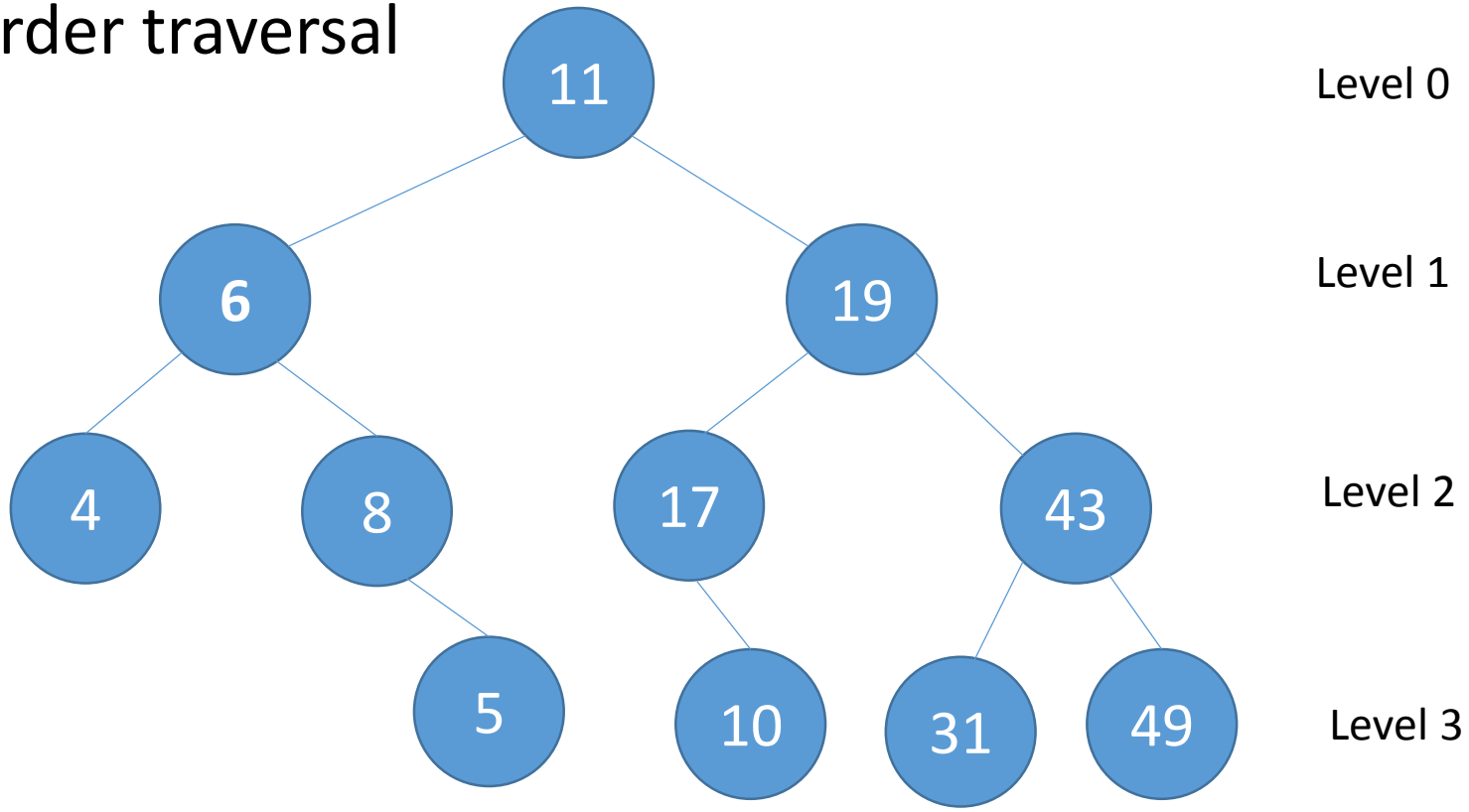
ECE264: Advanced C Programming

Summer 2019

Week 7: Binary Tree Traversal (contd.), Binary Search Trees, Misc. topics (const, variadic functions, macros, bitwise operations, bit fields), Parallel programming using threads

