Cairo University
Faculty of Computers and Artificial Intelligence
Computer Science Department

Advanced Data Structures

Red-Black Trees

Dr. Amin Allam

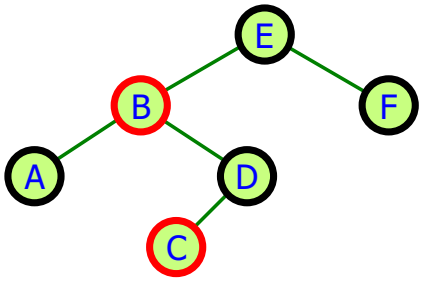
[For more details, refer to "Introduction to Algorithms" by *Thomas Cormen*, et al.]

1 Introduction

A *red-black tree* is a *balanced binary search tree* whose height grows only *logarithmically* to the number of nodes in the tree. Each node in the *red-black tree* has a colour, which is either *red* or *black*. The colour attribute helps balancing the tree. The colour attribute is just a boolean flag. Colours are useful only for visual illustration.

A *binary search tree* is a *red-black tree* if it satisfies the following properties (invariants):

1) Every node is either *red* or *black*.
2) The root is *black*.
3) The tree does not contain two adjacent *red* nodes.
4) For each node Q: All paths from Q to its descendant leaves have equal number of *black* nodes.



The last property is the key to understand the *intuition* behind *red-black trees*. Suppose we do not allow any *red* nodes, the last property is too strict since it holds only if the tree is full and the tree is perfectly balanced, and it can never hold when the number of nodes is not $2^n - 1$ for some $n \geq 1$. The incorporation of the first three properties allows the existence of *red* nodes, such that the number of *red* nodes on any path is not more than the number of *black* nodes. Now, applying the last property to only *black* nodes is possible, and guarantees that the maximum path length is at most twice the optimal one.

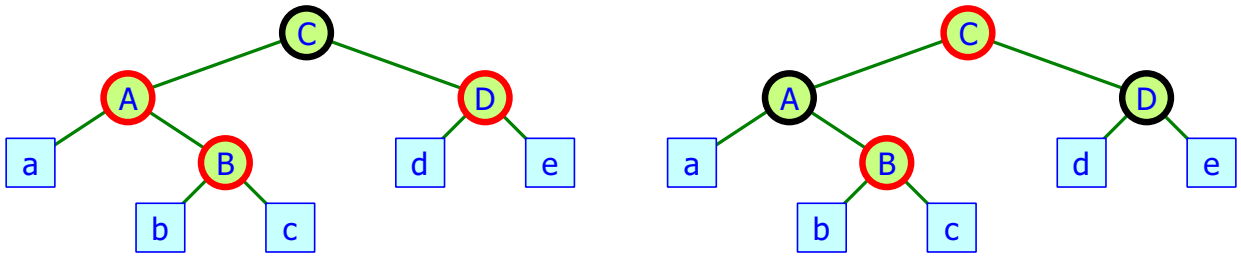
Proposition: Let $B(x)$ be the *black-height* of x, which is the number of *black* nodes on any simple path from x down to a leaf node (including x). The subtree rooted at any node x contains at least $2^{B(x)} - 1$ nodes.
Proof: (By induction on the height of x)
Base step: If the height of x is 0, x is a leaf. $B(x)$ is 0 or 1, depending on its colour. $2^{B(x)} - 1$ is 0 or 1. The subtree rooted at x contains exactly 1 node, which is at least $2^{B(x)} - 1$.
Induction step: Consider a node x whose height > 0 with two children. Each child has a *black-height* of $B(x)$ or $B(x-1)$ depending on its colour. Since the height of a child of x is $<$ the height of x, we can apply the inductive hypothesis to conclude that each child has at least $2^{B(x)-1} - 1 = 0$ nodes. Thus, the subtree rooted at x contains at least $2 \times (2^{B(x)-1} - 1) + 1 = 2^{B(x)} - 1$ nodes.

