Lecture Notes

on

Red Black Tree

GALGOTIAS UNIVERSITY
(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

July 2020

(Be safe and stay at home)

Red Black Tree

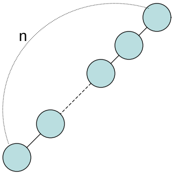
A balanced binary search tree

Review

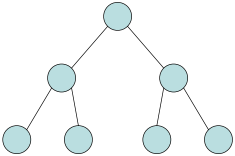
- Binary Search Tree (BST) is a good data structure for searching algorithm
- It supports
  - Search, find predecessor, find successor, find minimum, find maximum, insertion, deletion

Motivation

- The performance of BST is related to its height h
  – All the operation in the previous page is O(h)



Worst case: h = O(n)



Best case: h = O(log n)

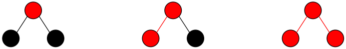
Motivation

- We want a balanced binary search tree
  - Height of the tree is O(log n)

- Red-Black Tree is one of the balanced binary search tree

Property

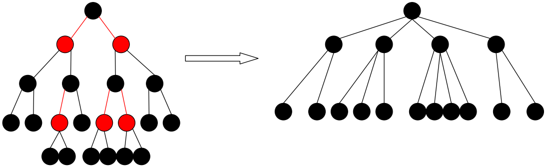
1. Every node is either red or black
2. The root is black
3. If a node is red, then both its children are black



4. For each node, all path from the node to descendant leaves contain the same number of black nodes
    - All path from the node have the same black height

Property

- Compact



Property

- The height of compacted tree is O(log n)

- Since no two red nodes are connected, the height of the original tree is at most 2 log n = O(log n)

Operation

- Since red-black tree is a balanced BST, it supports

  Search(tree, key)

  Predecessor(tree, key)

  Successor(tree, key)

  Minimum(tree)

  Maximum(tree)

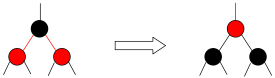
  in O(log n)-time
- It also support insertion and deletion with a little bit complicated step

Maintain Property

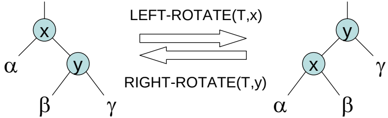
- Insertion and Deletion will violate the property of red-black tree
- How to maintain the property?
  - by Changing Color or Rotation

Maintain Property

- Changing color

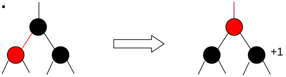


- Rotation



Common Problem

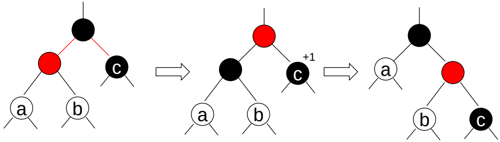
- A problem during Insertion and Deletion is **Doubly-Black** node
- Doubly-Black node is a node which has color of two black, it violate property 1
- For example:



(+1 means the node need another black to maintain the invariant of the property)

Common Problem

- A common problem and its solution are as following



Insertion

- When insert a node z, we set the color of z to red
- This may violate property 2 and 3

- For property 2, we set the color of root to black after insertion

Insertion

- To fix property 3, we will consider if
  - The z's parent is a left child or right child
  - The color of z's uncle y is red or black
  - z is a left child or right child
- We consider the z's parent is a left child first, the other case can be done by symmetric operation

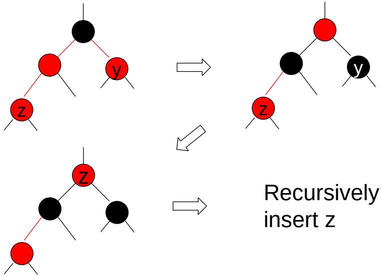
Insertion

There are 4 cases:

- Case 1: y is red and z is a left child
- Case 2: y is red and z is a right child
- Case 3: y is black and z is a left child
- Case 4: y is black and z is a right child

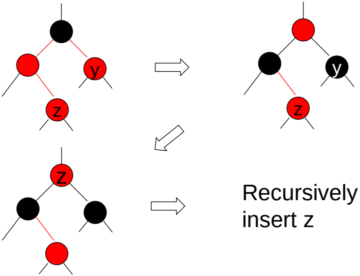
Insertion - Case 1

- Case 1: y is red and z is a left child



Insertion - Case 2

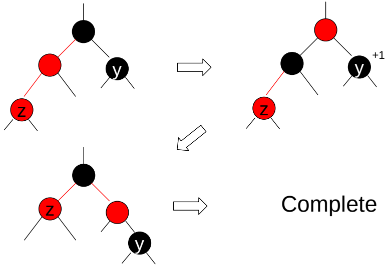
- Case 2: y is red and z is a right child