Breadth First Traversal (of a tree)

- Level order traversal



- 11, 6, 19, 4, 8, 17, 43, 5, 10, 31, 49

Breadth first traversal (of a tree)

```
void LOT(Node * root) {
 Queue q;
 push(&q, root);
 while (IsEmpty(&q) == false) {
   Node* headNode = Dequeue(&q)
   print(headNode->val)
   Enqueue(&q, headNode->leftChild);
   Enqueue(&q headNode->rightChild);
 }
}
```

Depth first traversal (of a tree) – iterative code

- Recall Preorder, Inorder, and Postorder were written as recursive codes

```
Preorder(Node* n) {
if(n->val == NULL)
       return;
print(n->val)
Preorder(n->leftChild);
Preorder(n->rightChild);
}
```

```
Inorder(Node* n) {
if(n->val == NULL)
       return;
Inorder(n->leftChild);
print(n->val)
Inorder(n->rightChild);
}
```

```
PostOrder(Node* n) {
if(n->val == NULL)
       return;
Postorder(n->leftChild);
Postorder(n->rightChild);
print(n->val)
}
```

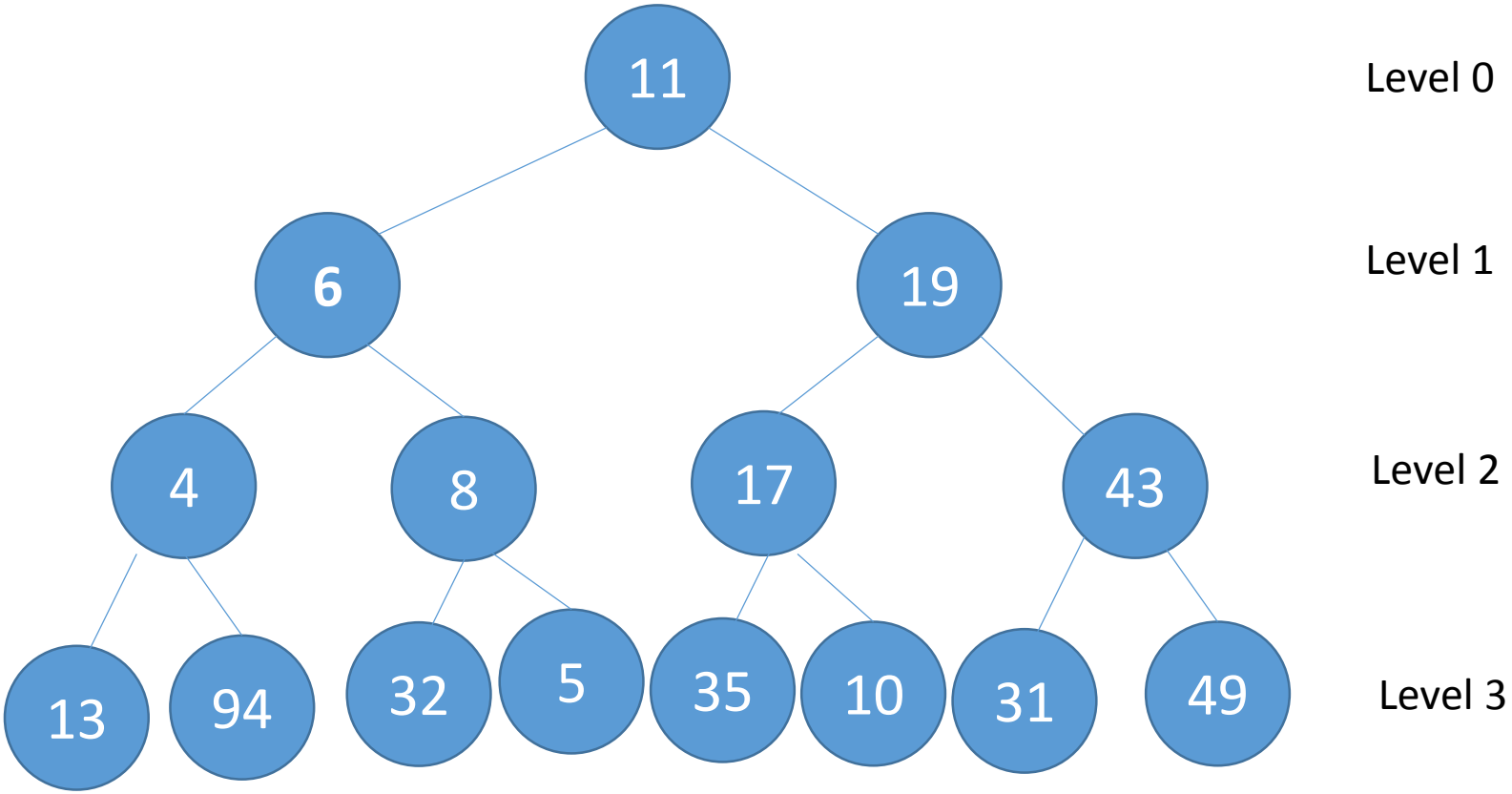
```
void Preorder(Node * root) {
 stack s;
 push(&s, root);
 while (IsEmpty(&s) == false) {
   Node* topNode = Pop(&s)
   print(topNode->val)
   Push(&q, topNode->rightChild);
   Push(&q topNode->leftChild);
 }
}
```

