eecs 281 Data Structures and Algorithms

Discussion 6: Week of Oct 10, 2011

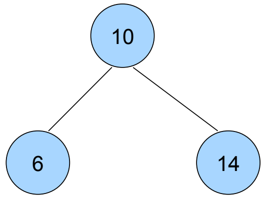
Summary

- 2-3 Trees
- 2-3-4 Trees
- Red-black Trees

2-3 Trees

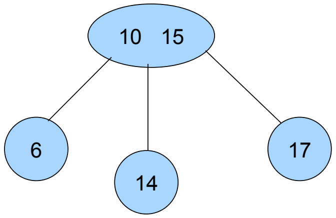
- Balanced tree
- Not a binary tree
- Each node has either
  - 1 value and 2 children
  - 2 values and 3 children

2 Nodes



- All elements to the left are less than top element
- All elements to the right are greater than top element

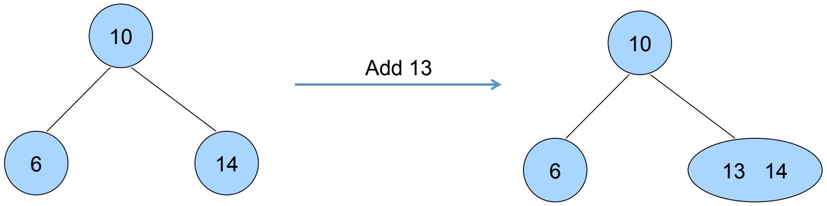
3 Nodes



- All elements to the left are less than first element
- All elements in the middle are between the two top elements
- All elements to the right are greater than second element

Inserting Elements

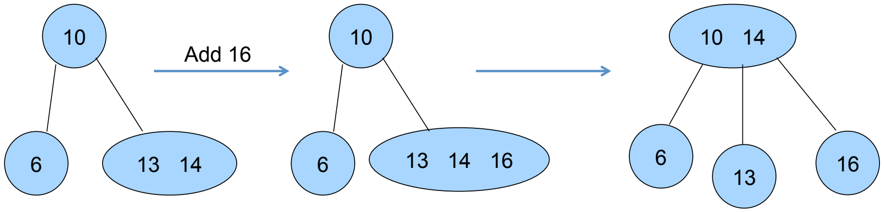
- Find leaf node to insert into.
- If leaf node is a 2-node, just add element



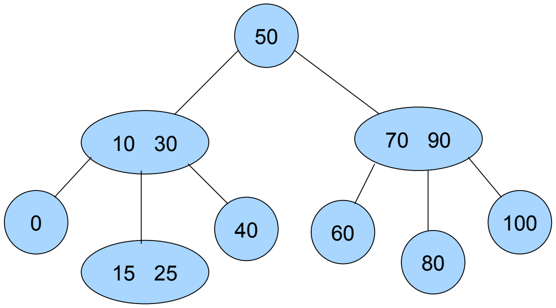
Inserting Elements

- If leaf node to insert into is a 3-node, must split that node.
- Middle element moves to parent
- Left element becomes middle child of parent
- Right element becomes right child of parent

Example

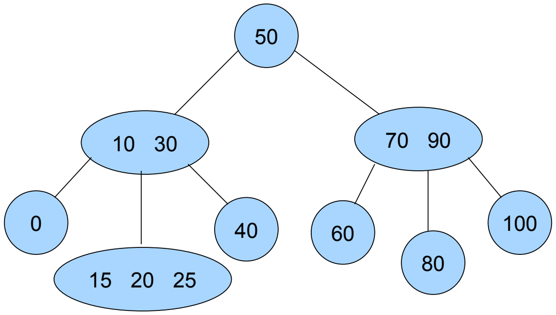


A more complicated example



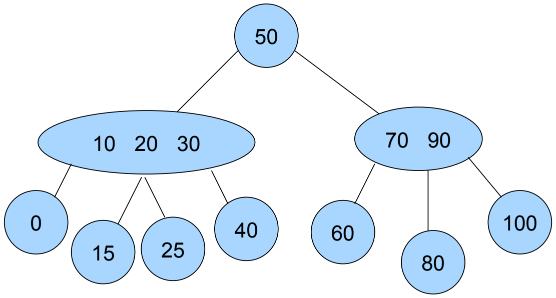
- Insert 20

A more complicated example



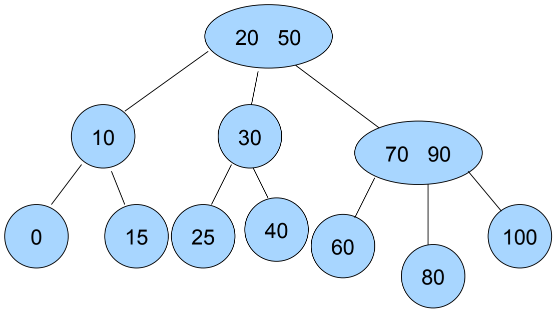
- Insert 20

A more complicated example



- Insert 20
- Move 20 to parent

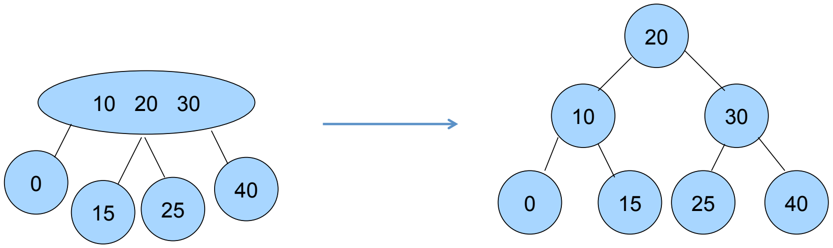
A more complicated example



- Insert 20
- Move 20 to parent
- Move 20 to parent again
- Must split the children

