Week 2

ASSIGNMENTS
h0 available and due before 10pm on Monday 1/28
h1 available and due before 10pm on Monday 2/4
p1 available and due before 10pm on Thursday 2/7

TA Lab Consulting - See schedule (cs400 home pages)
Peer Mentoring available - Friday 8am-12pm, 12:15-1:30pm in 1289CS

TAs and Peer Mentors will focus on helping students get p1 JUnit tests running in Lab. We can not guarantee that we can get your personal computers configured.

Module: Week 2 (start on week 3 before next week)

THIS WEEK

- Ready Set Program 1!
  - Read Assignment - there are getting started instructions there
  - Create and configure project for JUnit5, compile and run **TestDS_My**
  - Put tests in DataStructureADTTest (not the sub class).
- Testing: JUnit5
- Java: inner classes
- Determining Height of a Tree (Recursion Review)
- Binary Search Trees (BST) (Review?)
  - operations
  - implementing
  - complexities
- Classifying Binary Trees
- Balanced Search Trees
- George Adelson-Velsky and Evgenii Landis

NEXT WEEK

- X-team Exercise x1 (in-class exercise with your assigned teams)
- Watch for instructions to find your team number and how to meet

Writing JUnit5 Tests

What is JUnit?

- a testing framework for Java class
- has test runners that runs the test you write

How do I use it?

1. Add Junit to the build path
2. Add JUnit test case (class)
3. Add units test to test class
4. Run the test class

What does a @Test method look like?

- code to try things
- code to detect problems -> if (bad conditions) fail( )
  assertEquals(expr1, expr2)

Details:

1. **Import JUnit classes**

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class TestClass {

   @Test
   void test123_try_someting_check_results() {
      // try something
      // if (cond_expression) fail("descriptive failure message")
      // assertEquals(  expr1, expr2 )
      // fail if something unexpected occurs or if something expected does not occur.
   }

}
```

Eclipse imports these

@Test - annotations

Java's Inner Classes

https://docs.oracle.com/javase/tutorial/java/javaOO/innerclasses.html

What is an inner class?

- a class defined in another class
- make the inner class private, make members of inner class private

search

Define a generic Tree cass using an inner class for the individual node type of the tree.

```
public class Tree<K extends Comparable<K>> {
    private class Treenode<K> {
        private K data;
        private ListADT<Treenode<K>> children;
        //private constructors and methods
        ...
    }

    private Treenode<K> root;
    private int size;
    public Tree( ) { ... }        (constructor)

    add( );
    remove( );


}
```

t[ ]
height = 0

= levels.

Determining Height of a Tree (or sub-tree)

t1 → A
h = 1
height of a tree

tn → A, B, C, D
height = 3.

We define height of a tree as the # nodes on path from the root to the deepest leaf.

Write a recursive definition for the height of a general tree.

height (tree t)
h = 0, if t is null
h = 1, if root is leaf
h = 1 + max height of children — recursion if not a leaf

base cases

recursive cases

Complete the recursive height method based on the recursive definition.
Assume the method is added to a Tree class having a root instance variable.

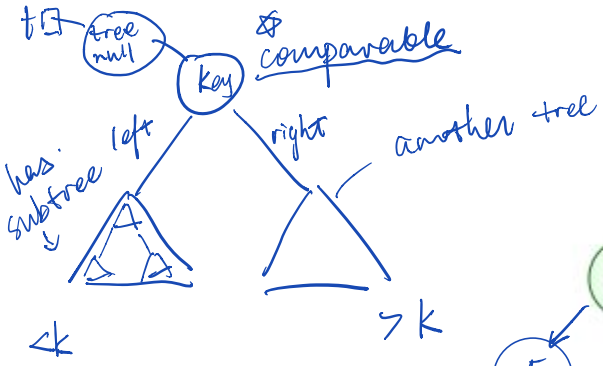
```
public int height() { if (root == null) return 0;        }
                      else return height(root )
```

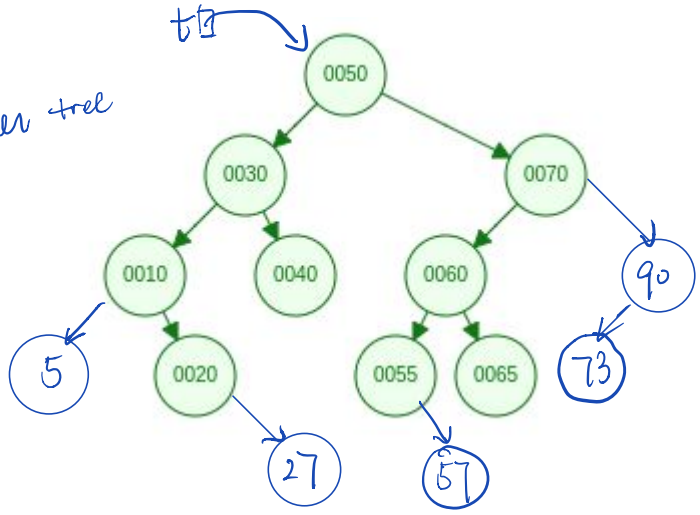
```
private int height ( TreeNode<T> node ) {

