A Logarithmic Random Access Tree

Tom Morgan

November 3, 2006

Abstract

Making a data structure that performs like a dynamic array but functions in logarithmic time for all operations is the goal, but is by no means a trivial one. The obvious solution is to use a tree of some sort, but how?

By using a binary tree in which values are stored at the leaf nodes and each node keeps track of how many leaves there are below it, we can quickly achieve logarithmic random access, insertion and deletion in the average case by all operations are linear on the worst case. The task now is to balance this tree, but standard binary tree balancing techniques can not be used heavy handedly as the tree has new and specific properties.

Keywords: data structure, algorithmic efficiency, tree

Background

Dynamic arrays (such as vectors and array lists) are commonly used among programs to fill the places of arrays when the size is unknown and intermediate insertions and deletions are necessary. They are generally implemented as arrays whose data is shifted around when necessary and copied to a larger array when extra space is needed. Tradition dynamic arrays have O(1) random access, O(N) insertion (except to the end where it is amortized O(1)) and O(N) deletion (except from the end where it is amortized O(1)).

Red-Black Trees have been extended to allow for logarithmic access of the nth element, however although this is similar to random access it is not identical. Random access uses sequential integral keys and will change the keys of the other elements as elements are added or removed.

Scope

I intend to both design and implement a fully functioning data structure. I will have it implemented in both C++ and Java and apply it to different possible problems. Also, the theory will be fully worked through and the efficiency proven.

Type of Resarch

My project falls under pure basic research. I am devising an algorithm to fundamentally expand the knowledge we have of data structures and algorithmic efficiency.

Procedure and Methodology

The primary work in devising an algorithm is conceptual, however putting it into code greatly helps in fleshing out the ideas as well as identifying flaws in the process. Similarly, when put into code the efficiency of the general case can be tested empirically.

The various attempts at randomly accessible trees are implemented as classes in Java, and a separate Java program is used for testing the classes. The testing program allows for both randomly inserting and deleting data as well as sequential insertion and deletion so both random and extreme cases are testable.

The original procedure was to make a randomly accessible tree and then balance it but now the focus is being shifted to modifying an already balanced tree to allow for random access.

(Expected) Results

Currently, I have made a randomly accessible data structure that is guaranteed balanced and logarithmic in the general case but not one that has efficient balancing allowing for guaranteed logarithmic operations. Eventually, I intend to modify a standard Red-Black tree to allow for random access and that will meet all my specified requirements.

Value

This data structure will be useable in a wide variety of applications. Just like any other data structure it will be potentially usable in any program. If it is found to be usable enough, it may even find its way into standard libraries like dynamic arrays have.