Splitting Root

- What if the root has 3 elements?
- Make a new root

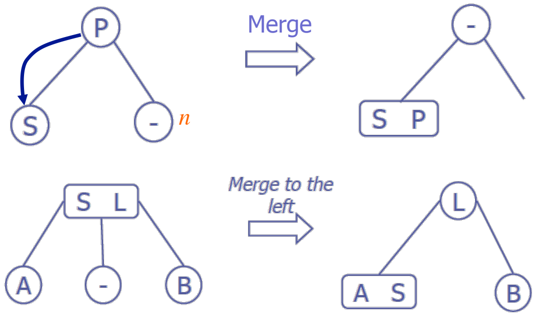


2-3 Tree Removal

- Easiest case:
  – Removing an element from a leaf node that is a 3-node. Just remove element
  – If leaf node you're removing from is a 2-node. We need to merge or rotate. Removing an element from a leaf node that is a 3-node is easy. Just remove element
- If node is not a leaf node
  – Swap element with next biggest element (in-order successor) and remove from leaf node.

Merging

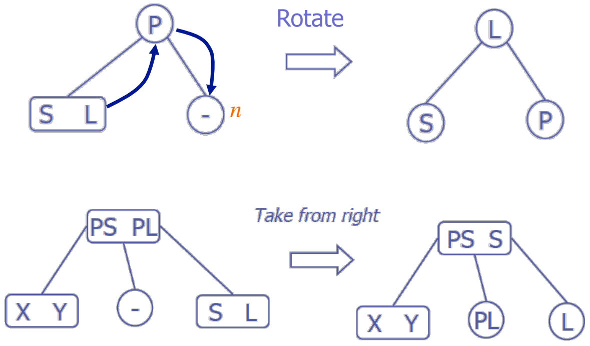
- Merge if sibling is a 2-node
- Merge elements from parent to child
- May leave parent node empty



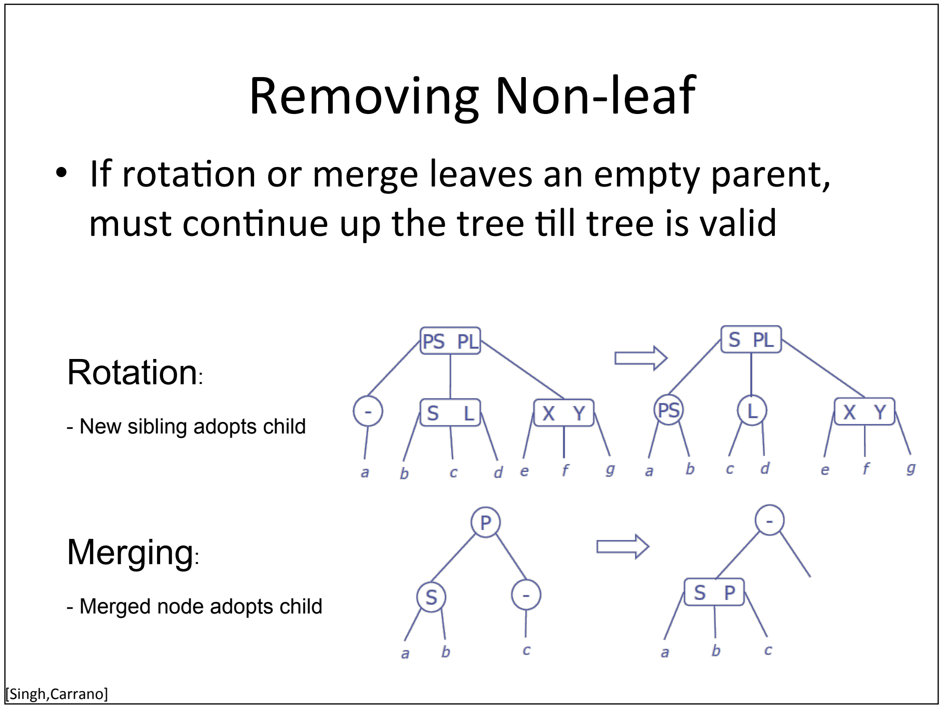
[Singh,Carrano]

Rotating

- Rotate if sibling is a 3-node
- Redistribute elements between sibling and parent
- Take elements from right sibling when parent is a 3-node



[Singh,Carrano]



Removing Non-leaf

- If rotation or merge leaves an empty parent, must continue up the tree till tree is valid

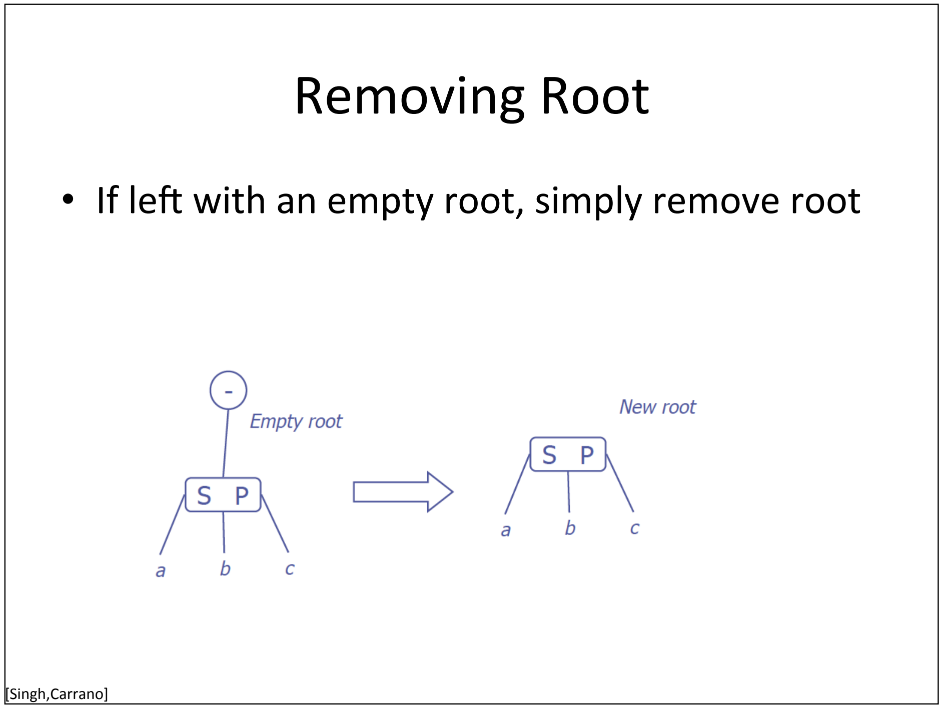
Rotation:

