CS280 Notes 2/17

by Christian Sagel

Rotating Nodes

Fundamental technique performed on BSTs.
Rotating Nodes: There’s left and right rotations.
We can say rotating or promoting a node.
Promoting a node is to replace its parent.
After a rotation the order must be preserved.
Balancing trees means manipulating pointers.

In a tree, root would be the head. Or it could be a subtree of a bigger tree.
We can see different representations of the tree even if they all contain the same data. It depends on the order the data was received.

Once we start balancing trees, the order doesn’t matter.
BTrees, like sorted linked lists, you can’t tell in what order did the data come in.
The predecessor: the right-most node in the left subtree will replace a node that was removed.

Getting sorted data on a BST is BAD! log2N!
On an AVL tree, it’s GOOD!

Splay Trees

If you know the distribution of your data.
Usually we access a subset of the data. We want to make that data faster to access than the other data
GIven that we are in a tree, every time we access an item in the tree we rotate it to get things up to the top of tree to make it to the root. Same thing for every time we do a find.

If we are constantly doing lookups in that list; after we find a node with the data we move that node to the top of the list by just swapping pointers. Very fast.
It becomes a working set, a small subset of data, the stuff that's hot goes to the front of your list or tree.
The implementation of this is trivial.

Moving the recently used item in the node is called splaying.
We want to splay a node two levels at a time. We promote the node to the position of its grandparent.

The algorithm depends on the orientation.
With BST we will end up with 4 different cases: Zig-zigs or Zig-zag in different orders.

Left-Left orientation (zig-zig)
Left-right orientation (zig-zag)
Right-left
Right-right

The special case: we want to splay a node C to the root.

If we are == parent->right, check for self
Helper function to find the orientation!
Splay (built-in cache mechanism)

Anytime we have unsorted lists that we need to transverse, makes sense to do this since the cost of splaying on a linked list is very cheap.
amortized: over the life of the data structure.

Expression Trees

Like binary trees, but not “sorted’ in the usual way.
Compilers fold code, constant folding. If the compiler finds literals, it’s gonna calculate that at compile time.
The key to knowing how an expression tree works is understanding that it uses postfix,
Languages have grammar, with production rules.
non-terminals on the left, terminals on the right
Internal nodes are operators, operands are external
Compilers are big on using unions.

An union is like a struct; in a struct every variable is in its own memory. In an union it's the size of the largest data byte. They share the memory. You can have only one of those active at the time, some kind of discriminator.
They are nice for saving a lot of space, mutually exclusive.
Unions are not standard between compilers.
Network programming: You want to avoid sending empty structs. Besides compressing, you want to save as much bytes as you can.
The difference between const and constexpr. Const can’t change, but the value is not known at compile time.
Constexpr, all values known at compile time.

On **Monday**, we will start balanced trees. Everything we have covered up until now will be on the next assignment.