Lecture 3

Instructor: Subrahmanyam Kalyanasundaram

13th August 2019

Last Class

We saw Binary Search Trees and the following:

Last Class

We saw Binary Search Trees and the following:

1. SEARCH
2. INSERT
3. SUCC (also PRED)

Last Class

We saw Binary Search Trees and the following:

1. Search
2. Insert
3. Succ (also Pred)

4. Today we see, Delete

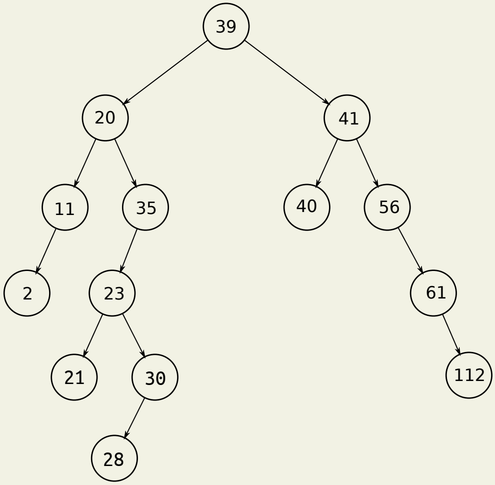
Successor

The SUCC(VAL) procedure is as follows:

- Find the node which stores *val*. Refer to this node as "*node*".
- Two cases:

Case 1: *node* has a right child.
Case 2: *node* does not have a right child.

Example



Delete

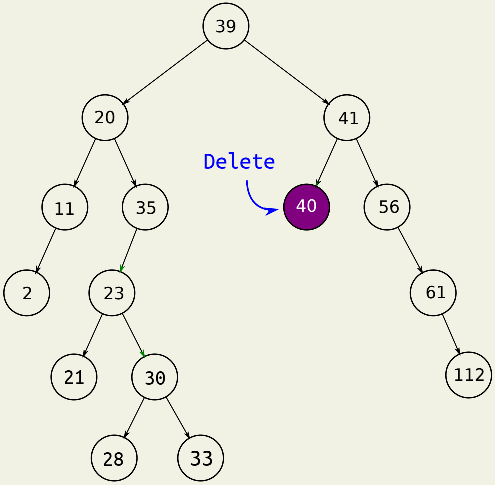
The Delete(val) procedure is as follows:
Find the node that has value *val*.

Delete

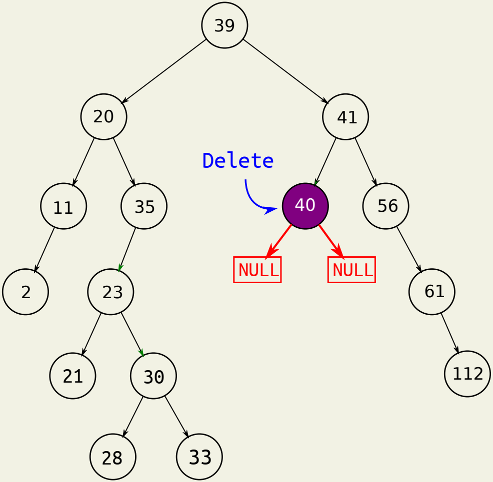
The Delete(val) procedure is as follows:
Find the node that has value *val*.
Three cases:

1. *node* has 0 children. (trivial)
2. *node* has 1 child. (splice)
3. *node* has 2 children:
    - Find successor node X with value x.
    - Splice X out of the tree.
    - Replace *val* with x.
    - Delete node X.

