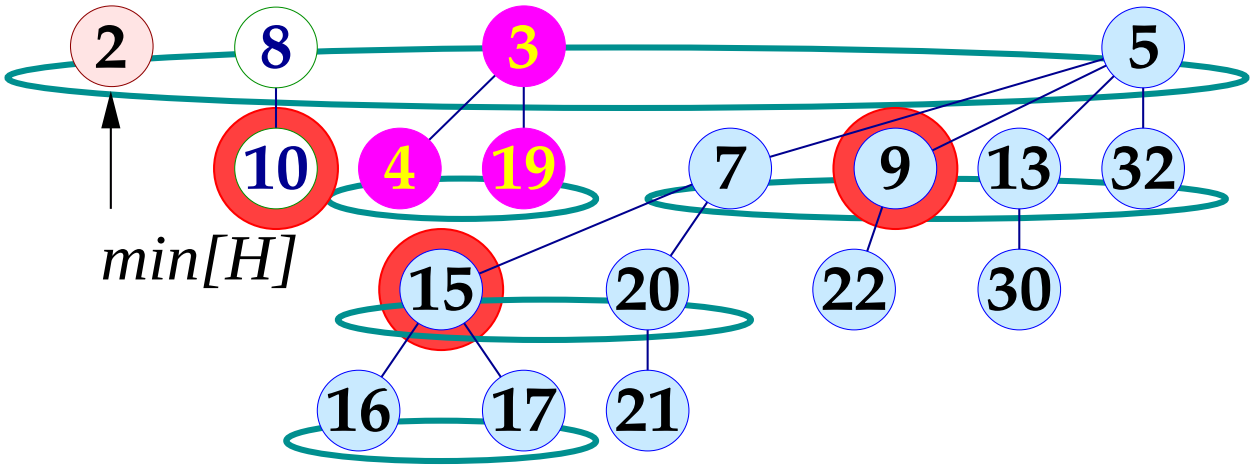
Chapter 20: Fibonacci Heaps

Fibonacci heaps are linked lists of heap-ordered trees (children's keys are at least that of their parent) with the following characteristics:

1. The trees are **not necessarily binomial**
2. Siblings are **bidirectionally linked**
3. There is a pointer $min[H]$ to the root with **the minimum key**
4. The root degrees are not unique.
5. A special attribute $n[H]$ maintains **the total number of nodes**
6. Each node has an additional boolean label $mark$, indicating whether **has lost a child since the last time it was made a child of another node**


2
8
3
5
10
4
19
7
9
13
32
min[H]
15
20
22
30
16
17
21

We assume that roots are always unmarked.

Then what is the running time
for finding the minimum key
in an n-node Fibonacci heap?

Amortized Cost Analysis a lá the Potential Method

Let $t(H)$ be the number of trees and $m(H)$ the number of marked nodes in H. We assume that the computation begins with H being empty.

Define the potential $\Phi(H)$ to be

$$\beta(t(H) + 2m(H))$$

for some fixed positive constant β. This β is the scale-up factor. The term $t(H)$ is for achieving an amortized cost of $O(\lg n)$ for the minimum extraction operation and the term $2m(H)$ is for achieving an amortized cost of $O(1)$ for the key decreasing operation.

Then the potential is always nonnegative. We stipulate that an X number of operations are executed on an initially empty Fibonacci heap. Then the potential at the beginning is 0.

How large can the potential be at the end?

How large can the potential be at the end?

At most $3\beta X$ because there can be at most X marked nodes and X trees at the end.

This means that the contribution of the potential to the amortized cost is $O(1)$.

Insertion of a Key

Suppose we wish to insert a key k into a Fibonacci heap H.