}
```

X Duplicate.

Binary Search Tree (BST) Review

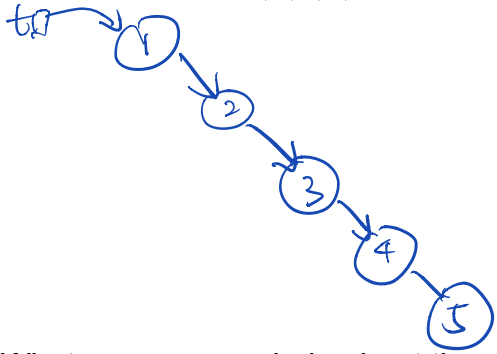
Image created at: https://www.cs.usfca.edu/~galles/visualization/BST.html





Insert 5, 27, 90, 73, 57 into the above BST tree (recall binary search algorithm)

Insert the values 1,2,3,4,5 into an empty BST sorted number → linear structure



What can you conclude about the shape of a BST when the values are inserted in sorted order?

It's linear O(N) FATAL FLAW

- shape depends upon insert order
- complexity depends upon shape

Will you only get this shape if inserted in sorted order?

X

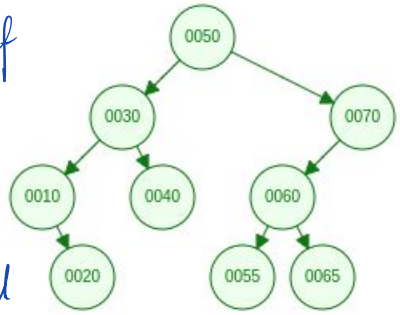
1, 9, 2, 8, 3, 7, 4, 6, 5

Practice Deleting from a BST

Delete 90 from this tree. no 90,
Delete 40 and then 65 from this tree.

① unlink 40
by setting right child of
30 to null

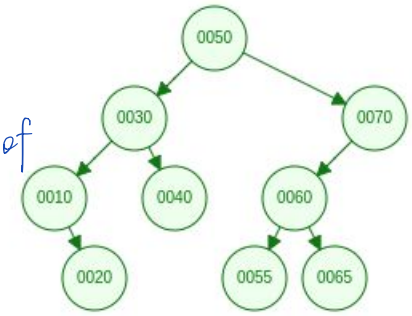
② unlink 65,
set right child of 60 to null



Delete 10 and then 70 from this tree.

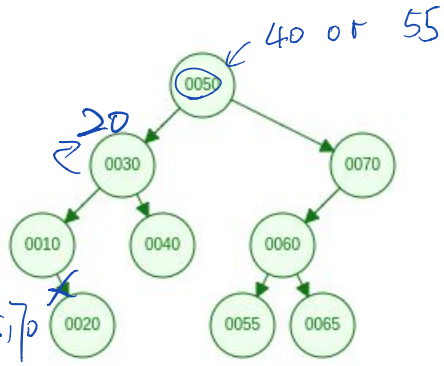
10: set left child
of 30
to 20 (right child of
10)

70: set right child
of 50 (root)
to 60 (left child of
70)



How do you delete 30 or 50 from this tree?

delete 30: ?



Δ in-order traversal

10, 20, 30, 40, 50, 55, 60, 65, 70

LVR

calling recursively

Δ Pre order

VLR 50, 30, 10, 20, 40, 70, 60, 55, 65

Δ post order

LRV 20, 10, 40, 30, 55, 65, 60, 70, 50

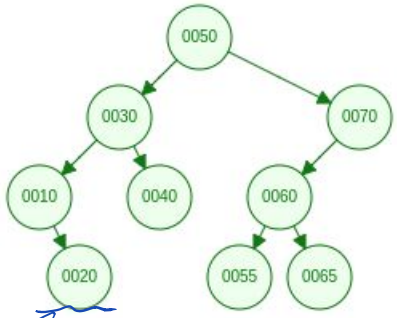
LRV 20, 10, 40, 30, 55, 60, 50

Δ level order
50, 30, 70, 10, 40, 60, 20, 55, 65

Delete 30 from this tree using the in-order predecessor

① replace 30 with value of in-order predecessor

② delete 20 from left of 30 subtree

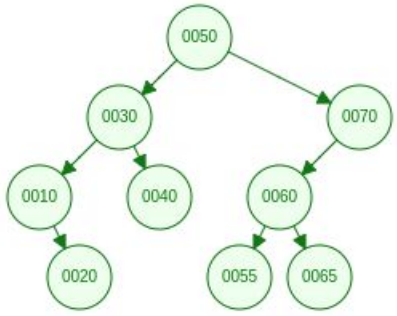


in-order predecessor of 30

Delete 50 from this tree using the in-order successor

① replace 50 with value of in-order successor

② delete 55 from right subtree



How do we find in-order predecessor or in-order successor?

furthest to right of left-subtree

greatest value in left-subtree

go left, then right as far as possible

— — — left right-subtree

lowest value in right-subtree

go right, then left . . .

Implementing BST Operations

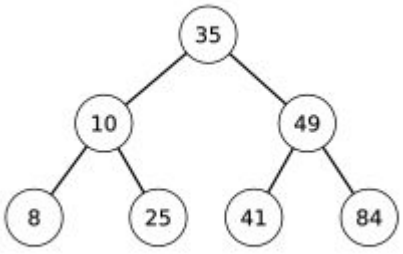
public boolean lookup(K key) { return lookup(root, key); }

private

helper method.

assume key is not null

