Data structures lab – week 5

Welcome back!

Week 4 recap

- More hints
- Comparing LL and BST
- Assignment 2 Q & A
  - Did you all finish yet?

Week 4 class evaluation

- We're down to only 6 respondents :-(
  - Probably midterm madness
- Selected comments (slightly edited):
  - ”A subscription option for the blog … for those of us... that would sooner check [email] than the website
  - ”More examples of actual C++ code”
  - ”I already worked with C++ so a lot of the stuff is old news to me”
- Shows the diversity of a class!
- Full survey results found online

Outline

- Announcements
  - The blog!
- Balanced trees
  - Something new and exciting
  - Analysis
  - Pretty graphs
- Assignment 3

Hints for success

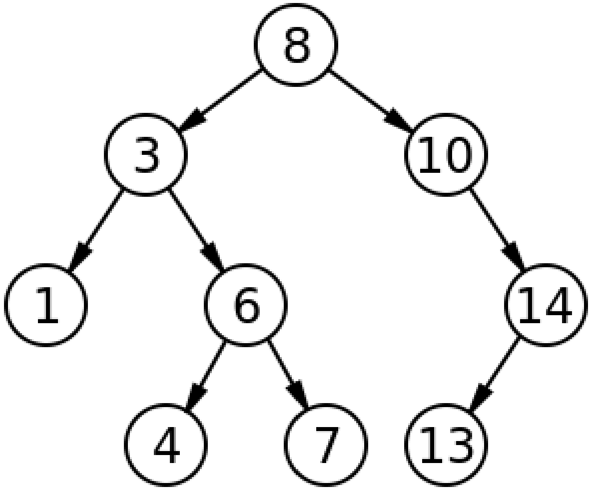
- Hint number 1: Read the assignment
- Hint number 2: Look at your code
- Hint number 3: Comply with standards
- Hint number 4: Use large test cases
- Hint number 5: Use the terminal
- Hint number 6: Use IX and g++
- Hint number 7: Fear the NULL
- Hint number 8: Use a debugger
- Hint number 9: Start earlier

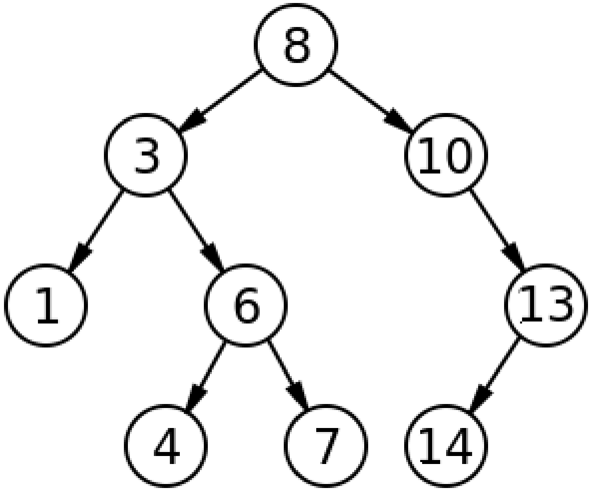
Hints for success

- This is almost ten!
- We need a tenth.
  - Right? Yes
  - Thou shalt send me suggestions either on blog on email.
  - I will make a poll.
  - Thou shalt vote for your favorite before next class.
- By the way, have you checked out the blog?

Wake-up quiz

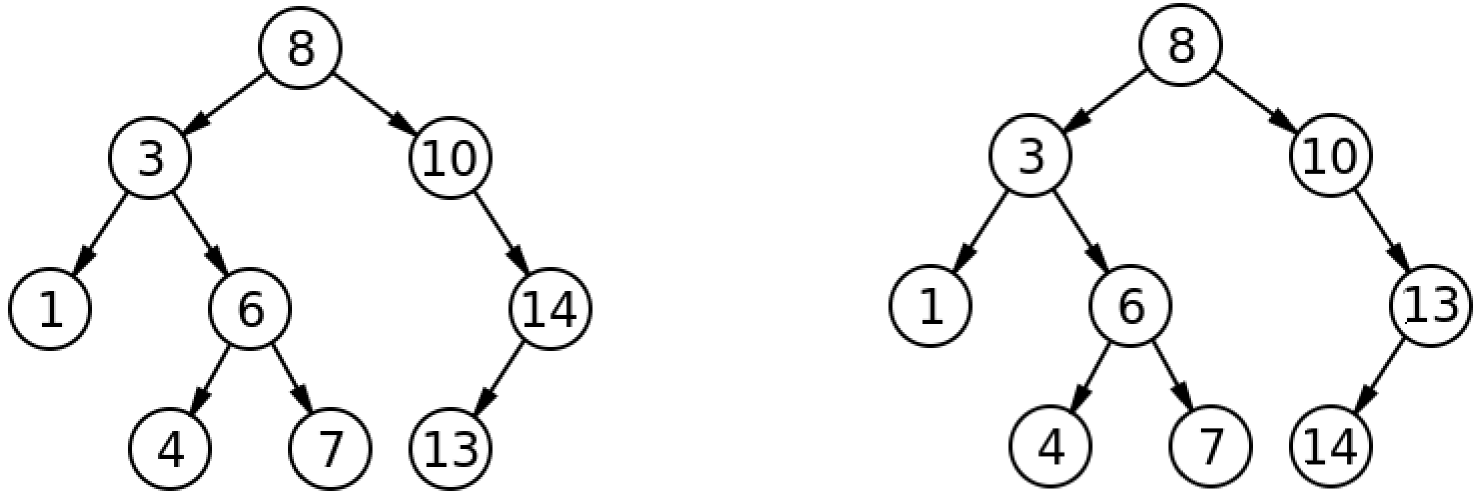
- Which of the following trees is a binary search tree?





