General Trees & Binary Trees

CSC212: Data Structures

Trees

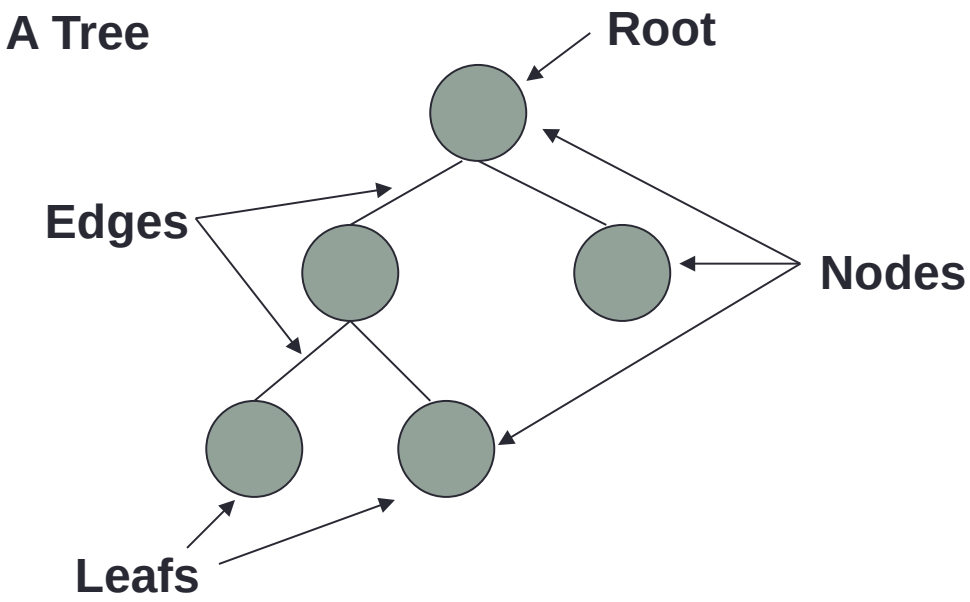
- Previous data structures (e.g. lists, stacks, queues) have a linear structure.
- Linear structures represent one-to-one relation between data elements.
- Trees have a nested or a **hierarchical structure.**
- Hierarchical structures represent **one-to-many** relation between data elements.

Trees

- Examples of situations were one-to-many relations exist… these can be represented as trees.
  - Relation between a parent and his children.
  - Relation between a person and books he owns.
  - Relation between a football team and the players on the team.
  - Card catalog in a library.

Trees: Some Terminology

- A tree is represented as a set of **nodes** connected by **edges**.



Trees: Comparison with Lists

A List

- Unique first element.
- Unique last element.
- Each element, other than the first and the last, has a unique predecessor and a unique successor.

A Tree

- Unique first node called root.
- Each node has successors, called its children.
- Each node has one predecessor, called parent.
- Leafs have no children.
- Root has no parent.

Trees: More Terminology

- **Simple path:** a sequence of distinct nodes in the tree.
- **Path length:** number of nodes in a path.
- **Siblings:** two nodes that have the same parent.
- **Ancestors:** given a node A in a tree, the parent of the node A and the ancestors of the parent of A, are ancestors of A.

Trees: More Terminology

Parent: a parent of a node is its predecessor.

Child: a child of a node is its successor.

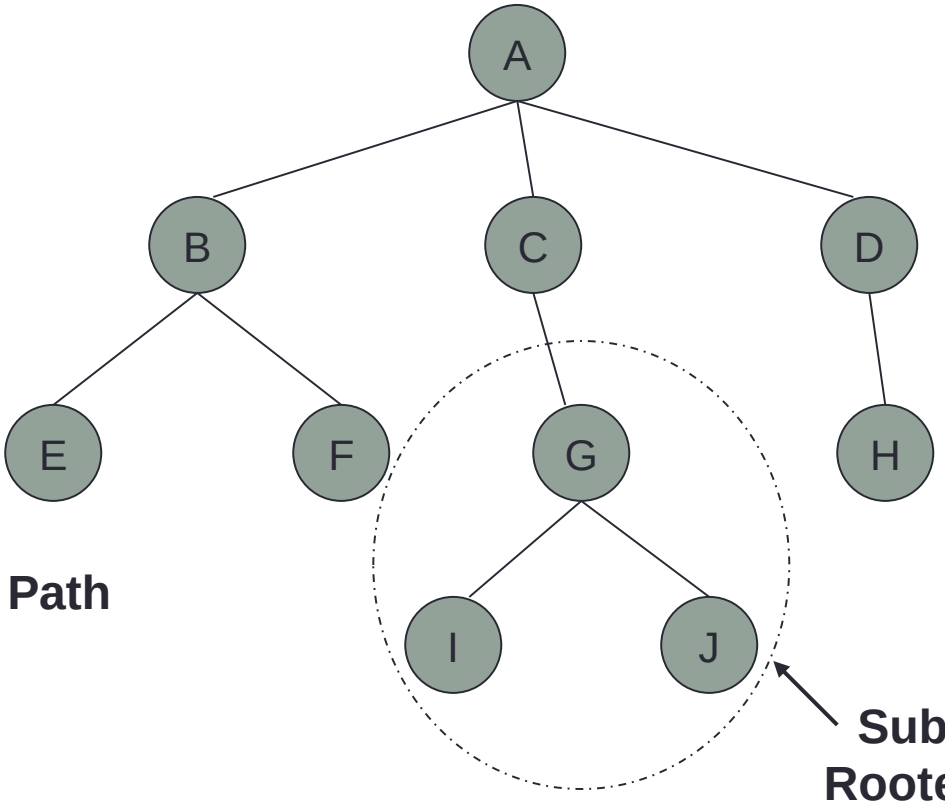
Root: a unique node without any predecessor.

Leafs: nodes without any children.

Descendents: given a node A in a tree, the children of A and all descendents of the children of A are descendents of A.

Trees: More Terminology

?Height



Level 1

Level 2

Level 3

Level 4

A-B-F is a Path

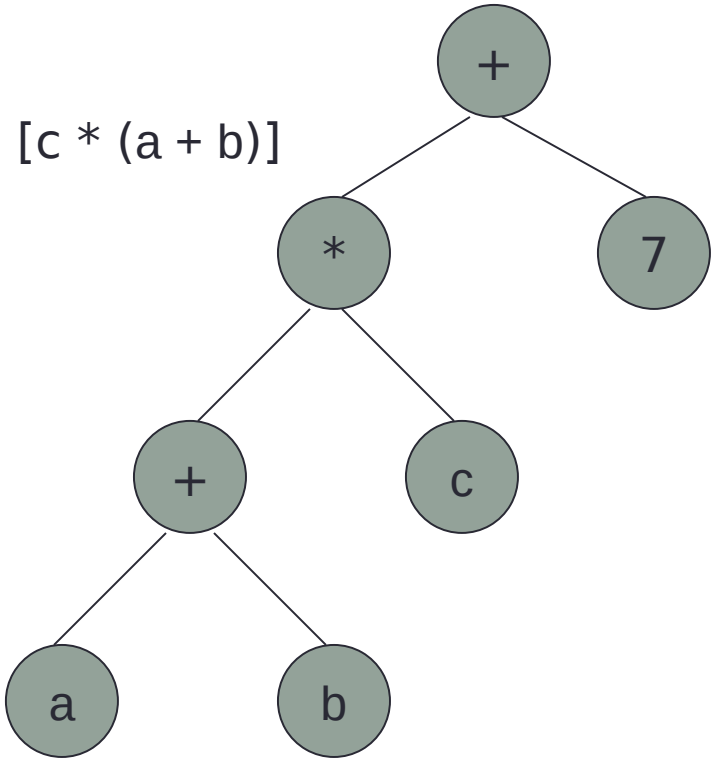
Sub-Tree Rooted at G

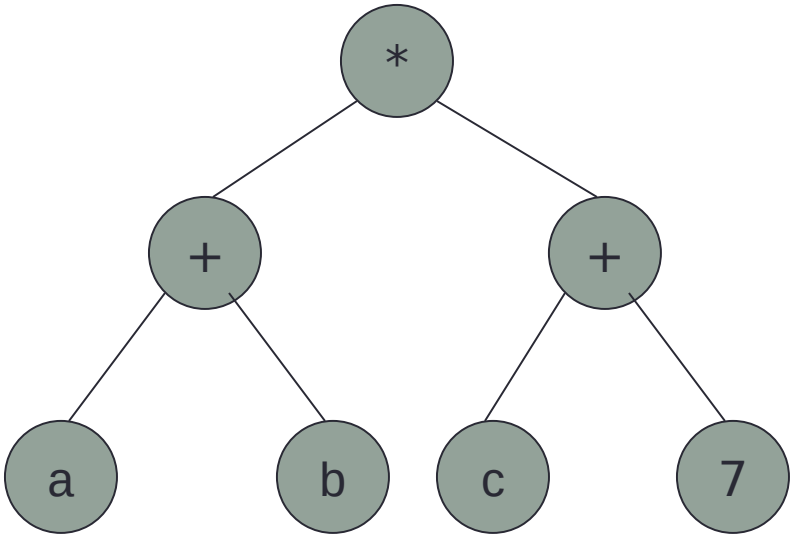
Binary Trees

- A binary tree is a tree with the following:
  1. Each node can have **at most two subtrees** and therefore at most two children.
  2. Each subtree is identified as being either the left subtree or the right subtree of the parent.
  3. It may be empty.
- Nodes in a binary tree may be composite e.g. of variable type ‘Type’.

Binary Trees

7 + [c * (a + b)]





(c + 7) * (a + b)

ADT Binary Tree

Elements: The elements are nodes, each node contains the following data type: Type and has LeftChild and RightChild references.

Structure: hierarchical structure; each node can have two children: left or right child; there is a root node and a current node.

Domain: the number of nodes in a binary tree is bounded; domain contains empty tree, trees with one element, two elements, …

ADT Binary Tree

Operations:

1. **Method** Traverse (Order ord)

   requires: Binary Tree (BT) is not empty.

   input: ord.

   results: Each element in the tree is processed exactly once by a user supplied procedure. The order in which nodes are processed depends on the value of ord (Order = {preorder, postorder, inorder})

   preorder: each node processed **before** any node in either of its subtrees.

   inorder: each node is processed **after** all its nodes in its **left** subtree and **before** any node in its **right** subtree.

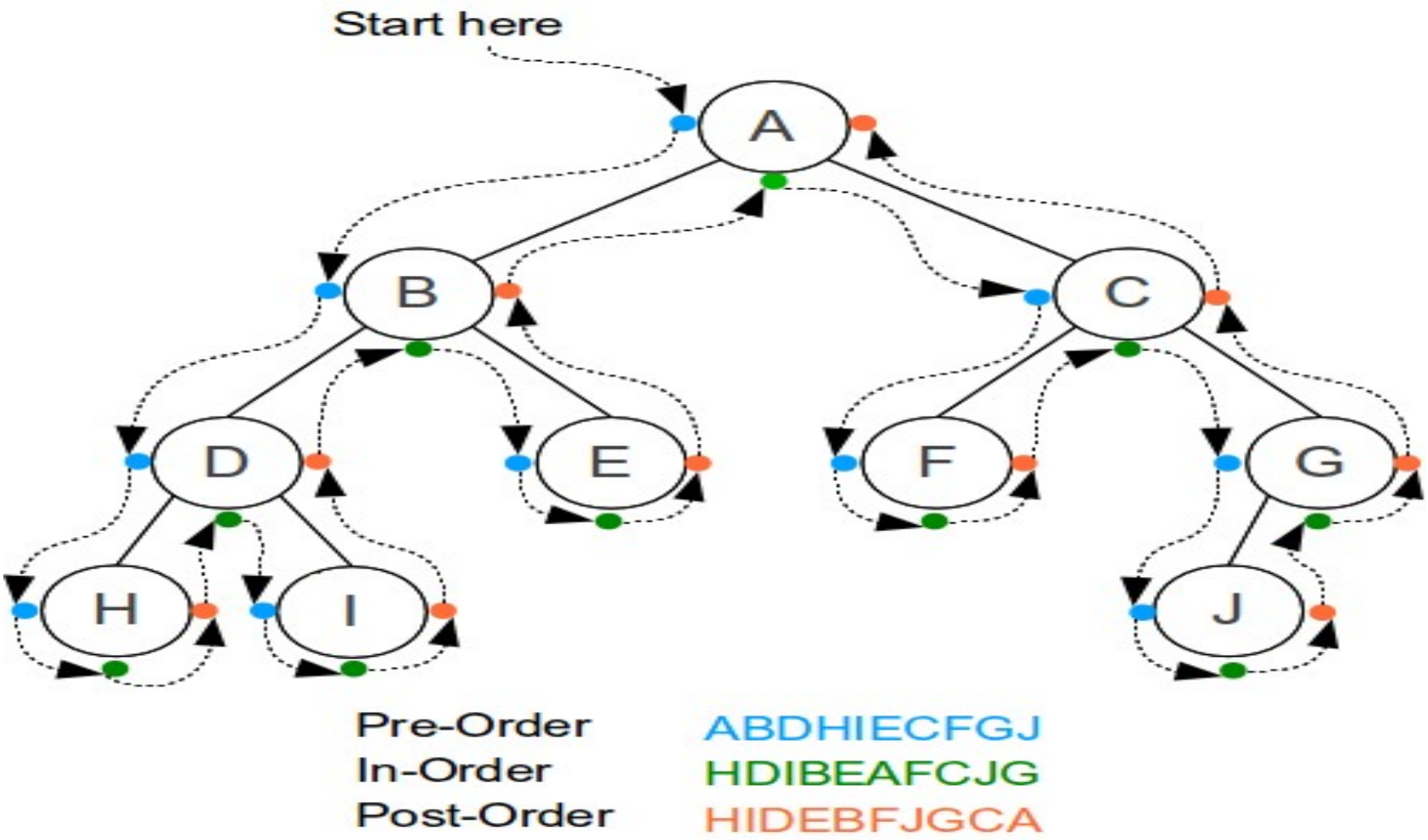
   postorder: each node is processed **after** all its nodes in both of its subtrees.

   output: none.

Tree Traversals

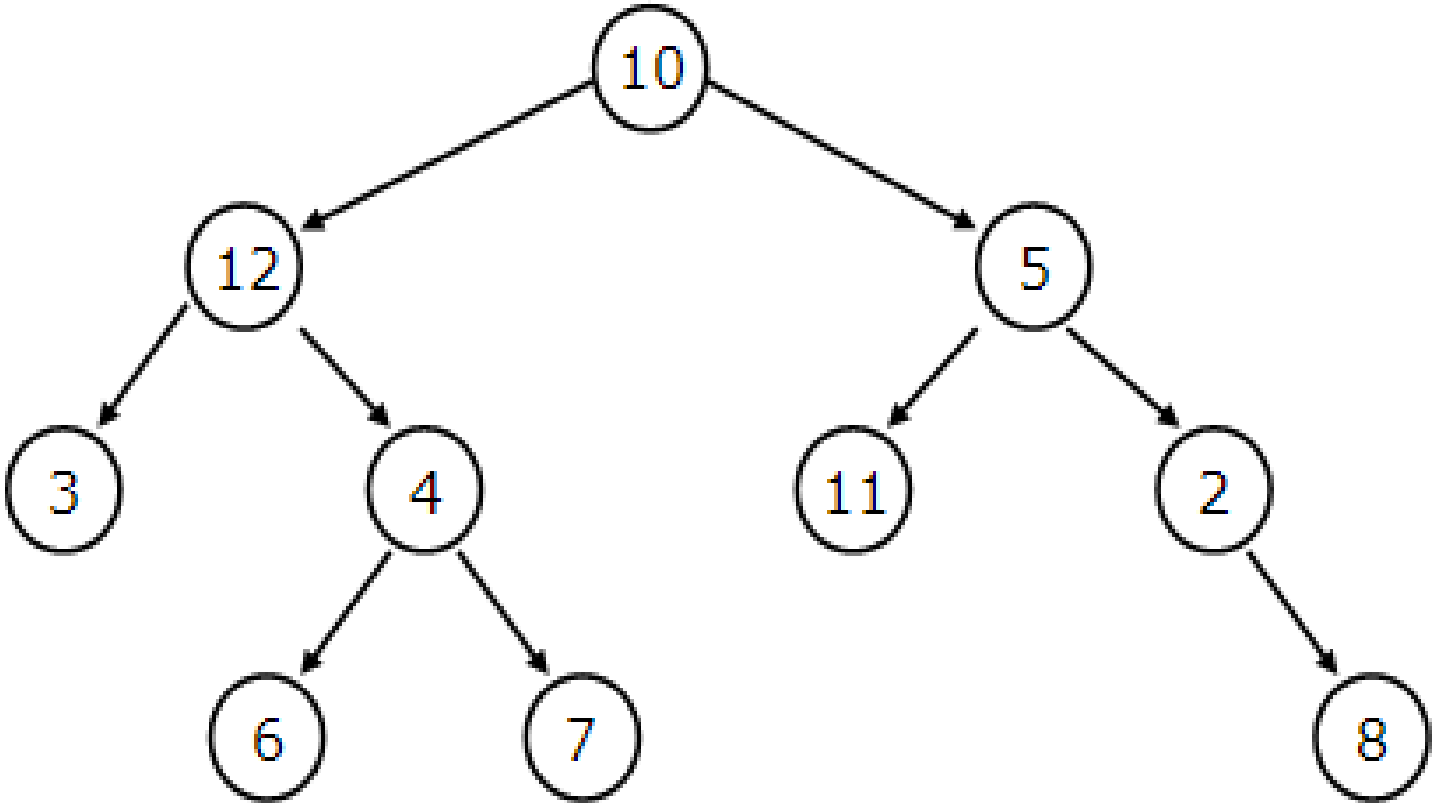
- To traverse a tree means to process (e.g. printing it) each element in the tree.
- **Tree traversals**
  - n! ways of traversing a tree of n nodes.
  - pre-order, in-order, post-order three natural traversals orders.
- **List traversals**
  - n! ways of traversing a list of n nodes.
  - front-to-back, or back-to-front. two natural traversals orders.

