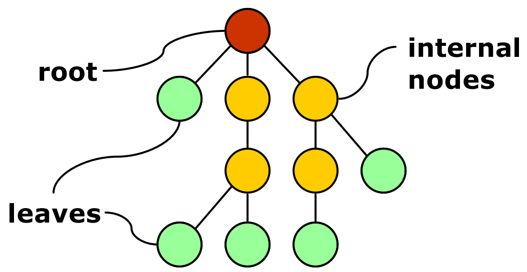
Week 8: Trees

Readings

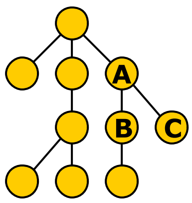
- **Required**
  - [Weiss] ch18.1 – 18.3
  - [Weiss] ch18.4.4
  - [Weiss] ch19.1 – 19.2
- **Exercises**
  - [Weiss] 18.1, 18.2, 18.3, 18.9
  - [Weiss] 19.1, 19.15 – 19.19
- **Fun**
  - http://www.seanet.com/users/arsen/avltree.html

Tree



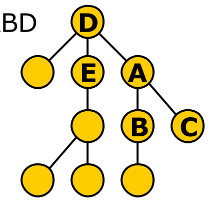
Relationship

- A is ***parent*** of B and C
- B and C are ***children*** of A
- B and C are ***siblings***



Relationship

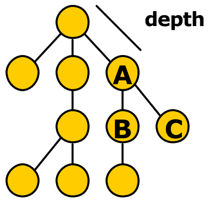
- E is ***uncle/auntie*** of BC
- D is ***ancestor*** of ABCDE
- B is ***descendant*** of ABD



A node is an ancestor of itself, and a descendant of itself.

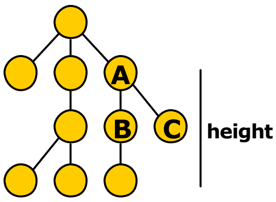
Depth

- **Length of path to the root.**
  - depth of A is 1



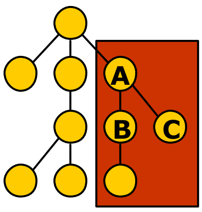
Height

- **Length of path to the deepest leaf.**
  - height of A is 2



Size

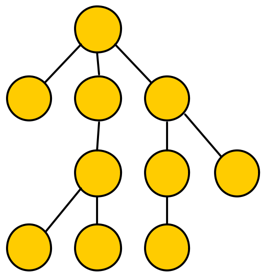
- **Number of descendants.**
  - size of A is 4



Applications

- Family Tree
- Directory Tree
- Organization Chart

Tree is recursive!

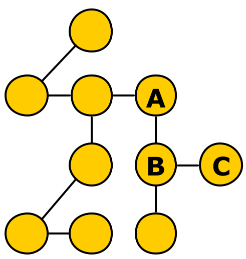


Implementation

- "first-child, next-sibling"

```
class TreeNode
{
  Object element;
  TreeNode firstChild;
  TreeNode nextSibling;
  // Methods..
}
```

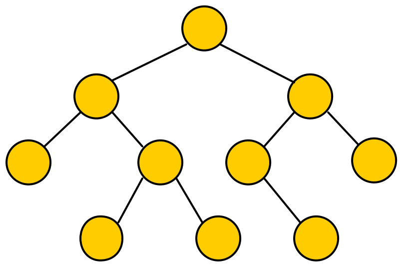

Implementation



Binary Trees

nus.soc.cs1102b.week8 21

Binary Tree



nus.soc.cs1102b.week8 22

Just like a tree, a binary tree is recursive in nature.

An Empty Binary Tree

nus.soc.cs1102b.week8 24

An empty binary tree is just a reference to null.

Implementation

```
class BinaryNode
{
  Object element;
  BinaryNode left;
  BinaryNode right;
  // Methods..
}

class BinaryTree
{
  BinaryNode root;
  // Methods
}
```

nus.soc.cs1102b.week8 25

We can add other members, such as a reference to parent (see successor()) and size of the subtree (see findKth()).