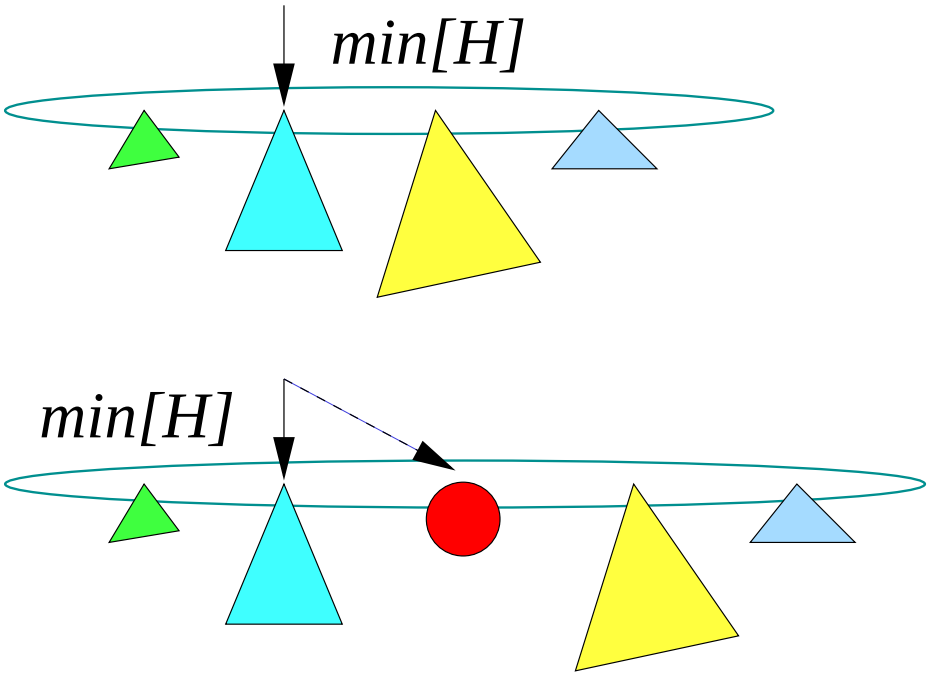
To do this we create a single-node tree T, whose unique node is unmarked and has k as the key. Then we insert T immediately to the right of $min[H]$. If k happens to be smaller than the current minimum key, set $min[H]$ to T. We also increment $n[H]$ by one.

What is the actual cost of this operation?

How much is the potential increased?

The actual cost is $O(1)$ and
the potential is increased by β.

So the amortized cost is $O(1)$.


min[H]
min[H]

Uniting Two Heaps

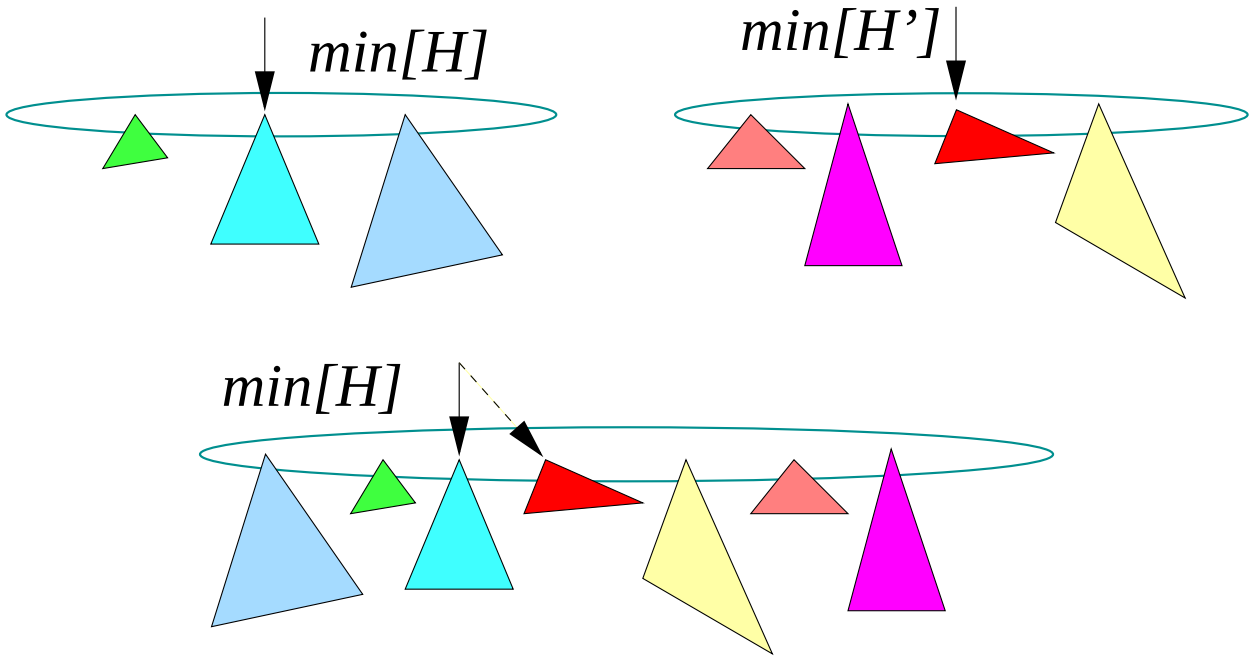
Suppose that we have two heaps, H and H', and want to unite them.

