Ballmachine

General Considerations

- Because of the queries of type "insert k balls", you might think that you have to insert multiple balls at once to be quick enough. But as there can never be more than N balls in the machine, and they are only removed one by one, only $O(Q)$ balls are ever added, so it is fine to insert them one by one.
- You can pre-compute for each node the smallest number inside of its subtree. This takes $O(N)$ with one DFS.
- Now you can easily simulate the whole process always taking one step at a time. This approach takes $O(QDR)$ time, where D is the depth of the tree, and R the maximum degree of any node. On the balanced-tree lower limit, it is $D = O(\log(N))$ and $R = 2$, so the total time is $O(N\log(N))$ which is fine. But on a degenerate tree (i.e. a line), it becomes $O(N^2)$ which is too slow.

Fast Insertion

- After sorting the (direct) childs of each node by lowest number in their respective subtree, do another DFS to compute the post-order-index of each node.
- These post-order-indices are the priorities, by which nodes are getting filled by the add-queries. Therefore you can keep all empty nodes in an appropriate datastructure that allowes you to find the node to be filled in $O(\log(N))$ time. STL-set or STL-priority-queue will both work here.

Fast Deletion

- Let $p(x)$ denote the parent of node x. You can precompute $p^{(2^k)}(x)$ for all x and all appropriate k. This will take $O(N\log(N))$ space, and with a little DP $O(N\log(N))$ time.
- Now you can speed up the the roll-down part of a removal query to only take $O(\log(D))$ time.