- New sibling adopts child

Merging:

- Merged node adopts child

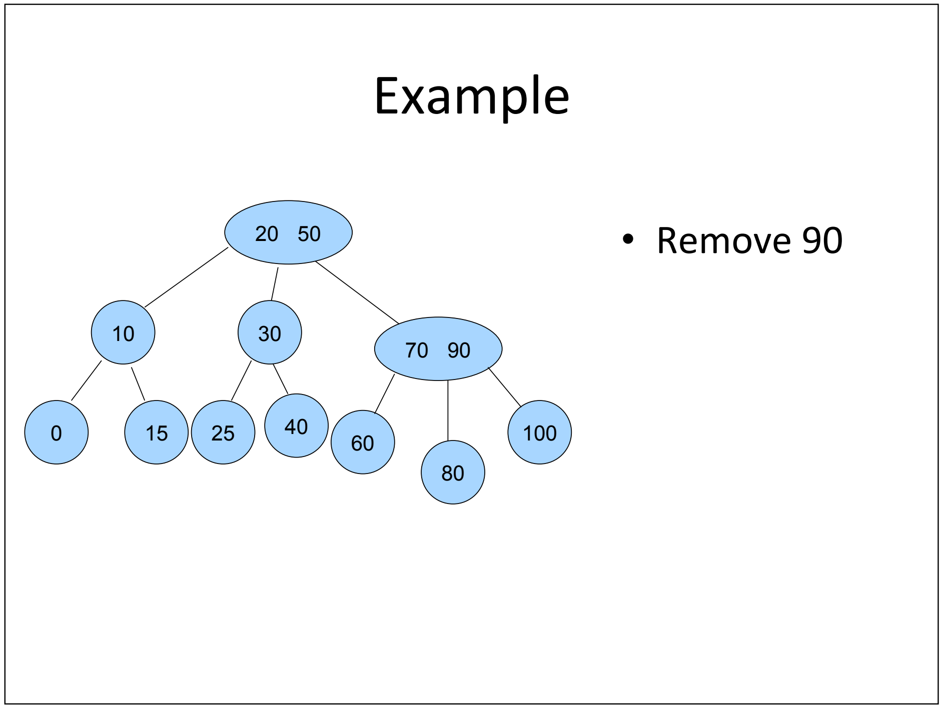
[Singh,Carrano]



Removing Root

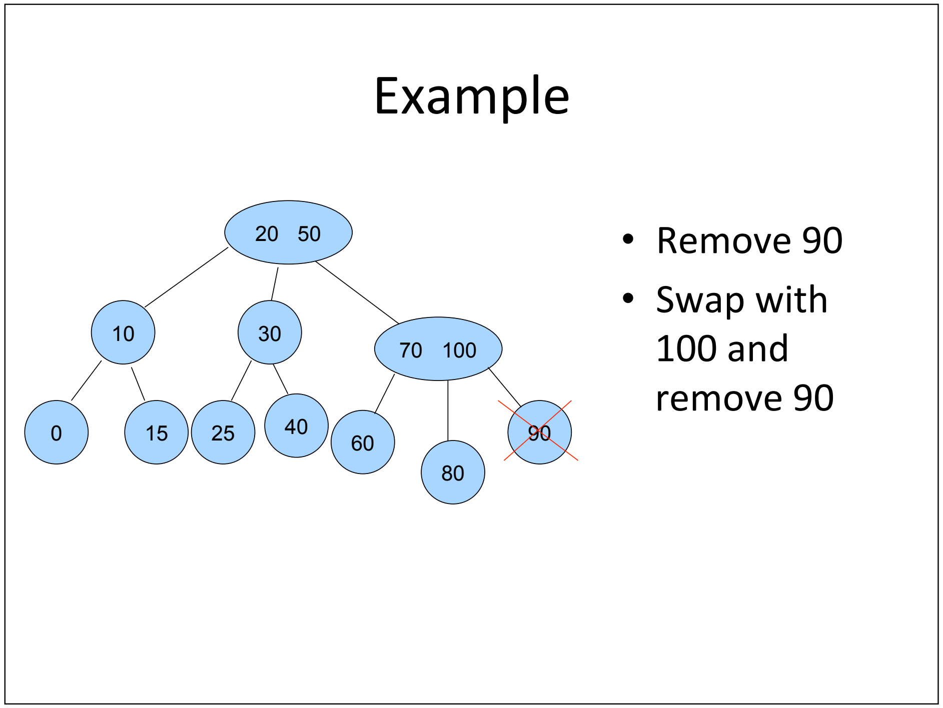
- If left with an empty root, simply remove root