To do this we insert the bidirectionally linked root list of H' after $min[H]$. If $min[H']$ has a smaller key than $min[H]$, then set $min[H]$ to $min[H']$. We also increase $n[H]$ by $n[H']$.

What is the actual cost of this operation?

How much is the potential increased?

The actual cost is $O(1)$ because the root lists are bidirectional. The potential will be the sum of the potentials of the two heaps.


min[H]
min[H']
min[H]

Deleting $min[H]$

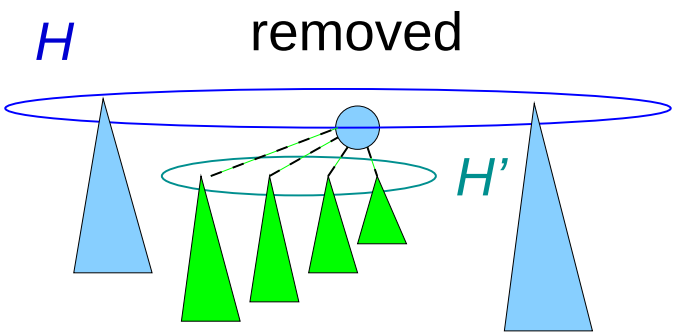
Let $x = min[H]$. Then remove x from the link of the roots and insert the child list of x to it. This requires $O(1)$ steps.

What more do we need to do?

The min is changed, so we need to recalculate this.

To do that we must examine all the keys at the root level.

We will take this opportunity to **"clean up."** *This is we will combine some trees together to make the bottom-level list shorter.*

