Trees 2: Application Example, Binary trees, Binary Search Trees, Trees, tR3es, tree$

CptS 223 - Fall 2017 - Aaron Crandall

Today’s Agenda

- Announcements
- Trees:
  - Application example
  - Binary trees
  - Binary search trees:
    - Notably analysis and examining uses
- Monday: starting AVL trees - a form of self balancing BST
- Also doing more tool use for development on Linux

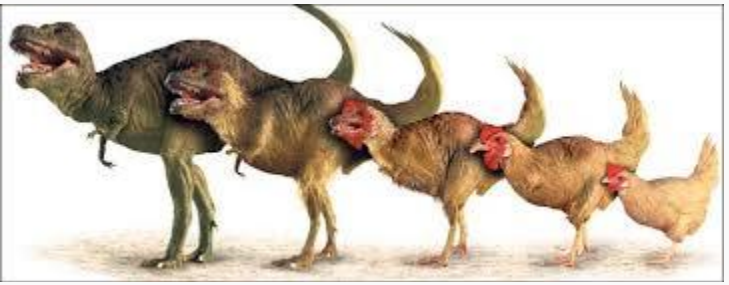
Announcements

- Midterm #1 will be October 2nd
  - Topics: Algorithm analysis, Big-O, Linux commands, Trees: BST, AVL, RedBlack, B+
- NEW SERVERS!
  - Centos 7 (instead of 6)
  - Git works out of the box!
  - g++ can do C++11 out of the box!
  - Hostnames: sig1.eecs.wsu.edu, sig2.eecs.wsu.edu, sig3.eecs.wsu.edu
  - Named over the UNIX process signals, FYI

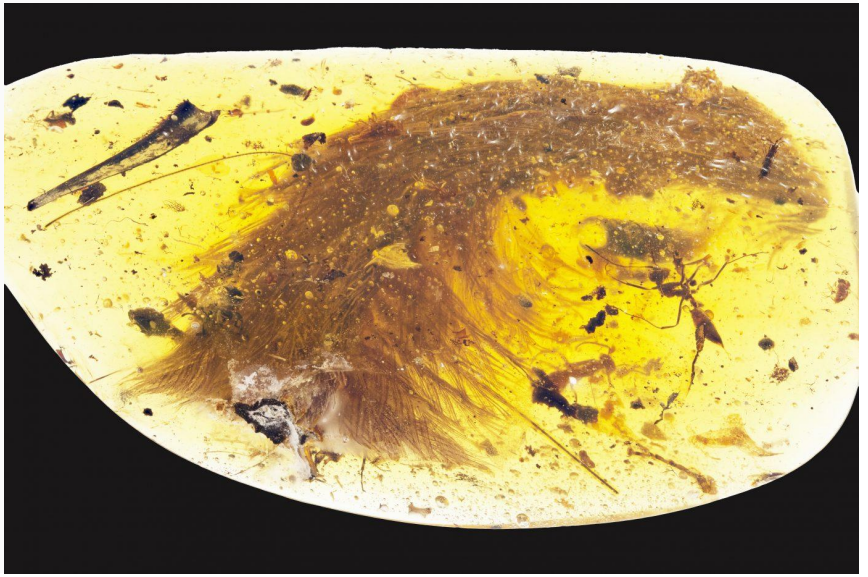
Thing of the day: Dinosaurs had feathers

"Feathered dinosaur tail captured in amber found in Myanmar"

Sadly, our scaly giant lizards are likely feathered.
Basically, the connection between dinosaurs and birds…
Yeah, birds are dinosaurs



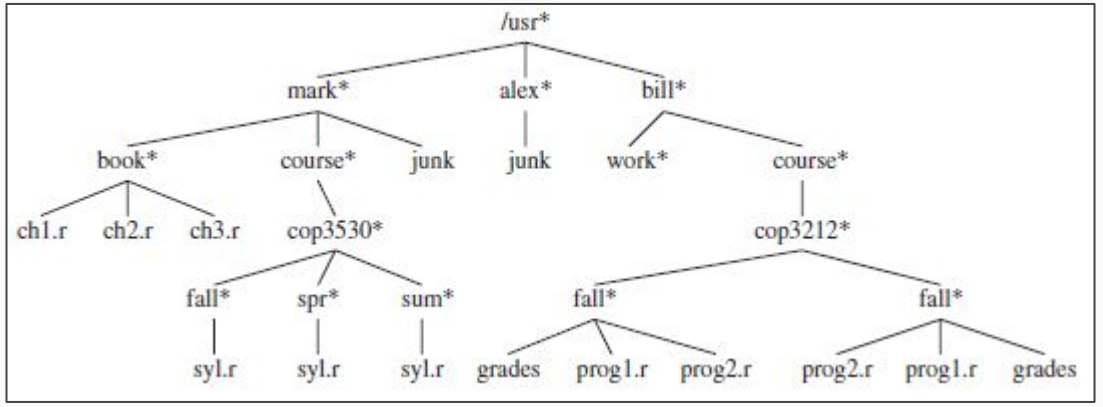
Sad, but true. I've seen the fossils in the British Museum of Natural history



Example tree - and a question!

