BALANCED SEARCH TREES

David Kauchak
CS 62 – Spring 2021

1

Binary Search Trees

BST – A binary tree where each node has a key, and every node’s key is:
- Larger than all keys in its left subtree. (everything left is smaller)
- Smaller than all keys in its right subtree. (everything right is larger)

Operations

Search – Does the key exist in the tree
Insert – Insert the key into tree
Delete – Delete the key from the tree

Admin

Last day for “normal” mentor hours, Friday (5/7)

More on mentor hours next week
Height of the tree

Most of the operations take time $O(\text{height})$

We said trees built from random data have height $O(\log n)$, which is asymptotically tight.

Two problems:
- We can’t always insure random data
- What happens when we delete nodes and insert others after building a tree?

Worst case height for binary search trees is $O(n)$.

Balanced trees

Make sure that the trees remain balanced!
- Red-black trees
- AVL trees
- 2-3 trees
- 2-3-4 trees
- B-trees
- ...

Height is guaranteed to be $O(\log n)$.

2-3 trees

2-node: one key and two children (left and right)
- everything in left is smaller than key
- everything right is greater than (or equal to) key

3-node: two keys ($k_1, k_2$) and three children, left, middle and right
- $k_1 < k_2$
- everything in left is less than $k_1$
- everything in middle is between $k_1$ and $k_2$ (greater than or equal to $k_1$ and less than $k_2$)
- everything in right is greater than (or equal to) $k_2$

Search

How do we search for a key?
Almost identical to BST search

Only difference: sometimes we have two keys

Anatomy of a 2-3 search tree

Which child?
Which child?

Not found!
Insertion

Like BST, insert always happens at a leaf

If the leaf is a 2-node, just insert it directly

Insertion

Insert(F)

If the leaf is a 2-node, just insert it directly

Insert(F)
Insertion
If the leaf is a 2-node, just insert it directly
Insert(F)

Like BST, insert always happens at a leaf
If the leaf is a 2-node, just insert it directly
If the leaf is a 3-node:
- We now have three values at this leaf
- Send the middle value up a node
- Make new 2-nodes out of the smallest and largest
Insertion

If the leaf is a 3-node:
- We now have three values at this leaf
- Send the middle value up a node
- Make new 2-nodes out of the smallest and largest

Insert(T)

Where should it go?

25

Insertion

Insertion

If the leaf is a 3-node:
- We now have three values at this leaf
- Send the middle value up a node
- Make new 2-nodes out of the smallest and largest

Insert(T)

26

Insertion

Insertion

If the leaf is a 3-node:
- We now have three values at this leaf
- Send the middle value up a node
- Make new 2-nodes out of the smallest and largest

Insert(T)

Where should it go?

27

Insertion

Insertion

If the leaf is a 3-node:
- We now have three values at this leaf
- Send the middle value up a node
- Make new 2-nodes out of the smallest and largest

Insert(T)

28
Insertion

If the leaf is a 3-node:
- We now have three values at this leaf
- Send the middle value up a node
- Make new 2-nodes out of the smallest and largest

Insert(I)

Where should it go?
If the leaf is a 3-node:
- We now have three values at this leaf
- Send the middle value up a node
- Make new 2-nodes out of the smallest and largest

**Insert(I)**

What now?
**Insertion**

If the leaf is a 3-node:
- We now have three values at this leaf
- Send the middle value up a node
- Make new 2-nodes out of the smallest and largest

**Practice**

Draw the 2-3 tree that results when you insert the keys: E A S Y Q U T I O N in that order in an initially empty tree.

When will the height of the tree change?
Practice

Draw the 2-3 tree that results when you insert the keys: 
E A S Y Q U T I O N in that order in an initially empty tree.

Running time

Worst case height: $O(\log n)$

Insert, search and delete are all $O(\log n)$

2-3 search trees in practice

A pain to implement

Overhead can often make slower than standard BST

Other balanced trees exist that provide the same worst case guarantee, but are faster (e.g., red-black trees)
Red-black tree high-level

https://www.cs.usfca.edu/~galles/visualization/RedBlack.html