7 Red-Black Trees

Binary search trees are an elegant implementation of the *dictionary* data type, which requires support for

```
item SEARCH (item),
void INSERT (item),
void DELETE (item),
```

and possible additional operations. Their main disadvantage is the worst case time $\Omega(n)$ for a single operation. The reasons are insertions and deletions that tend to get the tree unbalanced. It is possible to counteract this tendency with occasional local restructuring operations and to guarantee logarithmic time per operation.

2-3-4 trees. A special type of balanced tree is the 2-3-4 *tree*. Each internal node stores one, two, or three items and has two, three, or four children. Each leaf has the same depth. As shown in Figure 15, the items in the internal nodes separate the items stored in the subtrees and thus facilitate fast searching. In the smallest 2-3-4 tree of height h, every internal node has exactly two children, so we have 2^h leaves and $2^h - 1$ internal nodes. In the largest 2-3-4 tree of height h, every internal node has four children, so we have 4^h leaves and $(4^h - 1)/3$ internal nodes. We can store a 2-3-4 tree in a binary tree by expanding a node with $i > 1$ items and $i + 1$ children into i nodes each with one item, as shown in Figure 16.

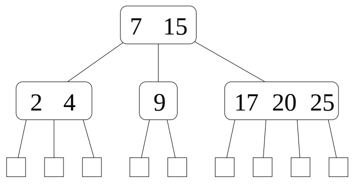

Figure 15: A 2-3-4 tree of height two. All items are stored in internal nodes.

Red-black trees. Suppose we color each edge of a binary search tree either red or black. The color is conveniently stored in the lower node of the edge. Such a edge-colored tree is a *red-black tree* if

(1) there are no two consecutive red edges on any descending path and every maximal such path ends with a black edge;

(2) all maximal descending paths have the same number of black edges.

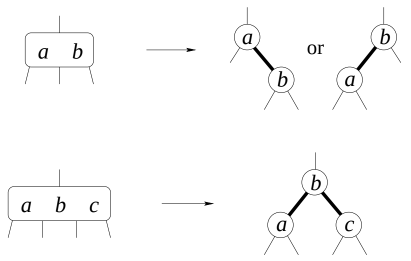

Figure 16: Transforming a 2-3-4 tree into a binary tree. Bold edges are called red and the others are called black.

The number of black edges on a maximal descending path is the *black height*, denoted as $bh(\varrho)$. When we transform a 2-3-4 tree into a binary tree as in Figure 16, we get a red-black tree. The result of transforming the tree in Figure 15 is shown in Figure 17.

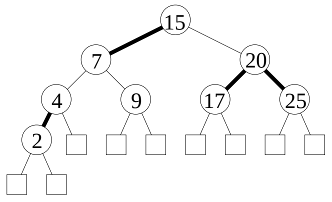

Figure 17: A red-black tree obtained from the 2-3-4 tree in Figure 15.

HEIGHT LEMMA. A red-black tree with n internal nodes has height at most $2\log_2(n + 1)$.

PROOF. The number of leaves is $n + 1$. Contract each red edge to get a 2-3-4 tree with $n + 1$ leaves. Its height is $h \leq \log_2(n + 1)$. We have $bh(\varrho) = h$, and by Rule (1) the height of the red-black tree is at most $2bh(\varrho) \leq 2\log_2(n + 1)$. ◻

Rotations. Restructuring a red-black tree can be done with only one operation (and its symmetric version): a *rotation* that moves a subtree from one side to another, as shown in Figure 18. The ordered sequence of nodes in the left tree of Figure 18 is

$$\ldots, \text{order}(A), \nu, \text{order}(B), \mu, \text{order}(C), \ldots,$$

and this is also the ordered sequence of nodes in the right tree. In other words, a rotation maintains the ordering. Function ZIG below implements the right rotation:

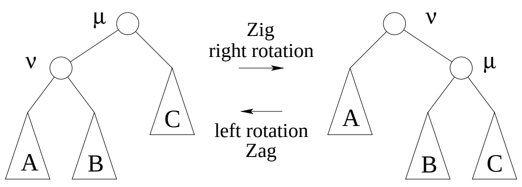

Figure 18: From left to right a right rotation and from right to left a left rotation.

```
Node $*$ ZIG(Node $*\mu$)
  assert $\mu \neq$ NULL and $\nu = \mu \to \ell \neq$ NULL;
  $\mu \to \ell = \nu \to r$;  $\nu \to r = \mu$;  return $\nu$.
```

