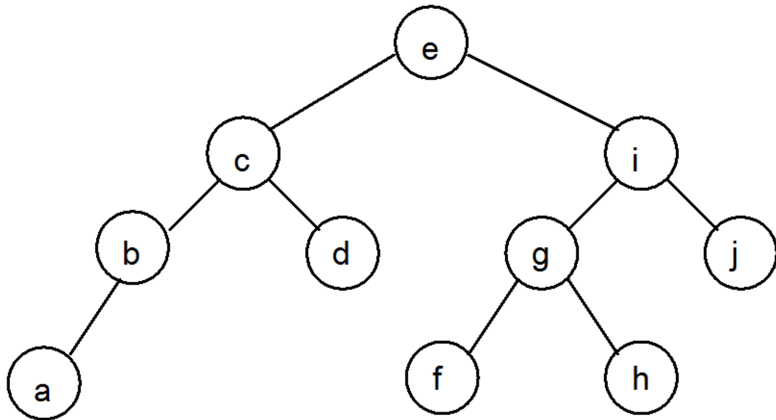
Binary Search Trees

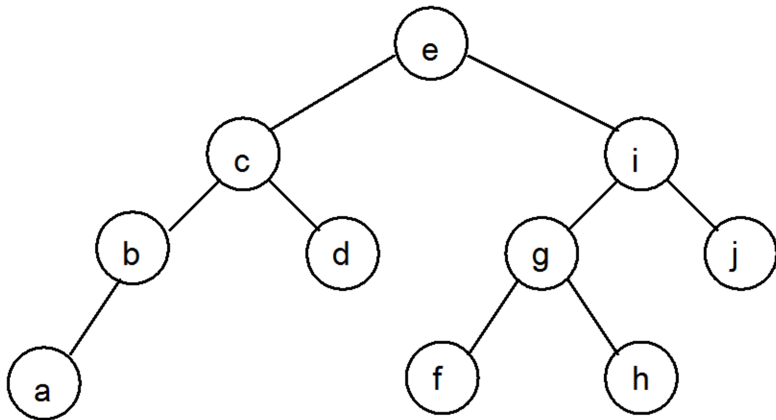
Binary search trees

- A binary search tree is a binary tree where ***all elements in the left subtree are less than elements in the right subtree***



Binary search trees

- A binary search tree is a binary tree where ***all elements in the left subtree are less than elements in the right subtree***



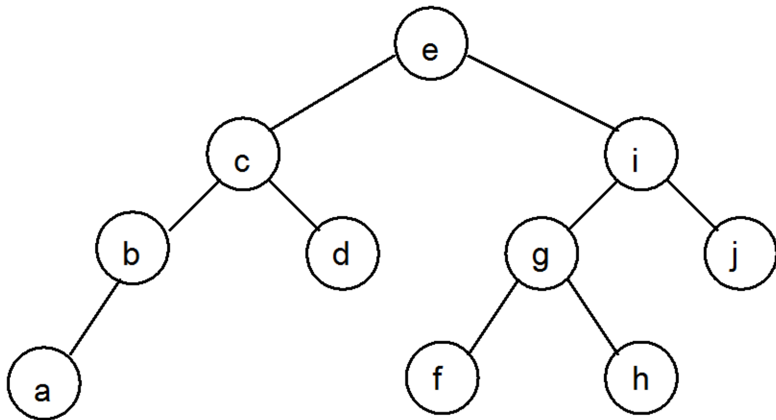
As we saw earlier, inorder traversal is

a b c d e f g h i j which is in sorted order

So this is a binary search tree

Binary search trees

- A binary search tree is a binary tree whose inorder traversal is in sorted order



As we saw earlier, inorder traversal is

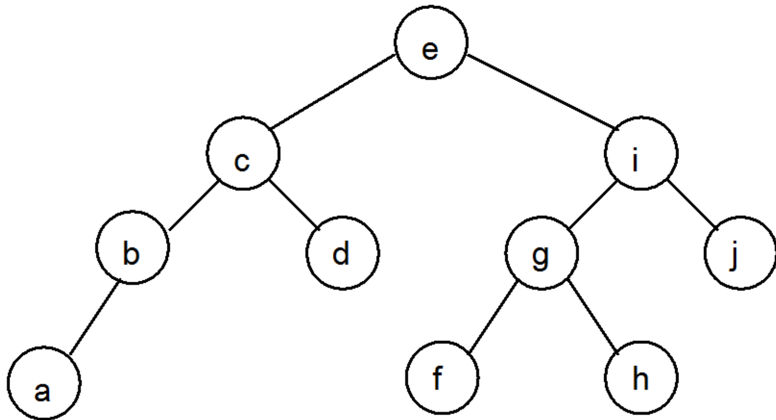
a b c d e f g h i j which is in sorted order

So this is a binary search tree

There are many more (binary) search trees that have inorder traversal a b c d e f g h i j

Binary search trees

- A binary search tree is a binary tree whose inorder traversal is in sorted order



As we saw earlier, inorder traversal is

a b c d e f g h i j which is in sorted order

So this is a binary search tree

All entries are unique!

Typical use … representing ***a set***

http://webdiis.unizar.es/asig

AVL tree applet

File Edit View Favorites Tools Help

JAVA MODELS

The inset below illustrates the behaviour of binary search trees.

Donald Knuth. "The Art of Computer Programming": Searching and Sorting Algorithms.

G.M. Adelson-Velskii and E.M. Landis. "An algorithm for the organization of information", 1962

D. Sleator and R. Tarjan. "Self-adjusting Binary Search Trees", 1985

"Symmetric binary B-trees. Data structure and maintenance algorithms.": R.Bayer, 1972

"A diochromatic framework for balanced trees.": L.J. Guibas and R. Sedgewick, 1978

SPL

R-B

AVL

23 01 39 00 11 35 74 03 19 25 37 62 89 02 09 16 21 59 63 93 20 50

Inserting 25. 25 inserted.

Insert Find Delete Min DeleteAll Traverse in-order

[demo](http://webdiis.unizar.es/asignaturas/EDA/AVLTree/avltree.html)

http://webdiis.unizar.es/asignaturas/EDA/AVLTree/avltree.html

Implementing search in a binary search tree

- Search
  - Can implement binary search in O(log n) time on average
  - Takes longer if the tree is badly balanced

- For every node X, value in all nodes in left subtree of X are less than value in X, and value of all nodes in right subtree are greater than value in X

- Algorithm is simple:

    if x < node **then** search left subtree
    if x > node **then** search right subtree

When the tree is balanced the path length to the leaves is log(n)

Find 17

http://webdiis.unizar.es/asic AVL tree applet

File Edit View Favorites Tools Help

JAVA MODELS

The inset below illustrates the behaviour of binary search trees.
Donald Knuth. "The Art of Computer Programming": Searching and Sorting Algorithms.
G.M. Adelson-Velskii and E.M. Landis. "An algorithm for the organization of information", 1962
D. Sleator and R. Tarjan. "Self-adjusting Binary Search Trees", 1985
"Symmetric binary B-trees. Data structure and maintenance algorithms.": R.Bayer, 1972
"A diochromatic framework for balanced trees.": L.J. Guibas and R. Sedgewick, 1978

23
01 39
00 11 35 74
03 19 25 37 62 89
02 09 16 21 59 63 93
20 50

SPL
R-B
AVL

Inserting 25. 25 inserted.

Insert | Find | Delete | Min | DeleteAll | Traverse | in-order

Find 17

http://webdiis.unizar.es/asig AVL tree applet

File Edit View Favorites Tools Help

JAVA MODELS

The inset below illustrates the behaviour of binary search trees.
Donald Knuth. "The Art of Computer Programming": Searching and Sorting Algorithms.
G.M. Adelson-Velskii and E.M. Landis. "An algorithm for the organization of information", 1962
D. Sleator and R. Tarjan. "Self-adjusting Binary Search Trees", 1985
"Symmetric binary B-trees. Data structure and maintenance algorithms.": R.Bayer, 1972
"A diochromatic framework for balanced trees.": L.J. Guibas and R. Sedgewick, 1978

23
17 39
01 35 74
00 11 25 37 62 89
03 19 59 63 93
02 09 16 21 50
20

SPL
R-B
AVL

Animation paused.

Insert Find Delete Min DeleteAll Traverse in-order

Find 17

http://webdiis.unizar.es/asig

AVL tree applet

File Edit View Favorites Tools Help

JAVA MODELS

The inset below illustrates the behaviour of binary search trees.

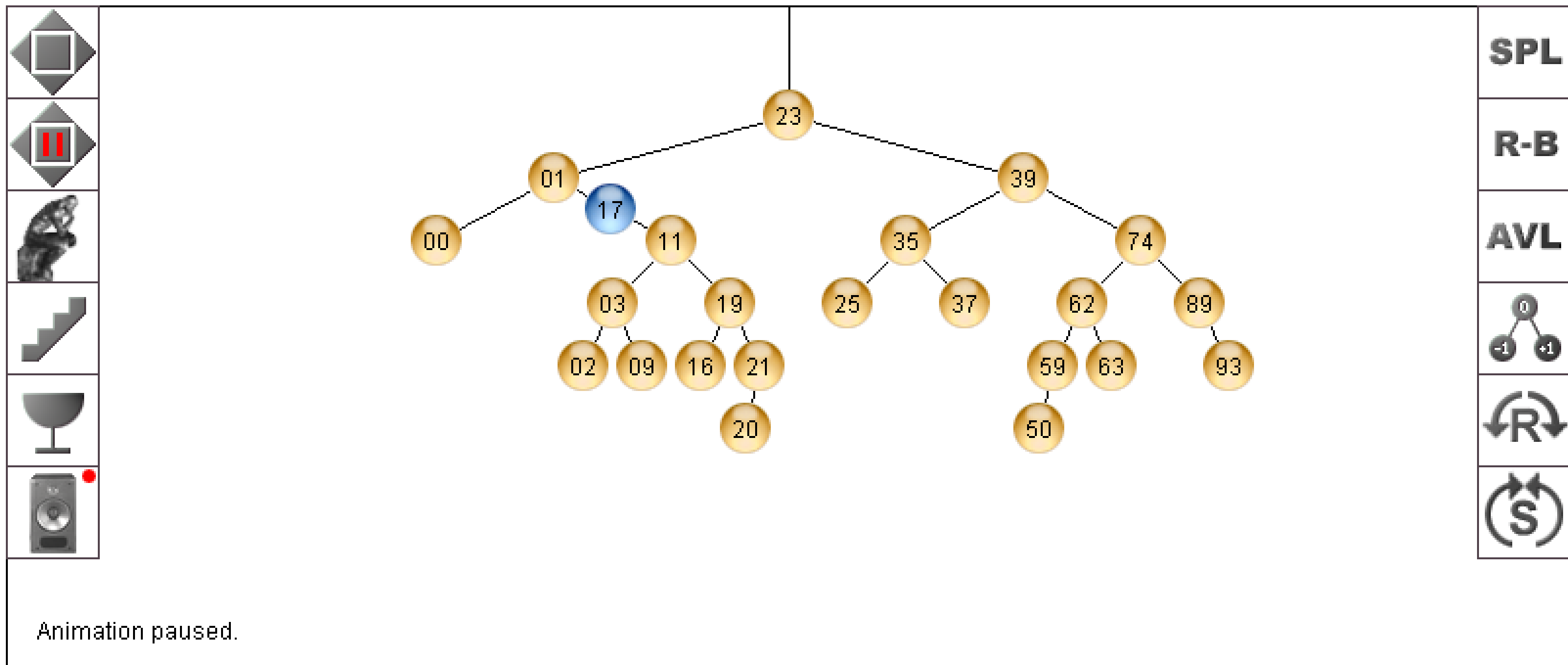
Donald Knuth. "The Art of Computer Programming": Searching and Sorting Algorithms.

G.M. Adelson-Velskii and E.M. Landis. "An algorithm for the organization of information", 1962

D. Sleator and R. Tarjan. "Self-adjusting Binary Search Trees", 1985

"Symmetric binary B-trees. Data structure and maintenance algorithms.": R.Bayer, 1972

"A diochromatic framework for balanced trees.": L.J. Guibas and R. Sedgewick, 1978



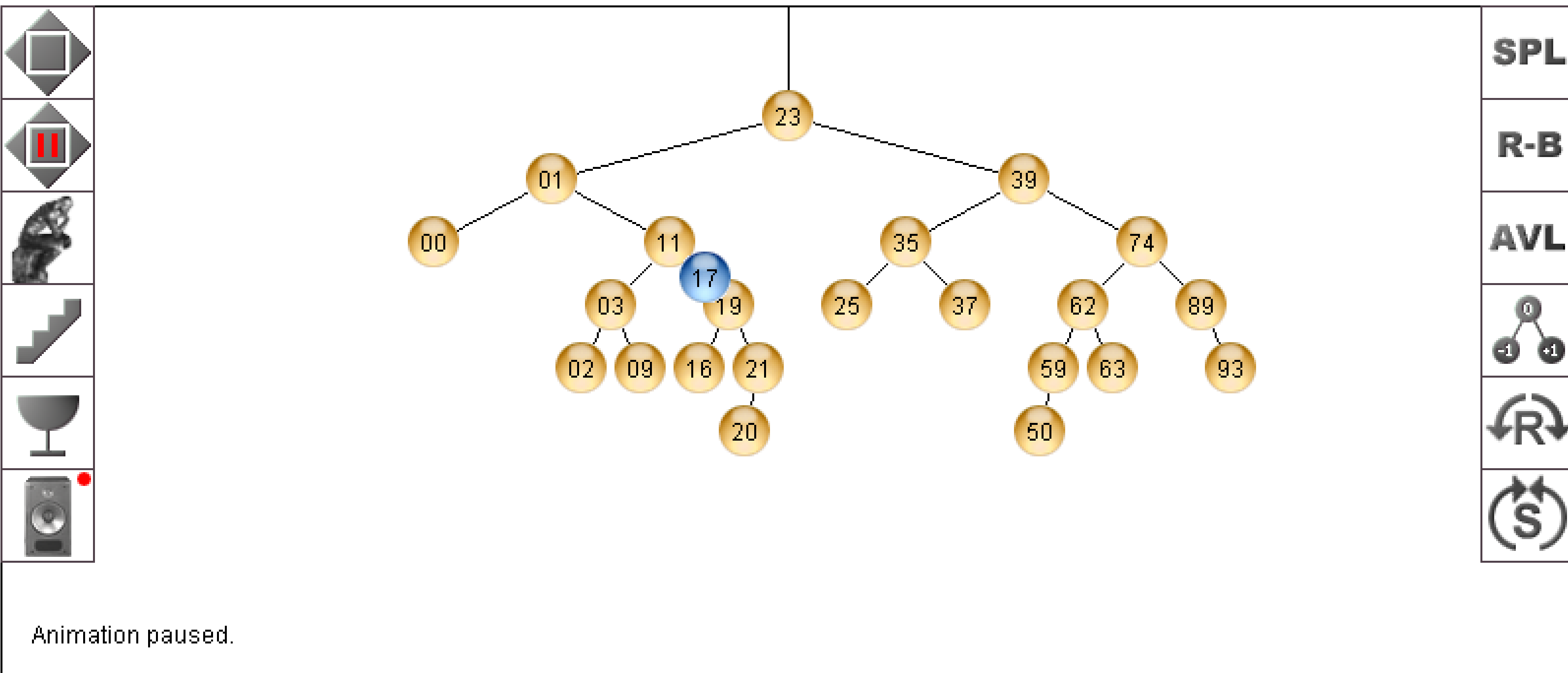
Find 17

http://webdiis.unizar.es/asig AVL tree applet

File Edit View Favorites Tools Help

JAVA MODELS

The inset below illustrates the behaviour of binary search trees.
Donald Knuth. "The Art of Computer Programming": Searching and Sorting Algorithms.
G.M. Adelson-Velskii and E.M. Landis. "An algorithm for the organization of information", 1962
D. Sleator and R. Tarjan. "Self-adjusting Binary Search Trees", 1985
"Symmetric binary B-trees. Data structure and maintenance algorithms.": R.Bayer, 1972
"A diochromatic framework for balanced trees.": L.J. Guibas and R. Sedgewick, 1978



Find 17

http://webdiis.unizar.es/asig

AVL tree applet

File Edit View Favorites Tools Help

JAVA MODELS

The inset below illustrates the behaviour of binary search trees.

Donald Knuth. "The Art of Computer Programming": Searching and Sorting Algorithms.

G.M. Adelson-Velskii and E.M. Landis. "An algorithm for the organization of information", 1962

D. Sleator and R. Tarjan. "Self-adjusting Binary Search Trees", 1985

"Symmetric binary B-trees. Data structure and maintenance algorithms.": R.Bayer, 1972

"A diochromatic framework for balanced trees.": L.J. Guibas and R. Sedgewick, 1978

SPL

R-B

AVL

23 01 39 00 11 35 74 03 19 25 37 62 89 17 02 09 16 21 59 63 93 20 50

Animation paused.

Insert Find Delete Min DeleteAll Traverse in-order

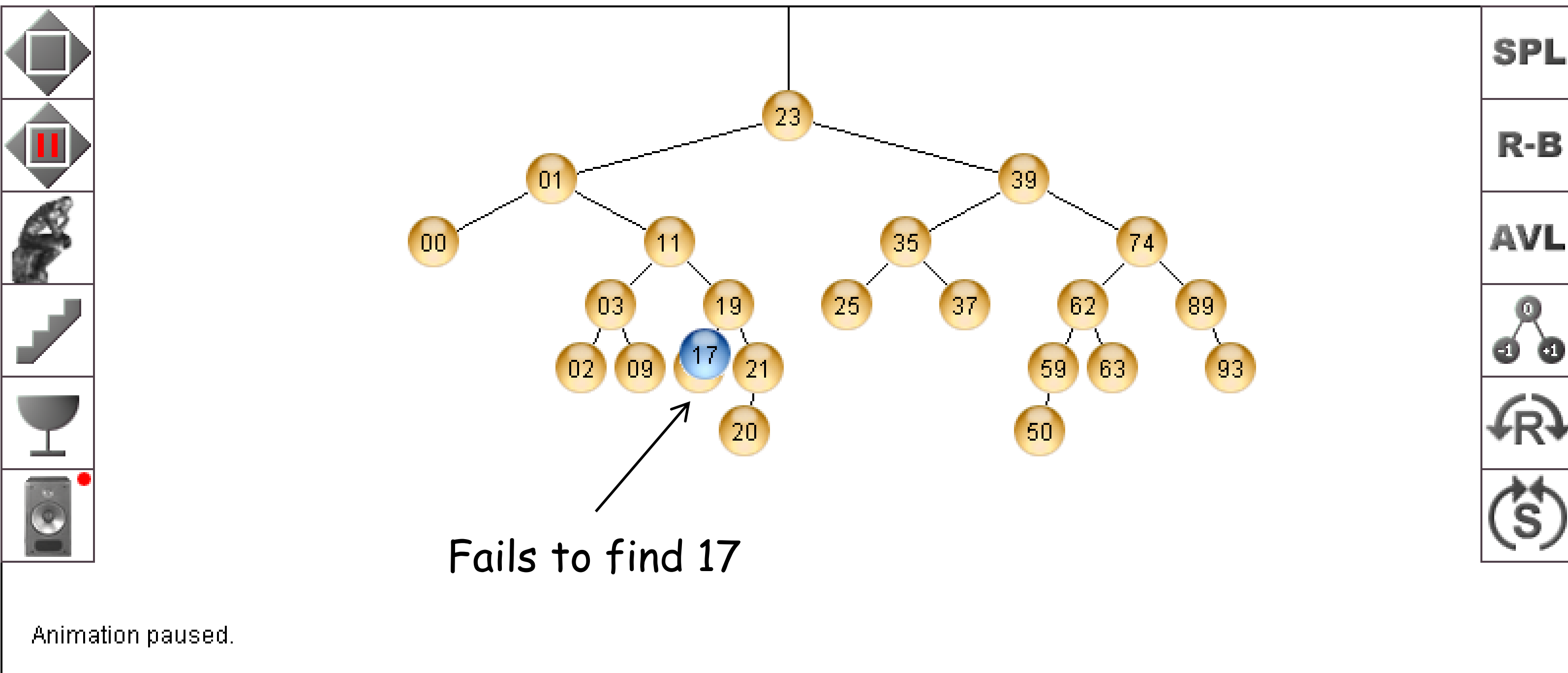
Find 17

http://webdiis.unizar.es/asic AVL tree applet

File Edit View Favorites Tools Help

JAVA MODELS

The inset below illustrates the behaviour of binary search trees.
Donald Knuth. "The Art of Computer Programming": Searching and Sorting Algorithms.
G.M. Adelson-Velskii and E.M. Landis. "An algorithm for the organization of information", 1962
D. Sleator and R. Tarjan. "Self-adjusting Binary Search Trees", 1985
"Symmetric binary B-trees. Data structure and maintenance algorithms.": R.Bayer, 1972
"A diochromatic framework for balanced trees.": L.J. Guibas and R. Sedgewick, 1978



Insert | Find | Delete | Min | DeleteAll | Traverse | in-order

Inserting a new node

This method works whether we are using the tree to implement a set, sequence etc. as long as it is an *injective* binary search tree.

Inserting a new node in general we must:

- Add to bottom of tree at all times (add a leaf)
- Keep the search tree property (left less than right)

http://webdiis.unizar.es/asig

AVL tree applet

File Edit View Favorites Tools Help

JAVA MODELS

The inset below illustrates the behaviour of binary search trees.

Donald Knuth. "The Art of Computer Programming": Searching and Sorting Algorithms.

G.M. Adelson-Velskii and E.M. Landis. "An algorithm for the organization of information", 1962

D. Sleator and R. Tarjan. "Self-adjusting Binary Search Trees", 1985

"Symmetric binary B-trees. Data structure and maintenance algorithms.": R.Bayer, 1972

"A diochromatic framework for balanced trees.": L.J. Guibas and R. Sedgewick, 1978

44 18 70 09 32 50 91 03 15 28 42 65 86 27 77 90

SPL R-B AVL

Animation resumed.

Insert Find Delete Min DeleteAll Traverse in-order

Insert 40

File Edit View Favorites Tools Help

JAVA MODELS

The inset below illustrates the behaviour of binary search trees.
Donald Knuth. "The Art of Computer Programming": Searching and Sorting Algorithms.
G.M. Adelson-Velskii and E.M. Landis. "An algorithm for the organization of information", 1962
D. Sleator and R. Tarjan. "Self-adjusting Binary Search Trees", 1985
"Symmetric binary B-trees. Data structure and maintenance algorithms.": R.Bayer, 1972
"A diochromatic framework for balanced trees.": L.J. Guibas and R. Sedgewick, 1978

40
44
18 70
09 32 50 91
03 15 28 42 65 86
27 77 90

SPL
R-B
AVL

Animation paused.

Insert Find Delete Min DeleteAll Traverse in-order

Insert 40

http://webdiis.unizar.es/asic

AVL tree applet

File Edit View Favorites Tools Help

JAVA MODELS

The inset below illustrates the behaviour of binary search trees.

Donald Knuth. "The Art of Computer Programming": Searching and Sorting Algorithms.

G.M. Adelson-Velskii and E.M. Landis. "An algorithm for the organization of information", 1962

D. Sleator and R. Tarjan. "Self-adjusting Binary Search Trees", 1985

"Symmetric binary B-trees. Data structure and maintenance algorithms.": R.Bayer, 1972

"A diochromatic framework for balanced trees.": L.J. Guibas and R. Sedgewick, 1978

Animation paused.

Insert Find Delete Min DeleteAll Traverse in-order

Insert 40

http://webdiis.unizar.es/asic

AVL tree applet

File Edit View Favorites Tools Help

JAVA MODELS

The inset below illustrates the behaviour of binary search trees.

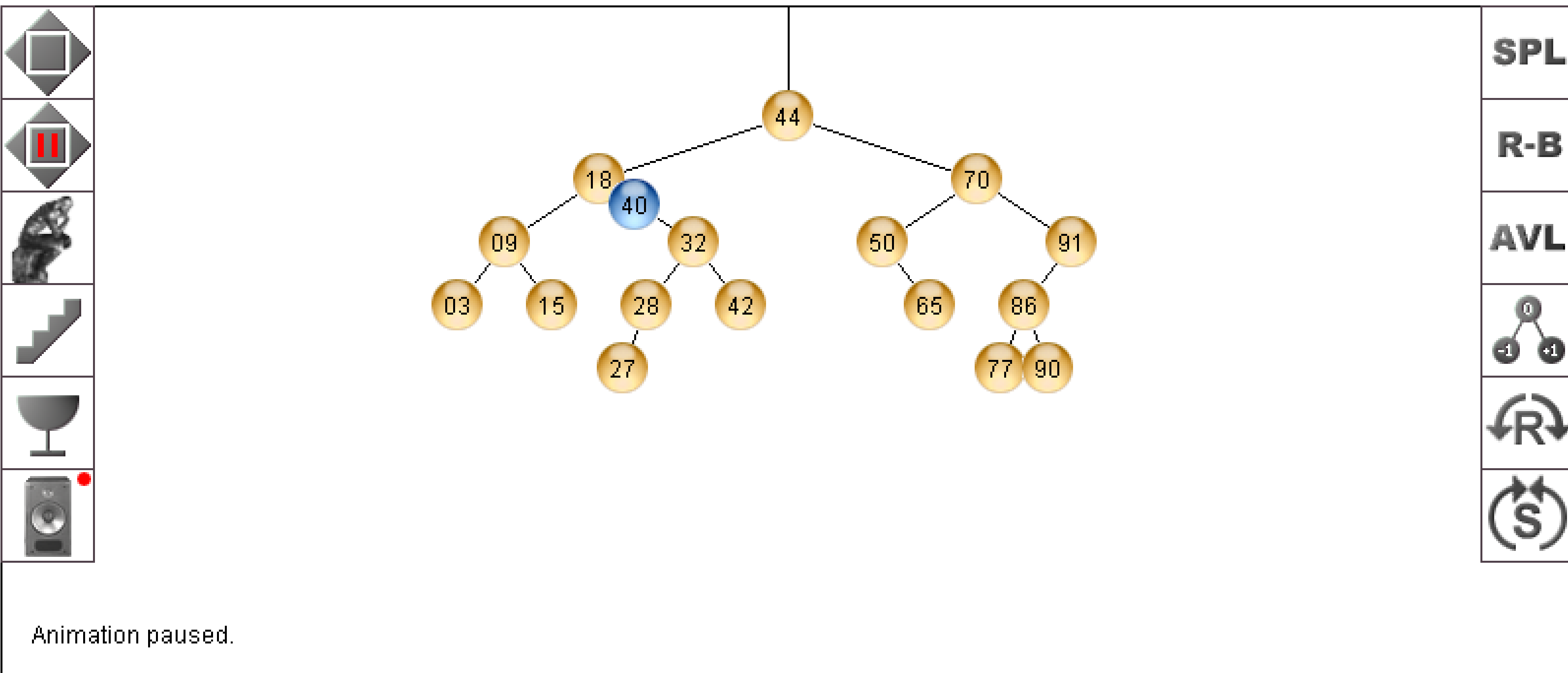
Donald Knuth. "The Art of Computer Programming": Searching and Sorting Algorithms.