[Singh,Carrano]



Example

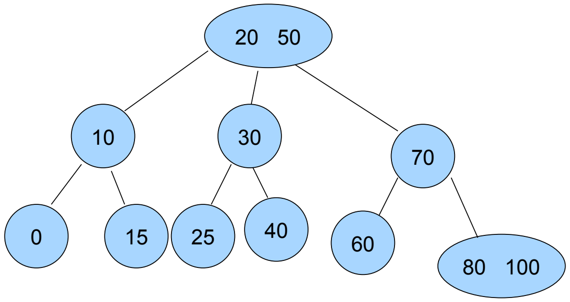
- Remove 90



Example

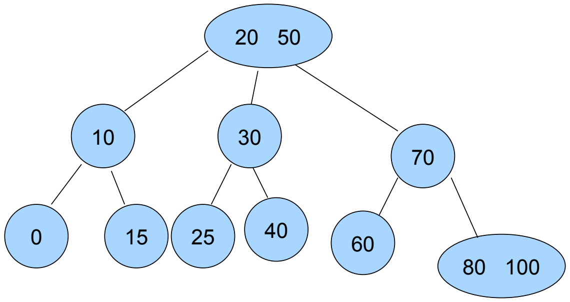
- Remove 90
- Swap with 100 and remove 90

Example



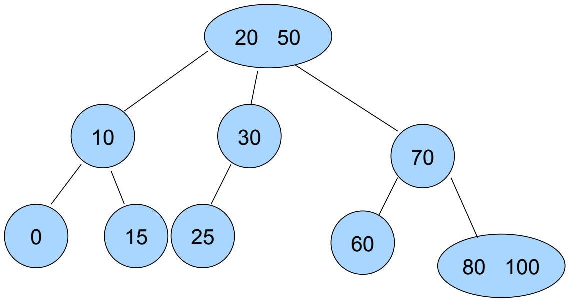
- Remove 90
- Swap with 100 and remove 90
- Merge

Example



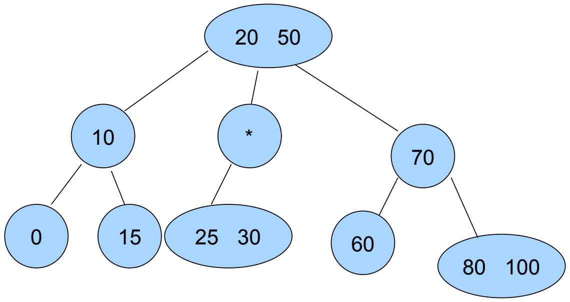
- Remove 40

Example



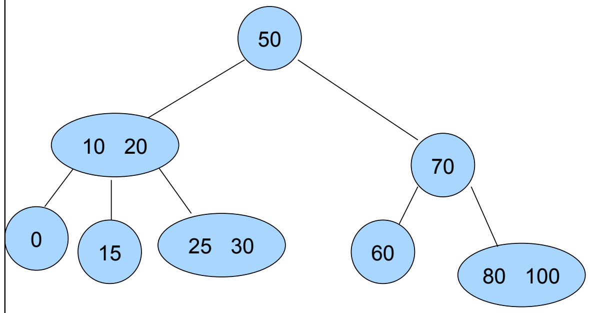
- Remove 40
- Merge 25 and 30

Example



- Remove 40
- Merge 25 and 30
- Merge again

Example



- Remove 40
- Merge 25 and 30
- Merge again

2-3-4 Trees

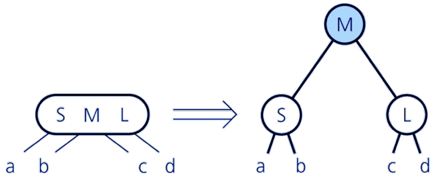
- Similar to 2-3 Trees
- Nodes can have
  - 2 elements, 1 child
  - 3 elements, 2 children
  - 4 elements, 3 children

Insertion

- Items inserted at leaf node
- 4 nodes are split early because they cannot hold another element
- On the way from the root to the leaf split 4-node that are visited
  - Insertion can be done in a single pass

Splitting a 4 node

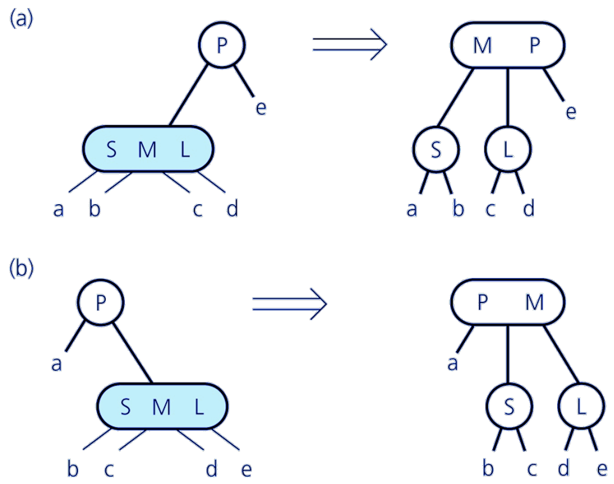
Without parent



[Carrano]

