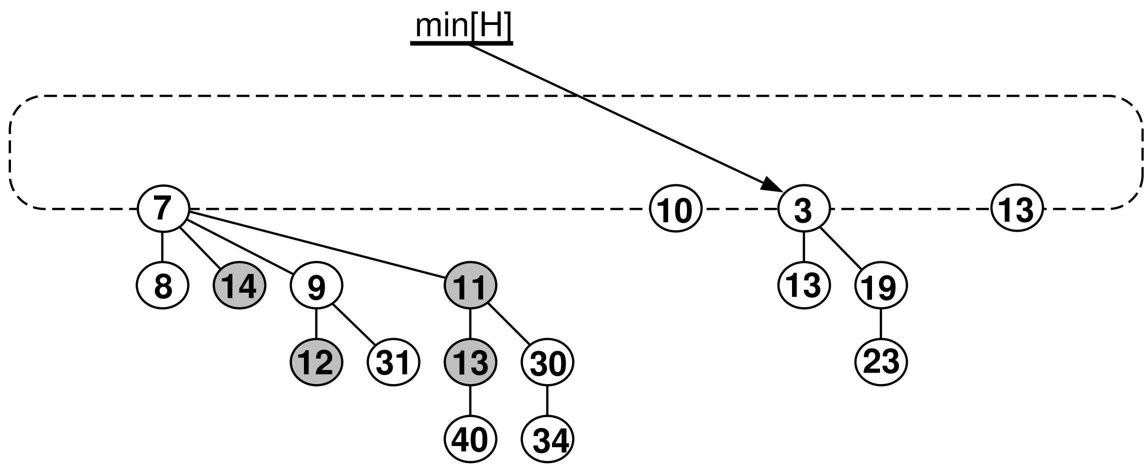
key$[x]$:	key of x,
parent$[x]$:	pointer to the parent of x,
child$[x]$:	pointer to an (arbitrary) child of x,
left$[x]$:	pointer to the left sibling of x,
right$[x]$:	pointer to the right sibling of x,
rank$[x]$:	number of children of x,
marked$[x]$:	boolean field indicating whether x has lost a child already.



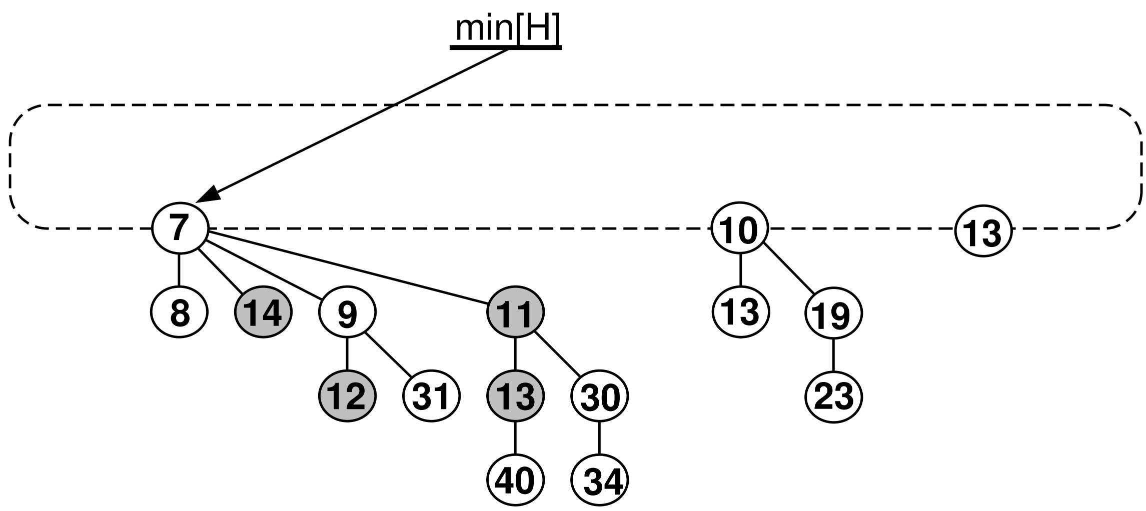
[Die Knoten x mit $\texttt{marked}[x]$ = $\textsc{True}$ sind grau unterlegt.]