Tree Traversals Example



https://en.wikipedia.org/wiki/Tree_traversal

Tree Traversals Example



ADT Binary Tree

Operations:

2. **Method** Insert (Type e, Relative rel, boolean inserted)

   (Relative = {leftchild, rightchild, root, parent})

   requires: (1) Full () is false and (2) either (a) rel = root and Empty() is true or (b) rel <> root and rel <> parent and Empty( ) is false.

   input: e, rel.

   results: **if** case (1) rel = leftChild, current node has a left child, or (2) rel = rightChild, current node has a right child, **then** inserted is false. **Else** a node containing e is added as rel of the current node in the tree and becomes the current node and inserted is true.

   output: inserted.

ADT Binary Tree

3. **Procedure** DeleteSub ()

   requires: Binary tree is not empty.

   input: none

   results: The subtree whose root node was the current node is deleted from the tree. If the resulting tree is not empty, then the root node is the current node.

   output: none.

4. **Procedure** Update (Type e).

   requires: Binary tree is not empty.

   input: e.

   results: the element in e is copied into the current node.

   output: none.

ADT Binary Tree

5. **Procedure** Retrieve (Type e)

   requires: Binary tree is not empty.

   input: none

   results: element in the current node is copied into e.

   output: e.

ADT Binary Tree

6. **Procedure** Find (Relative rel, boolean found)

   requires: Binary tree is not empty.

   input: rel.

   results: The current node of the tree is determined by the value of rel and previous current node..

   output: found.

7. **Procedure** Empty (boolean empty).

   requires: None.

   input: none

   results: If Binary tree is empty then empty is true; otherwise empty is false.

   output: empty.

ADT Binary Tree

8. **Procedure** Full (boolean full)

   requires: None.

   input: None.

   results: if the binary tree is full then full is true otherwise false.

   output: full.

ADT Binary Tree: Element

```
public class BTNode <T> {
  public T data;
  public BTNode<T> left, right;

  /** Creates a new instance of BTNode */
  public BTNode(T val) {
  data = val;
  left = right = null;
  }

  public BTNode(T val, BTNode<T> l, BTNode<T> r){
  data = val;
  left = l;
  right = r;
  }
}
```

ADT Binary Tree: Order & Relative Classes

- These definitions are in separate files and define:
  - The Order class.

```
public enum Order {preOrder, inOrder, postOrder};
```

  - The Relative class.

```
public enum Relative {Root, Parent, LeftChild, RightChild};
```

ADT Binary Tree: Implementation

```
public class BT<T> {
  BTNode<T> root, current;

  /** Creates a new instance of BT */
  public BT() {
      root = current = null;
  }

  public boolean empty(){
      return root == null;
  }
```

ADT Binary Tree: Implementation


```
public class BT<T> {
  BTNode<T> root, current;

  /** Creates a new instance of BT */
  public BT() {
      root = current = null;
  }

  public boolean empty(){
      return root == null;
  }
```

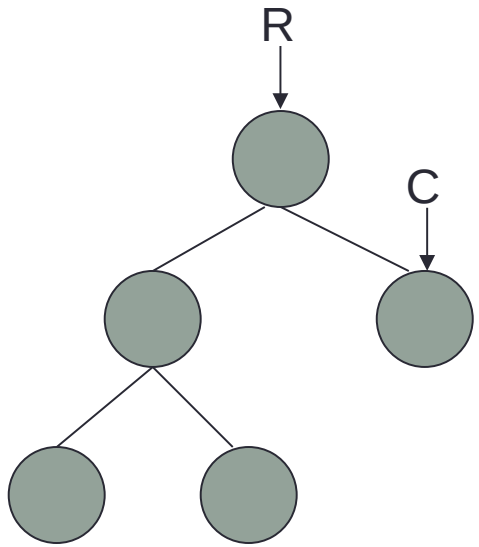
ADT Binary Tree: Implementation

```
public class BT<T> {
  BTNode<T> root, current;

  /** Creates a new instance of BT */
  public BT() {
      root = current = null;
  }

  public boolean empty(){
      return root == null;
  }
```

true



false

ADT Binary Tree: Implementation

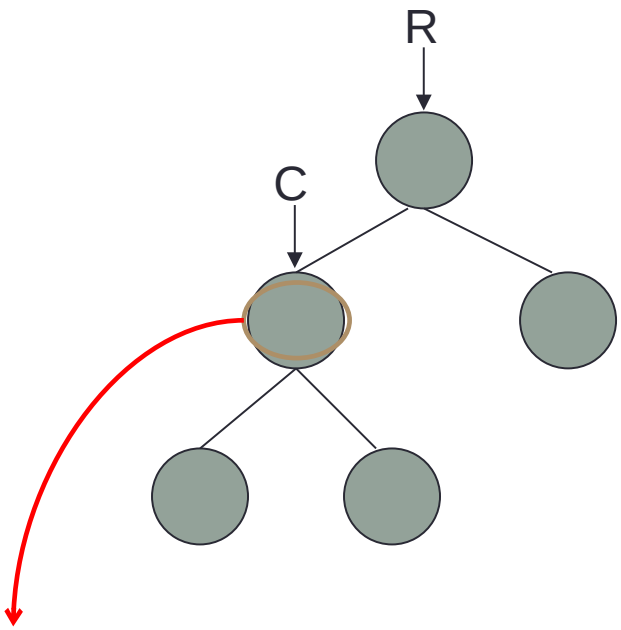
```
public T retrieve() {
    return current.data;
}

public void update(T val) {
    current.data = val;
}
```

ADT Binary Tree: Implementation

```
public T retrieve() {
    return current.data;
}
```

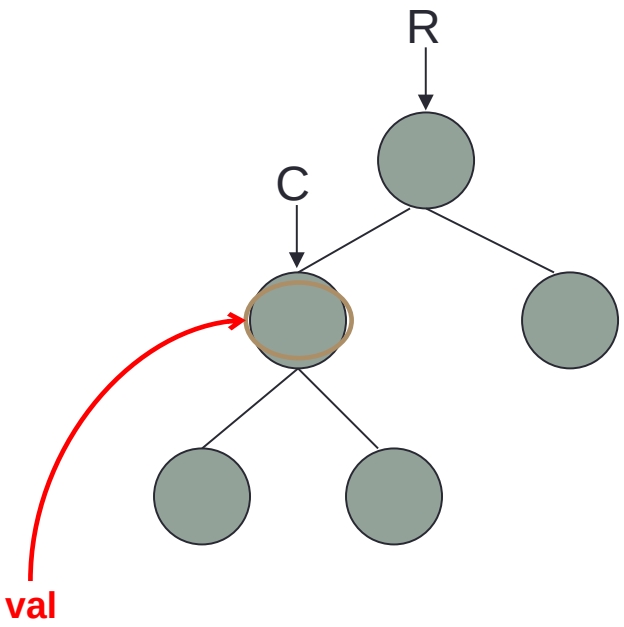
```
public void update(T val) {
    current.data = val;
}
```



ADT Binary Tree: Implementation

```
public T retrieve() {
      return current.data;
}
```

```
public void update(T val) {
      current.data = val;
}
```



ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #1

rel = Root

ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #1

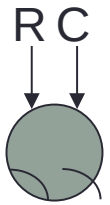
rel = Root

ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #1

rel = Root

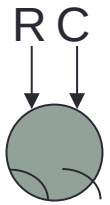


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #2

rel = LeftChild

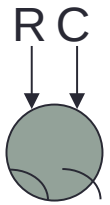


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #2

rel = LeftChild

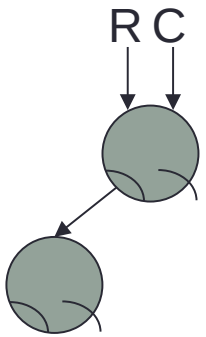


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #2

rel = LeftChild

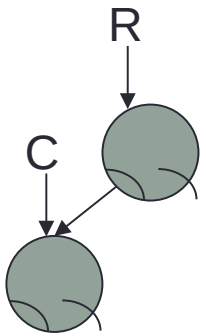


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #2

rel = LeftChild

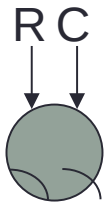


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #3

rel = RightChild

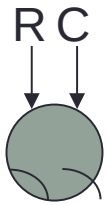


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #3

rel = RightChild

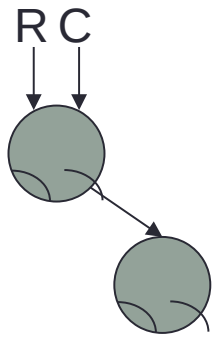


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #3

rel = RightChild

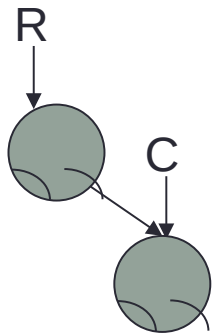


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #3

rel = RightChild

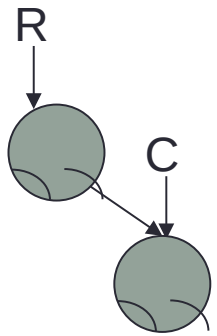


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #4

rel = Parent

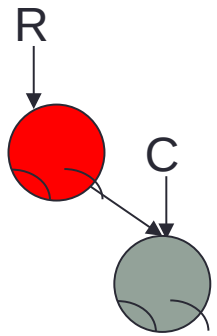


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent: //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #4