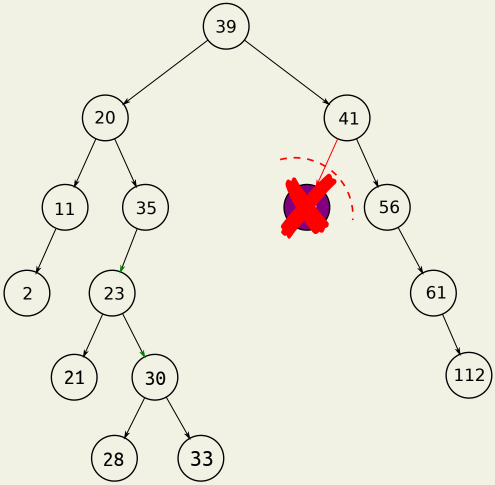
Example



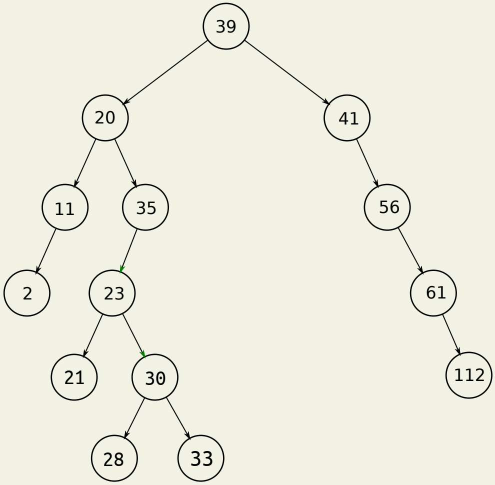
Example



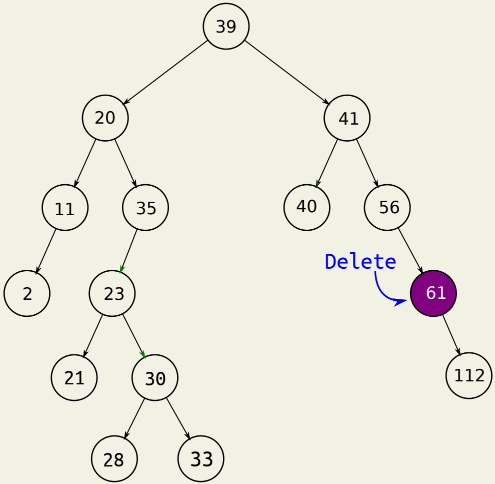
Example



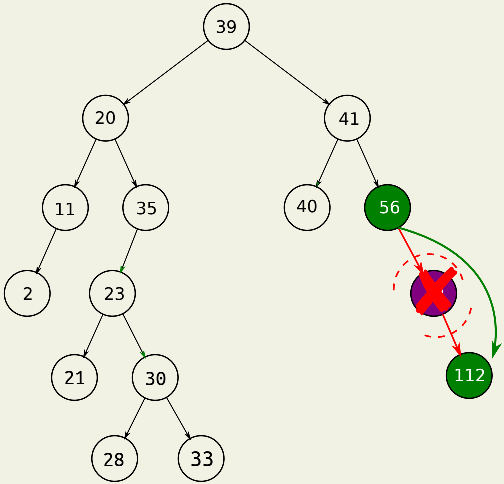
Example



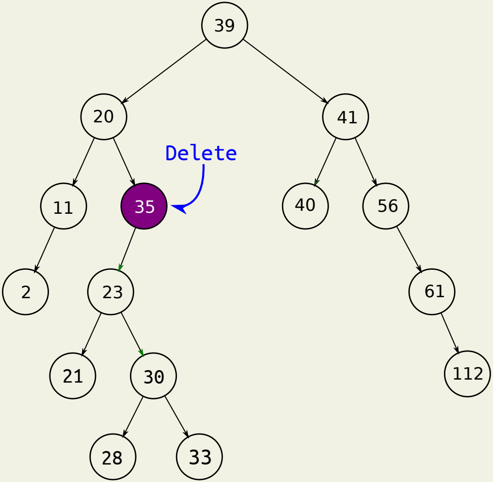
Example



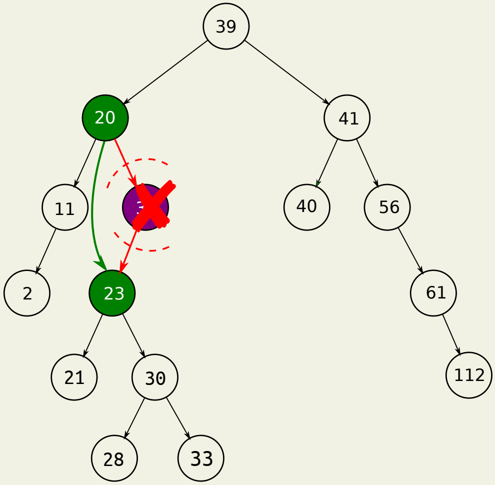
Example



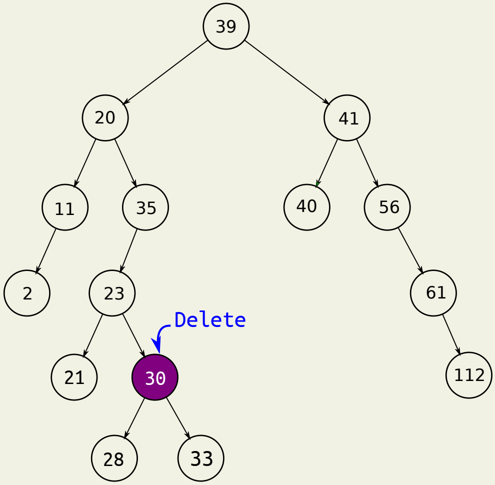
Example



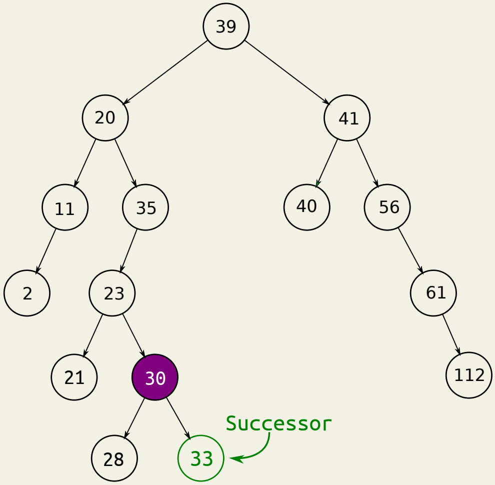
Example



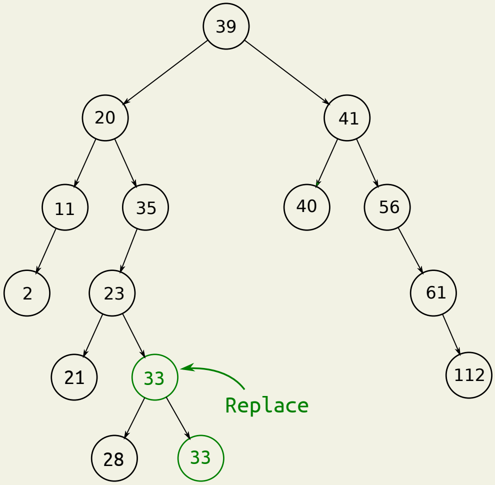
Example



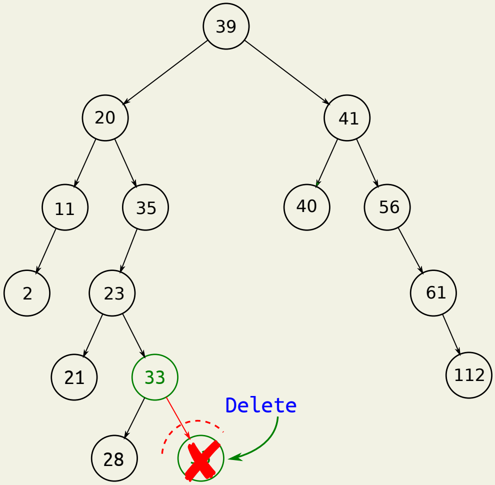
Example



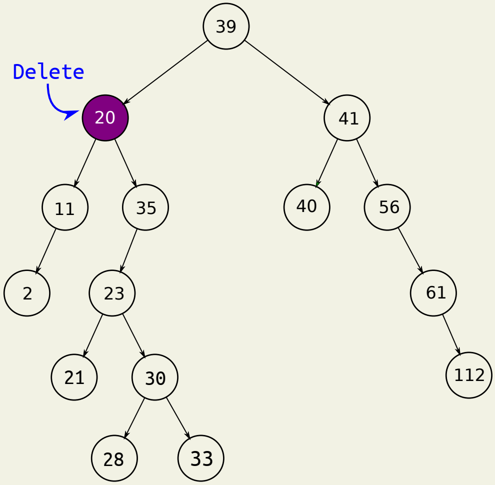
Example



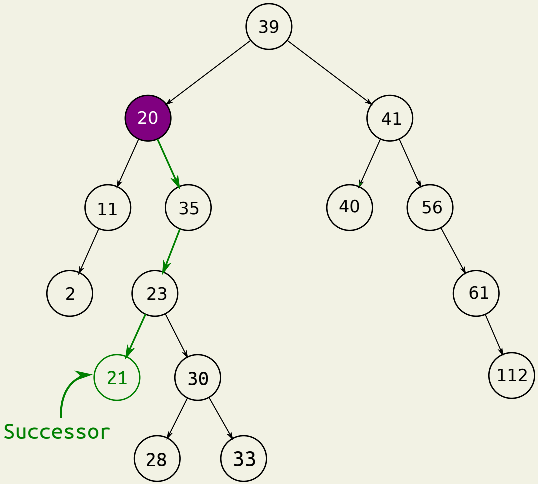
Example



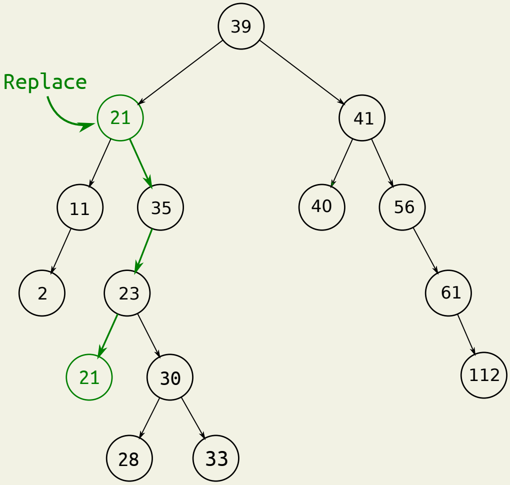
Example



