Session 04

Trees - I

Overview

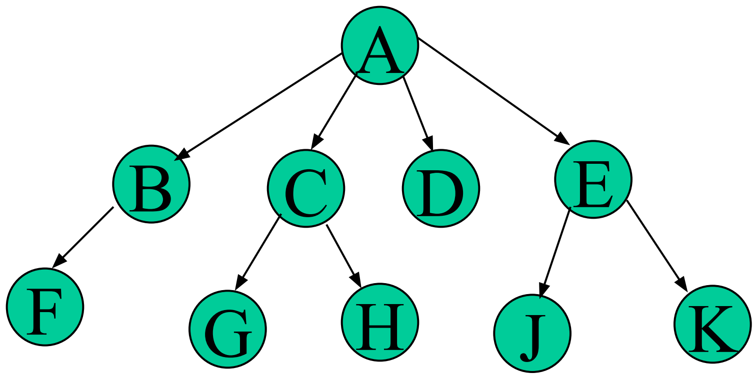
- Binary trees
  - Definition, Representation & implementation.
  - Binary search trees –construction, search and deletion.
- Tree traversals
  - inorder, preorder and postorder.
- Tree types
  - Search Trees, Expression trees, Degenerated Tree.
- Polish Notation

Introduction

- Lists, stacks, etc. are linear data structures - a unique next element is defined.
- Trees provide a non-linear structure and are used for representing hierarchical data structures.
  - Eg. Family tree
- Many applications exploit this
  - Efficient sorting, searching, priority queues, etc.

N-ary Tree

- Recursive definition of tree
  - Either an empty tree or consists of
    - a root node and
    - $T_1 \ldots T_n$ sub trees,
  - Node stores data value and references to all its sub-trees



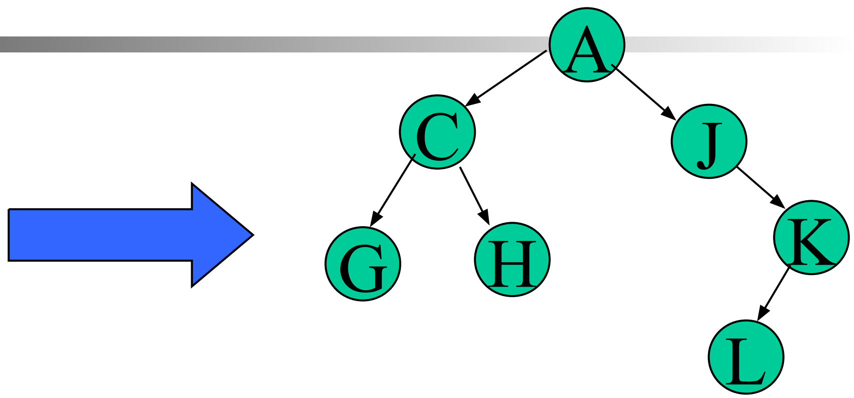
Maximum value of'n' in the tree is the arity of the tree.

Terms

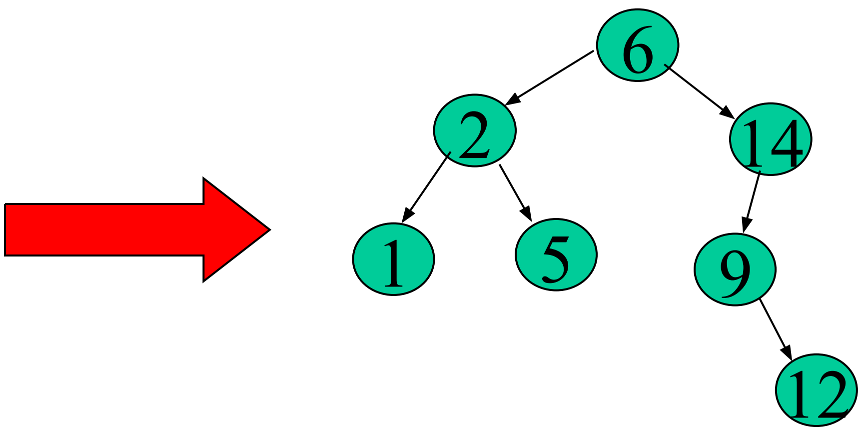
- root
- parent of a node
- children of a node
- sibling of a node
- internal node
- external node/ leaf node
- many terms borrowed from analogy of family tree and real tree.