Splitting a 4 node

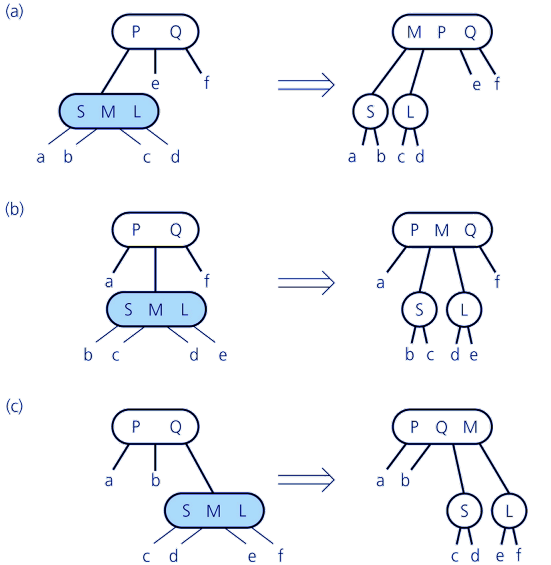
If parent is a 2-node



[Carrano]

Splitting a 4 node

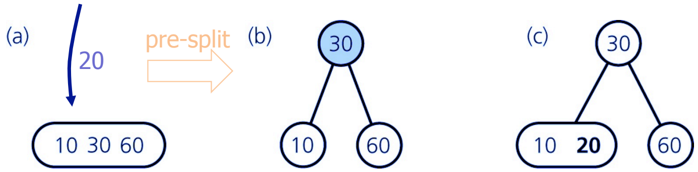
If parent is a 3-node



[Carrano]

Example

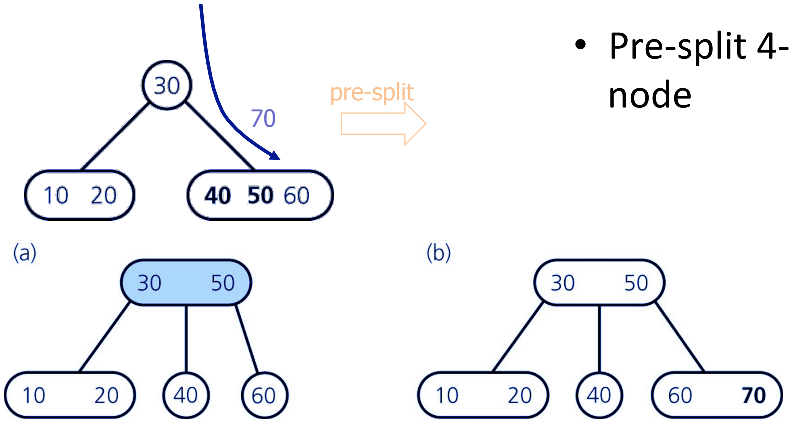
- Insert 20
- Pre-split root



[Carrano]

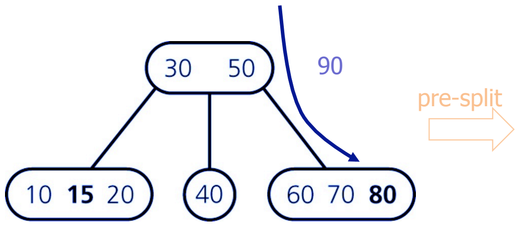
Example

- Insert 70
- Pre-split 4-node

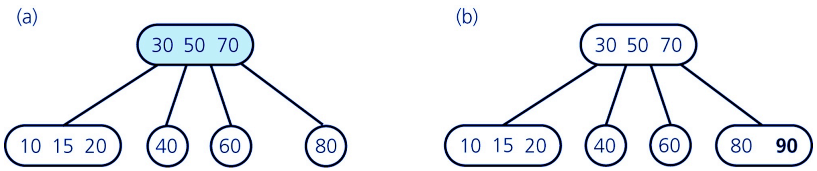


[Carrano]

Example



- Insert 90
- Pre-split 4-node

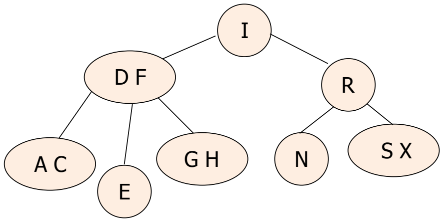


[Carrano]

Removal

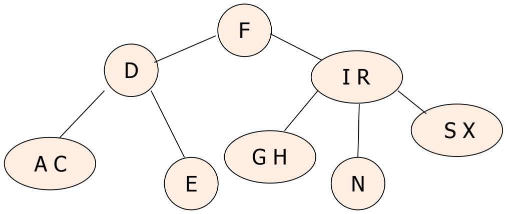
- Just like 2-3, if node is non-leaf, swap with in order successor
- Preemptively turn 2-nodes in 3-nodes
  - This way deletion can be done in one pass
  - **Rotate** if sibling is not a 2-node
  - **Merge** if sibling is a 2-node

Rotate



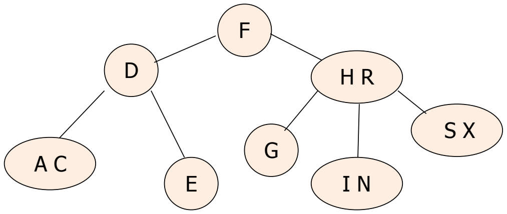
- Remove N
- Rotate so R is not a 2-node

Rotate



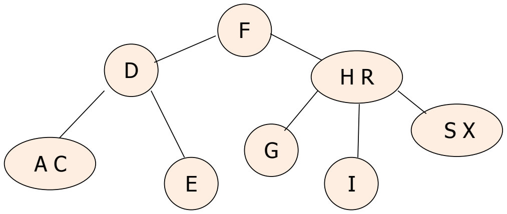
- Remove N
- Rotate so R is not a 2-node
- Rotate so N is not a 2-node

Rotate



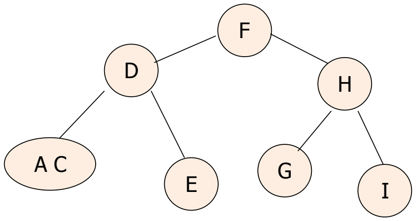
- Remove N
- Rotate so R is not a 2-node
- Rotate so N is not a 2-node
- Remove N

