Chapter 13 Red-Black Trees

Introduction to Algorithms

Mina Gabriel

13 Introduction

Binary search tree of height h can support any of the basic dynamic-set operations—such as SEARCH, PREDECESSOR, SUCCESSOR, MINIMUM, MAXIMUM, INSERT, and DELETE—in $O(h)$ time.
Thus, the set operations are fast if the height of the search tree is small.

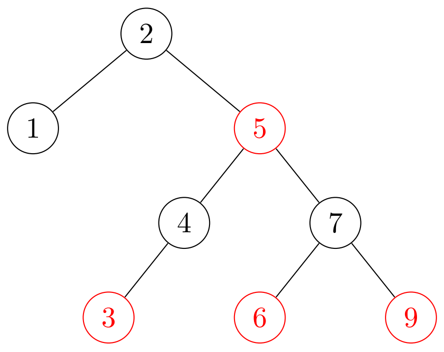
13.1 Properties of red-black trees

A red-black tree is a binary search tree with one extra bit of storage per node: its color, which can be either RED or BLACK.

Each node of the tree now contains the attributes *color, key, left, right, and p.*

A red-black tree is a binary tree that satisfies the following red-black properties:

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both its children are black.
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

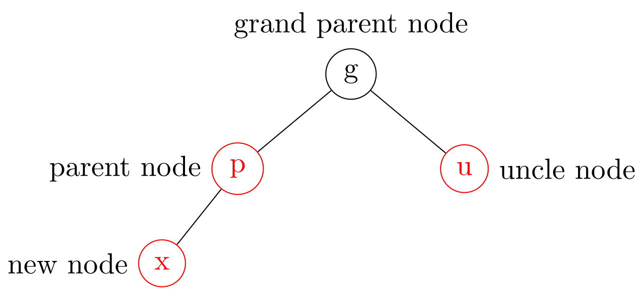


13.2 Details of the Insertion Process

In red-black tree we use two algorithms to do balancing:

- Recoloring.
- Rotation.

The color of the **Uncle** node -*sibling of parent node*- is important in determining the actions to be taken.



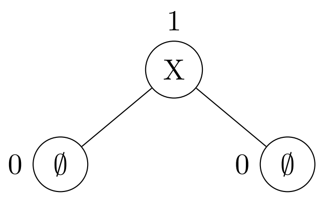
If uncle is red, we do recoloring. If uncle is black, we do rotations and/or recoloring.

13.3 Insertion Algorithms

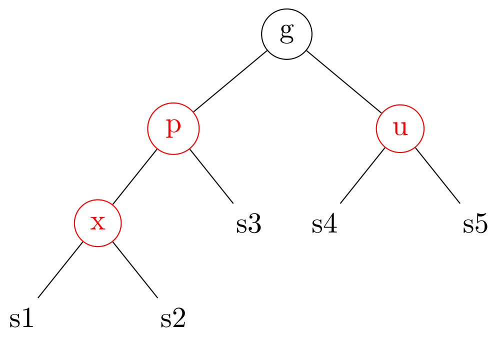
Perform a standard BST insertion and make the color of the newly inserted nodes as RED, Then:

13.3.1 CASE 0: X is root

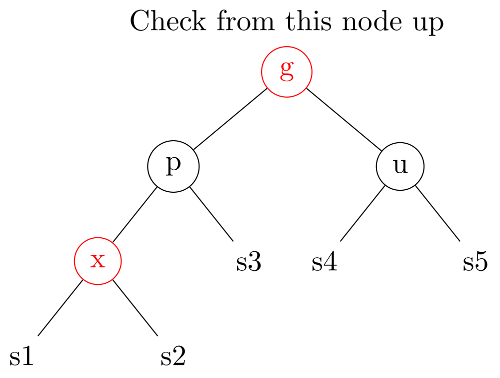
Change the color of X to Black, the height of the tree now increased by 1.



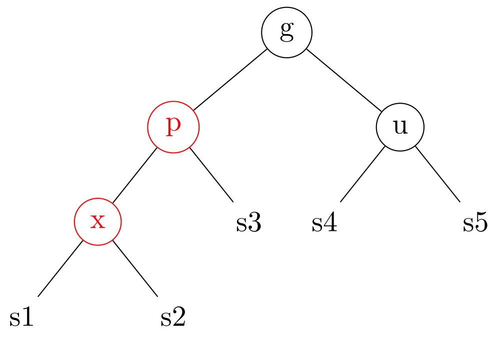
13.3.2 CASE 1: Both Parent and Parent Uncle of X are Red



- Change color of parent and uncle to Black.
- Change color of grand parent to Red.
- Recursively repeat the previous two steps with grand parent.

