Homework 7 due 10 pm Friday, March 30th

Program 4

Last Week
Finish Recursion, recursion practice, General Trees and Binary Search Tree examples, Tree Traversals, and Categorizing ADTs Part 2

This Week: Read: Red-Black Trees
Binary Search Tree (BST)
• BSTnode class
• BST class
• implementing print
• implementing lookup, insert, delete
• complexities of BST methods

CS Options/Courses

Classifying Binary Trees
Balanced Search Trees
Red-Black Trees
• tree properties
• print, lookup
• insert

Next Week
Exam 2: Tuesday, April 4th, 5-7pm
For the location of the building, click on the building name below or check out the campus map.

Lecture 1: Room 272 of Bascom Hall
Lecture 2: Room B10 of Ingraham Hall
Lecture 3: Room 2241 Chamberlin Hall

Exam 2 Information and Sample Questions (available on Canvas soon)
Solution to sample questions will be covered in your lecture M-4/3 or T-4/4

Bring: your UW Student ID, #2 pencils, and eraser (just in case)

Read: start Graphs
Finish Red-Black Trees
Start Graphs
BSTs and BSTnodes

BST – Binary Search Tree
- is a key-value oriented collection of items where duplicate keys are not allowed
- **goal**: combine speed of binary search for access in an array $O(\_\_\_\_)$ with speed of linking/unlinking in a chain of nodes $O(\_\_\_\_)$
- **shape constraint**: binary tree structure
- **order constraint**:

  - in lecture, we’ll explore BSTs with items having only a key value
  - see readings for items having a key value and an associated value

BSTnode Class

```java
class BSTnode<K> {
    private K key;
    private BSTnode<K> left, right;

    public BSTnode(K key, BSTnode<K> left, BSTnode<K> right) {
        this.key = key;
        this.left = left;
        this.right = right;
    }

    public K getKey() { return key; }
    public BSTnode<K> getLeft() { return left; }
    public BSTnode<K> getRight() { return right; }

    public void setKey(K newK) { key = newK; }
    public void setLeft(BSTnode<K> newL) { left = newL; }
    public void setRight(BSTnode<K> newR) { right = newR; }
}
```

→ Draw a picture of the memory layout of a BSTnode.
import java.io.*;  //for PrintStream

public class BST<K extends Comparable<K>> {

    private BSTnode<K> root;

    public BST() { root = null; }

    public void insert(K key)
        throws DuplicateException {

    }

    public void delete(K key) {

    }

    public boolean lookup(K key) {

    }

    public void print(PrintStream p) {

    }

    //add helpers ...

}
Implementing **print**

→ **Write a recursive definition** to print a binary tree given \( n \), a reference to a `BSTnode`.

```java
public void print(PrintStream p) {
    print(root, p);
}

private void print(BSTnode<K> n, PrintStream p) {
```

→ **Complete the recursive print method based** on the recursive definition.

```java
public void print(PrintStream p) {
    print(root, p);
}

private void print(BSTnode<K> n, PrintStream p) {
```
Implementing lookup

Pseudo-Code Algorithm

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

![Binary Search Tree Diagram]

```java
}
```
Implementing \texttt{insert}

High-Level Algorithm

\begin{verbatim}
private BSTnode\langle K \rangle insert(BSTnode\langle K \rangle\ n, K key)
  throws DuplicateException {
\end{verbatim}
Practice - Inserting into a BST

- Insert 5, 27, 90, 73, 57 into the tree above.

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

- Will you get a bad shape only if values are inserted in sorted order?
Implementing delete

High-Level Algorithm

```java
private BSTnode<K> delete(BSTnode<K> n, K key) {
```
Practice - Deleting from a BST

→ Delete 90 from the tree above.
→ Delete 40 and 65 from the tree above.

→ Delete 10 and 70 from the tree above and redraw the tree.

→ How do you delete 50 or 30 from the tree above?
Implementing delete (cont.)
Practice - Deleting from a BST

→ Delete 30 from the tree above using the _________________.

→ Delete 50 from the tree above using the _________________.