G.M. Adelson-Velskii and E.M. Landis. "An algorithm for the organization of information", 1962

D. Sleator and R. Tarjan. "Self-adjusting Binary Search Trees", 1985

"Symmetric binary B-trees. Data structure and maintenance algorithms.": R.Bayer, 1972

"A diochromatic framework for balanced trees.": L.J. Guibas and R. Sedgewick, 1978



Insert Find Delete Min DeleteAll Traverse in-order

Insert 40

http://webdiis.unizar.es/asig   AVL tree applet

File Edit View Favorites Tools Help

JAVA MODELS

The inset below illustrates the behaviour of binary search trees.

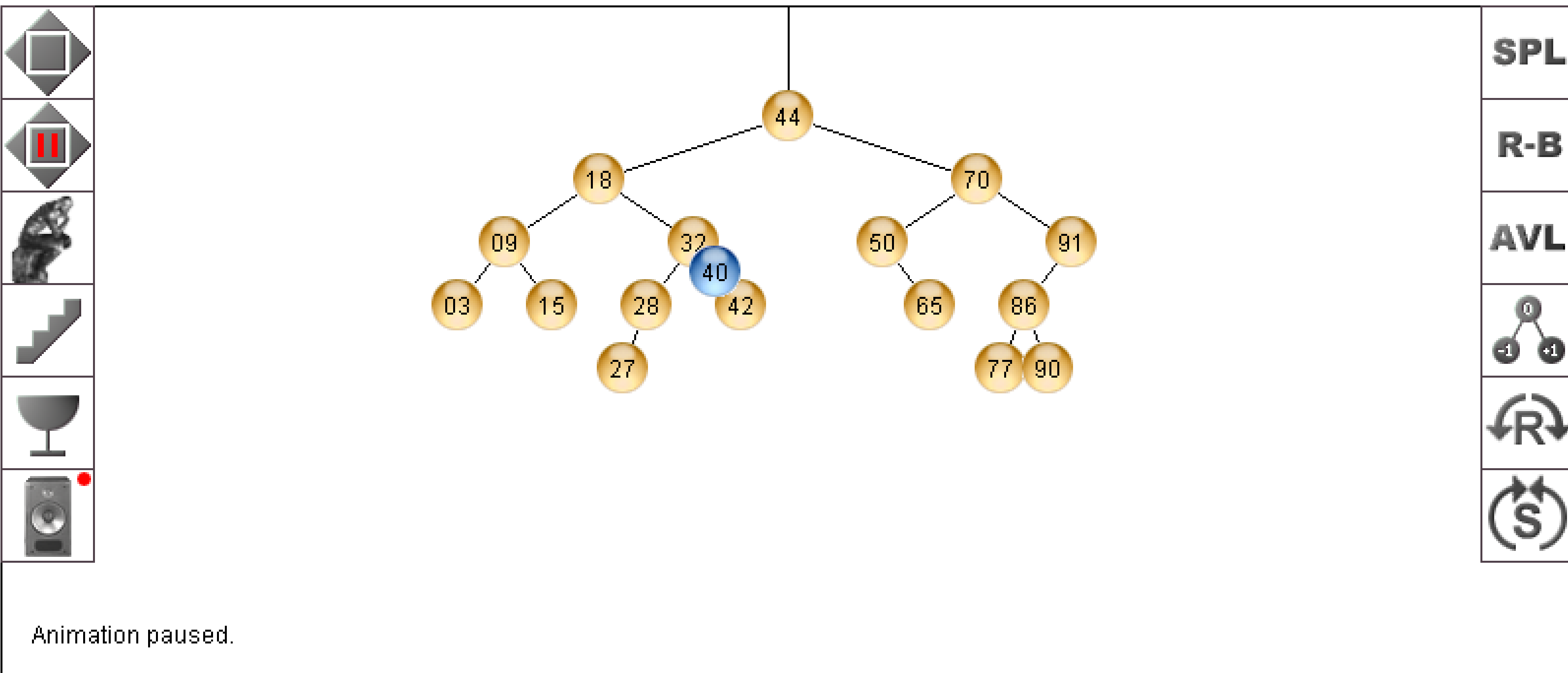
Donald Knuth. "The Art of Computer Programming": Searching and Sorting Algorithms.

G.M. Adelson-Velskii and E.M. Landis. "An algorithm for the organization of information", 1962

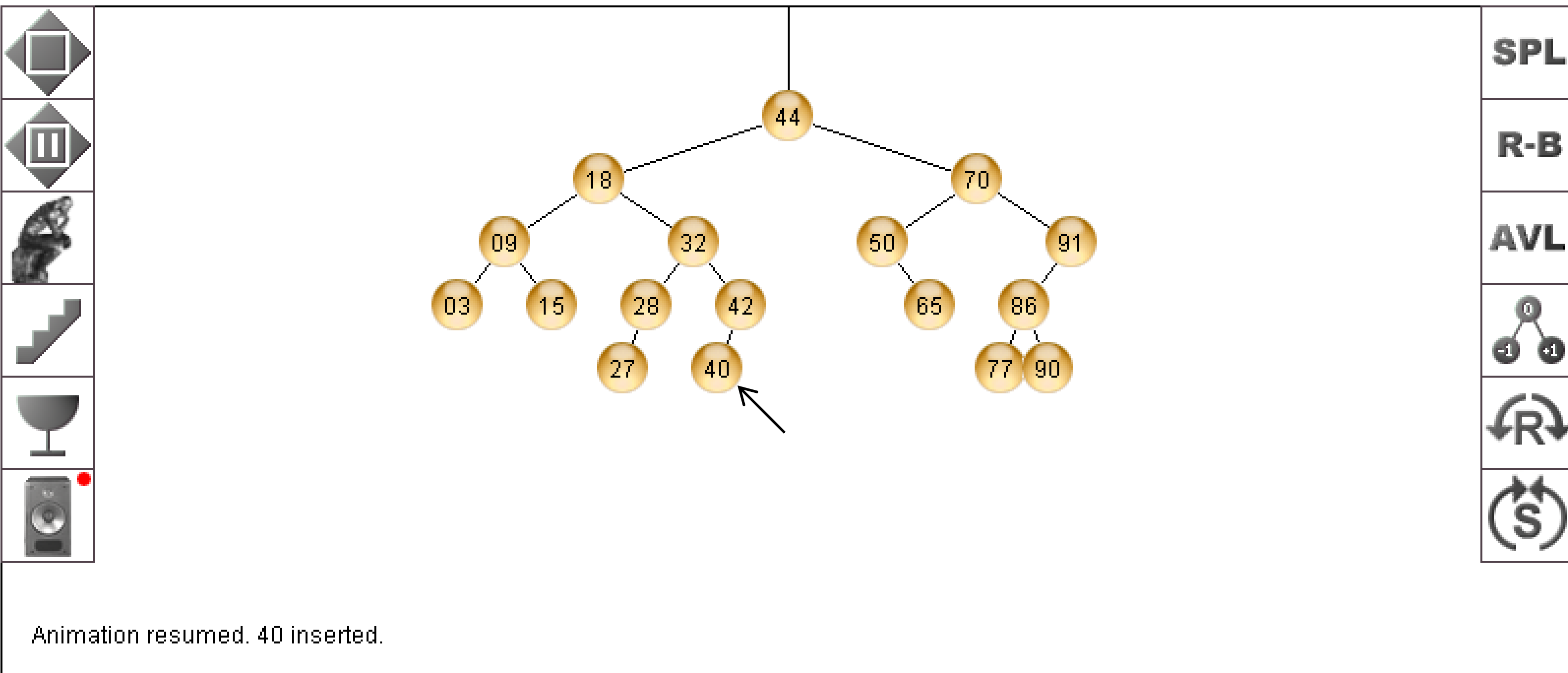
D. Sleator and R. Tarjan. "Self-adjusting Binary Search Trees", 1985

"Symmetric binary B-trees. Data structure and maintenance algorithms.": R.Bayer, 1972

"A diochromatic framework for balanced trees.": L.J. Guibas and R. Sedgewick, 1978

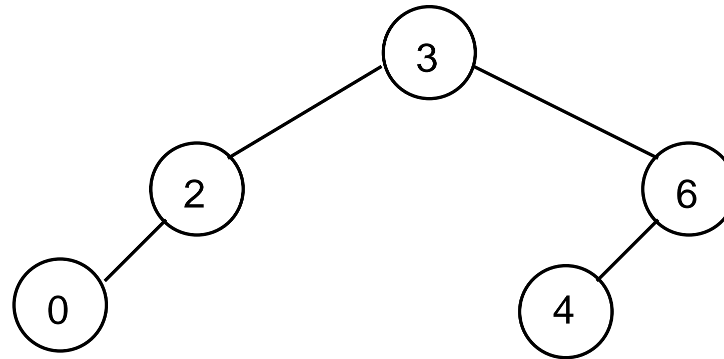


Insert 40

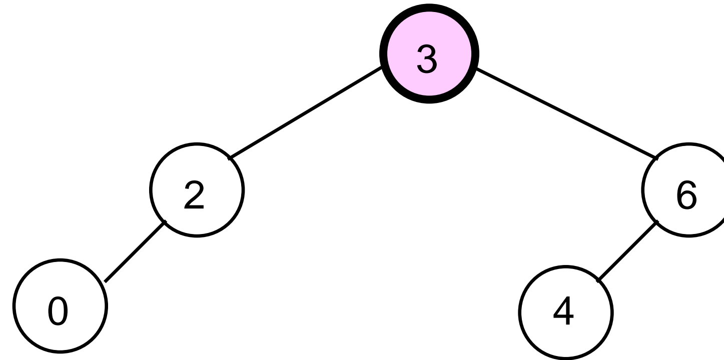


Another illustration of insertion …

Adding a node with value x = 5. (Example)

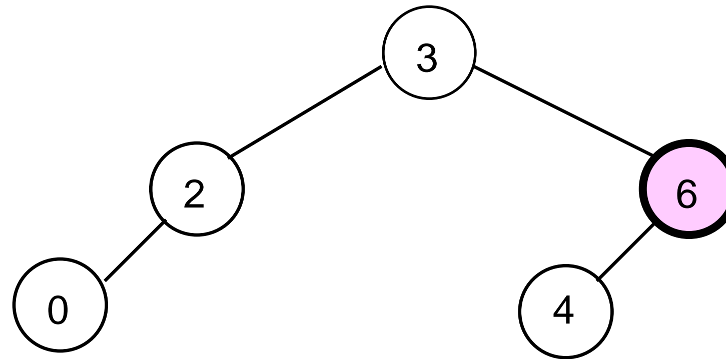


Adding a node with value x = 5. (Example)



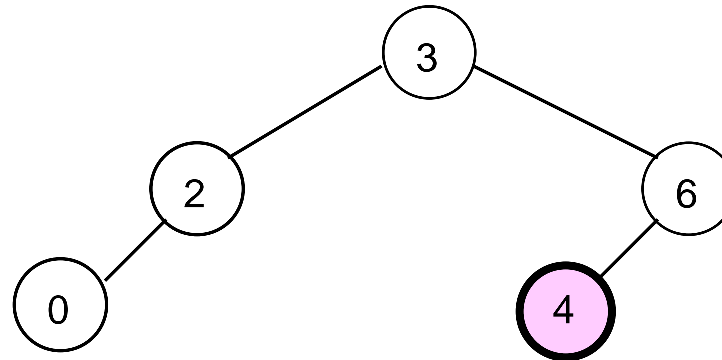
Compare 5 with node:
5 > 3 so go right

Adding a node with value x = 5. (Example)



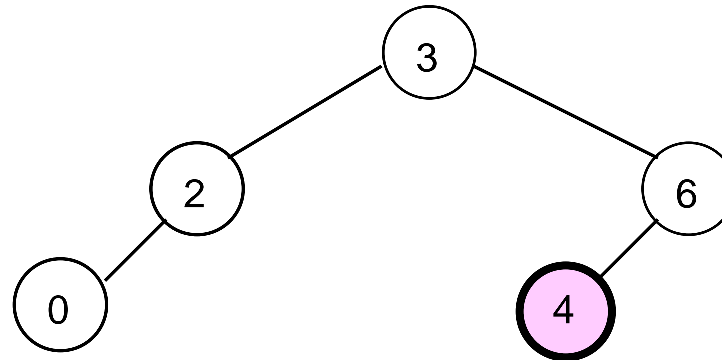
Compare 5 with node:
5 < 6 so go left

Adding a node with value x = 5. (Example)



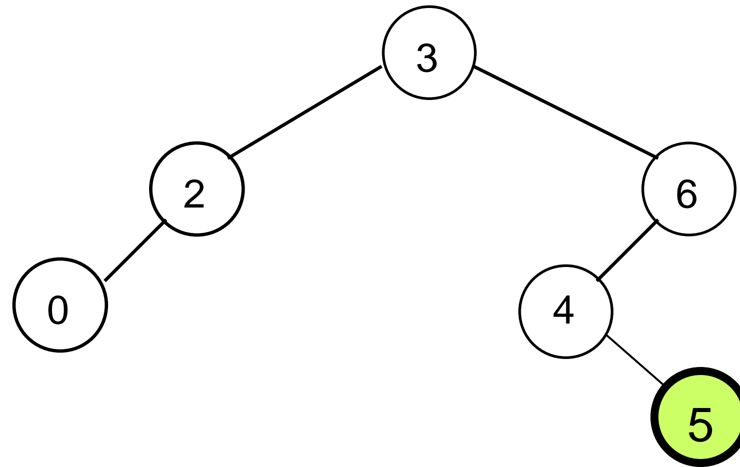
Compare 5 with node:
5 > 4 so go right

Adding a node with value x = 5. (Example)



Can’t go right as that is null
Insert node right of here, as a leaf

Adding a node with value x = 5. (Example)



Can't go right as that is null
Insert node right of here, as a leaf

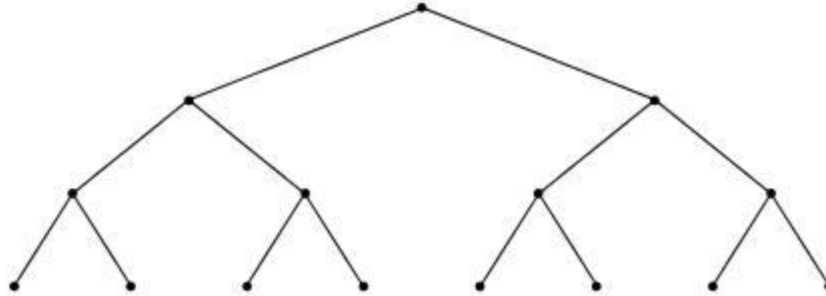
Why bother?

just
file it
under
“who cares?”

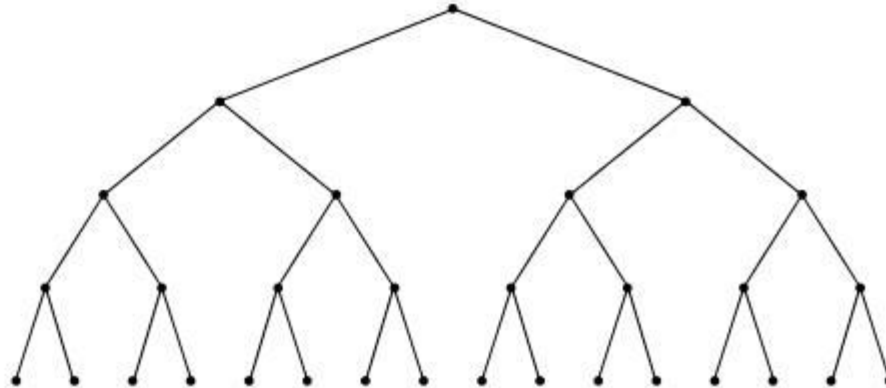
$n=3$, $\log(n+1)-1 = 1 \leq \text{height} \leq n-1$

$n=7$, $\log(n+1)-1 = 2 \le$ height $\le n-1$

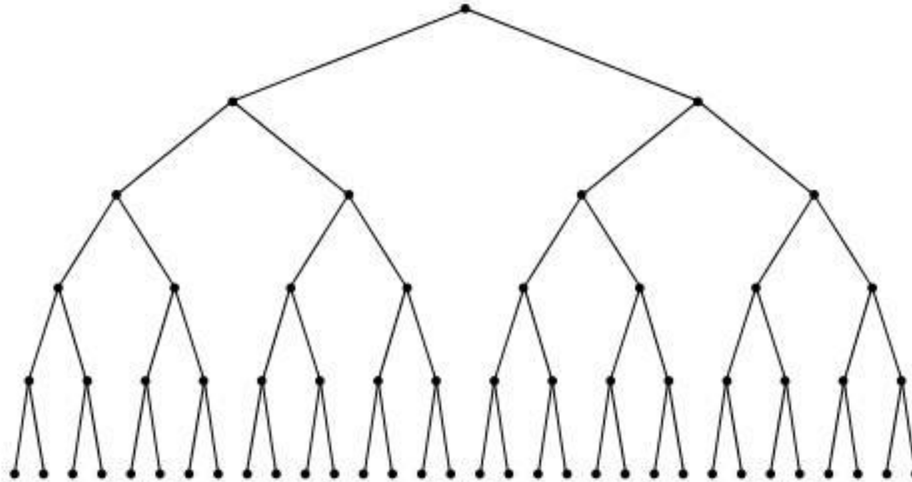
$n=15$, $\log(n+1)-1 = 3 \leq$ height $\leq n-1$



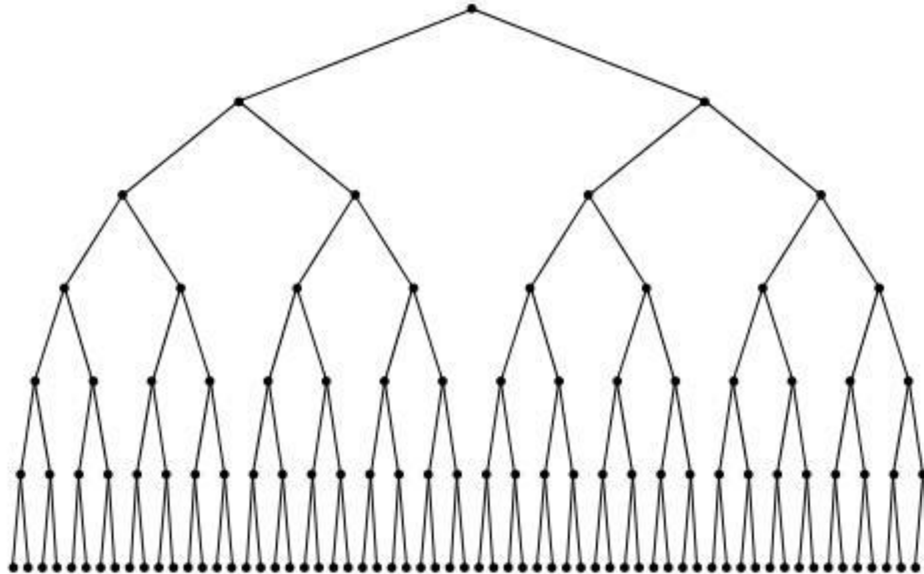
$n=31$, $\log(n+1)-1 = 4 \leq \text{height} \leq n-1$



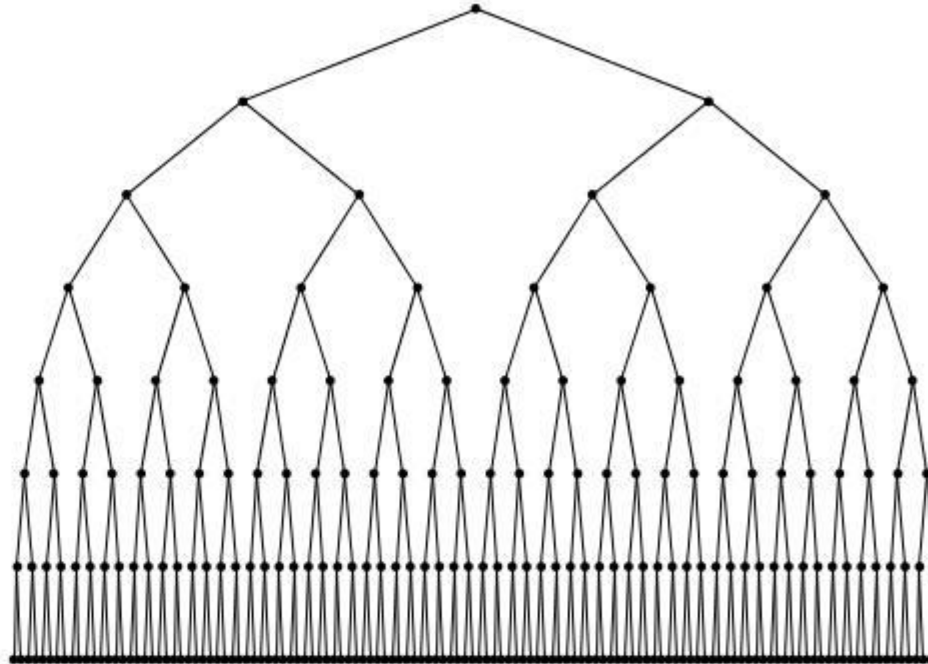
$n=63, \log(n+1)-1 = 5 \leq \text{height} \leq n-1$



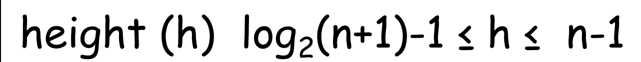
$n=127$, $\log(n+1)-1 = 6 \le \text{height} \le n-1$



n=255, $\log(n+1)-1 = 7 \le$ height $\le$ n-1



And this is crucial



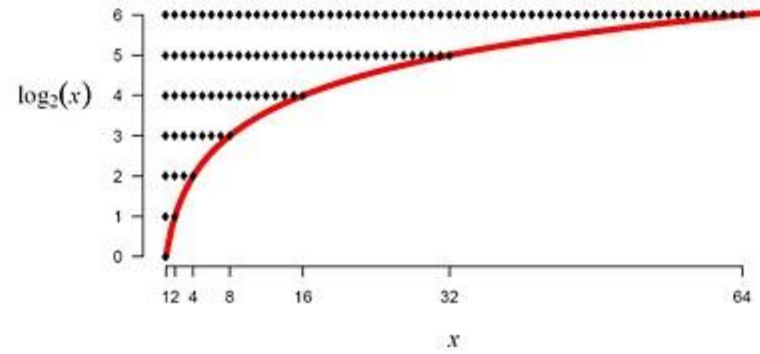
If we get it right we can access data in logarithmic time!