```
boolean lookup(BSTnode<K> n, K key) {
```




```
    if (key == null) throw IllegalArgument;
    if (key.compareTo(n.getKey()) == 0)
        return true;
    if (key.compareTo(n.getKey()) < 0)
        return lookup(n.getLeft(), key);
    return lookup(n.getRight(), key);
```

void

```
public insert(K key) { root = insert(root, key); }
```

```
/** pre-cond: key is NOT null */
private BSTnode<K> insert(BSTnode<K> n, K key)
        throws DuplicateKeyException {
    if (key == null) throw IllegalArgument("null key");
    if (n == null)  create a new node {
        BSTNode<K> newNode = new BSTnode<K>(key, null, null)
        return newNode;
    }
    if (n.key.equal(key)) throw DuplicateException n;
    if (n.key.compareTo(key) < 0)
        n.left = insert(n.left, key);
    else
        n.right = insert(n.right, key)
    return n;
```

public void delete (K key) { root = delete (root, key); }

Implementing BST Operations

delete

```
/** pre-cond: key is not null */
private BSTnode<K> delete(BSTnode<K> n, K key) {
    if (n == null) throw exception("key not found");
                   return null.
    if (key.compareTo(n.getKey()) == 0)  // find match, delete this
```

CASE #1: n has no children

```
        if (n.left == null && n.right == null)
            return null;
```

CASE #2: n has one child

// return n's other child to parent