Rotate



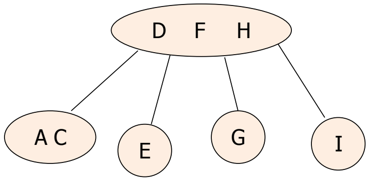
- Remove N
- Rotate so R is not a 2-node
- Rotate so N is not a 2-node
- Remove N

Merge



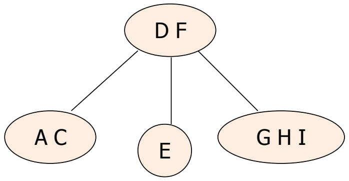
- Remove I
- Merge so H is not a 2-node

Merge



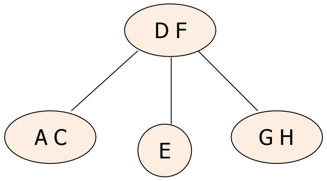
- Remove I
- Merge so H is not a 2-node
- Merge so I is not a 2-node

Merge



- Remove I
- Merge so H is not a 2-node
- Merge so I is not a 2-node
- Remove I

Merge



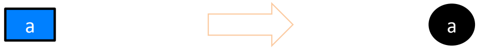
- Remove I
- Merge so H is not a 2-node
- Merge so I is not a 2-node
- Remove I

Red-Black Trees

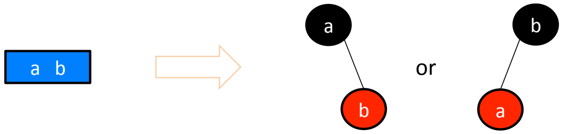
- Converts 2-4 trees into binary trees
- Red-Black Trees are BSTs where every node is colored red or black

Converting from 2-4 to red-black

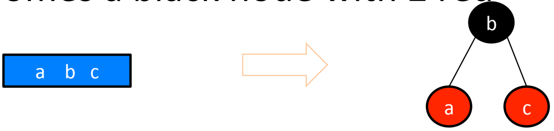
- 2 Node becomes a black node



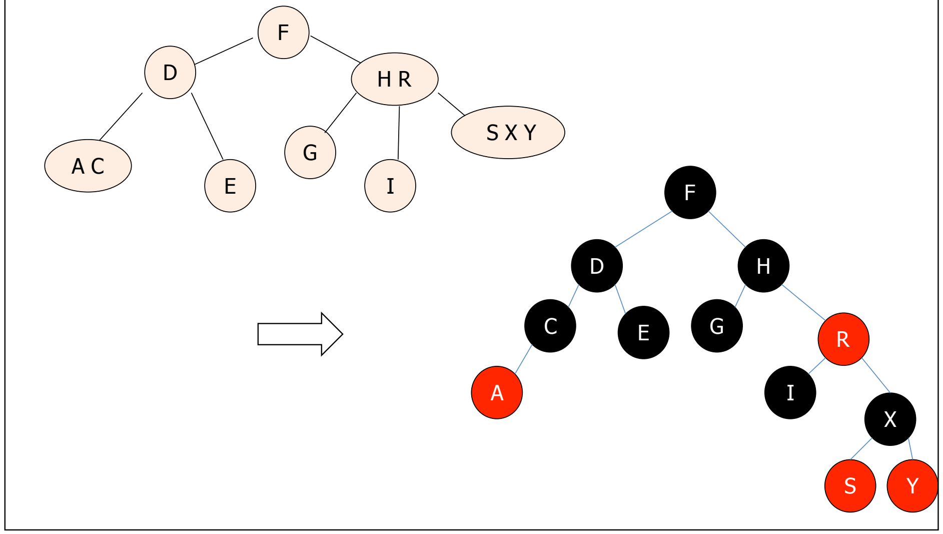
- 3 Node becomes a black node with one red child



- 4 Node becomes a black node with 2 red children



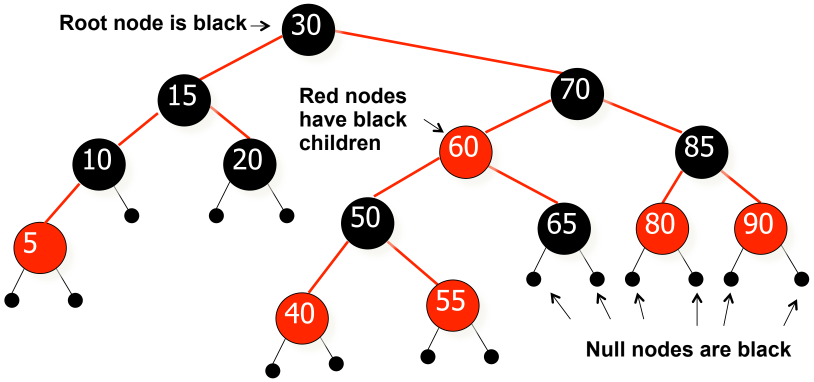
Converting from 2-4 to red-black



Red-Black Properties

- Every node is either red or black
- The root is black
- External Nodes (nulls) are black
- If a node is red, both children are black
- Every path from a node to a null has the same number of black nodes (the black height)

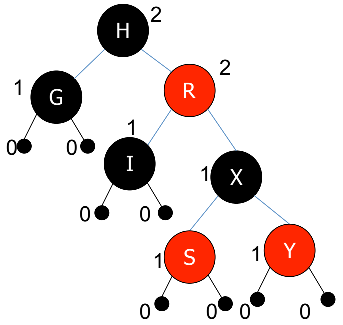
Example Red-Black Tree



- Every node is either red or black
- Each path from root to null have the same number of black nodes.

