CSCI 2270

Data Structures & Algorithms

Gabe Johnson

Lecture 5 Jan 25, 2013

Linked Lists (postmortem)
Intro to Binary Search Trees

Lecture Goals

1. Announcements
2. Linked Lists: Love or Hate?
3. Learn to learn
4. Intro to Binary Trees

Upcoming Homework Assignment

HW #2 **Due: Friday, Feb 1**

Binary Search Trees

This assignment will be a bit easier, to give people time to recover from the last round.

Binary Search Trees are binary trees that are used to store and find information in some predefined order.

Will be released over the weekend.

RetroGrade Maintenance

The grading system will be down periodically this weekend and possibly on Monday as I fix the (numerous) issues we found this week.

Linked Lists

Questions?

Bug reports?

Learn how to learn

When coding, ask yourself what *everything* means.

```
#import <iostream>
#import “my_functions.h”

using namespace std;

int main() {
   cout << “Hello World” << endl;
   return 0;
}
```

Learn how to learn

When coding, ask yourself *what-if* scenarios.

```
#import <iostream>
#import “my_functions.h”

using namespace std;

int main() {
   cout << “Hello World” << endl;
   return 0;
}
```

Learn how to learn

When coding, don't let lots of new changes stock up before trying things out. Baby steps FTW.

Write some code, run it, look at the output, think *critically* and ask "Is this doing what I think it should be doing?"

... and if not, *fix it right away*.

Learn how to learn

When coding, **the worst thing you can do** is to get stuck, be frustrated, but keep trying the same strategy that obviously is not working.

Be dynamic. Be creative. Develop strategies for

asking creative questions

and *teasing out answers to those questions.*

Learn how to learn

Put lots of print statements in your code. If you think the function is obviously returning 42 but something is going wrong, write a print function to convince yourself it really is. You'd be amazed at how often it returns 41. Or 43.

This can be messy, so do house-keeping once in a while, otherwise you won't be able to read the important stuff.

Debugging is Awesome

We will have a full-on lecture on the art of debugging in a few weeks.

For now:

edit the driver to stress test your code.

Turn your creative ‘what if’ questions into code that tests what your program is made of.

Debugging is Awesome

Example: you are working on insert(..).

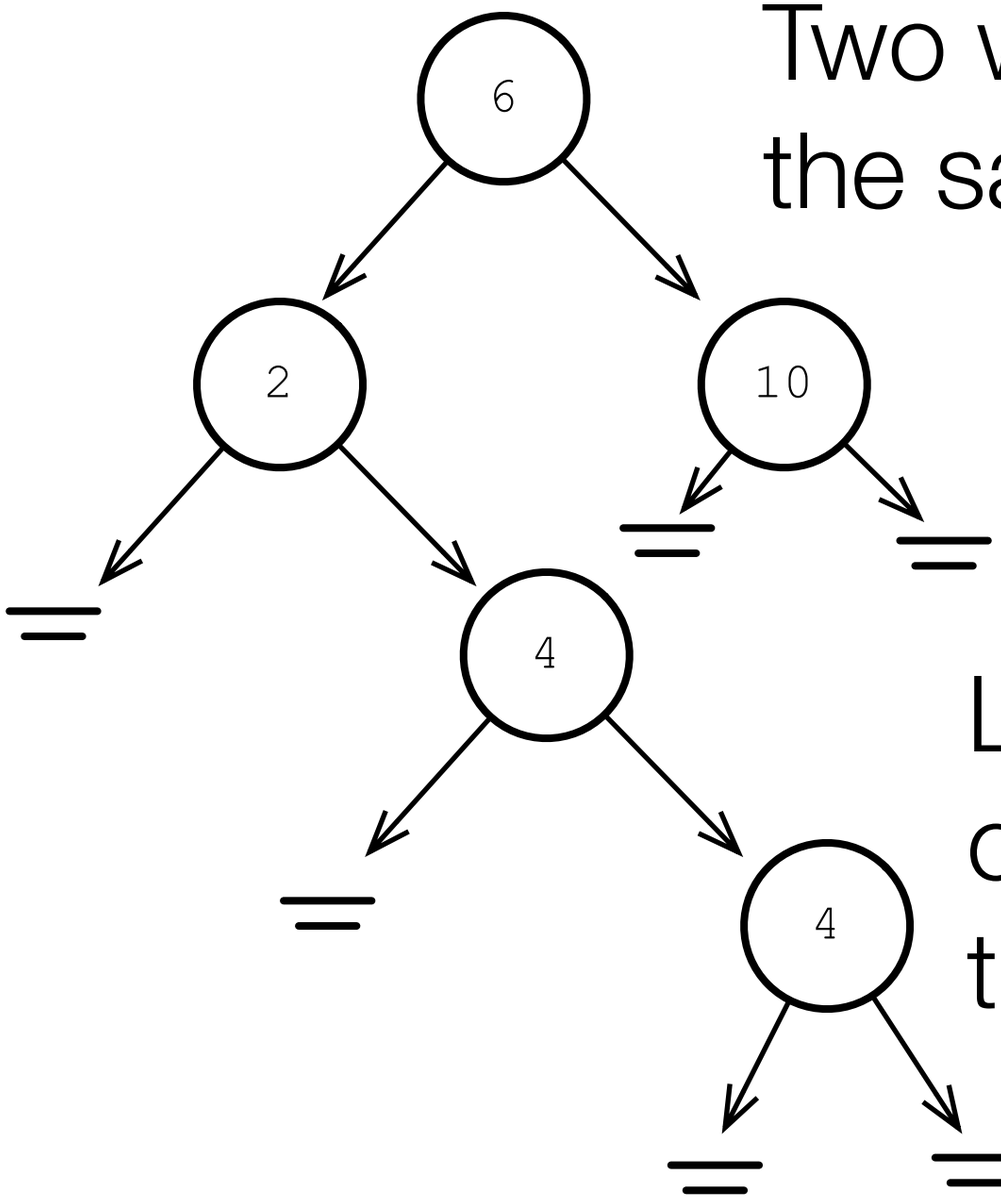
Look at linked_list_driver.cpp and see the sample code that tests append(..). Adapt it to fit your new code.

