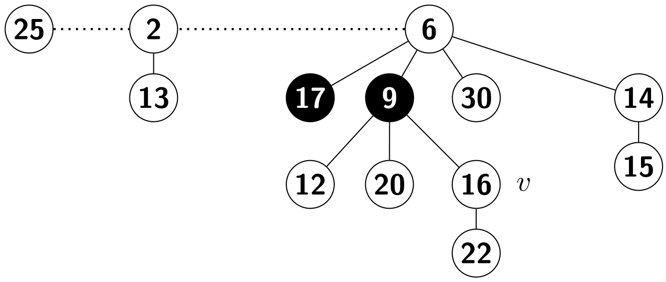
University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
Marc Fuchs

Algorithm Theory
Sample Solution Exercise Sheet 7

Due: Friday, 5th of December, 2025, 10:00 am

Exercise 1: Short questions *(6 Points)*

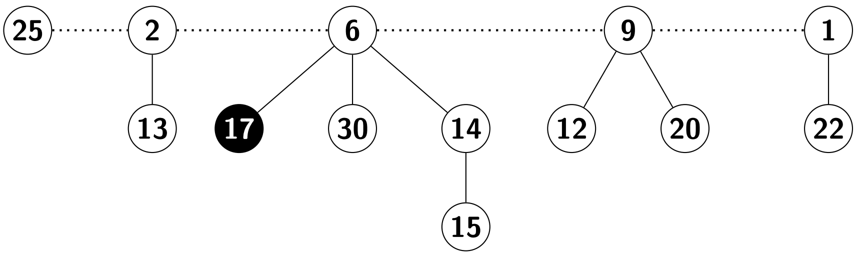
(a) Consider the following Fibonacci heap (black nodes are marked, white nodes are unmarked). How does the given Fibonacci heap look after a `decrease-key(v, 1)` operation and how does it look after a subsequent `delete-min` operation? *(4 Points)*



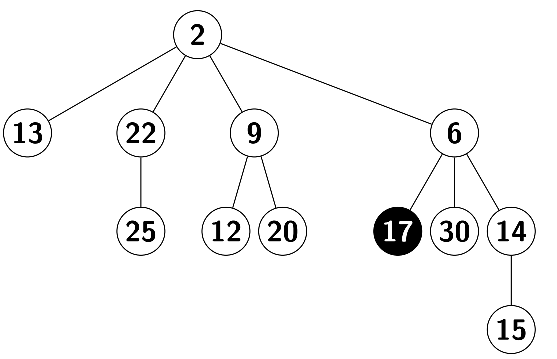
(b) Create a new method on the Fibonacci heap data structure called `Delete-node`(v), which deletes node v from the Fibonacci heap in $O(\log n)$ amortized time. Explain the runtime. *(2 Points)*
Hint: You may want to reuse the methods of Fibonacci heaps you already know.

Sample Solution

(a) After decrease-key:



After delete min:



(b) Let your delete min function be the following.
FIB-HEAP-DELETE(H, x):
1. FIB-HEAP-DECREASE-KEY(H, x, $-\infty$)
2. FIB-HEAP-EXTRACT-MIN(H)
We know that the 1st operation is $O(\log n)$ amortized the 2nd operation is $O(1)$ and thus we get the required runtime.

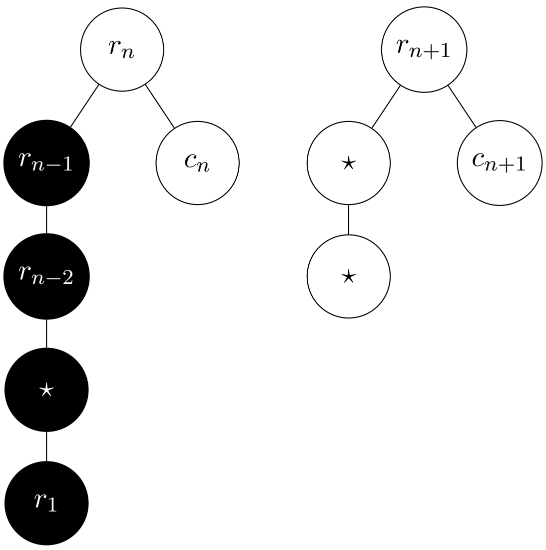
Exercise 2: Worst Case Decrease *(7 Points)*

We've seen in the lecture that Fibonacci heaps are only efficient in an *amortized* sense. However, the time to execute a single, individual operation can be large. Show that in the worst case, the `decrease-key` operation can require time $\Omega(n)$ (for any heap size n).