Binary Tree

- An N-ary tree with atmost most two sub-trees
  - a left subtree (LST)
  - a right subtree (RST)

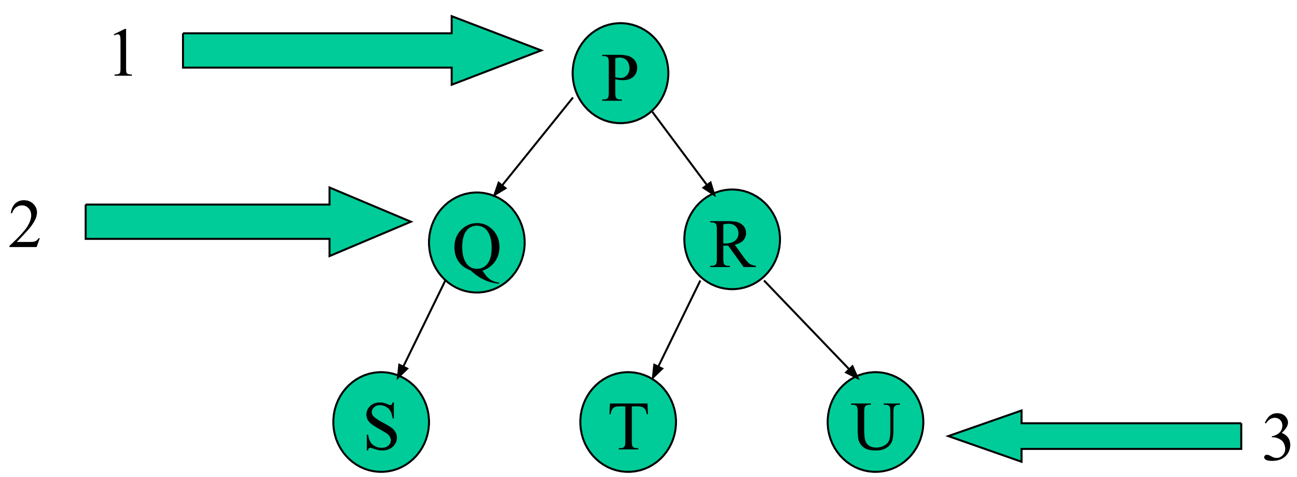


If value of LST, value of node and value of RST are in strict order for all nodes in the tree, we have a **Binary Search Tree**



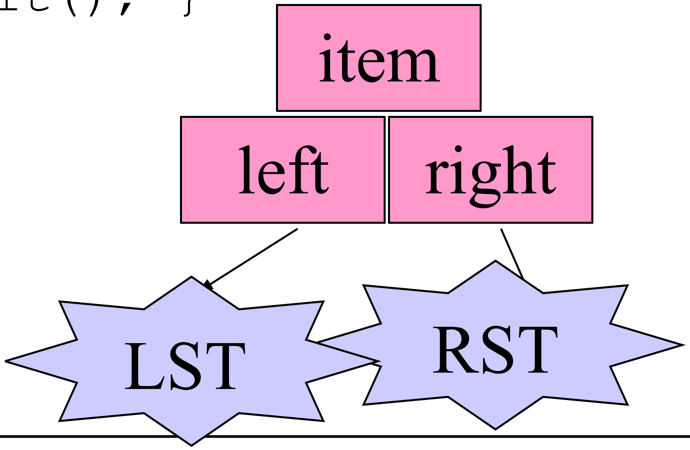
Height and Level

- level of root = 1
- level of a node = one + level of its parent
- height of a tree = maximum level of all nodes
- same as 1 + maximum of the height of its subtrees.



