Data Structures with C++ : CS189

Lecture 15-2: Loose Topics

Last Lecture

- For the last lecture I just have some rando accumulated topics
  - Some from book I couldn't fit in
  - Some I deleted during the Great Class Reorg of 2020.
- Test is Paper part on Tuesday and Program part on Thursday
  - Don't need any DSPS fixes because by splitting it up, everyone gets time and a half

Types of Trees

- We started with a BST, but bad input made it lopsided
- We added one kind of balancing and got good general performance
- We added heap structure (Treap) to get an unbalanced tree that weighted nodes by priority
- All of these are good ideas and you need to always consider your code for what to pick

Splay Tree

- Splay = Whatever node was last accessed becomes the root
  - Really fast if your code does something like accessing the same record twenty times in a row
    - Idea is sort of like a cache in that way
- Start with a Left and Right "hook"
- Every time you make a decision in the tree, break off the other side and hang it on its hook
- At the end, your target node (From PrivateFind) is the root and those two hooks become left and right

Dynamic Priority

- A Treap uses BST logic on the way down, but then uses heap logic to rotate back up
  - Same Heap rules
- Instead of shattering the entire tree in a splay, just increase the priority of the target cell
  - Nodes rise to their level more slowly
  - Both of these break balancing
  - One more use of PrivateFind

B-Trees

- Trees don't have to be binary
- Picture nodes that have four arrows out, and three internal data values
- This will naturally make the tree very wide and not as tall
  - Not a speed gain - you still compare extra mid-node values
- Now make data only allowed in leaves, and make leaves a set size, and boom. That's how databases work
  - Leaf size is OS Page size, 4KB last I checked
  - Can loop left-right through leaves since data is never in the middle of the tree

KD Trees

- You might need to store more complicated keys than a single int in your BST
- A KD tree has D sub-keys
- Every time you drop a level, you rotate through the keys
  - Students have Name and ID
  - At root, left or right based on Name
  - Next level, ID
  - Next level, Name
- Really good if you have complicated searches based on multiple keys, but breaks balancing

Huffman

- A txt file is a series of characters
- Each character in ASCII is one byte
- Having to use eight bits for Q is a waste
- Huffman is a way of compressing a file based on the frequency of the letters inside
  - Zip files take this further and identify arbitrary patterns
- Count the letter distribution in the file
- Make a tree using those distributions as weights
- More common letters are up top and have shorter FakeAscii codes

End

Not quite enough for a 189-2, but getting there