Complexities of BST Methods

Problem size: $N = \quad$

print:

lookup:

insert:

delete:
CS Options

CS Certificate

5 CS Courses (12 credits minimum)
- Data Structures – CS 367 (& possibly prereq 302)
- 2 CS Courses >=400 level
- 2 Other CS Courses

CS Major (Declare as soon as able – requirements are changing)

Basic CS
- Discrete Math – CS 240
- Programming + Data Structures – CS 302, CS 367
- Basic Systems – CS 252, CS 354
Math
- Calculus – MA 221, MA 222
- 2 Beyond Calc – STATS 324 (intro applied stats), MA 340 (linear algebra)
Group A Theory
- Algorithms – CS 577
Group B Hardware/Software
- OS – CS 537
Group C Applications
- AI – CS 540
Group D Electives
- 2 CS Courses >=400 level

CS Double Major

- Must complete major requirements
- Easy for Computer Engineering Majors
CS Courses

Take Next

- CS 240 Introduction to Discrete Mathematics
- CS/ECE 252 Introduction to Computer Engineering (prereq for CS 354)
- CS/ECE 354 Machine Organization and Basic Systems (prereq for many group B)
- (CS 368) Learning a New Programming Language (C++ for CS 537)

- NOTE: CS/ECE 352 Digital Systems Fundamentals no longer required (is pre-req for CS 552)

>= 400 can take after CS 367

- CS 407 Foundations of Mobile Systems (spring, popular)
- CS 534 Computational Photography
- CS 539 Introduction to Artificial Neural Networks and Fuzzy Systems
- CS 540 Introduction to Artificial Intelligence
- CS 570 Human Computer Interaction (spring)

>= 400 can take after CS 367 + CS354

- CS 536 Introduction to Compilers
- CS 537 Introduction to Operating Systems
- CS 564 Database Management Systems: Design and Implementation
- CS 552 Introduction to Computer Architecture – CS 352, 354, and 367

>= 400 can take after CS 367 + Math

- CS 412 Introduction to Numerical Methods – MA 222 + MA 234 or CS 240
- CS 435 Introduction to Cryptography – MA 320 or MA 340
- CS 475 Introduction to Combinatorics – MA 320, 340, 341, or 375
- CS 514 Numerical Analysis = CS240, CS367 & MA 340
- CS 520 Introduction to Theory of Computing – CS240 & 367
- CS 524 Introduction to Optimization – CS302 and MA 320 or 340
- CS 525 Linear Programming Methods – MA 320 or MA 340 or MA 443
- CS 533 Image Processing – MA 320 or MA 340 (fall)
- CS 559 Computer Graphics – MA 320 or MA 340
- CS 576 Introduction to Bioinformatics – MA 222 (fall)
- CS 577 Introduction to Algorithms – CS 240
Classifying Binary Trees

Full

Complete

Height-balanced

Balanced
Practice - Classifying Binary Trees

Identify which trees below are full, complete and/or height balanced.

A

B

C

D

E

F
Balanced Search Trees

Goal:

Idea:

AVL

BTrees
Red-Black Trees (RBT)

RBT:

Example:

Red-Black Tree Properties

- root property
- red property
- black property

Red-Black Tree Operations

- print
- lookup
- insert
- delete
Inserting into a Red-Black Tree

Goal: insert key value $K$ into red-black tree $T$
and ________________________________.

If $T$ is Empty

If $T$ is Non-Empty

- step down tree as done for BST
- add a leaf node containing $K$ as done for BST, and ____________________

⇒ Which of the properties might be violated as a result of inserting a red leaf node?

- root property
- black property
- red property

Non-Empty Case 1: $K$'s parent $P$ is black
Non-Empty Case 2

Non-Empty Case 2: K's parent P is red

Fixing an RBT

Tri-Node Restructuring is done if P's sibling S is null

Recoloring is done if P's sibling S is red
Practice

1. Starting with an empty RBT, show the RBT that results from inserting 7 and 14.

2. Redraw the tree from above and then show the result from inserting 18.

3. Redraw the tree from above and then show the result from inserting 23.

4. Redraw the tree from above and then show the result from inserting 1 and 11.

5. Redraw the tree from above and then show the result from inserting 20.
More Practice!

6. Redraw the tree from the previous page and then show the result from inserting 29.

7. Insert the same list of values into an empty BST: 7, 14, 18, 23, 1, 11, 20, 29

What does this demonstrate about the differences between a BST and RBT?
More Practice?

8. Show the result from inserting 25 in the RBT below.

9. Redraw the tree from above and then show the result from inserting 27.
Cascading Fixes

Fixing an RBT UPDATED!

Recoloring is done if P's sibling S is red

1. change P & S to black
2. if G is the root – done
   otherwise change G to red

Tri-Node Restructuring is done if P's sibling S null _______________
RBT Complexity

print

lookup

insert