Wake-up quiz

- Which of the following trees is a binary search tree?



- The left one is

Balanced trees

- BST is not balanced.
  - We've been talking a lot about this
- BST is pretty good in the average case
- We still want balance though
  - To guarantee O(lg n) height of our trees
- Cormen *et. al.* has the answer.

Balanced trees

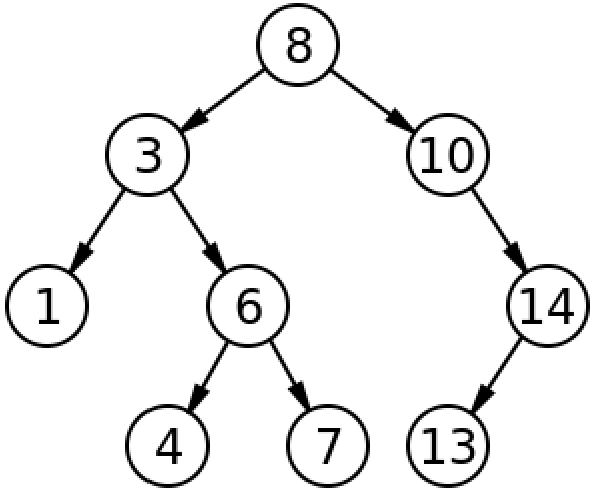
- Red-black trees!
- Invented by Bayer, 1972, based on B-trees.
- Guibas-Sedgewick, 1978, analysed and invented the red-black color idea.
  - We'll get back to Sedgewick later

Red-black trees

- Properties:
  - Every node is either red or black
  - The root is black
  - Every leaf is black
    - In Cormen, every leaf is a special NIL node.
  - If a node is red, both children are black
  - All simple paths from a node to descendant leaves contain the same number of black nodes.

Red-black trees

- Here is a BST



- Could this be turned into a red-black tree?
  - … If I'm allowed to color the nodes?

Red-black trees

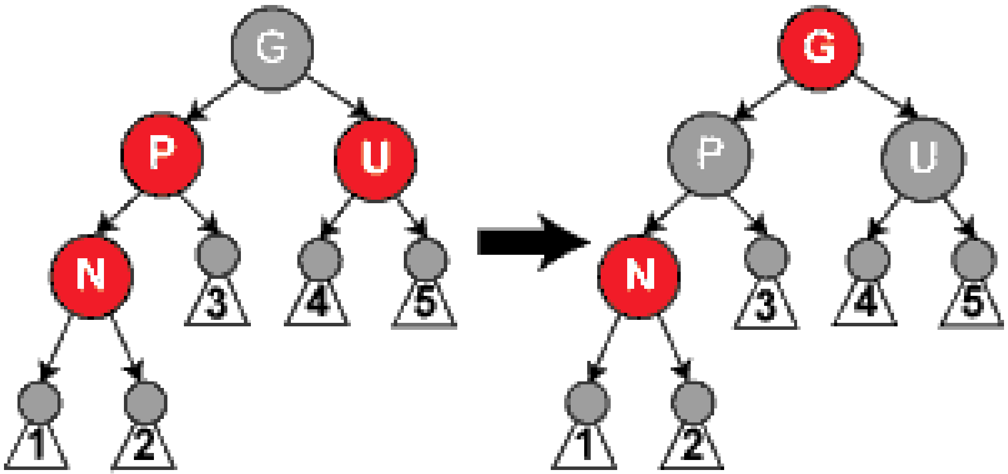
- Balancing happens at insertion
  - And deletion
- All other operations are the same as for BST
  - Yes, this is pretty clever.
- New question: Can we balance efficiently?

Red-black trees

- Insert operation outline for node *x*:
  - Insert *x* into tree
    - Using the standard BST method
  - Color *x* red
  - *Fix* the tree to comply with properties
- Concepts:
  - *x's* Uncle (*U*) is *x*'s parent's sibling
  - *x*'s Grandparent (*G*) is *x*'s parent's parent
    - Just like real life right?

