Algorithm Theory, Tutorial 4

Johannes Kalmbach

University of Freiburg

johannes.kalmbach@gmail.com

December 2019

General Hints

- Contact tutor (`johannes.kalmbach@gmail.com`) for questions concerning corrections etc.
- Contact forum (`daphne.informatik.uni-freiburg.de`) for everything else
- Suggestion: Submit in groups of two (better for understandable algorithms)
- Submit readable solutions (LaTeX as pdf, CLEAN handwriting (+ good scan if necessary))
- Spend enough time on exercise sheets and writeup (you and I have to understand your submission).

Fibonacci Heap

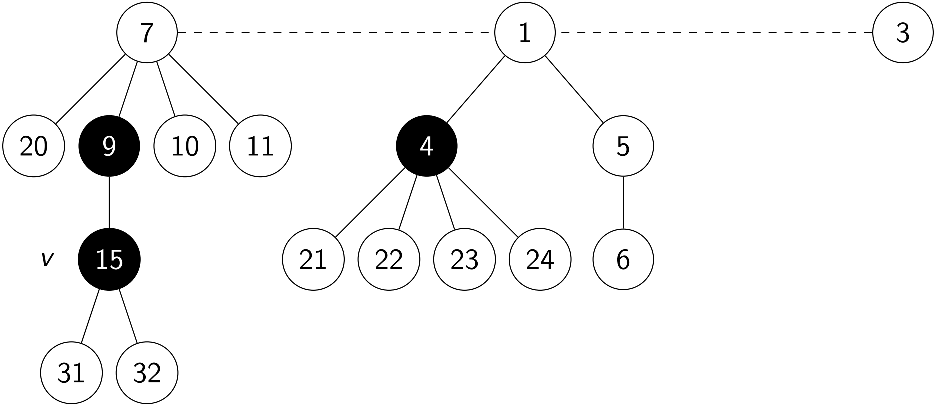
- Set of min-heaps, roots connected by a doubly linked list
- Always keep track of minimum
- Nodes may be marked (if not in rootlist)
- insert: Insert a new node (tree of size 1) into the root list ($O(1)$)
- Merge: Merge rootlists $O(1)$
- getMin: Return the minimum (we keep track of it) ($O(1)$)

Fiboacci Heap - Complex Operations

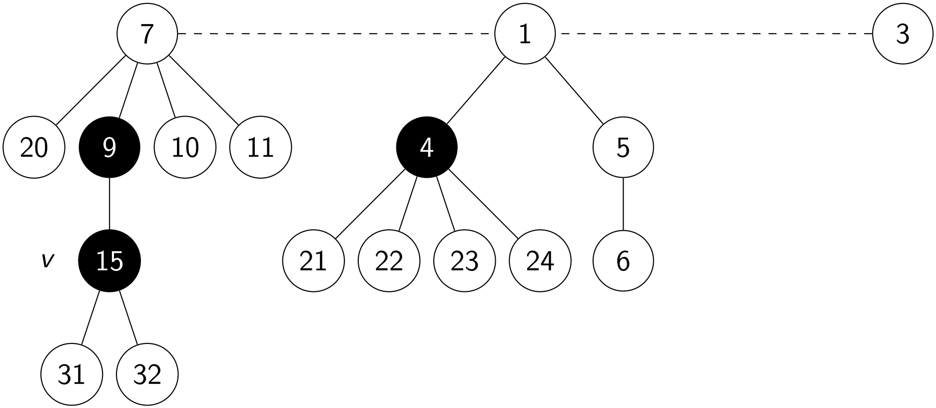
- decreaseKey: If necessary (min heap condition violated), cut node and add to root list (unmarked). Cut Ancestors until a non-marked ancestor is found. Mark that one (unless it is in rootlist). $O(NumMarkedDirectAncestors)$
- deleteMin: The min node is in the rootlist. Delete it and add its subtress to the rootlist.

