**Dated: December 07, 2017**

**DATA STRUCTURE**

**2-3-4 Trees**

**SUBMITTED TO:** Ms. Sehrish Hina

**GROUP MEMBERS:**

Ubaid Ul Haq [16k-3631]

Muaaz Anwar [16k-3628]

Muhammad Nabeel Farooqui [16k-3627]

**SECTION:** A

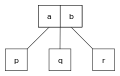
# Introduction:

In [computer science](https://en.wikipedia.org/wiki/Computer_science), a 2–3–4 tree is a self-balancing [data structure](https://en.wikipedia.org/wiki/Data_structure) that is commonly used to implement [dictionaries](https://en.wikipedia.org/wiki/Associative_array). The numbers mean a [tree](https://en.wikipedia.org/wiki/Tree_(data_structure)) where every [node](https://en.wikipedia.org/wiki/Node_(computer_science)) with children ([internal node](https://en.wikipedia.org/wiki/Internal_node#Terminology)) has either two, three, or four child nodes:

* 2-node has one [data element](https://en.wikipedia.org/wiki/Data_element), and internal has two child nodes.



* 3-node has two data elements, and internal has three child nodes.



* 4-node has three data elements, and internal has four child nodes.



2–3–4 trees are [B-trees](https://en.wikipedia.org/wiki/B-tree) of order 4, like B-trees in general, they can search, insert and delete in [**O**](https://en.wikipedia.org/wiki/Big-O_notation)**(log *n*)** time. One property of a 2–3–4 tree is that all external nodes are at the same depth. 2–3–4 trees are an [isometry](https://en.wikipedia.org/wiki/Isometry) of [red–black trees](https://en.wikipedia.org/wiki/Red%E2%80%93black_tree), meaning that they are equivalent data structures. In other words, for every 2–3–4 tree, there exists at least one red–black tree with data elements in the same order. However, can be difficult to implement in most programming languages because of the large number of special cases involved in operations on the tree.

# 

# Basic Operations:

* Insert
* Delete
* Search

**Insert:**

Since it is 2-3-4 tree so it would not have more than 3 data elements of each node. At very first 3 elements are inserted in the sorted order. When the fourth element comes so insertion is not allowed at that level as the place is full. We split the node from the middle element. That middle element becomes the root and left over are divided into left child and right child. This process is repeated until all the values are inserted.

**Delete:**

There are 2 cases in the deletion of 2-3-4 tree. After deletion it must be ensured that the number of keys in the node does not go below the minimum number.

1. Deleting the leaf node

This is the simplest case as the key from leaf node is only deleted. If whole node is deleted (there are no more keys) than key value from parent if moved to this node to maintain the properties of 2-3-4 trees.

2) Deleting the node with children

When we have to delete the root node that is having both left and right child. First we have to check if the number of elements are greater in left or right child. Suppose left child have the greater number of elements than right child. So the key is deleted and in order to maintain the ratio of keys and children the maximum element i.e successor is placed on the place of the key value that was removed. On the other hand predecessor is placed on the place of root node if the right child have the greater number of elements. If it is not possible to get value from children then the nodes are merged.

**Search:**

Elements lesser than the root node are on the left side while the elements greater than the root are on the right side. Searching is easier in 2-3-4 tree as compared to other trees. For example if we are searching for the value less than the root node so we will only have to traverse the left sub-tree. Same is the case with the greater value we will only have to traverse the right sub-tree. This will improve the time efficiency of 2-3-4 tree as compare to other trees.

Analysis

**2-3-4 Tree vs AVL:**

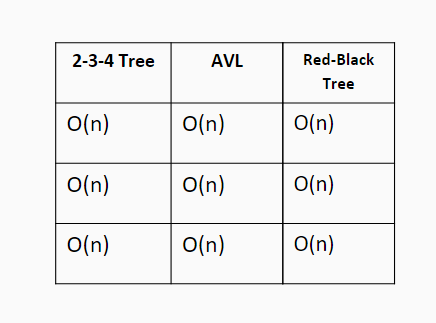
|  |  |  |
| --- | --- | --- |
|  | Time Complexity | |
|  | 2-3-4 Tree | AVL |
| Search | O(logn) | O(logn) |
| Insert | O(logn) | O(logn) |
| Delete | O(logn) | O(logn) |

In AVL tree while insertion and deletion the order of the tree violates so we have to perform the rotations to fix the tree while in 2-3-4 tree rotations is not required. Lesser number of childs means more traversing and more levels in AVL tree while greater number of childs gives lesser levels and less traversing in 2-3-4 tree.

**2-3-4 Tree vs RBT:**

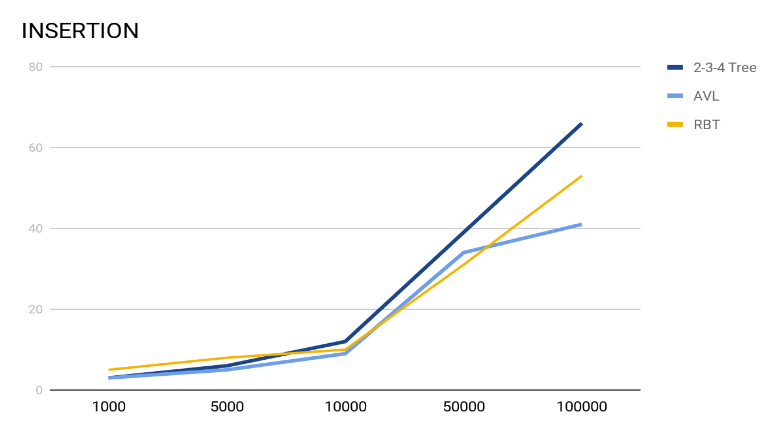
|  |  |  |
| --- | --- | --- |
|  | Time Complexity | |
|  | 2-3-4 Tree | Red-Black Tree |
| Search | O(logn) | O(logn) |
| Insert | O(logn) | O(logn) |
| Delete | O(logn) | O(logn) |

2-3-4 tree and red black tree are somehow same data structure. Expanding the node, splitting and merging in 2-3-4 tree is equivalent to the colour flippings and rotations in the red black tree. Introduction to red-black tree usually introduce 2-3-4 tree first, because they are conceptually simpler.



Space Complexity

# Graphical Analysis



# 

# 

# Applications:

* Dictionary.
* Search Engine and Database.
* Creating File Server.