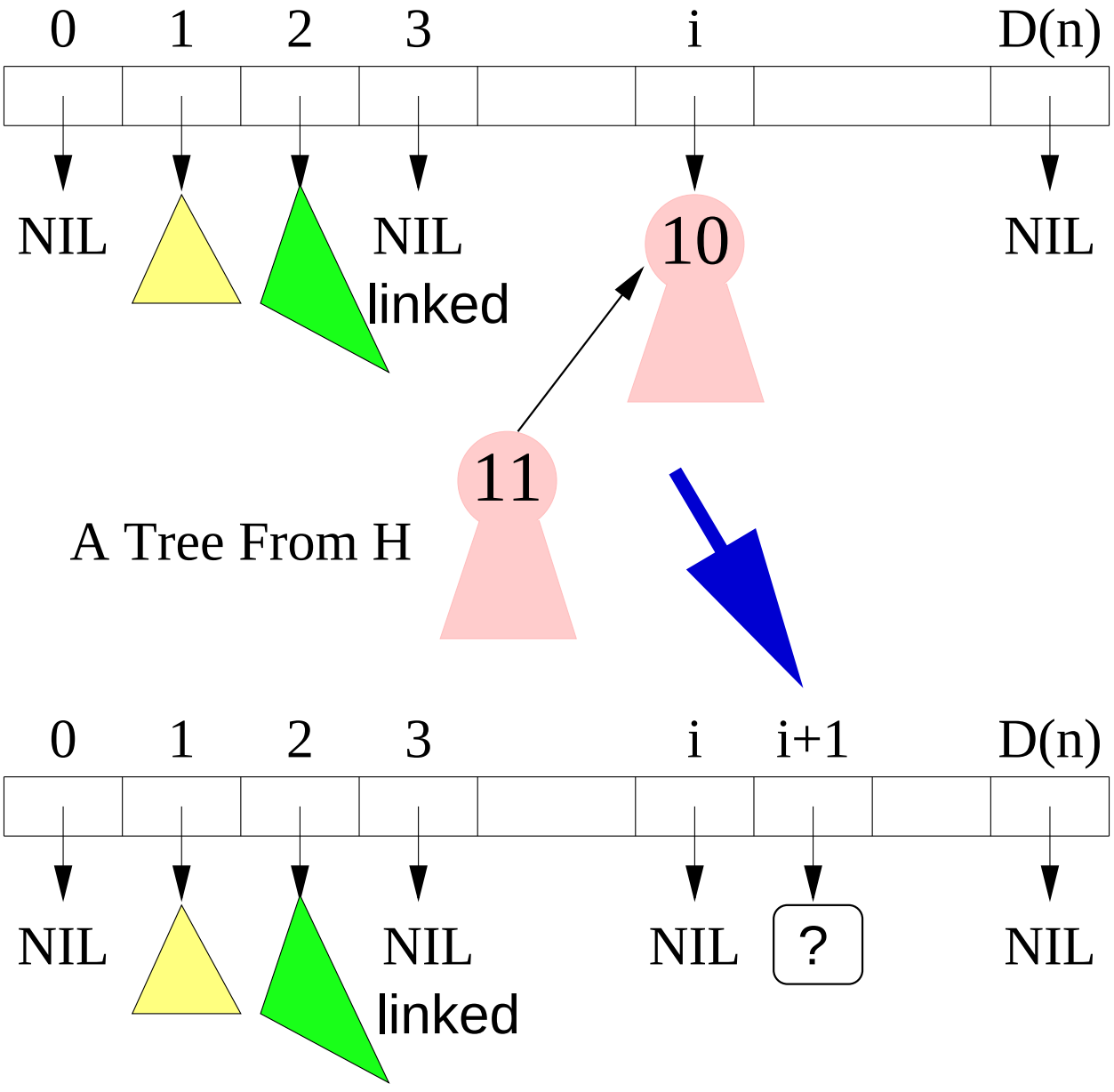

H
removed
H'

Clean up

We will unite trees so that in the remaining trees no two trees have the same root degree.

We think of a tree having root degree k as the integer 2^{k-1} and think of clean-up as binary addition by **linking two trees with the same root degree**, where a marked root that is made a child of another node is unmarked.

Use a binary counter $A[0..D]$, where for each i, $0 \leq i \leq D$, $A[i]$ is the pointer to a tree having root degree i. Starting with the empty counter (i.e. all $\mathbf{nil}$), we add the trees one by one.


0
1
2
3
i
D(n)
NIL
NIL
linked
10
NIL
11
A Tree From H
0
1
2
3
i
i+1
D(n)
NIL
NIL
linked
NIL
?
NIL

Bounding the Size of the Counter

Lemma A There is a constant α such that the maximum root degree in an n-node Fibonacci heap is at most $\alpha \lg n$.

So we let $D = \lfloor \alpha \lg n \rfloor$.

What is the actual cost of cleaning up?

We view "$A[i] \neq \mathbf{nil}$" as the bit being 1 "$A[i] = \mathbf{nil}$" as the bit being 0.

Incorporation of one tree **"sets" a bit at one position** and resets some bits. So, the total number of bits that are set is $t(H)$.

Note that **a bit cannot be reset unless it is already set**. Since the counter is initially zero, the total number of bits that are reset is $\leq t(H)$. Thus, the actual cost is $\leq ct(H)$ for some constant c.

The Amortized Cost

Let H' be the heap after "clean up." The amortized cost is
$ct(H) + \beta(t(H') + 2m(H') - t(H) - 2m(H))$.
We have $t(H') \leq D$ and $m(H') \leq m(H)$. So, the amortized cost of "clean-up" is at most

$$ct(H) + \beta(t(H') - t(H)) = (c - \beta)t(H) + \beta t(H').$$

Choose β so that $\beta \geq c$. Then the amortized cost is $O(D) = O(\lg n)$.