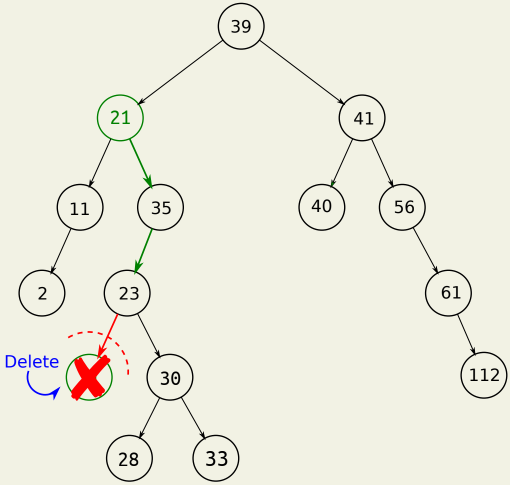
Example



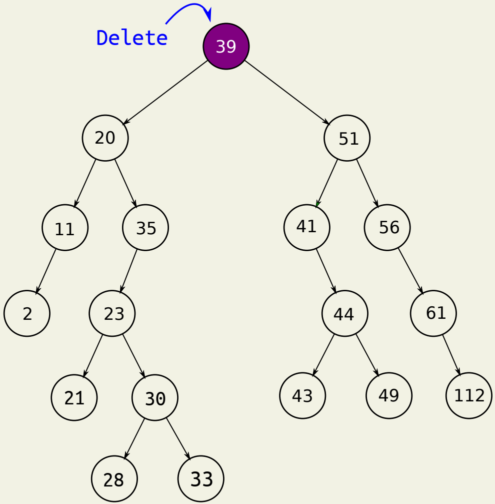
Example



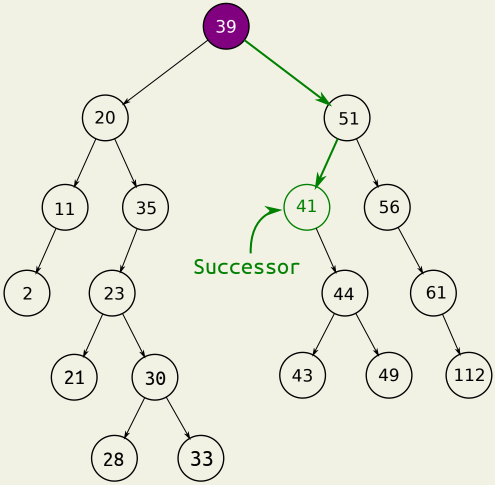
Example



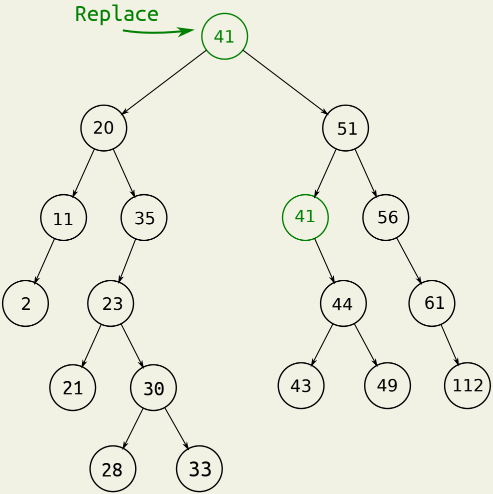
Example



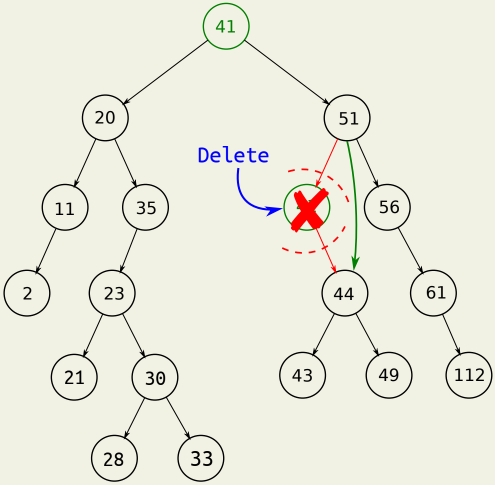
Example



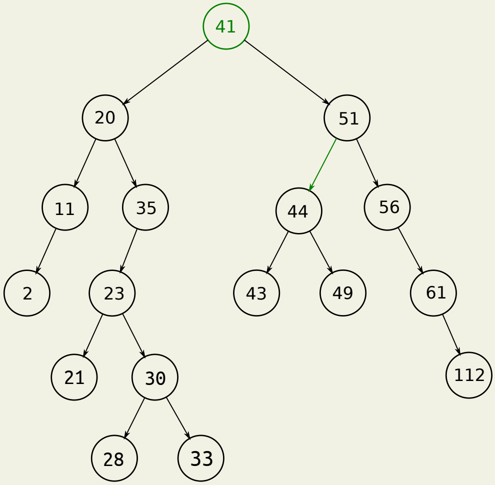
Example



Example



Example



Running Time

Worst case running times for a BST of height h:

- Insert – $O(h)$.
- Succ – $O(h)$.
- Search – $O(h)$.
- Delete – $O(h)$.

The height of a BST depends on the input sequence and can be n after inserting n elements in the worst case.

Balancing a BST

The biggest drawback of BSTs are that they can be quite “unbalanced”.