BST Implementation

```
class TreeNode{
    private Info item;
    private TreeNode left;  // reference to LST
    private TreeNode right; // reference to RST
    public TreeNode(){this(0);}
    public TreeNode(int newItem)
    {this.item = new Info(newItem), left=right=null;}
    public void visit() {item.visit(); }
}
```



BST Implementation..

```
class Info{
  int key;  // unique field of Info
   // …. Other data members
  public Info(int key){this.key = key;}
  // …. Other constructor to initialize Info object
  public boolean isLessThan(Info infoObj)
  {return key < infoObj.key}
  public boolean equals(Info infoObj)
  { return key==infoObj.key}
  public void visit()
  { System.out.println(key+" "+//additional data;}
  // …. get/set methods for the data members}
```

Implementation

- Variations used depending on application - We will ignore variations largely.
- We need a class to hold the tree and support operations such as–
  - insert an element
  - delete an element
  - search an element
  - traversal.

Implementation…

```
public class BinaryTree {
 private TreeNode root;
 public BinaryTree (){ root=null;}
 public boolean isEmpty(){ return root == null; }
 public TreeNode search(Info key){return
search(root,key);}
```

Implementation…

```
public void insert(Info item) { // }
 public void preorder() {preorder(root);}
 public void inorder() {inorder(root);}
 public void postorder() {postorder(root);}
 public void delete(Info item) { // }
 private TreeNode search(TreeNode root, Info
   key); }
```

Searching in BST

Looking at a node, we know whether to proceed left or right.

```
public TreeNode search(TreeNode p, Info el){
  if (p != null){
   if(el.equals(p.item)) return p; //Found
   else if (el.isLessThan(p.item))
   return search(p.left, el);
   else
   return search(p.right, el); }
  return p;
```

Traversal

- Means visiting all nodes of a tree, in some order, systematically.
- In general many traversals are possible (= n!, n is the number of nodes in the tree)
- Must ensure that all nodes are visited, once and only once.
- Two common ways to traverse the nodes
  - Breadth-first traversal (BFT)
    - Visiting all nodes at particular level before next level.
  - Depth-first traversal (DFT)
    - Traverse one sub-tree fully before moving to another.

Traversal

- As per the definition, there are three components at a node of a tree.
- Three different traversals commonly followed - PRE, IN, POST order decided by the order of visiting these components.
- We assume LST is visited before RST anyway - option decided by when to visit root.

Preorder Traversal

```
protected void preorder(TreeNode root){
   if (root != null){
      root.visit();
      preorder(root.left);
      preorder(root.right);
   }
}
```

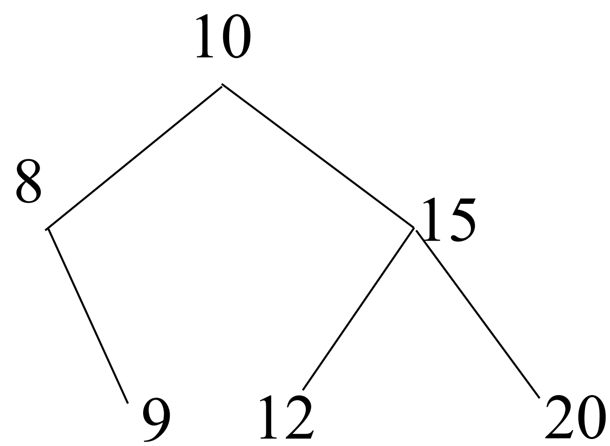
Inorder Traversal

```
protected void inorder(TreeNode root){
  if (root != null){
    inorder(root.left);
    root.visit();
    inorder(root.right);
  }
}
```

Postorder Traversal

```
protected void postorder(TreeNode root){
   if (root != null){
      postorder(root.left);
      postorder(root.right);
      root.visit();
   }
}
```