How can nodes be one of *two* things and still be in the same tree?

Directory OR File?



What are they inserting to the tree by? Also, the fully qualified path is kind of the true identity of the file, not just the filename.

<Show tree command here>

BST features

- Functions: Complexity:
  - contains(Comparable & x)
  - findMin( )
  - findMax( )
  - insert(Comparable & x) vs. insert(Comparable && x) --- See: std::move(x) for rvalues
  - remove(Comparable & x)
  - makeEmpty()
  - copy()

Explaining features

- contains(Comparable & x) - return true if item exists
- findMin( ) - returns pointer to minimum item
- findMax( ) - returns pointer to maximum item
- insert(Comparable & x) - inserts new item
- remove(Comparable & x) - removes a given item
- makeEmpty() - Deletes the tree
- copy() - copies the tree, returns pointer to new tree

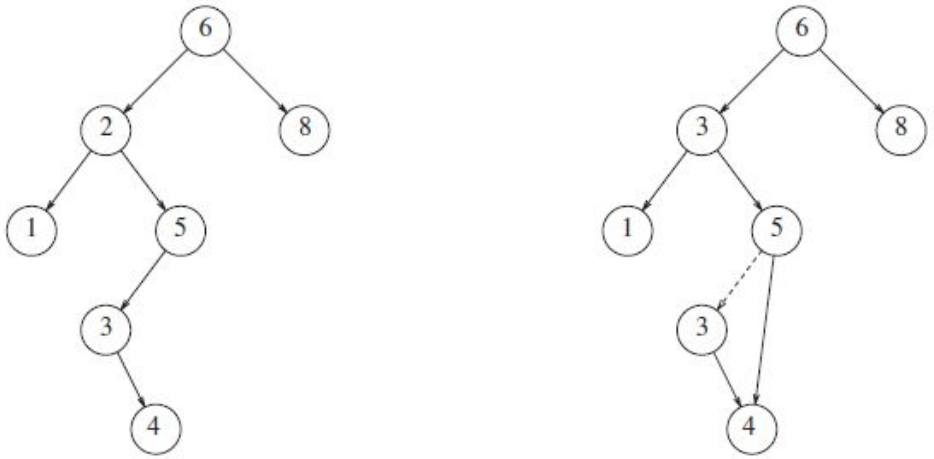
Some details about their insert(), delete(), contains() implementations

- Look at the interfaces they provide:
  - [$function name](const Comparator & x)
  - Does this require the caller to have knowledge of BinaryNode?
  - Would it be easier to have insert(BinaryNode* node)?
- What does this achieve?
  - Reduced coupling! Greater cohesion.
  - Prevents others from “seeing” your implementation, or needing to know of it
- Add(T val) vs. Add(BinaryNode<T> * val)
  - Which one would you rather be faced with when using someone’s library?

More details about delete(x)

- Was queried on why it always pulls from the right (in my example)
  - It's because a simple BST always deletes by pulling from one subtree or another
- Delete cases:
  - If x not in tree, do nothing
  - If x has one child, replace x with child node ala a linked list delete
  - If x has two children:
    - Find rightmost of left child or leftmost of right child
    - Copy min/max of subtree to node to be deleted
    - Recursively delete(new min/max in subtree) [see code to be clear]

Demoing delete(x)



Deletion of a node (2) with two children, before and after

```
/**
 * Internal method to remove from a subtree.
 * x is the item to remove.
 * t is the node that roots the subtree.
 * Set the new root of the subtree.
 */
void remove( const Comparable & x, BinaryNode * & t )
{
    if( t == nullptr )
        return;   // Item not found; do nothing
    if( x < t->element )
        remove( x, t->left );
    else if( t->element < x )
        remove( x, t->right );
    else if( t->left != nullptr && t->right != nullptr ) // Two children
    {
        t->element = findMin( t->right )->element;
        remove( t->element, t->right );
    }
    else
    {
        BinaryNode *oldNode = t;
        t = ( t->left != nullptr ) ? t->left : t->right;
        delete oldNode;
    }
}
```