```
        else if (n.left == null)  return n.right;
        else if (n.right == null) return n.left;
```

CASE #3: n has two children

```
        n.key = inOrderPredecessor(n.left)   //return key not node
        n.left = delete(n.left, n.key);
        //1. Find a replacement key
        //2. Replace n's key with Replacement
        //3. Delete replacement key from a prop Subtree
        // Key Concept: Find replacement
                        the minimize changes to structure


    return n.
```

Complexities of BST Operations

Problem size: N = # of data items

print: O(N) ~ must traverse all nodes

lookup: O(log N) -> O(H) H: height of the tree ($\log N \leq H \leq N$)

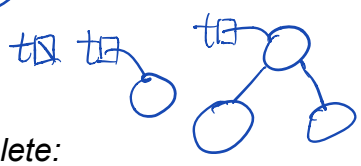
insert: O(H)

delete: O(H)

k-ary Tree Classification Terms

Binary at most 2 children
Trinary 3
K-ary K.

Perfect: (referred to as full in some readings) $(2^n - 1)$ nodes



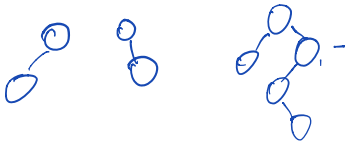
all interior trees have k children
all leaves at same level

Complete:

a heap is complete
- interior nodes have k children
- prefect to 2nd last level, last level filled left to right with leaves

Height-Balanced:

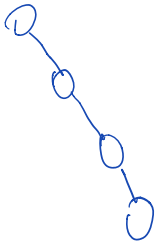
Every node has subtree height that differ by at most one.



-> not height balanced.

Balanced

a tree is balanced if its insert operation is $O(\log_k N)$



X balanced

Balanced Search Trees

Goal: keep height at $O(\log_k N)$ → # of nodes

Idea: have insert + delete maintain balance

How do we detect and fix?

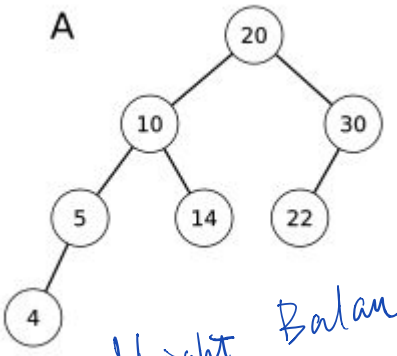
1. check height of each subtree (compute balanceFactors)
2. If |B.F.| > 1 – must rebalance

Balance Factor:

BF = height of left subtree – height of right subtree