Traversal of a Search Tree



Pre : 10 8 9 15 12 20

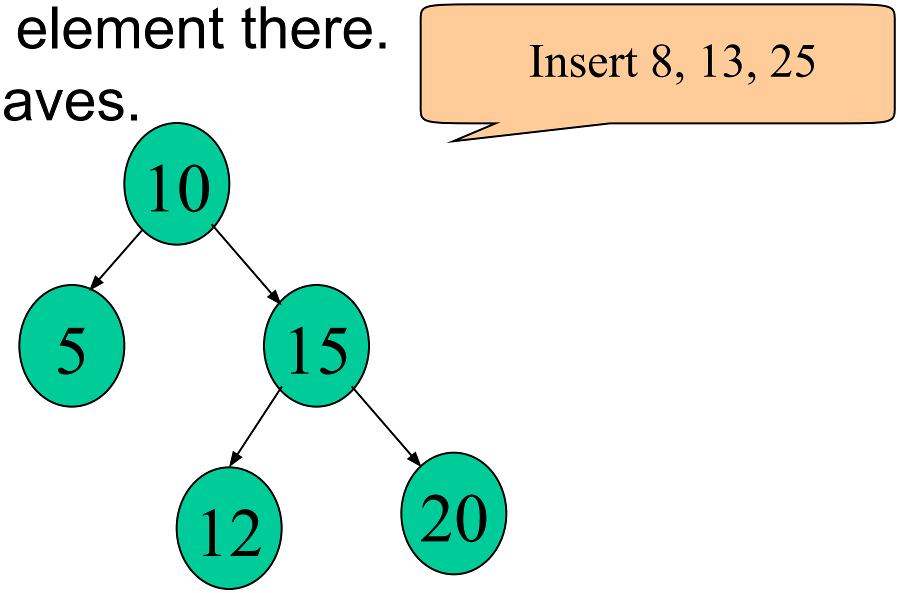
Post: 9 8 12 20 15 10

In : 8 9 10 12 15 20

- Inorder traversal generates sorted order!

Constructing BST

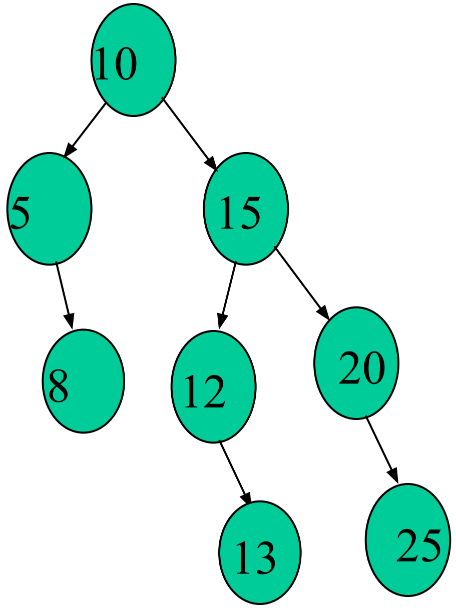
- If tree is empty, make the new node root of the tree.
- If tree is not empty, Search in the existing tree for the element to be inserted.
- We will reach a leaf node - the element is to be added as the left or right child of that node.
- Create that child and put the element there.
  - growth is always at the leaves.



