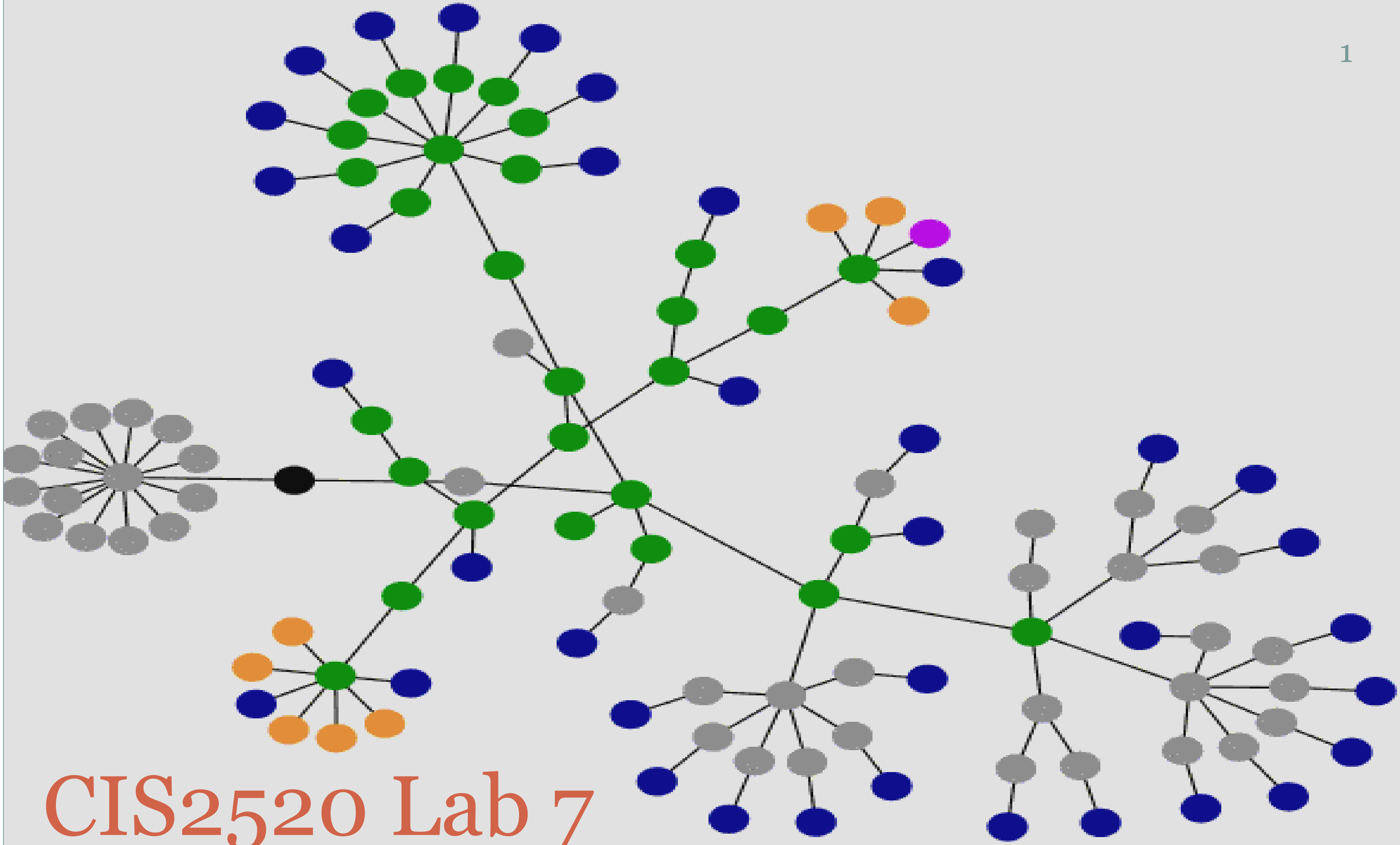


CIS2520 Lab 7

DATA STRUCTURE

Trees

- Traversals
- Binary Tree
- Binary Search Tree
- Heap
- AVL Tree
- Assignment 3

Traversals

Preoder: visit parent before children from left to right

Recursive:

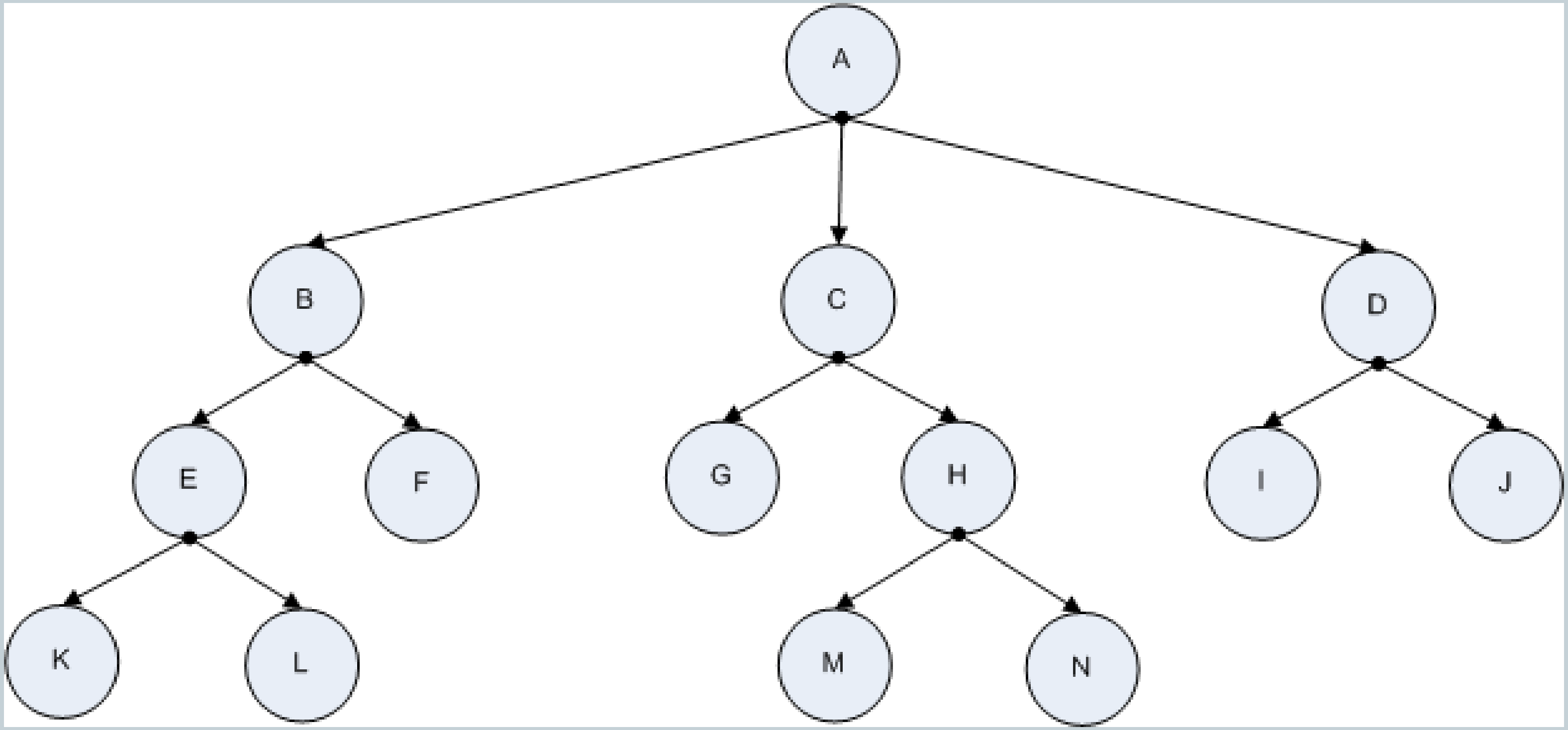
```
preorder(v){
        visit(v)
        for each child w of v
                preorder(w)
}
```

Traversals

Iterative:

```
Preorder(v){
    Stack s;
    iterator it;
    s.push(v);
    while(s.size() > 0 ){
        it = s.pop();
        visit(it);
        for each child w of it from right to left
            s.push(w);
    }
```