Black Height

- Black-height of node x is the number of black nodes on the path from x to an external node (including the external node but not counting x itself)
- If x is an external node
  - bh(x) = 0



Red-Black Tree Height

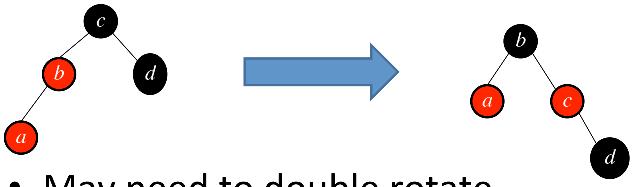
- The height of a red-black tree with *n* internal nodes is between $\log_2(n+1)$ and $2\log_2(n+1)$
- Height is constrained to O(log*n*)

Insertion – Bottom Up

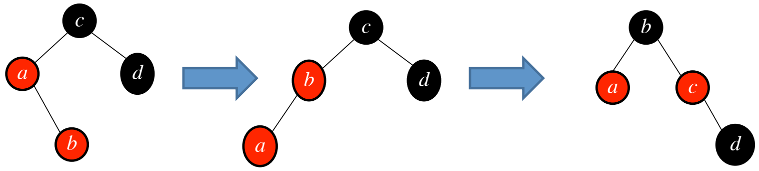
- New nodes are inserted as leaf nodes
- Must insert red node, inserting black violates black height rule
- If parent is black, done.
- If parent is red, violates "Red must have two black children" rule.

Insertion

- If sibling of parent is black, rotate.

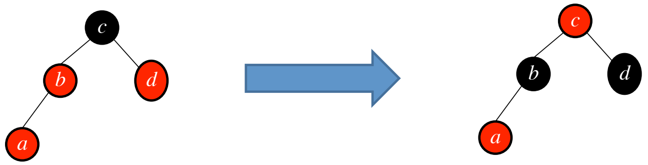


- May need to double rotate



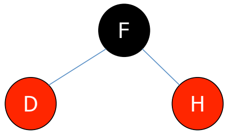
Recoloring

- If sibling of parent is red, recolor.



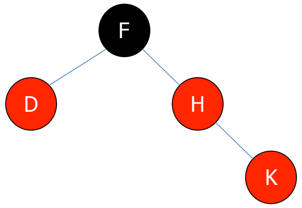
- Now that c is red, may cause double red again
  - Fix that double red the same way

Example



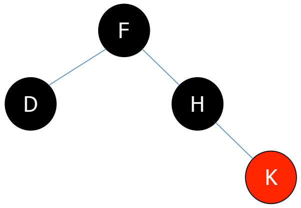
- Insert K

Example



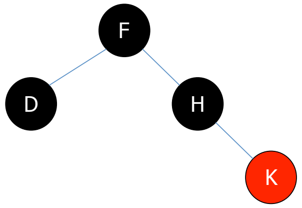
- Insert K
- Fix double red by recoloring

Example



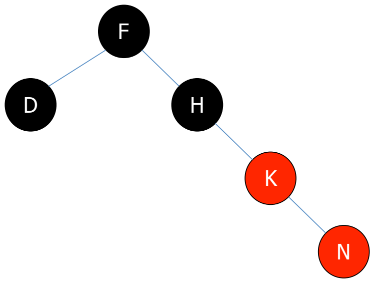
- Insert K
- Fix double red by recoloring
- Root must remain black

Example



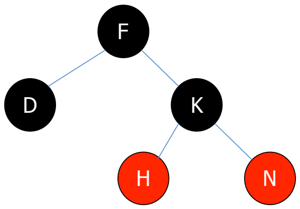
- Insert N

Example



- Insert N
- Fix double red by rotating

Example



- Insert N
- Fix double red by rotating

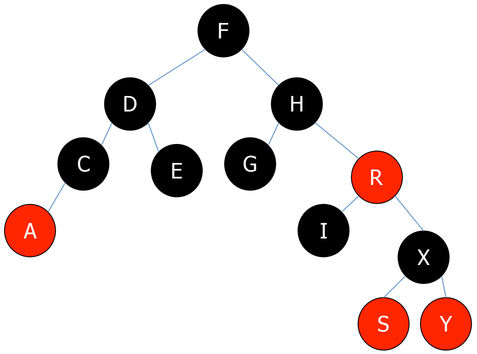
Removal

- Either removes a red or a black node
- If red, doesn’t violate any rules
- If black, could potentially violate rules

Removal

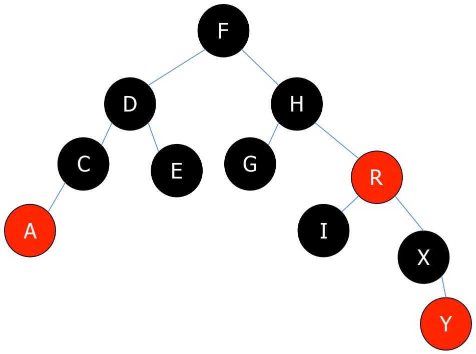
- If removing red leaf, just remove and you’re done
- If it is a single child parent, must be black. Delete, and recolor it’s child (which must be red) black.
- If the node has two children, swap node with in order successor
  - If in-order successor is red, remove it and you’re done
  - If in-order is a single child parent, apply previous rule