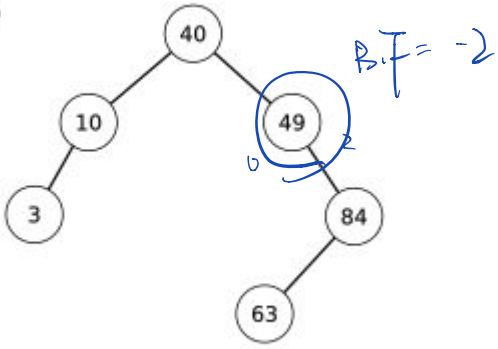
Height-Balanced Practice

A



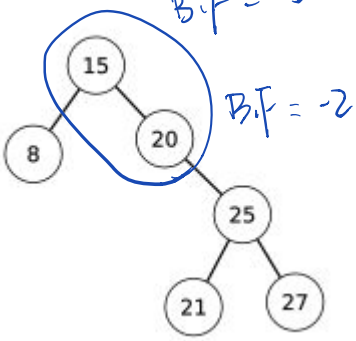
Height Balanced

B



B.F = -2

C

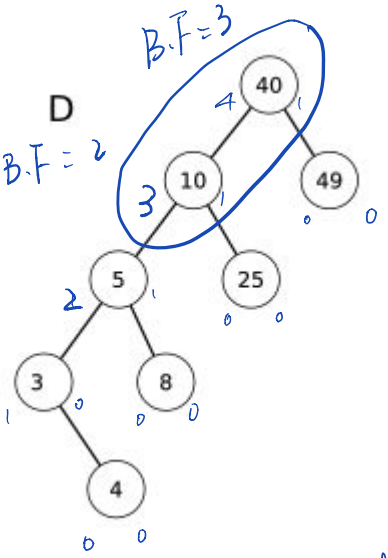


B.F = -3

B.F = -2

not height balanced

D

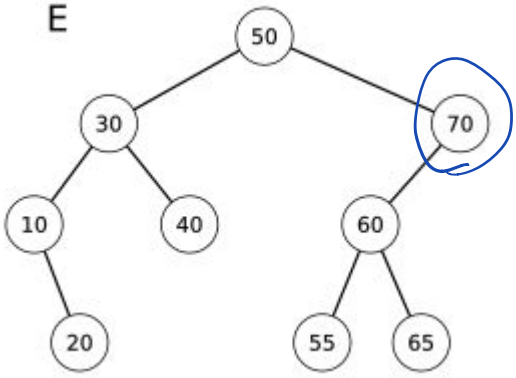


B.F = 3

B.F = 2

not height balanced

E

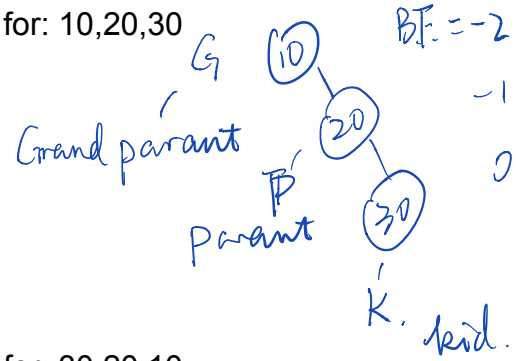


B.F = 2

not height balanced

Balancing I - Practice

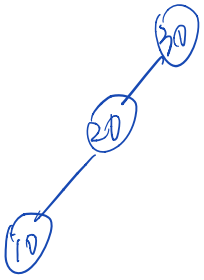
Draw BST for: 10,20,30



BF. = -2
-1
0

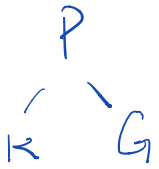
left rotate
P. left = G
return P

Draw BST for: 30,20,10

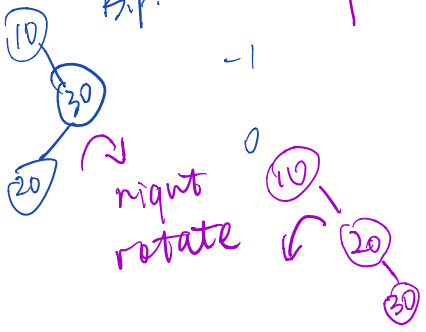


B.F = 2
2
0

right rotate
P. right = G
return P



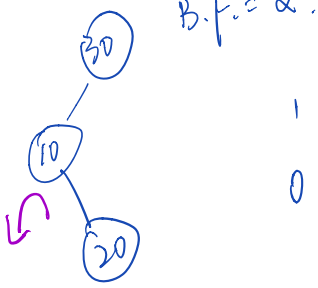
Draw BST for 10,30,20



B.F. = -2
-1

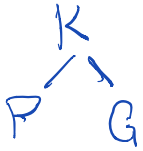
right-left rotate