Traversal



Preorder: A, B, E, K, L, F, C, G, H, M, N, D, I, J

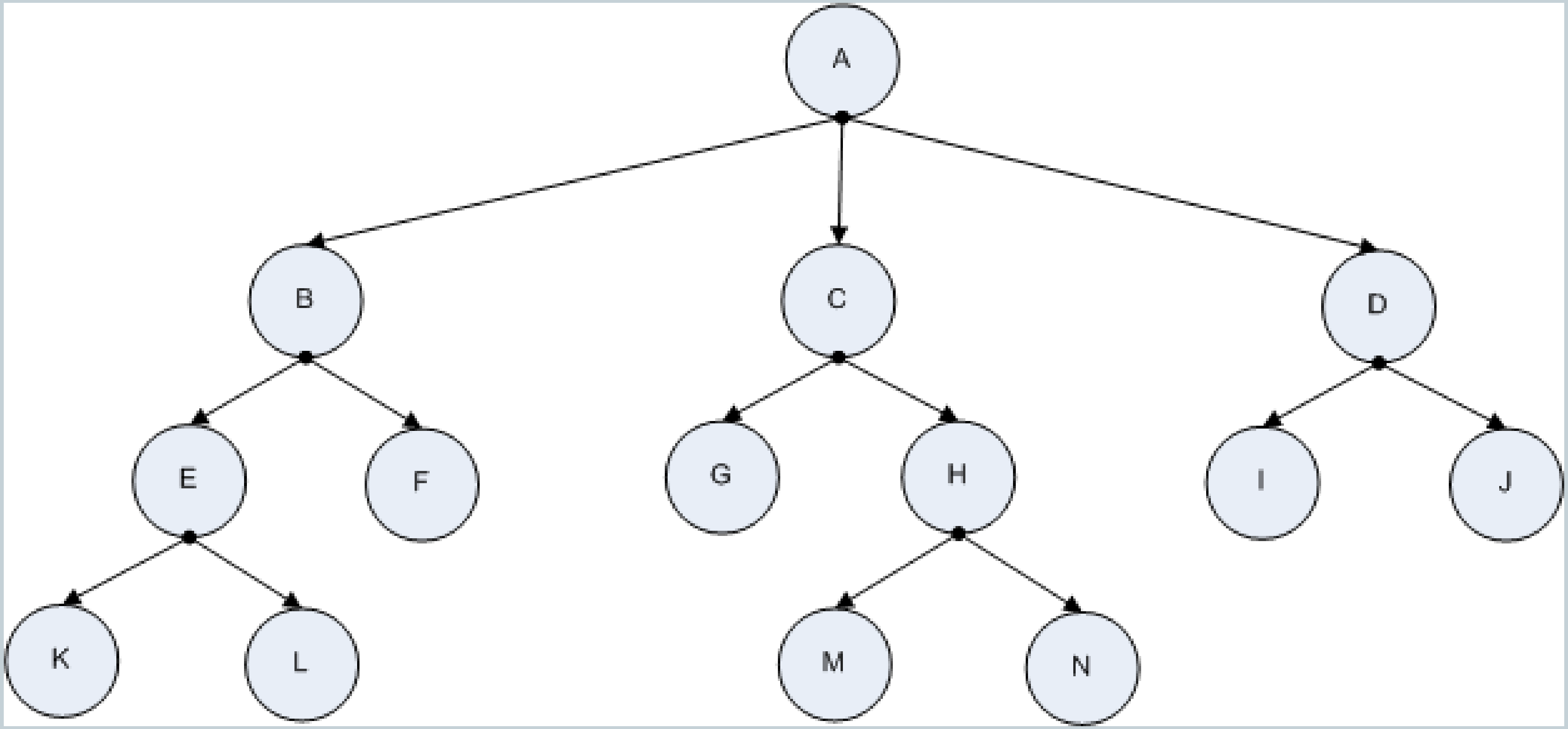
Traversals

Postorder: visit the children first before parent

Recursive:

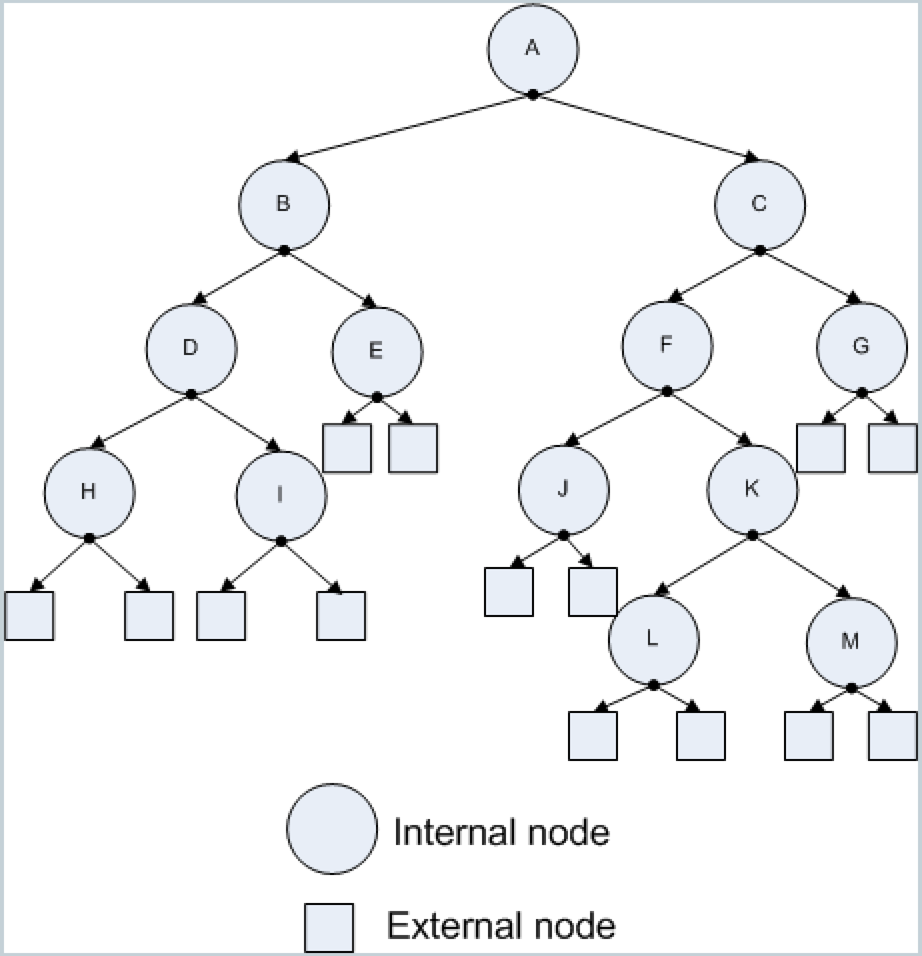
```
postorder(v){
        for each child w of v
                postorder(w)
        visit(v)
}
```

Traversals



Postorder: K, L, E, F, B, G, M, N, H, C, I, J, D, A

Binary Tree



Tree T with n nodes, let h be the height:

1. External nodes of T: at least h+1 and at most 2^h
2. Internal nodes of T: at least h and at most 2^h – 1
3. Total number of nodes: at least 2h + 1 and at most 2^(h+1) – 1
4. Height h is at least log(n+1) – 1 and at most (n-1)/2, that is log(n+1) – 1 <= h <= (n-1)/2

Binary Tree

Inorder: left child, parent, then right child

Recursive:

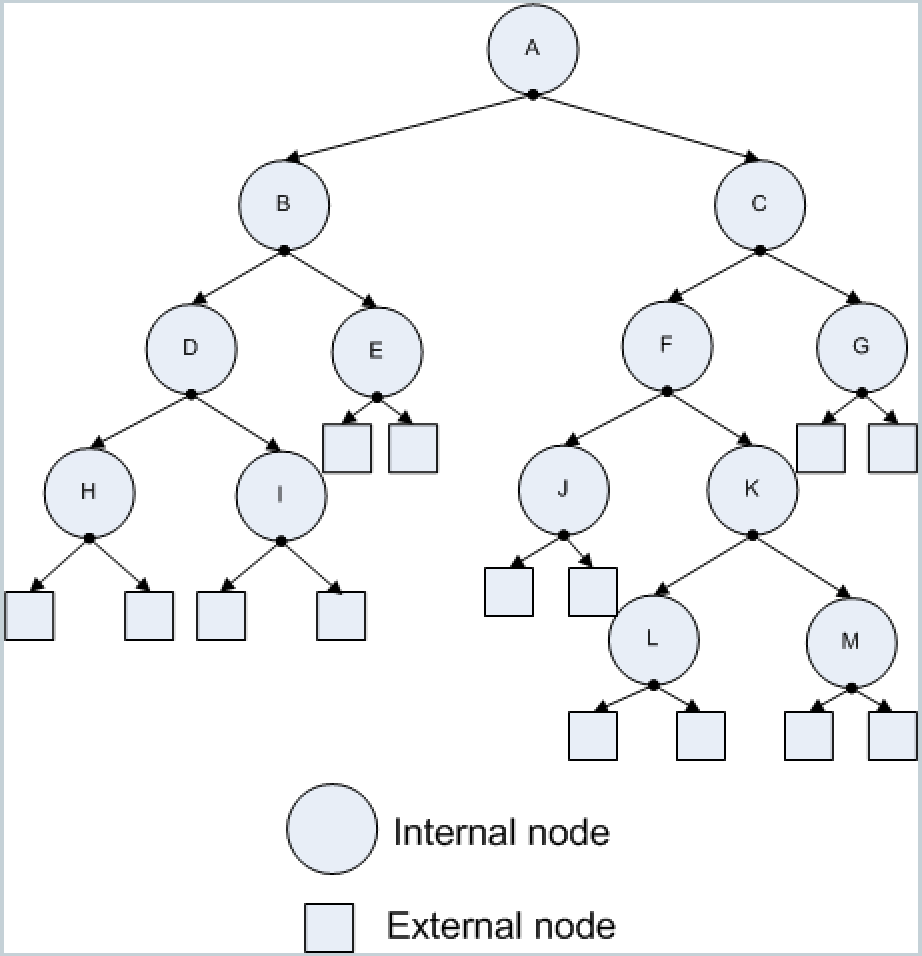
```
inorder(v){
    inorder(v.leftchild);
    visit(v);
    inorder(v.rightchild);
}
```

Binary Tree

Iterative:

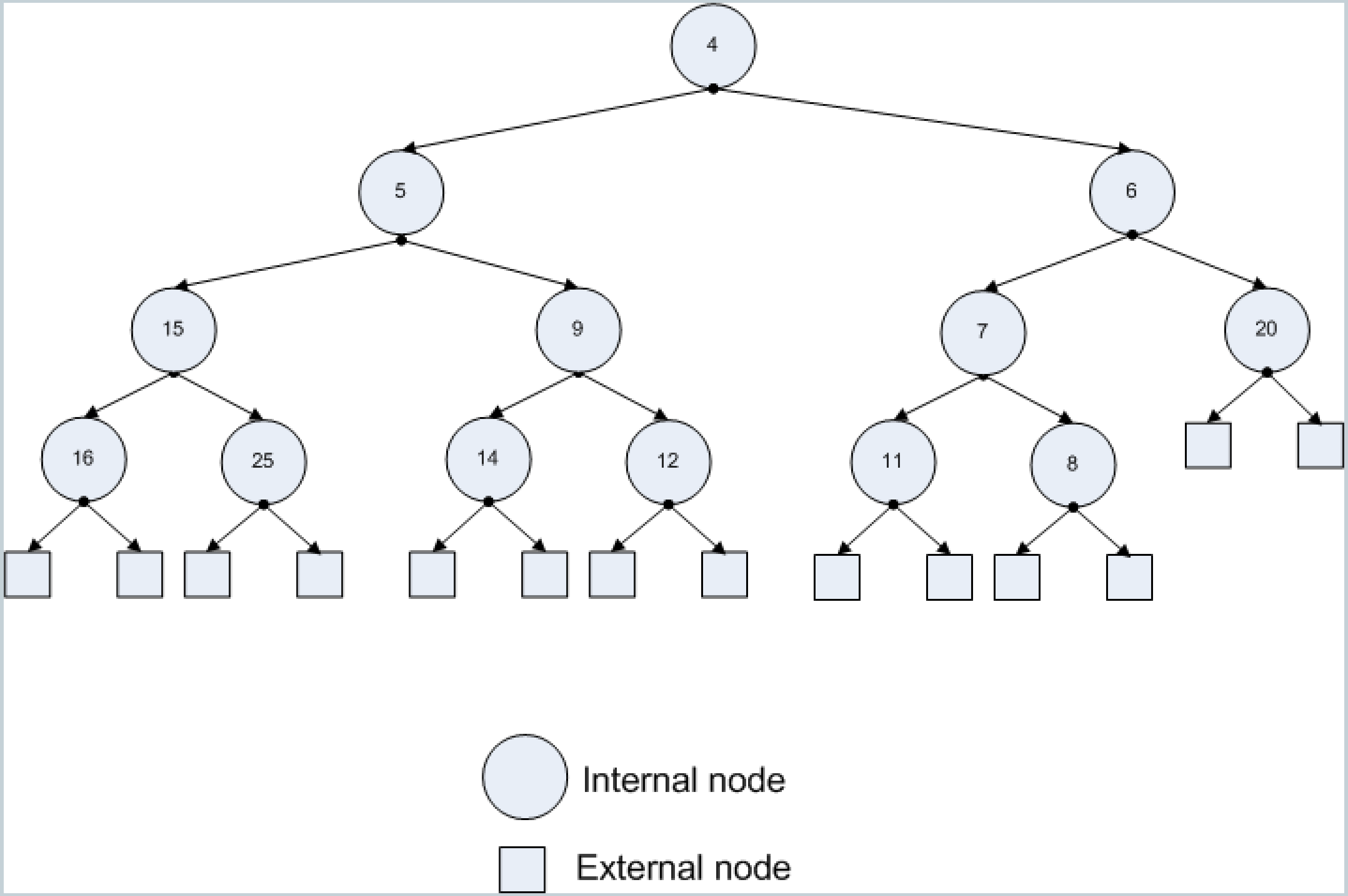
```
inorder(v){
    stack s;
    iterator it;
    it = v;
    while(it != null){

        while(it != null){
            if(it.right != null) s.push(it.right);
            s.push(it);
            it = it.left;
        }
        it = s.pop();
        while(s.size() > 0 && it.right == null){
            visit(it);
            it = s.pop();
        }
        visit(it);
        it = s.size()>0? s.pop():null;
}
```

Binary Tree



Inorder: H, D, I, B, E, A, J, F, L, K, M, C, G

Heap



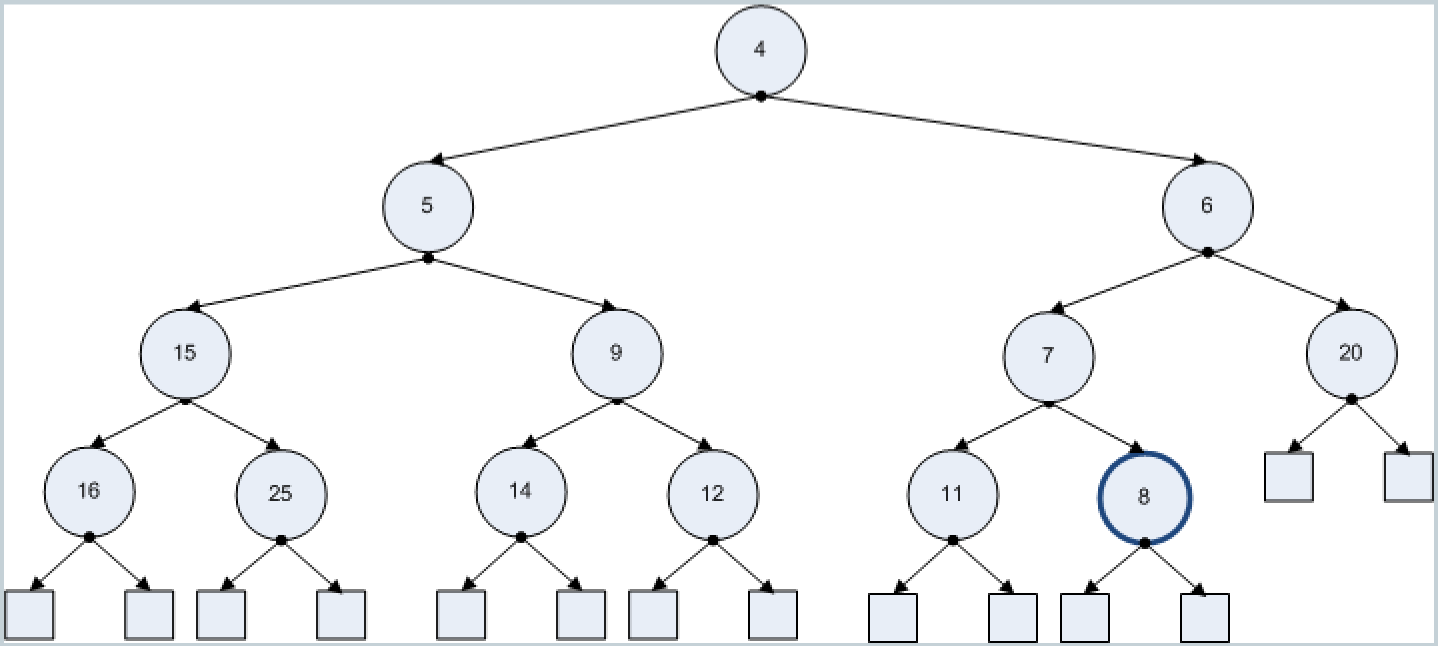
Heap

- Heap is a realization of priority queue using binary tree data structure
- A heap has two properties:
  - Heap-Order property: A key store at a node v is greater than that of v's parent
  - Complete Binary Tree: A binary tree T is complete if the levels 0,1….h – 1 has the maximum number of nodes possible

Heap

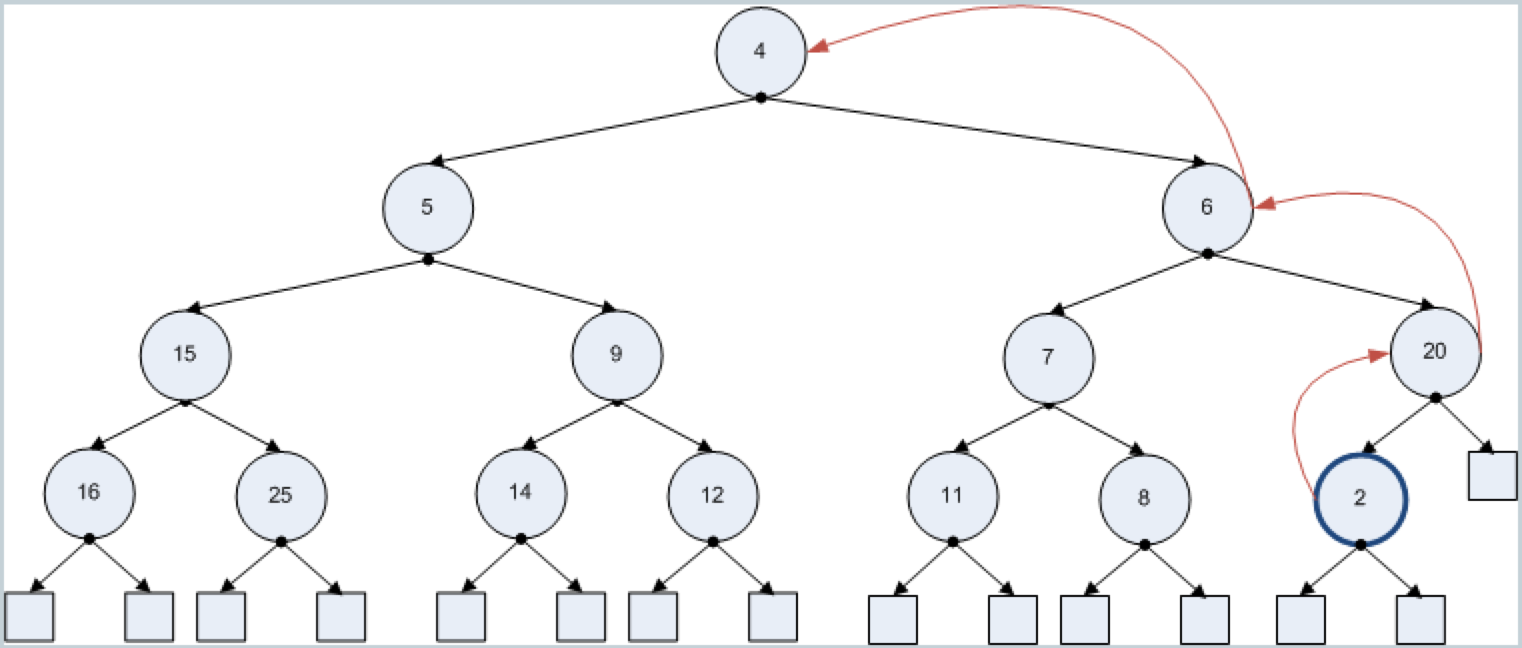
- Insertion: where to insert, perform upheap
- Keep track of the last node
  - Case 1: last node is a left child, insert the new value into the right child
  - Case 2: last node is a right child, go up the branch until you reach a left child, traverse down its right sibling node, traverse down its left branch until the lowest node is reached, insert the new value into its left child node
  - Case 3: the tree is empty, insert the new value into its root
  - Case 4: last node is the right most node, i.e. the last level is full, insert the new value into a left node starting a new level