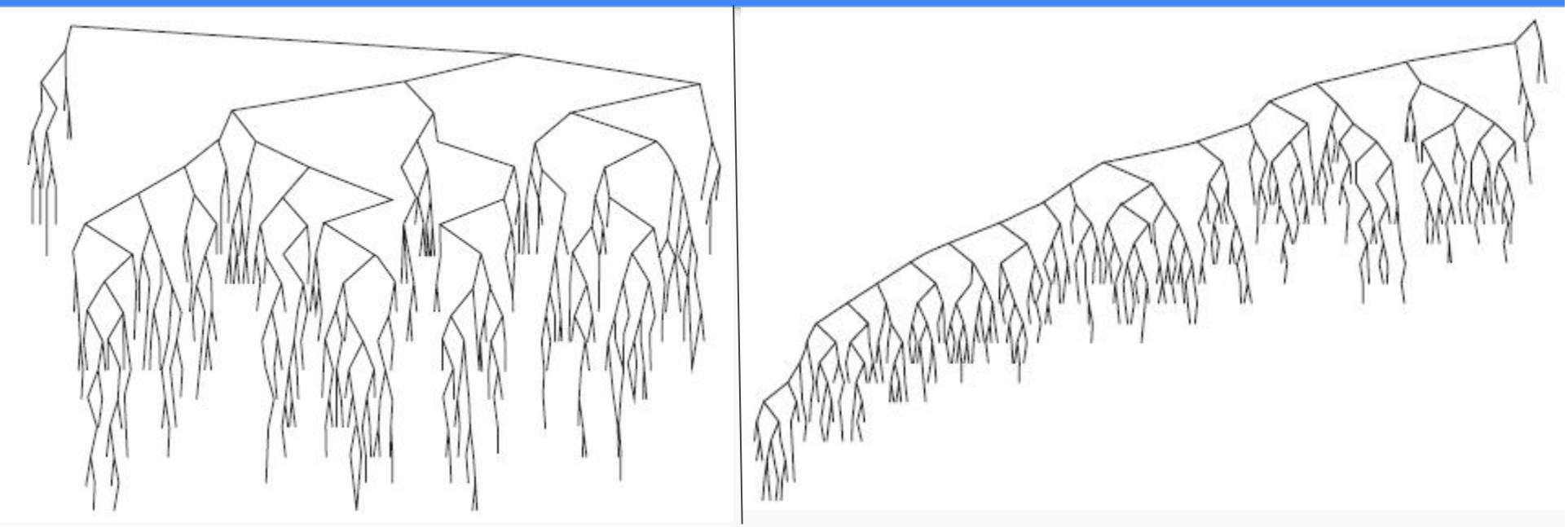
Is delete(x) actually O(log(N)) time?

- How many operations does it take?
- How much of the tree does it take to traverse?
- Is there a cost to copy, delete/free, that we need to consider?
- Is this changed by using lazy deletion?
  - Also: What IS lazy deletion?
  - Does lazy deletion effect tree height much?

But what's the impact?

- The authors hinted and discussed about BST ending up at O(√N) height
  - All due to delete behavior over time
- Imagine randomly inserting and deleting nodes
  - What's the effect of doing the deletes from the right subtree?
  - Apparently the effect happens after Θ(N^2) alternating inserts and deletes

That imbalance effect is serious!



Though, we don’t know *why* (in a formal sense) this happens.

But there’s solutions and other weird behaviors

- Solution: alternate largest left subchild with smallest right subchild when deleting. This fixes things nicely, but only in a general sense, and doesn’t solve bad input series like {1,2,3,4,5}. There’s more generally capable approaches to keeping trees balanced.
- Weird stuff: The imbalance appears when random inserts/deletes is done > Θ(N^2) times. But… if we only do o(N^2) (see little-O there), the trees actually *gain* balance.

Behavior of the algorithm over time

- There is a long term behavior of your data structure over time.
- As you use the structure, there might be strange issues under some circumstances, such as this imbalancing behavior
- Understanding and analyzing the large scale issues is important
  - This is why we pay the penalty of AVL, Red-Black, and B-Trees to achieve guaranteed balance

BST features to sum up

Functions:	Ave Complexity:
contains(Comparable & x)	O(log N)
findMin( )	O(log N)
findMax( )	O(log N)
insert(Comparable & x)	O(log N)
remove(Comparable & x)	O(log N)
makeEmpty()	O(N)
copy()	O(N)