Size of a Tree

size(T)

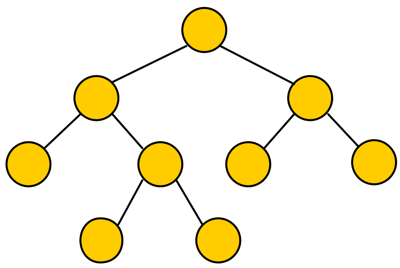
```
if T is empty
    return 0
else
    return 1+size(T.left)+size(T.right)
```


Height of a Tree

height(T)

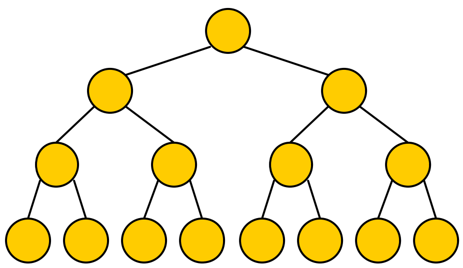
```
if T is empty
  return -1
else
  return 1 + max (height(T.left), height(T.right))
```


Full Binary Tree



In a full binary tree, every node must have either 0 or 2 children.

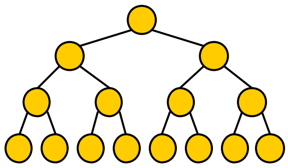
Complete Binary Tree



A complete binary tree is a full binary tree where all leaves are of the same depth.

Property

How many nodes in a complete binary tree of height h?



nus.soc.cs1102b.week8 32

Number of nodes = $2^{h+1} - 1$
Height is O(log N).

Binary Tree Traversal

nus.soc.cs1102b.week8 33

Post-order Traversal

```
postorder(T)
  if T is not empty then
     postorder(T.left)
     postorder(T.right)
     print T.element
```

nus.soc.cs1102b.week8 35

Pre-order traversal

```
preorder(T)
  if T is not empty then
     print T.element
     preorder(T.left)
     preorder(T.right)
```

nus.soc.cs1102b.week8 36

In-order Traversal

inorder(T)

if T is not empty **then**

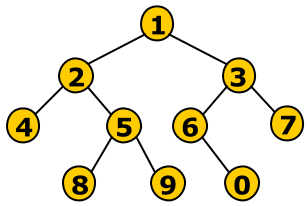
inorder(T.left)

print T.element

inorder(T.right)

nus.soc.cs1102b.week8 37

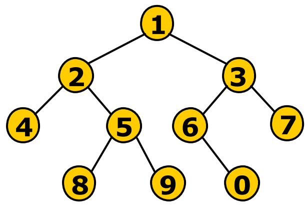
Traversal Example



Post-order: 4 895 2 06 7 3 1

nus.soc.cs1102b.week8 38

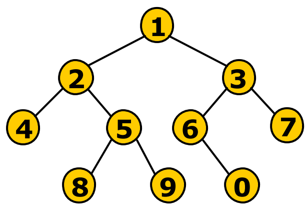
Traversal Example



Pre-order: 1 2 4 589 3 60 7

nus.soc.cs1102b.week8 39

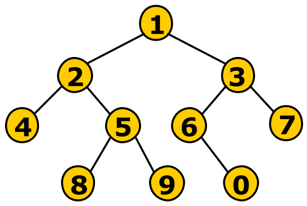
Traversal Example



In-order: 4 2 859 1 60 3 7

nus.soc.cs1102b.week8 40

Level-order Traversal

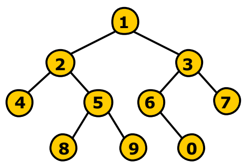


Level-order: 1234567890

nus.soc.cs1102b.week8 41

levelOrder(T)

```
if T is empty return
Q = new Queue
Q.enq(T)
while Q is not empty
   curr = Q.deq()
   print curr.element
   if T.left is not empty
        Q.enq(curr.left)
   if curr.right is not empty
        Q.enq(curr.right)
```



nus.soc.cs1102b.week8 42

What do you get when you replace the queue with a stack?

