Trees III
Lecture 20

Prof. Eric Polizzi
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)
3- 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
- Important 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 children...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

```
  10
  / \
 5   14
1   7   16
```

```
  10
  / \
 5   14
 1   7   16
```

- 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

![Tree diagram]

- 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
Deleting a node- Case 3

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

There are two subtree 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

![Diagram of a binary search tree with nodes 1, 3, 4, 5, 7, 10, 12, 14, 15, 16, 20, with arrows indicating the process of deleting a node and merging subtrees.]
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;
}
```
Deleting a node - Case 3 - Solution 2 - Textbook

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

- If successor is left descendant of right child of delNode (4 steps)
Tree represented as Arrays

- Using an array representation, the position of the nodes corresponds to its position on the tree
  - The index 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 $2 \times \text{index} + 1$
- This node's right child is $2 \times \text{index} + 2$
- This node's parent is $(\text{index} - 1)/2$

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

- The max grade is 100 (bonus points beyond 100)
- Do not cheat! Projects are individual, we use an automated software to check each and every code (it would take you more efforts to outsmart the software than writing the project). High-price to pay if caught.
- Do not use outside libraries it could lead to compilation problems (if they are not build-in). Projects are self-contained.