Take a look at the grading scripts in
github.com/johnsogg/grading-scripts
to see how your homework is graded.

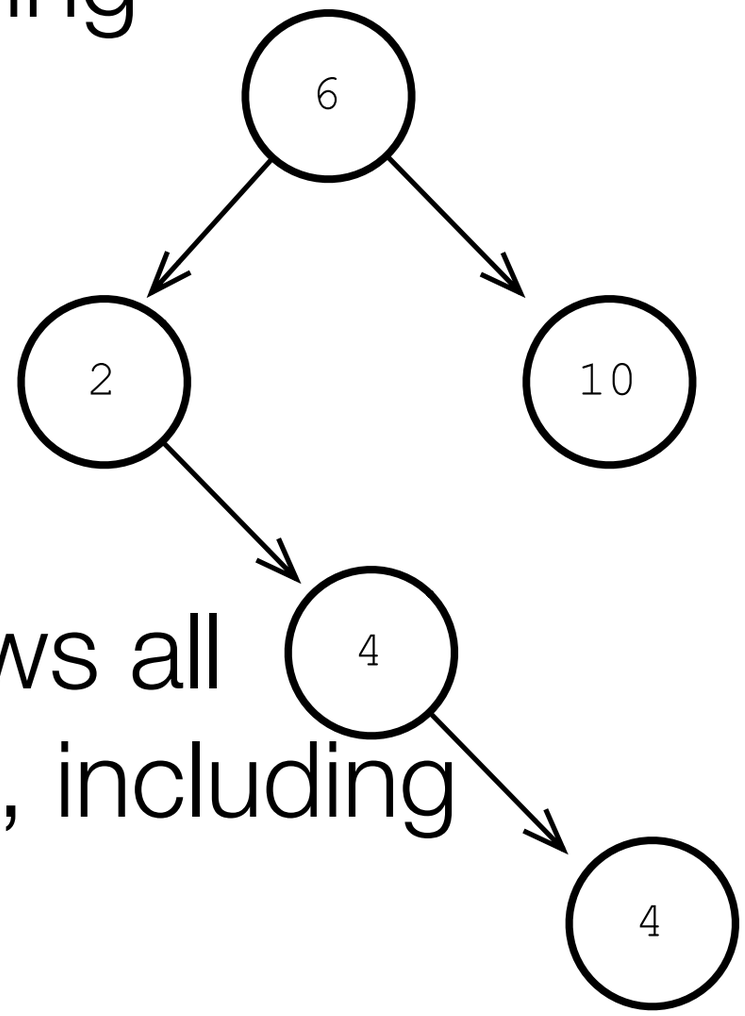
Learn. Adapt.

```
TEST(LinkedListTest, Insert) {
  node* top = build_three_node_list(7, 98, -34);

  // add at beginning
  node* beginning = init_node(5);
  insert(&top, 0, beginning);
  cout << "FYI: should be 5, 7, 98, -34: " << report(top) << endl;
  int vals[] = {5, 7, 98, -34};
  EXPECT_TRUE(expect_all(vals, 4, &top));

  // add in middle
  node* middle = init_node(20);
  insert(&top, 2, middle);
  cout << "FYI: should be 5, 7, 20, 98, -34: " << report(top) << endl;
  int vals2[] = { 5, 7, 20, 98, -34 };
  EXPECT_TRUE(expect_all(vals2, 5, &top));

  // add at end
  node* ending = init_node(800);
  insert(&top, 5, ending);
  cout << "FYI: should be 5, 7, 20, 98, -34, 800: " << report(top) << endl;
  int vals3[] = { 5, 7, 20, 98, -34, 800 };
  EXPECT_TRUE(expect_all(vals3, 6, &top));
}
```