rel = Parent

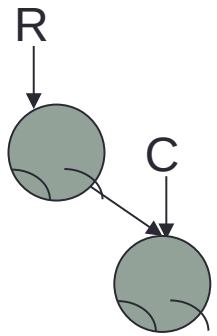


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #5

rel = LeftChild

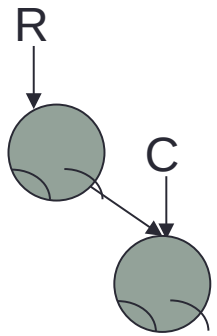


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #5

rel = LeftChild

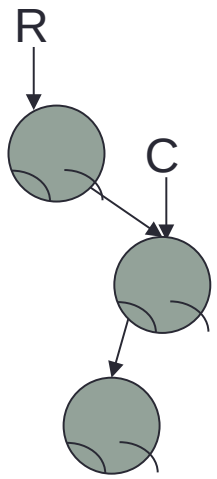


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #5

rel = LeftChild

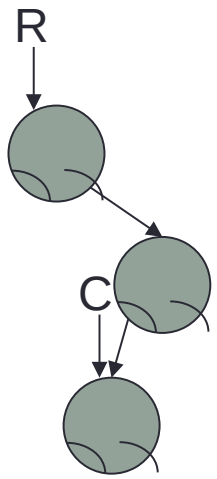


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #5

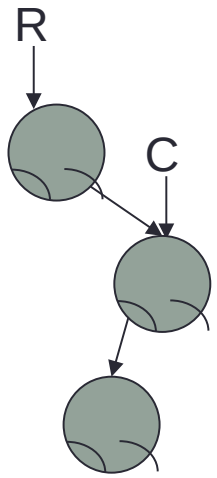
rel = LeftChild



ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Find Parent

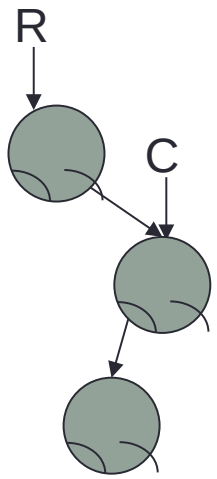


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #6

rel = RightChild

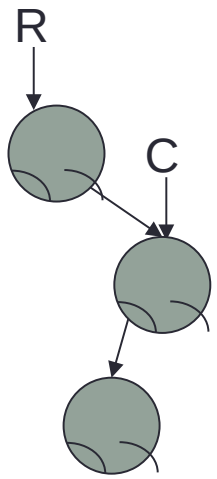


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #6

rel = RightChild

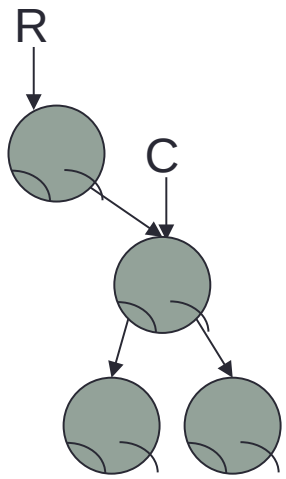


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #6

rel = RightChild

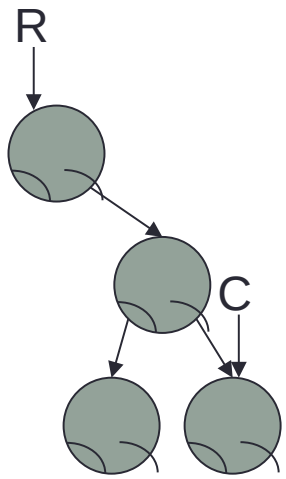


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #6

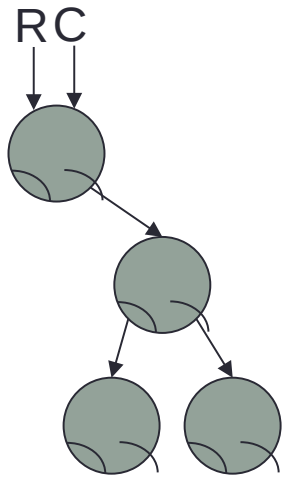
rel = RightChild



ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Find Root

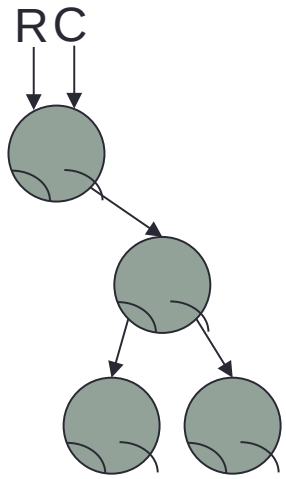


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #7

rel = LeftChild

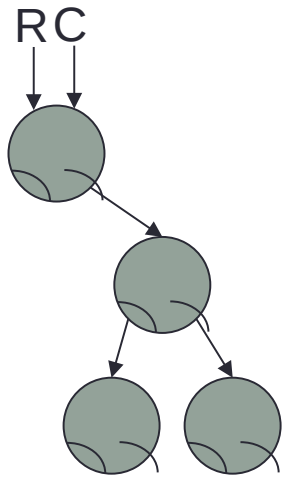


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #7

rel = LeftChild

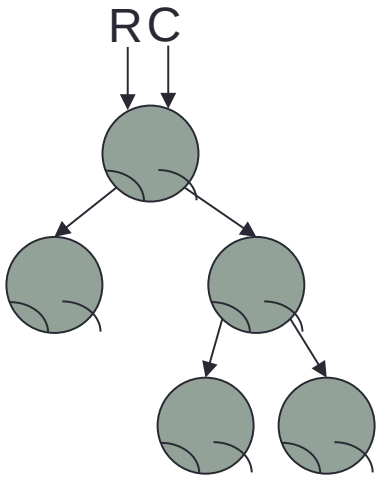


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #7

rel = LeftChild

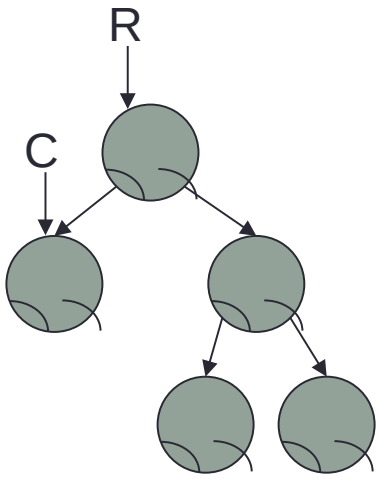


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #7

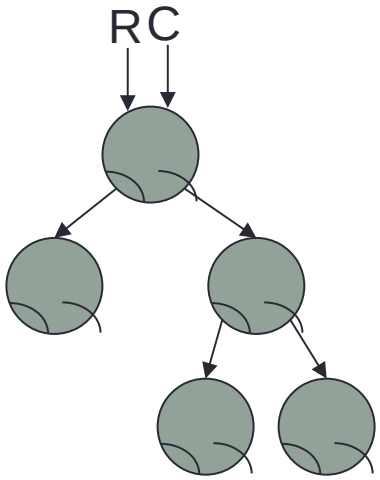
rel = LeftChild



ADT Binary Tree: Implementation

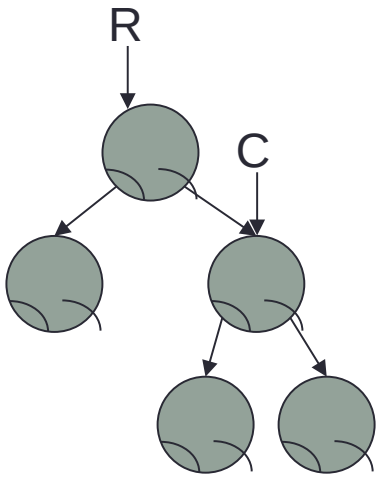
```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Find Root



ADT Binary Tree: Implementation

Find RightChild



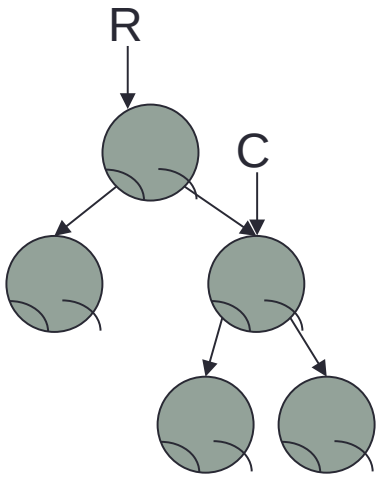

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #8

rel = LeftChild

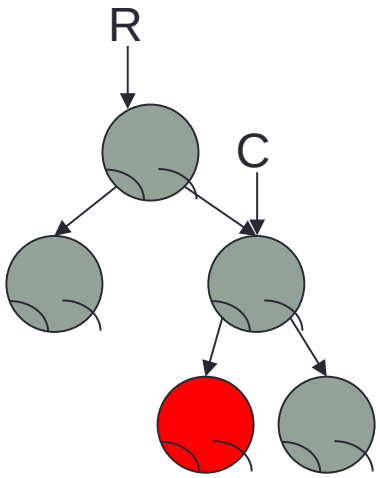


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #8

rel = LeftChild

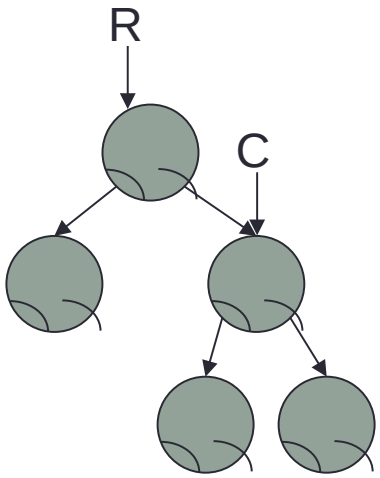


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #9

rel = RightChild

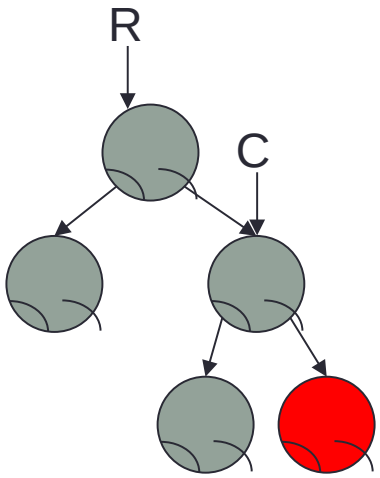


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #9

rel = RightChild

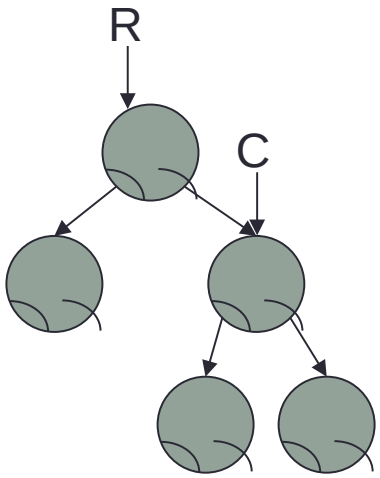


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent:  //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #10

rel = Parent

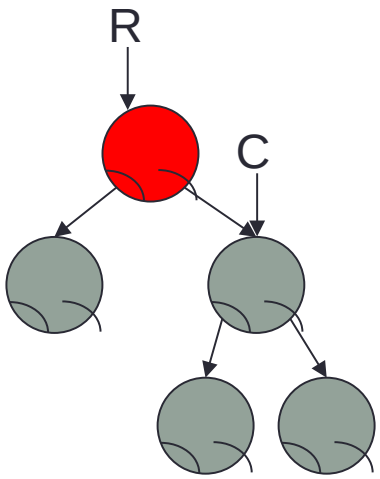


ADT Binary Tree: Implementation

```
public boolean insert(Relative rel, T val) {
      switch(rel) {
        case Root:
      if(!empty()) return false;
      current = root = new BTNode<T>(val);
      return true;
        case Parent: //This is an impossible case.
      return false;
        case LeftChild:
      if(current.left != null) return false;
      current.left = new BTNode<T>(val);
      current = current.left;
      return true;
        case RightChild:
      if(current.right != null) return false;
      current.right = new BTNode<T> (val);
      current = current.right;
      return true;
        default:
      return false;
      }
}
```

Example #10

rel = Parent



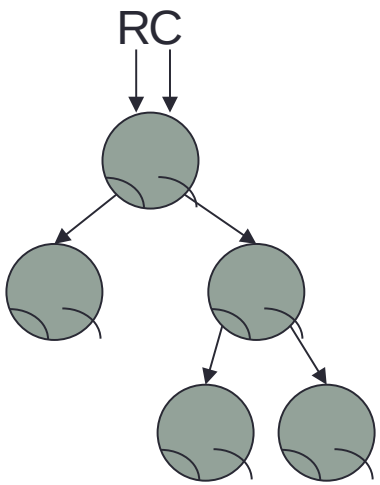
ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

ADT Binary Tree: Implementation

```
public void deleteSubtree(){
        if(current == root){
                current = root = null;
        }
        else {
                BTNode<T> p = current;
                find(Relative.Parent);
                if(current.left == p)
                        current.left = null;
                else
                        current.right = null;
                current = root;
        }
}
```

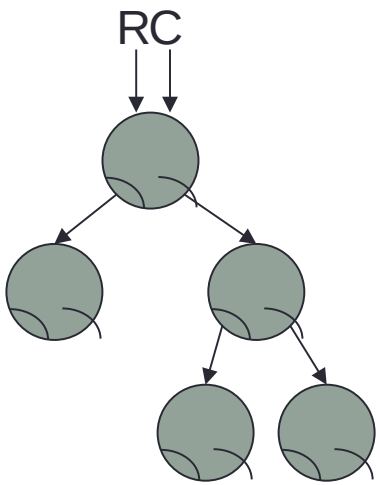
Example #1



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
        if(current == root){
                current = root = null;
        }
        else {
                BTNode<T> p = current;
                find(Relative.Parent);
                if(current.left == p)
                        current.left = null;
                else
                        current.right = null;
                current = root;
        }
}
```

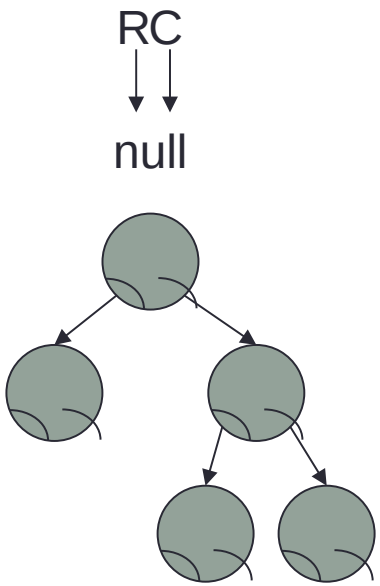
Example #1



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

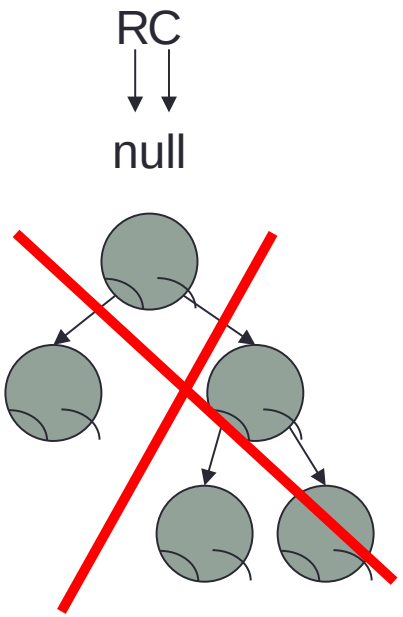
Example #1



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
        if(current == root){
                current = root = null;
        }
        else {
                BTNode<T> p = current;
                find(Relative.Parent);
                if(current.left == p)
                        current.left = null;
                else
                        current.right = null;
                current = root;
        }
}
```

Example #1



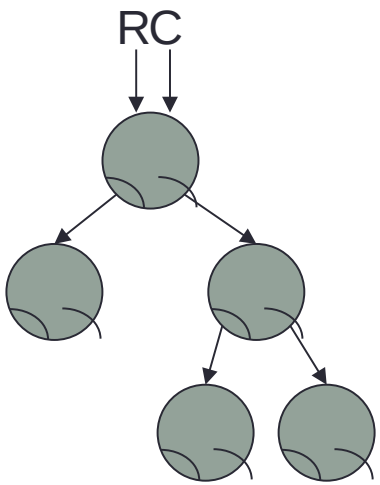
ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

Example #1

ADT Binary Tree: Implementation

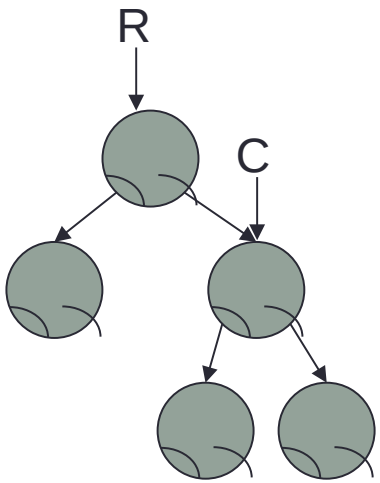
```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
        if(current == root){
                current = root = null;
        }
        else {
                BTNode<T> p = current;
                find(Relative.Parent);
                if(current.left == p)
                        current.left = null;
                else
                        current.right = null;
                current = root;
        }
}
```

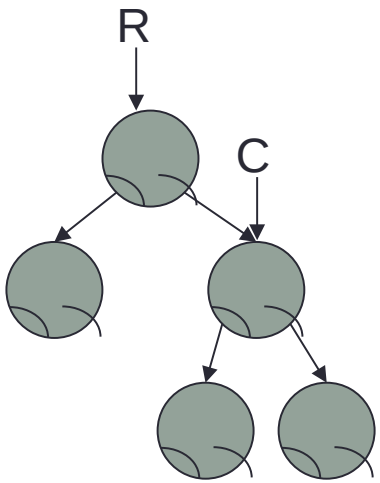
Find RightChild



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

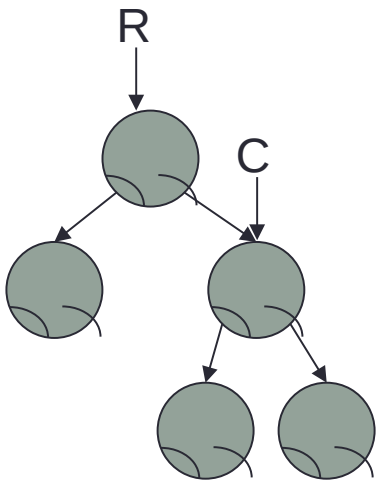
Example #2



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
        if(current == root){
                current = root = null;
        }
        else {
                BTNode<T> p = current;
                find(Relative.Parent);
                if(current.left == p)
                        current.left = null;
                else
                        current.right = null;
                current = root;
        }
}
```

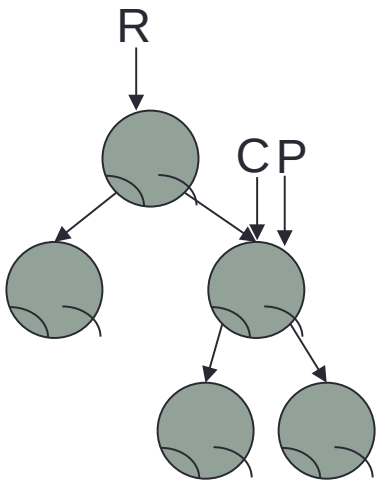
Example #2



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

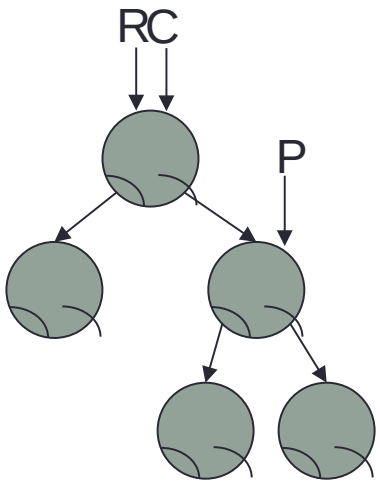
Example #2



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

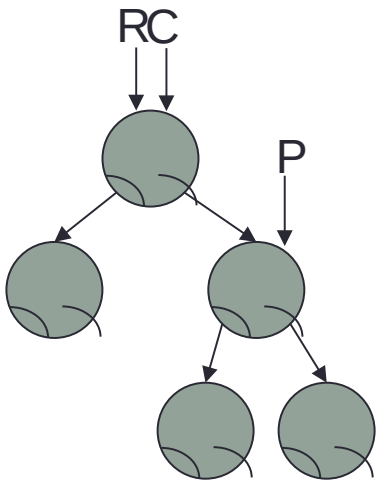
Example #2



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

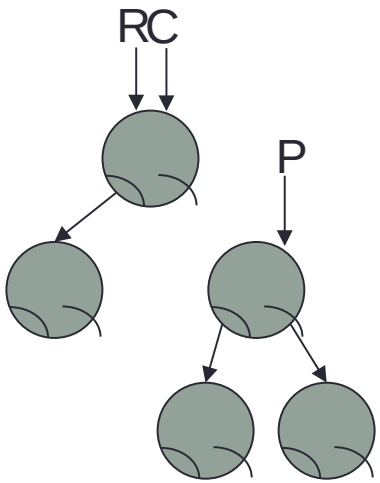
Example #2



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

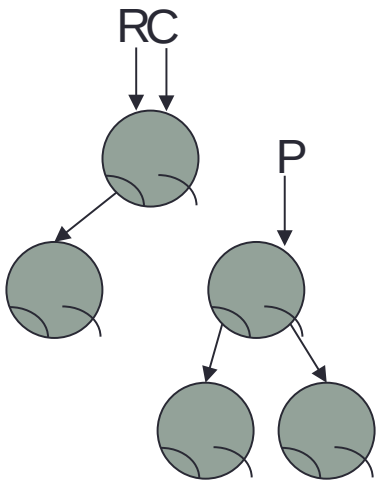
Example #2



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
        if(current == root){
                current = root = null;
        }
        else {
                BTNode<T> p = current;
                find(Relative.Parent);
                if(current.left == p)
                        current.left = null;
                else
                        current.right = null;
                current = root;
        }
}
```

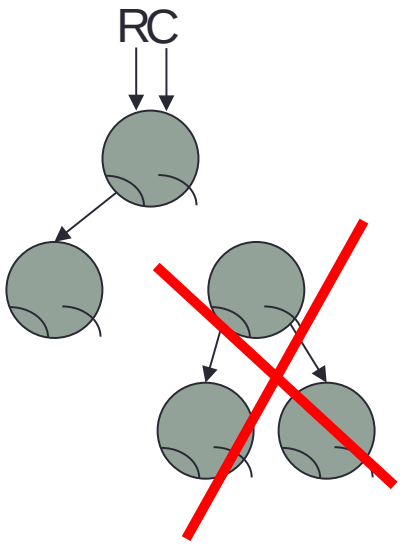
Example #2



ADT Binary Tree: Implementation

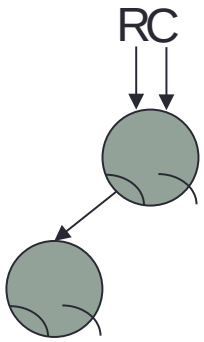
```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