Log to the base 2 of …

2 = 1
4 = 2
8 = 3
16 = 4
…
…
…
1024 = 10
…
…
…

Log to the base 2 of …

2 = 1
4 = 2
8 = 3
16 = 4
…
…
…
1024 = 10
…
…
…



Brilliant!

java implementation

Node

Node

Node - Notepad

File Edit Format View Help

```
public class Node {

    private String element;
    private Node   left;
    private Node   right;
    private Node   parent;

    public Node(){this(null,null,null,null);}

    public Node(String e, Node left,Node right,Node parent){
        this.element = e;
        this.left    = left;
        this.right   = right;
        this.parent  = parent;
    }

    public String  getElement(){return element;}
    public Node getLeft(){return left;}
    public Node getRight(){return right;}
    public Node getParent(){return parent;}

    public void setElement(String e){element = e;}
    public void setLeft(Node node){left = node; if (node != null) node.setParent(this);}
    public void setRight(Node node){right = node; if (node != null) node.setParent(this);}
    public void setParent(Node node){parent = node;}

    public boolean isRoot(){return parent == null;}
    public boolean isLeaf(){return left == null && right == null;}
    public boolean isInternal(){return left != null && right != null;}
    public boolean isLeftChild(){return parent.getLeft() == this;}
    public boolean isRightChild(){return parent.getRight() == this;}
    public boolean hasLeft(){return left != null;}
    public boolean hasRight(){return right != null;}

    public String toString(){return element.toString();}
}
```

Node - Notepad

File Edit Format View Help

```
public class Node {
    private String element;
    private Node   left;
    private Node   right;
    private Node   parent;
    public Node(){this(null,null,null,null);}

    public Node(String e, Node left,Node right,Node parent){
        this.element = e;
        this.left    = left;
        this.right   = right;
        this.parent  = parent;
    }

    public String  getElement(){return element;}
    public Node getLeft(){return left;}
    public Node getRight(){return right;}
    public Node getParent(){return parent;}

    public void setElement(String e){element = e;}
    public void setLeft(Node node){left = node; if (node != null) node.setParent(this);}
    public void setRight(Node node){right = node; if (node != null) node.setParent(this);}
    public void setParent(Node node){parent = node;}

    public boolean isRoot(){return parent == null;}
    public boolean isLeaf(){return left == null && right == null;}
    public boolean isInternal(){return left != null && right != null;}
    public boolean isLeftChild(){return parent.getLeft() == this;}
    public boolean isRightChild(){return parent.getRight() == this;}
    public boolean hasLeft(){return left != null;}
    public boolean hasRight(){return right != null;}

    public String toString(){return element.toString();}
}
```

Node - Notepad

File Edit Format View Help

```
public class Node {

    private String element;
    private Node   left;
    private Node   right;
    private Node   parent;

    public Node(){this(null,null,null,null);}

    public Node(String e, Node left,Node right,Node parent){
        this.element = e;
        this.left    = left;
        this.right   = right;
        this.parent  = parent;
    }

    public String  getElement(){return element;}
    public Node getLeft(){return left;}
    public Node getRight(){return right;}
    public Node getParent(){return parent;}

    public void setElement(String e){element = e;}
    public void setLeft(Node node){left = node; if (node != null) node.setParent(this);}
    public void setRight(Node node){right = node; if (node != null) node.setParent(this);}
    public void setParent(Node node){parent = node;}

    public boolean isRoot(){return parent == null;}
    public boolean isLeaf(){return left == null && right == null;}
    public boolean isInternal(){return left != null && right != null;}
    public boolean isLeftChild(){return parent.getLeft() == this;}
    public boolean isRightChild(){return parent.getRight() == this;}
    public boolean hasLeft(){return left != null;}
    public boolean hasRight(){return right != null;}

    public String toString(){return element.toString();}
}
```

Default constructor

Node

Node - Notepad

File Edit Format View Help

```
public class Node {

    private String element;
    private Node   left;
    private Node   right;
    private Node   parent;

    public Node(){this(null,null,null,null);}

    public Node(String e, Node left,Node right,Node parent){
        this.element = e;
        this.left    = left;
        this.right   = right;
        this.parent  = parent;
    }

    public String  getElement(){return element;}
    public Node getLeft(){return left;}
    public Node getRight(){return right;}
    public Node getParent(){return parent;}

    public void setElement(String e){element = e;}
    public void setLeft(Node node){left = node; if (node != null) node.setParent(this);}
    public void setRight(Node node){right = node; if (node != null) node.setParent(this);}
    public void setParent(Node node){parent = node;}

    public boolean isRoot(){return parent == null;}
    public boolean isLeaf(){return left == null && right == null;}
    public boolean isInternal(){return left != null && right != null;}
    public boolean isLeftChild(){return parent.getLeft() == this;}
    public boolean isRightChild(){return parent.getRight() == this;}
    public boolean hasLeft(){return left != null;}
    public boolean hasRight(){return right != null;}

    public String toString(){return element.toString();}
}
```

parameterised constructor
note use of **this**.

Node

Node - Notepad

File Edit Format View Help

```
public class Node {

    private String element;
    private Node   left;
    private Node   right;
    private Node   parent;

    public Node(){this(null,null,null,null);}

    public Node(String e, Node left,Node right,Node parent){
        this.element = e;
        this.left    = left;
        this.right   = right;
        this.parent  = parent;
    }

    public String  getElement(){return element;}
    public Node getLeft(){return left;}
    public Node getRight(){return right;}
    public Node getParent(){return parent;}

    public void setElement(String e){element = e;}
    public void setLeft(Node node){left = node; if (node != null) node.setParent(this);}
    public void setRight(Node node){right = node; if (node != null) node.setParent(this);}
    public void setParent(Node node){parent = node;}

    public boolean isRoot(){return parent == null;}
    public boolean isLeaf(){return left == null && right == null;}
    public boolean isInternal(){return left != null && right != null;}
    public boolean isLeftChild(){return parent.getLeft() == this;}
    public boolean isRightChild(){return parent.getRight() == this;}
    public boolean hasLeft(){return left != null;}
    public boolean hasRight(){return right != null;}

    public String toString(){return element.toString();}
}
```

getters

Node

Node - Notepad

File Edit Format View Help

```
public class Node {

    private String element;
    private Node   left;
    private Node   right;
    private Node   parent;

    public Node(){this(null,null,null,null);}

    public Node(String e, Node left,Node right,Node parent){
        this.element = e;
        this.left    = left;
        this.right   = right;
        this.parent  = parent;
    }

    public String  getElement(){return element;}
    public Node getLeft(){return left;}
    public Node getRight(){return right;}
    public Node getParent(){return parent;}

    public void setElement(String e){element = e;}
    public void setLeft(Node node){left = node; if (node != null) node.setParent(this);}
    public void setRight(Node node){right = node; if (node != null) node.setParent(this);}
    public void setParent(Node node){parent = node;}

    public boolean isRoot(){return parent == null;}
    public boolean isLeaf(){return left == null && right == null;}
    public boolean isInternal(){return left != null && right != null;}
    public boolean isLeftChild(){return parent.getLeft() == this;}
    public boolean isRightChild(){return parent.getRight() == this;}
    public boolean hasLeft(){return left != null;}
    public boolean hasRight(){return right != null;}

    public String toString(){return element.toString();}
}
```

setters

Node

Node - Notepad

File Edit Format View Help

```
public class Node {

    private String element;
    private Node   left;
    private Node   right;
    private Node   parent;

    public Node(){this(null,null,null,null);}

    public Node(String e, Node left,Node right,Node parent){
        this.element = e;
        this.left    = left;
        this.right   = right;
        this.parent  = parent;
    }

    public String  getElement(){return element;}
    public Node getLeft(){return left;}
    public Node getRight(){return right;}
    public Node getParent(){return parent;}

    public void setElement(String e){element = e;}
    public void setLeft(Node node){left = node; if (node != null) node.setParent(this);}
    public void setRight(Node node){right = node; if (node != null) node.setParent(this);}
    public void setParent(Node node){parent = node;}

    public boolean isRoot(){return parent == null;}
    public boolean isLeaf(){return left == null && right == null;}
    public boolean isInternal(){return left != null && right != null;}
    public boolean isLeftChild(){return parent.getLeft() == this;}
    public boolean isRightChild(){return parent.getRight() == this;}
    public boolean hasLeft(){return left != null;}
    public boolean hasRight(){return right != null;}

    public String toString(){return element.toString();}
}
```

predicates

Node - Notepad

File Edit Format View Help

```java
public class Node {

    private String element;
    private Node   left;
    private Node   right;
    private Node   parent;

    public Node(){this(null,null,null,null);}

    public Node(String e, Node left,Node right,Node parent){
        this.element = e;
        this.left    = left;
        this.right   = right;
        this.parent  = parent;
    }

    public String  getElement(){return element;}
    public Node getLeft(){return left;}
    public Node getRight(){return right;}
    public Node getParent(){return parent;}

    public void setElement(String e){element = e;}
    public void setLeft(Node node){left = node; if (node != null) node.setParent(this);}
    public void setRight(Node node){right = node; if (node != null) node.setParent(this);}
    public void setParent(Node node){parent = node;}

    public boolean isRoot(){return parent == null;}
    public boolean isLeaf(){return left == null && right == null;}
    public boolean isInternal(){return left != null && right != null;}
    public boolean isLeftChild(){return parent.getLeft() == this;}
    public boolean isRightChild(){return parent.getRight() == this;}
    public boolean hasLeft(){return left != null;}
    public boolean hasRight(){return right != null;}

    public String toString(){return element.toString();}
}
```

For printing

The BSTree class

BinarySearchTree (BSTree)

BSTree

BSTree - Notepad

File Edit Format View Help

```
public class BSTree {
    private Node root;
    private int size;

    public BSTree(){root = null; size = 0;}

    public Node root(){return root;}
    public boolean isEmpty(){return root == null;}
    public int size(){return size;}

    public void insert(String s){}
    //
    // insert the string s into a tree
    // (1) if the tree is empty
    //     then create a new node with s in it and setthe size of the tree to be 1 ... otherwise
    // (2) insert the string s into the tree rooted on the current node ... see below
    //

    private void insert(String s,Node node){}
    //
    // insert the string s into the tree rooted on the current node
    // (1) if s is less than the current node and the current node has a left child
    //     then insert s into the tree rooted on the left child ... otherwise
    // (2) if s is less than the current node and the current node has no left child
    //     then create a new node containing s, call it newNode
    //          make the left child of the current node be the newNode
    //          make the parent of the newNode be the current node ... otherwise
    // (3) if s is greater than the current node and the current node has a right child
    //     then insert s into the tree rooted on the right child ... otherwise
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //


    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //
```

BSTree

BSTree - Notepad

File Edit Format View Help

```
public class BSTree {
    private Node root;
    private int size;

    public BSTree(){root = null; size = 0;}

    public Node root(){return root;}
    public boolean isEmpty(){return root == null;}
    public int size(){return size;}

    public void insert(String s){}
    //
    // insert the string s into a tree
    // (1) if the tree is empty
    //     then create a new node with s in it and setthe size of the tree to be 1 ... otherwise
    // (2) insert the string s into the tree rooted on the current node ... see below
    //

    private void insert(String s,Node node){}
    //
    // insert the string s into the tree rooted on the current node
    // (1) if s is less than the current node and the current node has a left child
    //     then insert s into the tree rooted on the left child ... otherwise
    // (2) if s is less than the current node and the current node has no left child
    //     then create a new node containing s, call it newNode
    //          make the left child of the current node be the newNode
    //          make the parent of the newNode be the current node ... otherwise
    // (3) if s is greater than the current node and the current node has a right child
    //     then insert s into the tree rooted on the right child ... otherwise
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //


    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //
```

A binary search tree has a root where the root is a node
It also has a size, where size is the number of nodes in the tree

BSTree - Notepad

File Edit Format View Help

```
public class BSTree {
    private Node root;
    private int size;

    public BSTree(){root = null; size = 0;}

    public Node root(){return root;}
    public boolean isEmpty(){return root == null;}
    public int size(){return size;}

    public void insert(String s){}
    //
    // insert the string s into a tree
    // (1) if the tree is empty
    //     then create a new node with s in it and setthe size of the tree to be 1 ... otherwise
    // (2) insert the string s into the tree rooted on the current node ... see below
    //

    private void insert(String s,Node node){}
    //
    // insert the string s into the tree rooted on the current node
    // (1) if s is less than the current node and the current node has a left child
    //     then insert s into the tree rooted on the left child ... otherwise
    // (2) if s is less than the current node and the current node has no left child
    //     then create a new node containing s, call it newNode
    //          make the left child of the current node be the newNode
    //          make the parent of the newNode be the current node ... otherwise
    // (3) if s is greater than the current node and the current node has a right child
    //     then insert s into the tree rooted on the right child ... otherwise
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //


    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //
```

Default constructor, an empty tree with an empty root

BSTree

BSTree - Notepad

File Edit Format View Help

```
public class BSTree {
    private Node root;
    private int size;

    public BSTree(){root = null; size = 0;}

    public Node root(){return root;}
    public boolean isEmpty(){return root == null;}
    public int size(){return size;}

    public void insert(String s){}
    //
    // insert the string s into a tree
    // (1) if the tree is empty
    //     then create a new node with s in it and setthe size of the tree to be 1 ... otherwise
    // (2) insert the string s into the tree rooted on the current node ... see below
    //

    private void insert(String s,Node node){}
    //
    // insert the string s into the tree rooted on the current node
    // (1) if s is less than the current node and the current node has a left child
    //     then insert s into the tree rooted on the left child ... otherwise
    // (2) if s is less than the current node and the current node has no left child
    //     then create a new node containing s, call it newNode
    //          make the left child of the current node be the newNode
    //          make the parent of the newNode be the current node ... otherwise
    // (3) if s is greater than the current node and the current node has a right child
    //     then insert s into the tree rooted on the right child ... otherwise
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //


    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //
```

Get the root
Test if tree is empty
Get the size of the tree

BSTree

BSTree - Notepad

File Edit Format View Help

Insert string s into the tree

```
public class BSTree {
    private Node root;
    private int size;

    public BSTree(){root = null; size = 0;}

    public Node root(){return root;}
    public boolean isEmpty(){return root == null;}
    public int size(){return size;}

    public void insert(String s){}
    //
    // insert the string s into a tree
    // (1) if the tree is empty
    //     then create a new node with s in it and setthe size of the tree to be 1 ... otherwise
    // (2) insert the string s into the tree rooted on the current node ... see below
    //

    private void insert(String s,Node node){}
    //
    // insert the string s into the tree rooted on the current node
    // (1) if s is less than the current node and the current node has a left child
    //     then insert s into the tree rooted on the left child ... otherwise
    // (2) if s is less than the current node and the current node has no left child
    //     then create a new node containing s, call it newNode
    //          make the left child of the current node be the newNode
    //          make the parent of the newNode be the current node ... otherwise
    // (3) if s is greater than the current node and the current node has a right child
    //     then insert s into the tree rooted on the right child ... otherwise
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //


    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //
```

BSTree

BSTree - Notepad

File Edit Format View Help

Insert string s into the tree

```
public class BSTree {
    private Node root;
    private int size;

    public BSTree(){root = null; size = 0;}

    public Node root(){return root;}
    public boolean isEmpty(){return root == null;}
    public int size(){return size;}

    public void insert(String s){}
    //
    // insert the string s into a tree
    // (1) if the tree is empty
    //     then create a new node with s in it and setthe size of the tree to be 1 ... otherwise
    // (2) insert the string s into the tree rooted on the current node ... see below
    //
    private void insert(String s,Node node){}
    //
    // insert the string s into the tree rooted on the current node
    // (1) if s is less than the current node and the current node has a left child
    //     then insert s into the tree rooted on the left child ... otherwise
    // (2) if s is less than the current node and the current node has no left child
    //     then create a new node containing s, call it newNode
    //          make the left child of the current node be the newNode
    //          make the parent of the newNode be the current node ... otherwise
    // (3) if s is greater than the current node and the current node has a right child
    //     then insert s into the tree rooted on the right child ... otherwise
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //


    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //
```

BSTree - Notepad

File Edit Format View Help

```
public class BSTree {
    private Node root;
    private int size;

    public BSTree(){root = null; size = 0;}

    public Node root(){return root;}
    public boolean isEmpty(){return root == null;}
    public int size(){return size;}

    public void insert(String s){}
    //
    // insert the string s into a tree
    // (1) if the tree is empty
    //     then create a new node with s in it and set the size of the tree to be 1 ... otherwise
    // (2) insert the string s into the tree rooted on the current node ... see below
    //

    private void insert(String s,Node node){}
    //
    // insert the string s into the tree rooted on the current node
    // (1) if s is less than the current node and the current node has a left child
    //     then insert s into the tree rooted on the left child ... otherwise
    // (2) if s is less than the current node and the current node has no left child
    //     then create a new node containing s, call it newNode
    //          make the left child of the current node be the newNode
    //          make the parent of the newNode be the current node ... otherwise
    // (3) if s is greater than the current node and the current node has a right child
    //     then insert s into the tree rooted on the right child ... otherwise
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //

    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //
```

Insert string s into the subtree rooted on the current node

BSTree

BSTree - Notepad

File Edit Format View Help

```
public class BSTree {
    private Node root;
    private int size;

    public BSTree(){root = null; size = 0;}

    public Node root(){return root;}
    public boolean isEmpty(){return root == null;}
    public int size(){return size;}

    public void insert(String s){}
    //
    // insert the string s into a tree
    // (1) if the tree is empty
    //     then create a new node with s in it and setthe size of the tree to be 1 ... otherwise
    // (2) insert the string s into the tree rooted on the current node ... see below
    //

    private void insert(String s,Node node){}
    //
    // insert the string s into the tree rooted on the current node
    // (1) if s is less than the current node and the current node has a left child
    //     then insert s into the tree rooted on the left child ... otherwise
    // (2) if s is less than the current node and the current node has no left child
    //     then create a new node containing s, call it newNode
    //          make the left child of the current node be the newNode
    //          make the parent of the newNode be the current node ... otherwise
    // (3) if s is greater than the current node and the current node has a right child
    //     then insert s into the tree rooted on the right child ... otherwise
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //

    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //
```

If s is in the left subtree …

BSTree - Notepad

File Edit Format View Help

```
public class BSTree {
    private Node root;
    private int size;

    public BSTree(){root = null; size = 0;}

    public Node root(){return root;}
    public boolean isEmpty(){return root == null;}
    public int size(){return size;}

    public void insert(String s){}
    //
    // insert the string s into a tree
    // (1) if the tree is empty
    //     then create a new node with s in it and setthe size of the tree to be 1 ... otherwise
    // (2) insert the string s into the tree rooted on the current node ... see below
    //

    private void insert(String s,Node node){}
    //
    // insert the string s into the tree rooted on the current node
    // (1) if s is less than the current node and the current node has a left child
    //     then insert s into the tree rooted on the left child ... otherwise
    // (2) if s is less than the current node and the current node has no left child
    //     then create a new node containing s, call it newNode
    //          make the left child of the current node be the newNode
    //          make the parent of the newNode be the current node ... otherwise
    // (3) if s is greater than the current node and the current node has a right child
    //     then insert s into the tree rooted on the right child ... otherwise
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //


    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //
```

If s is in the left subtree and we need to create a new node

BSTree

BSTree - Notepad

File Edit Format View Help

```
public class BSTree {
    private Node root;
    private int size;

    public BSTree(){root = null; size = 0;}

    public Node root(){return root;}
    public boolean isEmpty(){return root == null;}
    public int size(){return size;}

    public void insert(String s){}
    //
    // insert the string s into a tree
    // (1) if the tree is empty
    //     then create a new node with s in it and setthe size of the tree to be 1 ... otherwise
    // (2) insert the string s into the tree rooted on the current node ... see below
    //

    private void insert(String s,Node node){}
    //
    // insert the string s into the tree rooted on the current node
    // (1) if s is less than the current node and the current node has a left child
    //     then insert s into the tree rooted on the left child ... otherwise
    // (2) if s is less than the current node and the current node has no left child
    //     then create a new node containing s, call it newNode
    //          make the left child of the current node be the newNode
    //          make the parent of the newNode be the current node ... otherwise
    // (3) if s is greater than the current node and the current node has a right child
    //     then insert s into the tree rooted on the right child ... otherwise
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //


    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //
```

If s is in the right subtree ...

BSTree - Notepad

File Edit Format View Help

```
public class BSTree {
    private Node root;
    private int size;

    public BSTree(){root = null; size = 0;}

    public Node root(){return root;}
    public boolean isEmpty(){return root == null;}
    public int size(){return size;}

    public void insert(String s){}
    //
    // insert the string s into a tree
    // (1) if the tree is empty
    //     then create a new node with s in it and setthe size of the tree to be 1 ... otherwise
    // (2) insert the string s into the tree rooted on the current node ... see below
    //

    private void insert(String s,Node node){}
    //
    // insert the string s into the tree rooted on the current node
    // (1) if s is less than the current node and the current node has a left child
    //     then insert s into the tree rooted on the left child ... otherwise
    // (2) if s is less than the current node and the current node has no left child
    //     then create a new node containing s, call it newNode
    //          make the left child of the current node be the newNode
    //          make the parent of the newNode be the current node ... otherwise
    // (3) if s is greater than the current node and the current node has a right child
    //     then insert s into the tree rooted on the right child ... otherwise
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //


    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //
```

If s is in the right subtree and we need to create a new node

finding a node

BSTree - Notepad

File Edit Format View Help

```
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //


    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //

    private Node find(String s,Node node){return null;}
    //
    // given a node and a string s
    // (0) we have found s if s is equal to the data in the node otherwise ...
    // (1) if s is less than the data in the node and the node has a left child
    //     then search for s is in the tree rooted at the left child ... otherwise
    // (2) if s is greater than the data in the node and the node has a right child
    //     then search for s is in the tree rooted at the right child ... otherwise
    // (3) the string s is not in the tree!
    //

    public void delete(String s){}
    //
    // (0) find the node in the tree that contains s
    // (1) if not found then nothing to delete ... done!
    // (2) if the node is the root and the root is a leaf, make the tree empty ... otherwise
    // (3) if the node is the root and the root has a right child and no left child
    //     then make the right child the root of the tree ... otherwise
    // (4) if the node is the root and the root has a left child and no right child
    //     then make the left child the new root of the tree ... otherwise
    // (5) delete the node using the steps in method delete(node) below
    // (6) Regardless, in cases (1) to (5), once done decrement the size counter
    //

    private void delete(Node node){}
    //
    // (1)   if the node is internal, i.e. has a left and right child
    // (1.1) then find the smallest node in the right subtree, call this minNode
    // (1.2) replace the contents of the node with the contents of the minNode
```

BSTree - Notepad

File Edit Format View Help

```
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //

    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //

    private Node find(String s,Node node){return null;}
    //
    // given a node and a string s
    // (0) we have found s if s is equal to the data in the node otherwise ...
    // (1) if s is less than the data in the node and the node has a left child
    //     then search for s is in the tree rooted at the lef
    // (2) if s is greater than the data in the node and the
    //     then search for s is in the tree rooted at the rig
    // (3) the string s is not in the tree!
    //

    public void delete(String s){}
    //
    // (0) find the node in the tree that contains s
    // (1) if not found then nothing to delete ... done!
    // (2) if the node is the root and the root is a leaf, make the tree empty ... otherwise
    // (3) if the node is the root and the root has a right child and no left child
    //     then make the right child the root of the tree ... otherwise
    // (4) if the node is the root and the root has a left child and no right child
    //     then make the left child the new root of the tree ... otherwise
    // (5) delete the node using the steps in method delete(node) below
    // (6) Regardless, in cases (1) to (5), once done decrement the size counter
    //

    private void delete(Node node){}
    //
    // (1)   if the node is internal, i.e. has a left and right child
    // (1.1) then find the smallest node in the right subtree, call this minNode
    // (1.2) replace the contents of the node with the contents of the minNode
```

The string s is present in the tree if the tree isn't empty and we can ***find*** a node that contains s

finding a node

BSTree

BSTree - Notepad

File Edit Format View Help

```
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //

    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //

    private Node find(String s,Node node){return null;}
    //
    // given a node and a string s
    // (0) we have found s if s is equal to the data in the node otherwise ...
    // (1) if s is less than the data in the node and the node has a left child
    //     then search for s is in the tree rooted at the left child ... otherwise
    // (2) if s is greater than the data in the node and the node has a right child
    //     then search for s is in the tree rooted at the right child ... otherwise
    // (3) the string s is not in the tree!
    //

    public void delete(String s){}
    //
    // (0) find the node in the tree that contains s
    // (1) if not found then nothing to delete ... done!
    // (2) if the node is the root and the root is a leaf, make the tree empty ... otherwise
    // (3) if the node is the root and the root has a right child and no left child
    //     then make the right child the root of the tree ... otherwise
    // (4) if the node is the root and the root has a left child and no right child
    //     then make the left child the new root of the tree ... otherwise
    // (5) delete the node using the steps in method delete(node) below
    // (6) Regardless, in cases (1) to (5), once done decrement th
    //

    private void delete(Node node){}
    //
    // (1)   if the node is internal, i.e. has a left and right child
    // (1.1) then find the smallest node in the right subtree, call this minNode
    // (1.2) replace the contents of the node with the contents of the minNode
```

find string s in the subtree rooted on the current node

finding a node

BSTree

BSTree - Notepad

File Edit Format View Help

```
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //


    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //

    private Node find(String s,Node node){return null;}
    //
    // given a node and a string s
    // (0) we have found s if s is equal to the data in the node otherwise ...
    // (1) if s is less than the data in the node and the node has a left child
    //     then search for s is in the tree rooted at the left child ... otherwise
    // (2) if s is greater than the data in the node and the node has a right child
    //     then search for s is in the tree rooted at the right child ... otherwise
    // (3) the string s is not in the tree!
    //

    public void delete(String s){}
    //
    // (0) find the node in the tree that contains s
    // (1) if not found then nothing to delete ... done!
    // (2) if the node is the root and the root is a leaf, make the tree empty ... otherwise
    // (3) if the node is the root and the root has a right child and no left child
    //     then make the right child the root of the tree ... otherwise
    // (4) if the node is the root and the root has a left child and no right child
    //     then make the left child the new root of the tree ... otherwise
    // (5) delete the node using the steps in method delete(node) below
    // (6) Regardless, in cases (1) to (5), once done decrement th
    //

    private void delete(Node node){}
    //
    // (1)   if the node is internal, i.e. has a left and right ch
    // (1.1) then find the smallest node in the right subtree, cal
    // (1.2) replace the contents of the node with the contents of the minNode
```

If s is equal to the data in the current node then return the current node as a result.