Binary Search Trees!



Two ways of drawing the same tree.



Left side shows all child pointers, including the null ones.

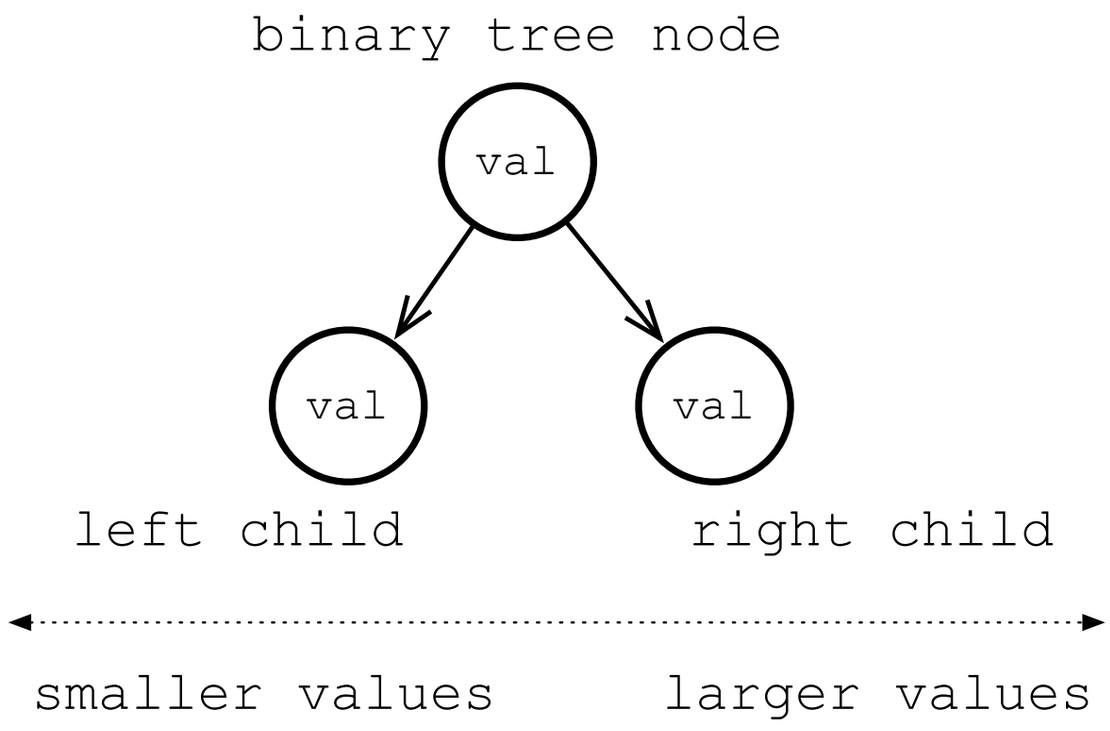
Binary (Search) Tree

A plain *Binary Tree* simply holds data in a tree where each node points to two children.

A *Binary* ***Search*** *Tree* is a class of Binary Tree where the data are stored in a specific order. This makes it convenient to use them for finding data quickly.

Binary Tree Node

A *Binary Tree* is a data structure that lets us store data in a tree structure that is composed of nodes. While Linked List nodes have a single 'next' field that points to another node, Binary Tree nodes have *two* references to child nodes.



Binary Tree Operations

- Insert: put data into the tree
- Search: find data in the tree
- Delete: remove nodes from the tree
- Sort: apply a sort order to data in the tree
- Traversal: ‘walk’ the tree to ‘visit’ nodes