Example #2



ADT Binary Tree: Implementation

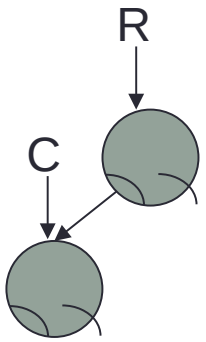
```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

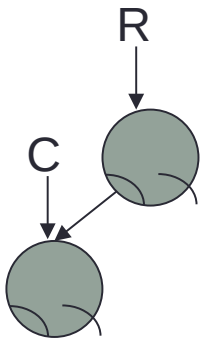
Find LeftChild



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
        if(current == root){
                current = root = null;
        }
        else {
                BTNode<T> p = current;
                find(Relative.Parent);
                if(current.left == p)
                        current.left = null;
                else
                        current.right = null;
                current = root;
        }
}
```

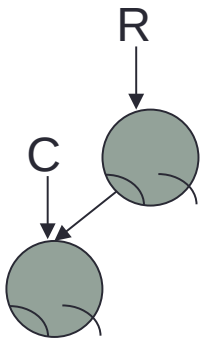
Example #3



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
        if(current == root){
                current = root = null;
        }
        else {
                BTNode<T> p = current;
                find(Relative.Parent);
                if(current.left == p)
                        current.left = null;
                else
                        current.right = null;
                current = root;
        }
}
```

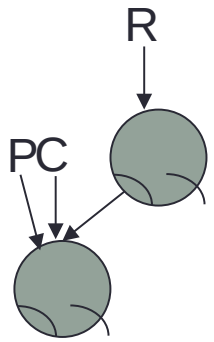
Example #3



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

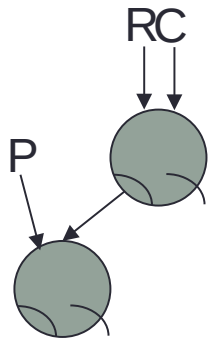
Example #3



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

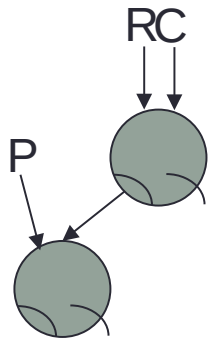
Example #3



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

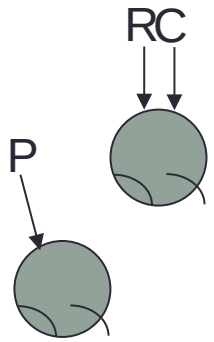
Example #3



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

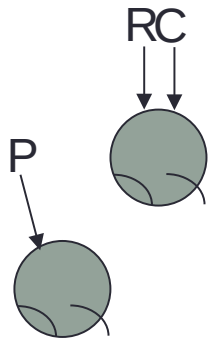
Example #3



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

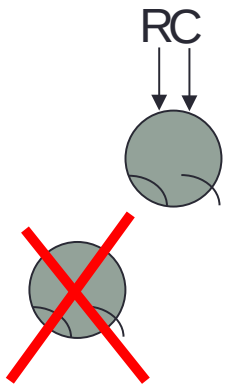
Example #3



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

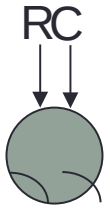
Example #3



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

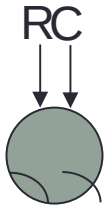
Example #4



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

Example #4



ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

Example #4

ADT Binary Tree: Implementation

```
public void deleteSubtree(){
    if(current == root){
        current = root = null;
    }
    else {
        BTNode<T> p = current;
        find(Relative.Parent);
        if(current.left == p)
            current.left = null;
        else
            current.right = null;
        current = root;
    }
}
```

Example #4

ADT Binary Tree: Implementation

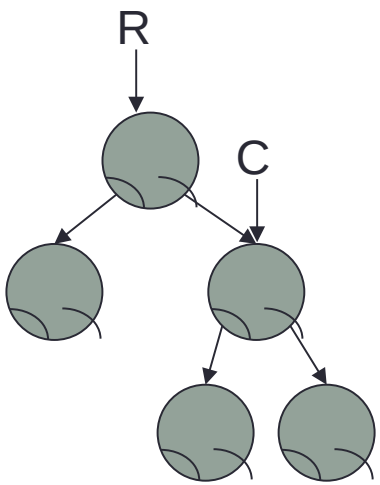
```
  public boolean find(Relative rel){
        switch (rel) {
          case Root:     // Easy case
        current = root;
        return true;
          case Parent:
        if(current == root) return false;
        current = findparent(current, root);
        return true;
          case LeftChild:
        if(current.left == null) return false;
        current = current.left;
        return true;
          case RightChild:
        if(current.right == null) return false;
        current = current.right;
        return true;
          default:
        return false;
        }
  }
}
```

ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #1

rel = Root

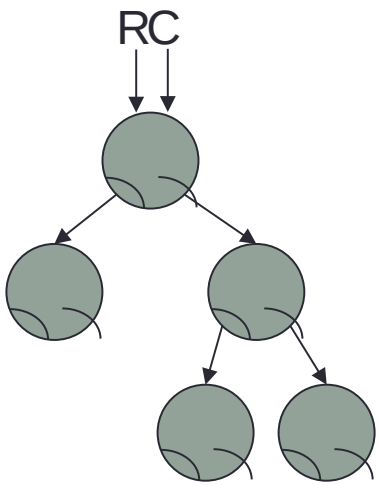


ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:    // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #1

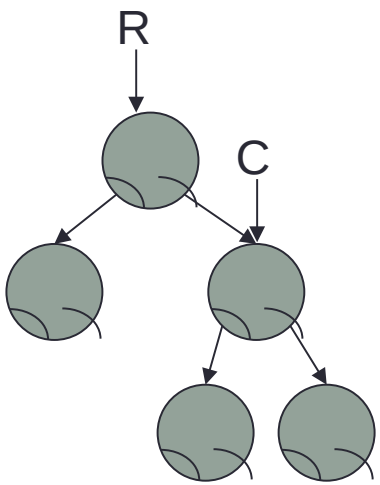
rel = Root



ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

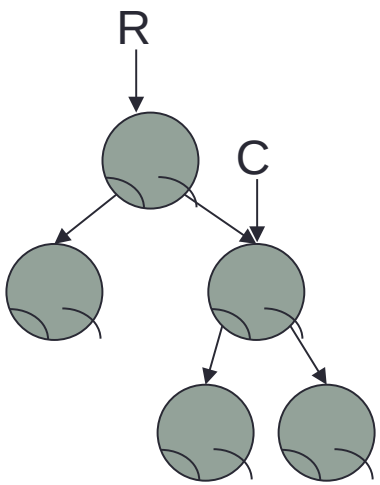
Example #2
rel = LeftChild



ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

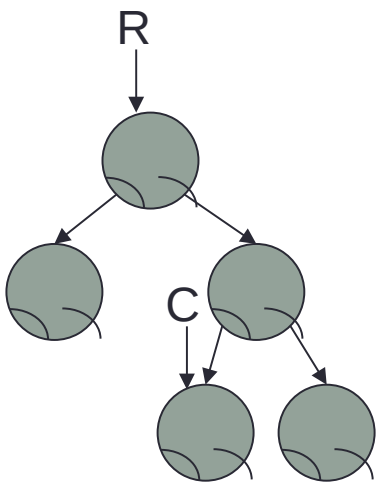
Example #2
rel = LeftChild



ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #2
rel = LeftChild

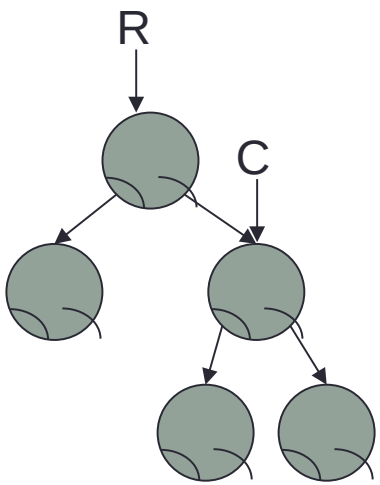


ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #3

rel = RightChild

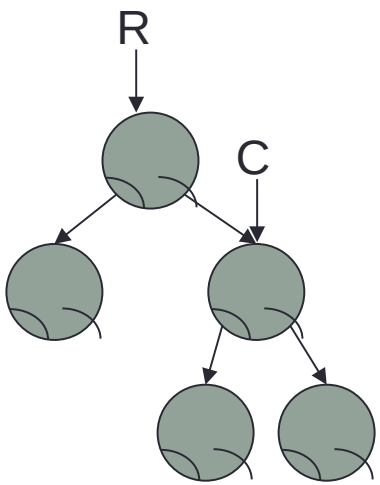


ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #3

rel = RightChild

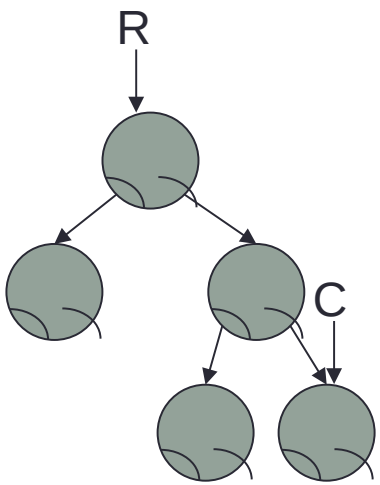


ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #3

rel = RightChild

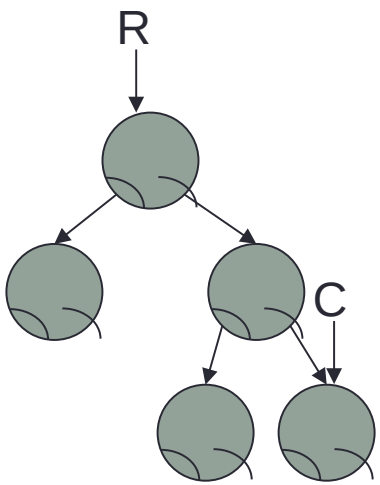


ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #4

rel = RightChild

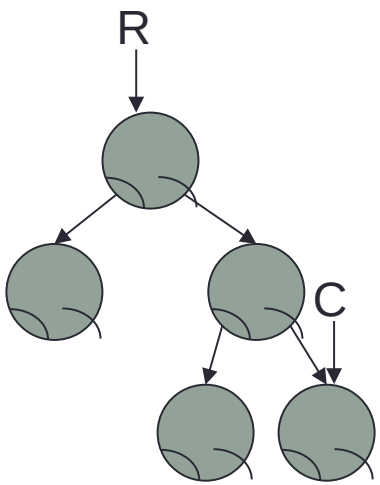


ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #4

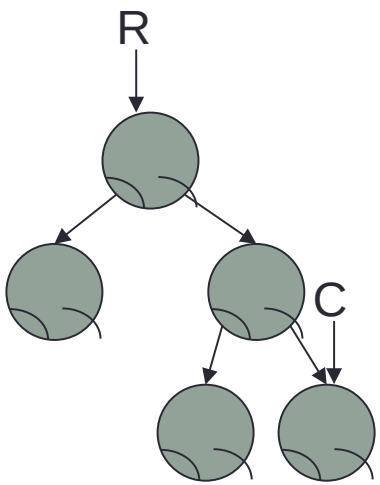
rel = RightChild



ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

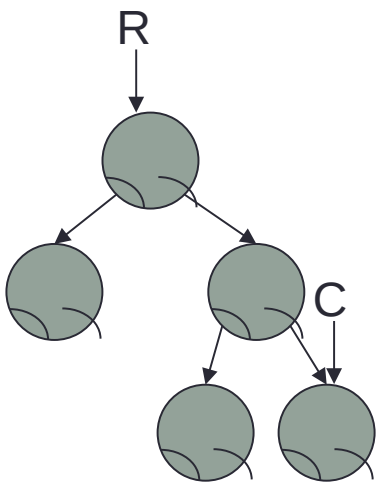
Example #5
rel = LeftChild



ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #5
rel = LeftChild

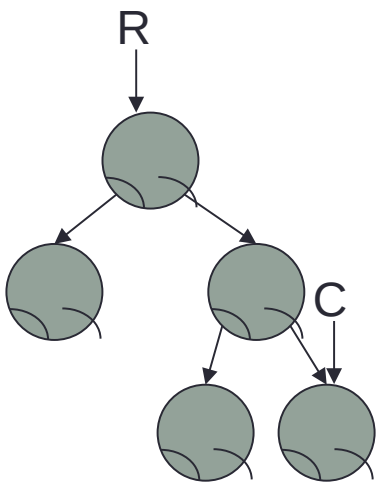


ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #6

rel = Parent

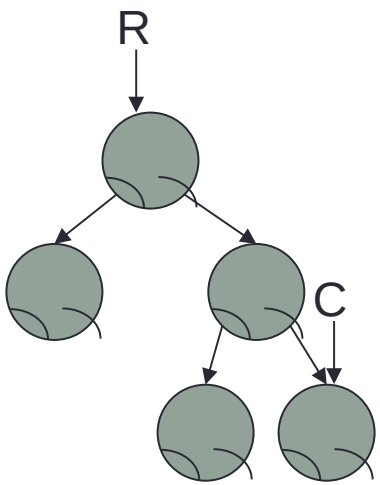


ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #6

rel = Parent

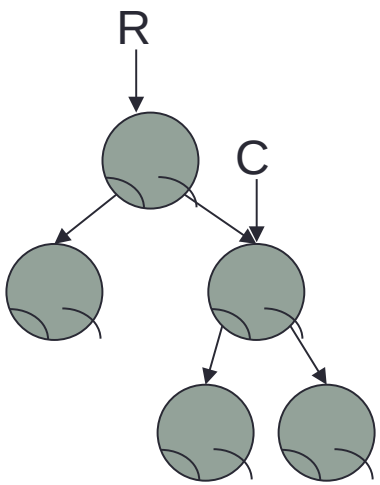


ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #6

rel = Parent

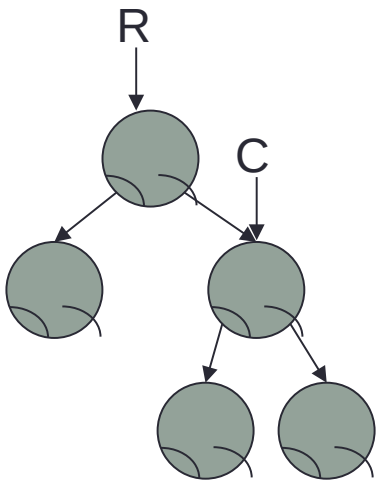


ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #7

rel = Parent

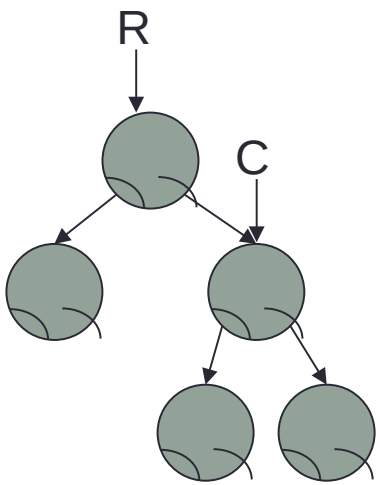


ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #7

rel = Parent

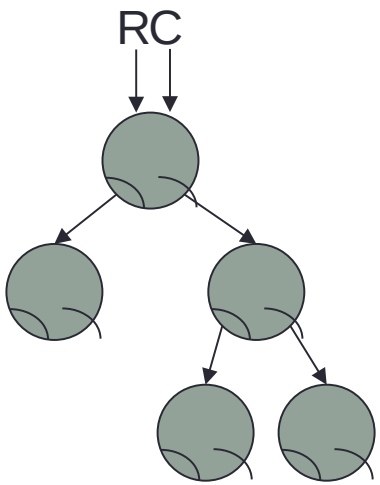


ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #7

rel = Parent

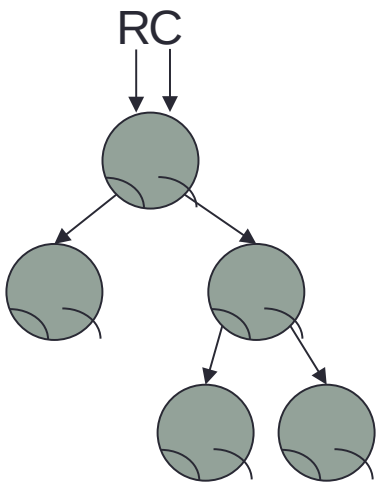


ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #8

rel = Parent

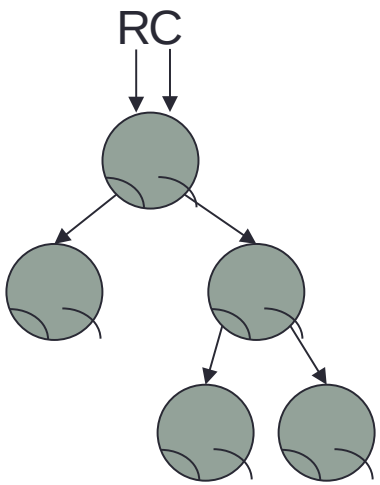


ADT Binary Tree: Implementation