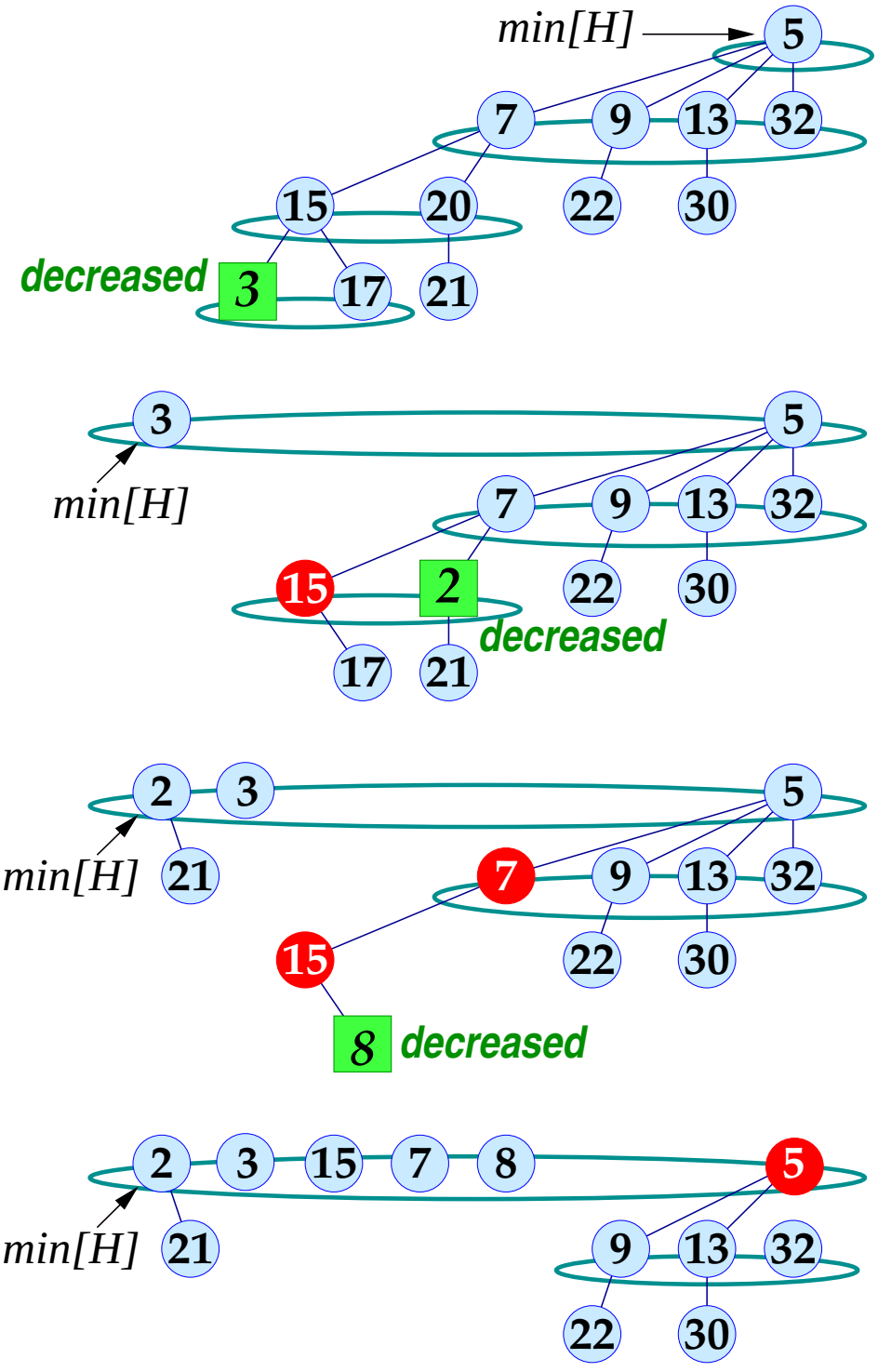
Decreasing a Key

Suppose that a key at node x is decreased. We resolve the heap-property violation by repeatedly exchanging the keys as we did for binary heaps. Then the cost can be as large as the depth of the tree. Unfortunately, this operation does not change the potential and we don't have a good bound on the depth of the tree.

An Alternative for Resolution

Let y be the parent of x. After decreasing $key[x]$, if $key[x] < key[y]$, we mark x and then repeat the following until x is unmarked:

- Insert x to the root list.
- Unmark x if x is marked.
- Adjust $min[H]$ if $key[min[H]] > key[x]$.
- Eliminate x from the list of children.
- Decrease $degree[y]$ by 1.
- If y is marked, then set x to y, set y to $p[x]$. Otherwise, if y is not a root, then mark y.


min[H]
5
7
9
13
32
15
20
22
30
decreased
3
17
21
3
min[H]
5
7
9
13
32
15
2
22
30
decreased
17
21
2
3
5
min[H]
21
7
9
13
32
15
22
30
8
decreased
2
3
15
7
8
5
min[H]
21
9
13
32
22
30

Amortized Cost Analysis

Suppose that s nodes are cut in a sequence. Then this operation

- requires γs steps for some constant γ,
- increases $t(H)$ by s, and
- decreases $m(H)$ by at least $s - 1$.