Constructing BST

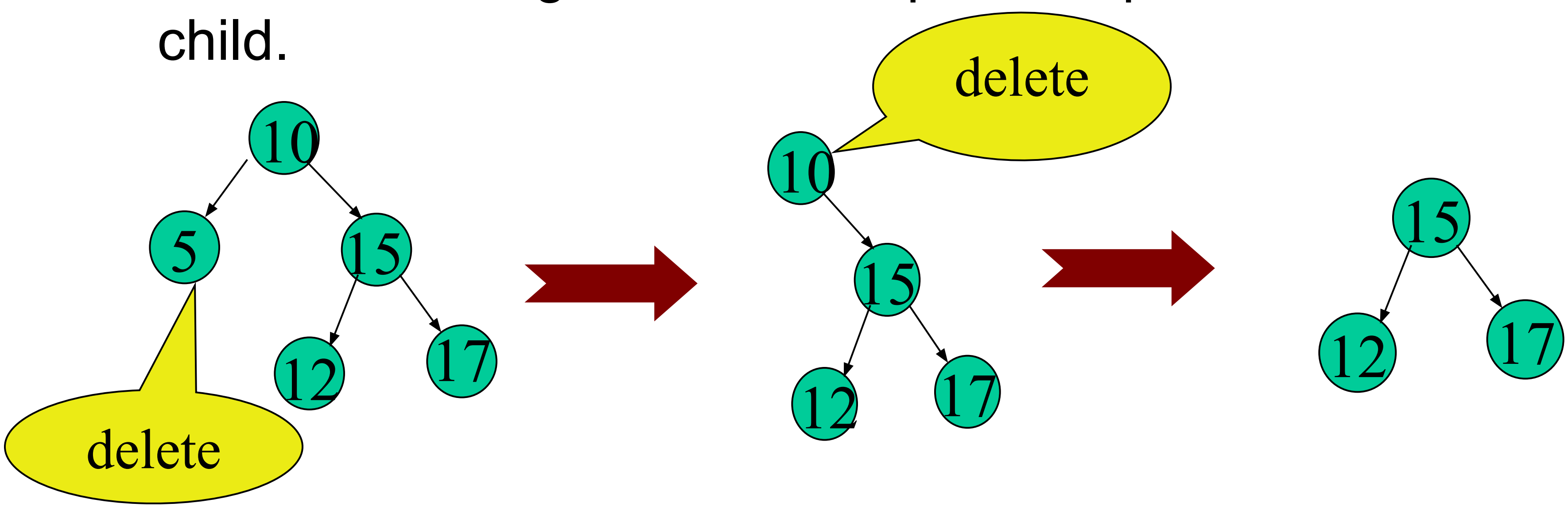
```
public void insert(Info item){
  TreeNode p = root, prev = null;
  if(root == null) {// tree is empty
   root = new TreeNode(item); return;  }
  while(p!=null){  // find a place for inserting new node
   prev = p;
   if(p.item.isLessThan(item)) p = p.right
   else p = p.left;  }

  if(prev.item.isLessThan(item))
    prev.right=new TreeNode(item);
  else if(prev.item.isGreaterThan(item))
    prev.left=new TreeNode(item);
}
```



Deletion in BST

- Deletion has to preserve BST criteria.
- Node is a leaf? Just delete the node, set parent's pointer to `null`.
- Node has a single child? Set parent's pointer to the child.

