On Left-balancing Binary Trees

J. Andreas Bærentzen (jab@imm.dtu.dk)

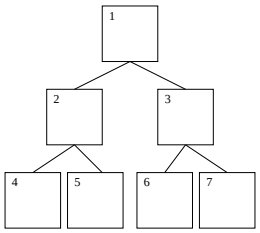
August 25, 2003

Keywords: kD tree, binary tree, balancing.

[*Note that this document is not meant to be read by itself. It is important to have some idea about binary trees. Probably there is nothing new in this document - see e.g. Sedgewick, Algorithms in C++. However, I found no document on the net discussing this issue.*]

There is a simple idea which makes it possible to store a binary tree structure in an array. The idea is that instead of explicitly storing pointers which point from a given node to its child nodes, we use arithmetic to compute the index of child nodes. The scheme works as follows: The root node is stored in position 1 in the array, and when a node is stored in position n in the array, the child nodes are stored in $2n$ and $2n+1$.

The scheme is illustrated below

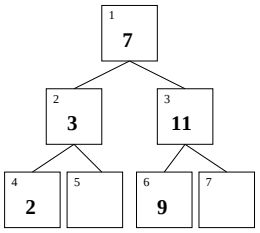


It should be mentioned that there are other ways of storing binary trees in arrays, but this method has the advantage that all nodes on a given level lie in a consecutive block. In addition, left and right children are adjacent which should make the scheme more cache friendly.

The scheme is useful not only for binary trees but also for so called k-D trees which are really binary trees where the key used differs between levels. However, for the sake of simplicity, I'll just discuss this scheme for binary trees. The technique is precisely the same for k-D trees.

The normal way of constructing a binary tree is to sort the elements and recursively split them into left and right parts with respect to the median element.

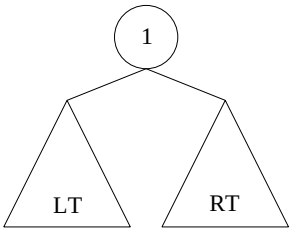
For instance, for the sequence 2,3,7,9,11 the median element is 7 which becomes the root node, and then the sequence recursively split. The left sequence is 2,3 and the right sequence is 9,11. When there are only two elements, the second element becomes the median, and the first element is the left (and only) child. The resulting tree is shown below



There are problems, though. The array element number 5 is unused. This is bad because it means that we have to use a larger array than what is needed to store all nodes. Moreover, it means that we have to store somewhere the information that the node in position 2 does not have a right child.

To get around this problem, we have to ensure that the tree becomes left balanced. Left balanced simply means that all levels except the bottommost are filled, and in the bottommost level, positions are filled from the left. In practice we compute the median in a special way (not just dividing by two). The median has to be computed in such a way as to ensure that we fill up the entire bottommost level of the left subtree before filling the bottommost level of the right subtree.

To do so, we first have to find out how many elements it takes to completely fill the tree on all levels except the bottommost (which is perhaps not possible to completely fill). If the number of levels excluding the bottommost is n this part of the tree will hold $M-1$ elements where $M = 2^n$. We divide these $M-1$ elements evenly between 1 element for the root and split the remaining in two even piles for the left tree ($LT = (M-2)/2$) and the right tree ($RT = (M-2)/2$). See below



The last step is to divide the remaining elements between the left part of the tree and the right part of the tree. To make the tree left balanced we simply fill up the lowest level of the left tree and put the remainder in the right tree.

In practice, we first find the greatest power of two $M = 2^n$ so that $M \leq N$ where N is the number of elements we wish to insert. The tree will hold $M-1$ elements on all levels excluding the bottommost. The bottommost level itself will hold M elements divided between $M/2$ in the left subtree and $M/2$ in the right subtree.

We compute the remainder $R = N - (M-1)$ and then if $R \leq M/2$

$$LT = (M-2)/2 + R \quad (1)$$

$$RT = (M-2)/2 \quad (2)$$

Otherwise, if $R > M/2$

$$LT = (M-2)/2 + M/2 \quad (3)$$

$$RT = (M-2)/2 + R - M/2 \quad (4)$$

In our example, (2,3,7,9,11) we have N=5 elements and M=4, so R=5-(4-1)=2. Hence LT is 3 and RT is 1. Therefore, 9 becomes the median element used as root node, and 2,3,7 is put into the left subtree whereas 11 becomes the right tree. We recurse to compute the entire tree and finally obtain the tree seen below