B-Tree supplementary slides

ECS, University of Southampton

B-Trees

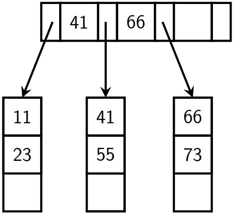
In the lecture on tries and B-Trees, the insertion process for a B-Tree is illustrated on an example where $M = 5$ and $L = 5$.

There are a number of variants of B-trees. The definition on lecture slide 7 requires all data items to be stored in the leaves (so the internal nodes cannot store data items).

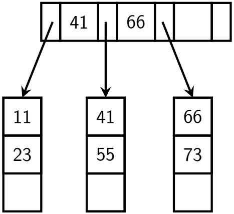
This variation is sometimes known as a B$+$ tree, although (annoyingly) this term can mean a number of things.
In particular, it is commonly understood that a B$+$ tree allows efficient in-order traversal of the data in the leaves by linking the leaves (this is *not* done in the example from the lecture).

It is perhaps easier to follow the logic behind the insertion procedure for this variant of B-tree in a smaller example.

B-Trees



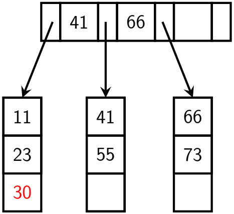
B-Trees



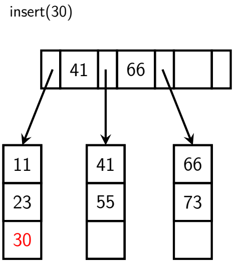
N.B. In this variant of a B-tree, the data items can only be stored in the *leaves* (so in the root node 41 and 66 are not data items, but they are in the leaves).

B-Trees

insert(30)

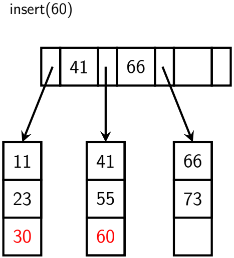


B-Trees

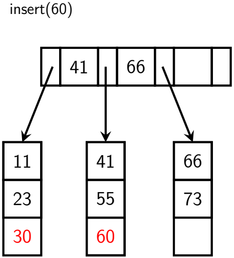


41 and 66 in the root act as separators and keys are inserted according to their membership in intervals defined by these separators. In this case $30 < 41$ and there is space to store the item in the first child (leaf) node.

B-Trees



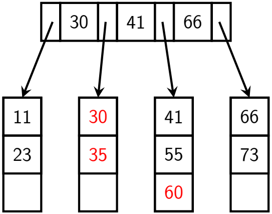
B-Trees



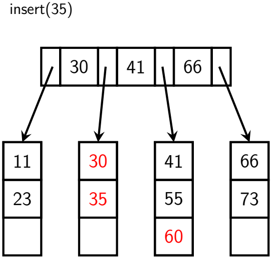
60 lies in the interval $[41, 66)$, and so is stored in the second leaf (again, space permits).

B-Trees

insert(35)



B-Trees



When we insert 35, we run out of space in the first leaf node, which cannot store the data items $11, 23, 30, 35$. We split the leaf into two parts $11, 23$ and $30, 35$; if we are "right-biased", we pick 30 as our median and insert 30 as a separator into the parent node if we can. If not, we follow the same splitting process with the parent node.