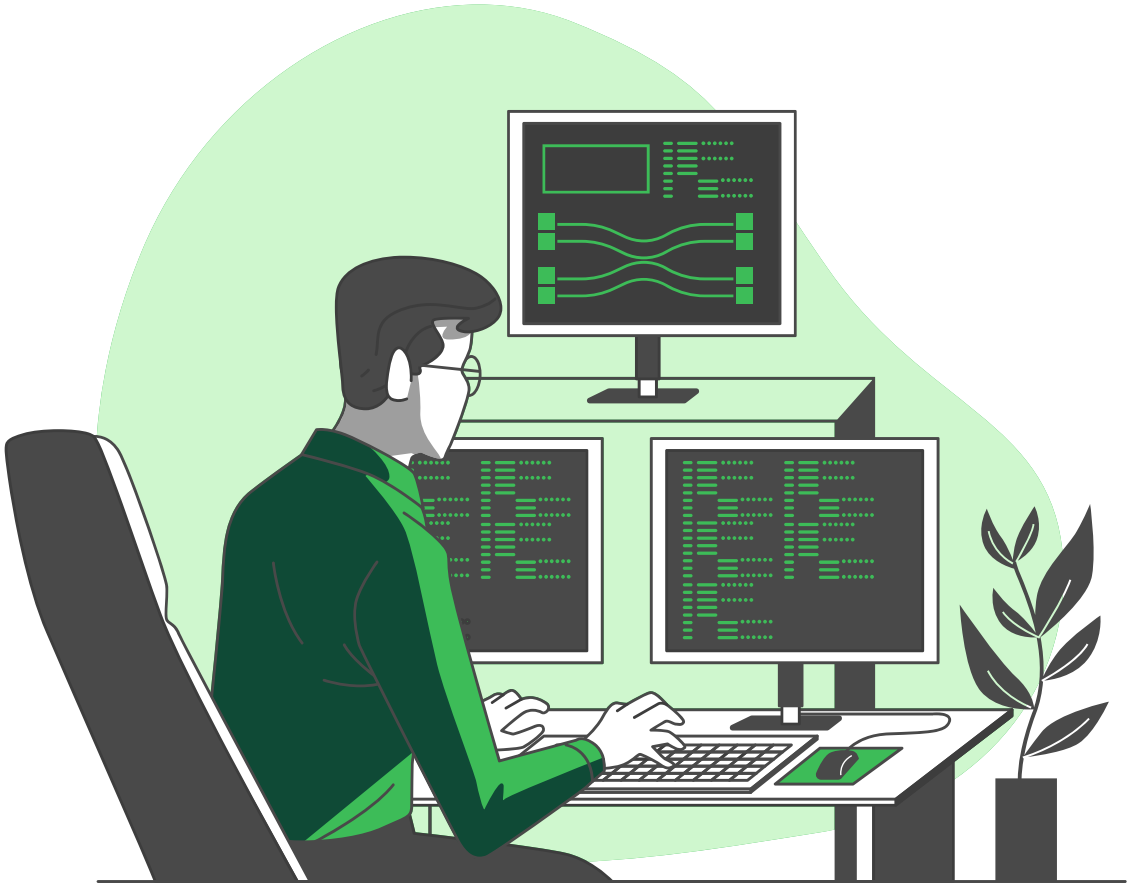


Red-Black Tree

CS 251 - Data Structures and Algorithms

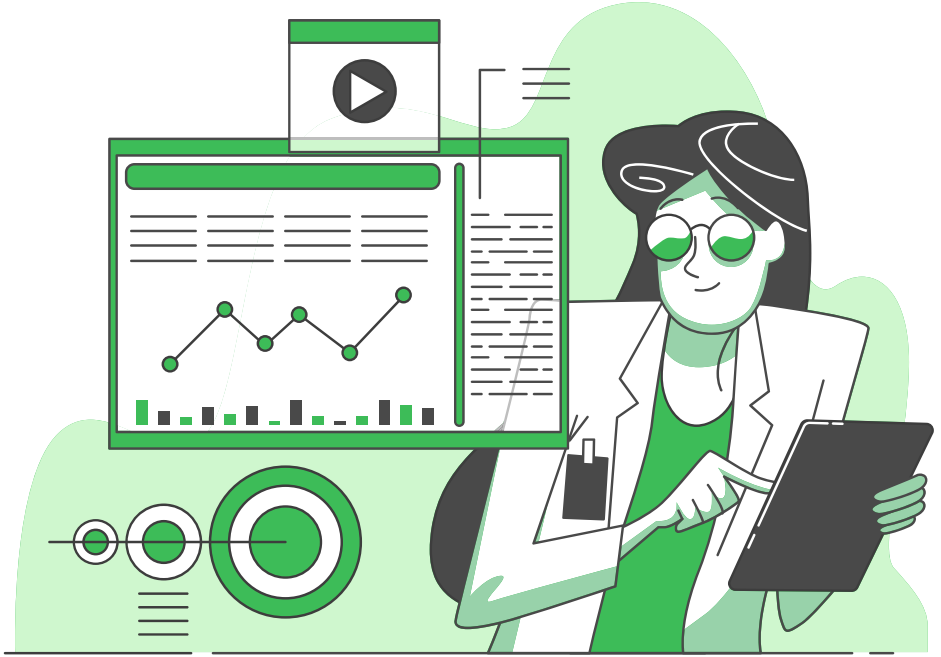
Note:
Slides complement the discussion in class

Table of Contents

01

Red-Black Tree

Enforcing “balance” on a binary search tree



01
Red-Black Tree

Enforcing “balance” on a binary search tree

L. J. Guibas and R. Sedgewick, "A dichromatic framework for balanced trees," 19th Annual Symposium on Foundations of Computer Science (sfcs 1978), Ann Arbor, MI, USA, 1978, pp. 8-21

A DICHROMATIC FRAMEWORK FOR BALANCED TREES

Leo J. Guibas
Xerox Palo Alto Research Center,
Palo Alto, California, and
Carnegie-Mellon University

and

Robert Sedgewick*
Program in Computer Science
Brown University
Providence, R. I.

ABSTRACT

In this paper we present a uniform framework for the implementation and study of balanced tree algorithms. We show how to imbed in this framework the best known balanced tree techniques and then use the framework to develop new algorithms which perform the update and rebalancing in one pass, on the way down towards a leaf. We conclude with a study of performance issues and concurrent updating.

0. Introduction

Balanced trees are among the oldest and most widely used data structures for searching. These trees allow a wide variety of operations, such as search, insertion, deletion, merging, and splitting to be performed in time $O(\lg N)$, where N denotes the size of the tree [AHU], [Kn]. (Throughout the paper lg will denote log to the base 2.) A number of different types of balanced trees have been proposed, and while the related algorithms are often conceptually simple, they have proven cumbersome to implement in practice. Also, the variety of such trees and the lack of good analytic results describing their performance has made it difficult to decide which is best in a given situation.

In this paper we present a uniform framework for the implementation and study of balanced tree algorithms. The framework deals exclusively with binary trees which contain two kinds of nodes: internal and external. Each internal node contains a key (chosen from a linear order) and has two links to other nodes (internal or external). External nodes contain no keys and have null links. If such a tree is traversed in symmetric order [Kn] then the internal nodes will be visited in increasing order of their keys. A second defining feature of the framework is that it allows one bit per node, called the *color* of the node, to store balance information. We will use *red* and *black* as the two colors. In section 1 we further elaborate upon this dichromatic framework and show how to imbed in it the best known balanced tree algorithms. In doing so, we will discover suprising new and efficient implementations of these techniques.

In section 2 we use the framework to develop new balanced tree algorithms which perform the update and rebalancing in one pass, on the way down towards a leaf. As we will see, this has a number of significant advantages over the older methods. We shall examine a number of variations on a common theme and exhibit full implementations which are notable for their brevity. One implementation is examined carefully, and some properties about its behavior are proved.

In both sections 1 and 2 particular attention is paid to practical implementation issues, and complete implementations are given for all of the important algorithms. This is significant because one measure under which balanced tree algorithms can differ greatly is the amount of code required to actually implement them.