Function ZAG is symmetric and performs a left rotation. Occasionally, it is necessary to perform two rotations in sequence, and it is convenient to combine them into a single operation referred to as a *double rotation*, as shown in Figure 19. We use a function ZIGZAG to implement a

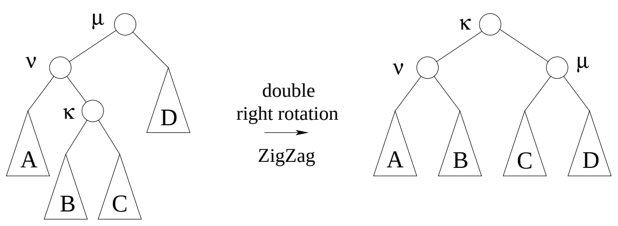

Figure 19: The double right rotation at μ is the concatenation of a single left rotation at ν and a single right rotation at μ.

double right rotation and the symmetric function ZAGZIG to implement a double left rotation.

```
Node $*$ ZIGZAG(Node $*\mu$)
  $\mu \to \ell =$ ZAG($\mu \to \ell$);  return ZIG($\mu$).
```

The double right rotation is the composition of two single rotations: ZIGZAG(μ) = ZIG$(\mu) \circ$ ZAG(ν). Remember that the composition of functions is written from right to left, so the single left rotation of ν precedes the single right rotation of μ. Single rotations preserve the ordering of nodes and so do double rotations.

Insertion. Before studying the details of the restructuring algorithms for red-black trees, we look at the trees that arise in a short insertion sequence, as shown in Figure 20. After adding 10, 7, 13, 4, we have two red edges in sequence and repair this by promoting 10 (A). After adding 2, we repair the two red edges in sequence by a single rotation of 7 (B). After adding 5, we promote 4 (C), and after adding 6, we do a double rotation of 7 (D).

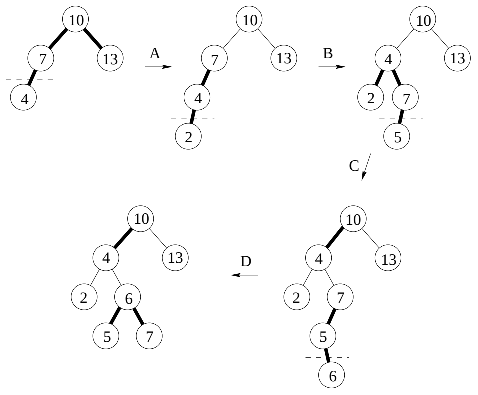

Figure 20: Sequence of red-black trees generated by inserting the items 10, 7, 13, 4, 2, 5, 6 in this sequence.

An item x is added by substituting a new internal node for a leaf at the appropriate position. To satisfy Rule (2) of the red-black tree definition, color the incoming edge of the new node red, as shown in Figure 21. Start the

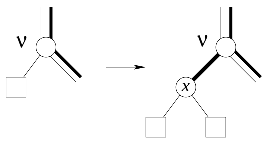

Figure 21: The incoming edge of a newly added node is always red.

adjustment of color and structure at the parent ν of the new node. We state the properties maintained by the insertion algorithm as invariants that apply to a node ν traced by the algorithm.

INVARIANT I. The only possible violation of the red-black tree properties is that of Rule (1) at the node ν, and if ν has a red incoming edge then it has exactly one red outgoing edge.

Observe that Invariant I holds right after adding x. We continue with the analysis of all the cases that may arise. The local adjustment operations depend on the neighborhood of ν.

`Case 1.` The incoming edge of ν is black. Done.

`Case 2.` The incoming edge of ν is red. Let μ be the parent of ν and assume ν is left child of μ.

`Case 2.1.` Both outgoing edges of μ are red, as in Figure 22. Promote μ. Let ν be the parent of μ and recurse.

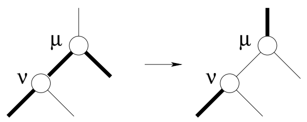

Figure 22: Promotion of μ. (The colors of the outgoing edges of ν may be the other way round).