Red-black trees

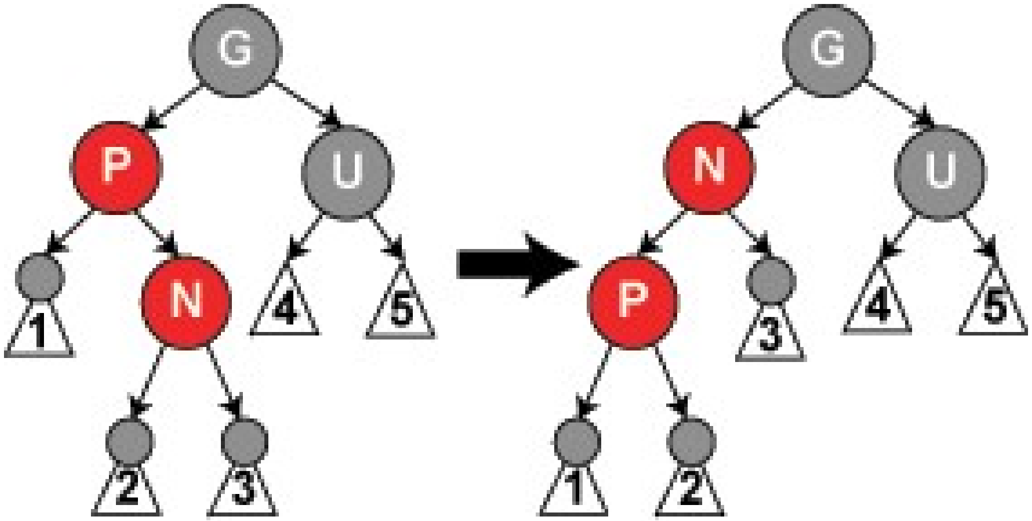
- Parent and uncle are red
- Change them to black
- Change their parent to red



- Apply recursively, so root ends up black

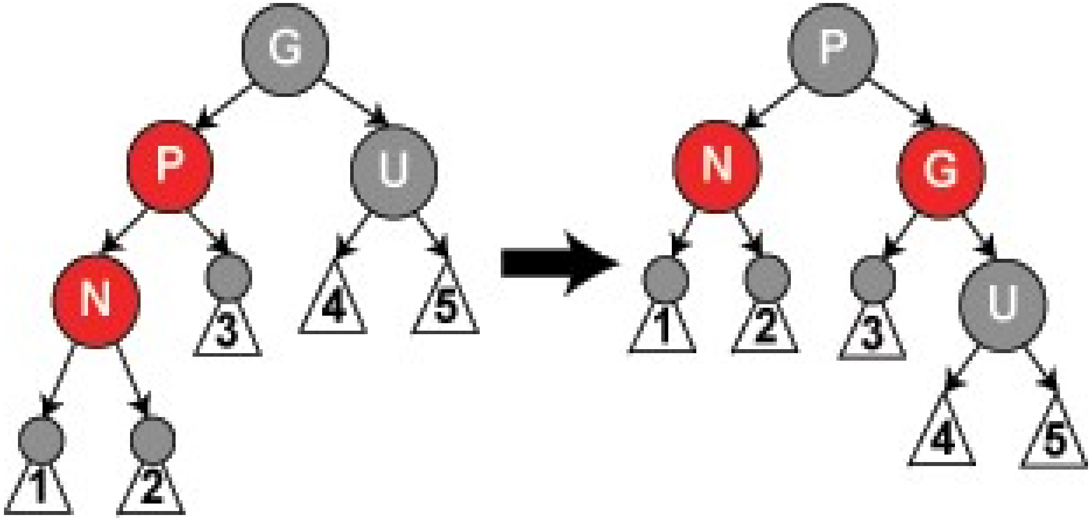
Red-black trees

- Parent is red, uncle is black
- *x* is right child of *P*, *P* is left child of *G*
- Rotate-Left at *P*



Red-black trees

- Parent is red, uncle is black
- *x* is left child of *P*, *P* is left child of *G*
- Rotate-Right at *G*

