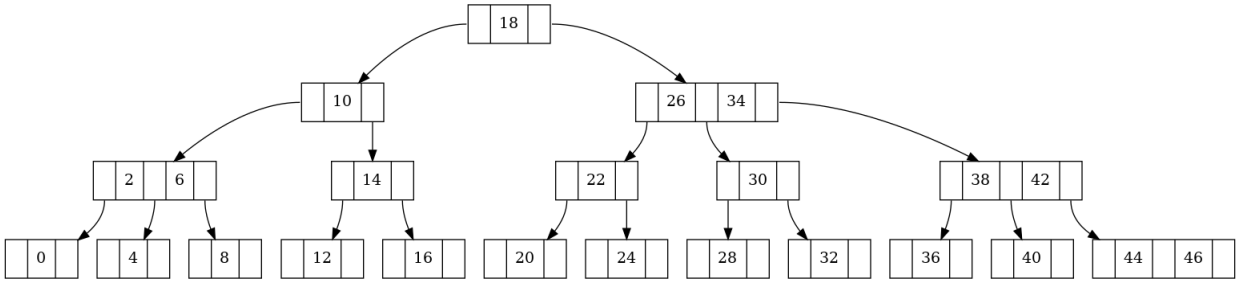
Tutorial 5

Data Structures 1, 21. 3. 2025 https://iuuk.mff.cuni.cz/~chmel/2425/ds1en/

Exercise 1 ((a,b)-trees in your hands)
In the figure, you can see a $(2,3)$-tree. Perform the following operations on this tree (always just one and then start with a new tree): Insert(7), Insert(48), Delete(44), Delete(40), Delete(32), Delete(30), Delete(16).



Solution

Exercise 2 (Correct parameter choice)
From the lecture, we know that an arbitrary sequence of m operations Insert and Delete on an originally empty $(a,2a)$-tree makes only $\mathcal{O}(m)$ node modifications (splits and merges). Show that this is not true for $(a,2a-1)$-trees. That is, for any m,n, design a sequence of m operations on a tree with $\Theta(n)$ nodes that changes $\Omega(m\log n)$ nodes in total.
You can start with $(2,3)$-trees and then generalize for an arbitrary a. You can also start with an arbitrary (valid) n-node tree and then show that you can indeed build the tree from the empty tree.

Solution
In general: a tree that has most nodes almost empty (only a children), and just the right path (spine) has all nodes full ($2a-1$ children). Let us use M to denote 1+ the maximum key in the tree, then the operations Insert(M),Delete(M) must first split all nodes and then, they must merge all of those back together.
After the insert, the lowest node will have $2a$ children, and thus it must split into two nodes with $a$children. But then the problem moves a layer up. When we delete, the node that contained M will have only $a-1$ children, and thus we need to add more children – but its only sibling also has only a children, so we must merge. This also changes the number of children of the node's parent, and the problem bubbles up to the root again.

Exercise 3 (Better fill factor)
Try to tweak the (a,b)-tree and its operations Insert, Delete so that all its nodes can be a little more full. In particular, we want to be able to build a $\left(\frac{2}{3}b,b\right)$-tree (for b a multiple of three).

Solution
The main trick is in suitable merging and splitting of the nodes. In standard (a,b)-trees, we join two nodes into one, and split one node into two. Thus, we want to merge three nodes into two and split two nodes into three.
For Insert: the element is inserted as usual, and if we have and overfull node, we attempt to move one of its elements into its sibling. If the sibling is also overfull, we have two nodes with $2b+1$ children in total, and we can split these into three nodes with $2b/3$ children each.
For Delete: we can also merge three nodes into two. If a node has $2b/3-1$ children, we can first try to steal a child from our sibling. If this is not possible because both of our siblings have too few children, then we have merge the three nodes into two: we have $2b$ children for the two nodes, which is just the right number.
Note that there is a special case for the leftmost and rightmost nodes in each layer – there, we have only one sibling, but we can reduce to the usual case by stealing unconditionally from the only sibling.

Exercise 4 ((a,b)-Join)
Design the Join operation for $(a,2a)$-trees: you have two $(a,2a)$-trees T_1,T_2 such that all keys in T_1 are smaller than all keys in T_2, and your goal is to build a single tree from the two trees. Beware that the trees can have different heights.
(For the analysis of Split, it might be useful to analyse the complexity more precisely than just $\mathcal{O}(\log n)$.)

Solution
We have two trees, one of those might be taller (WLOG the left tree is not higher than the right tree). What we do is, we find the leftmost node on a suitable level of the right tree, and we merge it with the roof of the left tree. This one node might be too large, but we can fix this as if we were inserting.
For the implementation, we actually need to know the depth of the tree, so we maintain a counter for it.
There is also a second problem: as we are working with the version where the values are in all nodes, we must add a value that is in the tree. We do this by maintaining a pointer to the minimum element in each tree and maintaining the invariant that the node containing the minimum key in the tree has at least $a + 1$ children so that we can steal the element without any node reorganizations necessary (here we need to have $(a, 2a)$-trees). Importantly, we do not need to do additional work after the join, and the pointer to the minimum element is still correct (we always use the minimum element of the right tree).
The complexity is then $\mathcal{O}(\log(|T_2|) - \log(|T_1|))$ – we only need to go down to the layer $\log(|T_2|) - \log(|T_1|)$ for the merge, and then we bubble back up again.

Bonus exercises

Exercise 5 ((a, b)-Split)
Design the Split operation: you have an $(a, 2a)$-tree T and a key k, and you want to split T into two trees such that one contains all nodes less than k and the other contains the rest. Moreover, we want the operation to run in time $\mathcal{O}(\log n)$.

Solution
We search for k according to which we are supposed to split (if it is not in the tree, we go into a leaf). Then we take all subtrees to the right of the path from the root to k and we split these from the original tree. Note that the split trees are valid (a, b)-trees, as we potentially broke only their roots.
We then take these cut-off trees and merge those using join as in previous exercise. The cost then telescopes to $\mathcal{O}(\log n)$.
We also need to fix the left tree, which is done similarly to fixing after a delete.