```
public boolean find(Relative rel){
      switch (rel) {
        case Root:     // Easy case
      current = root;
      return true;
        case Parent:
      if(current == root) return false;
      current = findparent(current, root);
      return true;
        case LeftChild:
      if(current.left == null) return false;
      current = current.left;
      return true;
        case RightChild:
      if(current.right == null) return false;
      current = current.right;
      return true;
        default:
      return false;
      }
  }
}
```

Example #8

rel = Parent



ADT Binary Tree: Implementation

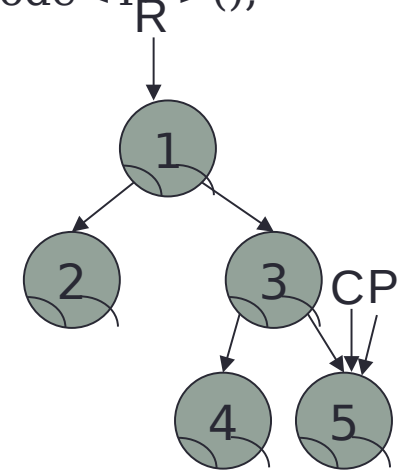
```
// Non-recursive version of findparent – uses pre-order traversal
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        // Stack is used to store the right pointers of nodes
        LinkStack<BTNode<T>> stack = new LinkStack<BTNode<T>>();
        BTNode<T> q = t;
        while(q.right != p && q.left != p) {
                if(q.right != null)
                        stack.push(q.right);

                if(q.left != null)
                        q = q.left;
                else
                        q = stack.pop(); // Go right here
        }
        return q;
}
```

ADT Binary Tree: Implementation

```
// Non-recursive version of findparent – uses pre-order traversal
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
      // Stack is used to store the right pointers of nodes
      LinkStack<BTNode<T>> stack = new LinkStack<BTNode<T>>();
      BTNode<T> q = t;
      while(q.right != p && q.left != p) {
            if(q.right != null)
                  stack.push(q.right);

            if(q.left != null)
                  q = q.left;
            else
                  q = stack.pop(); // Go right here
      }
      return q;
}
```



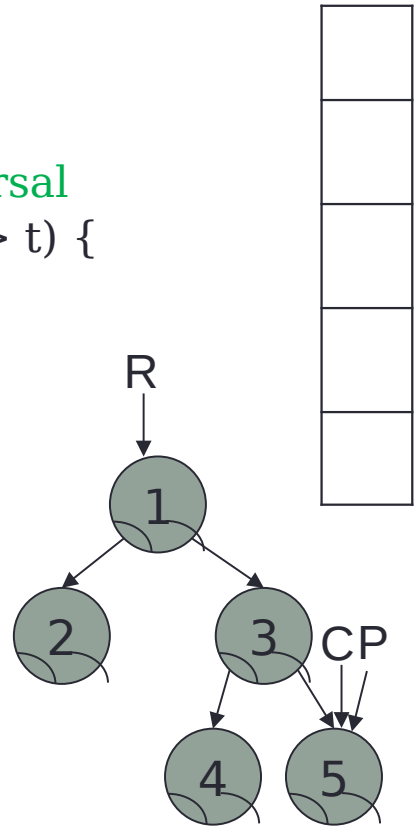
Example #1

p = current, t = root

ADT Binary Tree: Implementation

```
// Non-recursive version of findparent – uses pre-order traversal
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
      // Stack is used to store the right pointers of nodes
      LinkStack<BTNode<T>> stack = new
LinkStack<BTNode<T>>();
      BTNode<T> q = t;
      while(q.right != p && q.left != p) {
            if(q.right != null)
                  stack.push(q.right);

            if(q.left != null)
                  q = q.left;
            else
                  q = stack.pop(); // Go right here
      }
      return q;
}
```



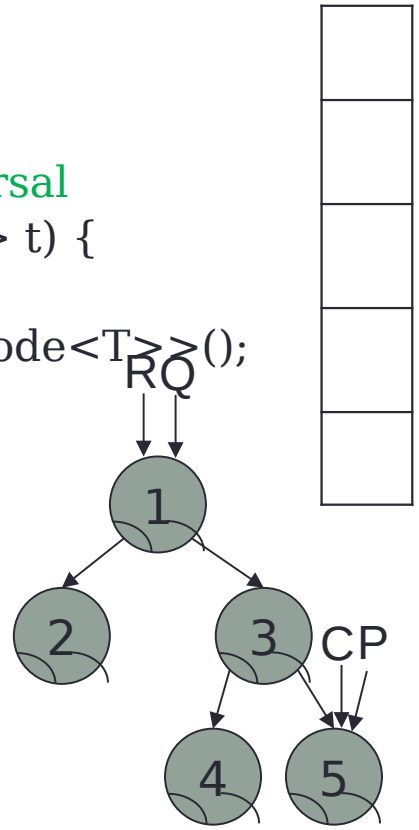
Example #1

p = current, t = root

ADT Binary Tree: Implementation

```
// Non-recursive version of findparent – uses pre-order traversal
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
      // Stack is used to store the right pointers of nodes
      LinkStack<BTNode<T>> stack = new LinkStack<BTNode<T>>();
      BTNode<T> q = t;
      while(q.right != p && q.left != p) {
            if(q.right != null)
                  stack.push(q.right);

            if(q.left != null)
                  q = q.left;
            else
                  q = stack.pop(); // Go right here
      }
      return q;
}
```



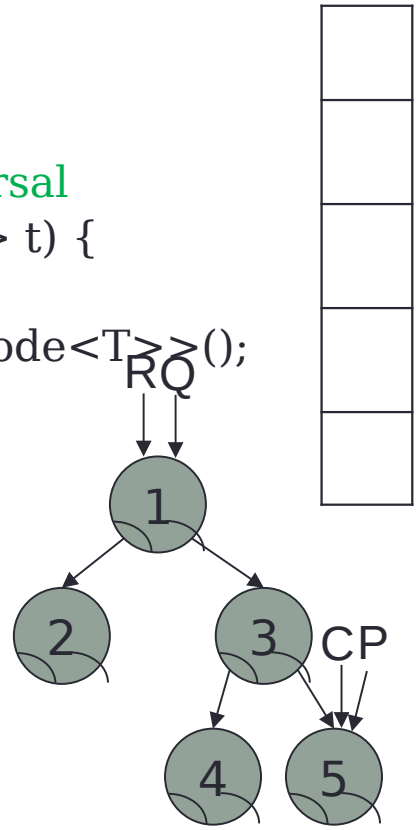
Example #1

p = current, t = root

ADT Binary Tree: Implementation

```
// Non-recursive version of findparent – uses pre-order traversal
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
      // Stack is used to store the right pointers of nodes
      LinkStack<BTNode<T>> stack = new LinkStack<BTNode<T>>();
      BTNode<T> q = t;
      while(q.right != p && q.left != p) {
            if(q.right != null)
                  stack.push(q.right);

            if(q.left != null)
                  q = q.left;
            else
                  q = stack.pop(); // Go right here
      }
      return q;
}
```



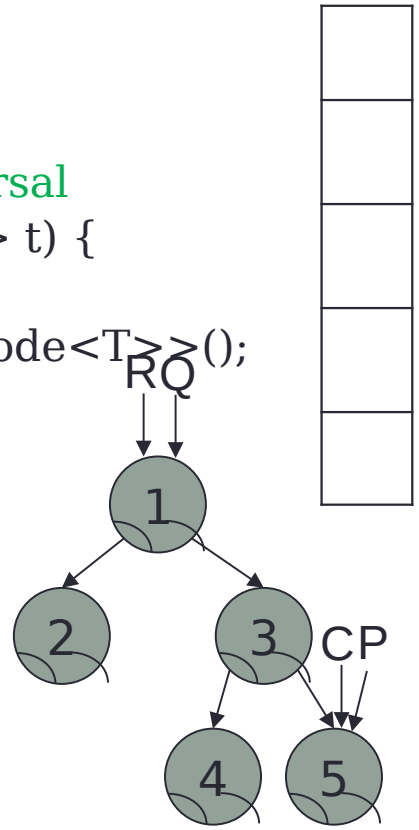
Example #1

p = current, t = root

ADT Binary Tree: Implementation

```
// Non-recursive version of findparent – uses pre-order traversal
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
      // Stack is used to store the right pointers of nodes
      LinkStack<BTNode<T>> stack = new LinkStack<BTNode<T>>();
      BTNode<T> q = t;
      while(q.right != p && q.left != p) {
            if(q.right != null)
                  stack.push(q.right);

            if(q.left != null)
                  q = q.left;
            else
                  q = stack.pop(); // Go right here
      }
      return q;
}
```



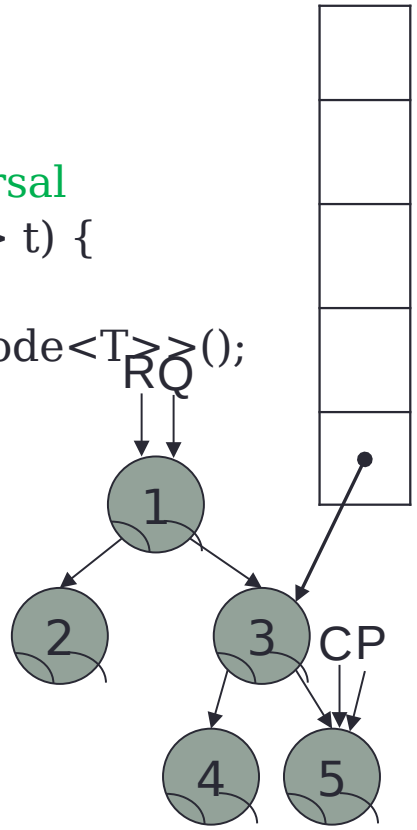
Example #1

p = current, t = root

ADT Binary Tree: Implementation

```
// Non-recursive version of findparent – uses pre-order traversal
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
      // Stack is used to store the right pointers of nodes
      LinkStack<BTNode<T>> stack = new LinkStack<BTNode<T>>();
      BTNode<T> q = t;
      while(q.right != p && q.left != p) {
            if(q.right != null)
                  stack.push(q.right);

            if(q.left != null)
                  q = q.left;
            else
                  q = stack.pop(); // Go right here
      }
      return q;
}
```



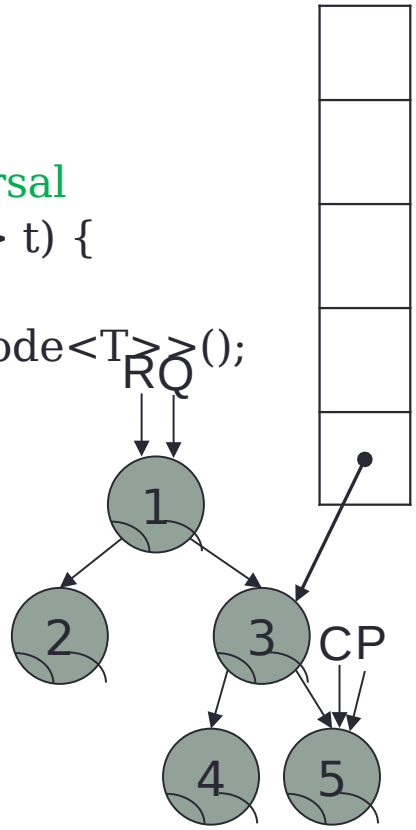
Example #1

p = current, t = root

ADT Binary Tree: Implementation

```
// Non-recursive version of findparent – uses pre-order traversal
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
      // Stack is used to store the right pointers of nodes
      LinkStack<BTNode<T>> stack = new LinkStack<BTNode<T>>();
      BTNode<T> q = t;
      while(q.right != p && q.left != p) {
            if(q.right != null)
                  stack.push(q.right);

            if(q.left != null)
                  q = q.left;
            else
                  q = stack.pop(); // Go right here
      }
      return q;
}
```



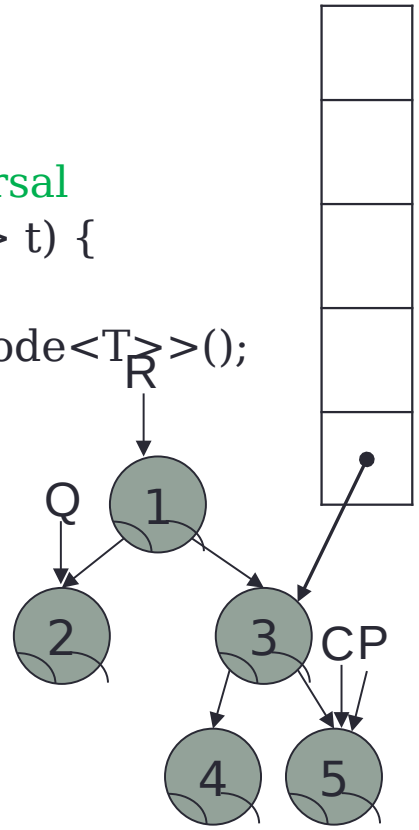
Example #1

p = current, t = root

ADT Binary Tree: Implementation

```
// Non-recursive version of findparent – uses pre-order traversal
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
      // Stack is used to store the right pointers of nodes
      LinkStack<BTNode<T>> stack = new LinkStack<BTNode<T>>();
      BTNode<T> q = t;
      while(q.right != p && q.left != p) {
            if(q.right != null)
                  stack.push(q.right);

            if(q.left != null)
                  q = q.left;
            else
                  q = stack.pop(); // Go right here
      }
      return q;
}
```



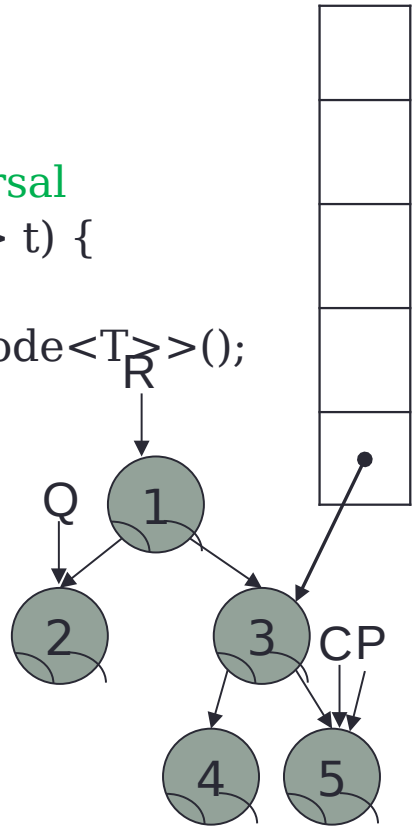
Example #1

p = current, t = root

ADT Binary Tree: Implementation