Binary Search Tree

nus.soc.cs1102b.week8 43

Dynamic Set Operation

- **insert** (key, data)
- **delete** (key)
- data = **search** (key)
- key = **findMin** ()
- key = **findMax** ()
- key = **findKth** (k)
- data[] = **findBetween** (low, high)
- **successor** (key)
- **predecessor** (key)

nus.soc.cs1102b.week8 44

Running Time

	Unsorted Array/List	Sorted Array	Sorted LinkedList
insert	O(1)	O(N)	
delete	O(N)	O(N)	
find	O(N)	O(logN)	
findMin	O(N)	O(1)	
findMax	O(N)	O(1)	

nus.soc.cs1102b.week8 45

Recap

	Unsorted array/list	Sorted Array	Sorted List
findKth	O(N)	O(1)	
find Between	O(N)	O(k + logN)	
sucessor	O(N)	O(log N)	

nus.soc.cs1102b.week8 46

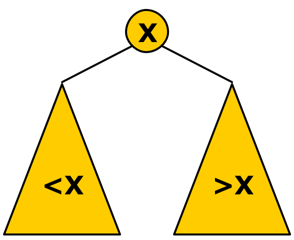
Variable k is the size of the output of findBetween( ).

Binary Search Tree

- All operations O(log N)
- findBetween O(k + logN)

nus.soc.cs1102b.week8 47

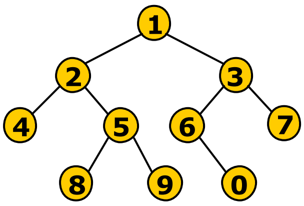
BST Property



nus.soc.cs1102b.week8 48

The BST property holds recursively, which means the left sub-tree and right sub-tree must be BST as well.

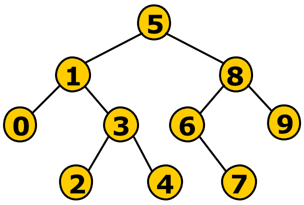
Example



Not a BST

nus.soc.cs1102b.week8 49

Example



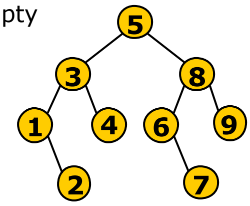
A BST

nus.soc.cs1102b.week8 51

What do you get when you traverse a BST in in-order?