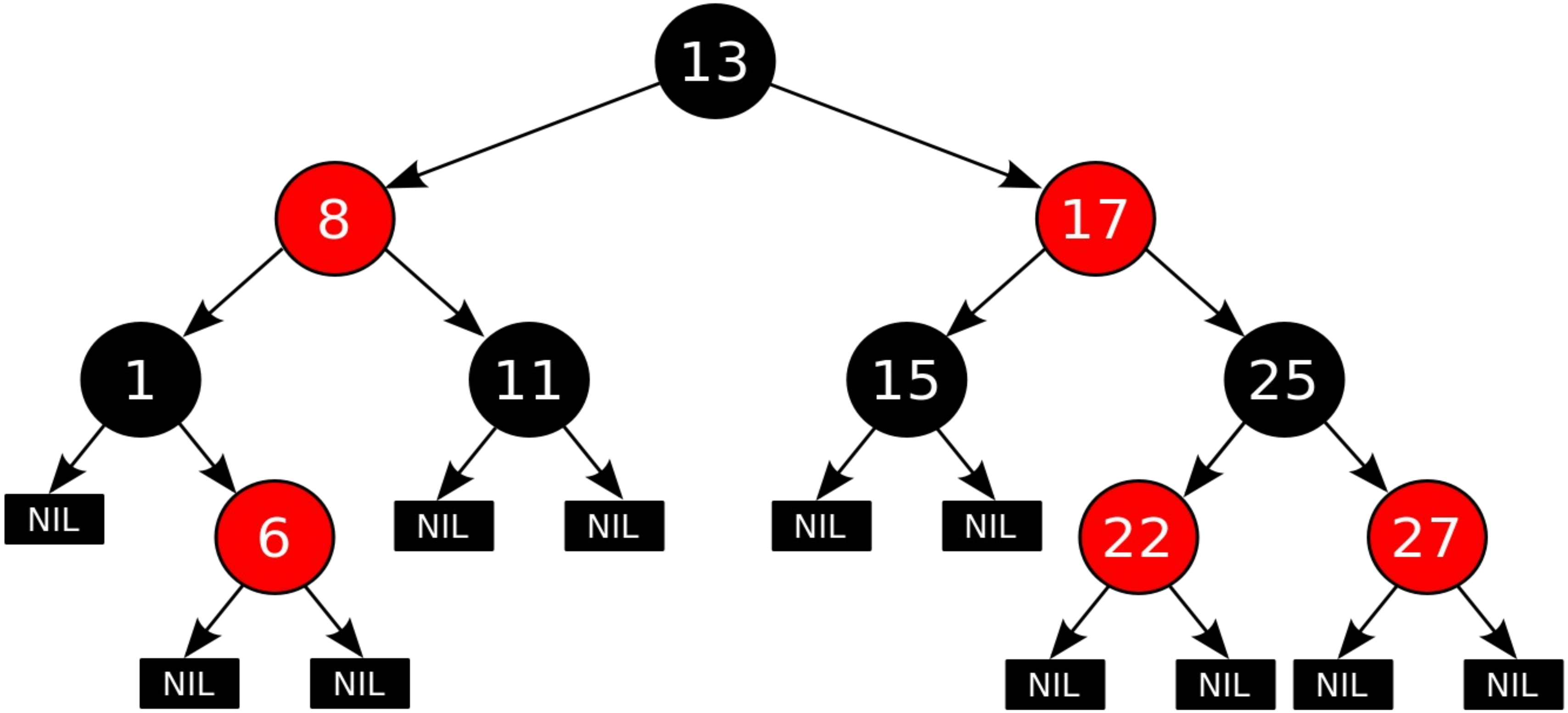


Red-black trees

- And then there's the opposite cases
- I won't go into detail with those
- Cormen *et. al.* is very detailed
  - Very!
- Implementing red-black trees can be difficult!
- Robert Sedgewick: ”Can we do better?”

