A Logarithmic Randomly Accessible Data Structure

Tom Morgan
TJHSST Computer Systems Lab 2006-2007

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 but all operations are linear on the worst case. To allow balance the tree, a Red Black Tree is used. The Red Black Tree is always balanced and by using its balancing mechanisms and changing it to use the random access system described above, the goal is reached.

```
public void deleteOneChild ()
{
	// n has at most one non-null child
	RBTNode t = getRight() == null ? getLeft() : getRight();
	replaceNode(t);
	if (getColor() == BLACK)
	{
		if (t.getColor() == RED)
			t.setColor(BLACK);
		else
			t.delete1();
	}
}
```

Procedures and Methods

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.

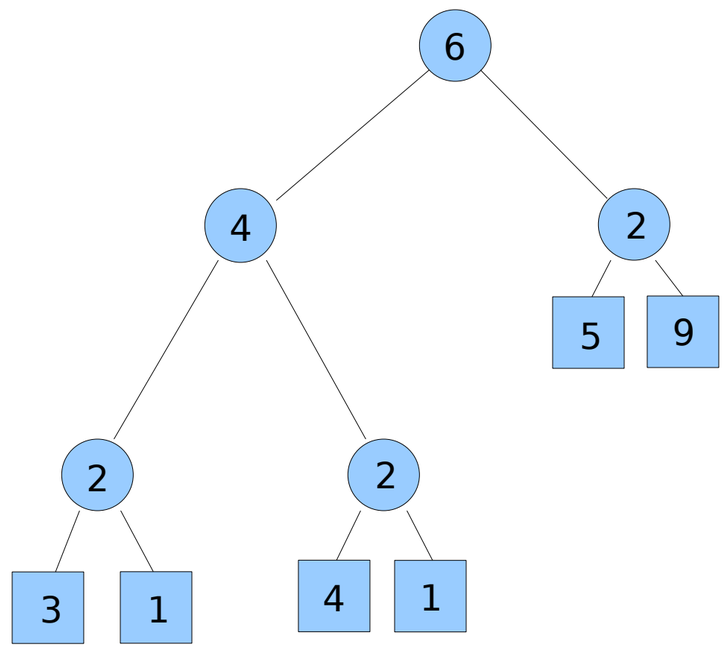
The various attempts at randomly accessible trees are implemented as classes in Java, and a separate Java program is used for testing the classes. The testing program allows for both randomly inserting and deleting data as well as sequential insertion and deletion so both random and extreme cases are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

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

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.



Results and Conclusions

Currently, I have made both a randomly accessible data structure that is guaranteed balanced and logarithmic in the general case and a standard Red Black Tree which is always logarithmic for insertion, deletion and lookup but is not randomly accessible at all. Now I am working to modify the Red Black Tree to give it random access capabilities.