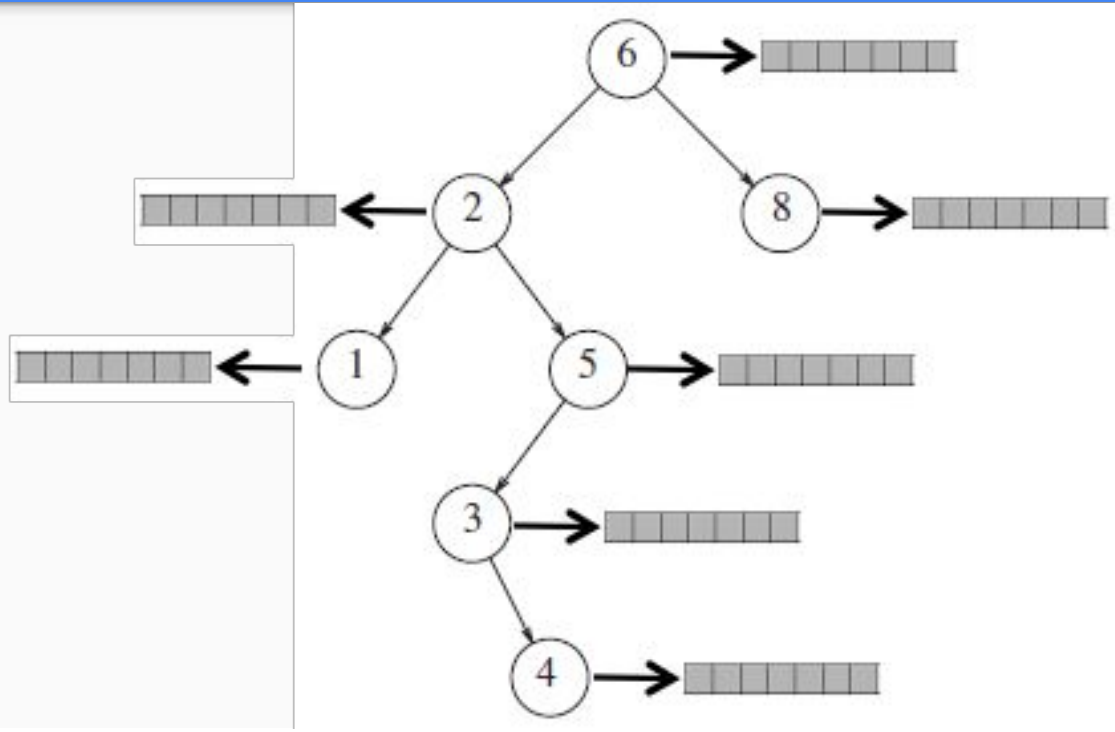
Hinted BST tree use - Priority queues

- Remember having to enqueue new elements in a priority queue having (N/2) kinds of behaviors?
- The BST is here to rescue you with (log(N)) behavior!
- Use a BST with the nodes pointing to linked lists of all elements sharing the same priority
- Dequeue returns the head of the leftmost queue
  - Deletes tree node if empty if space is an issue!
- Enqueue either creates a new tree node or appends to the list at the queue

BST priority queue nodes

```
struct BinaryNode {
        Comparable priorityScore;
        BinaryNode *left;
        BinaryNode *right;
        queue<workUnit*> workUnits;
}
```

- Time cost effects?
  - O(log(N)) + O(?)



Linux command of the day: pstree

- pstree - show a tree of processes and their children
- man pstree
- What do you think “init” is?

Programming tools!

- Much of this would be covered in CptS 224, but I'll do some to get you started.
- Plenty of debugging and visualization tools available
- Today: gdb & ddd (GUI for gdb)

How do I debug my program?

- Primary tool on the terminal is GDB
- A good GUI option (widely available) is DDD
  - DDD uses GDB and gives you a GUI output
- Install on Debian (ubuntu)
  - sudo apt-get install gdb ddd
- Install on CentOS (RedHat)
  - sudo yum install gdb ddd
- Install on Arch Linux
  - sudo pacman -S ddd

Building with debugging symbols

- Use the -g option (command line switch) with g++

```
g++ -g -Wall -std=c++0x -o myProg *.cpp *.h
```

- Will include names of functions, variables, and such for debugger to show you while you step through code

Invoking gdb

- On the command line, gdb is your friend

  gdb myProg

- Or with a “gui”

  gdb -tui myProg

  * Look up the -tui commands. It uses Emacs style escapes to change windows within the tool. Ctrl-x o is the big one

GDB Commands

- run Start program
- ctrl-c Pause running program
- list Current source
- break Set breakpoint (function, linenum, filename:line, many others!)
- next Go to next, staying out of functions
- step Step into functions
- until Go until next highest line in source
- continue Run until break or terminate
- print $var Print variable value

GDB commands

- backtrace Show series of function calls to get to current place
- info frame show stack frame memory
- info locals show local variable values
- break line# Set breakpoint at line#
- break file:line# Set breakpoint at source file & line #
- break func1 Set break at func1
- break TestClass::testfunc(int) set breakpoint at class function
- info breakpoints List the breakpoints
- disable break# ignore break#

DDD

Invoking

ddd myProg

GUI time!

Can be used via NX on the EECS SSH servers, or on your own Linux box

Also works if you forward your X session from the EECS SSH servers, including our new ones at sig1.eecs.wsu.edu:

ssh -X [username]@sig2.eecs.wsu

This works “out of the box” on Linux, but Windows and OSX need to install an X server like Exceed (windows) or the OSX X server