Cairo University
Faculty of Computers and Artificial Intelligence
Computer Science Department

Advanced Data Structures

Red-Black Trees

Dr. Amin Allam

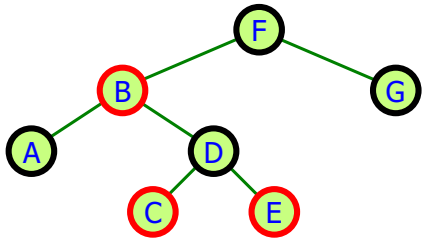
[For more details, refer to "Introduction to Algorithms" by *Thomas Cormen*, et al.]

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 is a boolean attribute that helps balancing the tree. We define the *height* of a node x to be $H(x) =$ the number of edges on a path from x down to the farthest *leaf*.

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

1) Every node is either *red* or *black*. The *root* is *black*.
2) Every *internal* (non-*leaf*) node has exactly two children.
3) The tree does not contain any two adjacent *red* nodes.
4) All paths from any given node x to its descendant leaves have equal number of *black* nodes $= B(x)$.



The combination of the second and fourth properties is the key to understand the *intuition* behind *red-black trees*. Suppose we do not allow any *red* nodes, these properties are too strict since they hold only if the tree is full and the tree is perfectly balanced, and they can never hold when the number of nodes is not $2^n - 1$ for some $n \geq 1$. The incorporation of the first and third 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: The subtree rooted at any node x contains at least $2^{B(x)} - 1$ nodes.
Proof: By induction on $H(x)$: *Base step:* Consider a *leaf* x If $H(x) = 0$. $B(x)$ is 0 or 1, depending on its colour. $2^{B(x)} - 1$ is 0 or 1. The subtree rooted at x contains 1 node, which is at least $2^{B(x)} - 1$. *Induction step:* Consider a node x whose $H(x) > 0$ with two children y and z. Each of $B(y)$ and $B(z)$ equals to $B(x)$ or $B(x) - 1$ depending on x colour. Since $H(y)$ and $H(z)$ are $< H(x)$, we can apply the inductive hypothesis to conclude that each child has at least $2^{B(x)-1} - 1$ nodes. Thus, the subtree rooted at x contains at least $2 \times (2^{B(x)-1} - 1) + 1 = 2^{B(x)} - 1$ nodes.

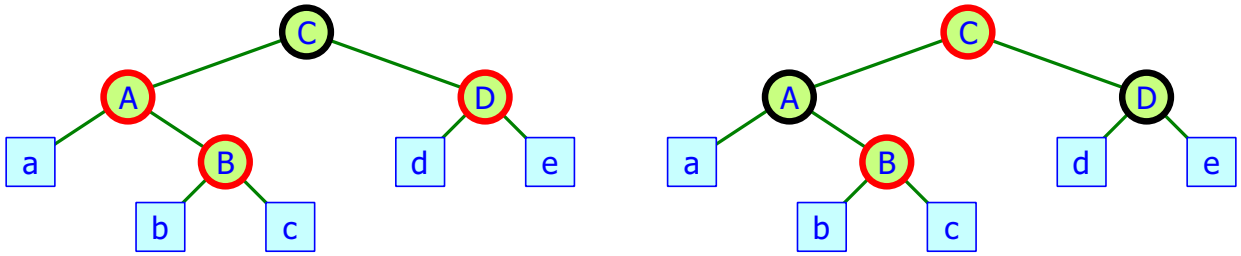
Lemma: A *red-black tree* has height $H(root) \leq 2\ log_2(n + 1)$ where n is the number of nodes.
Proof: Since there cannot be two adjacent *red* nodes, at least half of the nodes on any simple path from the *root* to a *leaf* must be *black*. Thus, $B(root) \geq H(root)/2$. Applying the proposition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

– In order to simplify the implementation and maintain *property 2*, we insert two additional *black* children for each *leaf*. These additional children are just *sentinels* which do not contain data.

