What is a 2-3 Tree?

This lesson is an introduction to 2-3 trees, its properties along with an example, and the basic operations that this data structure offers.

We'll cover the following

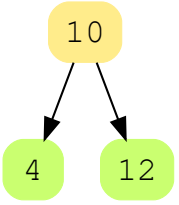
Introduction

A 2-3 Tree is another form of search tree, but is very different from a Binary Search Tree. Unlike BST, 2-3 Tree is a balanced and ordered search tree which provides a very efficient storage mechanism to guarantee fast operations. In this chapter, we will take a detailed look at 2-3 Trees' structure, the limitations it follows, and how elements are inserted and deleted from it.

One key feature of a 2-3 Tree is that it remains balanced, no matter how many insertions or deletion you perform. The leaf nodes are always present on the same level and are quite small in number. This is to make sure the height doesn't increase up to a certain level as the time complexity of all the operations is mainly dependent upon it. Ideally, we want the height to be in logarithmic terms because as the tree grows larger it will require more time to perform operations. In 2-3 Trees, the height is logarithmic in the number of nodes present in the tree. They generally come in 2 forms:

- *2 Node Tree*
- *3 Node Tree*

See the figures below to get the idea of how they are different. Given below is a 2-3 Tree with only two nodes. To keep it ordered, the left child key must be smaller than the parent node key. Similarly, right child key must be greater than the parent key.

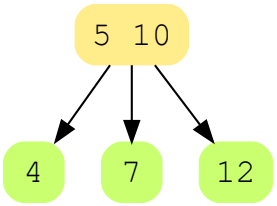


2-Node Tree

The figure below shows a **3-node** tree where each node can contain two keys and three

children at max. Here, parent node has 2 keys and 3 children which are all at the same level. Let's say the first key at parent node is **X** and we call the second one **Y**. As shown in the figure,

X key is greater than the left child and **Y** key is smaller than the child key at right. The middle child has the value that is greater than **X** and smaller than **Y**.



3-Node Tree

Concluding from the explanation above, 2-3 Trees acquire a certain set of properties to keep the structure balanced and ordered. Let's take a look at these properties.

Properties:

- All leaves are at the same height.
- Each internal node can either have 2 or 3 children.
- If the node has one key, it can either be a leaf node or has exactly two children. Let's say **X** is the key and LChild, and RChild refers to the left and right child of the node respectively, then:

$$LChild.Key < X < RChild.Key$$

- If the node has two keys, it can either be a leaf node or has exactly three children. Let's say **X**, and **Y** are the keys present at a node and LChild and RChild refer to the left and right child of the node respectively then:

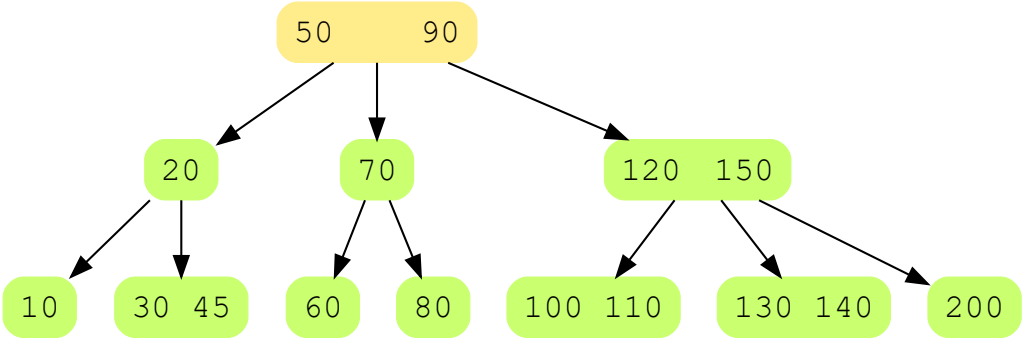
$$LChild.Key < X < MChild.Key < Y < RChild.Key$$

- Finally, the height of a 2-3 Tree with n number of nodes will always be lesser than:

$$log_2(n+1)$$

2-3 Tree (Example):

See the following example of a 2-3 Tree to get a better idea. Here, **20**, **70**, and **120-150** are *internal* nodes and all leaf nodes are present at the same level.



Operations:

The basic operations of 2-3 Trees are same as covered in previous lessons:

- Search
- Insertion
- Deletion

The time complexity of all three operations will also be in logarithmic terms. So they can be implemented to run in time $O(logn)$, where n is the number of nodes. We will discuss these operations and look at a few examples in the next chapter. *Adios!*

Back

Red-Black Tree Deletion

Next

2-3 Insertion

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/what-is-a-2-3-tree__introduction-to-trees__data-structures-for-coding-interviews-in-python)