One way to measure if a tree is balanced is to look at the difference between the longest path from root to leaf and the shortest path from root to leaf.

Balancing a BST

The biggest drawback of BSTs are that they can be quite “unbalanced”.

One way to measure if a tree is balanced is to look at the difference between the longest path from root to leaf and the shortest path from root to leaf.

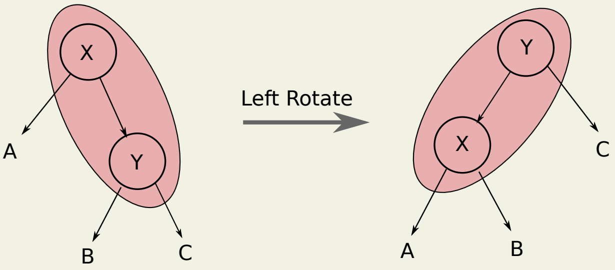
We want to make sure this difference does not get too large.

Balancing a BST

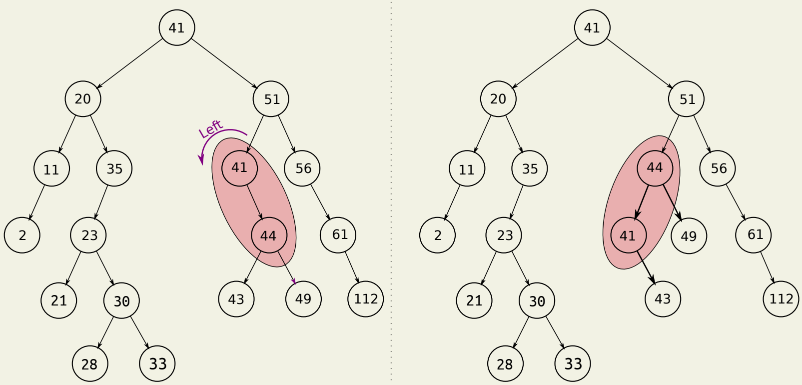
Balancing a BST is done by making structural changes to the underlying tree.
Rotations are operations on nodes of a BST. They are of two variants:

1. Left Rotate.
2. Right Rotate.

Left Rotate



Example



41
20
51
Left
11
35
41
56
2
23
44
61
21
30
43
49
112
28
33
41
20
51
11
35
44
56
2
23
41
49
61
21
30
43
112
28
33

Set

maintains a set of elements from the universe

A set has the following functions:

- INSERT(x) – Insert x into the set.
- SEARCH(x) – Return True if x is an element of the set.
- SUCC(x) – Returns the smallest value larger than x.
- PRED(x) – Returns the largest value smaller than x.
- GETMAX() – Returns the largest value in the set.
- GETMIN() – Returns the smallest value in the set.
- ISEMPTY() – Returns True if and only if the set is empty.
- DELETE(x) – Remove x from the set.

