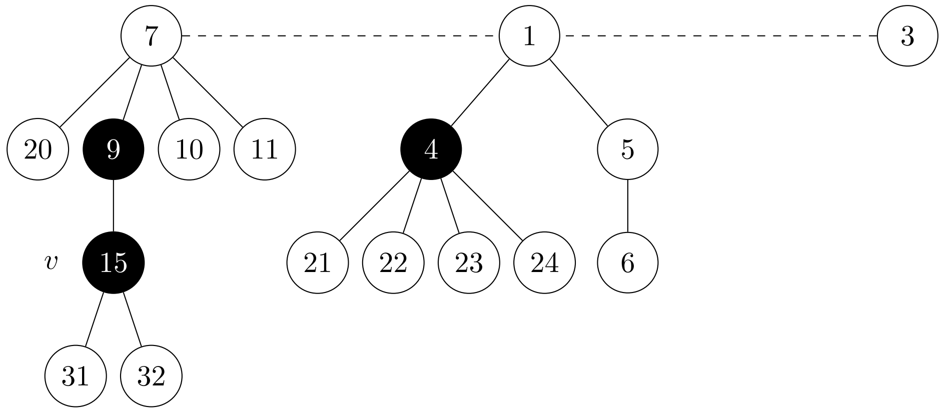
University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
P. Bamberger

Algorithm Theory
Sample Solution Exercise Sheet 4

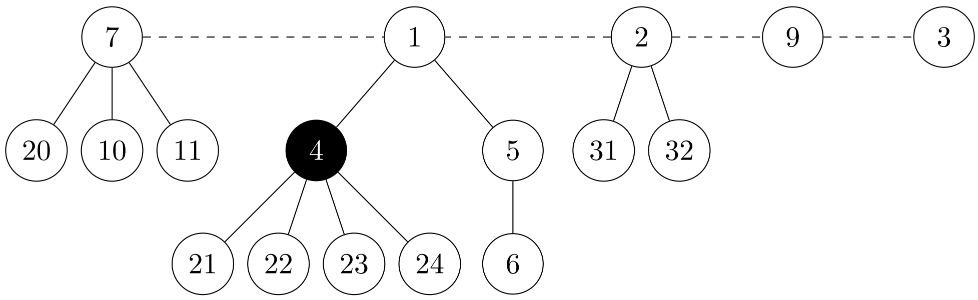
Exercise 1: Fibonacci Heaps I *(8 Points)*

Consider the following Fibonacci heap (black nodes are marked, white nodes are unmarked). How does the given Fibonacci heap look after a `decrease-key`$(v, 2)$ operation and how does it look after a subsequent `delete-min` operation?

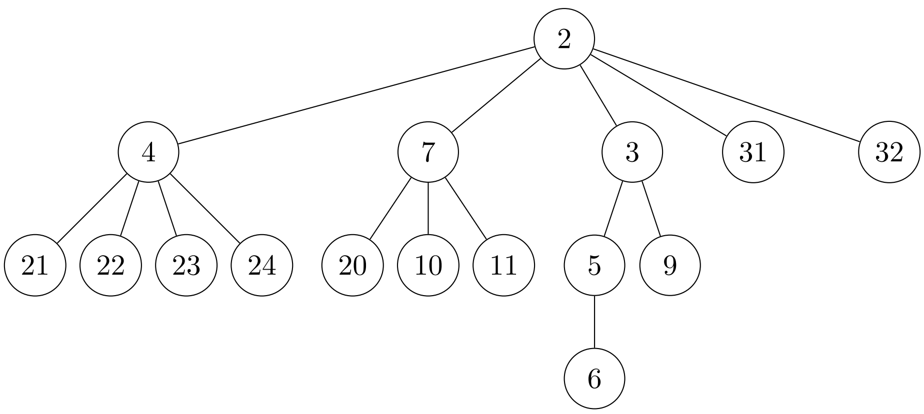


Sample Solution

State after `decrease-key`$(v, 2)$ operation:



State after `delete-min`: *(5 Points)*

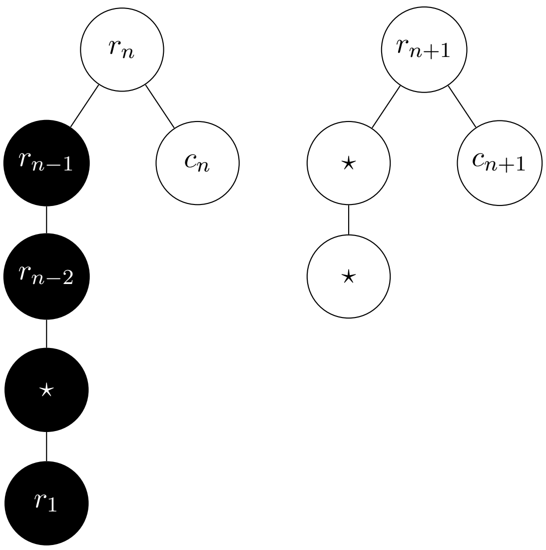


Exercise 2: Fibonacci Heaps II *(10 Points)*

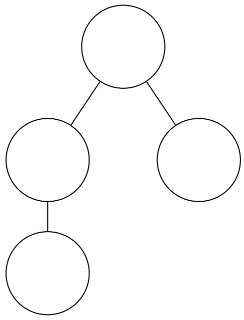
Show that in the worst case, the `delete-min` and the `decrease-key` operation on a Fibonacci heap can require time $\Omega(n)$.

Sample Solution

A costly *delete-min*:
First n elements are added to the heap, which causes them all to be roots in the root list. Deleting the minimum causes a *consolidate* call, which combines the remaining $n-1$ elements, which need at least $n-2$ *merge* operations, i.e., it costs $\Omega(n)$ time.
A costly *decrease-key* operation: (more difficult)
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 now have a 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: Union Find *(12 Points)*

Consider a sequence of operations on a disjoint-set forest using the union-by-size heuristic with path compression. Let f be the number of find-operations and n the number of make_set-operations.

Show that the total costs are $O(f + n \cdot \log n)$.

Sample Solution

As there are n make-set operations, the are at most $n-1$ union-operations. Each make-set or union costs $O(1)$, so the total costs for these operations is $O(n)$. Thus we have to show that the costs of all find-operations is $O(f + n \cdot \log n)$. Let $\alpha_1, \ldots, \alpha_f$ be the sequence of find-operations. Consider a single operation $\alpha_i = \text{find}(x)$. Let p_i be the path from x to the root r of the tree in which x is contained (i.e., $p_i = (x, x.parent, x.parent.parent, \ldots, r)$). The costs of α_i is $O(|p_i|)$ due to path compression. Let $\tilde{p_i}$ be the set of those nodes in p_i which are not the root or the direct child of the root. Then the costs of α_i can be written as $O(1 + |\tilde{p_i}|)$. Therefore, the costs of all find-operations is $O(f + \sum_{i=1}^{f} |\tilde{p_i}|)$. If an element x is contained in $\tilde{p_i}$, then it gets attached to the root after calling α_i. To be contained in some $\tilde{p_j}$ for $j > i$, the tree x is contained in must be attached to a larger tree (union-by-size heuristic). This can happen at most $\log n$ times. Therefore, x is contained in at most $\log n$ sets $\tilde{p_i}$. It follows that $\sum_{i=1}^{f} |\tilde{p_i}| \leq n \log n$.