Finding Minimum Element

while T.left is not empty
  T = T.left
return T.element

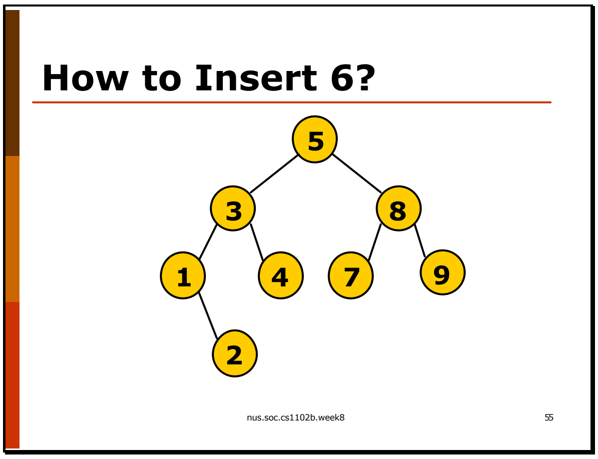


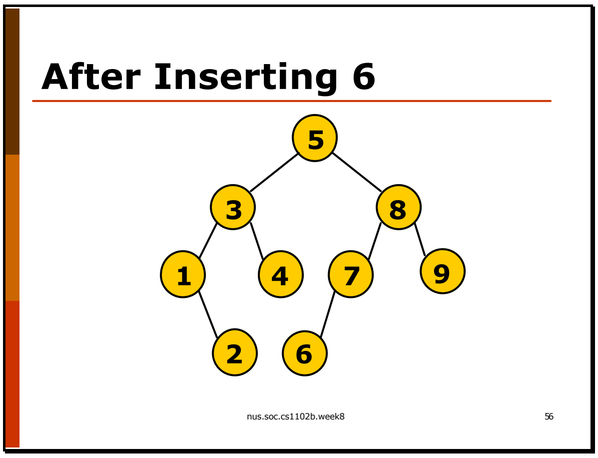
nus.soc.cs1102b.week8 52

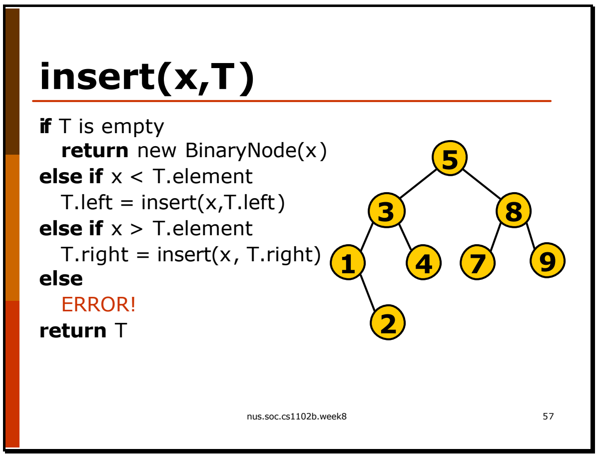
Finding x in T

while T is not empty
  if T.element == x **then**
    return T
  else if T.elements < x **then**
    T = T.left
  else
    T = T.right
return NOT FOUND

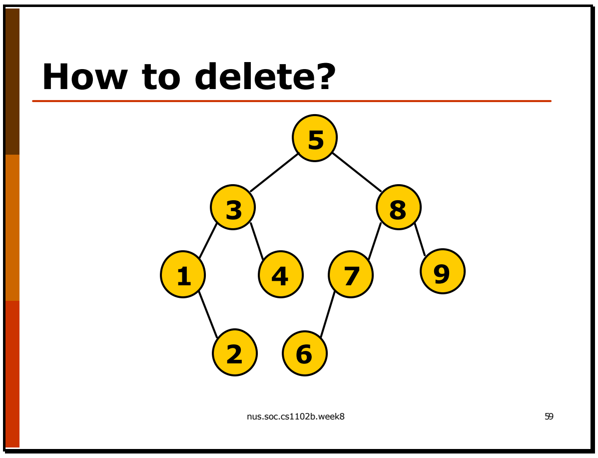
nus.soc.cs1102b.week8 53







Method insert(x,T) returns the new tree after inserting x into T.



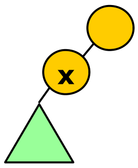
delete(x,T): Case 1

if T has no children
  if x == T.element
    return empty tree
  else
    NOT FOUND

Method delete(x,T) returns the new tree after deleting x from T.

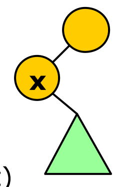
delete(x,T): Case 2

if T has 1 child T.left
  if x == T.element
    return T.left
  else
    T.left = delete(x, T.left)
  return T

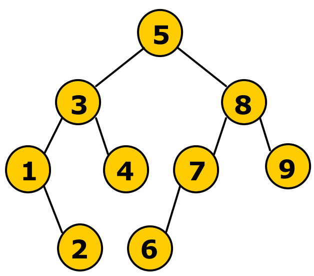


delete(x,T): Case 2

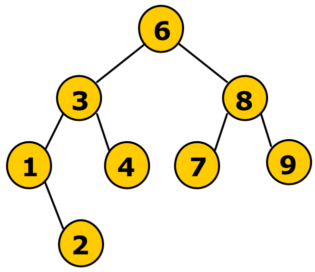
if T has 1 child T.right
  if x == T.element
    return T.right
  else
    T.right = delete(x,T.right)
  return T



delete(x,T): Case 3



delete(x,T): Case 3



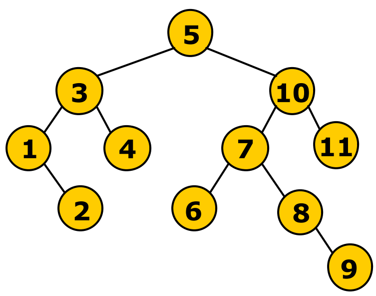
nus.soc.cs1102b.week8 65

delete(x,T): Case 3

```
if T has two children
  if x == T.element
     T.element = findMin(T.right)
     T.right = delete(T.element, T.right)
  else if x < T.element
     T.left = delete(x, T.left)
  else
     T.right = delete(x, T.right)
return T
```

nus.soc.cs1102b.week8 66

Successor



nus.soc.cs1102b.week8 68

Successor returns the next larger element in the tree.
Successor(5) is 6.
Successor(4) is 5.
11 does not have a successor.