Heap



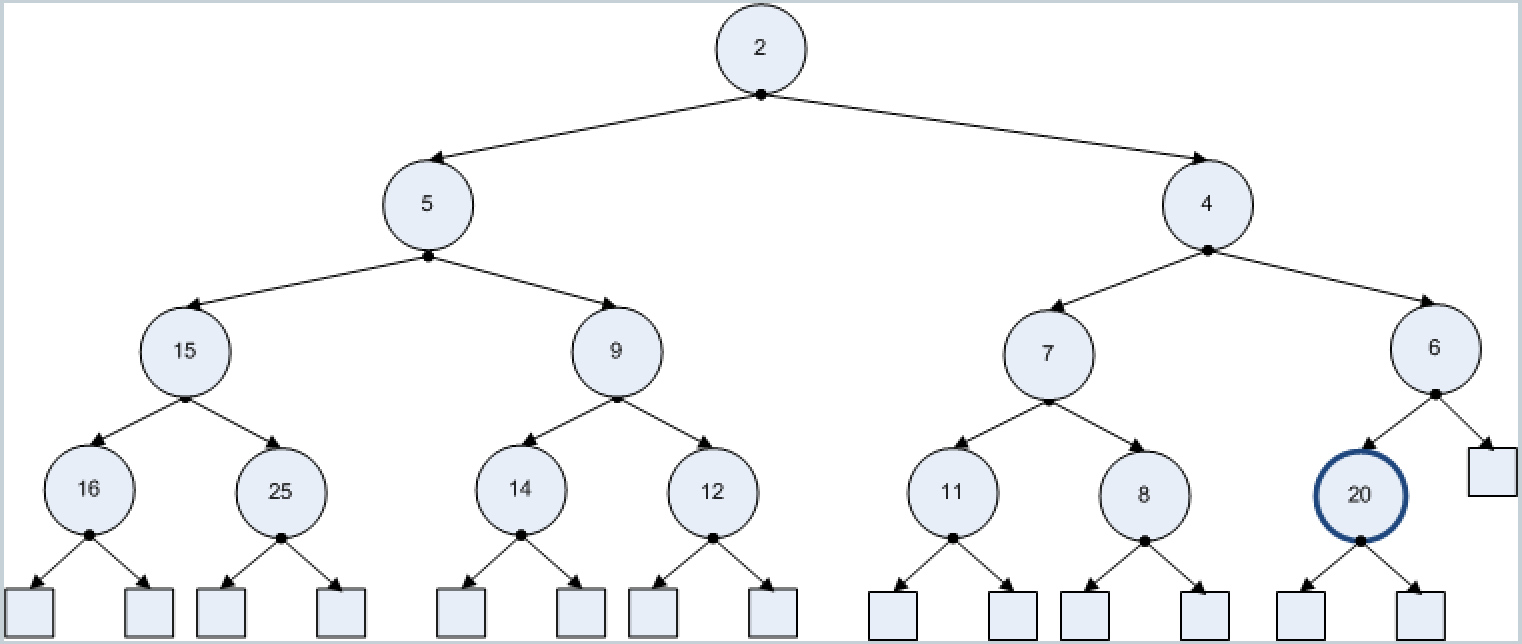
Case 1: inserted 8, last node was 11, the ordering is good, no upheap needed

Heap



Case 2: inserted 2, need to perform upheaps until the keys are reordered, 3 upheaps needed

Heap

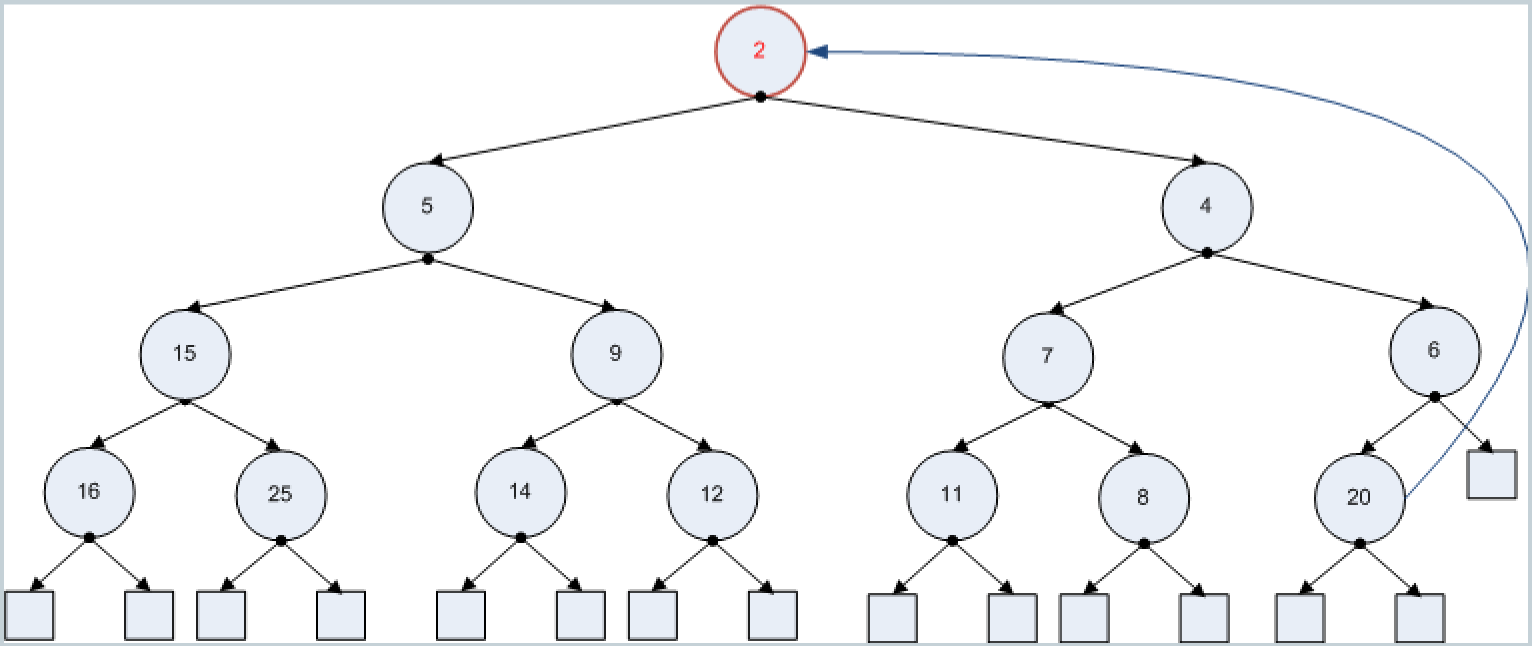


Inserted 2 and reordered

Heap

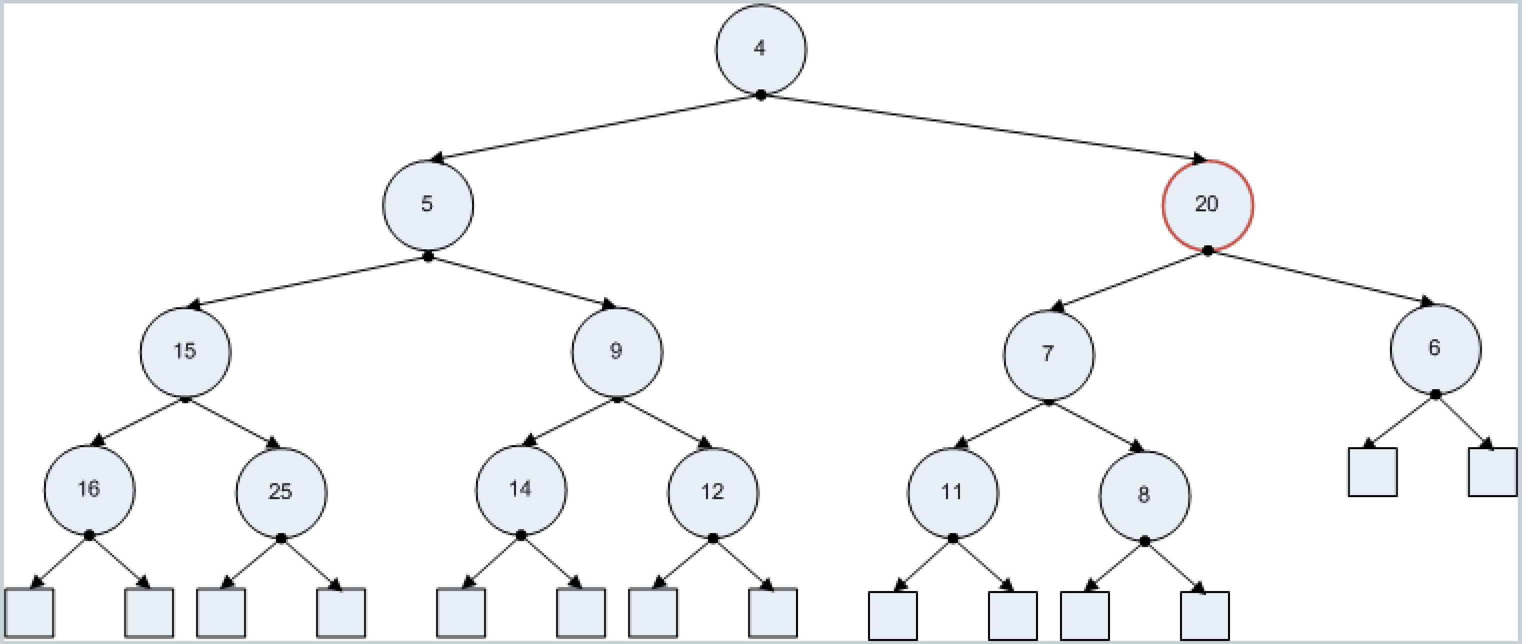
- Removal: remove the key at root, replace root with key at last node, downheap to reorder the tree
  - Case 1: key at r is removed, both children of r are external nodes, nothing to be done
  - Case 2: key at r is removed, left child s of r is an internal node while right child v is an external node. If key(r) > key(s), downheap on s until the tree is reordered.
  - Case 3: key at r is removed, left child s and right child v are both internal nodes. Let w be the child node with the smaller key, if key(r) > key(w), downheap on w until the tree is reordered.