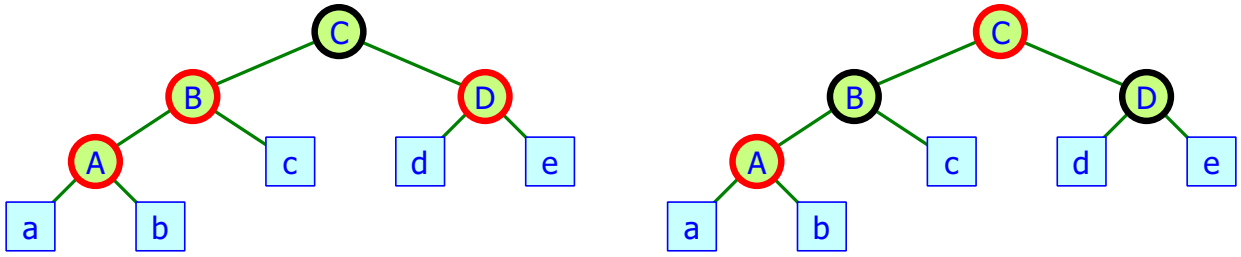
2 Insertion

A node is inserted and coloured *red* in order to maintain *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. The subtrees `a`, `b`, `c`, `d`, `e` have *black* roots.