Recursively
insert z

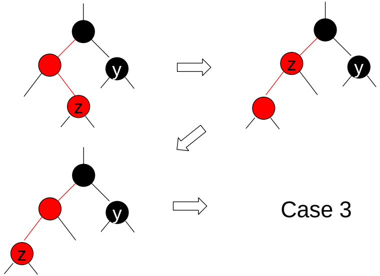
Insertion - Case 3

- Case 3: y is black and z is a left child



Insertion - Case 4

- Case 4: y is black and z is a right child



Insertion Analysis

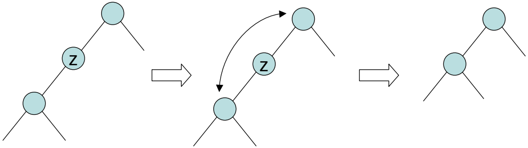
- Case 1 and 2 move z up 2 levels
- Case 3 and 4 will terminate after some number of steps
- The height of tree is finite and is O(log n)
- The running time is O(log n)
- At most 2 rotations

Deletion Review

- Review deletion of BST
- To delete a node z, there are 3 cases
- Case1: z has no child

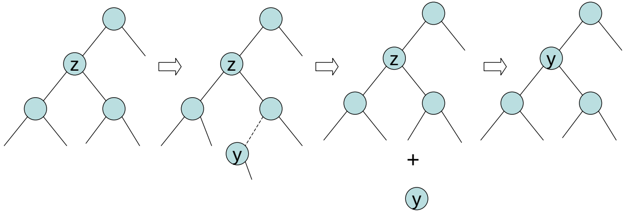
Deletion Review

- Case 2: z has one child



Deletion Review

- Case 3: z has two children



Deletion

- From now on, we always call the deleted node to be z
- If z is red, it won't violate any property
- If z is a leaf, it won't violate any property
- Otherwise z is black and has a child, it will violate property 2, 3, and 4
- For property 2, set the color of root to black after deletion

Deletion

To fix property 3 and 4:

- If z's child x (which is the replacing node) is red, set x to black. Done!
- If x is black, add another black to x, so that x will be a doubly black node, and property 3 and 4 are fixed. But property 1 is violated

Deletion

- To fix property 1, we will consider if
  - x is a left child or right child
  - The color of x's sibling w is red or black
  - The colors of w's children
- We consider x is a left child first, the other case can be done by symmetric operation

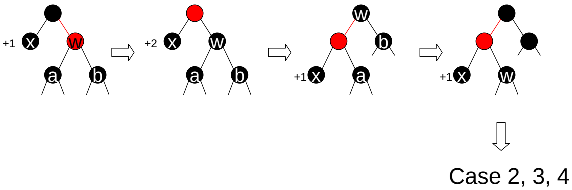
Deletion

There are 4 cases:

- Case 1: w is red
- Case 2: w is black, both w's children are black
- Case 3: w is black, w's left child is red, w's right child is black
- Case 4: w is black, w's right child is red

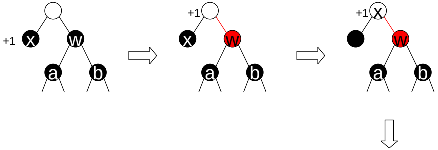
Deletion - Case 1

- Case 1: w is red



Deletion - Case 2

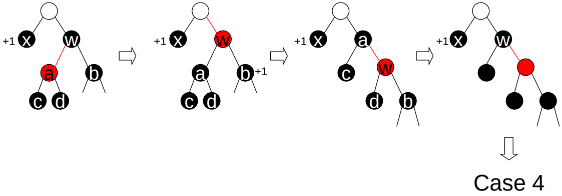
- Case 2: w is black, both w's children are black



Recursively delete x

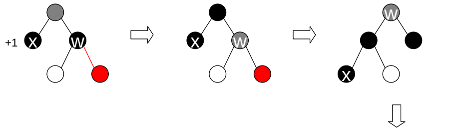
Deletion - Case 3

- Case 3: w is black, w's left child is red, w's right child is black



Deletion - Case 4

- Case 4: w is black, w's right child is red



Deletion Analysis

- Case 2 move x up 1 level
- Case 1, 3 and 4 will terminate after some number of steps
- The height of tree is finite and is O(log n)
- The running time is O(log n)
- At most 3 rotations

Conclusion

- Red-Black Tree is a balanced binary search tree which supports the operation search, find predecessor, find successor, find minimum, find maximum, insertion and deletion in O(log n)-time