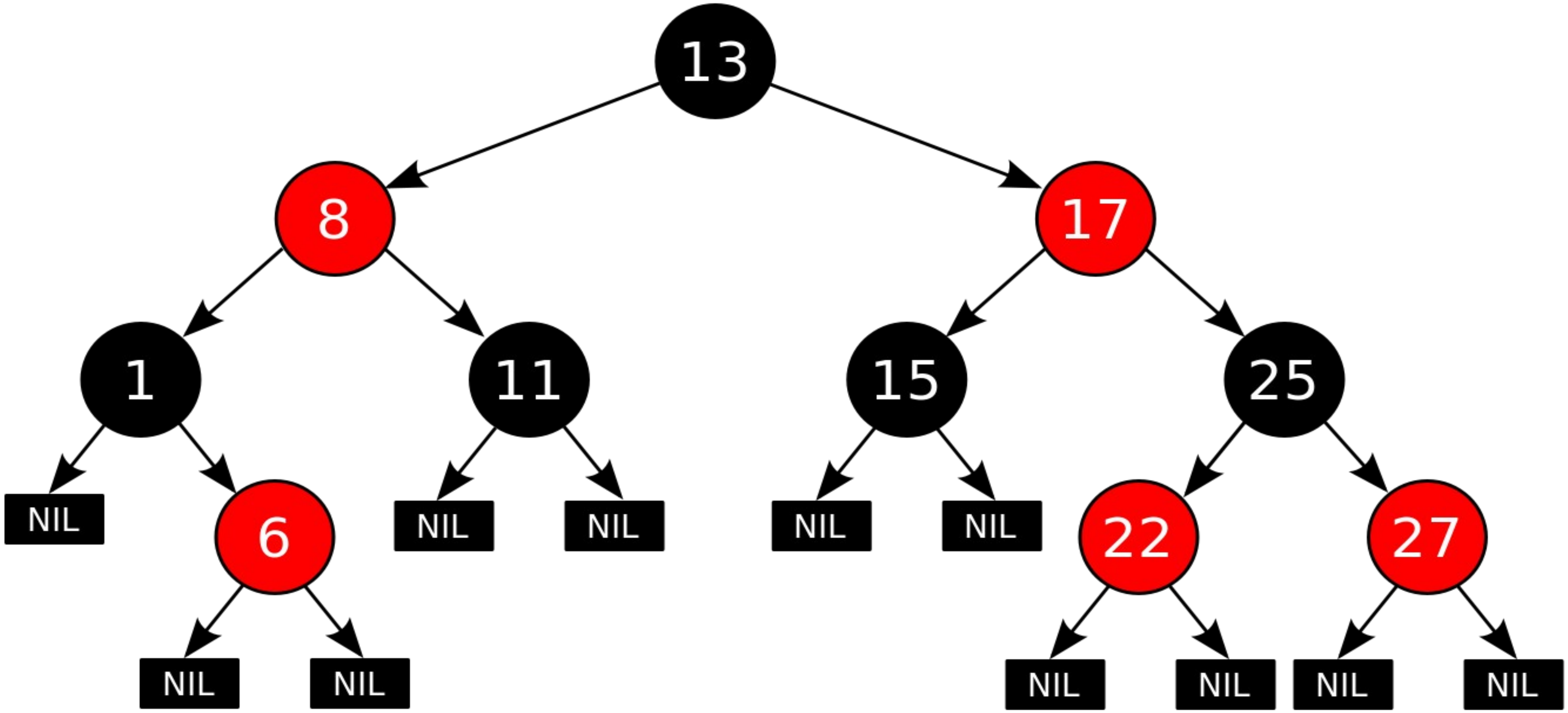
Wake-up quiz – RB trees

- Is this a red-black tree?



Wake-up quiz – RB trees

- Is this a red-black tree?



- Yes!

Balanced trees – something new

- Robert Sedgewick, Fall 2007:
  - ”Can we do better?”
- Introduced the left-leaning red-black tree.
  - Require red nodes to ”lean” left
- What does left-leaning mean?
  - Good to know 2-3-4 trees
  - I'll give you a quick tour.

Left-leaning RB tree

- Slides found at: http://www.cs.princeton.edu/~rs/talks/LLRB/RedBlack.pdf
- I will briefly go through some of them.
- Used with kind permission by Robert Sedgewick himself.

Left-leaning RB trees

- Claims to be faster than normal RB trees.
- How do we test the claim?
  - Look at the analysis
  - Try it out!

LLRB analysis

- We want to test LLRB trees.
- We want to compare the *find* running time for LLRB with RB
- We want to compare the *insert* running time for LLRB with RB
- Let's throw a normal BST in there as well
- We hope we can see a difference
  - This is our hypothesis

LLRB analysis – recipe

1) Implement BST (already done)
2) Implement RB tree (from Cormen *et. al.*)
3) Implement LLRB (from Sedgewick)
4) Run tests
5) Look at results
6) Conclude

Implementation – general

- Add color boolean to each node
  - Red is true
  - Black is false
- Add a special `NIL` node

```
const bool RED = true;
const bool BLACK = false;

struct RBNode {
    int key;
    bool color;
    RBNode * left;
    RBNode * right;
    RBNode * p;
    RBNode();
    RBNode(int,bool);
} nilNode;
```

Implementation – general

- Already have basic algorithms in place
- Inorder tree walk is good for seeing if your tree is correct.

```
void inorderTreeWalk(RBNode * x) {
  if (x != &nilNode) {
    inorderTreeWalk(x->left);
    cout << x->key << endl;
    inorderTreeWalk(x->right);
  }
}
```

Implementation – general

- Tree search, also used by all three

```
RBNode* iterativeTreeSearch(RBNode * x, int k)
{
  while (x != &nilNode && k != x->key) {
    if (k < x->key)
      x = x->left;
    else
      x = x->right;
  }
  return x;
}
```

(2) RB implementation

- Look in Cormen, chapter 13
- Implement rotations
- Implement insert
  - Basically change the BST insert a bit
- Implement insert-fixup
  - This is the tricky part!