BSTree - Notepad

File Edit Format View Help

```
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //


    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //

    private Node find(String s,Node node){return null;}
    //
    // given a node and a string s
    // (0) we have found s if s is equal to the data in the node otherwise
    // (1) if s is less than the data in the node and the node has a left child
    //     then search for s is in the tree rooted at the left child ... otherwise
    // (2) if s is greater than the data in the node and the node has a right child
    //     then search for s is in the tree rooted at the right child ... otherwise
    // (3) the string s is not in the tree!
    //

    public void delete(String s){}
    //
    // (0) find the node in the tree that contains s
    // (1) if not found then nothing to delete ... done!
    // (2) if the node is the root and the root is a leaf, make the tree empty ... otherwise
    // (3) if the node is the root and the root has a right child and no left child
    //     then make the right child the root of the tree ... otherwise
    // (4) if the node is the root and the root has a left child and no right child
    //     then make the left child the new root of the tree ... otherwise
    // (5) delete the node using the steps in method
    // (6) Regardless, in cases (1) to (5), once don
    //

    private void delete(Node node){}
    //
    // (1)   if the node is internal, i.e. has a lef
    // (1.1) then find the smallest node in the righ
    // (1.2) replace the contents of the node with the contents of the minNode
```

If s is less than the data in the current node and the current node has a left child then try and find s in the subtree rooted on the left child

finding a node

BSTree - Notepad

File Edit Format View Help

```
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //


    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //

    private Node find(String s,Node node){return null;}
    //
    // given a node and a string s
    // (0) we have found s if s is equal to the data in the node otherwise ...
    // (1) if s is less than the data in the node and the node has a left child
    //     then search for s is in the tree rooted at the left child ... otherwise
    // (2) if s is greater than the data in the node and the node has a right child
    //     then search for s is in the tree rooted at the right child ... otherwise
    // (3) the string s is not in the tree!
    //

    public void delete(String s){}
    //
    // (0) find the node in the tree that contains s
    // (1) if not found then nothing to delete ... done!
    // (2) if the node is the root and the root is a leaf, make the tree empty ... otherwise
    // (3) if the node is the root and the root has a right child and no left child
    //     then make the right child the root of the tree ... otherwise
    // (4) if the node is the root and the root has a left child and no right child
    //     then make the left child the new root of the tree ... otherwise
    // (5) delete the node using the steps in method
    // (6) Regardless, in cases (1) to (5), once don
    //

    private void delete(Node node){}
    //
    // (1)   if the node is internal, i.e. has a lef
    // (1.1) then find the smallest node in the righ
    // (1.2) replace the contents of the node with the contents of the minNode
```

If s is greater than the data in the current node and the current node has *a right* child then try and find s in the subtree rooted on the right child

finding a node

BSTree

BSTree - Notepad

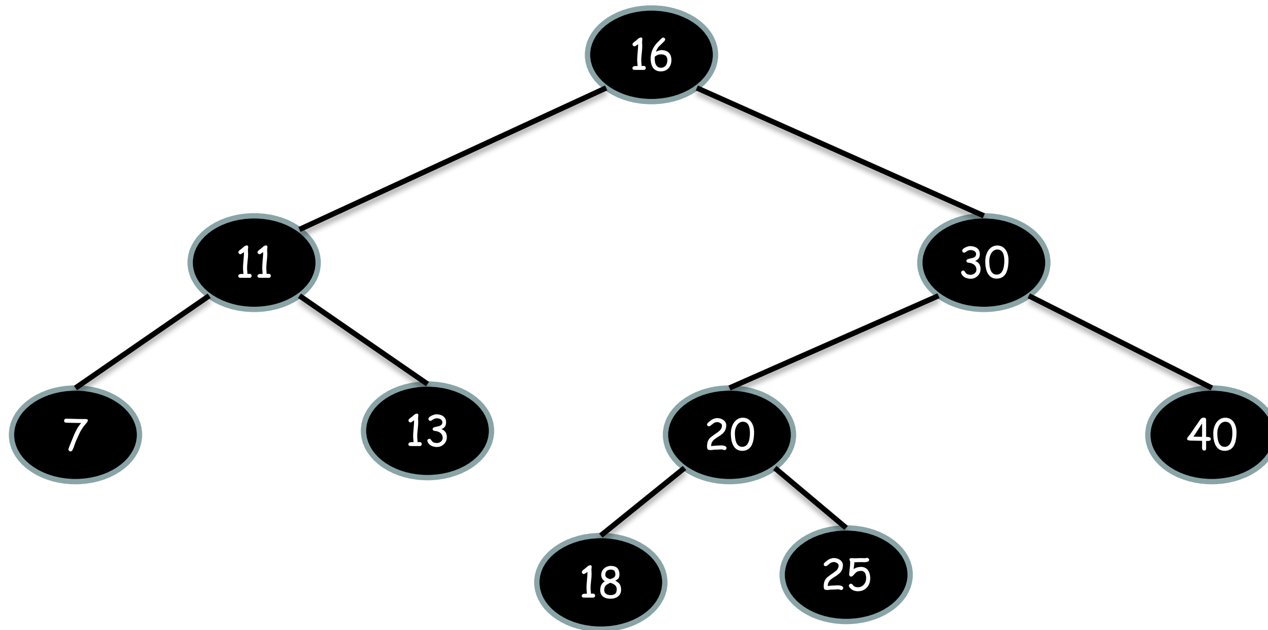
File Edit Format View Help

```
    // (4) if s is greater than the current node and the current node has no right child
    //     then create a new node containing s, call it newNode
    //          make the right child of the current node be the newNode
    //          make the parent of the newNode be the current node
    //


    public boolean isPresent(String s){return root != null && find(s,root)!= null;}
    //
    // s is present if the tree isn't empty and we can find a node that contains s
    //

    private Node find(String s,Node node){return null;}
    //
    // given a node and a string s
    // (0) we have found s if s is equal to the data in the node otherwise ...
    // (1) if s is less than the data in the node and the node has a left child
    //     then search for s is in the tree rooted at the left child ... otherwise
    // (2) if s is greater than the data in the node and the node has a right child
    //     then search for s is in the tree rooted at the right child ... otherwise
    // (3) the string s is not in the tree!
    //

    public void delete(String s){}
    //
    // (0) find the node in the tree that contains s
    // (1) if not found then nothing to delete ... done!
    // (2) if the node is the root and the root is a leaf, make the tree empty ... otherwise
    // (3) if the node is the root and the root has a right child and no left child
    //     then make the right child the root of the tree ... otherwise
    // (4) if the node is the root and the root has a left child and no right child
    //     then make the left child the new root of the tree ... otherwise
    // (5) delete the nod
    // (6) Regardless, in
    //

    private void delete(N
    //
    // (1)   if the node
    // (1.1) then find the smallest node in the right subtree, call this minNode
    // (1.2) replace the contents of the node with the contents of the minNode
```

s is less than the current node and it has no left child or s is greater than the current node and it has no right child … deliver null!