Section 3 deals with the analysis of the algorithms. New results are given for the worst case performance, and a technique for studying the average case is described. While no balanced tree algorithm has yet satisfactorily submitted to an average case analysis, empirical results are given which show that the various algorithms differ only slightly in performance. One implication of this is that the top-down algorithms of section 2 can be recommended for most applications because of their simplicity.

Finally, in section 4, we discuss some other properties of the trees. In particular, a one-pass top down deletion algorithm is presented. In addition, we consider how to decouple the balancing from the updating operations and we explore parallel updating.

1. The Uniform Framework

In this section we present a uniform framework for describing balanced trees. We show how to embed in this framework the most widely used balanced tree schemes, namely B-trees [BaM], and AVL trees [AVL]. In fact, this embedding will give us interesting and novel implementations of these two schemes.

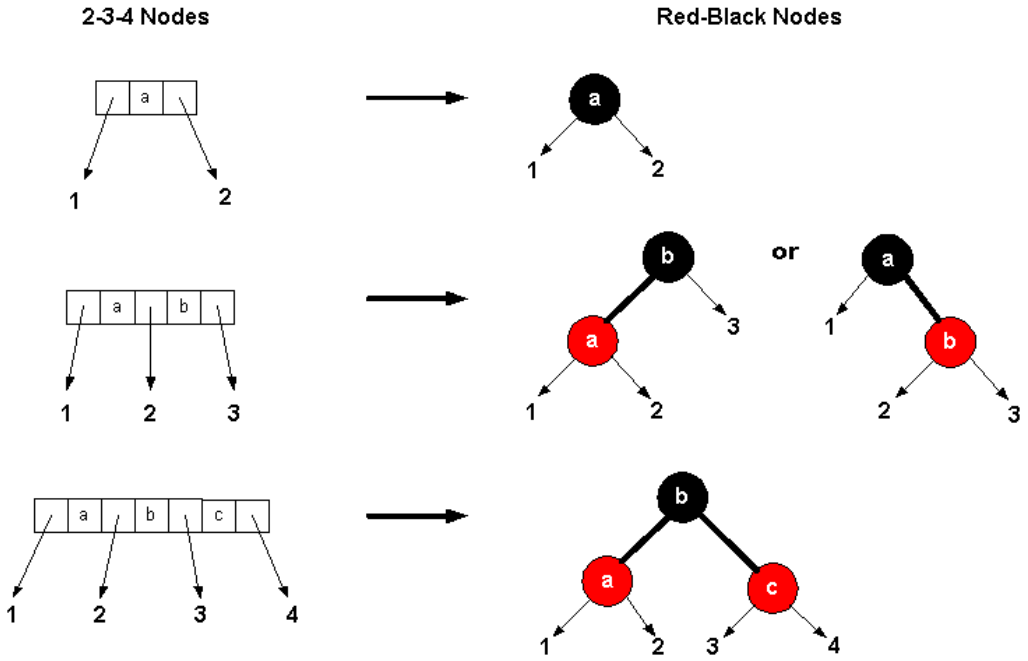
We consider rebalancing transformations which maintain the symmetric order of the keys and which are local to a small portion of the tree for obvious efficiency reasons. These transformations will change the structure of the tree in the same way as the single and double rotations used by AVL trees [Kn]. The difference between the various algorithms we discuss arises in the decision of *when* to rotate, and in the manipulation of the node colors.

For our first example, let us consider the implementation of 2-3 trees, the simplest type of B-tree. Recall that a 2-3 tree consists of 2-nodes, which have one key and two sons, 3-nodes, which have two

* This work was done in part while this author was a Visiting Scientist at the Xerox Palo Alto Research Center and in part under support from the National Science Foundation, grant no. MCS75-23738.

