Optimal BST vs Huffman

The Optimal BST (Binary Search Tree) and Huffman Codes have many similarities which makes it all the more important to understand the vital differences. Note that the Huffman Code is an Optimal Prefix Code, so the remarks for Huffman Code apply to Prefix Code.

Similarities

1. Both have an alphabet $\Omega$ of symbols and the English alphabet $\Omega = \{a,\ldots,z\}$ provides an easy example.

2. For both each letter $\alpha$ is given a frequency $p(\alpha)$. $(p, \text{probability}, \text{and } f, \text{frequency are used interchangeably.})$ We generally assume $\sum_{\alpha} p(\alpha) = 1$, though this is only a technical convenience.

3. For both a binary tree $T$ is created.

4. For both one wants that $T$ which minimizes

\[ \sum_{\alpha \in \Omega} p(\alpha)d(\alpha) \quad (1) \]

where $d(\alpha)$ is the depth of $\alpha$ in the tree – meaning the distance from the root.

Differences

1. For the BST the letters of $\Omega$ are \textit{ordered} and the labelling must have the property that all nodes in the left subtree (right subtree) of a node must occur before (after) that node in the ordering. For the Huffman code $\Omega$ is not ordered.

2. For the BST all nodes of $T$ (both interior and leaf, including the root) receive a label $\alpha \in \Omega$. In the Huffman code the \textit{leaves} of $T$ receive labels $\alpha \in \Omega$.

Implementation Both BST and Huffman codes have ingenious implementation that uses many of our techniques, but they are very different implementations.

Application

1. The Optimal BST allows for quick lookup of the average letter, where “average” is in terms of the given probabilities. We may consider the time to find $\alpha$ as $d(\alpha)$ and so (1) gives the average time.
2. The Huffman Code allows for data compression. A text in which each letter $\alpha$ appears with probability $d(\alpha)$ will become a string of binary bits. Summation (1) gives the average number of bits used per letter.

Don’t confuse them!