Exercise

What data structure do you need to use for writing an iterative code of Postorder traversal?

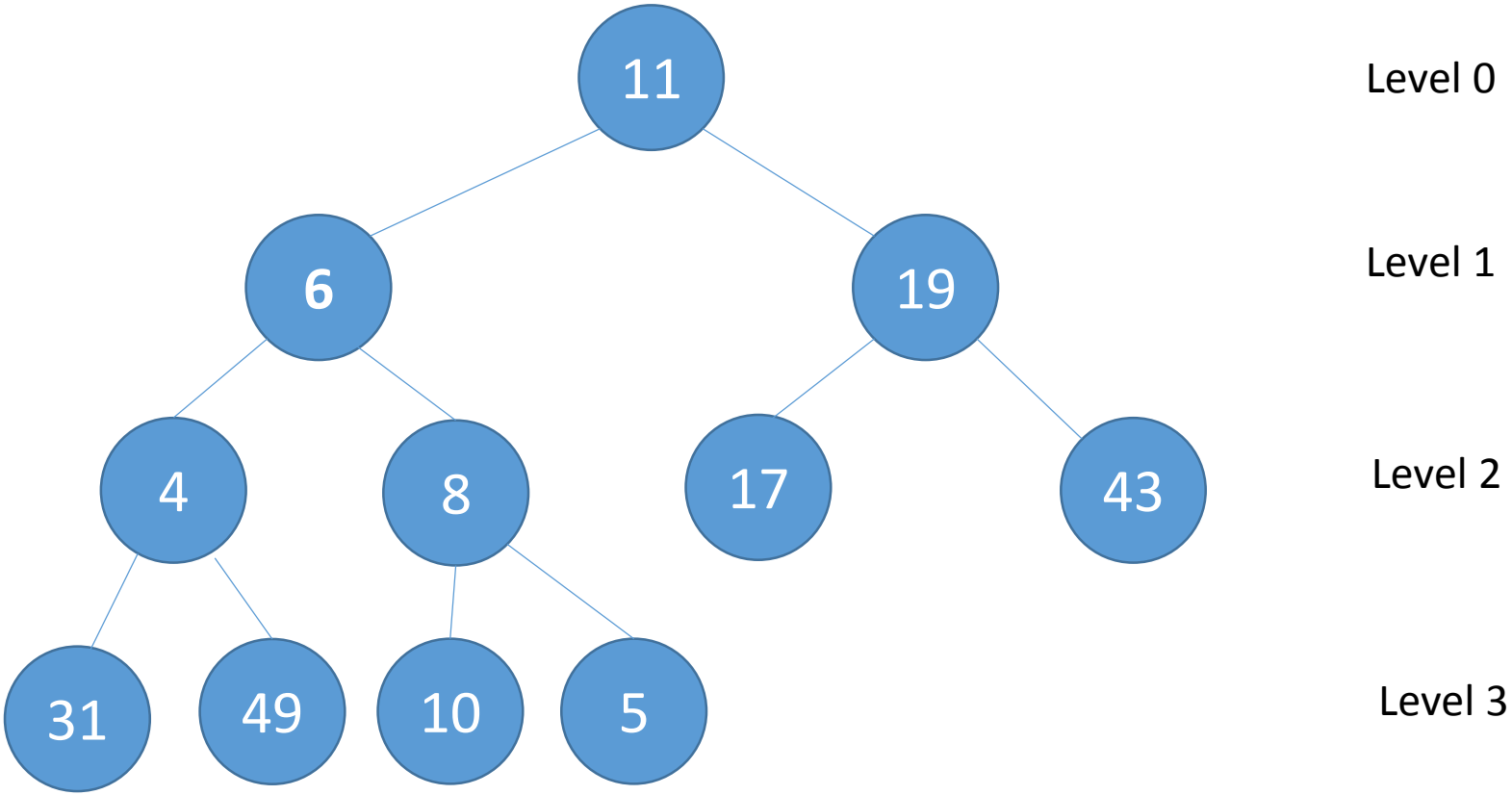
Full Binary Tree