`Case 2.2.` Only one outgoing edge of μ is red, namely the one from μ to ν.

`Case 2.2.1.` The left outgoing edge of ν is red, as in Figure 23 to the left. Right rotate μ. Done.

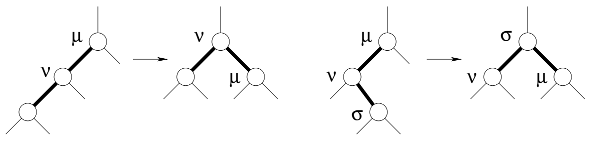

Figure 23: Right rotation of μ to the left and double right rotation of μ to the right.

`Case 2.2.2.` The right outgoing edge of ν is red, as in Figure 23 to the right. Double right rotate μ. Done.

Case 2 has a symmetric case where left and right are interchanged. An insertion may cause logarithmically many promotions but at most two rotations.

Deletion. First find the node π that is to be removed. If necessary, we substitute the inorder successor for π so we can assume that both children of π are leaves. If π is last in inorder we substitute symmetrically. Replace π by a leaf ν, as shown in Figure 24. If the incoming edge of π is red then change it to black. Otherwise, remember the incoming edge of ν as 'double-black', which counts as two black edges. Similar to insertions, it helps to understand the deletion algorithm in terms of a property it maintains.

INVARIANT D. The only possible violation of the red-black tree properties is a double-black incoming edge of ν.

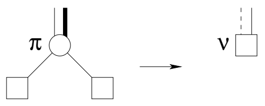

Figure 24: Deletion of node π. The dashed edge counts as two black edges when we compute the black depth.

Note that Invariant D holds right after we remove π. We now present the analysis of all the possible cases. The adjustment operation is chosen depending on the local neighborhood of ν.

`Case 1.` The incoming edge of ν is black. Done.

`Case 2.` The incoming edge of ν is double-black. Let μ be the parent and κ the sibling of ν. Assume ν is left child of μ and note that κ is internal.

`Case 2.1.` The edge from μ to κ is black.

`Case 2.1.1.` Both outgoing edges of κ are black, as in Figure 25. Demote μ. Recurse for $\nu = \mu$.

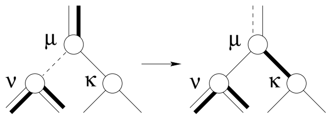

Figure 25: Demotion of μ.

`Case 2.1.2.` The right outgoing edge of κ is red, as in Figure 26 to the left. Change the color of that edge to black and left rotate μ. Done.

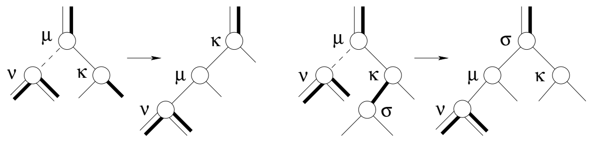

Figure 26: Left rotation of μ to the left and double left rotation of μ to the right.

`Case 2.1.3.` The right outgoing edge of κ is black, as in Figure 26 to the right. Change the color of the left outgoing edge to black and double left rotate μ. Done.

`Case 2.2.` The edge from μ to κ is red, as in Figure 27. Left rotate μ. Recurse for ν.

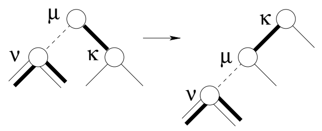

Figure 27: Left rotation of μ.

Case 2 has a symmetric case in which ν is the right child of μ. Case 2.2 seems problematic because it recurses without moving ν any closer to the root. However, the configuration excludes the possibility of Case 2.2 occurring again. If we enter Cases 2.1.2 or 2.1.3 then the termination is immediate. If we enter Case 2.1.1 then the termination follows because the incoming edge of μ is red. The deletion may cause logarithmically many demotions but at most three rotations.

Summary. The red-black tree is an implementation of the dictionary data type and supports the operations search, insert, delete in logarithmic time each. An insertion or deletion requires the equivalent of at most three single rotations. The red-black tree also supports finding the minimum, maximum and the inorder successor, predecessor of a given node in logarithmic time each.