```
// Non-recursive version of findparent – uses pre-order traversal
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
      // Stack is used to store the right pointers of nodes
      LinkStack<BTNode<T>> stack = new LinkStack<BTNode<T>>();
      BTNode<T> q = t;
      while(q.right != p && q.left != p) {
            if(q.right != null)
                  stack.push(q.right);

            if(q.left != null)
                  q = q.left;
            else
                  q = stack.pop(); // Go right here
      }
      return q;
}
```



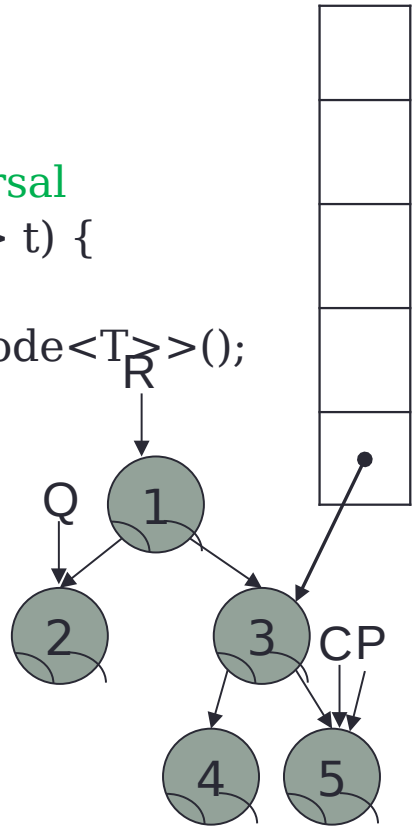
Example #1

p = current, t = root

ADT Binary Tree: Implementation

```
// Non-recursive version of findparent – uses pre-order traversal
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
      // Stack is used to store the right pointers of nodes
      LinkStack<BTNode<T>> stack = new LinkStack<BTNode<T>>();
      BTNode<T> q = t;
      while(q.right != p && q.left != p) {
            if(q.right != null)
                  stack.push(q.right);

            if(q.left != null)
                  q = q.left;
            else
                  q = stack.pop(); // Go right here
      }
      return q;
}
```



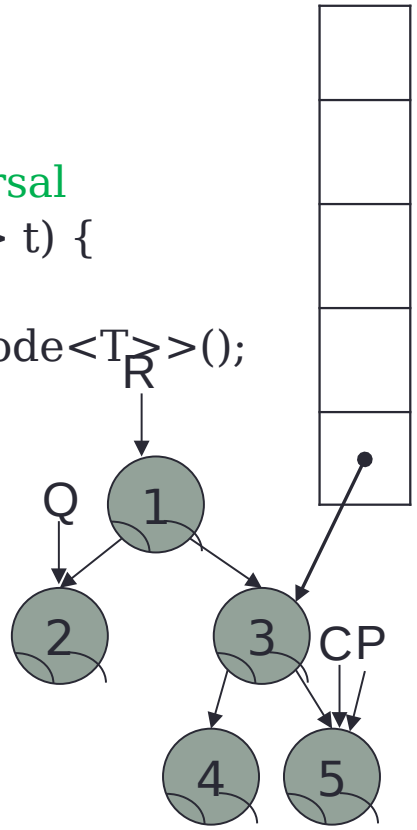
Example #1

p = current, t = root

ADT Binary Tree: Implementation

```
// Non-recursive version of findparent – uses pre-order traversal
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
      // Stack is used to store the right pointers of nodes
      LinkStack<BTNode<T>> stack = new LinkStack<BTNode<T>>();
      BTNode<T> q = t;
      while(q.right != p && q.left != p) {
            if(q.right != null)
                  stack.push(q.right);

            if(q.left != null)
                  q = q.left;
            else
                  q = stack.pop(); // Go right here
      }
      return q;
}
```



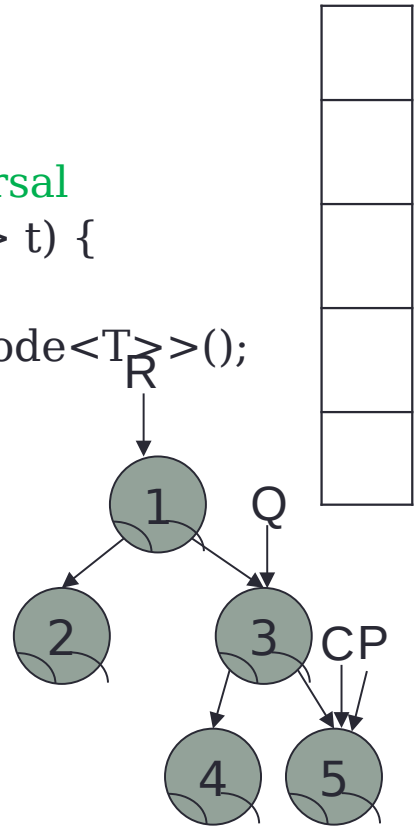
Example #1

p = current, t = root

ADT Binary Tree: Implementation

```
// Non-recursive version of findparent – uses pre-order traversal
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
      // Stack is used to store the right pointers of nodes
      LinkStack<BTNode<T>> stack = new LinkStack<BTNode<T>>();
      BTNode<T> q = t;
      while(q.right != p && q.left != p) {
            if(q.right != null)
                  stack.push(q.right);

            if(q.left != null)
                  q = q.left;
            else
                  q = stack.pop(); // Go right here
      }
      return q;
}
```



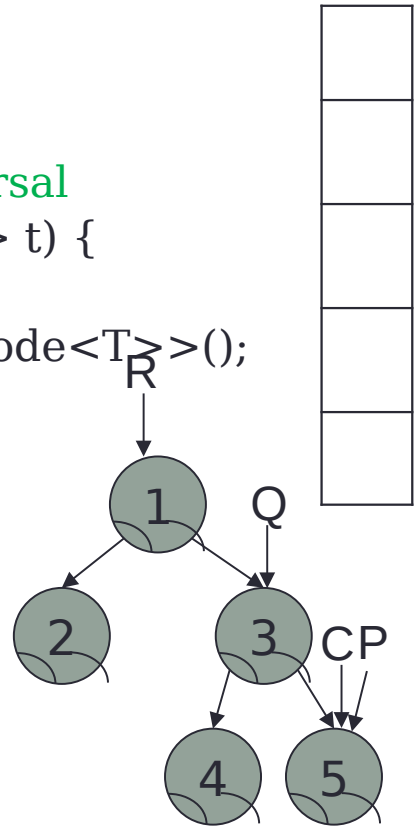
Example #1

p = current, t = root

ADT Binary Tree: Implementation

```
// Non-recursive version of findparent – uses pre-order traversal
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
      // Stack is used to store the right pointers of nodes
      LinkStack<BTNode<T>> stack = new LinkStack<BTNode<T>>();
      BTNode<T> q = t;
      while(q.right != p && q.left != p) {
            if(q.right != null)
                  stack.push(q.right);

            if(q.left != null)
                  q = q.left;
            else
                  q = stack.pop(); // Go right here
      }
      return q;
}
```



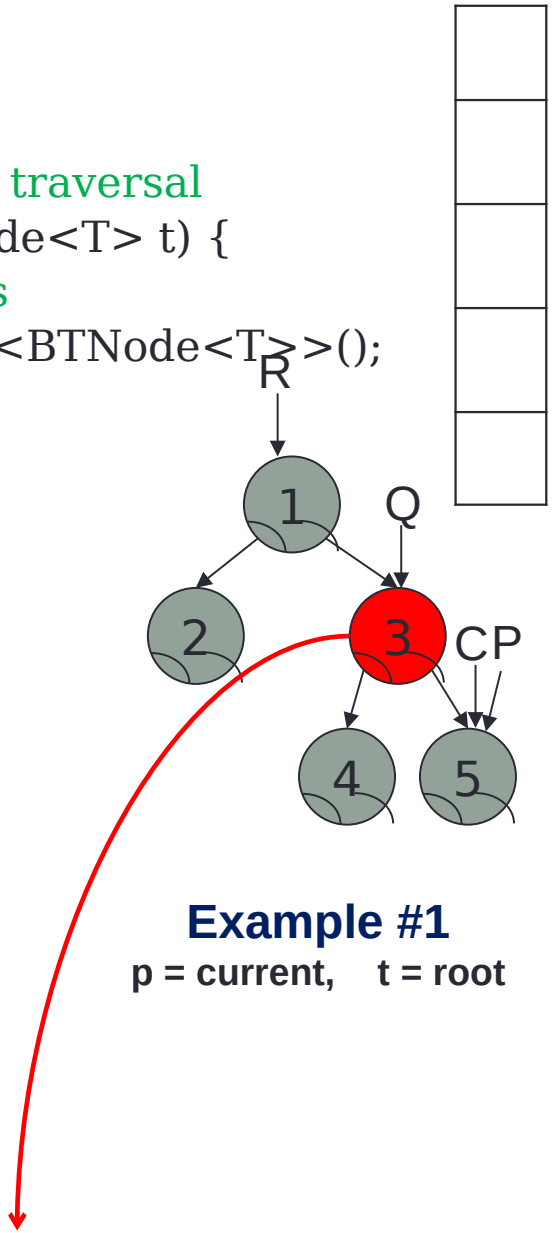
Example #1

p = current, t = root

ADT Binary Tree: Implementation

```
// Non-recursive version of findparent – uses pre-order traversal
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
      // Stack is used to store the right pointers of nodes
      LinkStack<BTNode<T>> stack = new LinkStack<BTNode<T>>();
      BTNode<T> q = t;
      while(q.right != p && q.left != p) {
            if(q.right != null)
                  stack.push(q.right);

            if(q.left != null)
                  q = q.left;
            else
                  q = stack.pop(); // Go right here
      }
      return q;
}
```



Example #1

p = current, t = root

ADT Binary Tree: Implementation

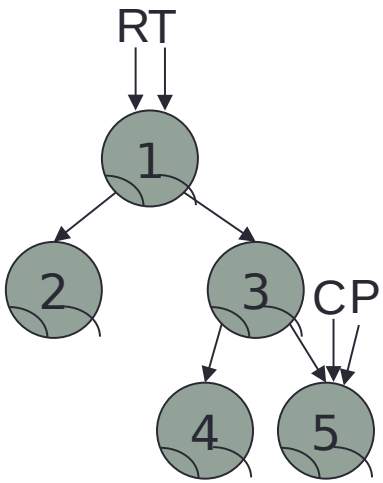
```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```

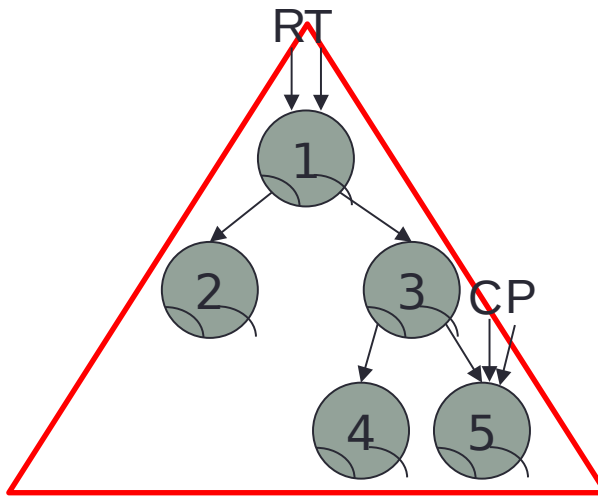


Example #1

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```

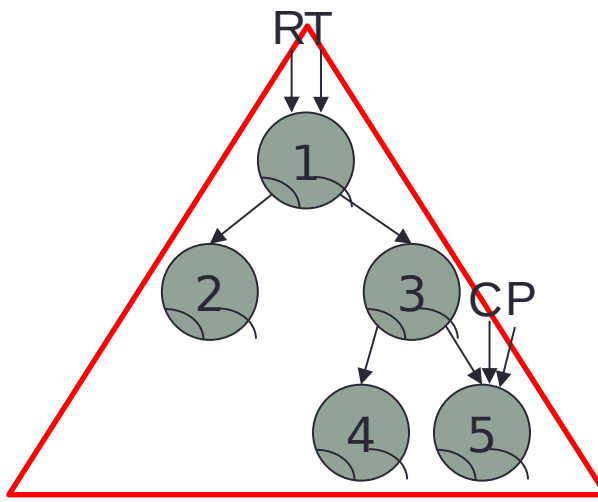


Example #1

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```

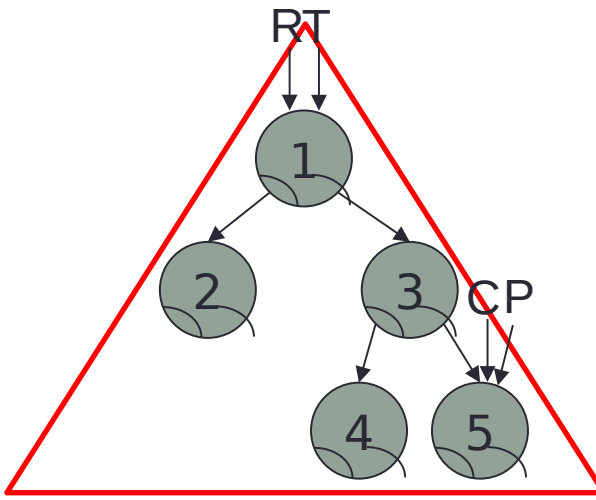


Example #1

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```

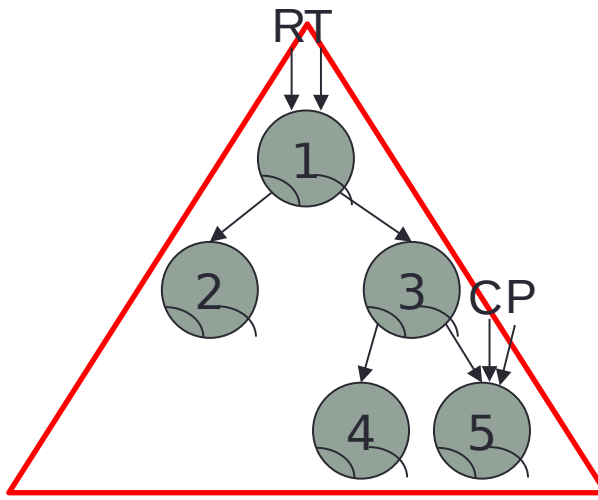


Example #1

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #1

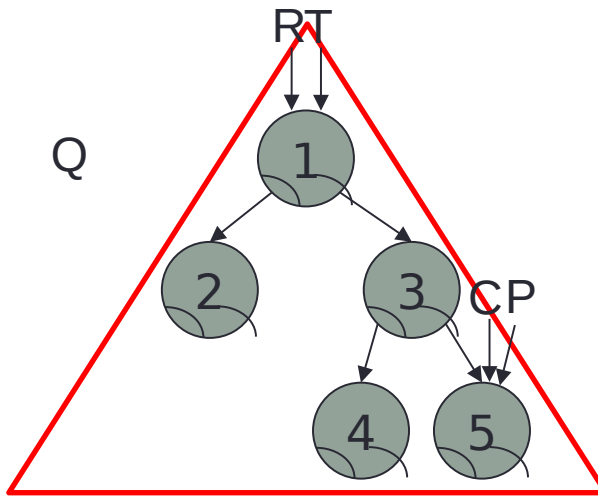
findparent(C,R)

findparent(P,T.Left)