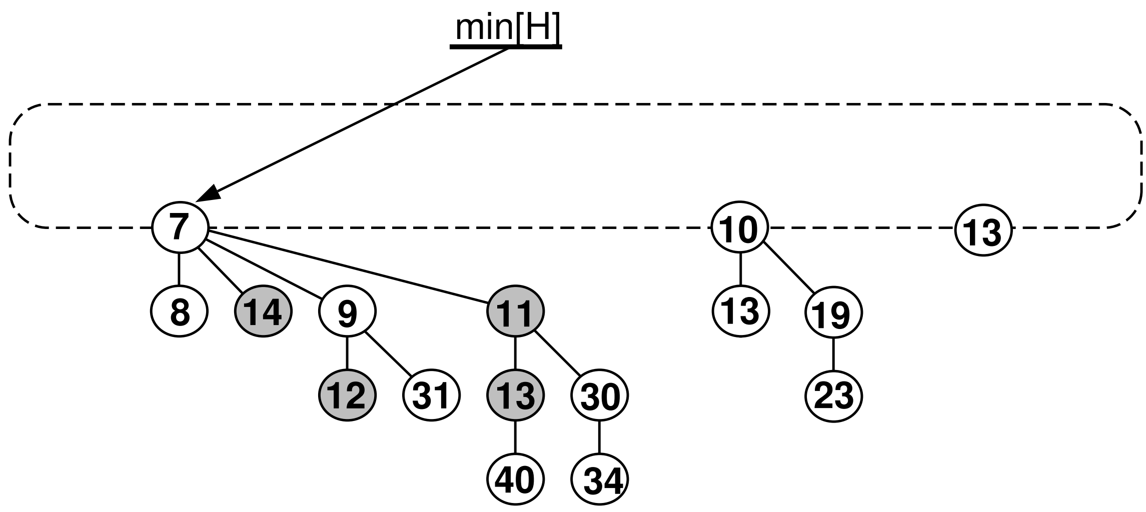
<u>$\textsc{Insert}(H, 13)$:</u>



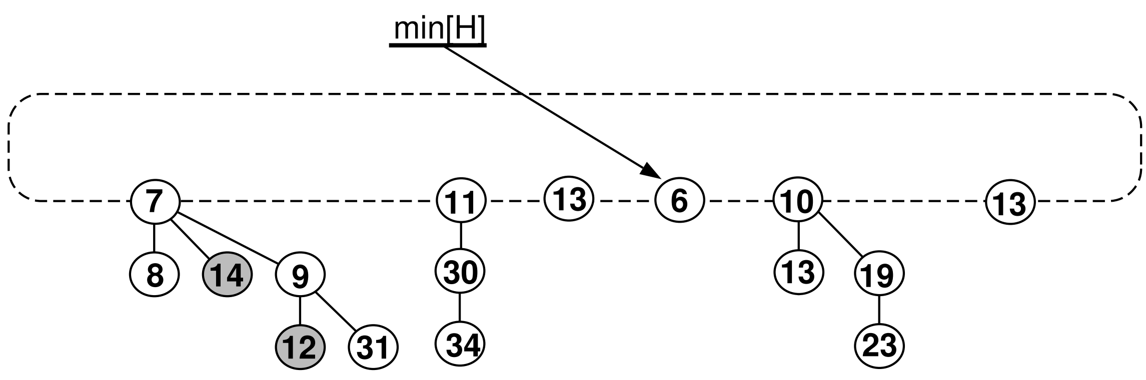


$\textsc{ExtractMin}(H)$:





$\textsc{DecreaseKey}(H,$ <Pointer auf Knoten mit Schlüssel 40>$, 6)$:



```
Extract-Min(H)
z := min[H];
if z ≠ nil then
    for each child x of z do
        parent[x] := nil; marked[x] := FALSE;
        add x to the root list of H;
    remove z from the root list of H;
n[H] := n[H] − 1;
    if root list of H empty
        then min[H] := nil
        else Consolidate(H);
return z;
```

```
Consolidate(H)
for i := 1 to ⌊log_{3/2} n[H]⌋ do A[i] := nil;
repeat
    remove an arbitrary node x from the root list of H;
    while A[rank[x]] ≠ nil do
        y := A[rank[x]]; A[rank[x]] := nil;
        if key[x] > key[y] then exchange x ↔ y;
        make y a child of x, incrementing rank[x];
        marked[x] := FALSE;
    enddo ;
    A[rank[x]] := x;
until root list of H is empty;
min[H] := nil;
for all A[i] ≠ nil do
        add A[i] to the root list of H, updating min[H] if necessary;
```

`Decrease-Key`(H, x, k)

if $k >$ key$[x]$ **then error** (“new key is greater than current key”);
key$[x] := k$;
if parent$[x] \neq \mathbf{nil}$ and key$[x] <$ key[parent$[x]$] **then** `Cut`(H, x);

`Cut`(H, x)

$y :=$ parent$[x]$;
parent$[x] := \mathbf{nil}$; marked$[x] := \textsc{False}$;
remove x from the child list of y, decrementing rank$[y]$;
add x to the root list of H, updating $\min[H]$ if necessary;
if parent$[y] \neq \mathbf{nil}$ **then**
    if marked$[y] = \textsc{True}$ **then**
        `Cut`(H, y)
    else
        marked$[y] := \textsc{True}$;