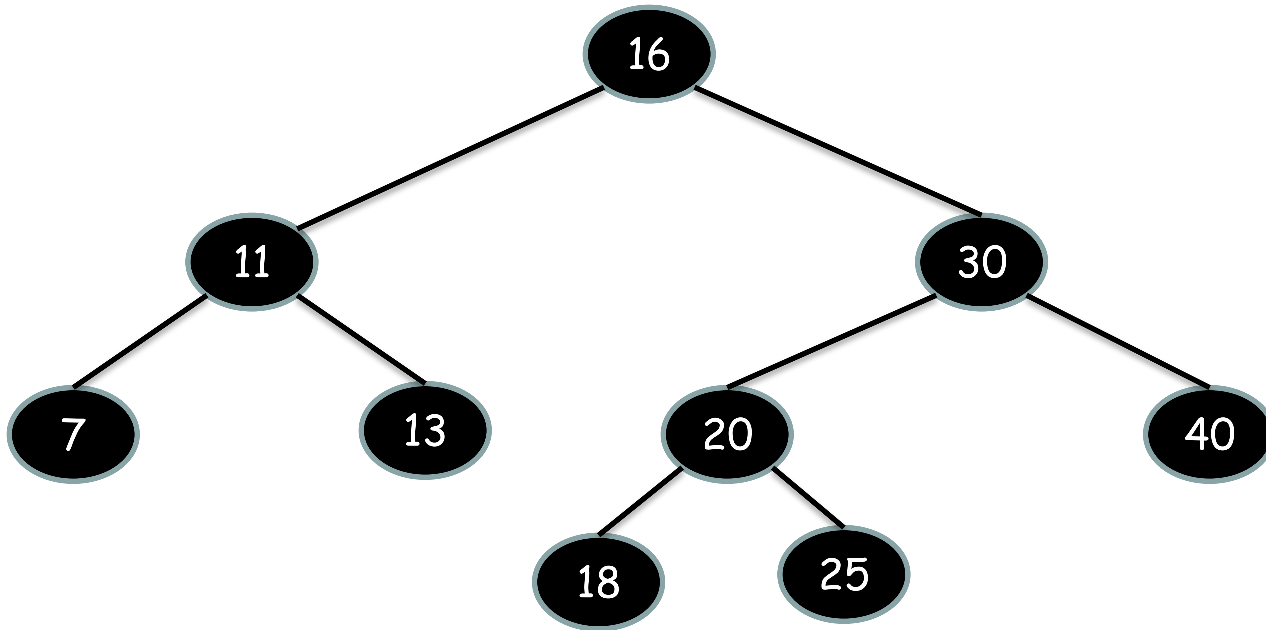
Deletion of a node

Deletion of a node



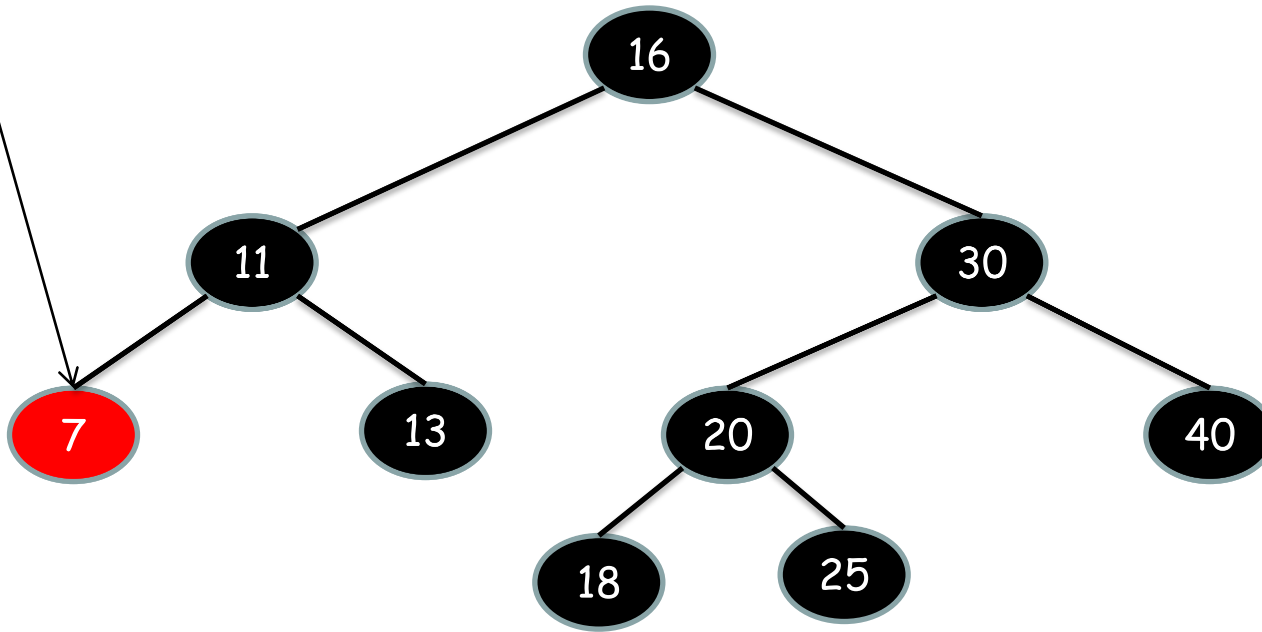
Delete 7

Deletion of a node



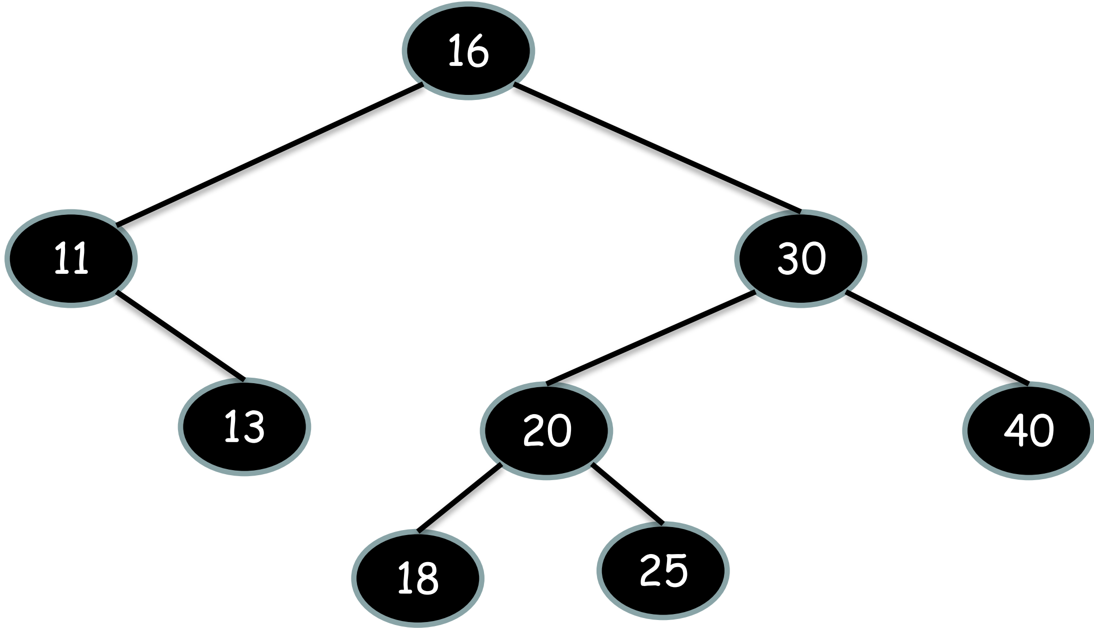
Deletion of a node

Delete 7



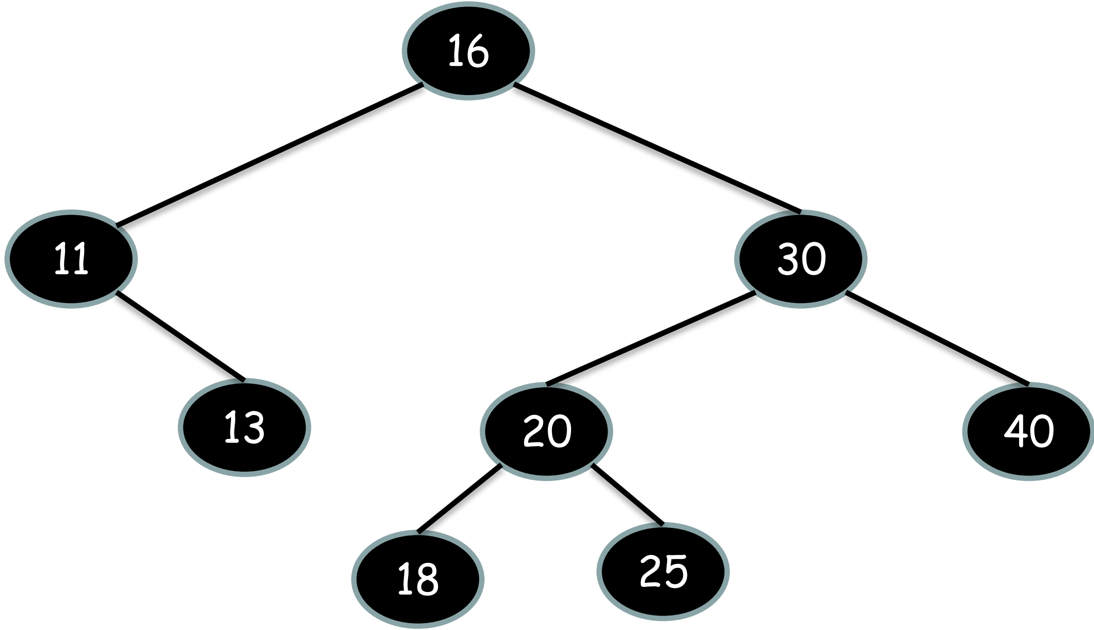
Delete 7

Deletion of a node



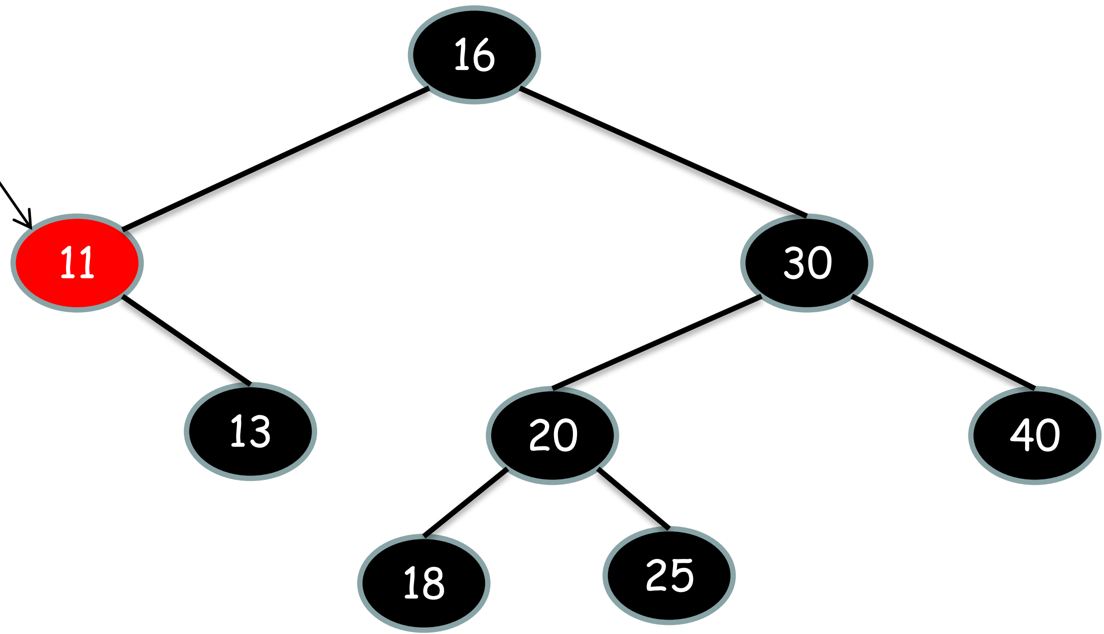
Delete 11

Deletion of a node



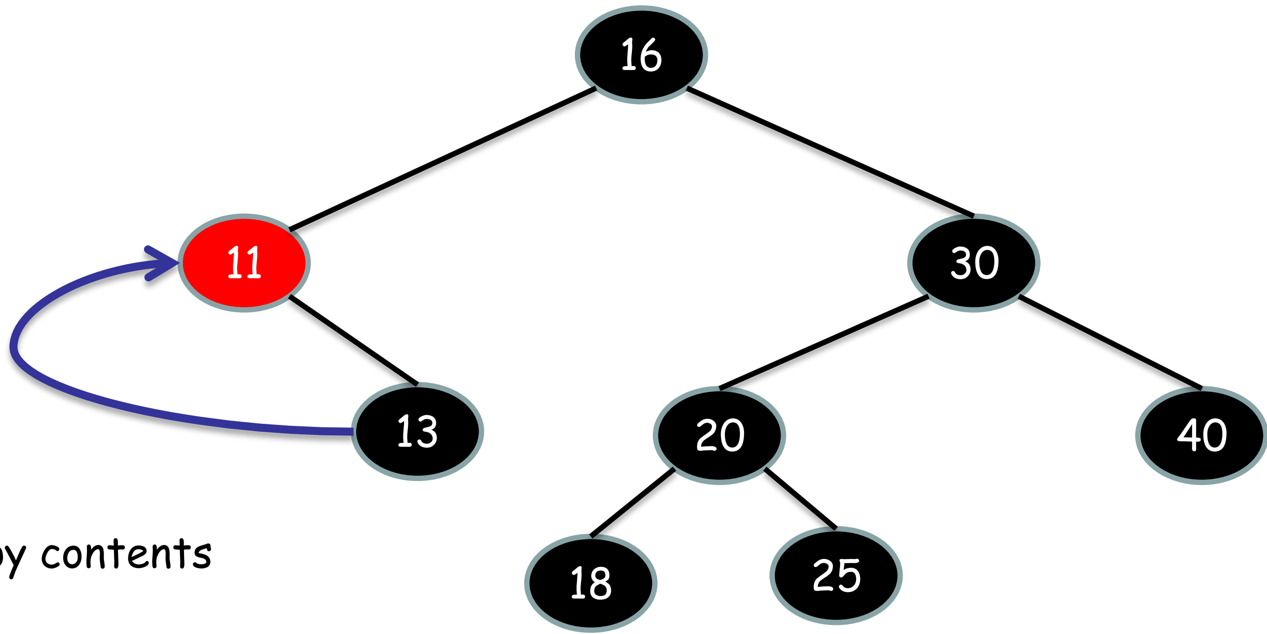
Deletion of a node

Delete 11



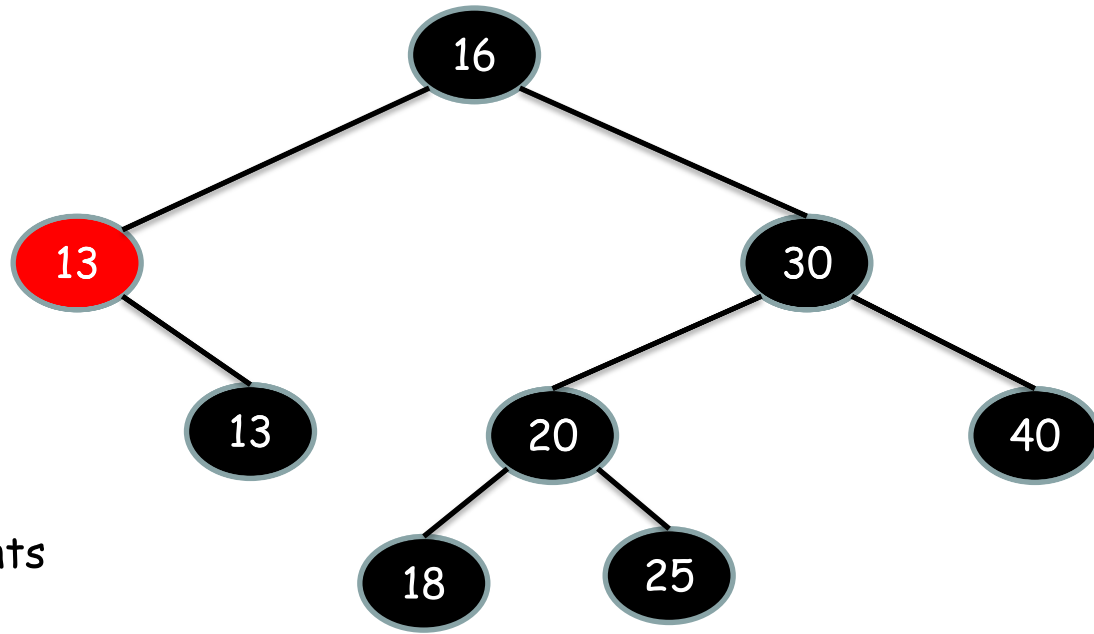
Delete 11

Deletion of a node



Delete 11

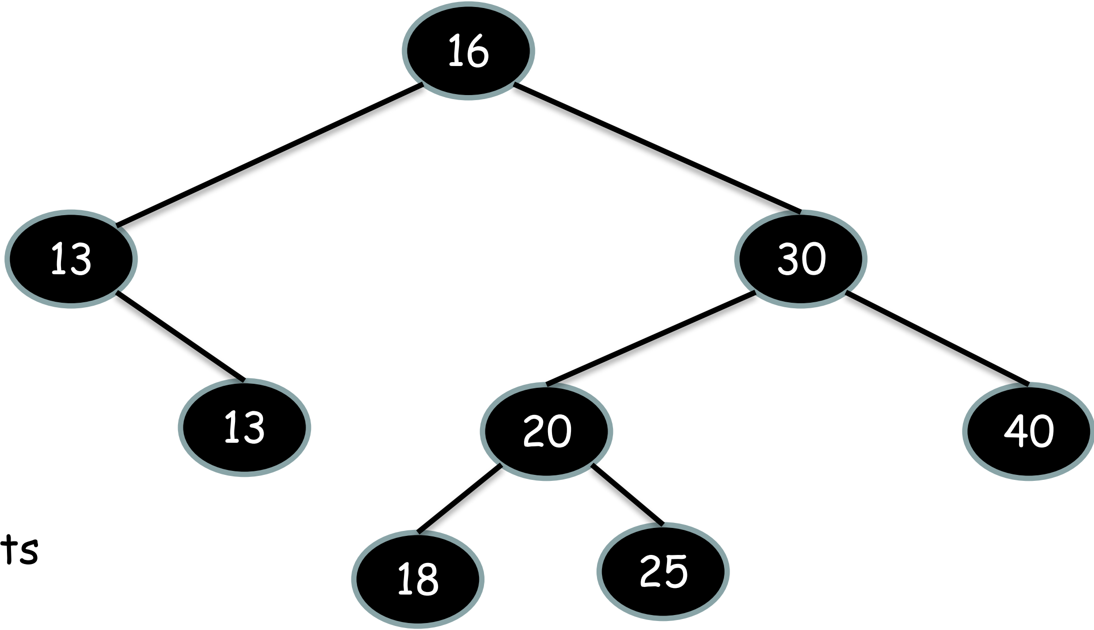
Deletion of a node



Copy contents

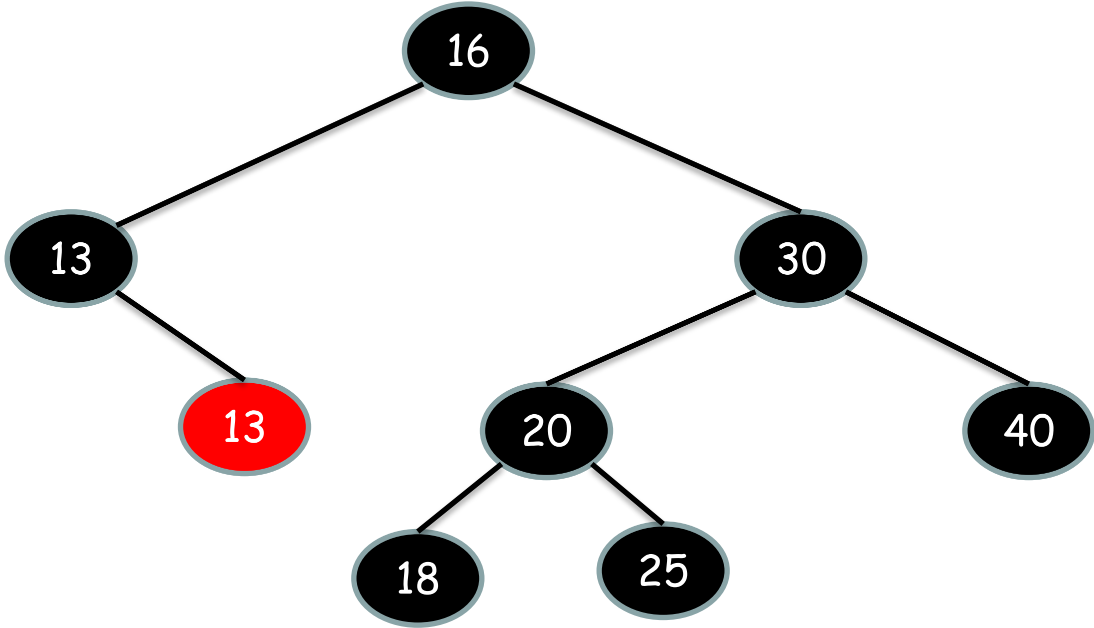
Delete 11

Deletion of a node



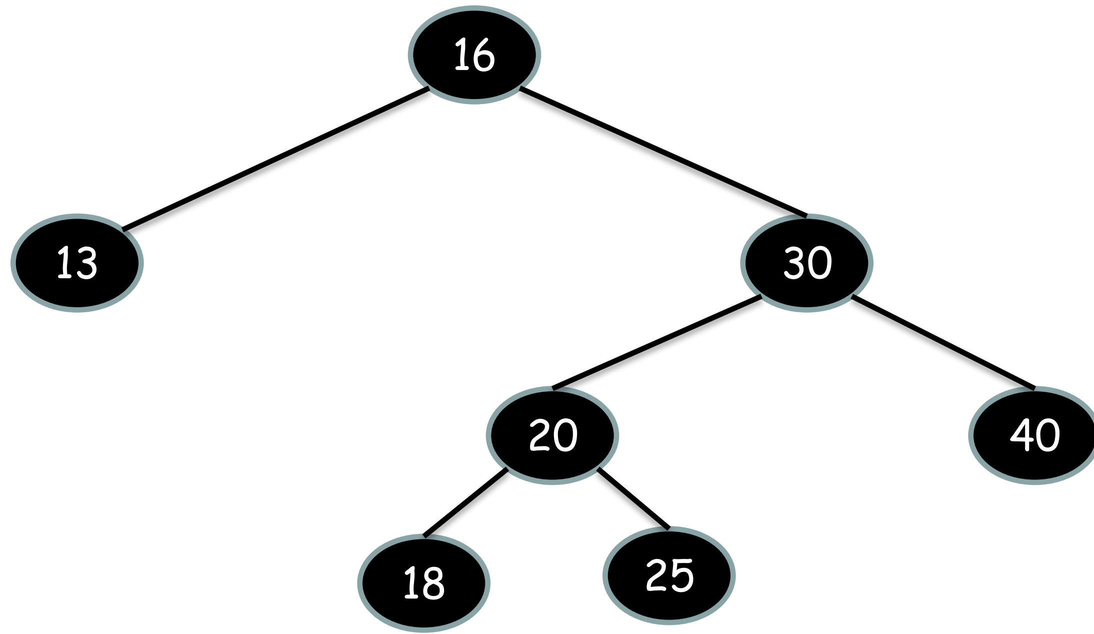
Copy contents

Delete 11



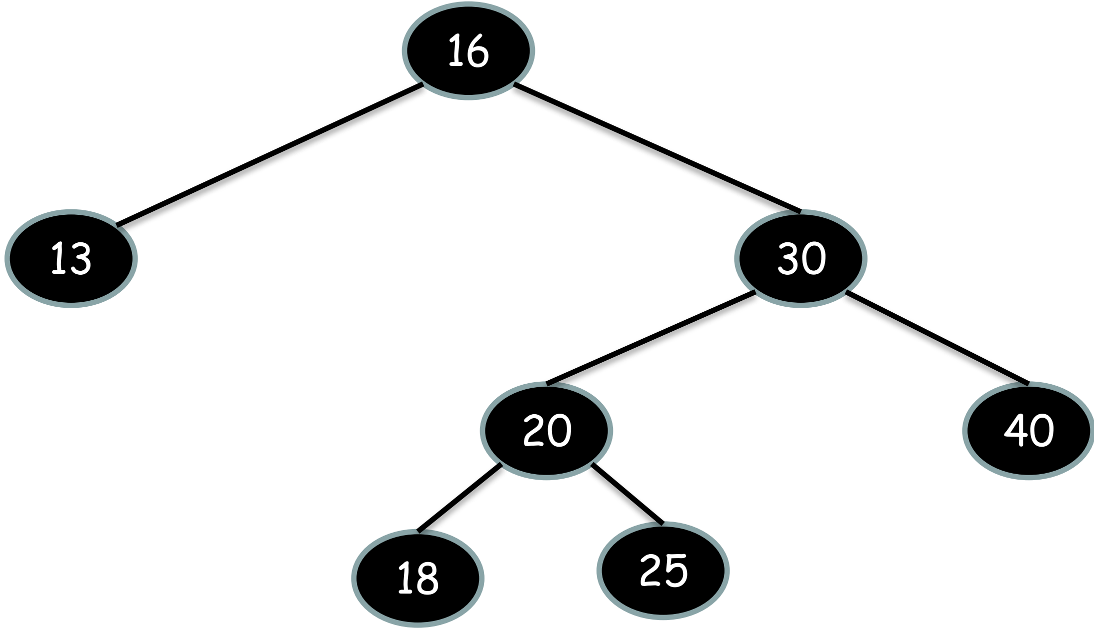
Delete leaf

Delete 11



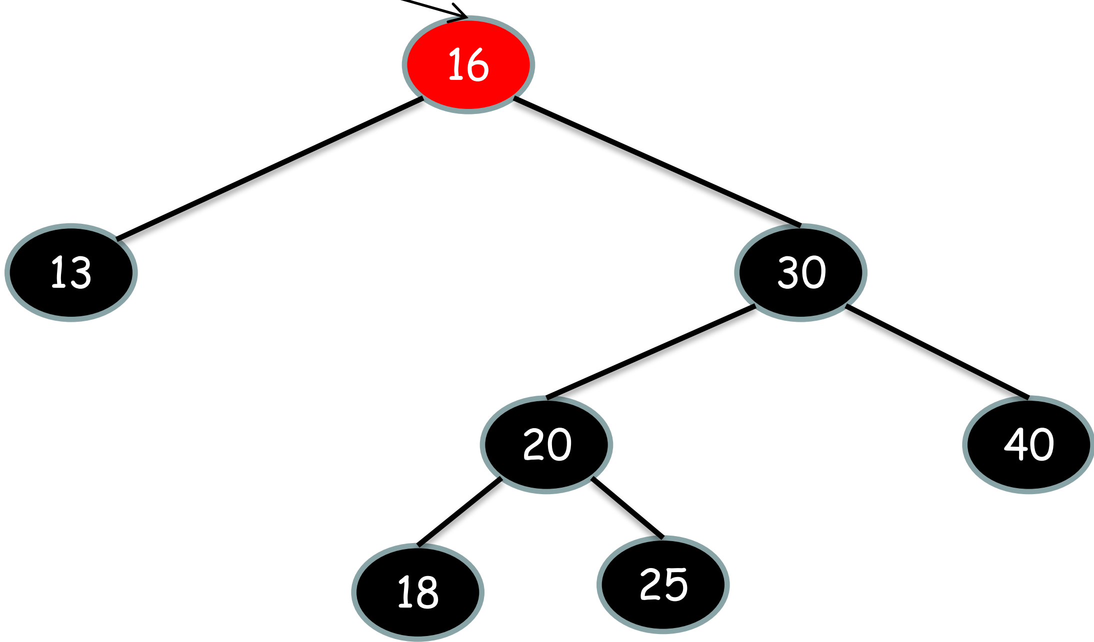
Delete 16

Deletion of a node



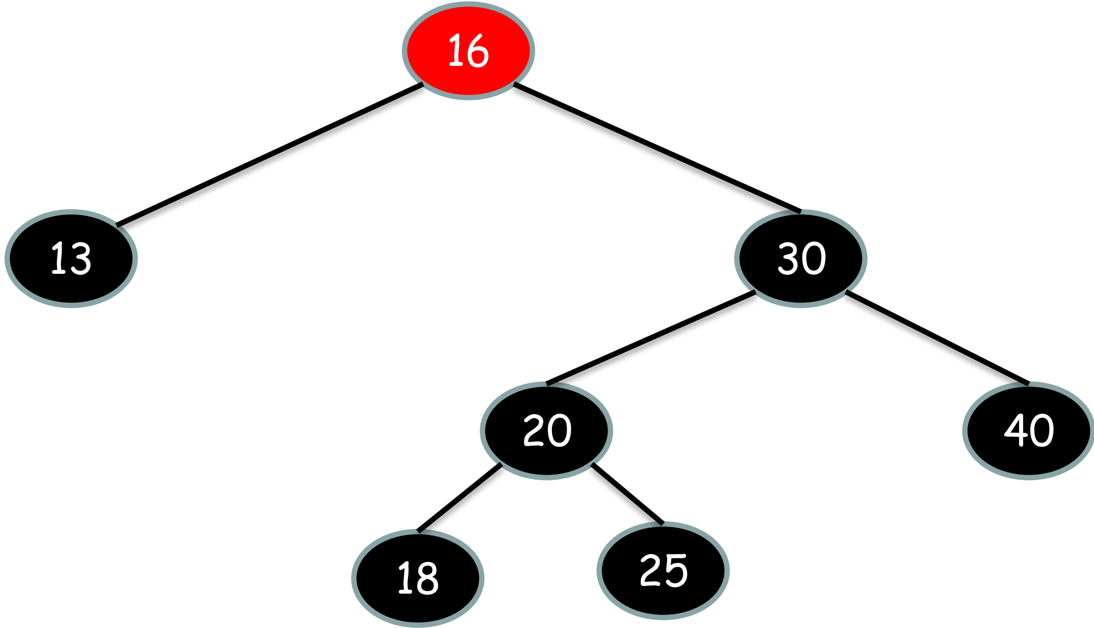
Delete 16

Deletion of a node



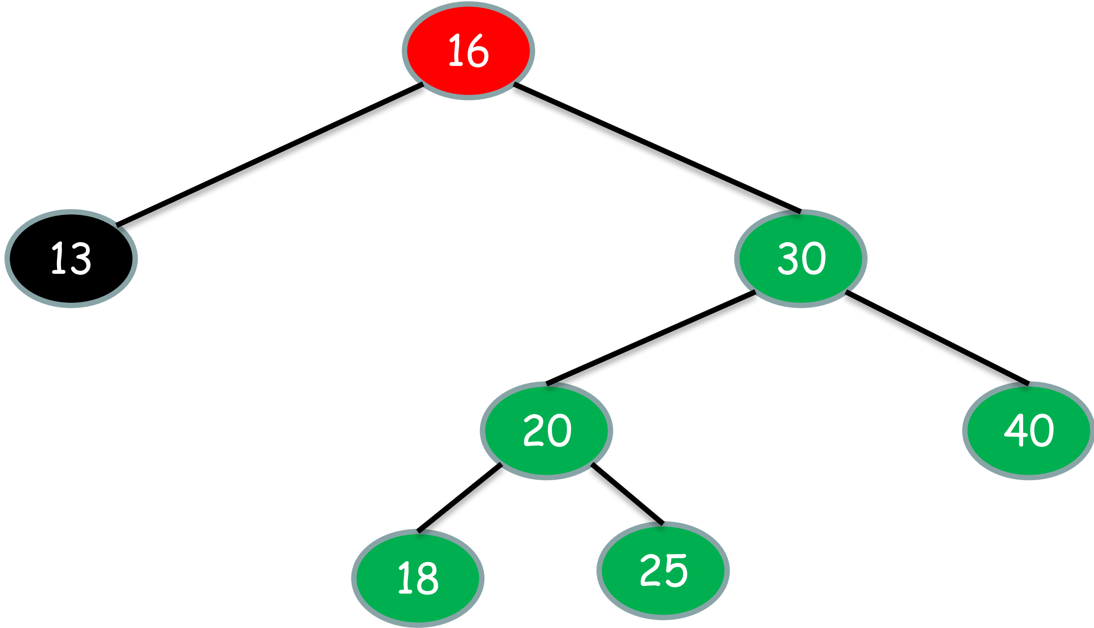
Delete 16

Deletion of a node



Find the smallest node in the right subtree

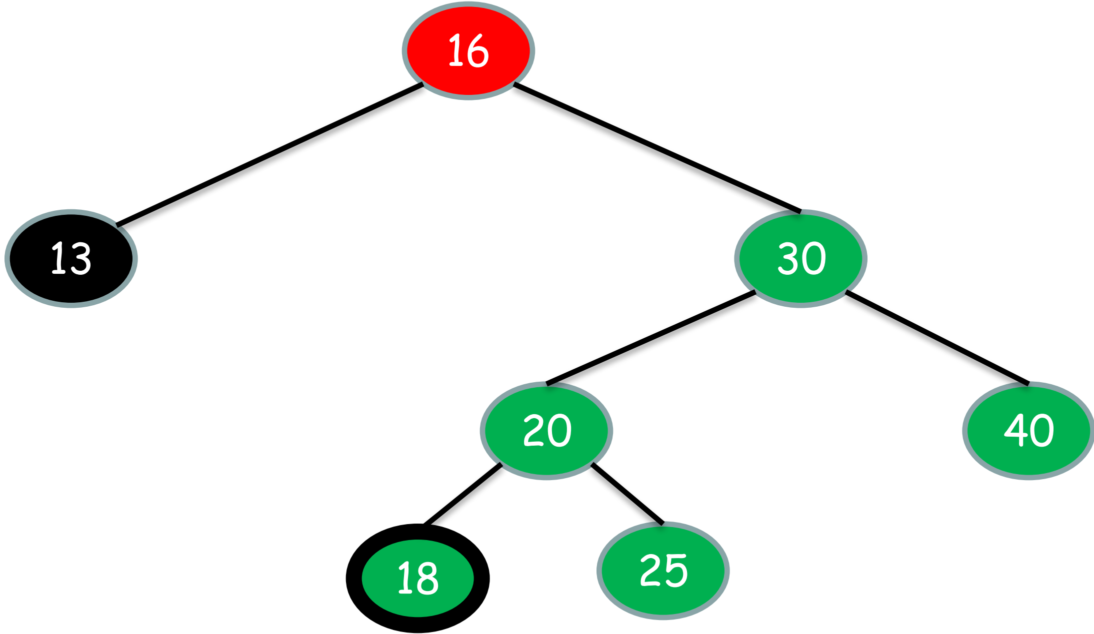
Delete 16



Find the smallest node in the right subtree

Delete 16

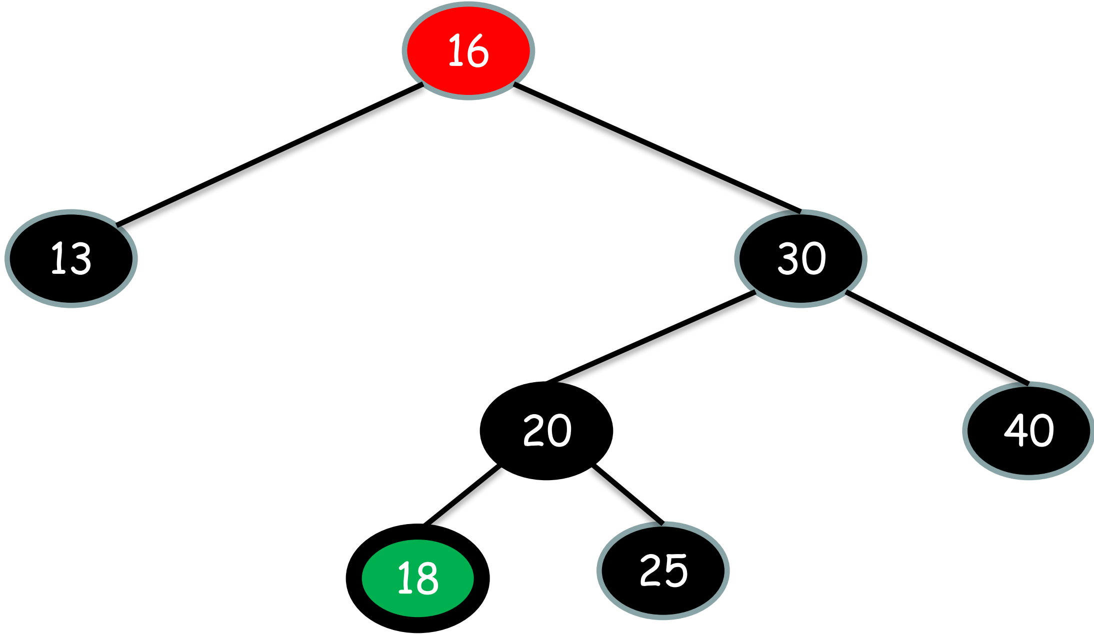
Deletion of a node



Find the smallest node in the right subtree

Delete 16

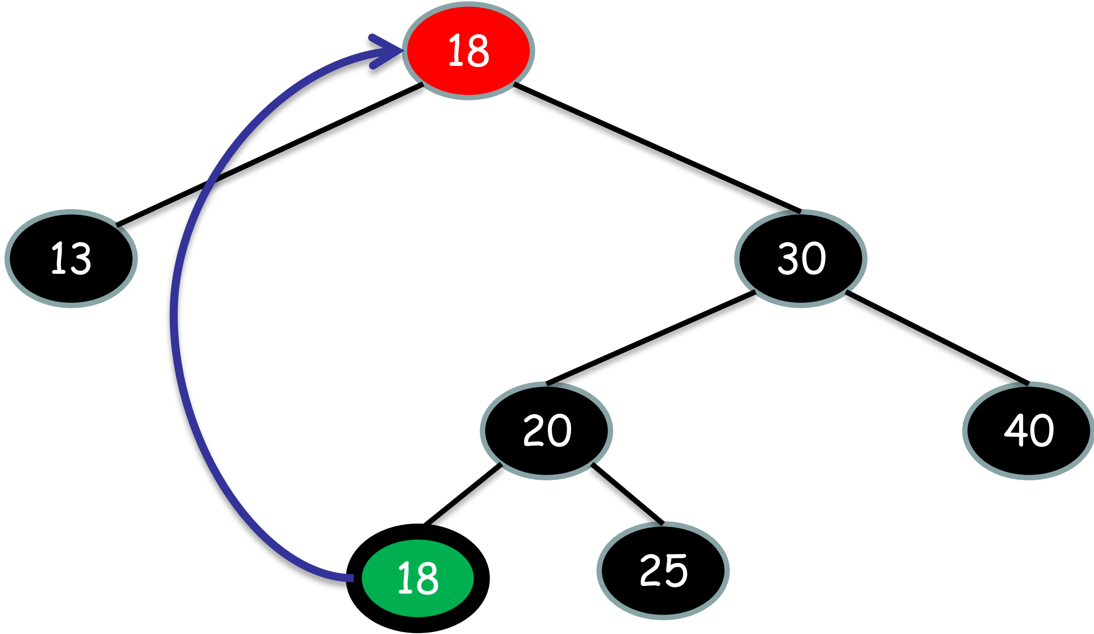
Deletion of a node



Copy contents

Delete 16

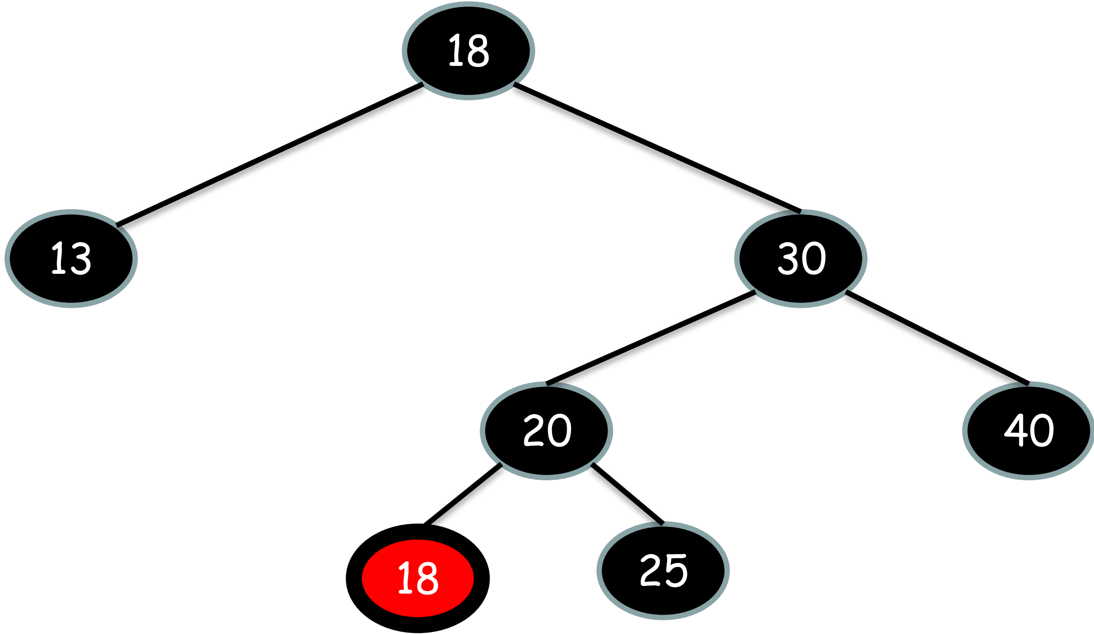
Deletion of a node



Copy contents

Delete 16

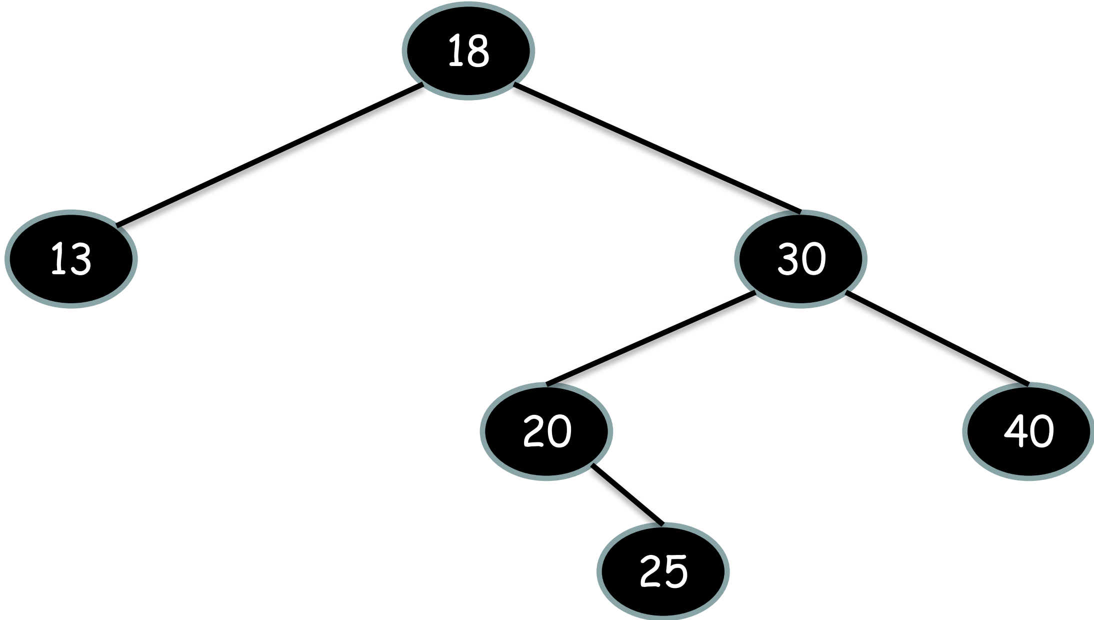
Deletion of a node



Delete leaf

Delete 16

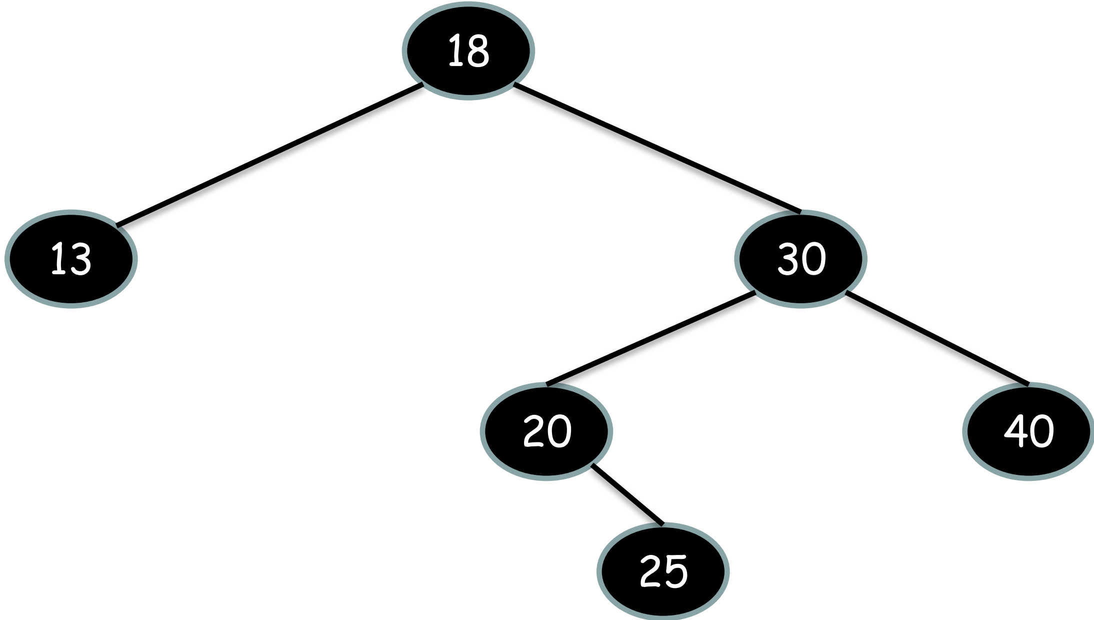
Deletion of a node



Delete leaf

Delete 16

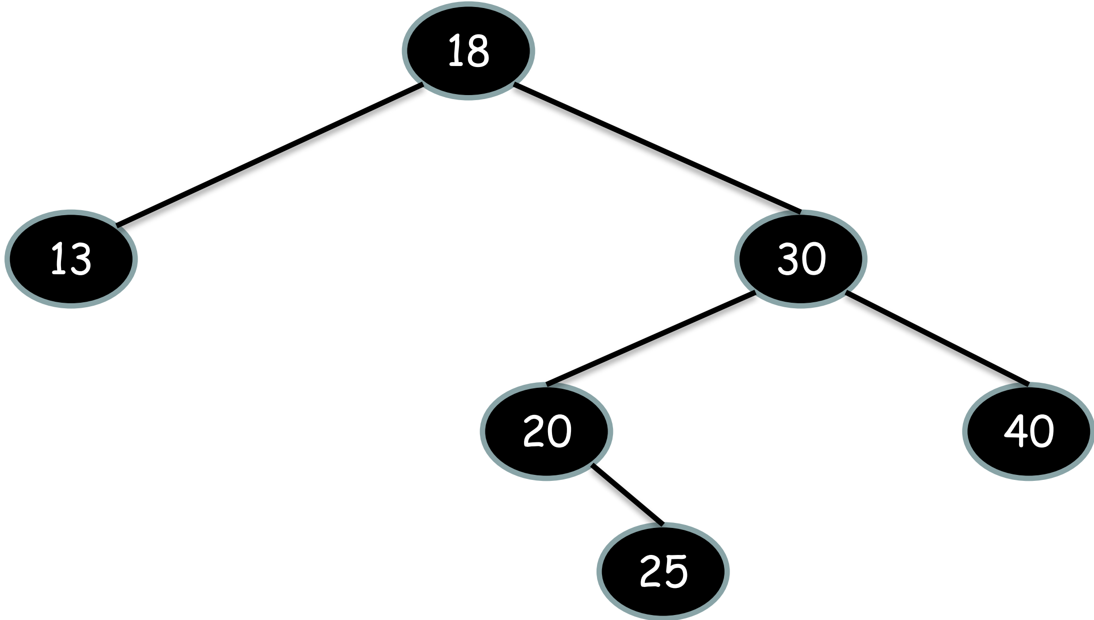
Deletion of a node



This process maintains the inorder property

Delete 16

Deletion of a node

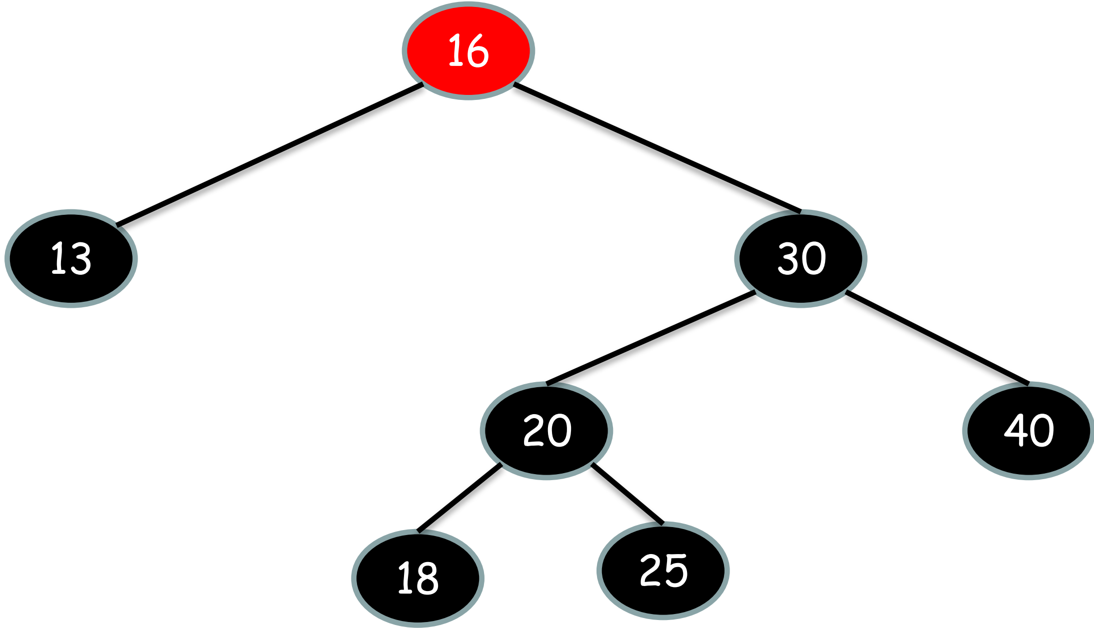


This process maintains the inorder property

There is a symmetric equivalent: find largest node in left branch …

Delete 16

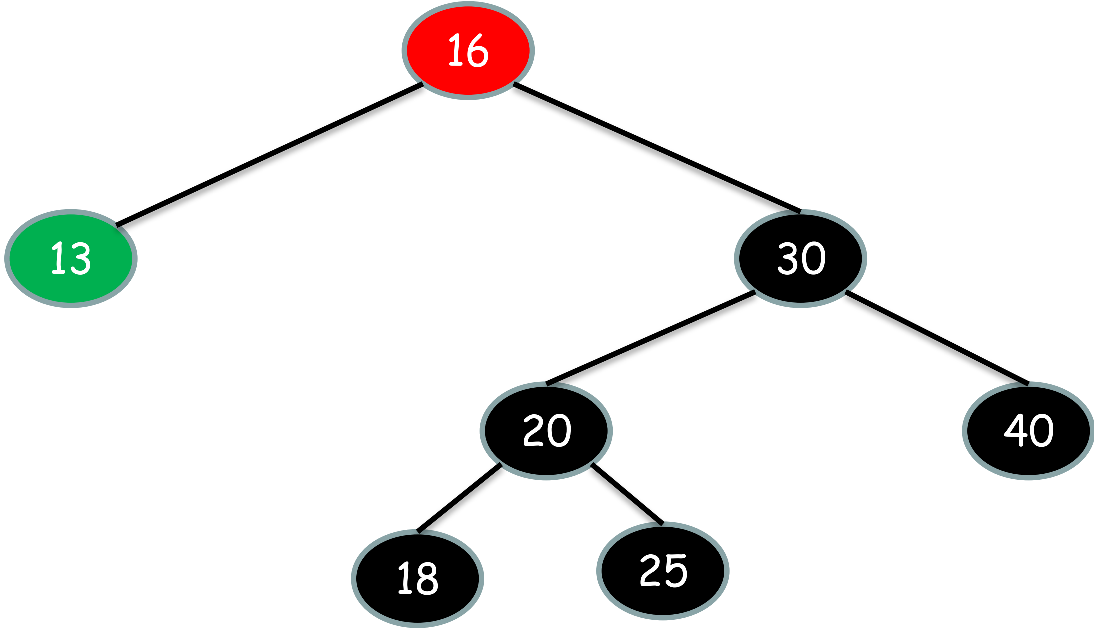
Deletion of a node



Find the ***largest*** node in the ***left*** subtree

Delete 16

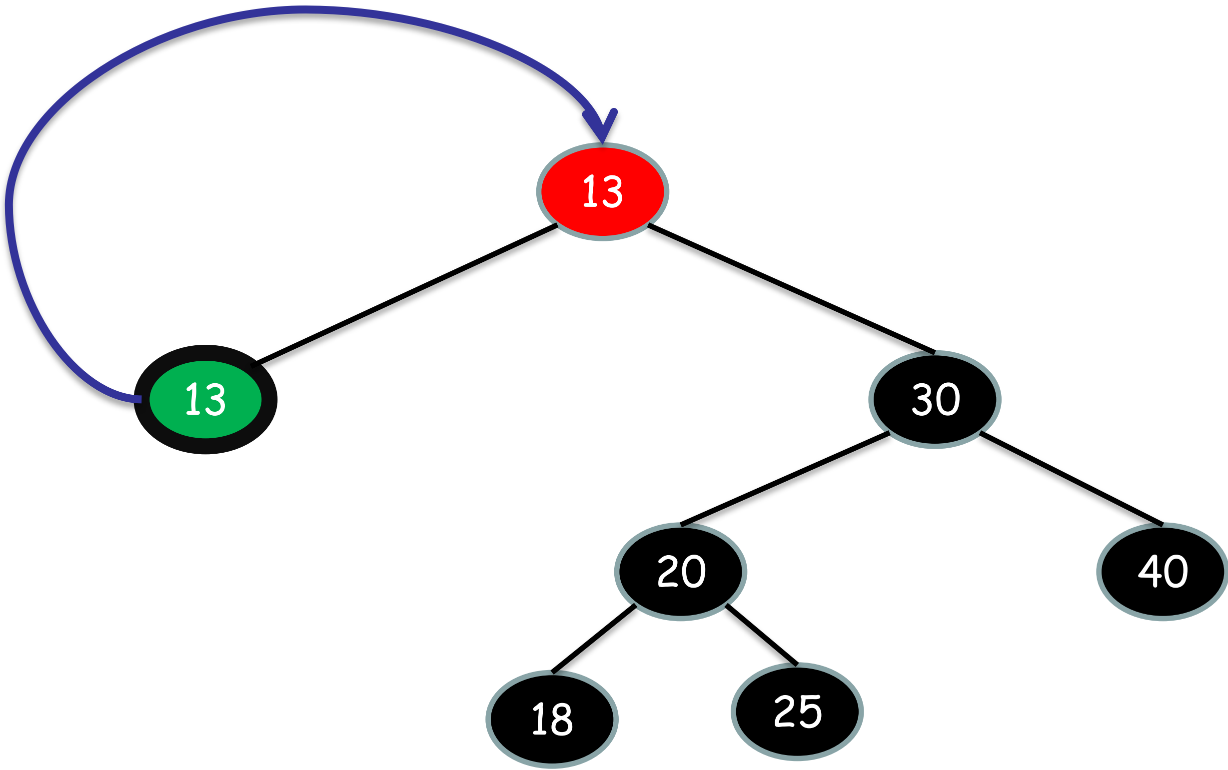
Deletion of a node



Find the ***largest*** node in the ***left*** subtree

Delete 16

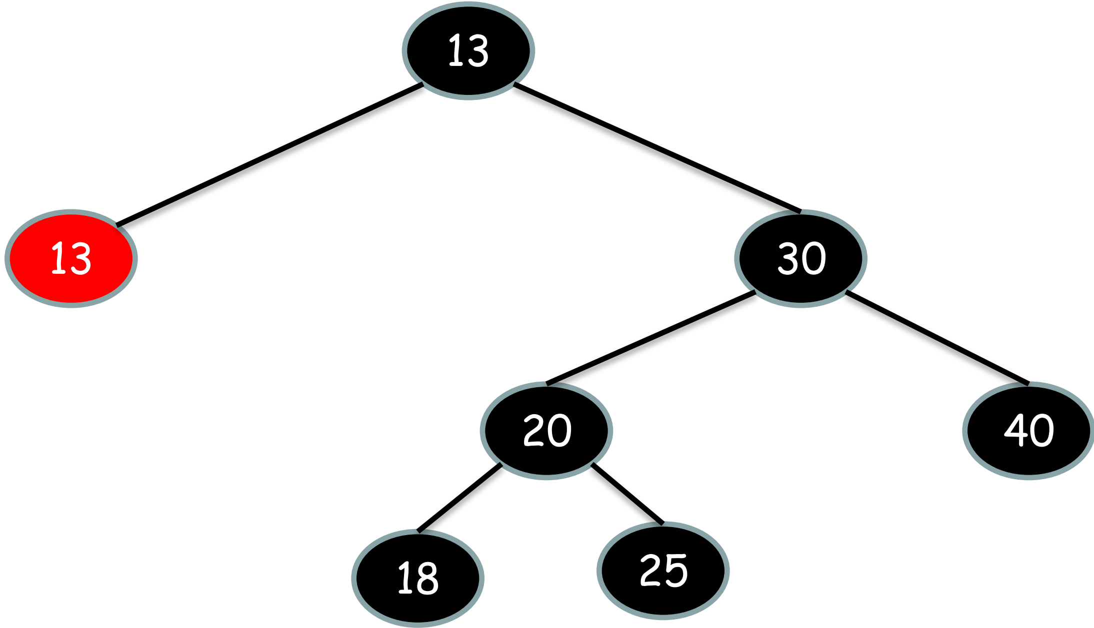
Deletion of a node



Copy contents

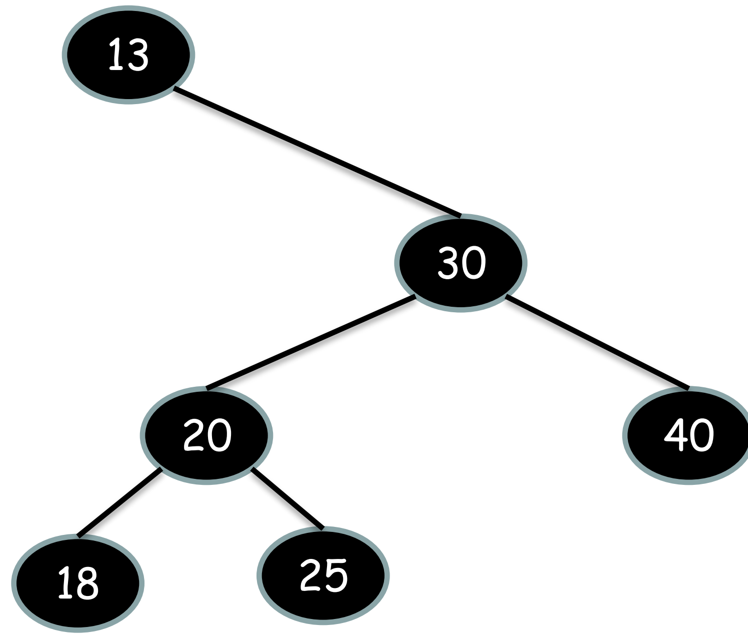
Delete 16

Deletion of a node



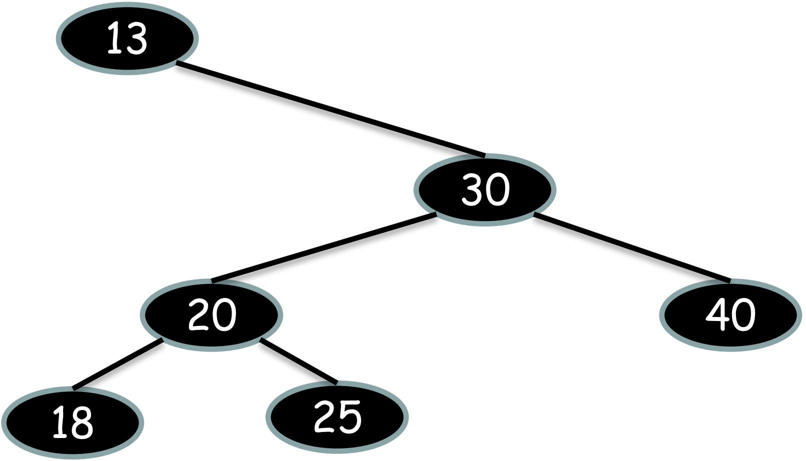
Delete leaf

Delete 16

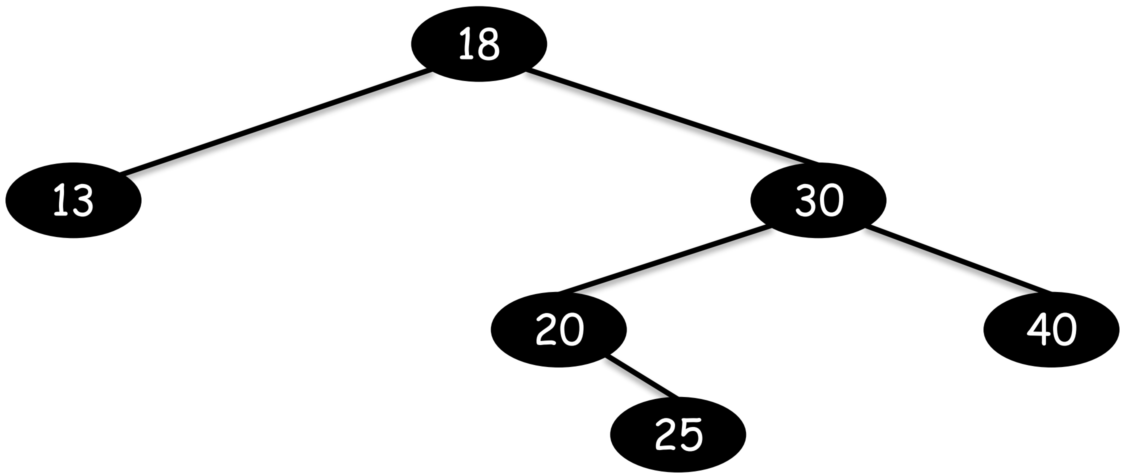


Delete leaf

Delete 16



Therefore we have 2 options resulting in two different trees, both valid



Well baby, is there an algorithm for this?

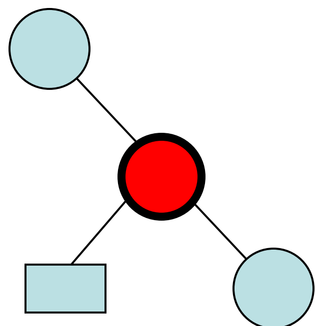
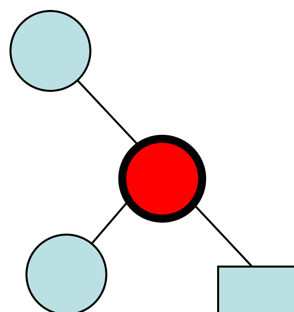
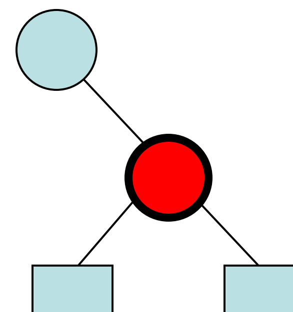
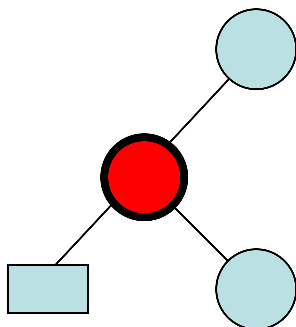
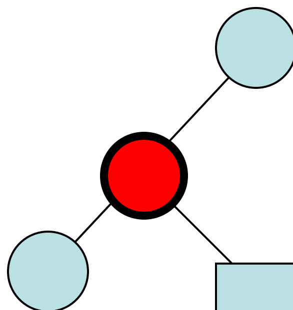
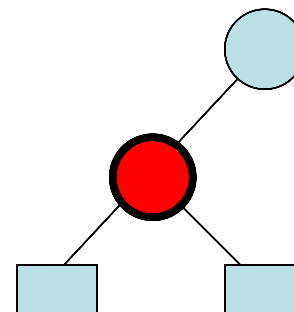
Deletion of a node

BSTree - Notepad

File Edit Format View Help

```
    //

    public void delete(String s){}
    //
    // (0) find the node in the tree that contains s
    // (1) if not found then nothing to delete ... done!
    // (2) if the node is the root and the root is a leaf, make the tree empty ... otherwise
    // (3) if the node is the root and the root has a right child and no left child