K. right = P
K. left = G
return K

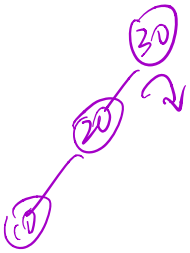
Draw BST for 30,10,20



B.F. = 2.
1
0

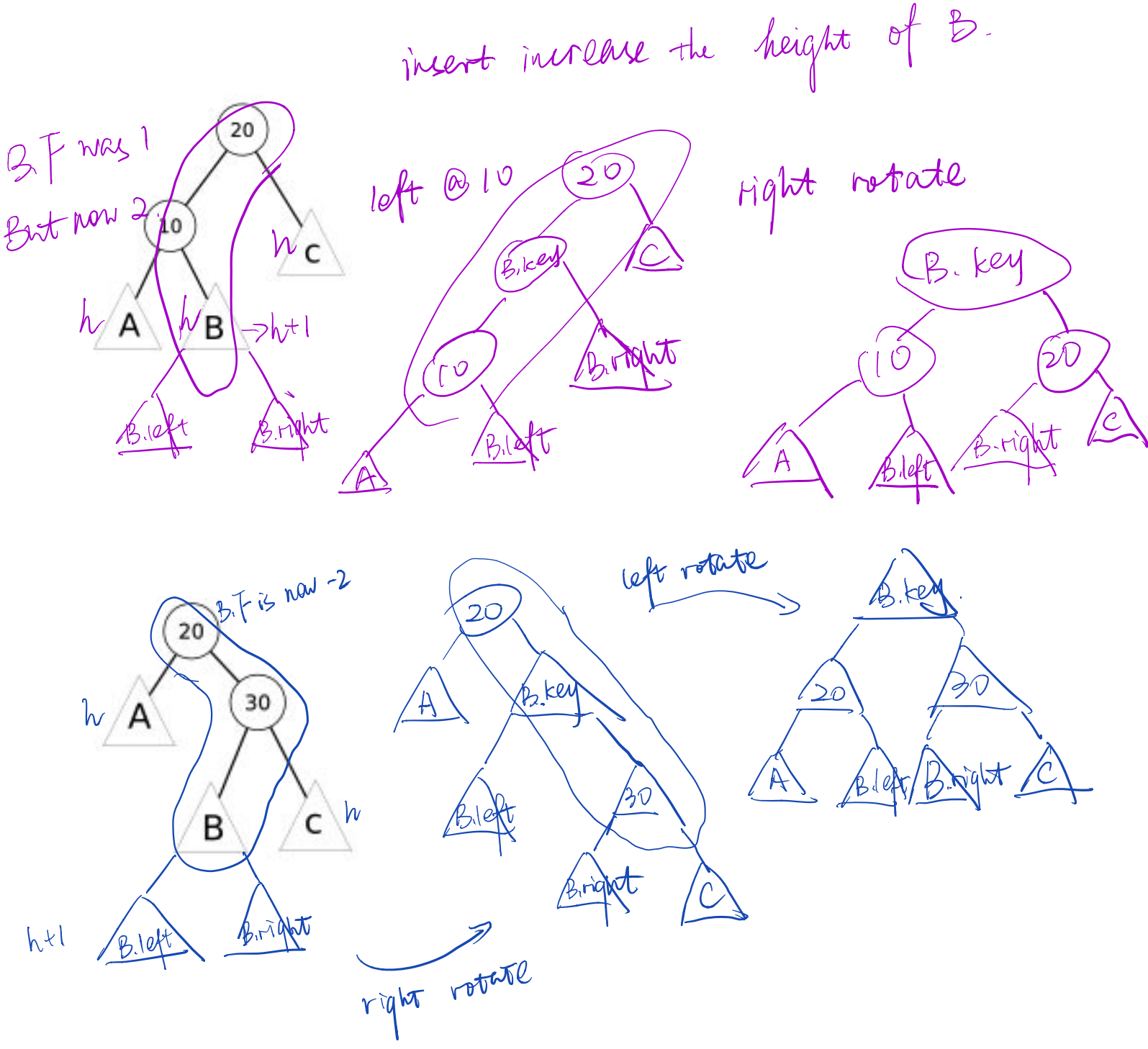
left-right rotate
K. left = P
K. right = G



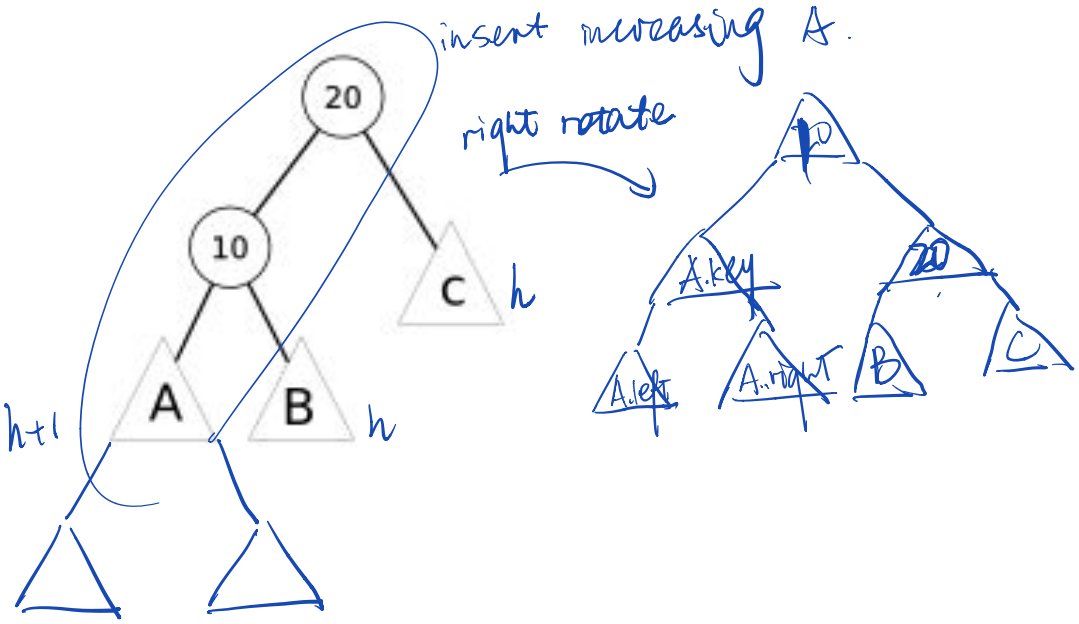


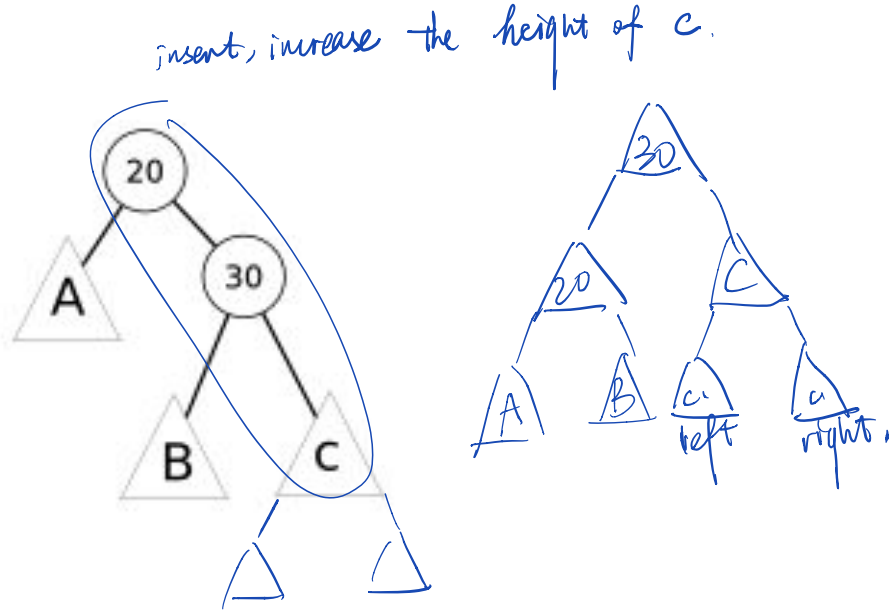
Balancing II - General Balancing

Assume: a triangle represents sub-tree of unknown height, but that trees shown are *height-balanced*, and show the relative difference (balance factor of -1, 0, 1)



Balancing III - More General Balancing





Georgy Adelson-Velsky and Evgenii Landis

height balancing

use one of 4 rotation sequence

Complexity of operations

still O(H) bad AVL paper "proved"

$H \approx 1.44 \log_2 N$

Implementation

must know or compute B.F for each Node

detect of balance when |B.F.| > 1.

How can we know height of each sub-tree?