Heap



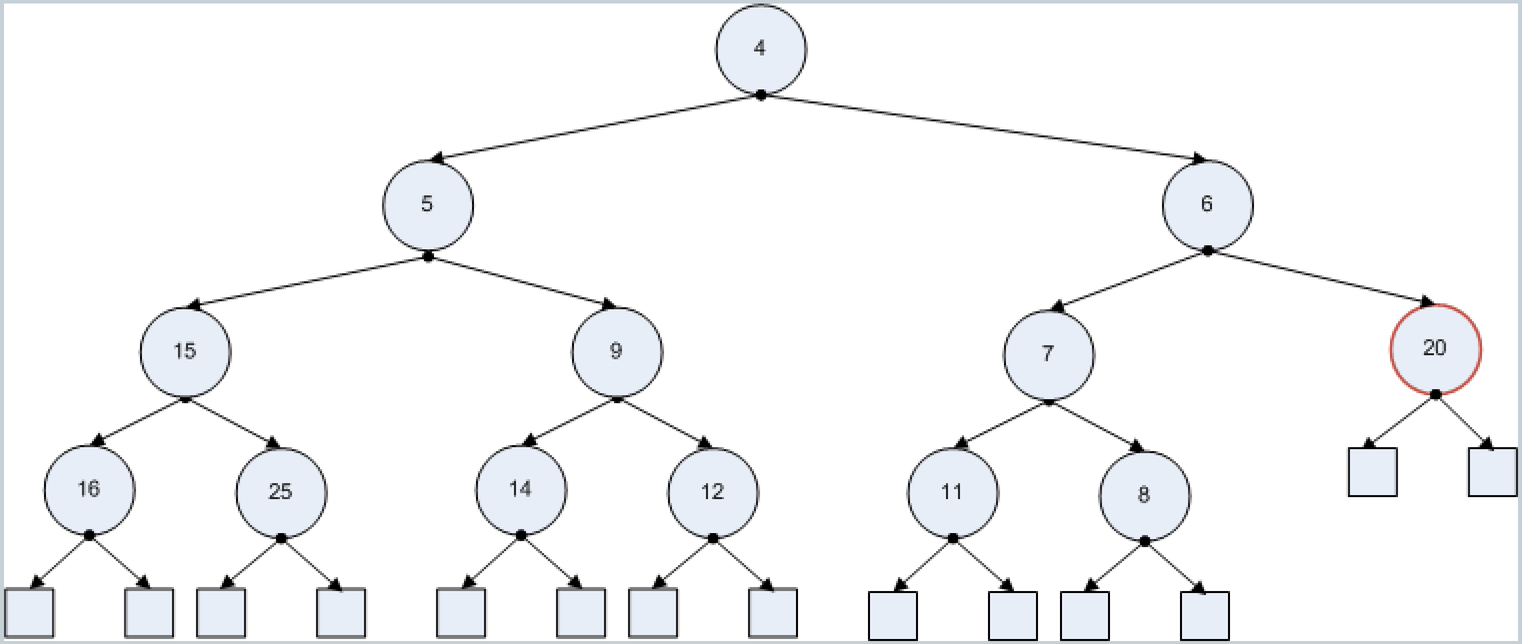
Remove key 2: replace key at root with key at last node, 20.
Downheap on 4, which is smaller than 5, until tree is reordered.

Heap



Downheap on 6 which is smaller than 7.

Heap



The tree is reordered

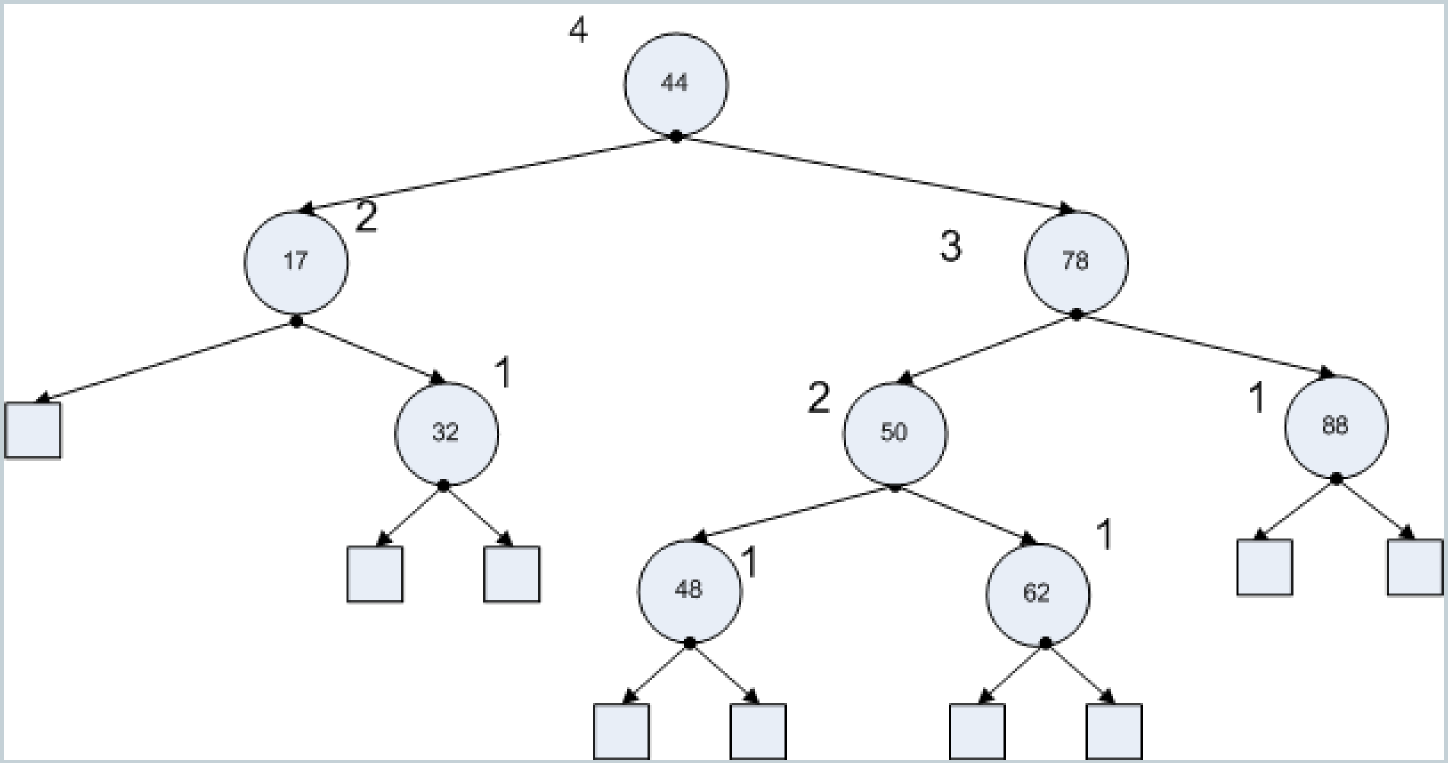
AVL Tree

- Binary search tree with a balanced property
- Height-Balance Property: For every internal node v of T, the heights of the children of v can differ by at most 1
- Need an algorithm to detect imbalance

AVL Tree

- Store the inverse value of the height of the branches at each node.
  - External node has a height value of 0
  - Internal node has a value of the height of its longest branch from the external node

AVL Tree



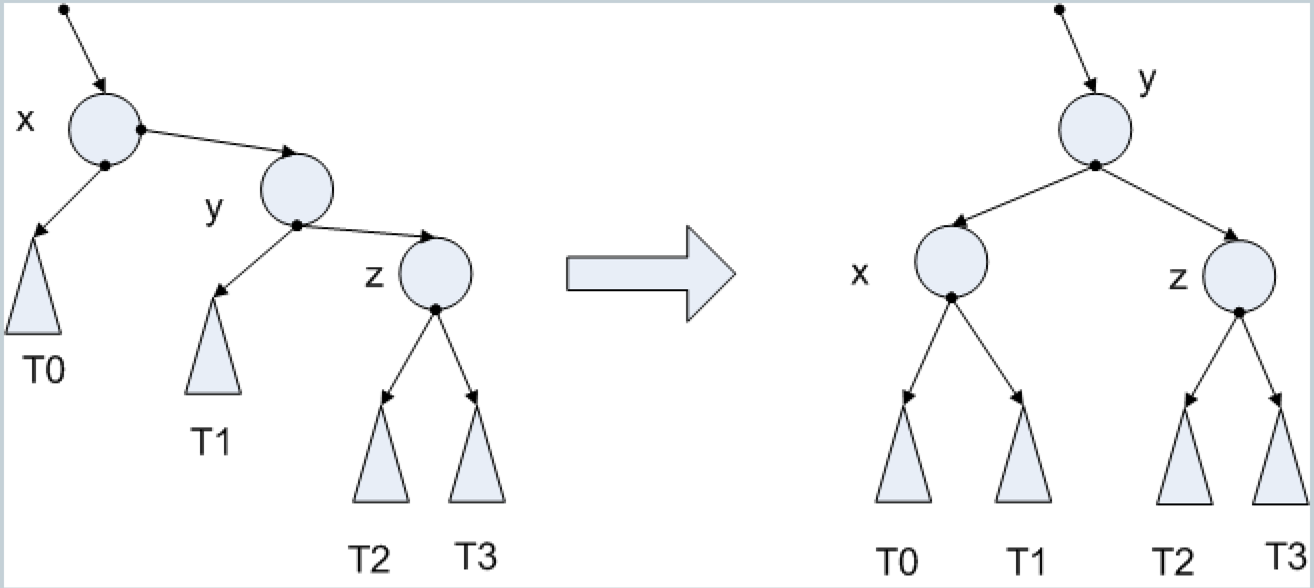
Note: the height value of each node is the maximum value of the longest branch from the external node

