CSCB63

Design and Analysis of Data Structures

Worksheet 3 – Augmented AVL Trees

Augment the tree Take 2

Q: How can we *augment the nodes* of AVL trees so that we can perform all our *queries* efficiently?

Q: What *property of subtrees* could help us with questions about *rank* ?

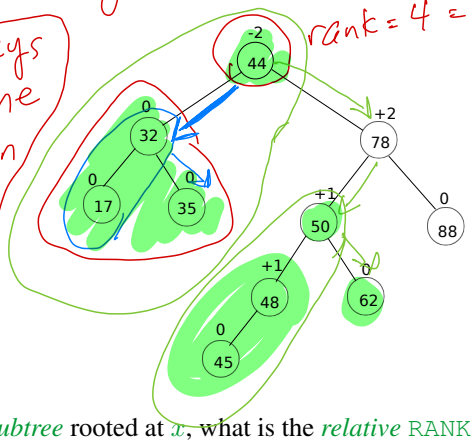
A. size of subtrees to compute rank.

Q: How is this related to '*rank*'?

A. the number of keys we skip over while searching for our key is the the rank (+1 for our key) of our given key.

rank(x) = 1 + # keys that come before x in the tree.

Relative Rank



rank = 4 = size left subtree +1

suppose x = 35

Q: Now with respect to the *left subtree* rooted at x, what is the *relative* `RANK(x)` ?

A. relative rank of a key x is the size of the left subtree +1. rank(35) = size(tree rooted 17) + 1 + 1 = 3

For example, add size fields to each node and then calculate the rank of 62.

rank(62) = size(44's left child) + 1 + size(50's left child) + 1
= 3 + 1 + 2 + 1 + 1 = 8

So the *rank* of a node is related to the *size of the subtrees* rooted at neighbouring nodes.

Computing `RANK(k)` : Given *key k*, do a

- `SEARCH(k)` keeping track of the *rank of the current node.*
- Each time you go *down a level* you must:
  → add to our rank (so far) all the left subtrees and nodes we pass by because they are less than k.
- Think of this as the "*relative*" *rank* of the key to the left of the subtree you are exploring.

Computing Rank as we Search

each node v has:
key field
size field: has # of nodes in tree rooted at v.

Week 3 Lecture 2 Worksheet Interval Trees

Collections of Intervals

Scenario. You have a set of *time intervals* representing when TA's have office hours.

Closed time intervals: $\{x \in \mathbb{R} \mid l \leq x \leq h\} = [l,h]$.

Representation: Just use l and h.

Operations:

- *insert*(l,h): Store $[l,h]$ in the collection.
- *delete*(l,h): Delete $[l,h]$.
- *search*(l,h): Return a *stored interval* that *overlaps* with $[l,h]$.

  Search represents finding when a TA is available when you are.

Goal. Want $O(\lg n)$ time each.

The data structure

Q. How can we do this?

A. Use a *balanced binary search tree* (AVL, Red Black Tree, weight balanced tree ...) to store the intervals.

Q. For BST order, how do we *compare* $[l,h]$ with $[l',h']$?

- If $l < l'$, then $[l,h] < [l',h']$.
- If $l = l'$ and $h < h'$, then $[l,h] < [l',h']$.

Q. Is this *sufficient*?

A.

Each node x_i stores:

- l_i and h_i: interval's two ends, and the key

CSCB63 Winter 2021

Week 6 Lecture 1

Dijkstra's Shortest Path Algorithm

Anna Bretscher

March 16, 2021

CSCB63 Winter 2021

Week 7 Lecture 1 - Amortized Analysis

Anna Bretscher

March 3, 2021

CSCB63 Winter 2021

Week 8 Lecture 1 - Disjoint Sets

Anna Bretscher

March 8, 2021