CS 400 Lecture 4:

2-3 Trees,
B-Trees,
JUnit

Andy Kuemmel, Instructor

Agenda for Tonight

5:30 Discuss current assignments….
5:45 Notes: 2-3 Trees and Insert
6:25 Break and team discussion time
6:45 Notes: B-Trees and Delete
7:25 Break
7:40 JUnit Demo
8:10 Putting it all together: AVLTreeTest.java
8:30 Adjourn

Current Assignments

Program 1: Will be graded by Wednesday next week

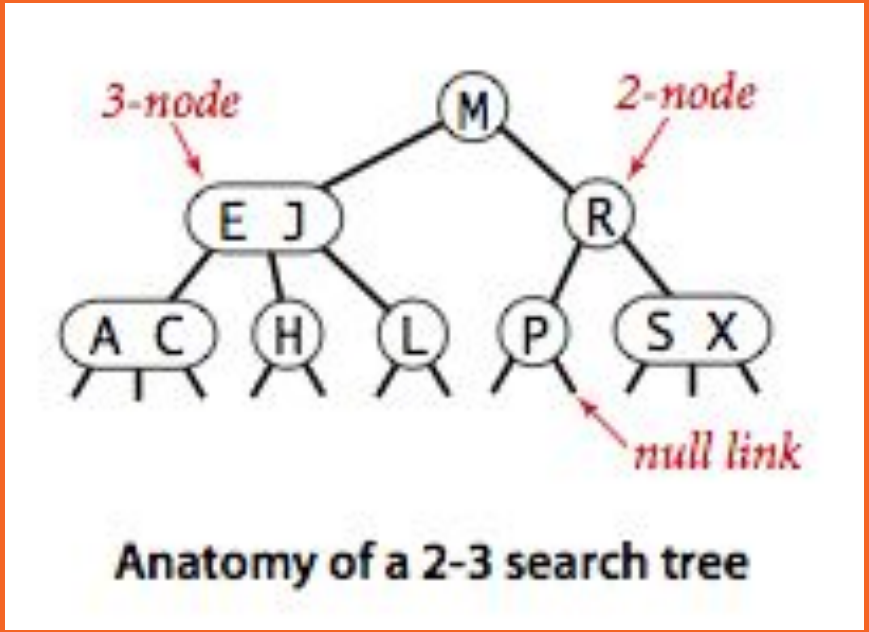
Homework 1: Quiz on Linux….team assignment…..

Program 2a: AVLTree.java, due 9/25, all have an automatic extension until Saturday
You will need a working AVLTree.java for Program 2b.

Program 2b: TestAVLTree.java, due 10/2, all have an automatic extension until 10/6

At the CS400 level, the TA and I will not directly debug your code line by line, but will give you guidance on how you can debug your code

Notes: 2-3 Trees
no rotating
2 or 3 children



Anatomy of a 2-3 search tree

https://algs4.cs.princeton.edu/33balanced/images/23tree-anatomy.png

Rules for a 2-3 Tree

1.) Each node has 2 or 3 Children

2.) Each node has 1 or 2 keys

3.) All Leaves are at the same level (the bottom)

4.) All Data is in sorted order.

5.) All inserting is done into a leaf.

Wikipedia:
https://en.wikipedia.org/wiki/2-3_tree

2-3 Tree Insert

```
If the root is null,
     create a new 2-node with this key
Else
     find the leaf where the insertion goes
     If leaf is a 2-node, make it into a 3-node
     If leaf is a 3-node
          find the middle value and propagate it up
          split the current node into two 2-nodes with smallest and largest

//Check the parent
If it is too large, push the propagation up the tree

If the propagation goes up to the root
     create a new 2-node with the propagated value
```

Try to insert: 10, 50, 30, 80, 100, 150, 90

X-Team Time

Take 5-10 minutes to talk to each other about completing the HW 1

These will not be your final project teams. You will choose your team members for the final project.

Break until:

Notes: B-Trees

same idea as 2-3 Trees
but larger capacity

Rules for a B-Tree with Degree N (not rigid)

1.) Each node has up to N children

2.) The root has between 1 and N-1 keys

3.) Every other node has between (N-1)/2 and N-1 keys

3.) All Leaves are at the bottom

4.) All Data is in sorted order.

5.) All inserting is done into a leaf.

Visualizing: (try Max Degree = 5)
https://www.cs.usfca.edu/~galles/visualization/BTree.html

B-Trees were developed for Databases

External disk storage broken into blocks.

It takes a long time to access a block.

The programmer can customize a B-Tree to match the size of a block.

Once the disk block is retrieved into working memory, access is fast.

A very large N leads to a tree with a small height.

a 2-3 Tree is a B-Tree with N=3

1.) Each node has up to N children

2.) The root has between 1 and N-1 keys

3.) Every other node has between (N-1)/2 and N-1 keys

4.) All Leaves are at the bottom

5.) All Data is in sorted order.

6.) All inserting is done into a leaf.

Deleting from a B-Tree

1. Find the element to delete (if not present...do nothing)
2. If the element is in an internal node
        Find the inorder predecessor   (.....or successor)
        Overwrite the deleted element with predecessor (or s.)
        Remove the predecessor from its leaf node
    Else
        just remove the element from the leaf node

3. If current node is too small (see rule 3)
        If the sibling has enough elements to share
            Take from parent and promote sibling
        Else
            Merge current node with sibling and take from parent
4. Repeat Step 3 for ancestors

Break Until:

JUnit4

A package that is built into Eclipse

Helps us run tests and see results

Setting up AVLTreeTest.java

1. Find your Program2 project.

2. Right-click on project:
   Build Path → Add Libraries → JUnit → JUnit4

3. Paste or Copy in the AVLTreeTest.java from Canvas

4. Read the code there and run it

5. Add methods and run it