Summary previous lecture

- **Methods...so far**
  - find
  - insert
  - Traverse (in-order, pre-order, post-order)
Summary of Traversal

**in-order**
1. Visit the left subtree
2. Visit the node (ex: display it)
3. Visit the right subtree

**pre-order**
1. Visit the node (ex: display it)
2. Visit the left subtree
3. Visit the right subtree

**post-order**
1. Visit the left subtree
2. Visit the right subtree
3. Visit the node (ex: display it)

Result is (In-Order) : 1 5 7 10 14 16
Result is (Pre-Order) : 10 5 1 7 14 16
Result is (Post-Order): 1 7 5 16 14 10
Deleting a node from BST

- Common but complicated procedure
- Inportant in many tree applications
- Easy alternative to 'avoid explicit deletion': use a boolean flag 'isDeleted' for all nodes...however, memory can fill up with deleted nodes.
- Explicit deletion is then preferred
- From textbook:
  <<...studying details builds character>>

- Three cases to consider
  - Case 1: Node is a leaf … easy
  - Case 2: Node has only one child … still easy
  - Case 3: Node has two childrens...more difficult
Deleting a node- Case 1

- Node is a leaf that needs to be disconnected from its parent
- Assume we want to remove 7

First, one needs to find node 7, remember its parent 5 and if it is a left or a right child.

- Right child of node 5 is set to null
- In Java, node 7 awaits the garbage collector
Deleting a node- Case 2

- Node has one child, so it has only 2 connections: one going to its parent and one going to its child
- Assume we want to remove 14

First, one needs to find node 14, remember its parent 10, if it is a left of a right child, and if its only child is left or right
- Second, we connect its parent to its only child
- Remark: working with references make it easy to move entire subtree
Deletion of a node - Case 3

- Node has two children
- Assume we want to remove node 10

There are two subtrees for node 10, we **cannot** simply lift left/right subtree up
- We propose to investigate two different solutions for this problem
Deleting a node - Case 3 - Solution 1

- All nodes in the left subtree is less than all nodes in the right subtree
- How to merge two split subtree into one subtree?
  - Find maximum node in left subtree... Node 7 here
  - Join the right subtree into that node
  - Lift node's left child up
Delete a node - Case 3 - Solution 2 - Textbook

- Replace the node to be deleted by its successor
  - Find its inorder successor... Node 12 here
  - Replace it
Finding the successor

- Find the minimum value that is greater than the original node

```java
public Node getSuccessor(Node delNode) {
    Node successorParent = delNode;
    Node successor = delNode;
    Node current = delNode.right;
    while (current != null) {  // until no more left
        successorParent = successor;  // children
        successor = current;
        current = current.left;
    }
    if (successor != delNode.right) {
        successorParent.left = successor.right;
        successor.right = delNode.right;
    }
    return successor;
}
```
Some Examples:

- If successor is Right Child of delNode (2 steps)

- If successor is left descendant of right child of delNode (4 steps)
Display Tree

- Motivations: How to show tree structures?
  - How does the tree look like after insertion/removal of nodes?
  - Easy approach → rotate the tree by 90 degrees
    - Most left is root
    - Right subtree of root is up
    - Left subtree of root is down

```java
public void recShow(Node current, String indStr)
{
    if (current != null){
        recShow(current.right, indStr + " ");
        System.out.println(indtStr + current.key);
        recShow(current.left, indStr + " ");
    }
}
```

1- Print right subtree
2- Print current Node
3- Print left subtree

```java
public void show()
{
    System.out.println("The tree looks like: ");
    recShow(root," ");
}
```
Tree represented as Arrays

- Using an array representation, the position of the nodes corresponds to its position on the tree
  - The node 0 is the root, 1 is the root's left child, 2 is the root's right's child, etc.
  - Cells of the array representing tree positions with no node are filled with 0 or null

Important properties, if node's number is index:

- This node's left child is 2*index+1
- This node's right child is 2*index+2
- This node's parent is (index-1)/2

Remarks: array approach is not that effective but could be useful in particular situations