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #1

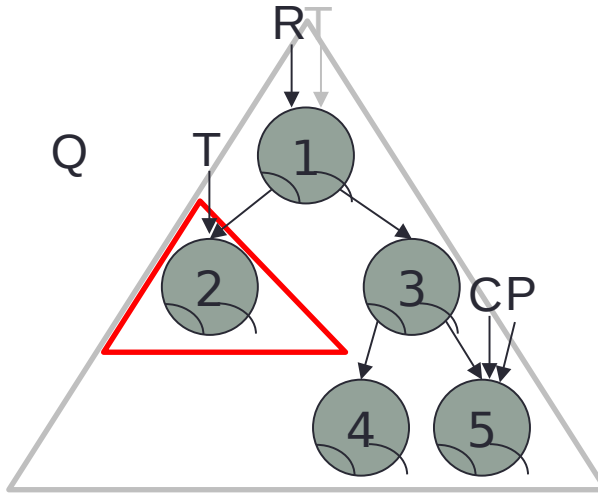
findparent(C,R)

findparent(P,T.Left)

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #1

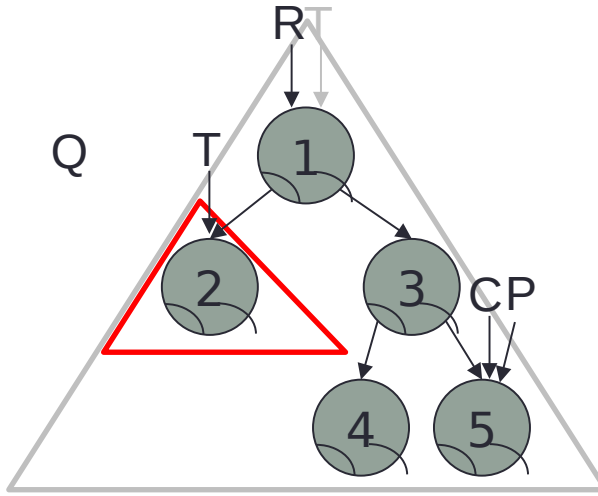
findparent(C,R)

findparent(P,T.Left)

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #1

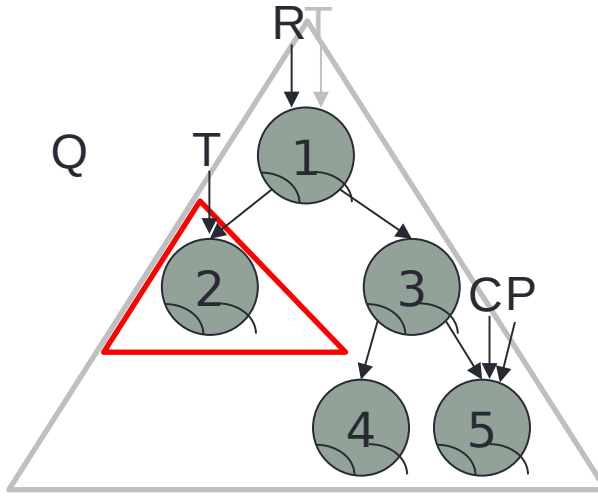
findparent(C,R)

findparent(P,T.Left)

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #1

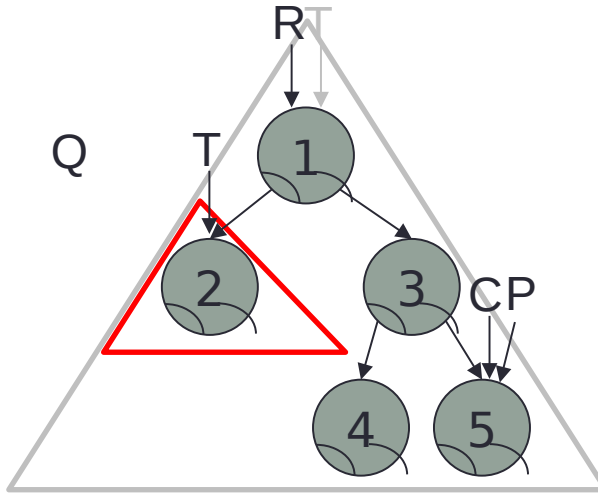
findparent(C,R)

findparent(P,T.Left)

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```

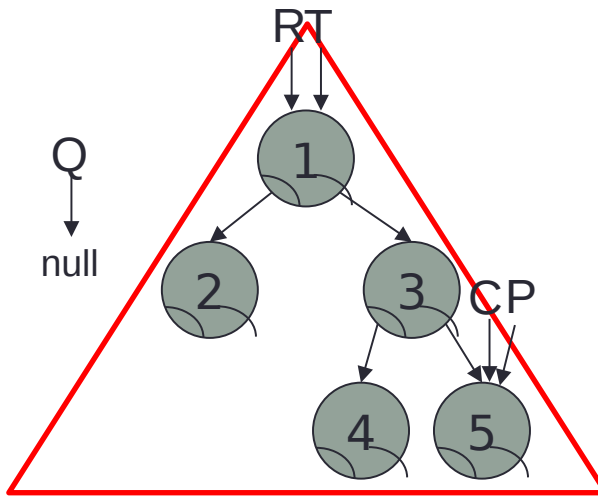


Example #1

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```

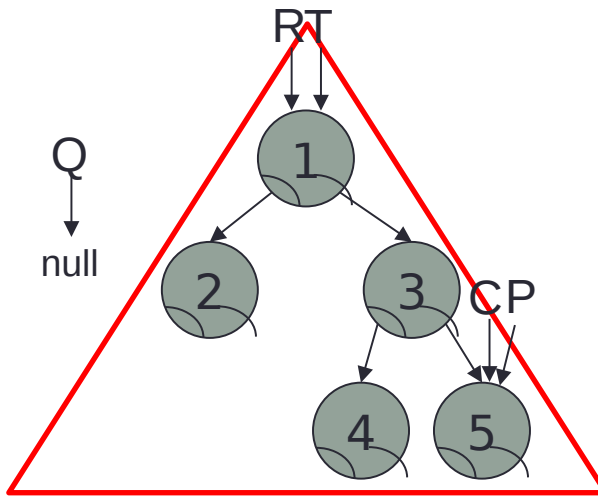


Example #1

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #1

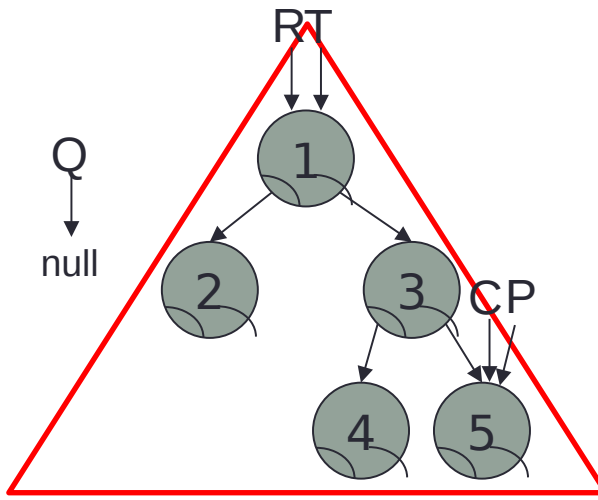
findparent(C,R)

findparent(P,T.Right)

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #1

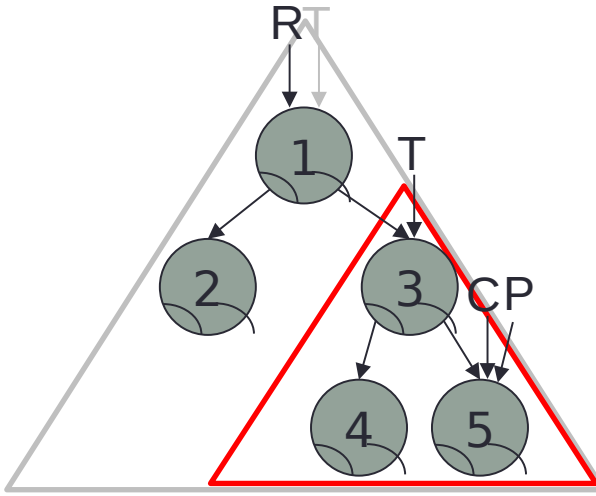
findparent(C,R)

findparent(P,T.Right)

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #1

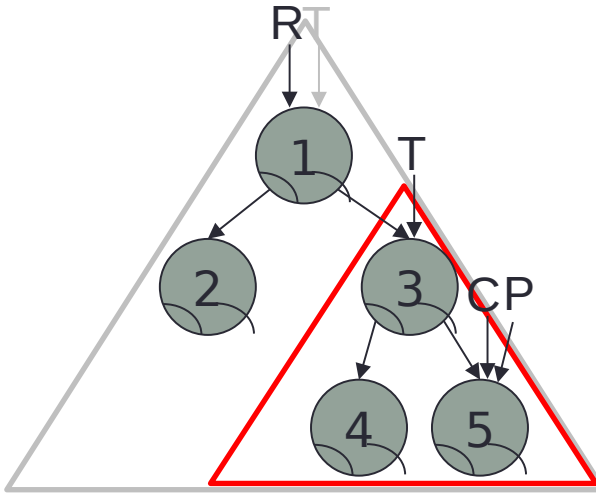
findparent(C,R)

findparent(P,T.Right)

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #1

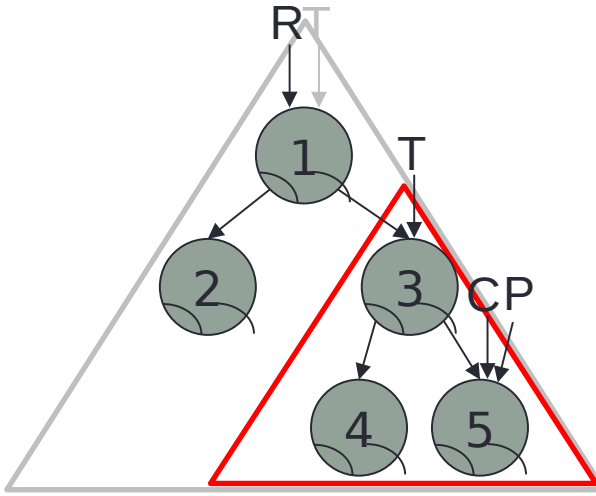
findparent(C,R)

findparent(P,T.Right)

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #1

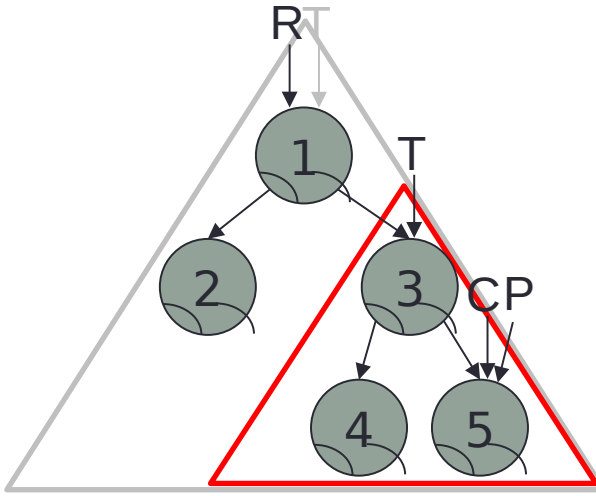
findparent(C,R)

findparent(P,T.Right)

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #1

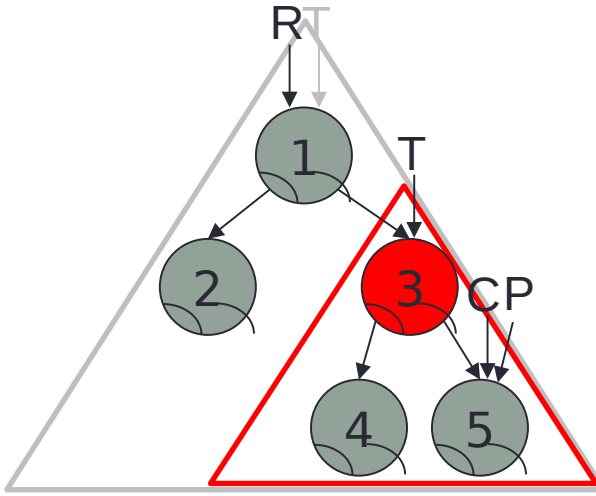
findparent(C,R)

findparent(P,T.Right)

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```

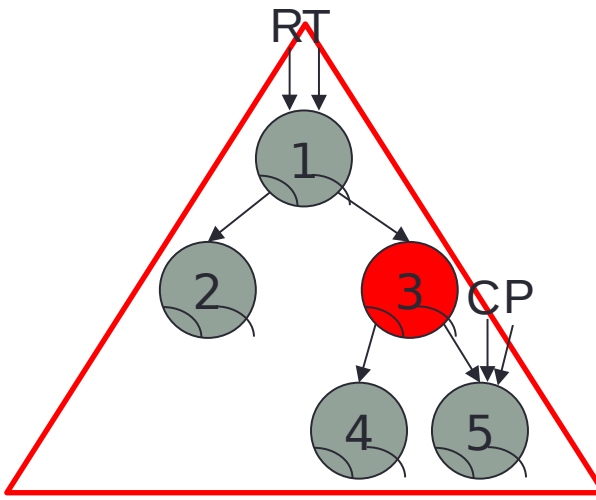


Example #1

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```

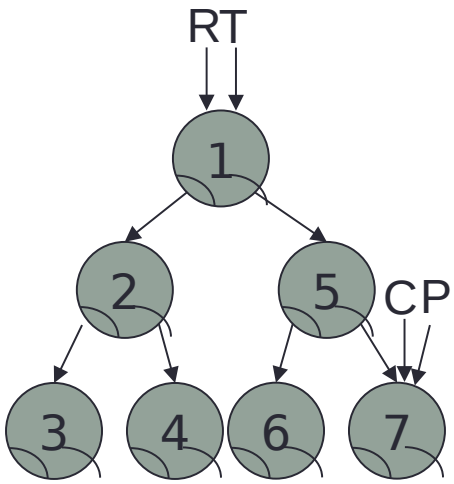


Example #1

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```

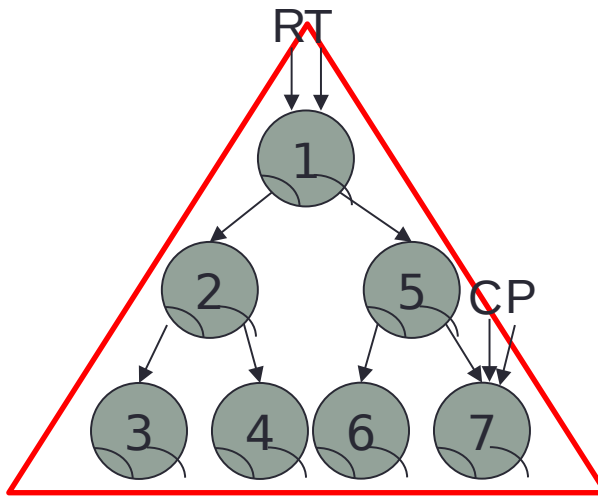


Example #2

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```

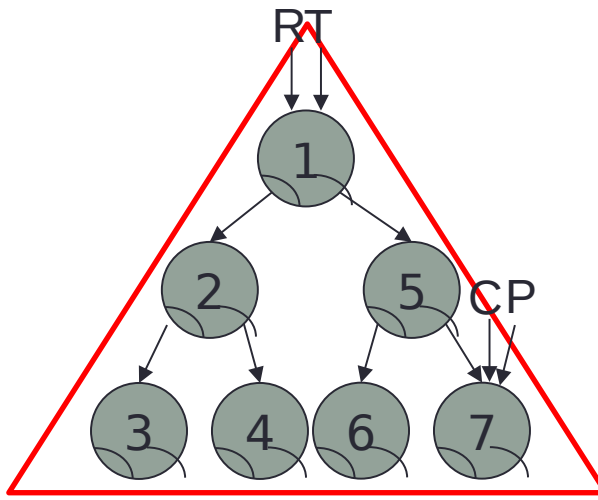


Example #2

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #2

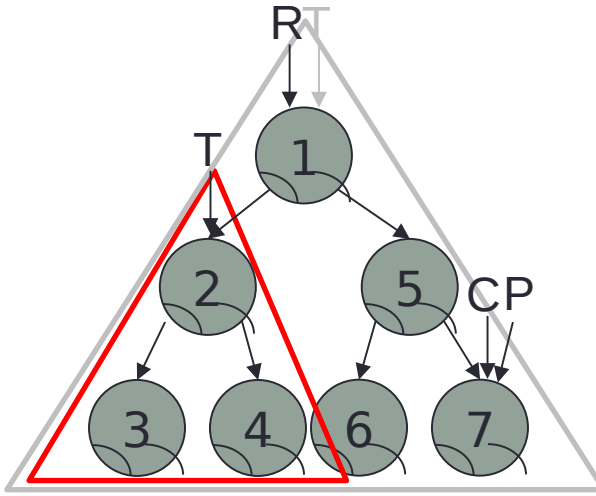
findparent(C,R)

findparent(P,T.Left)