- Every node except leaf has two children



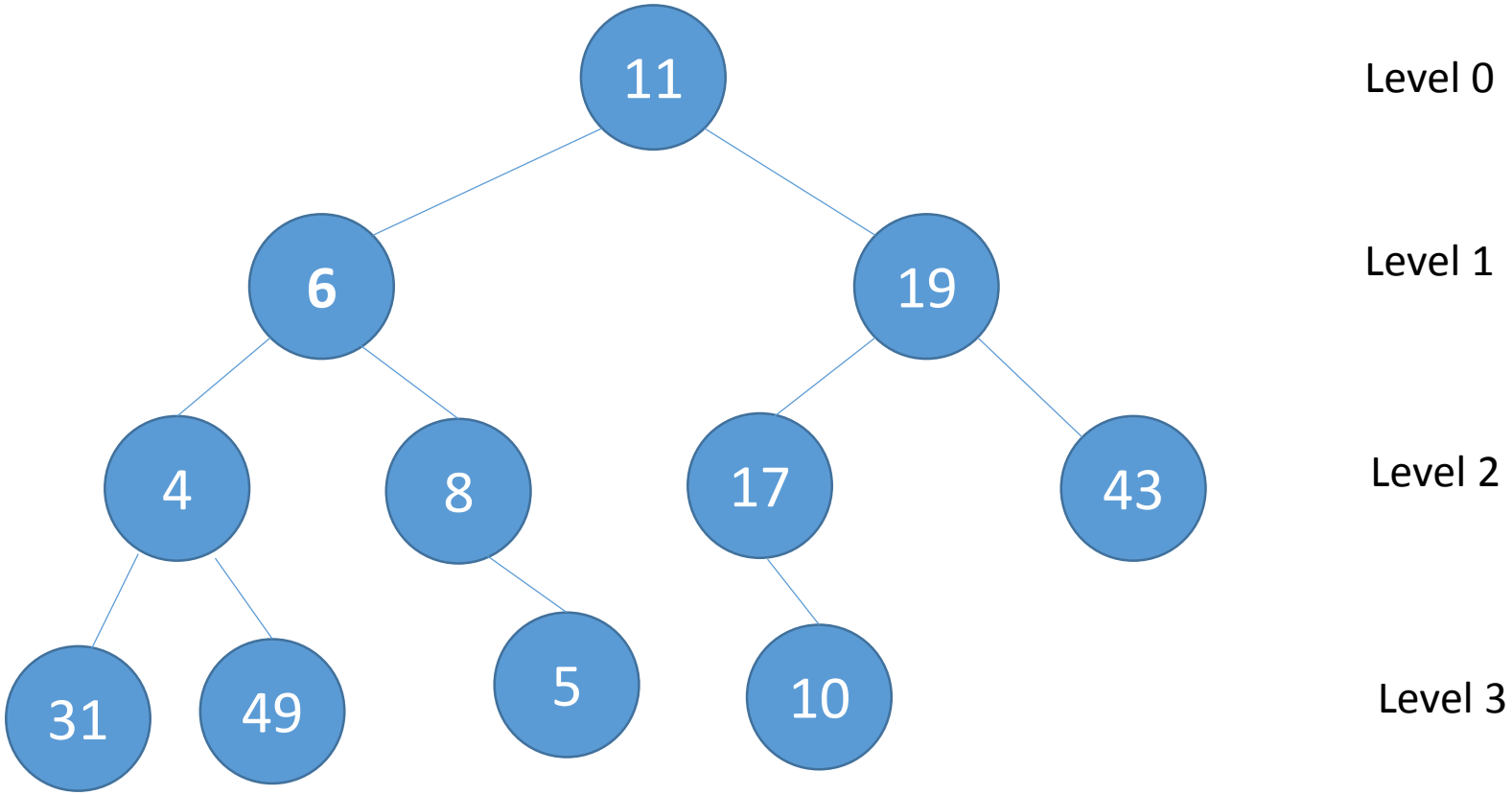
Complete Binary Tree

- Every level except the last is filled and all