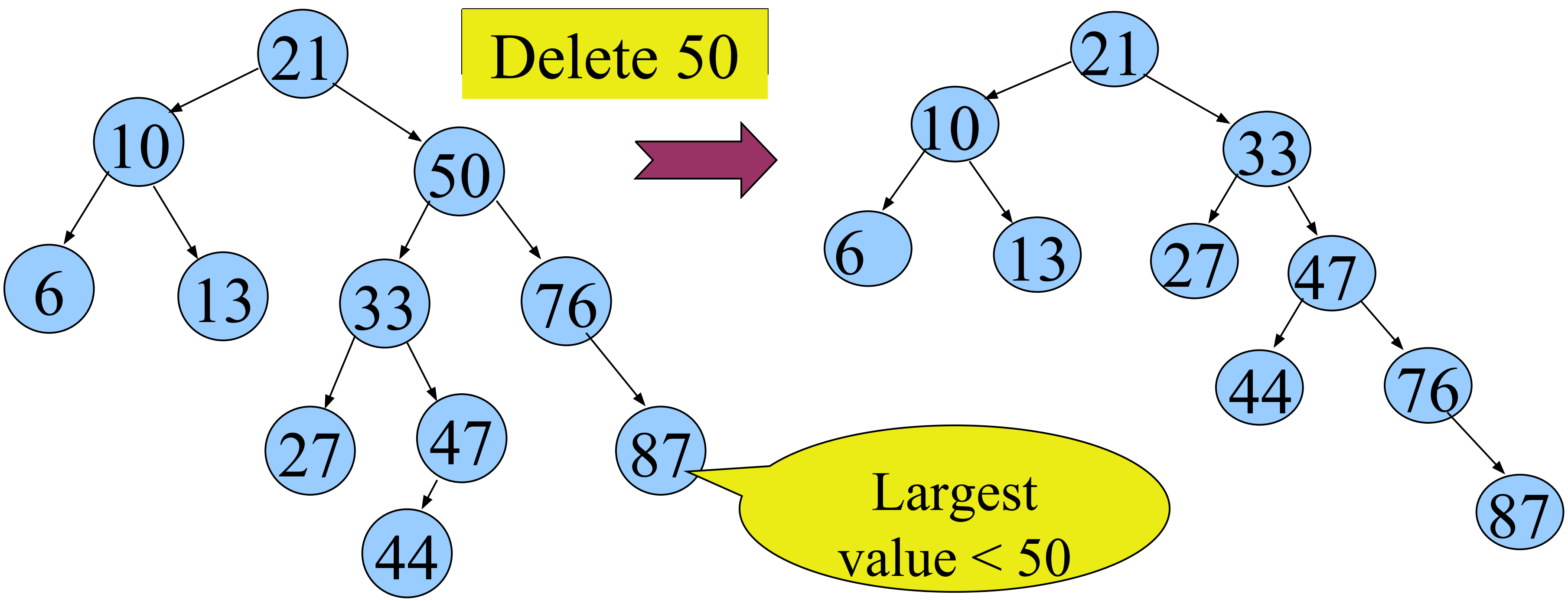


Deletion of Nodes

- Node has both children.
- There are many solutions - must ensure the Search tree property.
- Two ways
  - Deletion by merging
  - Deletion by copying

Deletion by Merging

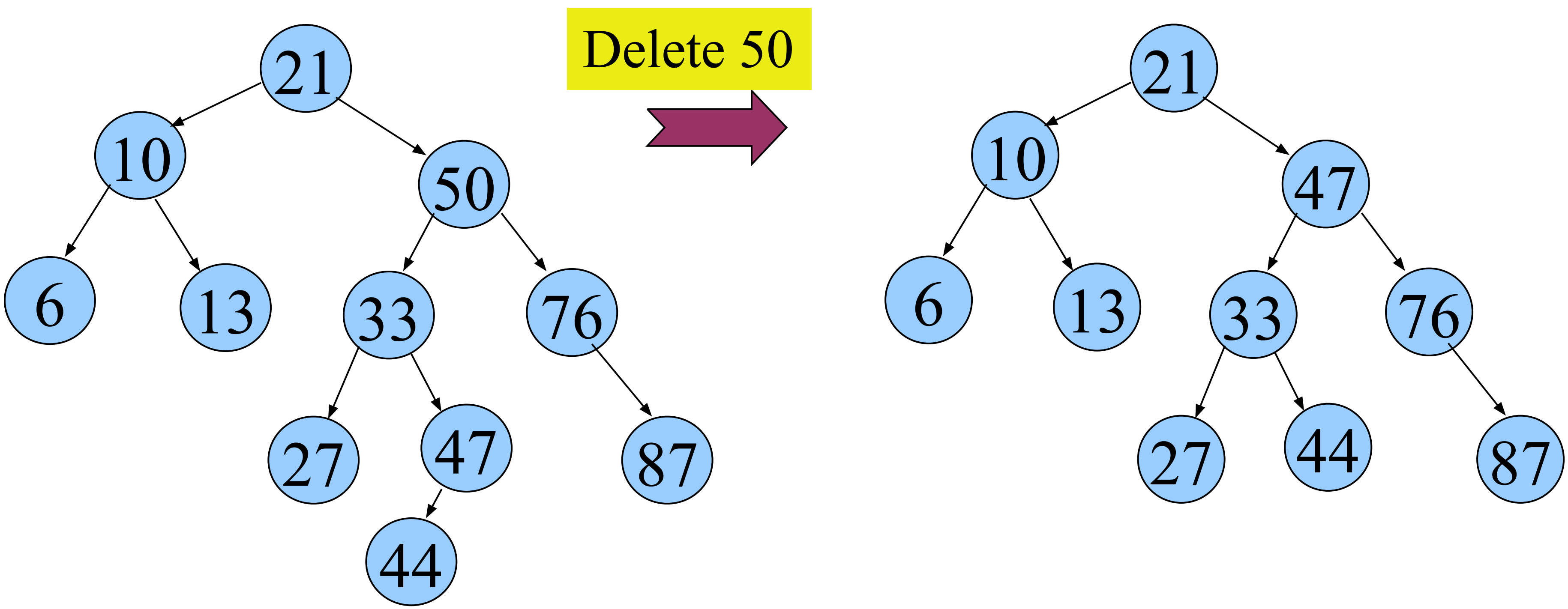
Merge node's RST with its LST and make a single tree by attaching RST as RST of rightmost descendent of LST



Can also merge node's LST with its RST in a similar way.

