Exercise 7.1

maandag 2 oktober 2023 17:15

(i) Without loss of generality, assume the nodes of the tree are numbered with labels from 1 to n. Then, we can observe that the difference in label between a node and its child halves for each level we go down in the tree. As long as the difference in labels remains at least B, the nodes with these labels must be in separate blocks; hence, reading in these nodes and their corresponding blocks will take $\Theta\left(\log_2 \frac{n}{B}\right)$ I/Os. Now, for any x, it holds *(in a complete and balanced binary search tree)* that if the difference between the label of a node v and the label of its child w is x, then the number of nodes in the subtree rooted at w is at most $2x - 1$. Thus, at some point on the path from the root to a leaf *(more precisely, when the difference between a node and its child becomes $\frac{1}{2}B$)*, then we must have that the size of the subtree rooted in that point becomes B, at which point this subtree (which contains the remainder of the path to a leaf) is contained in at most two blocks.
If the memory can only contain a single block, then it holds that, for each remaining level of the tree, it might (in the worst case) be the case that continuously swapping which of these two blocks in memory is necessary. In that case, the number of I/Os for this part of the tree required would grow as $O(\log_2 B)$. In the best case, however, or when the memory can contain at least two blocks, then only a constant number of I/Os is required to load these blocks. Hence, the number of this I/Os for this part of the tree is at least $\Omega(1)$. In total, this gives at least $\Omega\left(\log_2 \frac{n}{B} + 1\right) = \Omega\left(\log_2 \frac{n}{B}\right)$ I/Os and at most $O\left(\log_2 \frac{n}{B} + \log_2 B\right) = O\left(\log_2\left(\frac{n}{B} \cdot B\right)\right) = O(\log_2 n)$ I/Os.

(ii) A more efficient way of forming blocks stores every subtree of height $\lfloor \log_2 B \rfloor$ in a single block. In this strategy, it is necessary to only load a single block for every B levels of the tree *(ignoring any rounding)*, which takes at most $O\left(\frac{\log_2 n}{\log_2 B}\right) = O(\log_{\mathrm{B}} n)$ I/Os.
starting from root, take children, children, until block full
This is somewhat similar to the concept of B-trees, in the sense that B elements of the tree are combined to form a unit of the tree, through which the algorithms go once. The main difference is that in a B-tree, the B elements are combined into a single node, whereas this solution combines them into a subtree. From the I/O perspective, however, this behaves very similarly.