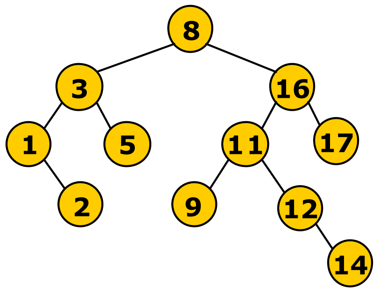
Successor(T)

```
// find next largest element
if T.right is not empty
  return findMin(T.right)
else if T is a left child
  return parent of T
else T is a right child
  let x be the first ancestor of T that is a left
  child
  return parent of x
```

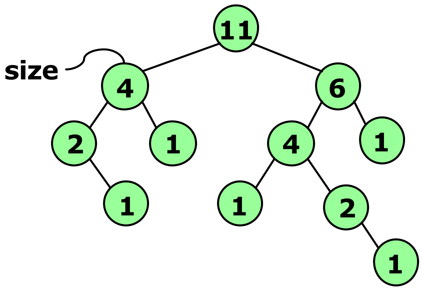
nus.soc.cs1102b.week8 69

- What happen if we cannot find such an x? This means that there is no successor for T. (i.e. T is the maximum).
- We need a reference to the parent for this operation, so that we can traverse up the tree.
- Second and third case can actually be combined into one.
- Question: why is the algorithm on the left correct? Think about it using the property of BST.

findKth(T,K)



Size of a Tree



Observation:

- if a node,T, has 6 elements in its right sub-tree, we know that T is the 7th largest element in the tree.
- The 1st,2nd,…6th largest elements must be in the right sub-tree.
- The 9th largest element in T is the 2nd largest element in the left sub-tree of T. (9 – 6 – 1 = 2)

findKthSmallest(T,K)

```
let L be the size of T.left
if K == L + 1
  return T.element
else if K <= L
  return findKthSmallest(T.left, K)
else
  return findKthSmallest(T.right, K – L – 1)
```


findKthLargest(T,K)

```
let L be the size of T.right
if K == L + 1
  return T.element
else if K <= L
  return findKthLargest(T.right, K)
else
  return findKthLargest(T.left, K – L – 1)
```

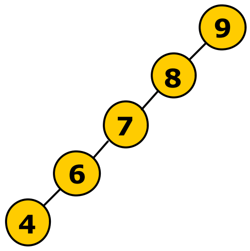
Running Time

- find O(h)
- findMin O(h)
- insert O(h)
- delete O(h)
- successor O(h)
- findKth O(h)

nus.soc.cs1102b.week8 74

BUT

h = O(N)



nus.soc.cs1102b.week8 75

When you insert nodes in increasing order, you get a skewed tree. Therefore h is actually in O(N).

```java
/**
 * Return the node containing the successor of x.  This method is part of
 * BinarySearchTree class.  I assume that BinaryNode has a member called
 * parent.  If a node is the root, parent points to null, otherwise it
 * points to its parent.  (Modifying insert/delete to maintain the parent
 * pointer is a good exercise to help you understand BinarySearchTree.)
 *
 * @param x the item whose successor we want to search for.
 * @return the successor or null if no successor exists.
 */
public BinaryNode successor( Comparable x )
{
    BinaryNode t = find(x, root);
    if (t.right != null)
    {
        // right child is not empty, just call findMin on the right
        // child.
        return findMin(t.right);
    }
    else  // t has no right child
    {
        if (t.parent == null)
        {
            // t is the root and has no right child.  so t must be
            // the largest.  (i.e. no successor).
            return null;
        }
        else if (t.parent.left == t)
        {
            // t is a left child, return the parent.
            return t.parent;
        }
        else if (t.parent.right == t)
        {
            // t is a right child.  find the first ancestor that is
            // a left child.
            BinaryNode p = t.parent;
            while (p.parent != null)
            {
                if (p.parent.left == p)
                {
                    // p is the first ancestor that is a left child.
                    // return its parent.
                    return p.parent;
                }
                else
                {
                    // proceed to the next ancestor.
                    p = p.parent;
                }
            }
            // reach the root and found nothing.  t must be the largest.
            return null;
        }
    }
    return null; // to make compiler happy.
}
```