CSCI 136: Data Structures and Advanced Programming
Lecture 25
Trees, part 3
Instructor: Dan Barowy
Williams

Topics
- Binary tree traversals
- Binary search trees

Your to-dos
1. Read before Mon: Bailey, Ch. 15-15.3.
2. Lab 8 (solo lab), due Tuesday 4/26 by 10pm.

Announcements
CS Colloquium
Friday, April 22
2:35pm in Wege
Crowdsourcing with Video Games
Seth Cooper (Northeastern University)

People and computers working together can solve problems neither could solve alone. To support this, video games provide a compelling approach: they are a natural space for problem solving and can foster the engagement necessary for people to make a contribution. Games have been used in this way to crowdsource approaches to protein design, software verification, and activity recognition. In this talk I will discuss a variety of approaches to making games that crowdsource solutions to problems, using image classification as a common application.
Announcements

Spring pre-registration begins Wed, April 27 and runs until Fri, May 6.

The best way to get into the CS course you want is to pre-register.

Common “next steps” after CSCI 136:
CSCI 237: Computer Organization
CSCI 256: Algorithms
CSCI 334: Principles of Programming Languages
also, some electives.

FoldIt!

Quiz

Binary tree traversals
Suppose you are asked to write an `Iterator<T>` for a binary tree. What order do you choose?

Remember that tree nodes store data (`T`). A **traversal** corresponds with the order that that data is returned.

**Pre-order traversal:** Return data from each node **before its children**, and then return child data from **left to right**.

Returns the sequence: `a, b, d, e, c, f, g`

**In-order traversal:** Return data from each node **after its left child** and **before its right child**.

Returns the sequence: `d, b, e, a, f, c, g`

**Post-order traversal:** Return data from each node **after its children**; return child data from **left to right**.

Returns the sequence: `d, e, b, f, g, c, a`
Binary tree traversals

**Level-order traversal** (aka *breadth-first order*): Return data from each node in *level* \(i\) before data in *level* \(i+1\).

```
      1
     / \
    2   3
   / \ / \  
  4  a b 6 f  
   \ / \  /  
    5 d e 7 g
```

Returns the sequence: \(a, b, c, d, e, f, g\)

What traversal should I use?

Suppose I encode the arithmetic expression \(1 - 2^4 \times 2\) using the following tree and want to “evaluate” it.

```
       -
      /  
    1   ×
    / 
   2 ^
  /   
 4   2
```

Binary search tree

A **binary search tree** is a binary tree that maintains the *binary search property* as elements are added or removed. In other words, the *key* in each node:

- must be \(\geq\) any *key* stored in the left subtree, and
- must be \(\leq\) any *key* stored in the right subtree.

As with other ordered structures, order is maintained *on insertion*. 

Ordered Trees
Key, Value nodes

Note that I said key instead of element.

Storing a key and a value in each node allows the greatest flexibility when arranging a tree. I.e., the key type $K$ need not be the value type $V$.

Restriction: keys must be comparable in some way (e.g., \texttt{Comparable\textless K\textgreater} or \texttt{Comparator\textless K\textgreater}).

Example

Insert the following elements: $71, 20, 27, 17, 91, 14, 87$
Assume $K$ and $V$ are the same.
Example
Insert the following elements: 71, 20, 27, 17, 91, 14, 87
Assume K and V are the same.

Example
Insert the following elements: 71, 20, 27, 17, 91, 14, 87
Assume K and V are the same.

Example
Insert the following elements: 71, 20, 27, 17, 91, 14, 87
Assume K and V are the same.

Example
Insert the following elements: 71, 20, 27, 17, 91, 14, 87
Assume K and V are the same.
Example

Insert the following elements: 71, 20, 27, 17, 91, 14, 87
Assume K and V are the same.

Binary Search Tree

Let’s implement this together.

Recap & Next Class

Today:
- Binary tree traversals
- Binary search trees

Next class:
- Tree balance
- Asymptotic analysis