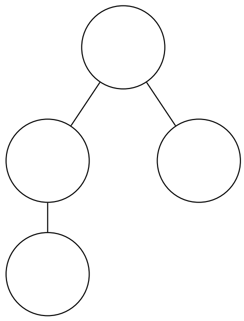
Hint: Describe an execution in which there is a decrease-key operation that requires linear time.

Sample Solution

A costly *decrease-key* operation:
We construct a degenerated tree. Assume we already have a tree T_n in which the root r_n has two children r_{n-1} and c_n, where c_n is unmarked and r_{n-1} is marked and has a single child r_{n-2} that is also marked and has a single child r_{n-3} and so on, until we reach a (marked or unmarked) leaf r_1. In other words, T_n consists of a line of marked nodes, plus the root and one further unmarked child of the root. We give the root r_n some key k_n.
We now add another 5 nodes to the heap and delete the minimum of them, causing a *consolidate*. In more detail let us add a node r_{n+1} with key $k_{n+1} \in (0, k_n)$, one with key 0 and 3 with keys $k' \in (k_{n+1}, k_n)$. When we delete the minimum, first both pairs of singletons are combined to two trees of rank 1, which are combined again to one binomial tree of rank 2, with the node r_{n+1} as the root and we name its childless child c_{n+1} (confer the picture for the current state).



Since also T_n has rank 2 we now combine it with the new tree and r_{n+1} becomes the new root. We now decrease the key of c_n to 0 as well as the keys of the two unnamed nodes and delete the minimum after each such operation, as to cause no further effect from *consolidate*. Decreasing the key of c_n, however, will now mark its parent r_n, as it is not a root anymore. Thus the remaining heap is of exactly the same shape as T_n, except that its depth did increase by one: a T_{n+1}.
Can we create such trees? We sure can by starting with an empty heap, adding 5 nodes, deleting one, resulting in a tree of the following form:



We cut off the lowest leaf and we end up with T_1. The rest follows via induction.
Obviously, a *decrease-key* operation on r_1 will cause a cascade of $\Omega(n)$ cuts if applied to a heap consisting of such a T_n.

Exercise 3: Fibonacci Heap simplification *(7 Points)*

Suppose we "simplify" Fibonacci heaps such that we do *not* mark any nodes that have lost a child and consequentially also do *not* cut marked parents of a node that needs to be cut out due to a `decrease-key`-operation.
Is the *amortized* running time

(a) ... of the `decrease-key`-operation still $\mathcal{O}(1)$? *(2 Points)*

(b) ... of the `delete-min`-operation still $\mathcal{O}(\log n)$? *(5 Points)*

Explain your answers.
Remark: You should NOT re-do the amortized analysis or search for a new potential function. Instead, think of what the implications are, i.e., do the statements from the lecture still work after this change? Especially, can we still bound the size/rank of a tree/node as we did in the lecture?

Sample Solution

When nodes are not marked, they can lose all its children without getting back to the root-list. Note that in the "normal" heaps, a node v can lose only one child, cause then it will get marked. Losing another child would then bring up v to the root list. What is the implication?
The "Rank of Children" lemma from the lecture does not hold, i.e., there could be child u_i of v with rank $< i-1$. Note that also the "Size of Trees" statement from the lecture does thus not hold anymore.

(a) Yes. Not having to cut all your marked ancestor nodes only makes `decrease-key` faster. In fact each individual `decrease-key` operation has now runtime $\mathcal{O}(1)$.

(b) No. As described above, the size of trees of rank k can not be upper bounded by F_{k+2}, and thus we do not have an upper bound on the maximum rank of the data structure that we need to get the $O(\log n)$ desired runtime. In particular, we can actually create a tree of rank $\Theta(n)$ (the construction is simple: if we can construct a tree of k nodes, i.e., a root node with $k-1$ children, we can just add enough dummy nodes to construct the same tree again, merge both, and delete all the not-needed dummy nodes. So we get a new tree of $k+1$ nodes where the root has degree exactly k.). Since `delete-min` has amortized runtime linear in the maximum rank, it can have a higher amortized running time (i.e., $\omega(\log n)$) than in the implementation with marked nodes.