AVL Tree

- Insertion and deletion can cause the tree to become imbalanced
- Need an algorithm to restructure the tree to restore balance

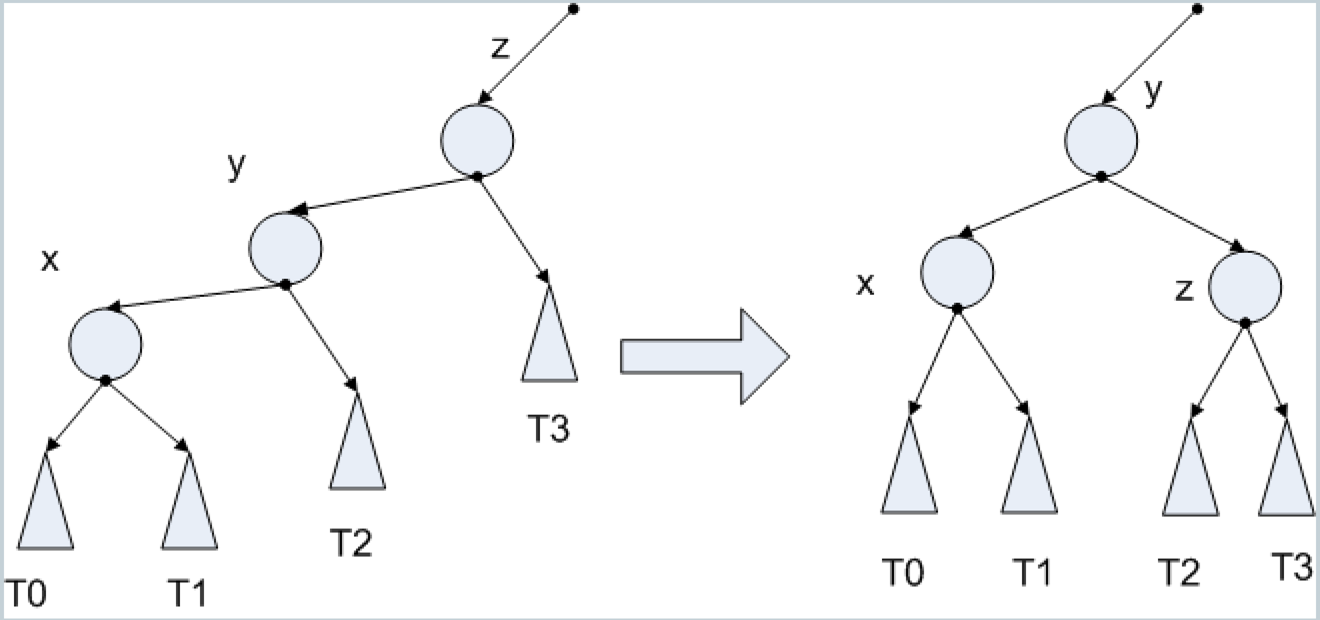
AVL Tree

- 4 strategies to restructure the tree
- Case 1: single left rotation



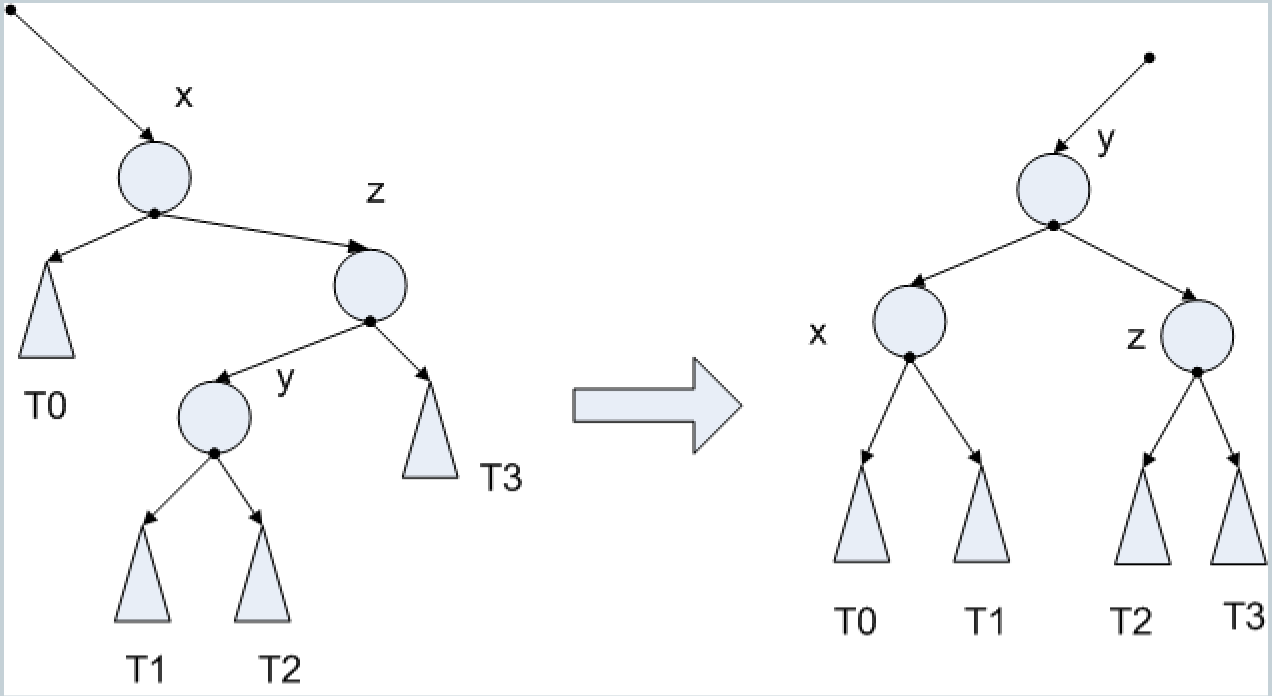
AVL Tree

- Case 2: single right rotation



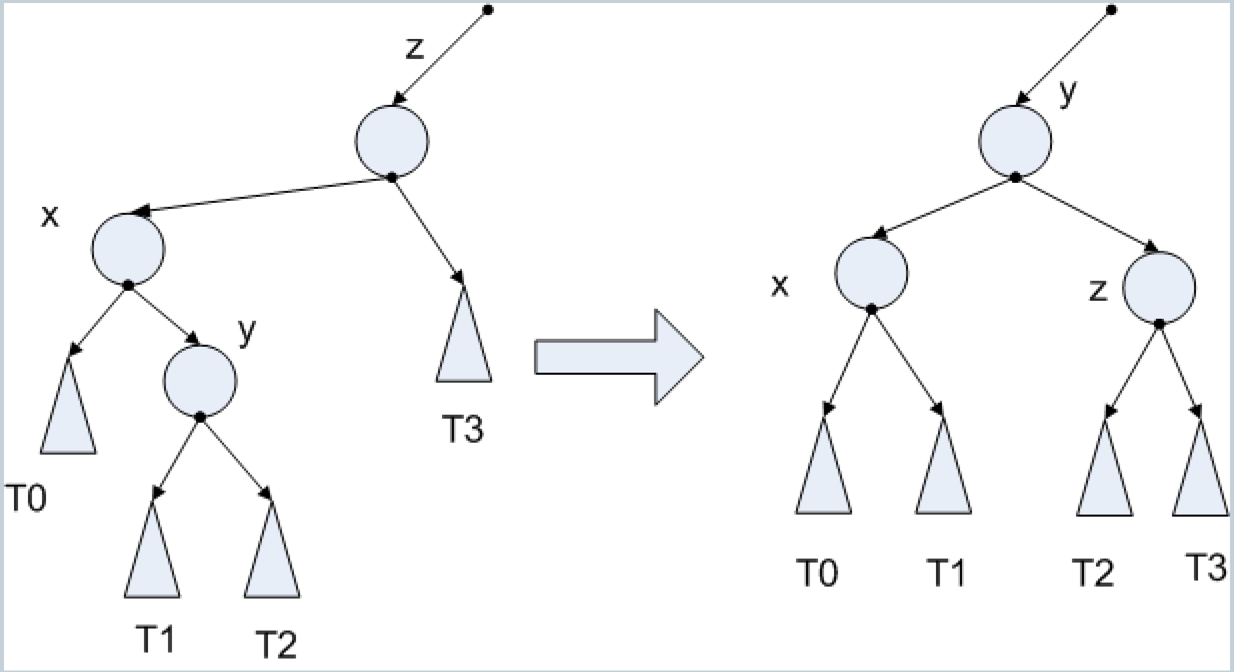
AVL Tree

- Case 3: double left rotations



AVL Tree

- Case 4: double right rotations