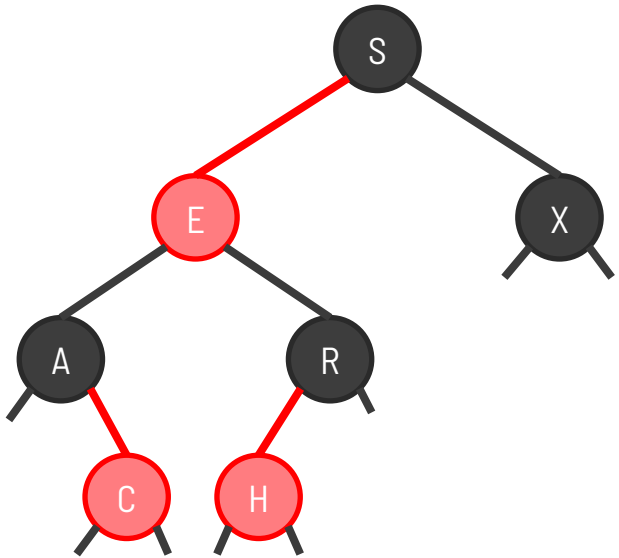
https://doi.org/10.1109/SFCS.1978.3

2-3-4 Tree ↔ Red-Black Tree



https://pontus.digipen.edu/~mmead/www/Courses/CS280/Trees-Mapping2-3-4IntoRB.html

Red-Black Tree

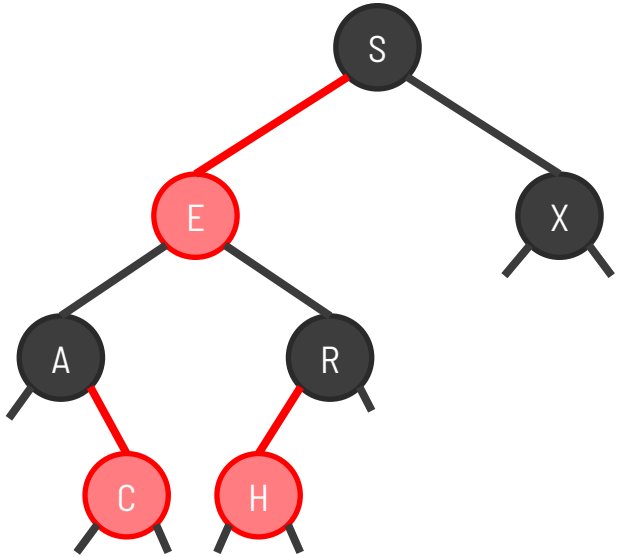


A type of self-balancing binary search tree.
The following properties apply:

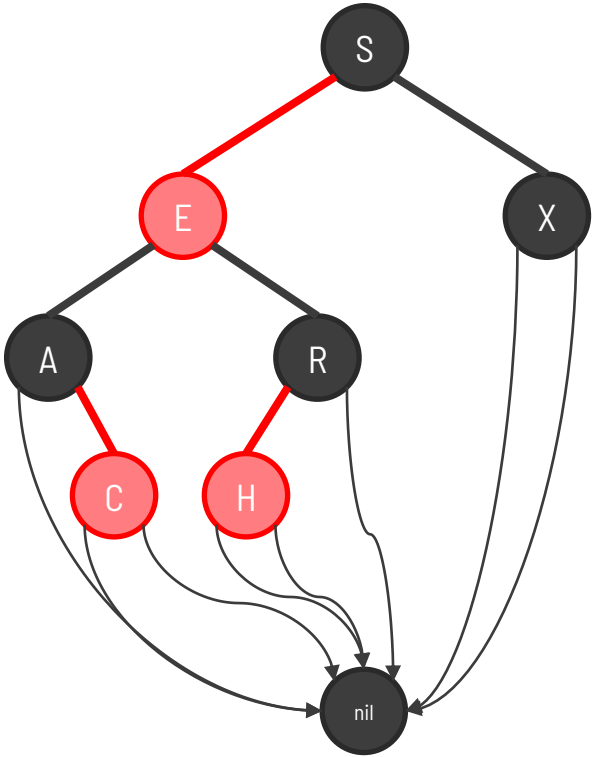
1. Every node is either **red** or **black**.
2. The root is **black**.
3. Every null link is **black**.
4. If a node is **red**, both children are **black**.
5. All paths from the root to a null link have the same number of **black** nodes/links **after** the root (aka. **black height**).

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. Introduction to Algorithms (The MIT Press) (p. 308).

Black Height



black-height = 2
(the root does not count)



Done!

Do you have any questions?