Augmenting Data Structures

Nirdosh Bhatnagar

1. Introduction

It is unusual to have to design an all-new data structure from scratch. It is more common to take a data structure that you know and store additional information in it. With the new information, the data structure can support new operations. But you have to figure out how to *correctly maintain* the new information *without loss of efficiency.*

2. Dynamic Order Statistic

We've seen algorithms for finding the ith element of an unordered set in $O(n)$ time. We next, discuss a structure to support finding the ith element of a dynamic set in $O(\lg n)$ time. We also explore the following questions.

- What operations do dynamic sets usually support?
- What structure works well for these?
- How could we use this structure for order statistics?
- How might we augment it to support efficient extraction of order statistics?

We've seen algorithms for finding the ith element of an unordered set in $O(n)$ time. OS-Trees is a structure to support finding the ith element of a dynamic set in $O(\lg n)$ time. It supports standard dynamic set operations like SEARCH, MINIMUM, MAXIMUM, SUCCESSOR, PREDECESSOR, INSERT, and DELETE. It also supports these order statistic operations: OS-SELECT(x, i); and OS-RANK(T, x).

We want to support the usual dynamic-set operations from the R-B trees, Plus:

- OS-SELECT(x, i): returns pointer to node containing the ith smallest key of the subtree rooted at x.
- OS-RANK(T, x): returns the rank of the linear order determined by an inorder walk of tree T. The rank of an element is the position at which it would be printed in an inorder walk of the tree.

Description of an Order-Statistic Tree

An order-statistic tree is simply a red-black tree with additional information stored at each node. Besides the usual red-black tree attributes of a node: *color, key, left, right*, and p, we have another attribute *size*. This attribute contains the umber of (internal) nodes in the subtree rooted at node x (including x itself), that is the size of the subtree. If we define the sentinel's size to be 0 - that is, we set *T.ni.size* to be 0 then we have the identity

$$x.size = x.left.size + x.right.size + 1$$

The keys need not be distinct in the order-statistic tree. Therefore, in order to avoid this ambiguity for an order-statistic tree, we introduce the notion of rank of an element. The rank of an element is the position at which it would be printed in an inorder walk of the tree.

3. **Interval Trees**

Red-black trees can be augmented to support dynamic sets of intervals. Intervals are convenient for representing events that each occupy a continuous period of time. We might, for example, wish to query a database of time intervals to find out what events occurred during a given interval.

Read the section on interval trees in the CLRS textbook.