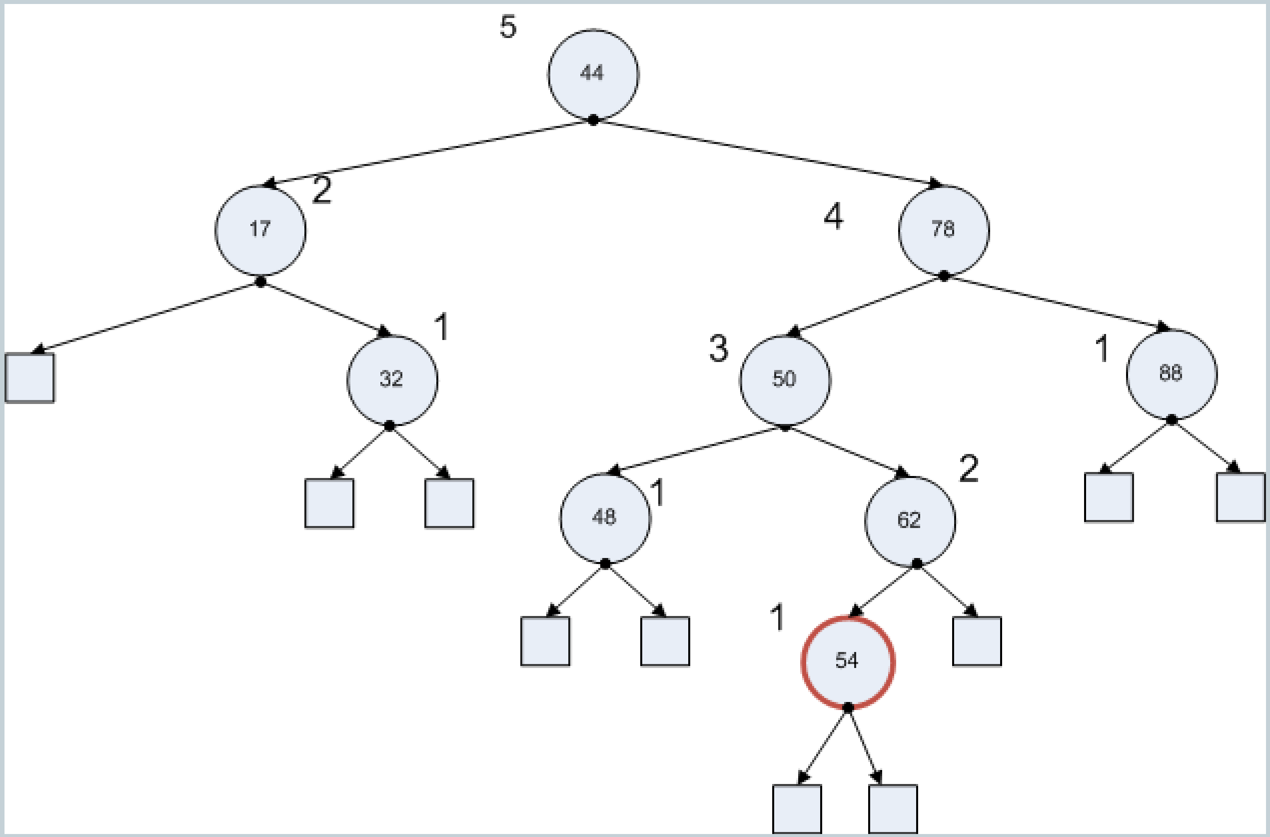


AVL Tree

- Need an algorithm to decide where to restructure in a tree

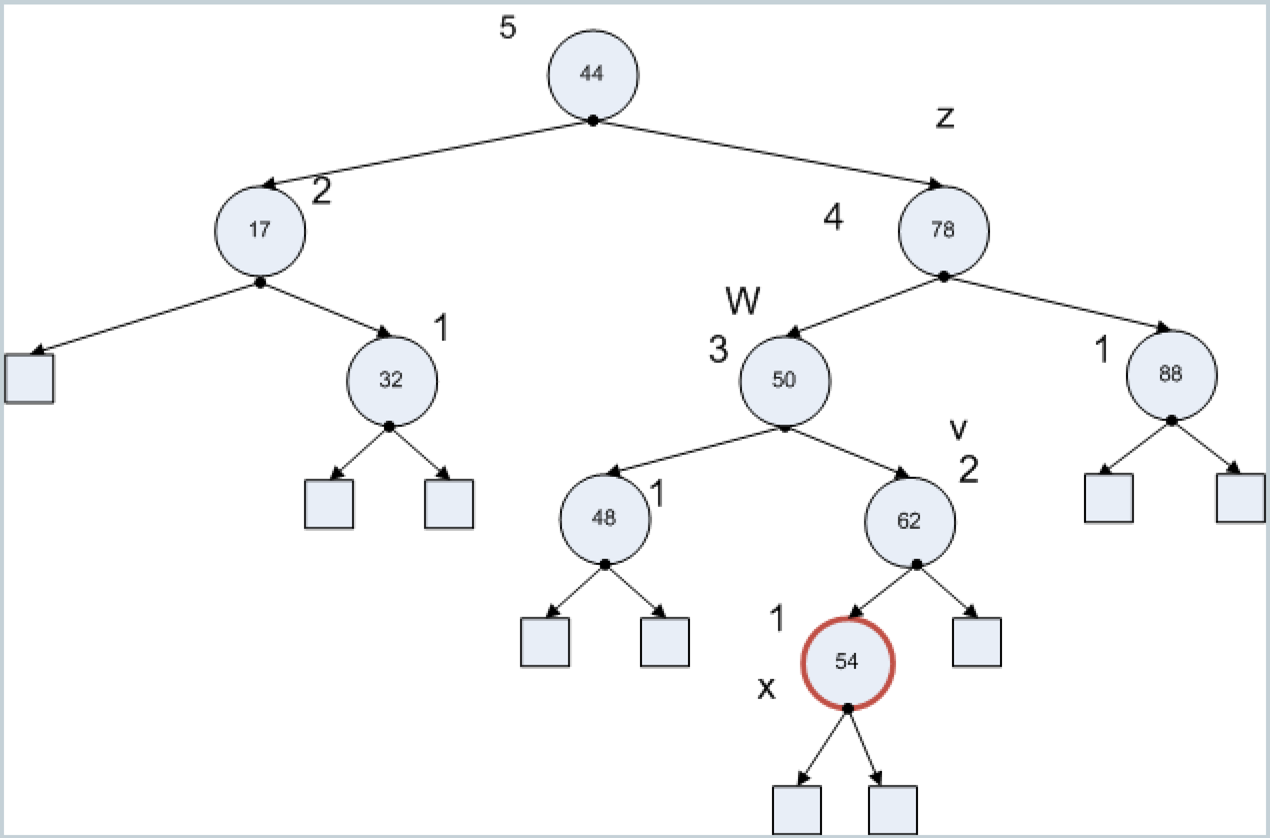
- Insertion:
  - Let x be the node that is inserted. Go up the branch along x until a node z is detected where z's subtrees are imbalanced
  - Let w be z's child and v be w's child along x's branch. Restructure w, v, z

AVL Tree



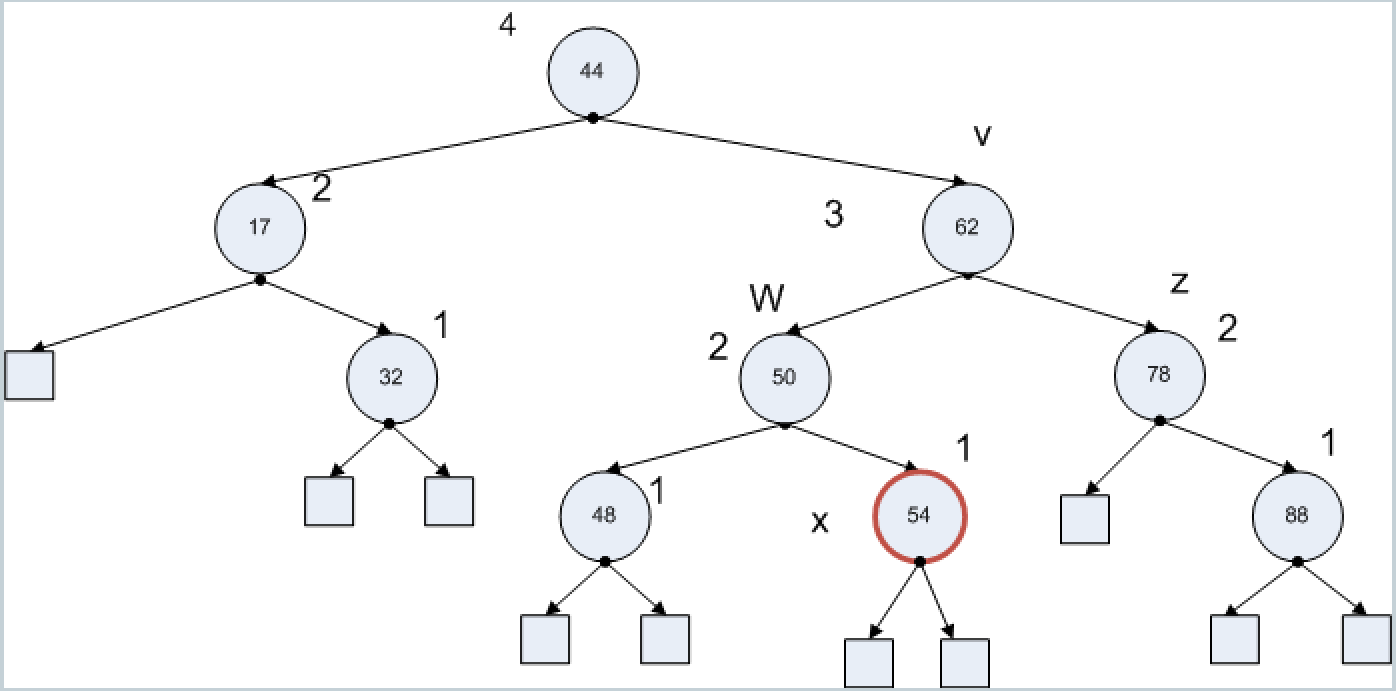
Insertion of 4 caused the tree to be imbalanced