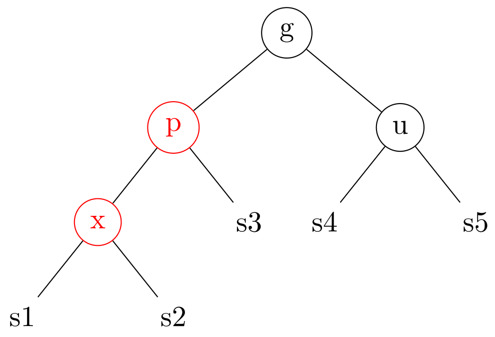


13.3.3 CASE 2: Parent is Red but uncle is Black

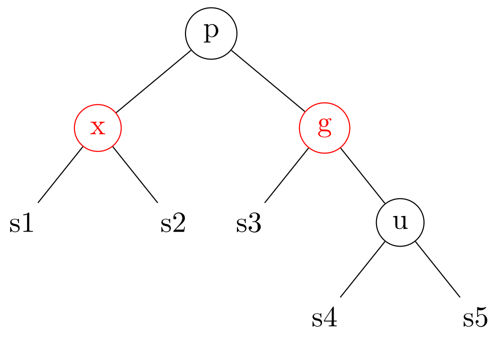


All four cases when Uncle is Black

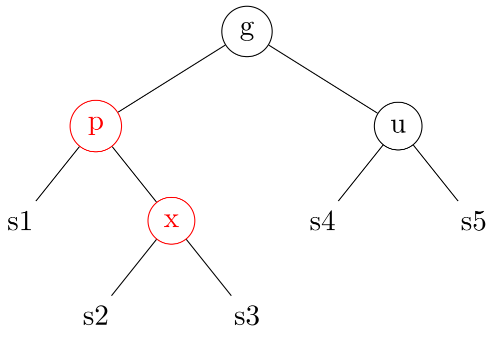
Case 2 A: Left Left Case



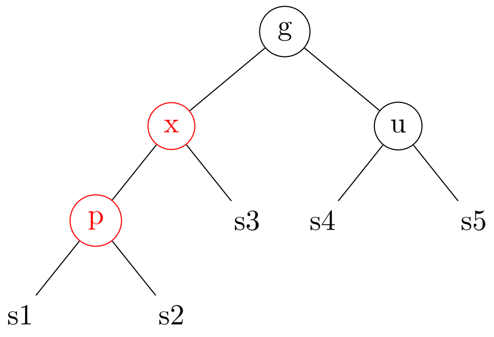
- Right Rotate g.
- Swap color of g and p.



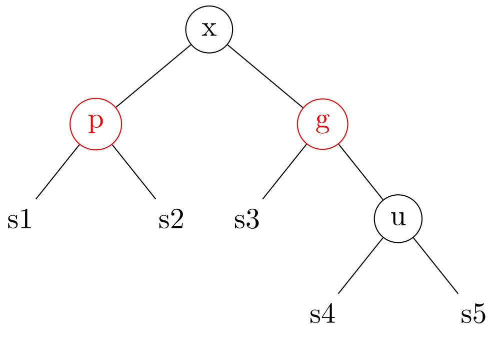
Case 2 B: Left Right Case



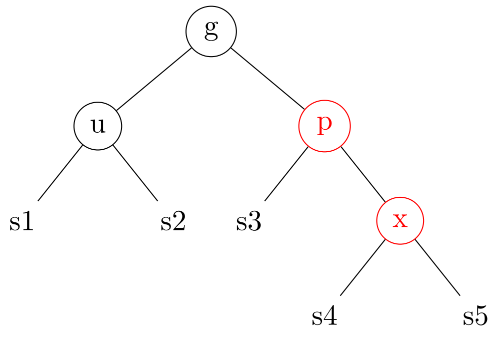
- Left Rotate p.



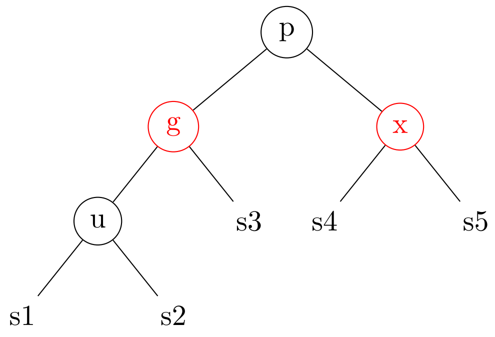
- Apply **Case 2 A: Left Left Case.**



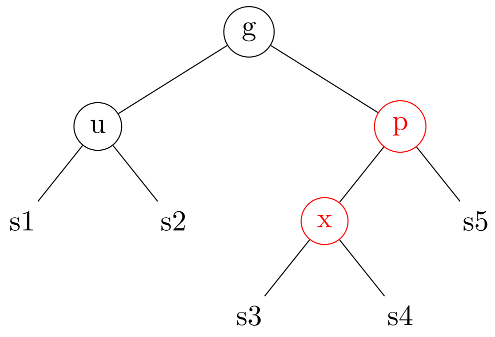
Case 2 C: Right Right Case



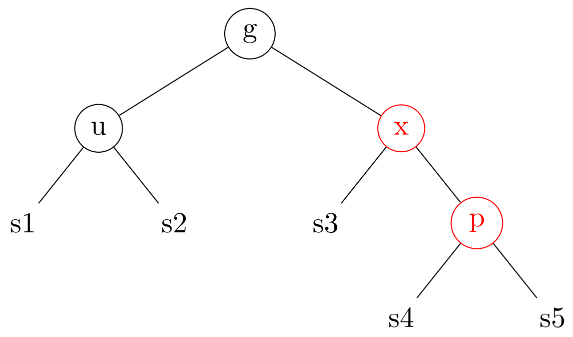
- Left rotate g
- Swap colors of g and p