Data Structures for Set

Many choices of data structure to implement set:

- Array
- Sorted Array
- Heap
- Binary Search Tree
- Balanced Binary Search Trees

Data Structures for Set

Many choices of data structure to implement set:

- Array
- Sorted Array
- Heap
- Binary Search Tree
- Balanced Binary Search Trees

We now study a balanced Binary Search Tree called Red-Black Trees.

Red-Black Trees

Red-Black Trees (RBT) are Binary Search Trees that balance themselves!

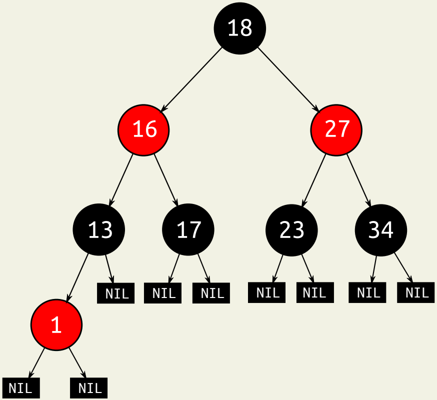
Red-Black Trees

Red-Black Trees (RBT) are Binary Search Trees that balance themselves! RBTs have the following properties:

1. All nodes are colored either Red or Black.
2. The root node is black.
3. The leaf nodes (NIL) are black.
4. Both children of a red node are black.
5. For any node, all paths from the node to the descendant leaves have the same number of black nodes.

Example

1. Every node is colored either Red or Black.
2. The root node is black.
3. The leaf nodes (NIL) are black.
4. Both children of a red node are colored black.
5. For any node, all paths from the node to the descendant leaves have the same number of black nodes.



A Red-Black Tree supports all procedures of a BST:

- INSERT(*val*) – Inserts *val* into the RBT rooted at *node*.
- SEARCH(*val*) – Returns True of *val* exists in the BST rooted at *node*. False otherwise.
- SUCC(*val*) – Returns the smallest element greater than *val* in the RBT.
- PRED(*val*) – Returns the largest element lesser than *val* in the RBT.
- DELETE(*val*) – Deletes *val* from the RBT.

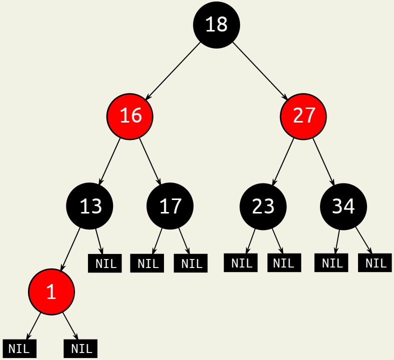
A Red-Black Tree supports all procedures of a BST:

- INSERT(*val*) – Inserts *val* into the RBT rooted at *node*.
- SEARCH(*val*) – Returns True of *val* exists in the BST rooted at *node*. False otherwise.
- SUCC(*val*) – Returns the smallest element greater than *val* in the RBT.
- PRED(*val*) – Returns the largest element lesser than *val* in the RBT.
- DELETE(*val*) – Deletes *val* from the RBT.

The procedures in green are implemented exactly like in a BST.

Black-Height

The black-height of a node X is the number of black colored nodes encountered on a path starting from X to any leaf (excluding X itself).



The black-height of the node with value 13 is 1.
The black-height of the root node is 2.
The black height of an red-black tree is the black height of its root.

Observations

Claim

A red-black tree with black-height β has height at most 2β.

Claim

A red-black tree with black-height β has height at most 2β.

Proof sketch:

- Try to construct the longest possible path with at most β many black nodes.
- Property 4 will force you to color every alternate node black.

Observations

Theorem

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

Observations

Theorem

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

Proof sketch

- Show that for any node X with black-height β, the number of *internal* nodes in the subtree rooted at X is at least $2^\beta - 1$.
- Conclude that with n internal nodes, the black-height must be at most $\log(n+1)$.
- Use previous claim that height is at most twice the black-height to conclude the Theorem.