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #2

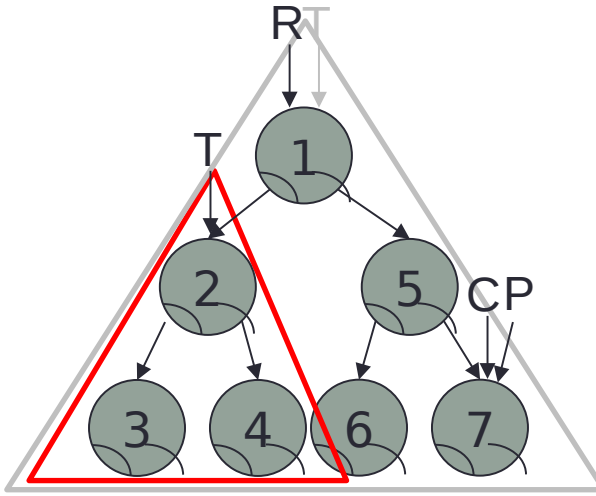
findparent(C,R)

findparent(P,T.Left)

ADT Binary Tree: Implementation

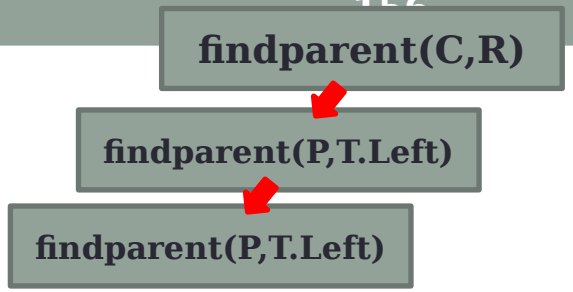
```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



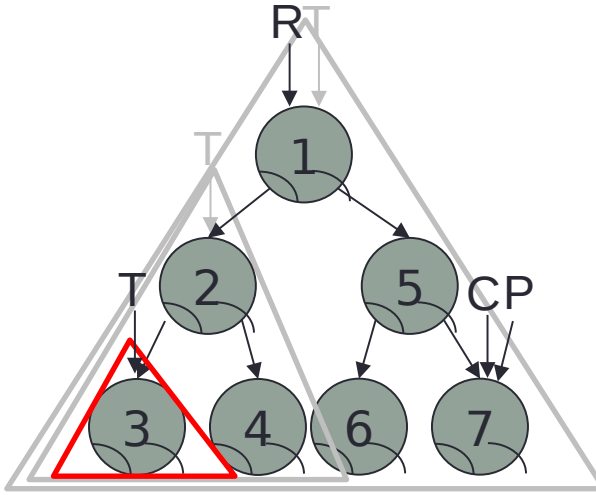
Example #2

ADT Binary Tree: Implementation



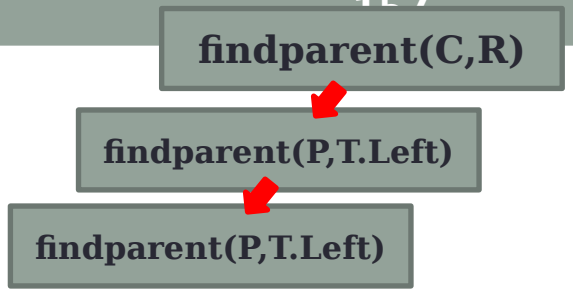

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



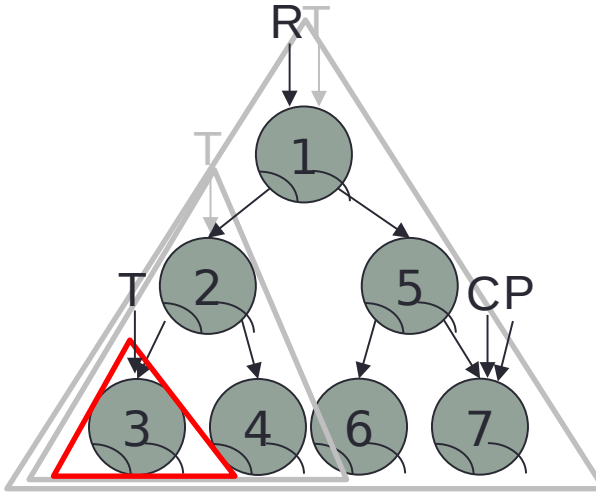
Example #2

ADT Binary Tree: Implementation




```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #2

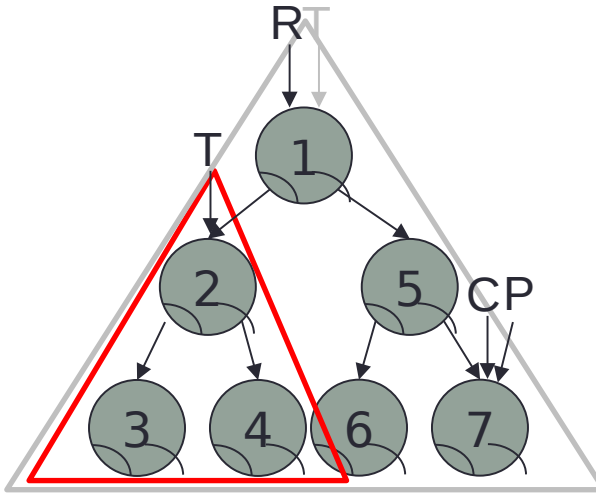
findparent(C,R)

findparent(P,T.Left)

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #2

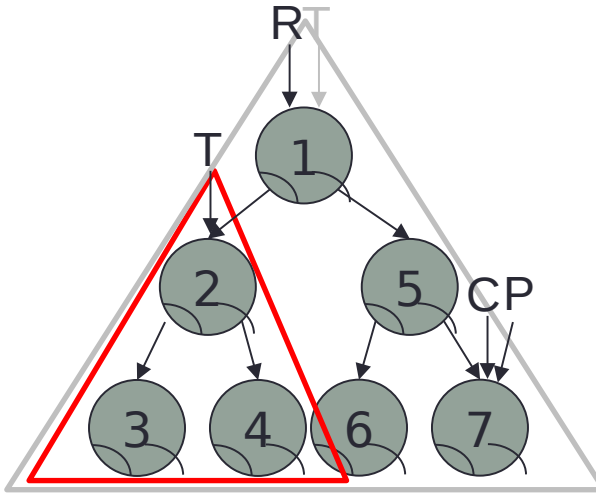
findparent(C,R)

findparent(P,T.Left)

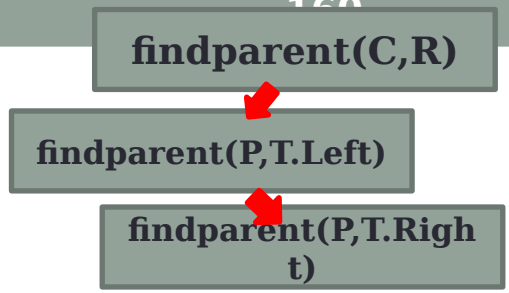
ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



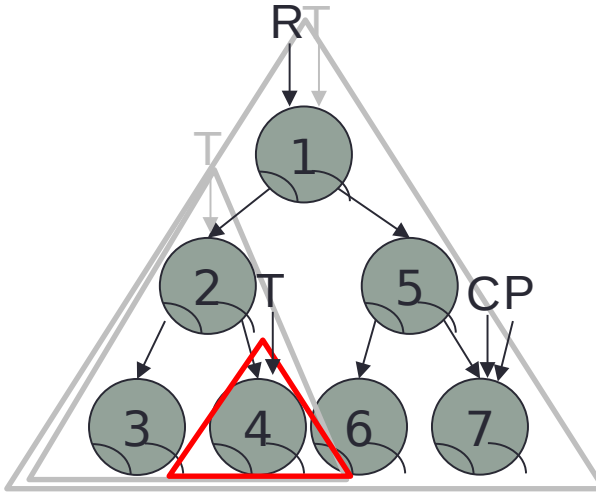
Example #2



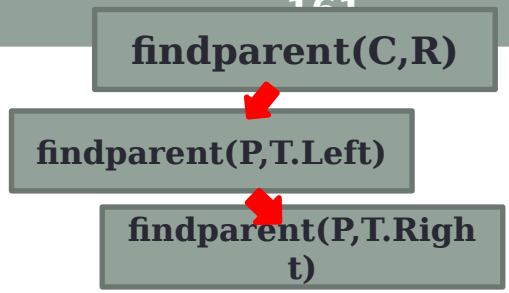
ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



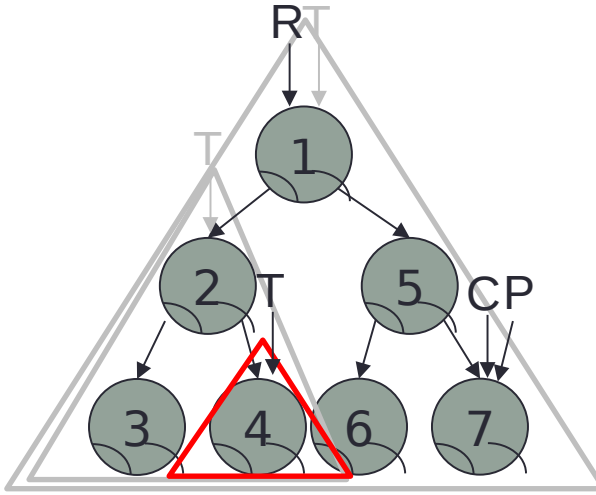
Example #2



ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #2

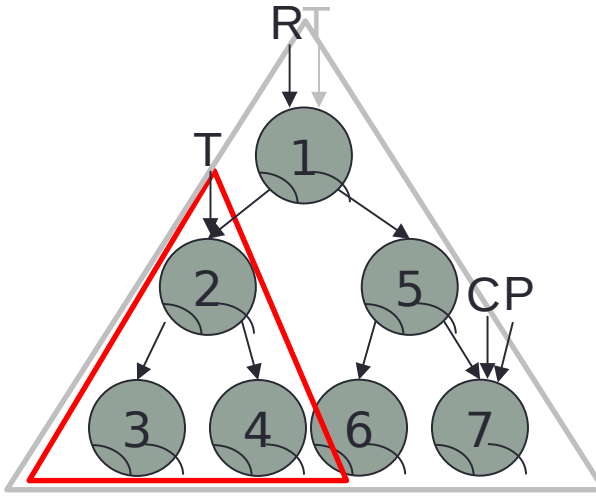
findparent(C,R)

findparent(P,T.Left)

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```

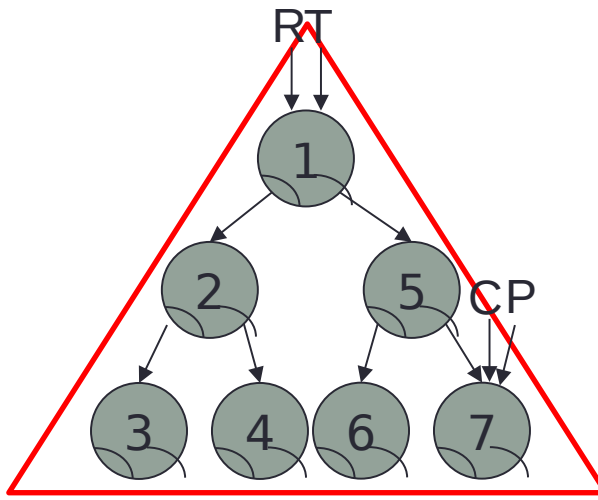


Example #2

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```

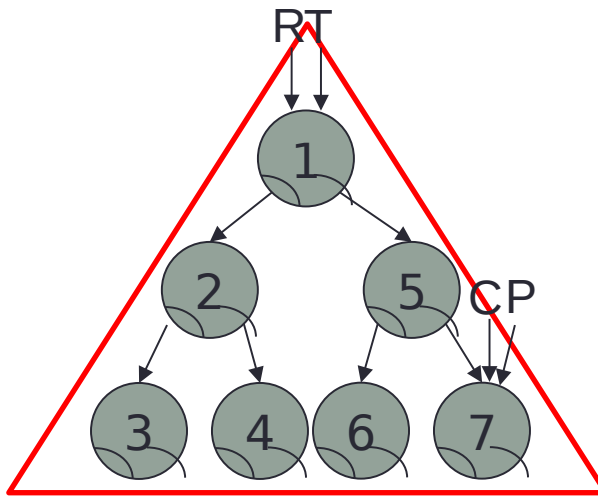


Example #2

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #2

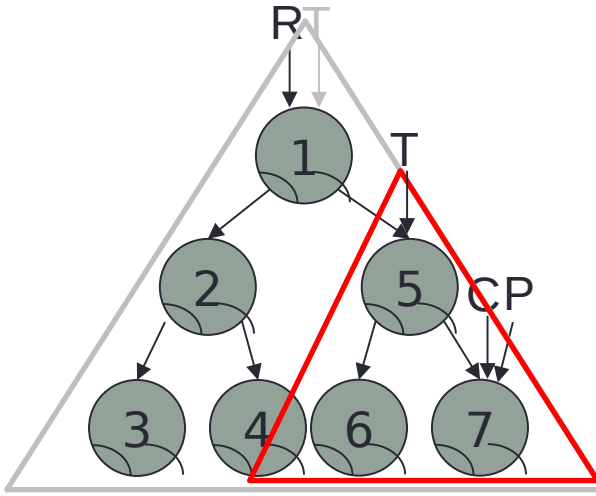
findparent(C,R)

findparent(P,T.Right)

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #2

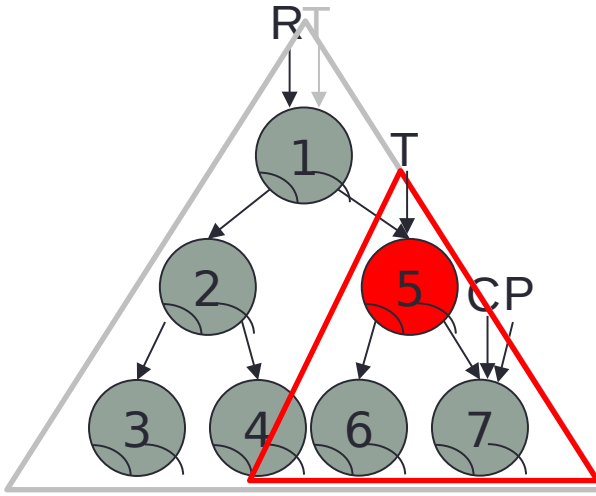
findparent(C,R)

findparent(P,T.Right)

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```

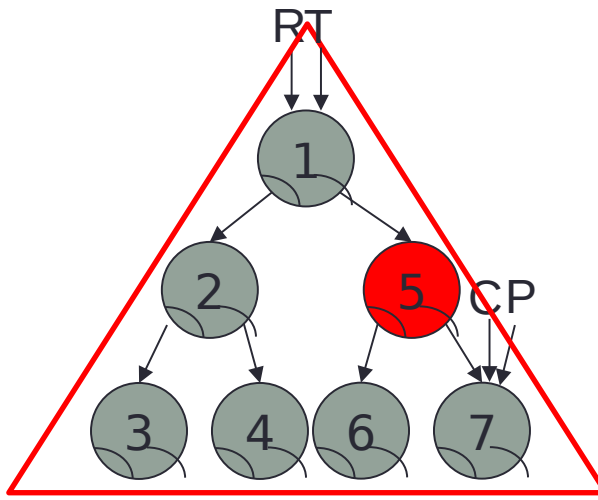


Example #2

ADT Binary Tree: Implementation

```
// Recursive version of findparent – preorder traversal used
private BTNode<T> findparent(BTNode<T> p, BTNode<T> t) {
        if(t == null)
                return null; // empty tree

        if(t.right == null && t.left == null)
                return null;
        else if(t.right == p || t.left == p)
                return t;       // parent is t
        else {
                BTNode q = findparent(p, t.left);
                if (q != null)
                        return q;
                else
                        return findparent(p, t.right);
        }
  }
}
```



Example #2