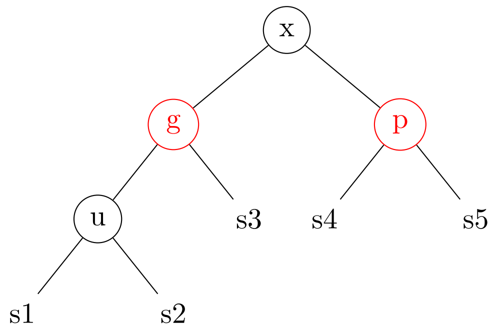
Case 2 D: Right Left Case



- Right rotate p



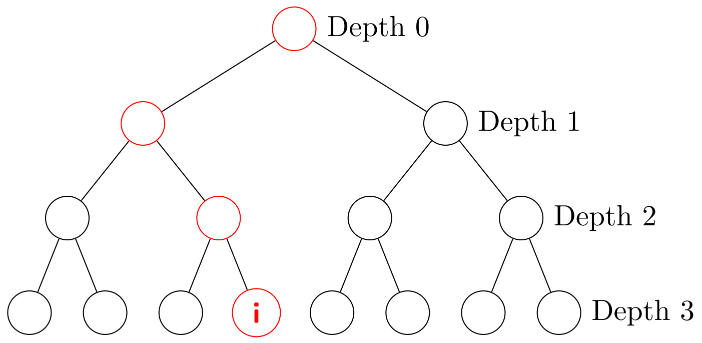
- Apply **Case 2 C: Right Right Case**



13.4 Red-Black Tree Height

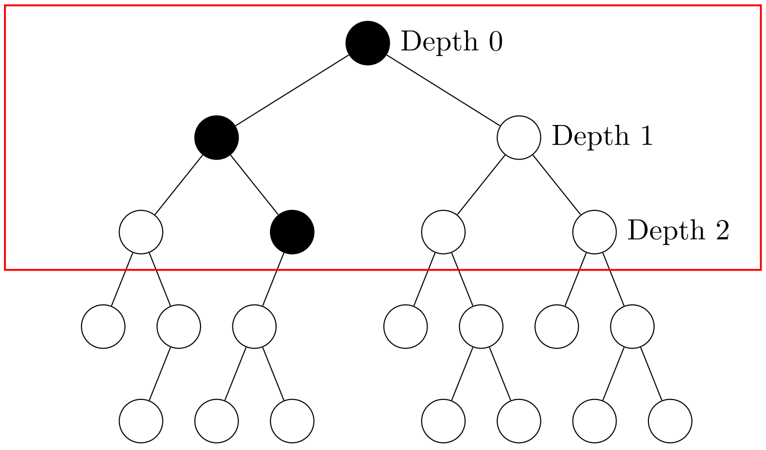
Lemma 13.1
A red-black tree with n internal nodes has height at most $2\,log_2(n+1)$

Proof
For any full and completer binary tree, we note the following:



- The number of node from the root to height k has $k+1$ nodes.
  - Example: the number of node from the root to node i is 4
- The relationship between the total number of node n and k can be defined with the following formula:
$$n = 2^{k+1} - 1$$
  - Example: for a tree with max-height equal 3 we must have $n = 15$

In a Red-Black tree we must have a full-complete binary sub-tree, at some level, the max-depth of this tree plus 1 is equal to the max number of black node we have in the tree.



The maximum number of nodes from the root to the leave nodes of the sub-tree is 3 or $k+1$

we need to find the formula of $k+1$, this formula will give us the max number of black nodes of a red-black tree, we know that:

$$n = 2^{k+1} - 1$$

Then

$$n + 1 = 2^{k+1}$$
$$k + 1 = log_2(n + 1)$$

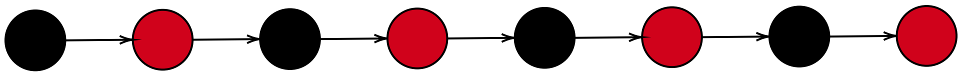
Therefor the max-number of black in any path in a red-black tree is $log_2(n + 1)$

We know from section 13.1 that:

For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

since we can't have any two consecutive red nodes, then the max number of allowed red node in any given nodes must be equal the number of black nodes.

for example if we know that a path has 4 black nodes, the max number if red nodes we can add in this path is 4, thus the max number of nodes in the path must not exceed 8.



$$\begin{aligned}\text{Maximum Height} &= \text{Maximum Black Nodes} + \text{Maximum Red Nodes}\\ &= log_2(n+1) + log_2(n+1)\\ &= 2 \times log_2(n+1)\end{aligned}$$