AVL Tree



Use double right rotation on z, w, v

AVL Tree



Balance is restored

AVL Tree

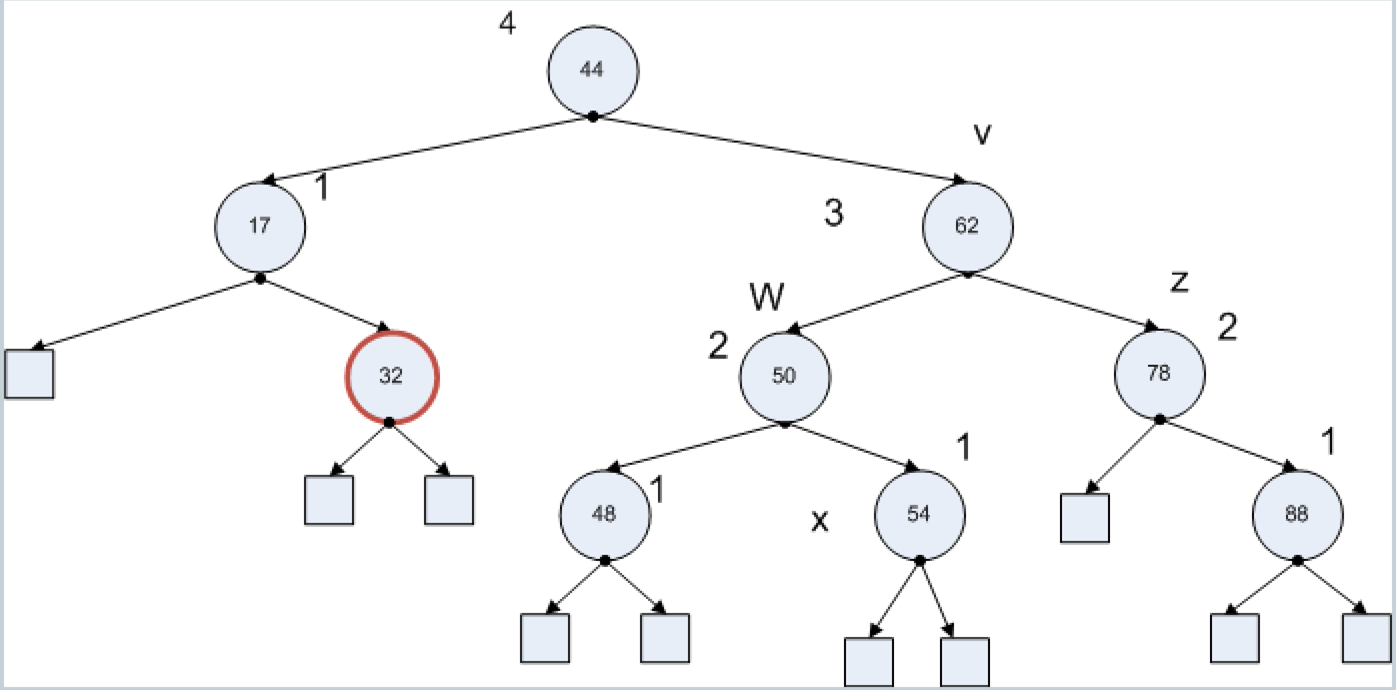
- Removal can cause the tree to become imbalance

- Where to restructure:
  - Let w be the node removed
  - Let z be the first imbalanced node encountered going up from w
  - From z, pick the child y of z that has the highest height value.
  - From y, pick the child of x of y that has the highest height value.
  - Restructure z, y, x

AVL Tree

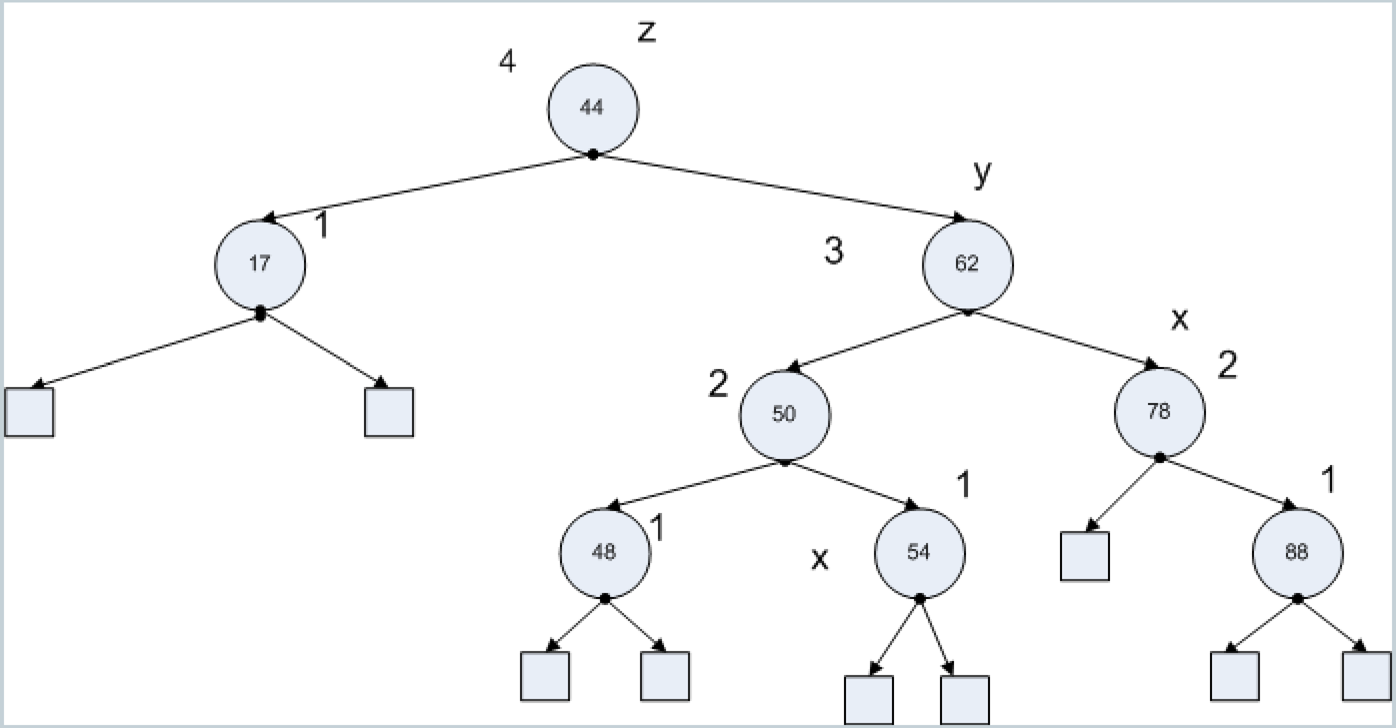
- Note: if the children of y has the same height values, then an arbitrary child can be picked, but multiple restructurings might be necessary depending on the choice of x, y, z

AVL Tree



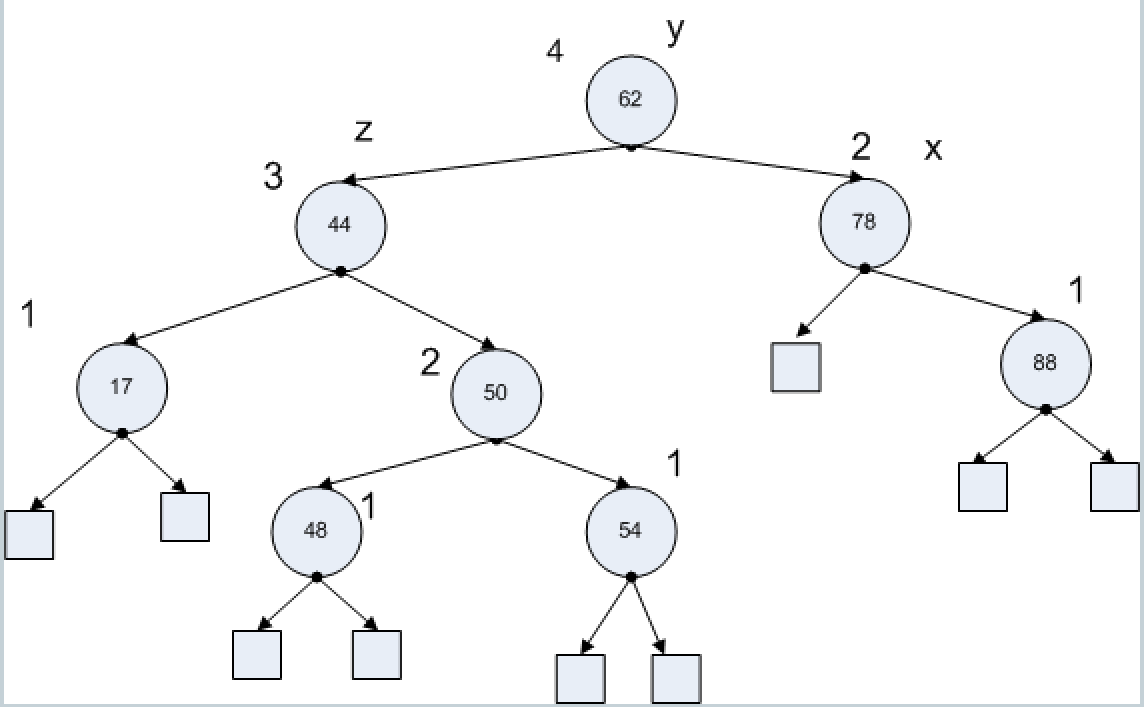
Removing 32 causes the tree to become imbalanced

AVL Tree



Going up along 32 branch, z is the first node encountered that is imbalanced, 17 has a value of 1, and 62 has the value of 3. Pick 62 as y. Since 50 and 78 both have a value of 2, arbitrarily pick 78 as x.

AVL Tree



Use single left rotation to restructure x, y, z