Lemma: A *red-black tree* has height $\leq 2\ log_2(n+1)$ where n is the number of nodes.
Proof: Let h be the height of the tree. Since there cannot be two adjacent *red* nodes, at least half of the nodes on any simple path from the root to a leaf must be *black*. Thus, $B(root) \geq h/2$. Applying the proposition above, $n \geq 2^{B(x)} - 1 \geq 2^{h/2} - 1$. So, $h \leq 2\ log_2(n+1)$.

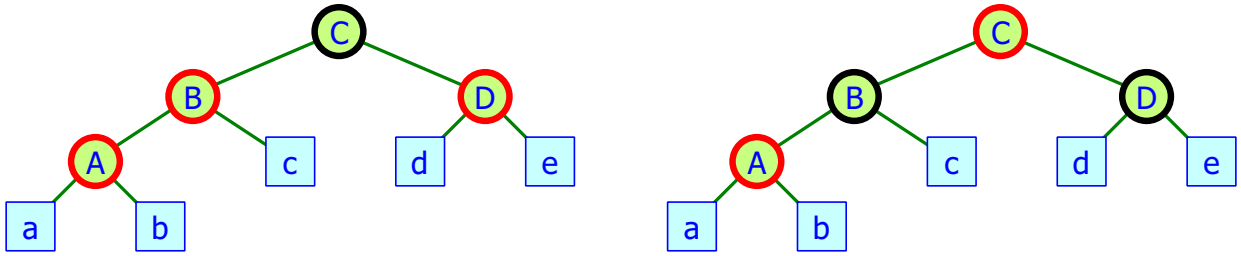
2 Insertion

A node is inserted and coloured *red* in order to preserve *property 4*. If its parent is *red*, one or more of the following cases is followed to preserve *property 3*. If the root becomes *red*, it is just converted into *black*. The first two cases should be continued. Symmetric cases are not shown.