(2) RB implementation

```
void RBLeftRotate(RBTree& rbbst, RBNode& x) {
   RBNode * y;
   y = x.right;
   x.right = y->left;
   if (y->left != &nilNode)
      y->left->p = &x;
   y->p = x.p;
   if (x.p == &nilNode)
      rbbst.root = y;
   else if (&x == x.p->left)
      x.p->left = y;
   else
      x.p->right = y;
   y->left = &x;
   x.p = y;
}
```

(2) RB implementation

```
void RBRightRotate(RBTree& rbbst, RBNode& y) {
   RBNode * x;
   x = y.left;
   y.left = x->right;
   if (x->right != &nilNode)
      x->right->p = &y;
   x->p = y.p;
   if (y.p == &nilNode)
      rbbst.root = x;
   else if (&y == y.p->left)
      y.p->left = x;
   else
      y.p->right = x;
   x->right = &y;
   y.p = x;
}
```

(2) RB implementation

- Insert, 22 lines of code
  - Without comments
- Insert-fixup, 40+ lines of code
  - Without comments
- Maybe it doesn't scare you away
  - But there sure are many places where things can go wrong

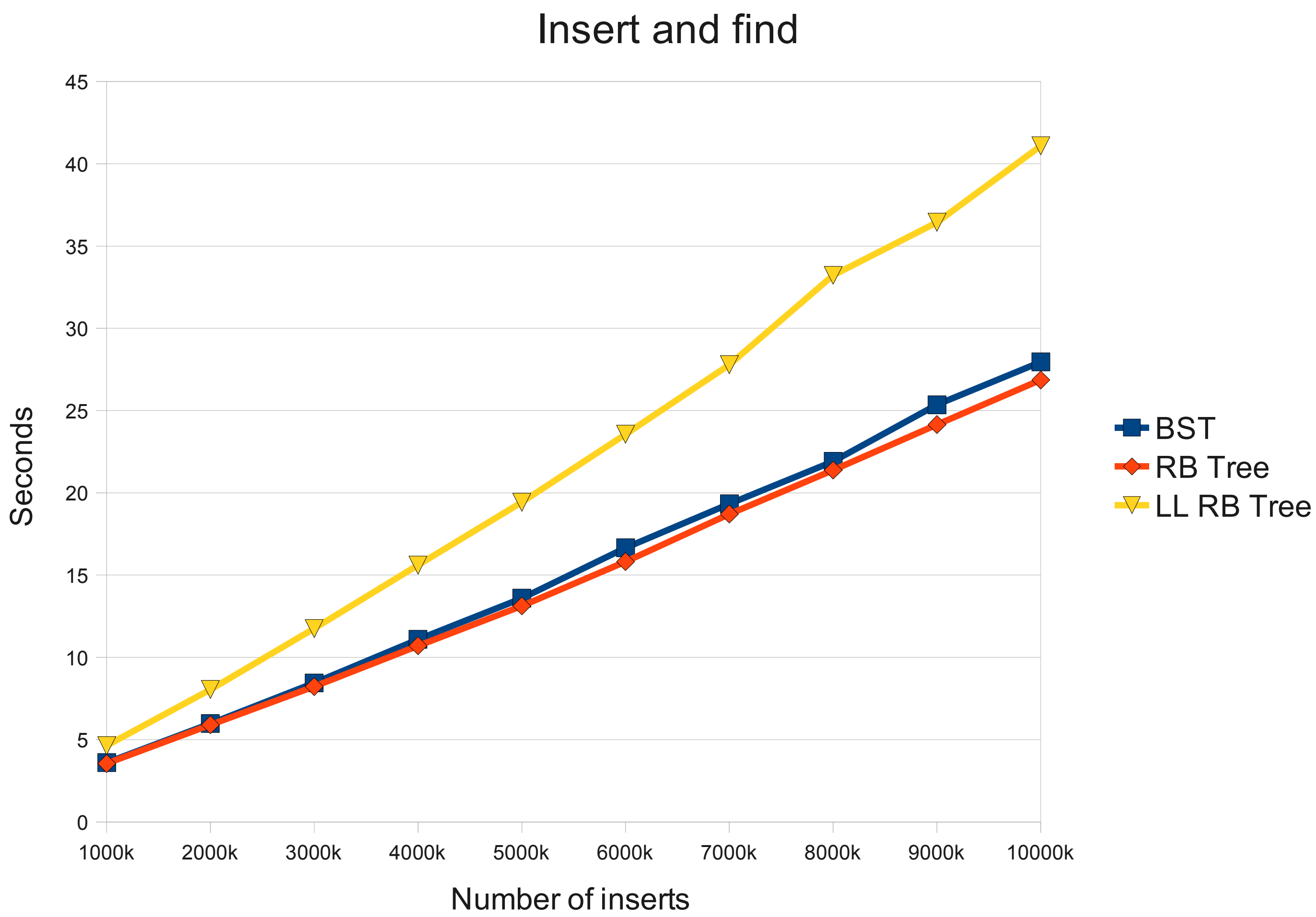
(3) LLRB implementation

- Left/right rotate: 9 lines each
- Color flip: 6 lines
- Insert, total 23 lines.
  - Sure is a lot smaller
  - Easier to understand

