Today - Lectures 12 & 13 CS163

1. Removal Algorithm
2. Tree efficiency (Topic #9)
3. Advanced Trees (Topic #10) - 2-3, 2-3-4

   Next: AVL & Red Black Trees!

Announcements:

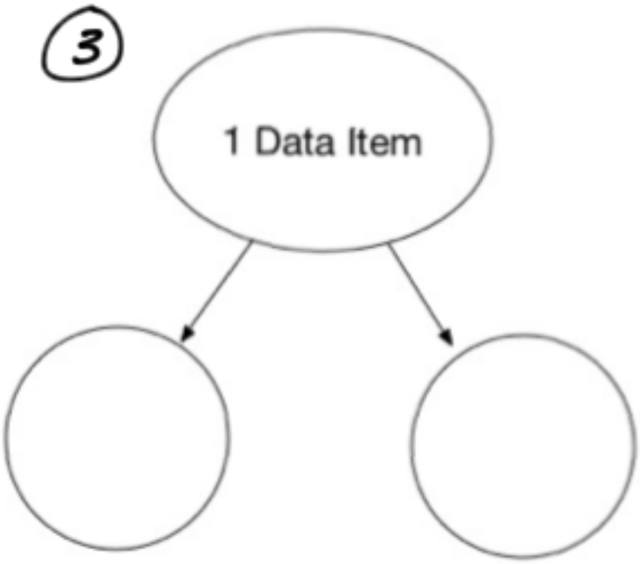
2-3 Tree

100% balanced, 100% of the time:

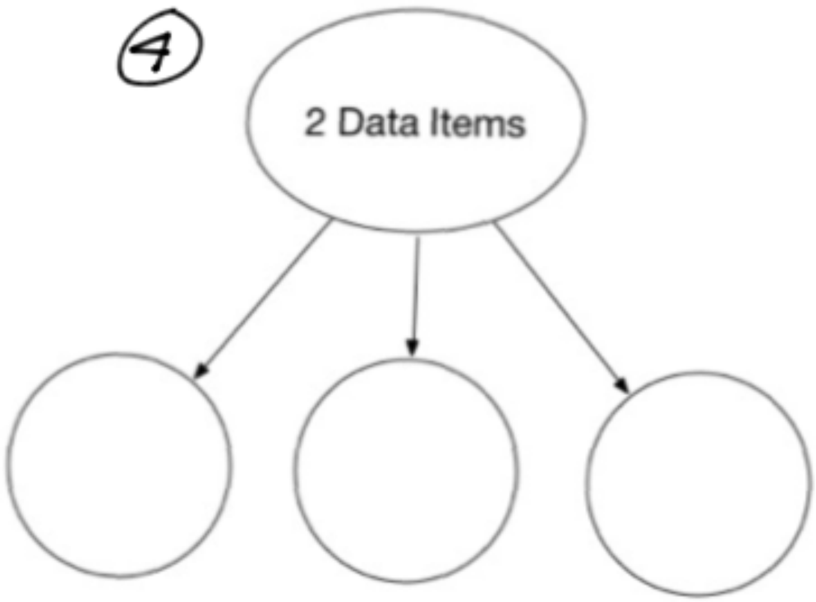
1

2

3



4



2-3 Tree Insertion Algorithm

1) Add at a leaf
2) if the leaf has only 1 Data item, store the new data in that node
3) if the node has 2 data items,
   a) find the middle data item
   b) push it up
   c) split the node

4) Much like the BST, but when a node has 2 data items, the $\boxed{\text{Left}}$ subtree is less than the smallest data item. –
The $\boxed{\text{MIDDLE}}$ subtree is greater than the smallest but less than the larger data item
The $\boxed{\text{RIGHT}}$ is greater than the largest " ".

```
struct node
{
    data * array[2];
        data ** array;
    node * child[3];
};
```

5 pointers

Build a 2-3 Tree:

50 20 15 55 32 10

What happens when data is sorted?

10 20 30 40 50 60

2-3-4 Tree


```
struct node
{
    data *items[3];
    node *child[4];
};
```

7 pointers per node

2-3-4 Insertion Algorithm

1) Travel to the appropriate leaf to add
2) As we traverse down the tree, ANYTIME a node with 3 pieces of data is encountered push UP the middle data item and "split" the node. Then, continue traversing.
3) There WILL ALWAYS be room in the leaf for the new item being added
4) Provides consistent run time perfomance at the cost of Memory overhead

Build a 2-3-4 Tree

50 20 15 55 32 10 45 25 70 5

What if data is inserted in sorted order?

10 20 30 40 50 60 70 80 90

ON Your Own... Create 2-3 & 2-3-4 Trees

7 12 3 13 21 5 8 50

Think about how the order inserted affects the shape & effectiveness of the memory used.