Exercise 1

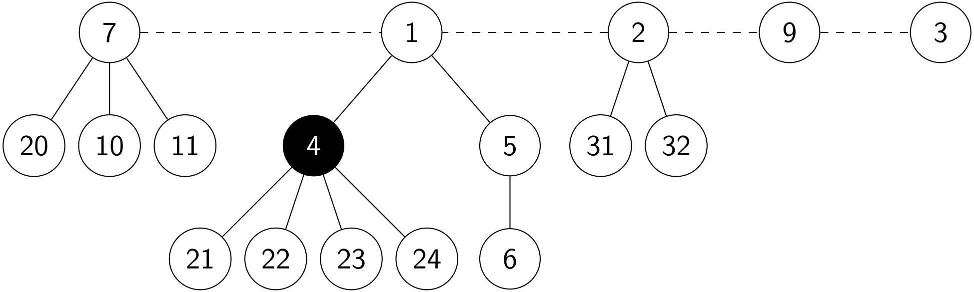
Consider the following Fibonacci heap (black nodes are marked, white nodes are unmarked). How does the given Fibonacci heap look after a `decrease-key`$(v, 2)$ operation and how does it look after a subsequent `delete-min` operation?

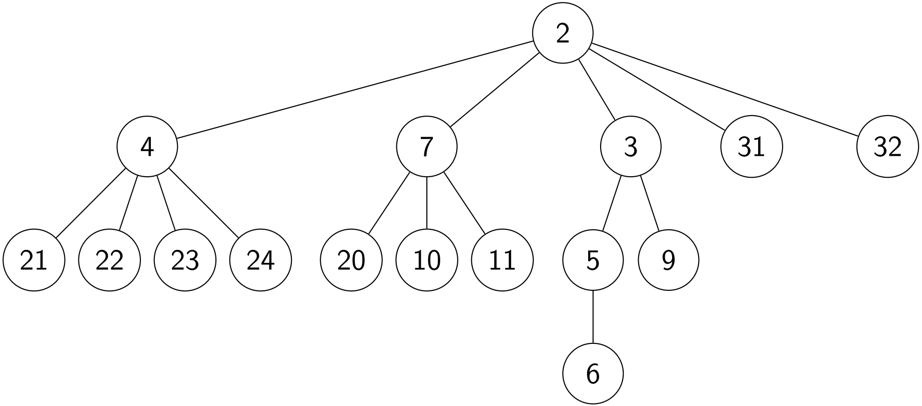


DecreaseKey(v, 2)



After DecreaseKey(v, 2), next: DeleteMin




2
4
7
3
31
32
21
22
23
24
20
10
11
5
9
6

Costly Operations

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

- Strategy: What kind of heap do we need s.t. the operation becomes costly.
- Verify that such a heap can indeed be created by legal operations.
- Second part is important!

DeleteMin (+ Consolidate)

- Consolidate needs to look at all Elements in the rootlist.
- We need $\Omega(n)$ elements in the rootlist, the call deleteMin will be $\Omega(n)$. How do we achieve this?

DecreaseKey in $\Omega(n)$

Expensive if $\Omega(n)$ nodes in a single (ancestry) chain, all of them marked. Then decreaseKey on the lowest node will cause $\Omega(n)$ cuts.

How to obtain Such a tree : Induction

Union Find, Implemented as a set of forests with path compression + union by size

- makeSet :
- find :
- union :

Exercise 3

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 $O(f + n \cdot \log n)$.

- What do we know from the lecture?
-
- What about union operations?

- Cost of n times makeSet:
-
- Cost of $n-1$ unions:

Cost of find

- Remains to show: f find Operations can be run in $O(f + n \log n)$
- NOT safe to assume that we first perform all unions first and then all finds. (This runs in $O(n)$ and we might need intermediate finds for the union operations).
- Idea: look at all the find-Operations for **one single element** element.
- find compresses the complete path to the current root.
- union operations might make the path to the root longer (but compressions remain)
- Example:

- Cost of a single find: $O(1 + \mathit{UncompressedPathToCurrentRoot})$
- Total cost for one element:

$$\sum_{f_i} 1 + \mathit{UncompressedPathToCurrentRoot})$$

- Since we always Compress the path on *find*, and the total path length is always $O(\log n)$ (union by size, from lecture), the *UncompressedPathToCurrentRoot* sum to at most $O(\log n)$ for a single element. Thus for a single element the finds cost

$$O(f_i + \log n)$$

If we sum this over all n elements we get $O(f + n \log n)$ as desired