(4) Run tests

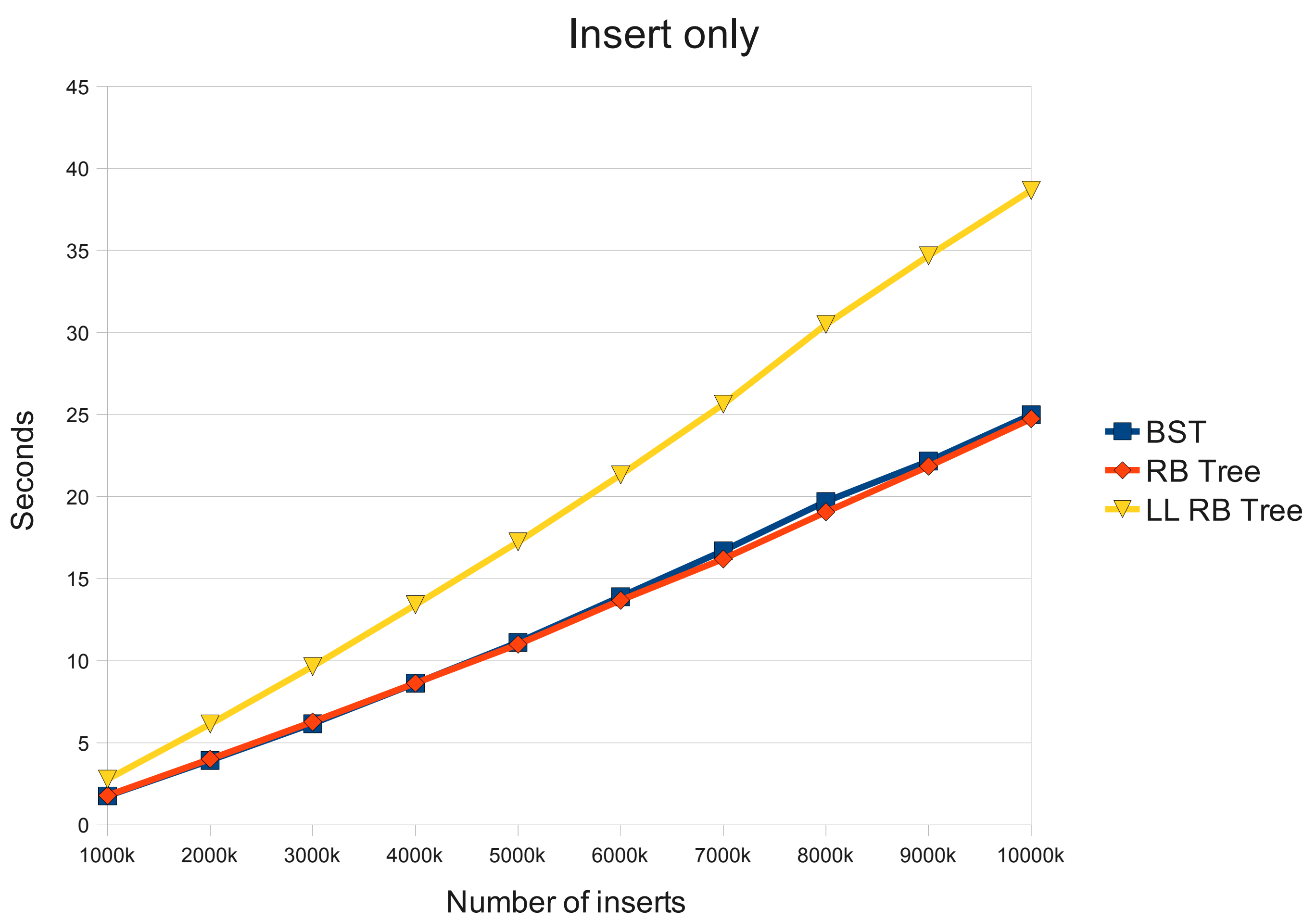
- 1-10 million inserts
  - To test O(lg n) insert claim
- 1 million finds
  - To test O(lg n) height claim.
- Is BST, RB or LLRB the fastest?
  - What do you think?
    - The classroom should be filled with anticipation by now.