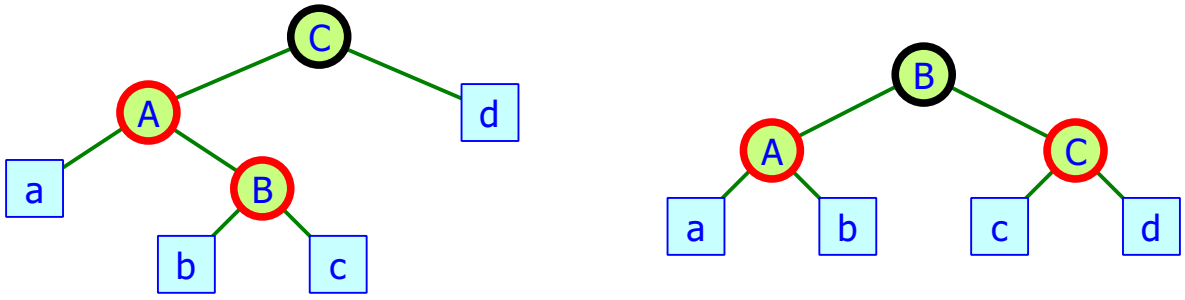
2.1 Case-1



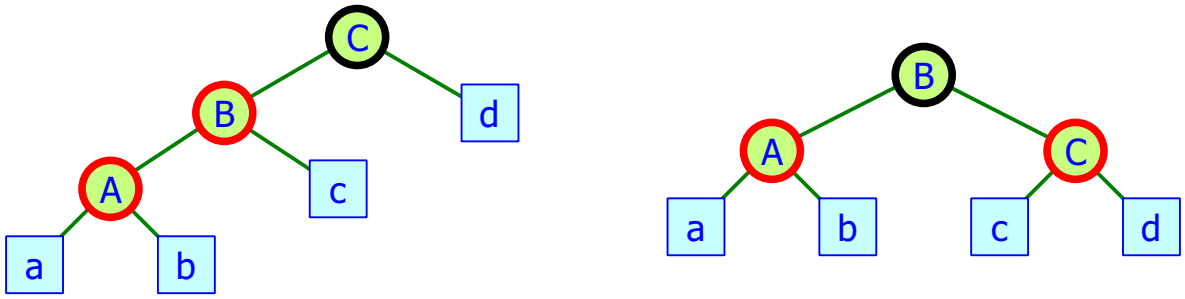
2.2 Case-2



2.3 Case-3



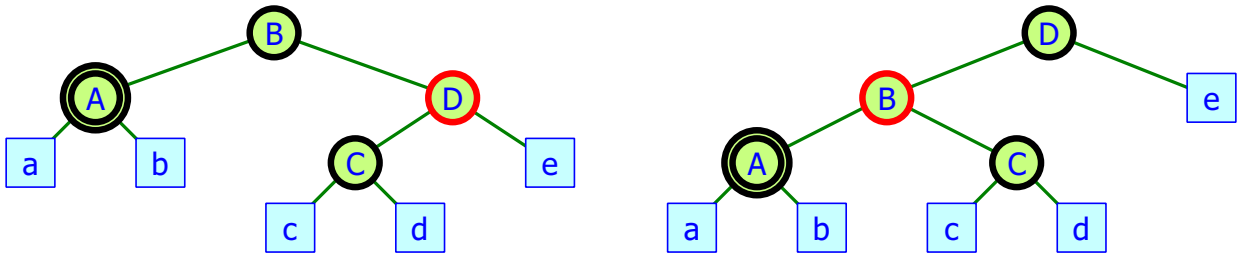
2.4 Case-4



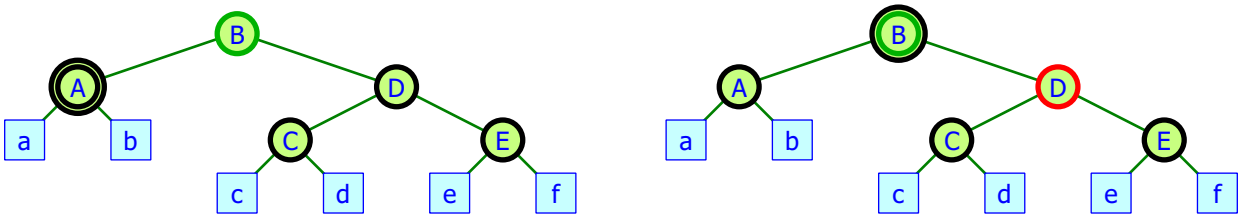
3 Deletion

If the physically deleted node was *red*, no changes are needed. Otherwise, the child of the physically deleted node gets an *additional black* in order to maintain *property 4*. The *additional black* should be removed from that node to preserve *property 1*. If it was originally *red*, it is just 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 first two cases should be continued. Symmetric cases are not shown.