    //     then make the right child the root of the tree ... otherwise
    // (4) if the node is the root and the root has a left child and no right child
    //     then make the left child the new root of the tree ... otherwise
    // (5) delete the node using the steps in method delete(node) below
    // (6) Regardless, in cases (1) to (5), once done decrement the size counter
    //

    private void delete(Node node){}
    //
    // (1)   if the node is internal, i.e. has a left and right child
    // (1.1) then find the smallest node in the right subtree, call this minNode
    // (1.2) replace the contents of the node with the contents of the minNode
    //       NOTE: this preserves inorder property
    // (1.3) minNode is NOT internal, therfore deleteNotInternal(minNode)
    // (2)   node is not internal, therefore deleteNotInternal(node)
    //

    private Node getMin(Node node){return null;}
    //
    // deliver the node with smallest element in the subtree rooted on node
    // (1) if node has a left child
    //     then find the smallest node in the tree rooted on the left child ... otherwise
    // (2) node has no left child and is therefore the smallest child.
    //     Deliver that node as a result
    //

    private void deleteNotInternal(Node node){}
    // (0) the node is a leaf or has one child
    // (1) get the parent of the current node to be deleted, and call it v
    // (2) if the node is a right child and has no left child of its own
    //     then the parent's right child is now the current node's right child ... otherwise
    // (3) if the node is a right child and has no right child of its own
    //     then the parent's right child becomes the current node's left child ... otherwise
```

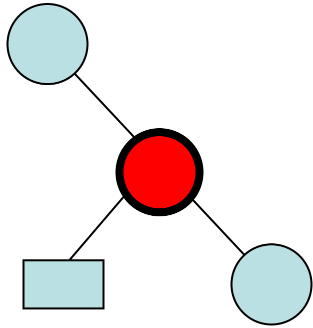
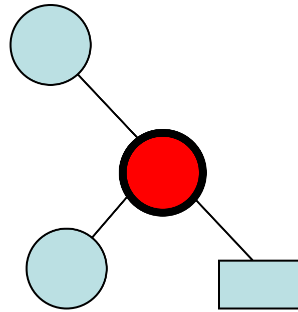
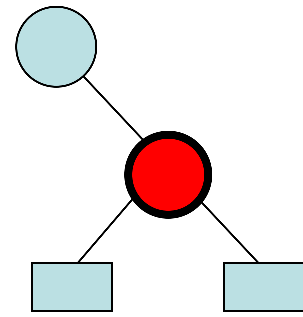
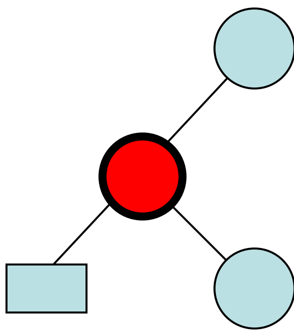
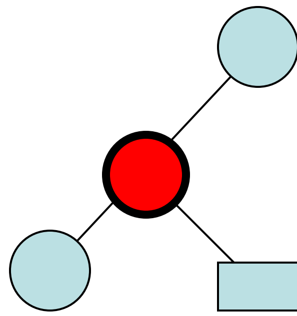
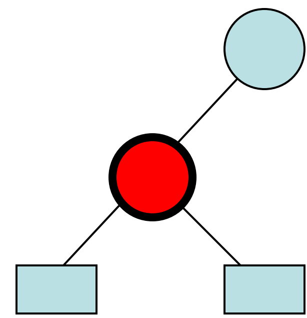
Deletion of a non-internal node (a node with less than 2 children)

Non-internal – less than 2 children

6 cases to consider

Deletion of a non-internal node

Non-internal – less than 2 children

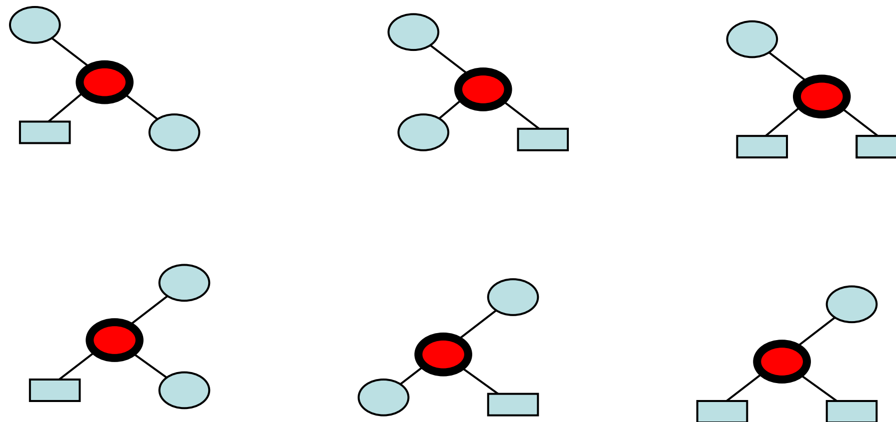
Actually only 4 cases to consider

Deletion of a non-internal node

BSTree - Notepad

File Edit Format View Help

```
private void deleteNotInternal(Node node){}
// (0) the node is a leaf or has one child
// (1) get the parent of the current node to be deleted, and call it v
// (2) if the node is a right child and has no left child of its own
//     then the parent's right child is now the current node's right child ... otherwise
// (3) if the node is a right child and has no right child of its own
//     then the parent's right child becomes the current node's left child ... otherwise
// (4) if the node is a left child and has no left child of its own
//     then the parent's left child becomes the current node's right child ... otherwise
// (5) the node is a left child and has no right child of its own
//     consequently the parent's left child becomes the current node's left child
//
```



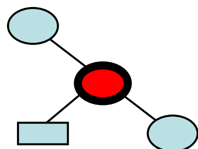
Assuming we have already found the node to delete

