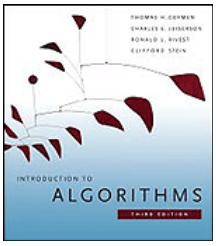

Chapter 19 - Fibonacci Heaps

Introduction to Algorithms, Third Edition

by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein

Previous Next

19.3 Decreasing a Key and Deleting a Node

In this section, we show how to decrease the key of a node in a Fibonacci heap in $O(1)$ amortized time and how to delete any node from an n-node Fibonacci heap in $O(D(n))$ amortized time. In Section 19.4, we will show that the maximum degree $D(n)$ is $O(\lg n)$, which will imply that FIB-HEAP-EXTRACT-MIN and FIB-HEAP-DELETE run in $O(\lg n)$ amortized time.

Decreasing a Key

In the following pseudocode for the operation FIB-HEAP-DECREASE-KEY, we assume as before that removing a node from a linked list does not change any of the structural attributes in the removed node.

```
FIB-HEAP-DECREASE-KEY(H, x, k)
1 if k > x.key
2     error "new key is greater than current key"
3 x.key = k
4 y = x.p
5 if y ≠ NIL and x.key < y.key
6     CUT(H, x, y)
7     CASCADING-CUT(H, y)
8 if x.key < H.min.key
9     H.min = x
```

```
CUT(H, x, y)
1.remove x from the child list of y, decrementing y.degree
2. add x to the root list of H
3. x.p = NIL
4. x.mark = FALSE
```

```
CASCADING-CUT(H, y)
1. z = y.p
2. if z ≠ NIL
3.     if y.mark== FALSE
4.         y.mark= TRUE
5.     else CUT(H, y, z)
6.         CASCADING-CUT(H, z)
```

The FIB-HEAP-DECREASE-KEY procedure works as follows. Lines 1–3 ensure that the new key is no greater than the current key of x and then assign the new key to x. If x is a root or if $x.key \geq y.key$, where y is x's parent, then no structural changes need occur, since min-heap order has not been violated. Lines 4–5 test for this condition.

If min-heap order has been violated, many changes may occur. We start by ***cutting*** x in line 6. The CUT procedure "cuts" the link between x and its parent y, making x a root.

We use the *mark* attributes to obtain the desired time bounds. They record a little piece of the history of each node. Suppose that the following events have happened to node *x*:

1. at some time, *x* was a root,
2. then *x* was linked to (made the child of) another node,
3. then two children of *x* were removed by cuts.

As soon as the second child has been lost, we cut *x* from its parent, making it a new root. The attribute *x.mark* is TRUE if steps 1 and 2 have occurred and one child of *x* has been cut. The CUT procedure, therefore, clears *x.mark* in line 4, since it performs step 1. (We can now see why line 3 of FIB-HEAP-LINK clears *y.mark*: node *y* is being linked to another node, and so step 2 is being performed. The next time a child of *y* is cut, *y.mark* will be set to TRUE.)

We are not yet done, because *x* might be the second child cut from its parent *y* since the time that *y* was linked to another node. Therefore, line 7 of FIB-HEAP-DECREASE- KEY attempts to perform a ***cascading-cut*** operation on *y*. If *y* is a root, then the test in line 2 of CASCADING-CUT causes the procedure to just return. If *y* is unmarked, the procedure marks it in line 4, since its first child has just been cut, and returns. If *y* is marked, however, it has just lost its second child; *y* is cut in line 5, and CASCADING-CUT calls itself recursively in line 6 on *y*'s parent *z*. The CASCADING-CUT procedure recurses its way up the tree until it finds either a root or an unmarked node.

Once all the cascading cuts have occurred, lines 8–9 of FIB-HEAP-DECREASE-KEY finish up by updating *H.min* if necessary. The only node whose key changed was the node *x* whose key decreased. Thus, the new minimum node is either the original minimum node or node *x*.

Figure 19.5 shows the execution of two calls of FIB-HEAP-DECREASE-KEY, starting with the Fibonacci heap shown in Figure 19.5(a). The first call, shown in Figure 19.5(b), involves no cascading cuts. The second call, shown in Figures 19.5(c)–(e), invokes two cascading cuts.

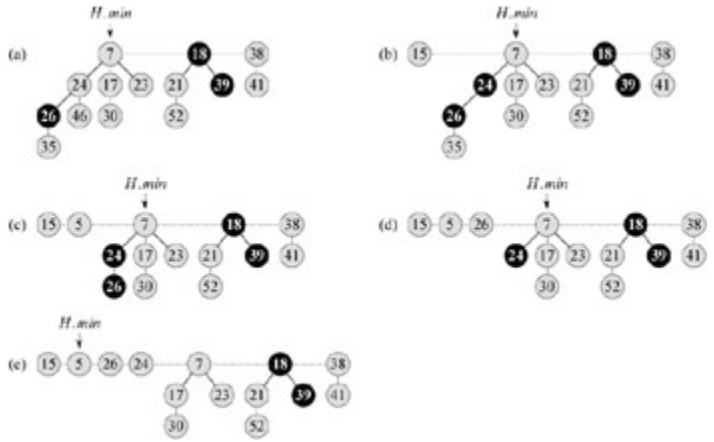

Figure 19.5: Two calls of FIB-HEAP-DECREASE-KEY. **(a)** The initial Fibonacci heap. **(b)** The node with key 46 has its key decreased to 15. The node becomes a root, and its parent (with key 24), which had previously been unmarked, becomes marked. **(c)–(e)** The node with key 35 has its key decreased to 5. In part (c), the node, now with key 5, becomes a root. Its parent, with key 26, is marked, so a cascading cut occurs. The node with key 26 is cut from its parent and made an unmarked root in (d). Another cascading cut occurs, since the node with key 24 is marked as well. This node is cut from its parent and made an unmarked root in part (e). The cascading cuts stop at this point, since the node with key 7 is a root. (Even if this node were not a root, the cascading cuts would stop, since it is unmarked.) Part (e) shows the result of the FIB-HEAP-DECREASE-KEY operation, with *H.min* pointing to the new minimum node.

We shall now show that the amortized cost of FIB-HEAP-DECREASE-KEY is only $O(1)$. We start by determining its actual cost. The FIB-HEAP-DECREASE-KEY procedure takes $O(1)$ time, plus the time to perform the cascading cuts. Suppose that a given invocation of FIB-HEAP-DECREASE-KEY results in c calls of CASCADING-CUT (the call made from line 7 of FIB-HEAP-DECREASE-KEY followed by c - 1 recursive calls of CASCADING-CUT). Each call of CASCADING-CUT takes $O(1)$ time exclusive of recursive calls. Thus, the actual cost of FIB-HEAP- DECREASE-KEY, including all recursive calls, is $O(c)$.

We next compute the change in potential. Let *H* denote the Fibonacci heap just prior to the FIB-HEAP-DECREASE-KEY operation. The call to CUT in line 6 of FIB-HEAP-DECREASE-KEY creates a new tree rooted at node *x* and clears *x*'s mark bit (which may have already been FALSE). Each call of CASCADING-CUT, except for the