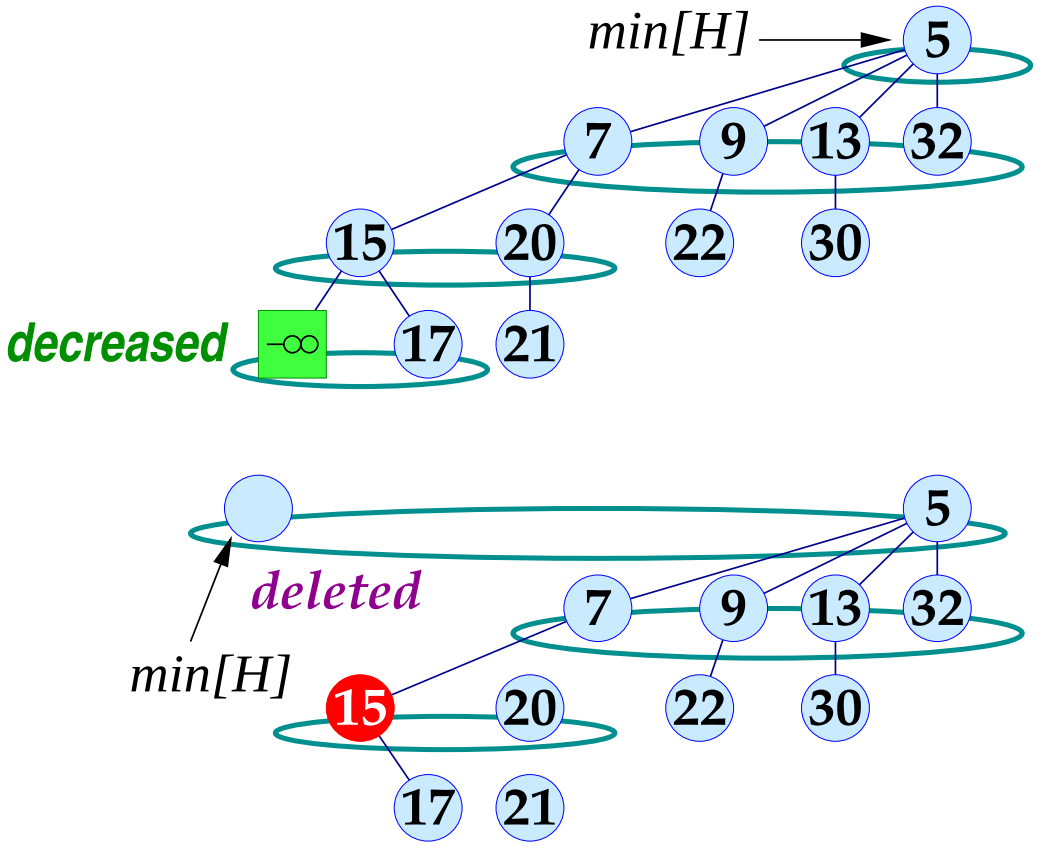
The amortized cost is at most

$$\beta(s - 2(s - 1)) + \gamma s = (\gamma - \beta)s + 2\beta.$$

Choose β so that $\beta \geq \gamma$. Then the cost becomes $O(1)$.

Deleting a Node

Decrease the key to $-\infty$ and then remove the minimum key. The amortized cost is $O(1)$.


min[H]
5
7
9
13
32
15
20
22
30
decreased
−∞
17
21
5
deleted
7
9
13
32
min[H]
15
20
22
30
17
21

Bounding the Maximum Degree

For a node x, let $k = \text{degree}[x]$. Let $y_1, y_2 \ldots y_k$ be the children of x. Then $\text{degree}[y_1] \geq 0$ and $\text{degree}[y_i] \geq i - 2$ for $i = 2 \ldots k$.

Proof: Node y_i had same degree as x when linked to x, and has lost at most one child.

Fibonacci numbers: $F_k = F_{k-1} + F_{k-2}$

$$F_{k+2} = 1 + \sum_{i=0}^{k} F_i$$

$$F_{k+2} \geq \phi^k, \phi = (1 + \sqrt{5})/2$$

Bounding the Maximum Degree

For any node x with $k = \text{degree}[x]$:

$$\text{size}(x) \geq F_{k+2} \geq \phi^k$$

Proof: let s_k be minimum size for degree k.

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

By induction over k, $s_k \geq F_{k+2}$.