nodes at the last level are as far left as possible



Exercise

- Complete or Full ?

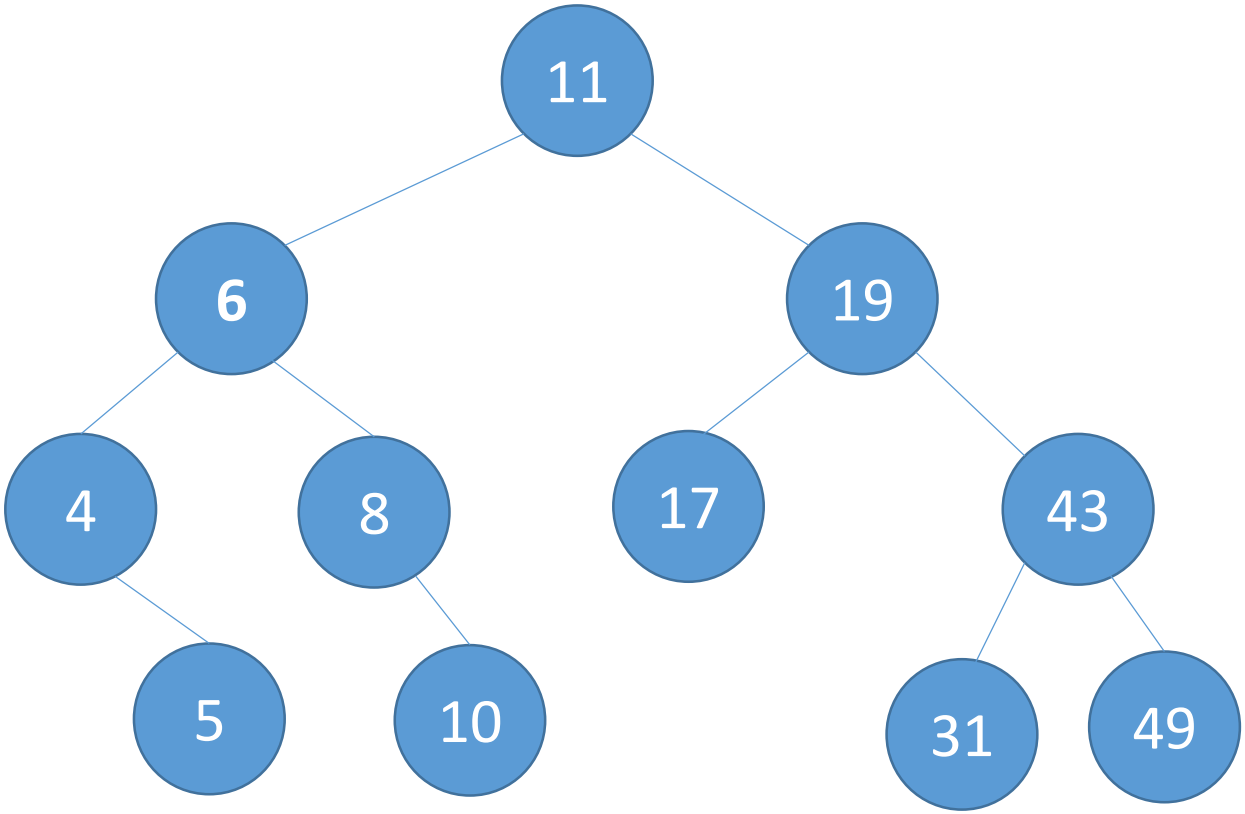


Binary Search Trees (BST)