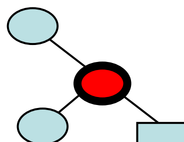
Deletion of a non-internal node

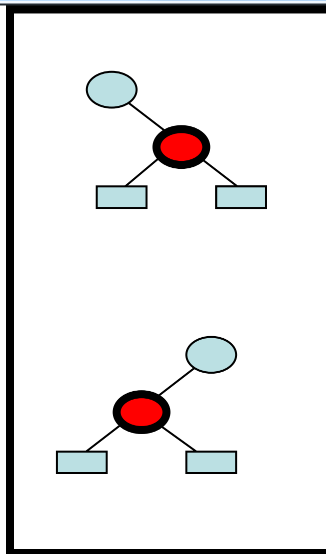
BSTree - Notepad

File Edit Format View Help

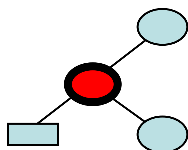
```
private void deleteNotInternal(Node node){}
// (0) the node is a leaf or has one child
// (1) get the parent of the current node to be deleted, and call it v
// (2) if the node is a right child and has no left child of its own
//     then the parent's right child is now the current node's right child ... otherwise
// (3) if the node is a right child and has no right child of its own
//     then the parent's right child becomes the current node's left child ... otherwise
// (4) if the node is a left child and has no left child of its own
//     then the parent's left child becomes the current node's right child ... otherwise
// (5) the node is a left child and has no right child of its own
//     consequently the parent's left child becomes the current node's left child
//
```

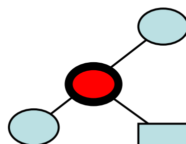






subsumed





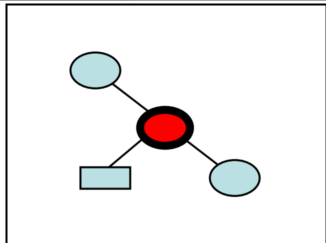
Assuming we have already found the node to delete

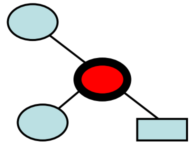
Deletion of a non-internal node

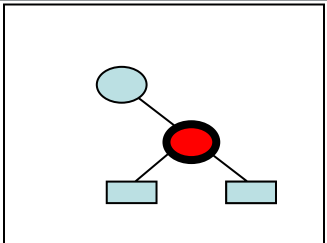
BSTree - Notepad

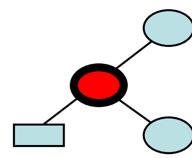
File Edit Format View Help

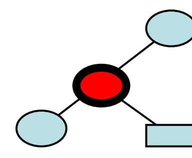
```
private void deleteNotInternal(Node node){}
// (0) the node is a leaf or has one child
// (1) get the parent of the current node to be deleted, and call it v
// (2) if the node is a right child and has no left child of its own
//     then the parent's right child is now the current node's right child ... otherwise
// (3) if the node is a right child and has no right child of its own
//     then the parent's right child becomes the current node's left child ... otherwise
// (4) if the node is a left child and has no left child of its own
//     then the parent's left child becomes the current node's right child ... otherwise
// (5) the node is a left child and has no right child of its own
//     consequently the parent's left child becomes the current node's left child
//
```

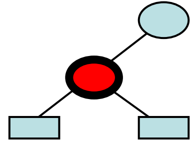










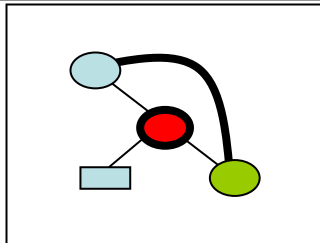


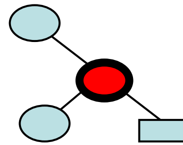
Deletion of a non-internal node

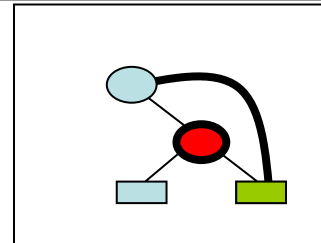
BSTree - Notepad

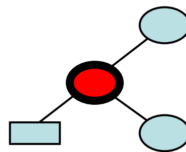
File Edit Format View Help

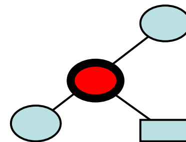
```
private void deleteNotInternal(Node node){}
// (0) the node is a leaf or has one child
// (1) get the parent of the current node to be deleted, and call it v
// (2) if the node is a right child and has no left child of its own
//     then the parent's right child is now the current node's right child ... otherwise
// (3) if the node is a right child and has no right child of its own
//     then the parent's right child becomes the current node's left child ... otherwise
// (4) if the node is a left child and has no left child of its own
//     then the parent's left child becomes the current node's right child ... otherwise
// (5) the node is a left child and has no right child of its own
//     consequently the parent's left child becomes the current node's left child
//
```

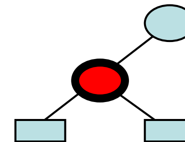










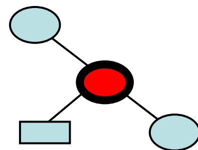
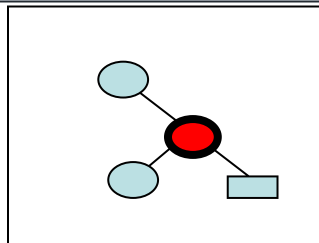
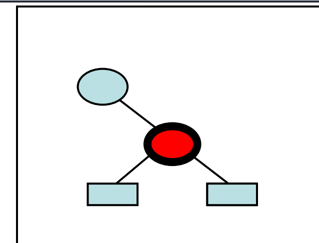
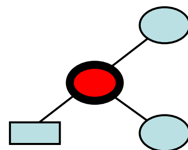
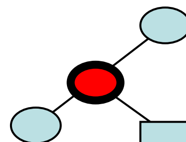
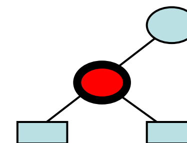


Deletion of a non-internal node

BSTree - Notepad

File Edit Format View Help

```
private void deleteNotInternal(Node node){}
// (0) the node is a leaf or has one child
// (1) get the parent of the current node to be deleted, and call it v
// (2) if the node is a right child and has no left child of its own
//     then the parent's right child is now the current node's right child ... otherwise
// (3) if the node is a right child and has no right child of its own
//     then the parent's right child becomes the current node's left child ... otherwise
// (4) if the node is a left child and has no left child of its own
//     then the parent's left child becomes the current node's right child ... otherwise
// (5) the node is a left child and has no right child of its own
//     consequently the parent's left child becomes the current node's left child
//
```

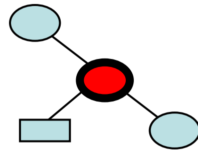
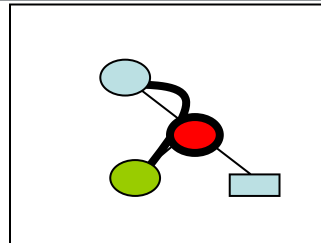
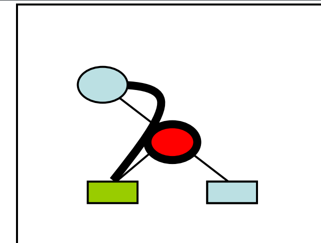
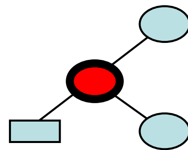
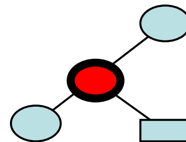
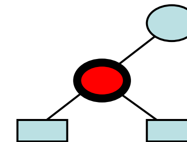







Deletion of a non-internal node

BSTree - Notepad

File Edit Format View Help

```
private void deleteNotInternal(Node node){}
// (0) the node is a leaf or has one child
// (1) get the parent of the current node to be deleted, and call it v
// (2) if the node is a right child and has no left child of its own
//     then the parent's right child is now the current node's right child ... otherwise
// (3) if the node is a right child and has no right child of its own
//     then the parent's right child becomes the current node's left child ... otherwise
// (4) if the node is a left child and has no left child of its own
//     then the parent's left child becomes the current node's right child ... otherwise
// (5) the node is a left child and has no right child of its own
//     consequently the parent's left child becomes the current node's left child
//
```

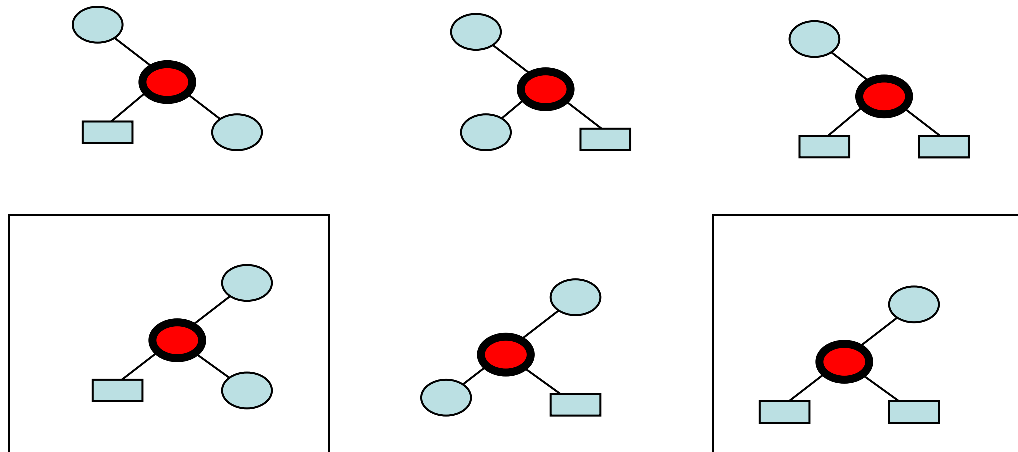







Deletion of a non-internal node

BSTree - Notepad

File Edit Format View Help

```
private void deleteNotInternal(Node node){}
// (0) the node is a leaf or has one child
// (1) get the parent of the current node to be deleted, and call it v
// (2) if the node is a right child and has no left child of its own
//     then the parent's right child is now the current node's right child ... otherwise
// (3) if the node is a right child and has no right child of its own
//     then the parent's right child becomes the current node's left child ... otherwise
// (4) if the node is a left child and has no left child of its own
//     then the parent's left child becomes the current node's right child ... otherwise
// (5) the node is a left child and has no right child of its own
//     consequently the parent's left child becomes the current node's left child
//
```

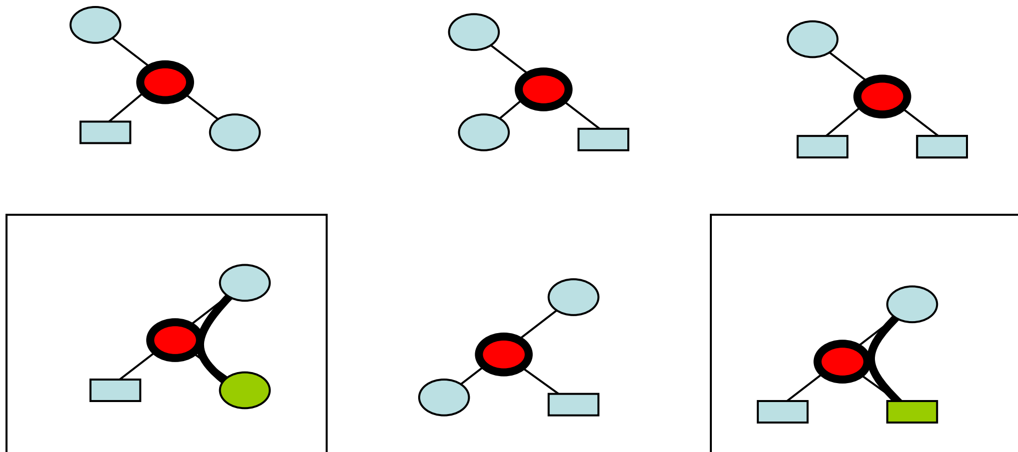


Deletion of a non-internal node

BSTree - Notepad

File Edit Format View Help

```
private void deleteNotInternal(Node node){}
// (0) the node is a leaf or has one child
// (1) get the parent of the current node to be deleted, and call it v
// (2) if the node is a right child and has no left child of its own
//     then the parent's right child is now the current node's right child ... otherwise
// (3) if the node is a right child and has no right child of its own
//     then the parent's right child becomes the current node's left child ... otherwise
// (4) if the node is a left child and has no left child of its own
//     then the parent's left child becomes the current node's right child ... otherwise
// (5) the node is a left child and has no right child of its own
//     consequently the parent's left child becomes the current node's left child
//
```

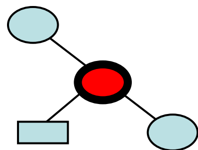
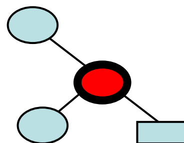
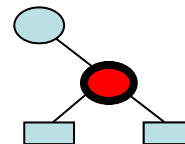
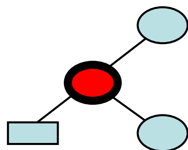
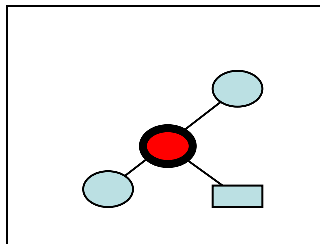
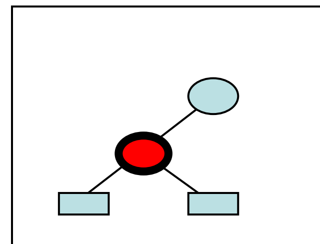


Deletion of a non-internal node

BSTree - Notepad

File Edit Format View Help

```
private void deleteNotInternal(Node node){}
// (0) the node is a leaf or has one child
// (1) get the parent of the current node to be deleted, and call it v
// (2) if the node is a right child and has no left child of its own
//     then the parent's right child is now the current node's right child ... otherwise
// (3) if the node is a right child and has no right child of its own
//     then the parent's right child becomes the current node's left child ... otherwise
// (4) if the node is a left child and has no left child of its own
//     then the parent's left child becomes the current node's right child ... otherwise
// (5) the node is a left child and has no right child of its own
//     consequently the parent's left child becomes the current node's left child
//
```

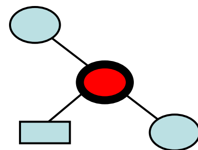







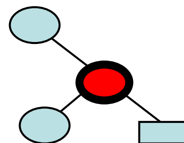
Deletion of a non-internal node

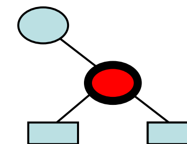
BSTree - Notepad

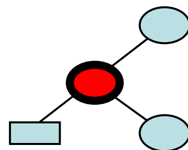
File Edit Format View Help

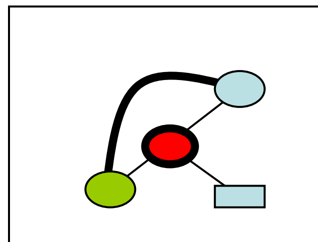
```
private void deleteNotInternal(Node node){}
// (0) the node is a leaf or has one child
// (1) get the parent of the current node to be deleted, and call it v
// (2) if the node is a right child and has no left child of its own
//     then the parent's right child is now the current node's right child ... otherwise
// (3) if the node is a right child and has no right child of its own
//     then the parent's right child becomes the current node's left child ... otherwise
// (4) if the node is a left child and has no left child of its own
//     then the parent's left child becomes the current node's right child ... otherwise
// (5) the node is a left child and has no right child of its own
//     consequently the parent's left child becomes the current node's left child
//
```

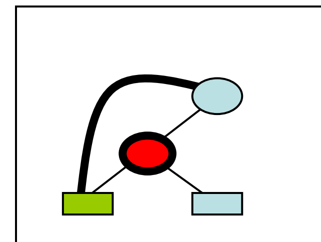






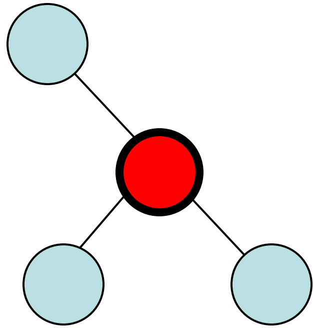


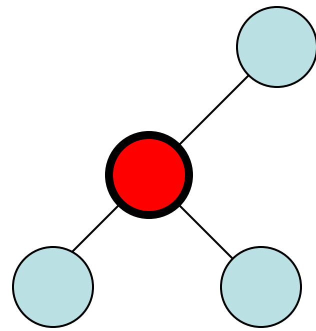




outnow.ch

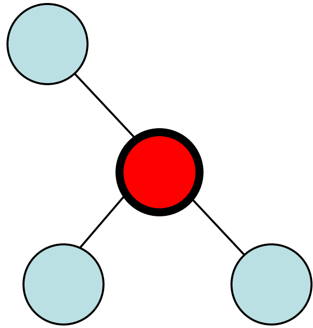
Deletion of an internal node

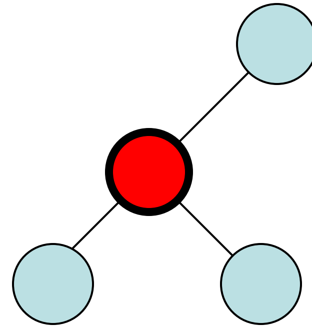




Only 2 cases to consider

Deletion of an internal node

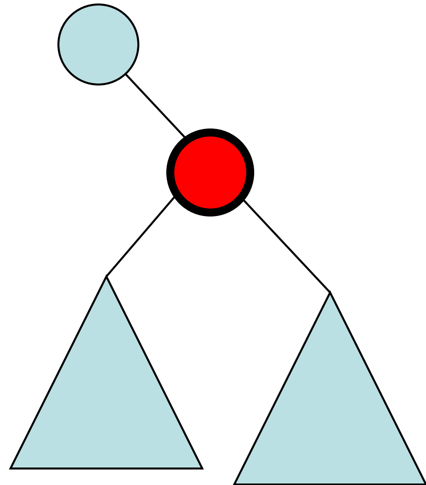




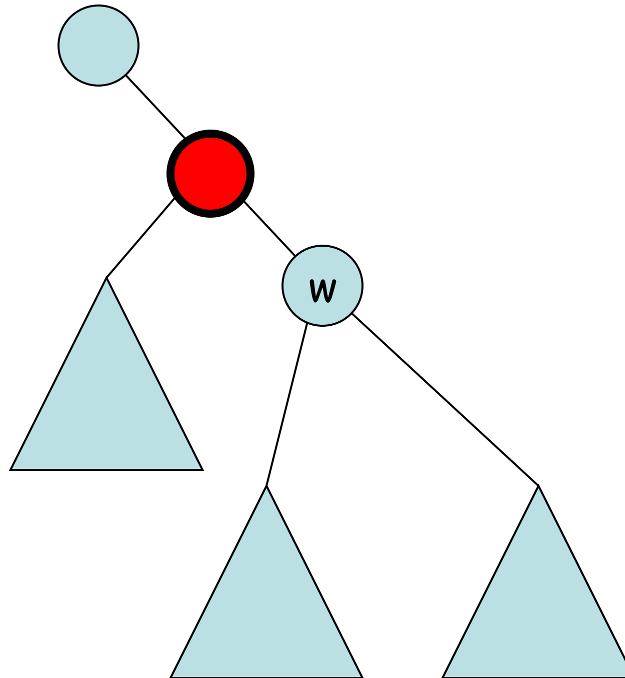
Internal – 2 children

Actually only 1 case to consider

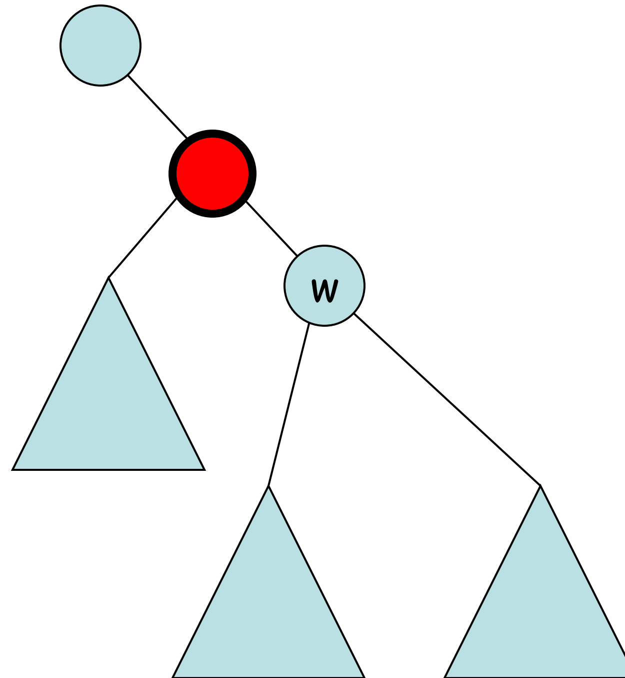
1. get the node, call it w, to the right of this node



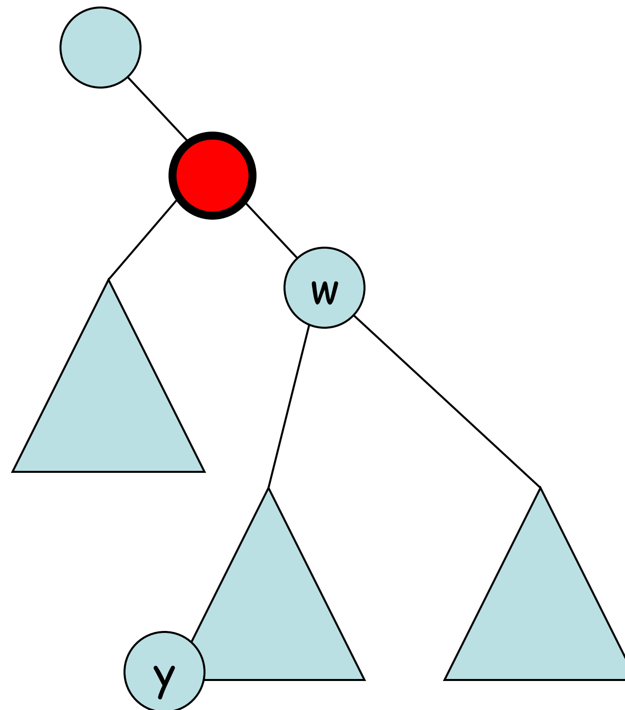
1. get the node, call it w, to the right of this node



1. get the node, call it w, to the right of this node
2. get the "smallest" node, call it y, in the subtree rooted at w

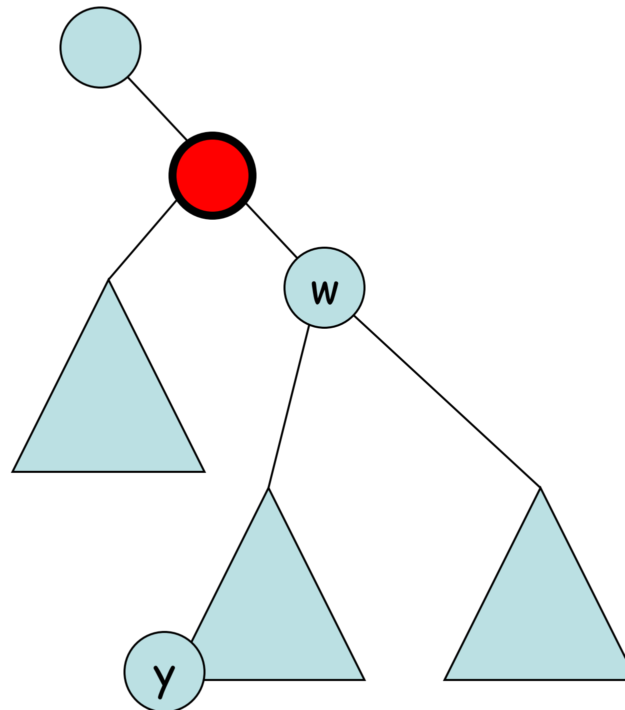


1. get the node, call it w, to the right of this node
2. get the "smallest" node, call it y, in the subtree rooted at w

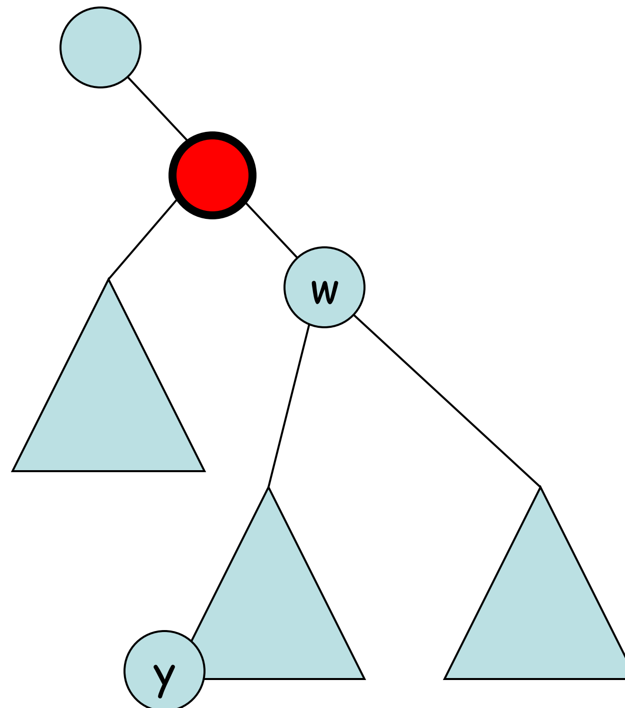


Deletion of an internal node

1. get the node, call it w, to the right of this node
2. get the "smallest" node, call it y, in the subtree rooted at w

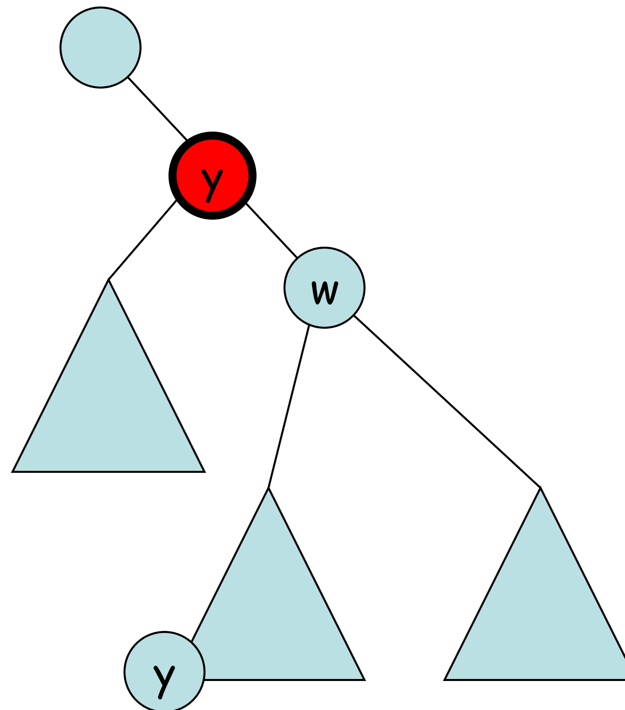


1. get the node, call it w, to the right of this node
2. get the "smallest" node, call it y, in the subtree rooted at w
3. replace what's in node with what's in y

