2-3-trees are one instance of a class of data structures called balanced trees. These data structures provide an efficient worst case instantiation for the Dictionary abstract data type. Recall that a dictionary supports the operations SEARCH, INSERT, and DELETE on a set of items drawn from an ordered collection $U$, called the universe. Balanced trees perform each operation in worst-case time $O(\log n)$, where $n$ is the number of items stored in the dictionary at the time of the call.

1 The Data Structure

In a 2-3 tree, all the leaves are at the same level and each internal node has 2 or 3 children. All the records stored in the dictionary are held in the leaves. Recall that each record includes a key, the keys being used to order the records. Traversing the leaves of the 2-3 tree in left-to-right order yields the records in sorted order. Each internal node of the 2-3 tree stores a copy of the largest key that appears in any one of the leaves below it.

A 2-3 tree storing $n$ nodes has a height between $\lceil \log_3 n \rceil$ and $\lfloor \log_2 n \rfloor$. Thus, the length of any path from the root to a leaf is $O(\log n)$.

2 Operations

Search. A search in a 2-3 tree proceeds in a manner very similar to that of an ordinary binary search tree. The details and the proof of the desired logarithmic running time are omitted from this handout and left to the reader as a simple exercise.

Insertion. An insertion begins by performing a search to determine where the item would be located (if it were present in the tree). The item is inserted as a leaf at the resulting location. The new leaf's parent $p$ now may have either three or four children. If it has three children, the insertion is complete. Otherwise, $p$ is replaced by two nodes $p_1$ and $p_2$, where the two leftmost children of $p$ are placed under $p_1$ and the two rightmost children under $p_2$. Of course, the left-to-right order of the children is maintained. This operation is called a node partition and is repeated at each successively higher level of the tree along the path from the inserted item to the root, as required to remove nodes with four children. A special case arises if the root is replaced by two nodes; then, a new root node is created; its children are the two nodes newly formed from the old root. The details on how to update the copies of keys stored at internal nodes are straightforward and left as an exercise for the reader. It is easily seen that deletion takes $O(\log n)$ time.

Deletion. A deletion proceeds as follows: Again, a search is performed to find the item to be deleted. The leaf containing the item is deleted. At this point the parent $p$ has either one or two children. If it has two children, deletion is complete. Otherwise, if $p$ has only one child, $p$'s siblings are checked: If $p$ has a left sibling with three children, let $s$ be that sibling; otherwise, if $p$ has a
right sibling with three children, let $s$ be that sibling. If such an $s$ exists, node $s$ gives $p$ the child closest to $p$’s child. If no such $s$ exists, $p$ and its left sibling are merged into a single node. This process is repeated at each successively higher level of the tree along the path from the deleted item to the root, as required to remove nodes with one child. If the root ends up with one child, the root is simply removed and its sole child becomes the new root.

3 An Alternative Definition

Instead of storing a single key at each internal node, one can also store either one or two keys at such nodes: in the case where the node has two children, the maximum key appearing in a leaf of the tree rooted by the left child, and in the case where the node has three children, one additionally stores the maximum key appearing as a leaf in the tree rooted by the middle child.

By storing these keys at internal nodes, the search procedure may be somewhat more efficient, especially if nodes are stored as records on disk: all of the information necessary to choose which child to examine next is stored in the node itself. However, keeping this information properly maintained is just slightly more tedious.