csce750 — Analysis of Algorithms
Fall 2019 — Lecture Notes: Augmenting Data Structures

This document contains slides from the lecture, formatted to be suitable for printing or individual reading, and with some supplemental explanations added. It is intended as a supplement to, rather than a replacement for, the lectures themselves — you should not expect the notes to be self-contained or complete on their own.

1 Introduction

In many cases, we can use existing data structures in unexpected ways by **augmenting** the data they store. CLRS 14

Basic steps:

1. **Choose** an underlying data structure.
2. **Determine** additional information to maintain in that data structure.
3. **Modify** the operations of that data structure to maintain that information.
4. **Develop** new operations using that information.

This approach is much more common than "starting from scratch" with a new data structure.

Many examples are based on balanced search trees.

2 Dynamic order statistics

Suppose we want a data structure that supports these operations:

- INSERT(k)
- SEARCH(k)
- DELETE(k)
- SELECT(i) — find the i^{th} smallest element
- RANK(v) — how many elements are smaller than the one at node v?

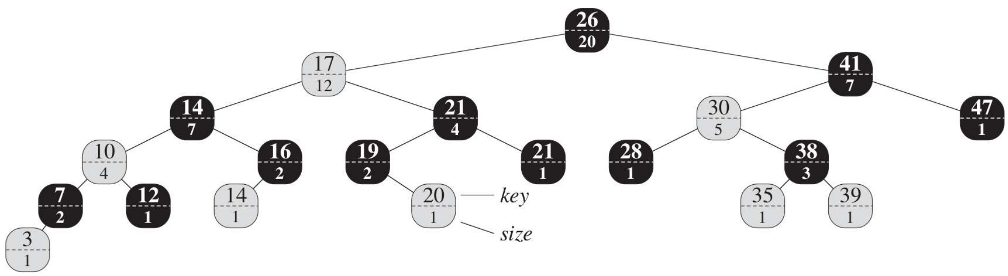
This differs from the standard selection problem, because the set is **dynamic** — elements may be added or deleted.

We can form a data structure that supports these operations by augmenting your favorite rotation-based balanced binary search tree data structure.

3 *Order statistic trees*

In addition to the usual attributes (key, left, right, parent), add a new attribute:

- Store the number of nodes in the subtree rooted at v as v.size.

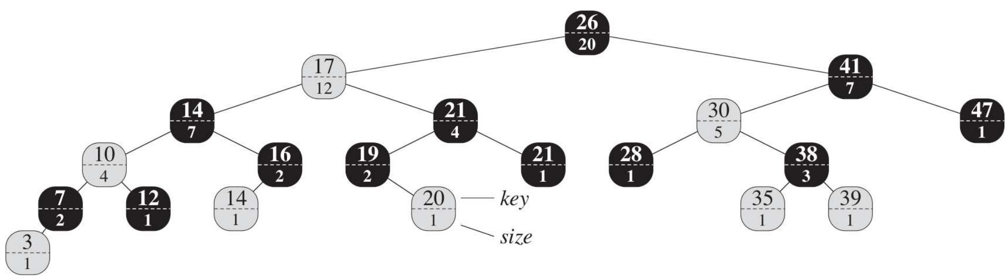


(CLRS Fig. 14.1)

$$v.\text{size} = v.\text{left.size} + x.\text{right.size} + 1$$

4 *Maintaining the size attribute:* INSERT

For each INSERT: Increment the size of each node along the way.



5 *Maintaining the size attribute:* ROTATE

Each rotation changes the size for only two nodes: The two nodes incident to the edge being rotated.

6 SELECT *in Order Statistic Trees*

The SELECT operation in an order statistic tree looks much like QUICKSELECT:

```
OST-SELECT(v, i)
  r = v.left.size + 1
  if i = r then
    return v
  else if i < r then
    return OST-SELECT(v.left, i)
  else
    return OST-SELECT(v.right, i − r)
  end if
```

This takes time proportional to the height of the tree, which is $\Theta(\lg n)$ for a balanced binary search tree.

7 Computing rank in order statistic trees

How can we use this data structure to compute the **rank** of a given node in the tree?

(CLRS 342)