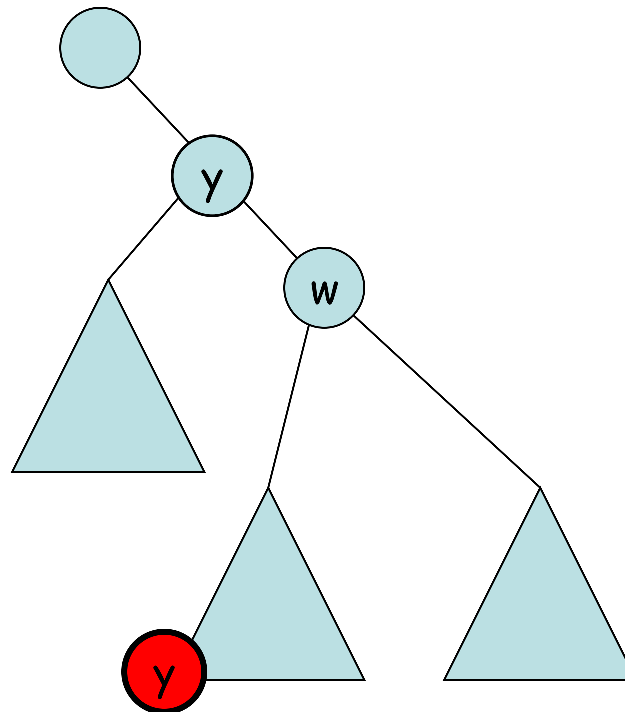


Deletion of an internal node

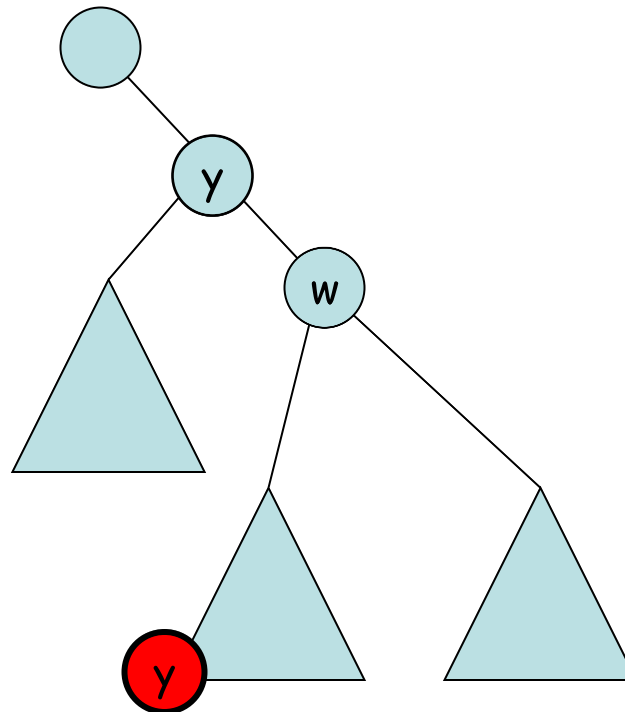
1. get the node, call it w, to the right of this node
2. get the "smallest" node, call it y, in the subtree rooted at w
3. replace what's in node with what's in y



1. get the node, call it w, to the right of this node
2. get the "smallest" node, call it y, in the subtree rooted at w
3. replace what's in node with what's in y
4. node y is not-internal, delete it (as before)



1. get the node, call it w, to the right of this node
2. get the "smallest" node, call it y, in the subtree rooted at w
3. replace what's in node with what's in y
4. node y is not-internal, delete it (as before)



Deletion of an internal node

1. get the node, call it w, to the right of this node
2. get the "smallest" node, call it y, in the subtree rooted at w
3. replace what's in node with what's in y
4. node y is not-internal, delete it (as before)

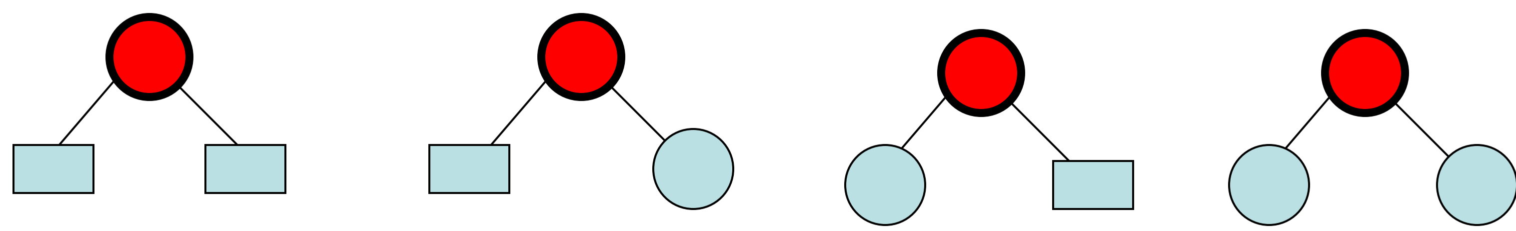
BSTree - Notepad

File Edit Format View Help

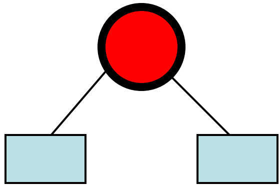
```
private void delete(Node node){}
//
// (1)   if the node is internal, i.e. has a left and right child
// (1.1) then find the smallest node in the right subtree, call this minNode
// (1.2) replace the contents of the node with the contents of the minNode
//       NOTE: this preserves inorder property
// (1.3) minNode is NOT internal, therfore deleteNotInternal(minNode)
// (2)   node is not internal, therefore deleteNotInternal(node)
//

private Node getMin(Node node){return null;}
//
// deliver the node with smallest element in the subtree rooted on node
// (1) if node has a left child
//     then find the smallest node in the tree rooted on the left child ... otherwise
// (2) node has no left child and is therefore the smallest child.
//     Deliver that node as a result
//
```

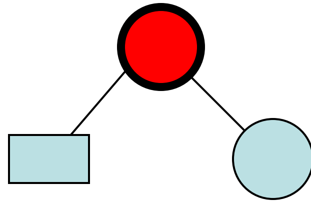
Deletion of a node

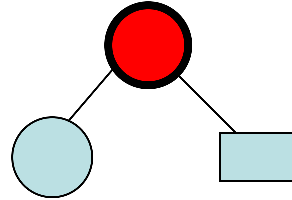
Deletion of the root

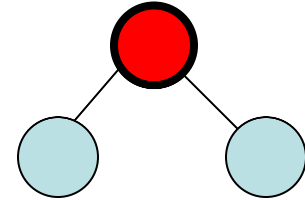


Only 4 cases to consider



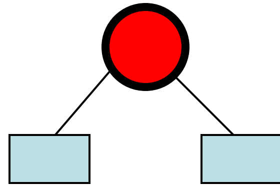


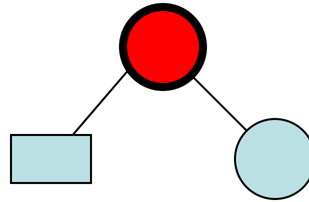


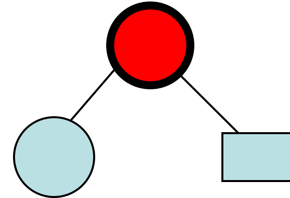


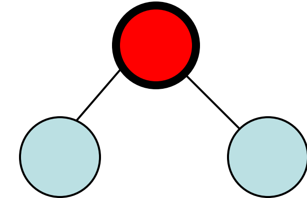
Only 4 cases to consider … or is there?

Deletion of the root







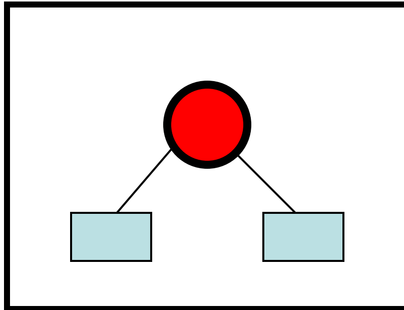


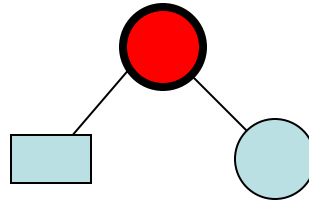
BSTree - Notepad

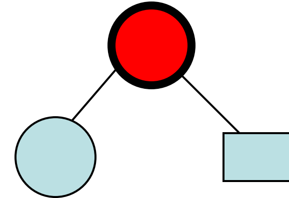
File Edit Format View Help

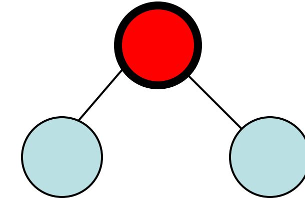
```
public void delete(String s){}
//
// (0) find the node in the tree that contains s
// (1) if not found then nothing to delete ... done!
// (2) if the node is the root and the root is a leaf, make the tree empty ... otherwise
// (3) if the node is the root and the root has a right child and no left child
//     then make the right child the root of the tree ... otherwise
// (4) if the node is the root and the root has a left child and no right child
//     then make the left child the new root of the tree ... otherwise
// (5) delete the node using the steps in method delete(node) below
// (6) Regardless, in cases (1) to (5), once done decrement the size counter
//
```

Deletion of the root







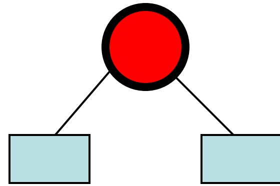
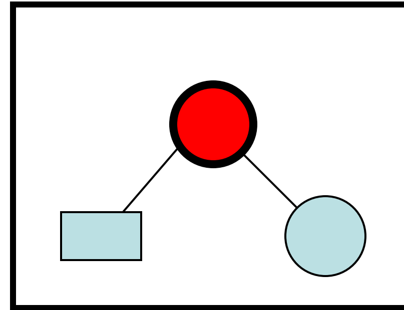
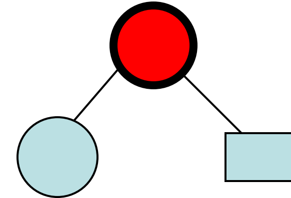
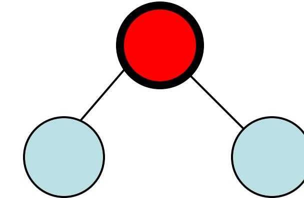


BSTree - Notepad

File Edit Format View Help

```
public void delete(String s){}
//
// (0) find the node in the tree that contains s
// (1) if not found then nothing to delete ... done!
// (2) if the node is the root and the root is a leaf, make the tree empty ... otherwise
// (3) if the node is the root and the root has a right child and no left child
//     then make the right child the root of the tree ... otherwise
// (4) if the node is the root and the root has a left child and no right child
//     then make the left child the new root of the tree ... otherwise
// (5) delete the node using the steps in method delete(node) below
// (6) Regardless, in cases (1) to (5), once done decrement the size counter
//
```

Deletion of the root

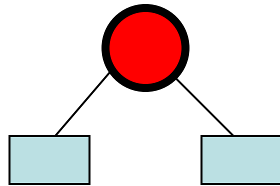





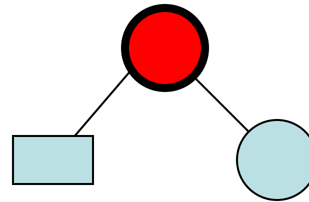
BSTree - Notepad

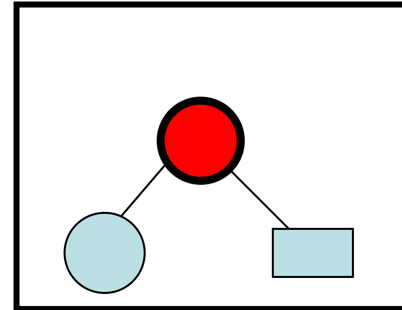
File Edit Format View Help

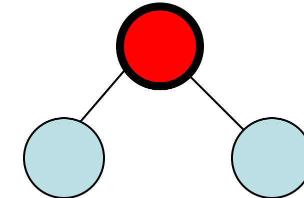
```
public void delete(String s){}
//
// (0) find the node in the tree that contains s
// (1) if not found then nothing to delete ... done!
// (2) if the node is the root and the root is a leaf, make the tree empty ... otherwise
// (3) if the node is the root and the root has a right child and no left child
//     then make the right child the root of the tree ... otherwise
// (4) if the node is the root and the root has a left child and no right child
//     then make the left child the new root of the tree ... otherwise
// (5) delete the node using the steps in method delete(node) below
// (6) Regardless, in cases (1) to (5), once done decrement the size counter
//
```

Deletion of the root







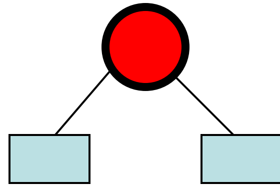


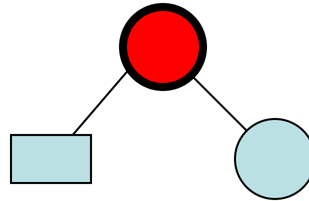
BSTree - Notepad

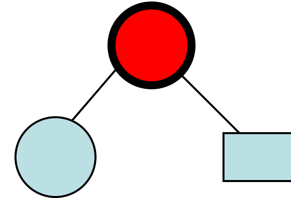
File Edit Format View Help

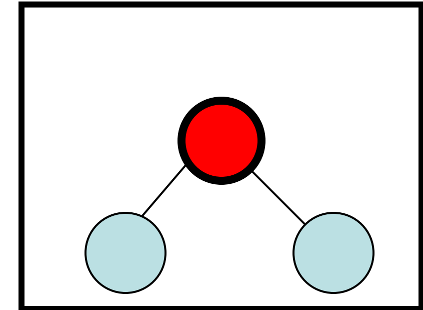
```
public void delete(String s){}
//
// (0) find the node in the tree that contains s
// (1) if not found then nothing to delete ... done!
// (2) if the node is the root and the root is a leaf, make the tree empty ... otherwise
// (3) if the node is the root and the root has a right child and no left child
//     then make the right child the root of the tree ... otherwise
// (4) if the node is the root and the root has a left child and no right child
//     then make the left child the new root of the tree ... otherwise
// (5) delete the node using the steps in method delete(node) below
// (6) Regardless, in cases (1) to (5), once done decrement the size counter
//
```

Deletion of the root









BSTree - Notepad

File Edit Format View Help

```
public void delete(String s){}
//
// (0) find the node in the tree that contains s
// (1) if not found then nothing to delete ... done!
// (2) if the node is the root and the root is a leaf, make the tree empty ... otherwise
// (3) if the node is the root and the root has a right child and no left child
//     then make the right child the root of the tree ... otherwise
// (4) if the node is the root and the root has a left child and no right child
//     then make the left child the new root of the tree ... otherwise
// (5) delete the node using the steps in method delete(node) below
// (6) Regardless, in cases (1) to (5), once done decrement the size counter
//
```

BSTree - Notepad

File Edit Format View Help

```
private Node find(String s,Node node){return null;}
//
// given a node and a string s
// (0) we have found s if s is equal to the data in the node otherwise ...
// (1) if s is less than the data in the node and the node has a left child
//     then search for s is in the tree rooted at the left child ... otherwise
// (2) if s is greater than the data in the node and the node has a right child
//     then search for s is in the tree rooted at the right child ... otherwise
// (3) the string s is not in the tree!
//
```

As before ...

All of the code for deletion

All of the code

Deletion of a node

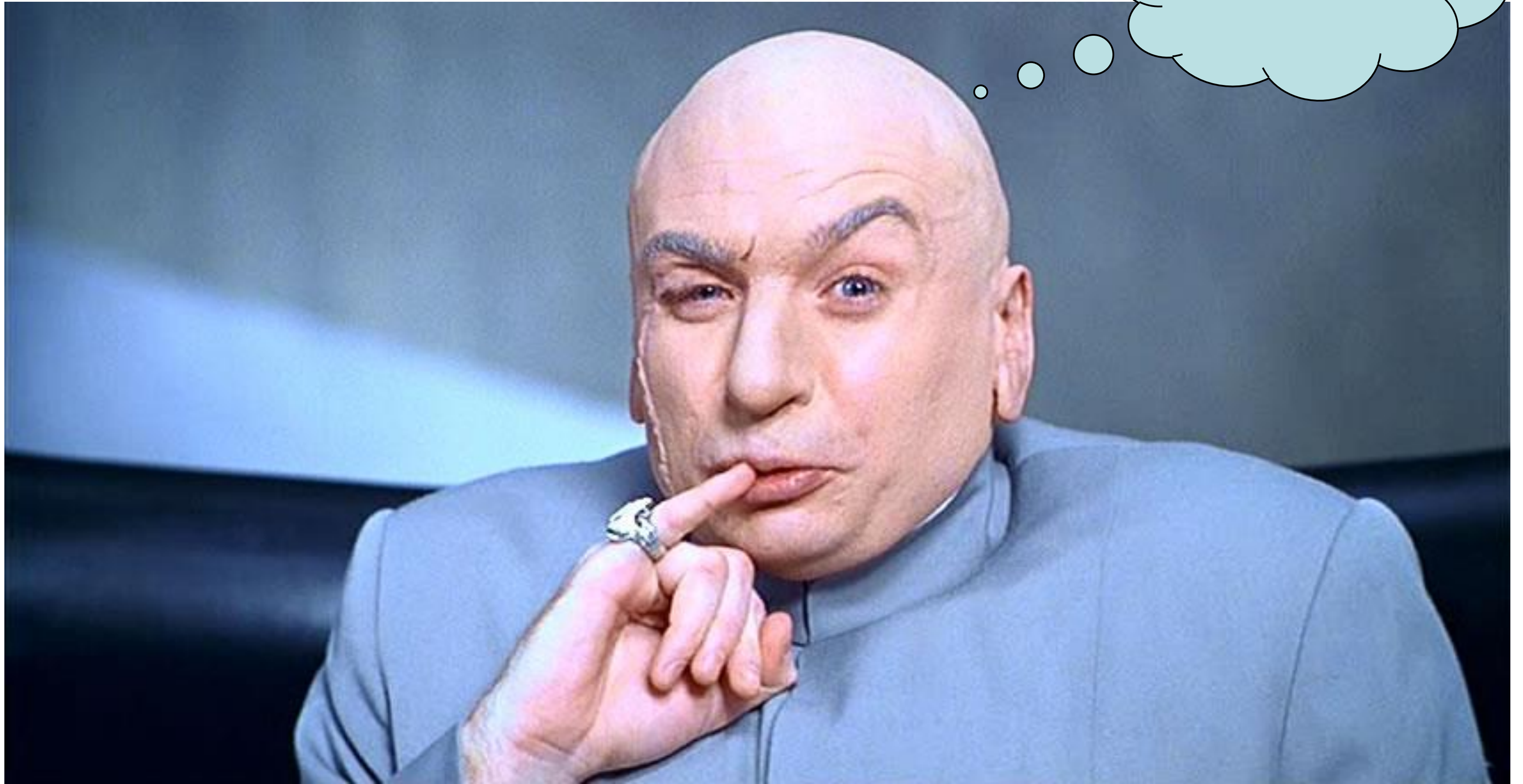
BSTree - Notepad

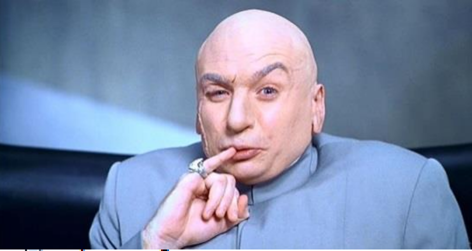
File Edit Format View Help

```
    //

    public void delete(String s){}
    //
    // (0) find the node in the tree that contains s
    // (1) if not found then nothing to delete ... done!
    // (2) if the node is the root and the root is a leaf, make the tree empty ... otherwise
    // (3) if the node is the root and the root has a right child and no left child
    //     then make the right child the root of the tree ... otherwise
    // (4) if the node is the root and the root has a left child and no right child
    //     then make the left child the new root of the tree ... otherwise
    // (5) delete the node using the steps in method delete(node) below
    // (6) Regardless, in cases (1) to (5), once done decrement the size counter
    //

    private void delete(Node node){}
    //
    // (1)   if the node is internal, i.e. has a left and right child
    // (1.1) then find the smallest node in the right subtree, call this minNode
    // (1.2) replace the contents of the node with the contents of the minNode
    //       NOTE: this preserves inorder property
    // (1.3) minNode is NOT internal, therfore deleteNotInternal(minNode)
    // (2)   node is not internal, therefore deleteNotInternal(node)
    //

    private Node getMin(Node node){return null;}
    //
    // deliver the node with smallest element in the subtree rooted on node
    // (1) if node has a left child
    //     then find the smallest node in the tree rooted on the left child ... otherwise
    // (2) node has no left child and is therefore the smallest child.
    //     Deliver that node as a result
    //

    private void deleteNotInternal(Node node){}
    // (0) the node is a leaf or has one child
    // (1) get the parent of the current node to be deleted, and call it v
    // (2) if the node is a right child and has no left child of its own
    //     then the parent's right child is now the current node's right child ... otherwise
    // (3) if the node is a right child and has no right child of its own
```

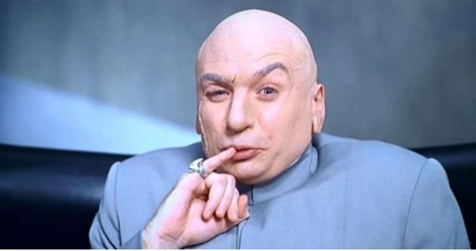
outnow.ch

Is there a demo?





```java
import java.io.* ;

public class Test {

    public static void main(String[] args) throws Exception, FileNotFoundException {

        String commands = "\nBTree Tester (version 1.3247179) \n" +
                          "insert (+), delete (-), present (?), read, \n" +
                          "root, show, draw, size, clear, quit (q)";

        System.out.println(commands);
        Scanner sc = new Scanner(System.in);
        Scanner fin = null;

        System.out.print("> ");
        String command = sc.next();
        BSTree t1 = new BSTree();
        Graphic graphic = new Graphic(t1);
        String s = "";

        if (args.length > 0){
            fin = new Scanner(new File(args[0]));
            while (fin.hasNext()){s = fin.next(); t1.insert(s);}
            fin.close();
        }

        while (!command.equals("quit") && !command.equals("q")){

            if (command.equals("help")) System.out.println(commands);

            if (command.equals("size")) System.out.println(t1.size());

            if (command.equals("insert") || command.equals("ins") || command.equals("+")){
                System.out.print("insert >> ");
                s = sc.next();
                t1.insert(s);
                graphic.draw();
            }
```



Command Prompt - java Test data200.txt

```
C:\Andrea\bstree\answer>javac *.java

C:\Andrea\bstree\answer>java Test data200.txt

BTree Tester (version 1.3247179)
insert (+), delete (-), present (?), read,
root, show, draw, size, clear, quit (q)
> draw
> show
((((((-,abase,(-,about,-)),abyss,(((-,admix,-),adult,-),ahead,-)),alder,-),alla
augur,((-,azure,-),baldy,-))),balsa,(((((-,bebop,-),befog,((-,belie,(-,bench,-))
e,-),bream,(-,breve,-)),bulky,-))),bundy,(-,burnt,(-,butyl,-))),bylaw,((-,cadre,
oud,-)),clump,-)),conic,((((-,cooky,-),coral,(-,corps,-)),count,-),cover,-)))),c
,-),death,((-,debug,-),degas,(-,demit,-)))),devil,((-,dirty,(-,drake,-)),drape,(
voy,(((-,exalt,-),expel,((-,fault,(-,fetus,-)),first,(((-,flick,-),flown,-),flus
,gaunt,((-,geese,-),glint,((-,gloss,-),grant,-)))))),group,((((((-,guile,((-,haz
-,idyll,-)),iliac,(-,infix,-)),jolly,-),kraft,(-,laugh,(-,layup,((-,leave,-),led
((-,magic,-),march,-)),marry,(-,medal,-)),messy,(-,metal,(-,mixup,-))),molar,-),
,(-,panda,(-,parry,-)))))))),pence,(((-,piggy,((-,pixel,-),pizza,-)),price,((-,
))),raven,((-,reave,(-,repel,-)),rheum,-))),ripen,(((((-,robot,-),rowdy,-),rumen
corn,(-,serif,(-,sever,-))),shall,(-,shawl,(-,sheep,-))))),shirt,((-,shoot,-),sh
),sloth,-),slurp,((-,soapy,-),spear,-)))),spitz,(-,spoof,((((-,spurn,-),stack,((
,-))),sushi,((((-,swear,(-,swipe,(-,swoop,(-,tappa,-)))),tenon,((-,tepee,-),terr
ithe,-))),topaz,((((-,trace,(-,trail,-)),tramp,(-,trash,-)),treat,((-,trial,-),t
)),urine,(-,utter,(((-,vacua,-),vowel,-),waist,(-,wheat,-))))))),whose,((-,widen
>
> help

BTree Tester (version 1.3247179)
insert (+), delete (-), present (?), read,
root, show, draw, size, clear, quit (q)
>
```

Standard Draw

File

Your mission, should you choose to accept it …

Implement insert, find , height, the traversals, bfs, dfs, ……