Deletion by Copying

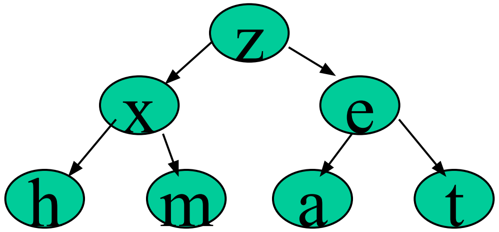
Reducing problem to case 1 or 2 by replacing the node to be deleted with its inorder predecessor(or successor).



Types of Trees

- Expression tree: internal nodes – operators; leaf nodes - operands
- Search tree: info fields of nodes satisfy certain order.
- Complete Binary Tree (CBT): All the internal nodes have both the children and all the leaves are at same level.
- Strictly Binary Tree: Every node has either 0 or 2 children.
- Degenerated tree: Almost like a list.

What could be an input sequence for degenerated binary search tree?



Polish Notation

- Unambiguous binary expression representation by removing parentheses from it.
- Used by compiler/interpreters for expression evaluation.
- Two types
  - Prefix notation - Operator precedes the operands.
  - Postfix notation - Operator succeeds the operands.

Polish Notation…

- The traditional form is called infix (a+b) * (c+d)

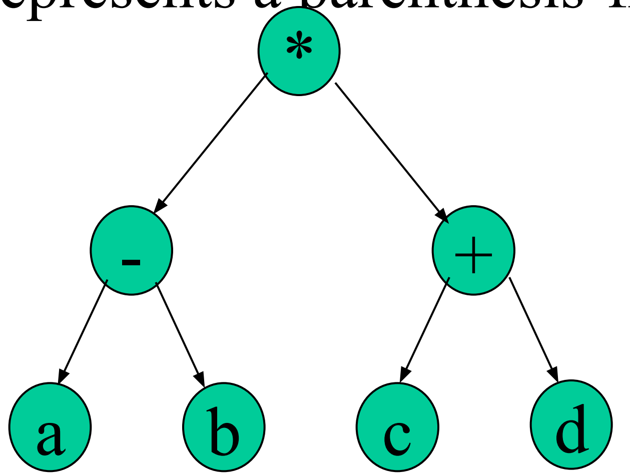
Infix: `(a+b)*(c+d)`

Prefix: `*+ab+cd`

Postfix: `ab+cd+*`

Traversal of Expression Tree

- Represents a parenthesis-free expression.



Preorder: * - a b + c d

Postorder: a b - c d +*

Inorder: a - b * c + d

Problem?

- We get prefix, postfix and infix representation of the expression by suitable traversal.

Application of Binary Trees

- Given a **prefix expression, convert it to infix expression**. Use parentheses to maintain the precedence of operators. Assume only binary operators and single character operands.
- Examples

Prefix		Infix
`+ a b`	→	`(a + b)`
`+ + a b c`	→	`((a + b) + c)`
`* + a b + c d`	→	`(( a + b )*(c + d ))`

Prefix to Infix Conversion

- Convert prefix expression to binary tree form.
- Perform inorder traversal on the binary tree to get the infix expression.
- For expression tree, we know -
  - Internal nodes will be operators
  - Leaf nodes will be operands

Prefix to Binary Tree Form

```
public TreeNode createExprTree(){
 TreeNode root;
 read(char);
 while(char != '\n'){
   if(char is operator){
     root = new Node(char);
     root.left = createExprTree();
     root.right = createExprTree();
 }
```

Prefix to Binary Tree Form

```
else if(char is operand)
   return new TreeNode(char);
 }// End of while
 return root;
}// End of createExprTree()
```

Summary

- Binary tree – definition and terminologies.
- Binary search trees – construction, searching and deletion.
- Traversals in Binary trees – preorder, inorder, postorder.
- Expression trees – unambiguous representation of expression; prefix & postfix notation