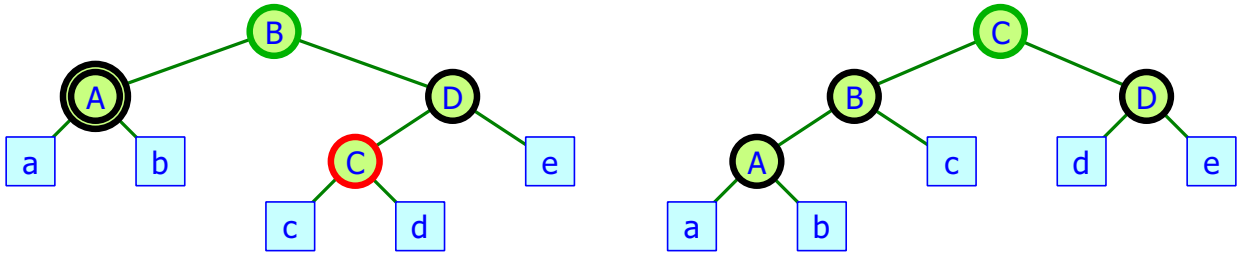
3.1 Case-1



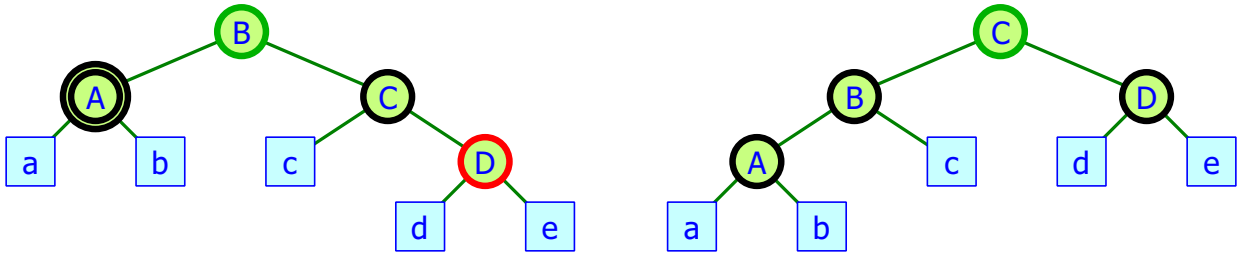
3.2 Case-2



3.3 Case-3



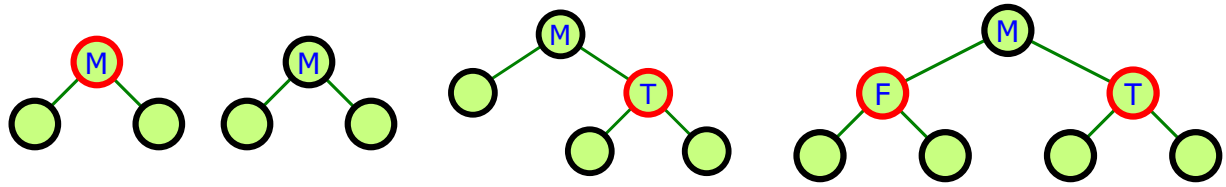
3.4 Case-4



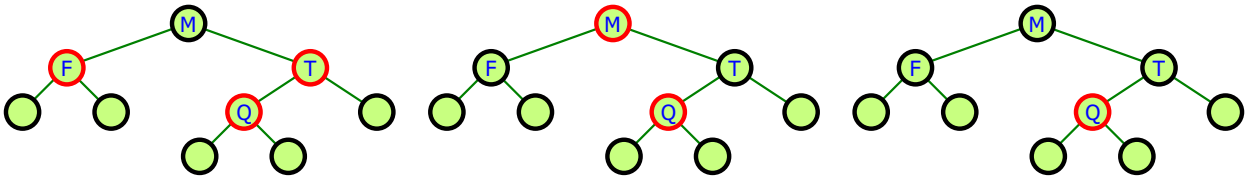
4 Example:

Starting from an initial empty tree, draw all intermediate trees and draw the tree after each of the following operations:
Insert(M), Insert(T), Insert(F), Insert(Q), Insert(P), Delete(F), Delete(Q), Delete(T).

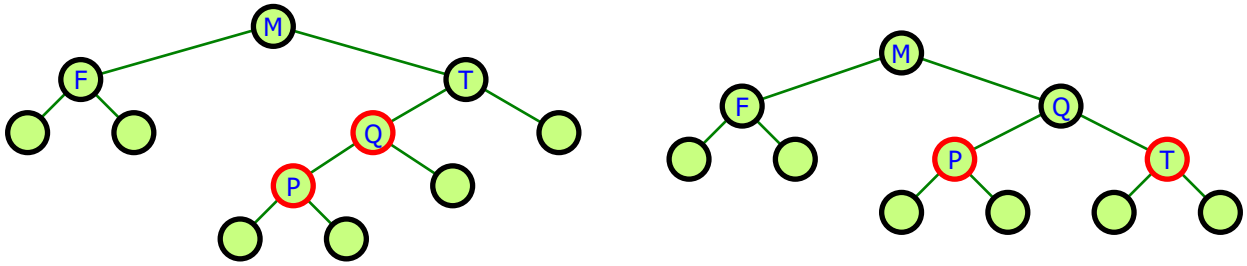
4.1 Insert(M), Insert(T), Insert(F)



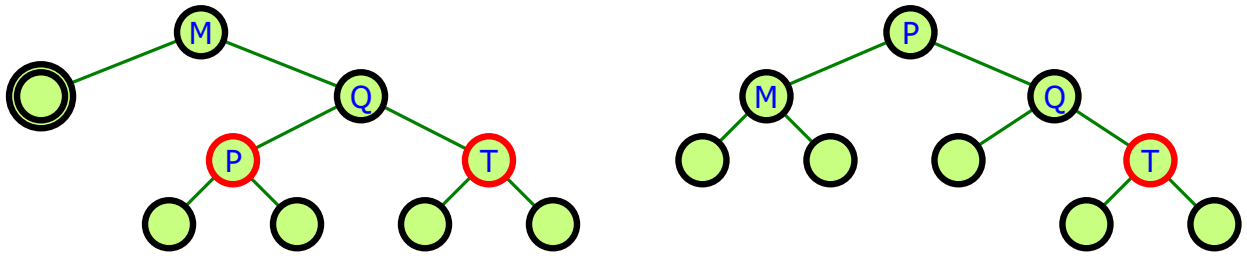
4.2 Insert(Q)



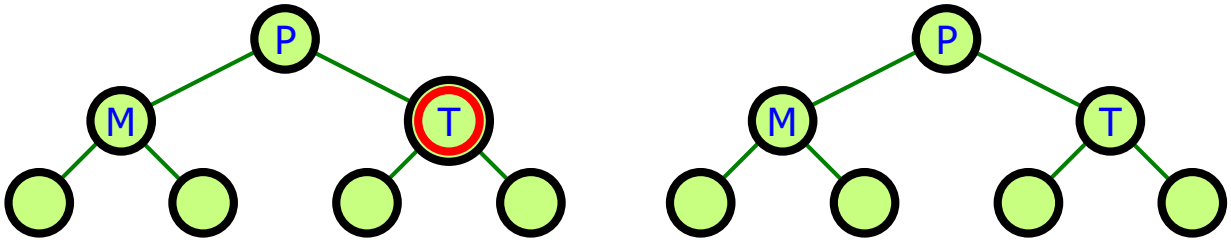
4.3 Insert(P)



4.4 Delete(F)



4.5 Delete(Q)



4.6 Delete(T)