(5) Results



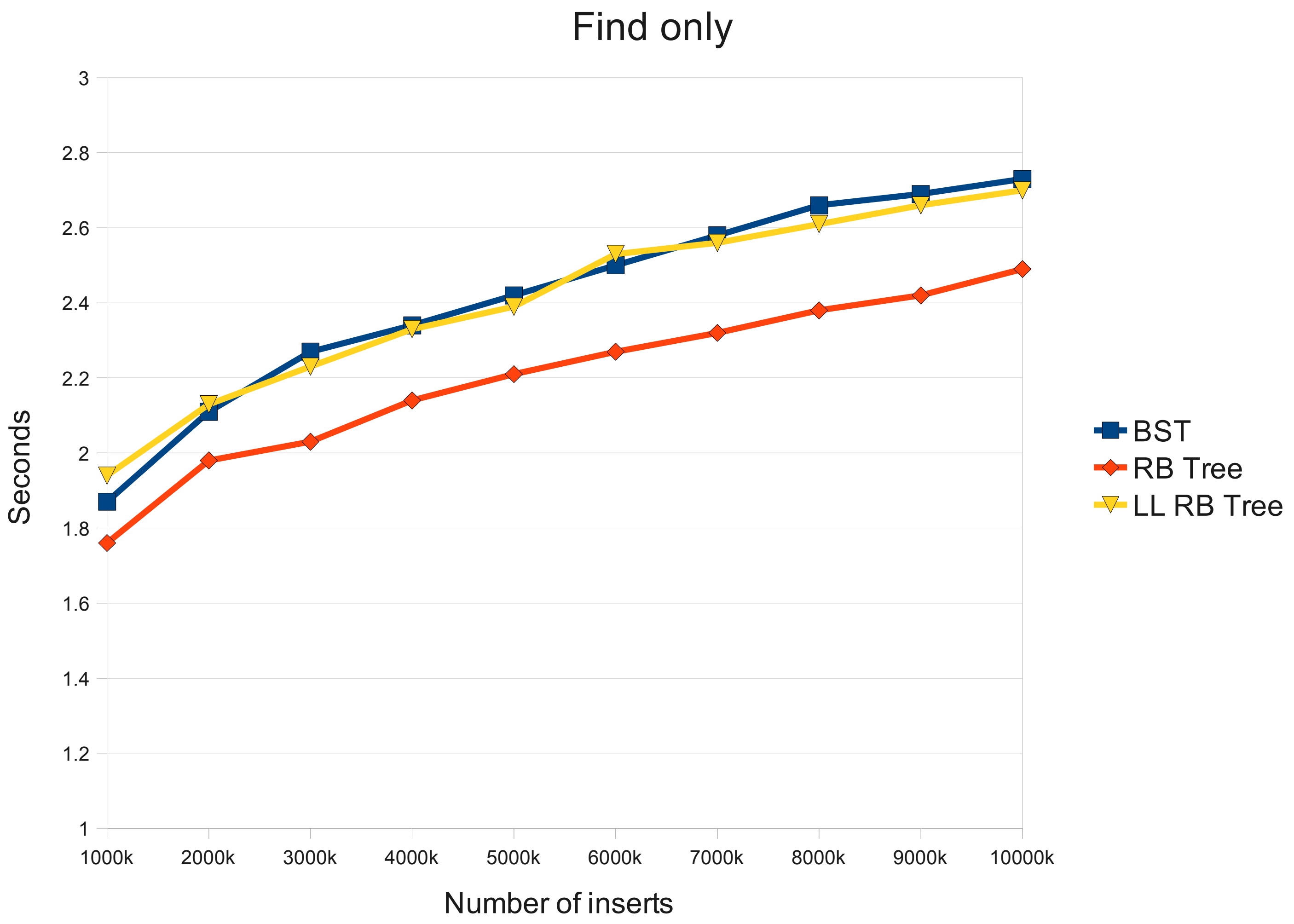
Insert and find
Seconds
45
40
35
30
25
20
15
10
5
0
1000k
2000k
3000k
4000k
5000k
6000k
7000k
8000k
9000k
10000k
Number of inserts
BST
RB Tree
LL RB Tree

(5) Results



Insert only
45
40
35
30
25
20
15
10
5
0
Seconds
1000k
2000k
3000k
4000k
5000k
6000k
7000k
8000k
9000k
10000k
Number of inserts
BST
RB Tree
LL RB Tree

(5) Results


Find only
Seconds
3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
1000k
2000k
3000k
4000k
5000k
6000k
7000k
8000k
9000k
10000k
Number of inserts
BST
RB Tree
LL RB Tree

(6) Conclusion

- Normal BST is not so bad!
  - Random insertions are good.
  - Implementation is simple.
  - In worst case, 1 mil. Insert, 1. mil. Finds
    - Very bad!
      - I stopped the process after 22 minutes.
- RB is the best all around.

(6) Conclusion

- Where is the promised land?
- Why is the LLRB not the best?
- Several possible explanations:
  - Recursive
    - Programs like iteration.
  - Bad implementation by me?
    - I hope not.
  - Not enough testcases

Assignment 3 – part 1

- Implement a left-leaning red-black tree.
  - Support insert
  - Do not bother about deletion
  - Support find
    - You should already have this from A2.
- Use *any language* you like
  - Except Java!
- Testcase generator from A2 works fine for testing

Assignment 3 – part 2

- Try to finish part 1 by Thursday of week 6
  - But don't turn it in yet
  - The final due date is February 18, 2010
- Part 2 is not finalized.
  - Coming soon

Thank you

Questions?