Example



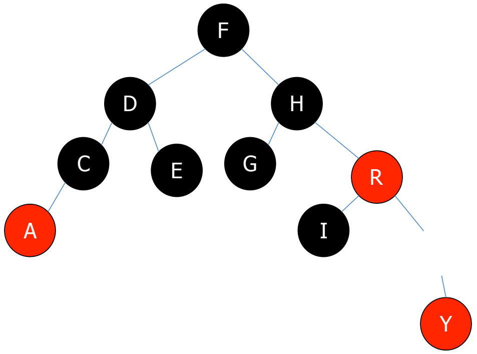
- Remove S

Example



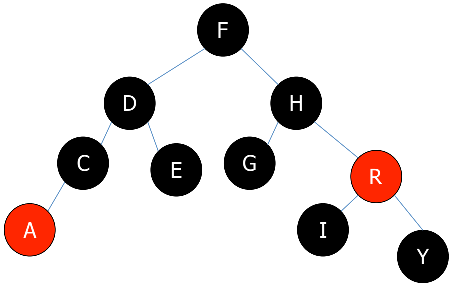
- Remove X

Example



- Remove X
- Delete it

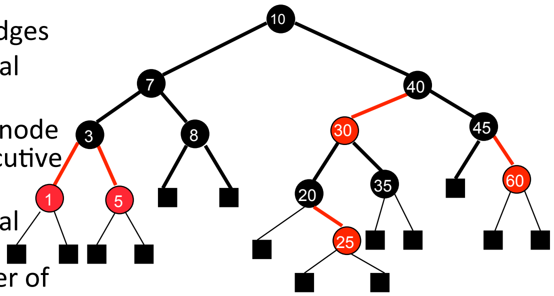
Example



- Remove X
- Delete it
- Recolor child black

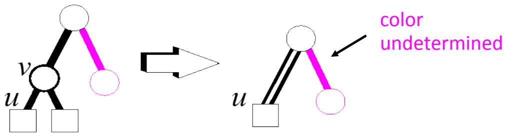
Colored Edges

- Colored edges definition
  - child pointers are colored red or black
  - the root has black edges
  - pointer to an external node is black
  - no root-to-external-node path has two consecutive red edges
  - every root to external node path has the same number of black edges



Black-Leaf Removal

- To remove black leaf, replace the node with an external node and color the edge “double black”

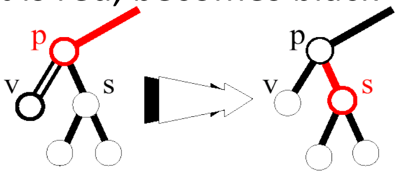


- To eliminate double black edge:
  - If there is a red edge nearby, turn that black.
  - Also can rotate or recolor

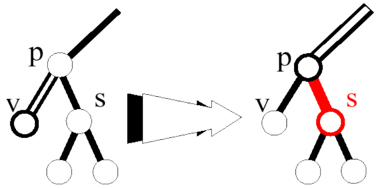
[Šaltenis]

Black Sibling with black nephew

- Sibling becomes red
- If parent is red, becomes black



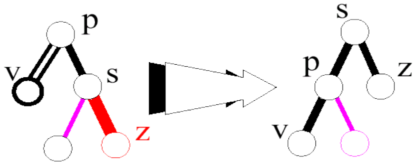
- If black, becomes double black



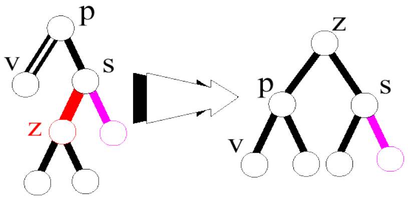
[Šaltenis]

Black Sibling with red nephew

- Rotate and recolor red nephew

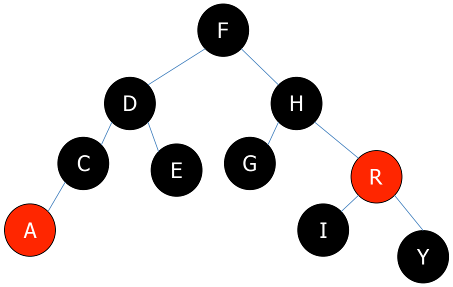


- May need to double rotate



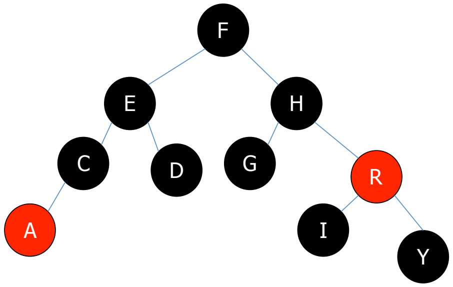
[Šaltenis]

Example



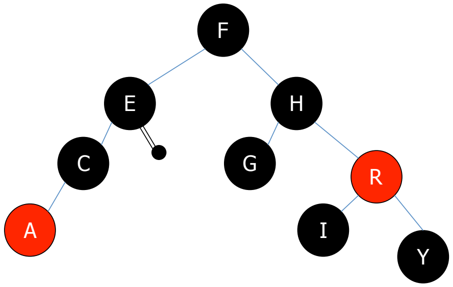
- Remove D
- Swap D & E

Example



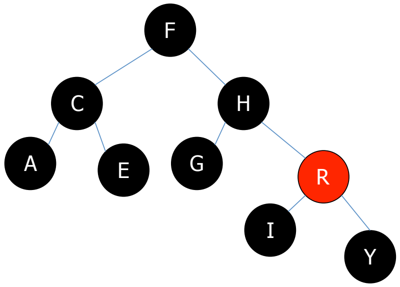
- Remove D
- Swap D & E
- Delete D

Example



- Remove D
- Swap D & E
- Delete D
- Double black external node

Example



- Remove D
- Swap D & E
- Delete D
- Double black external node
- Rotate and recolor