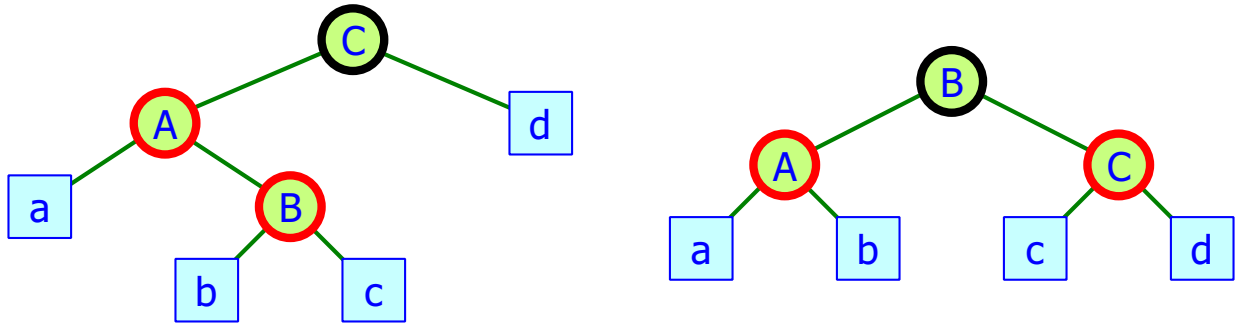
2.1 Case-1



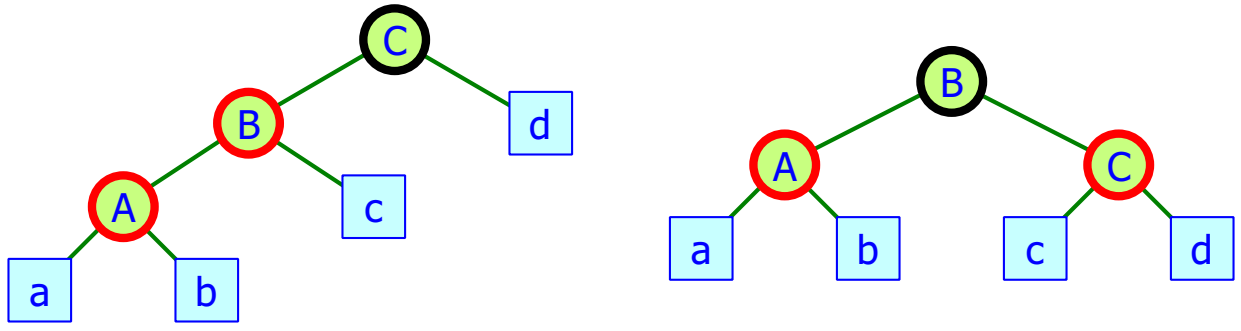
2.2 Case-2



2.3 Case-3



2.4 Case-4

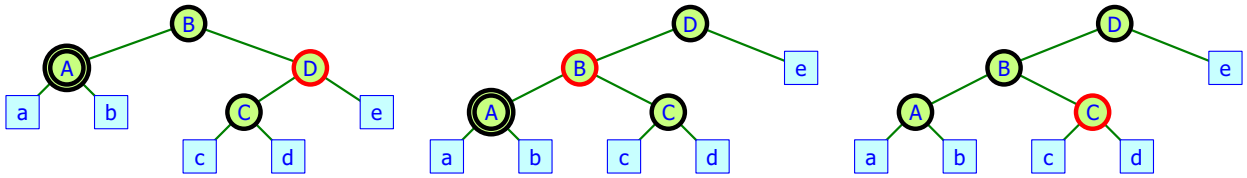


3 Deletion

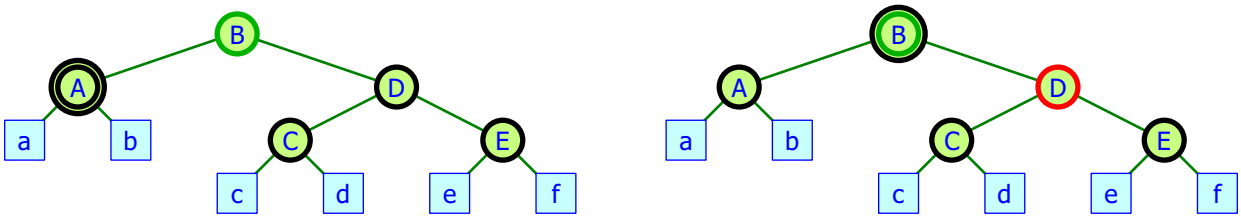
The child of the physically deleted node gets an additional *black*. If it was originally *red*, it is converted into *black*. Otherwise, one or more of the following cases is followed in order to remove the additional *black* while maintaining other properties. The second case should be continued. Symmetric cases are not shown.

Suppose that the deleted node was a leaf, how to assign additional colour to its child? It can be done using a special case. However, to simplify the implementation, we insert two additional *black* children for each leaf. These additional children does not contain data, and just act as *sentinels* that simplify coding.

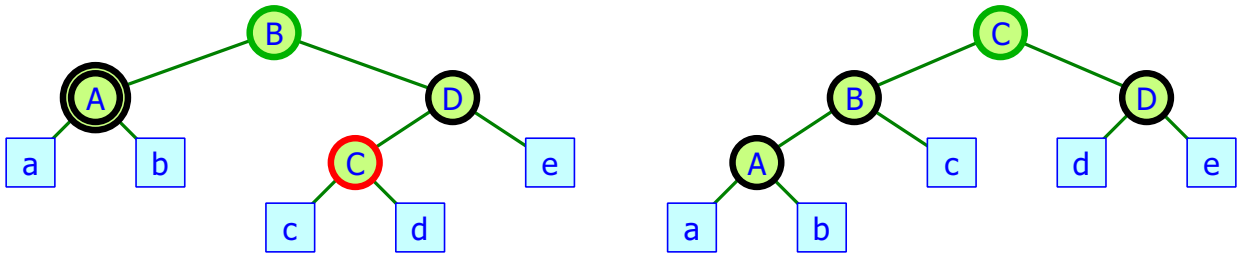
3.1 Case-1



3.2 Case-2



3.3 Case-3



3.4 Case-4