- For efficient sorting, searching, retrieving
- BST Property:
  - Keys in left subtree are lesser than parent node key
  - Keys in right subtree are greater than parent node key
  - Duplicate keys not allowed

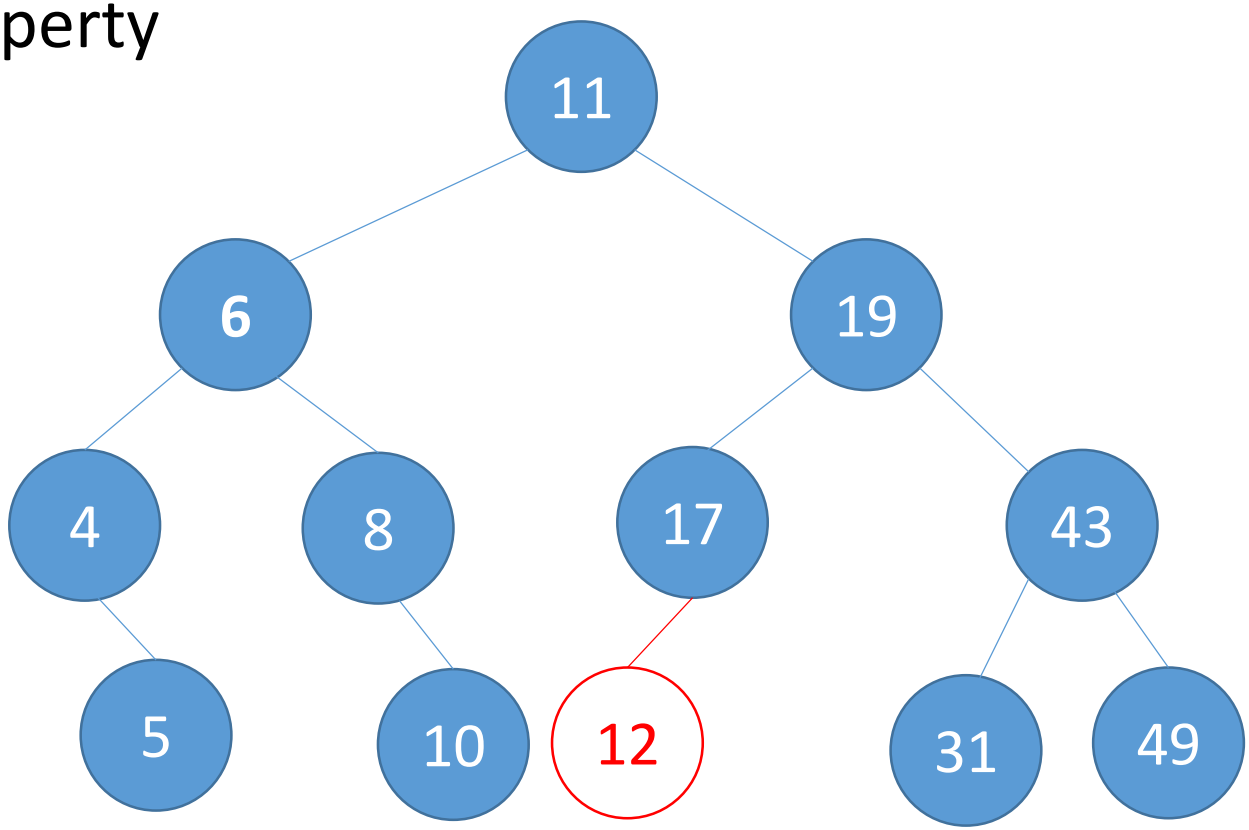
Binary Search Tree

- Example



Binary Search Tree

- Insertion: inserts element without violating the BST property



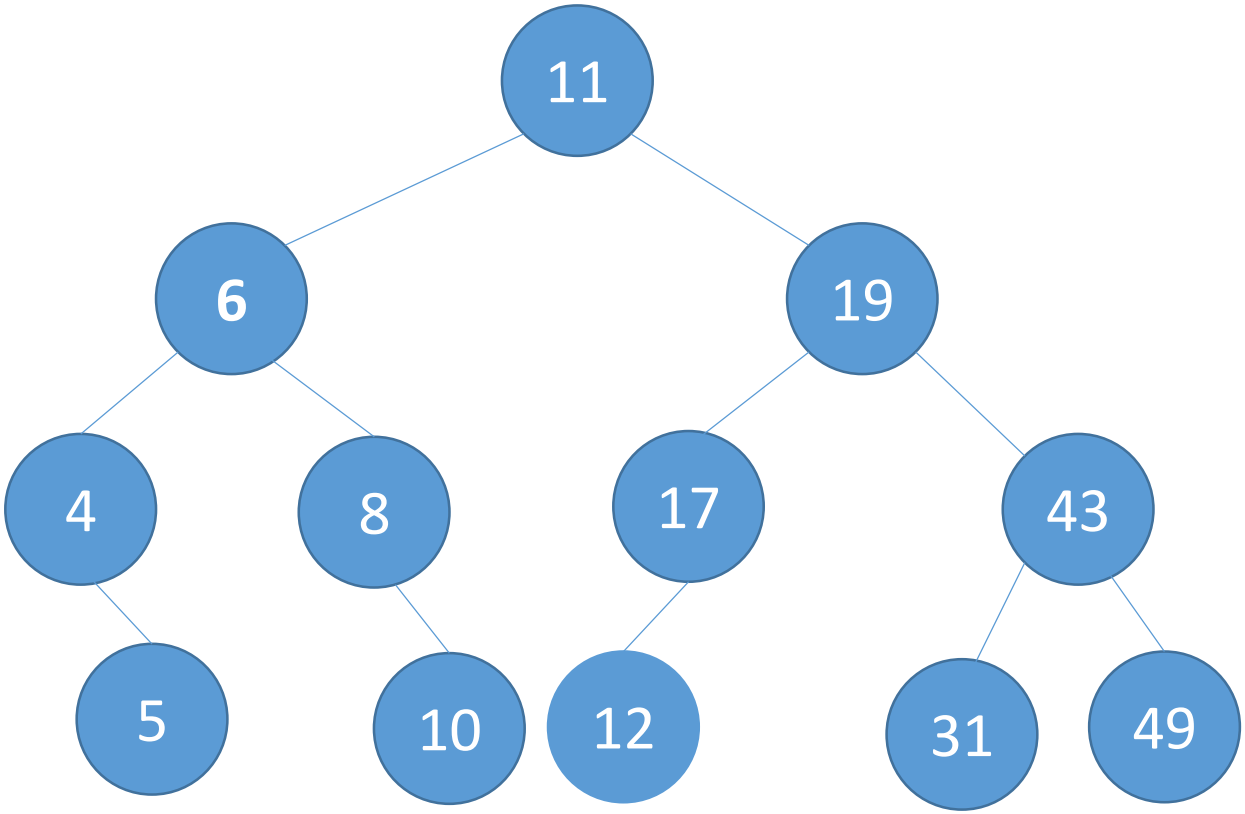
Binary Search Tree

- Insertion

```
1 bool add(TreeNode **rootPtr, int key) {
2      if (*rootPtr == NULL) {
3             *rootPtr = buildNode(key);
4             return true;
5      } else if ((*rootPtr)->val == key) {
6             return false;
7      } else if ((*rootPtr)->val < key) {
8             return add(&((*rootPtr)->right), key);
9      } else {
10            return add(&((*rootPtr)->left), key);
11     }
12 }
```

Binary Search Tree

- Search: returns true if key exists. False otherwise.



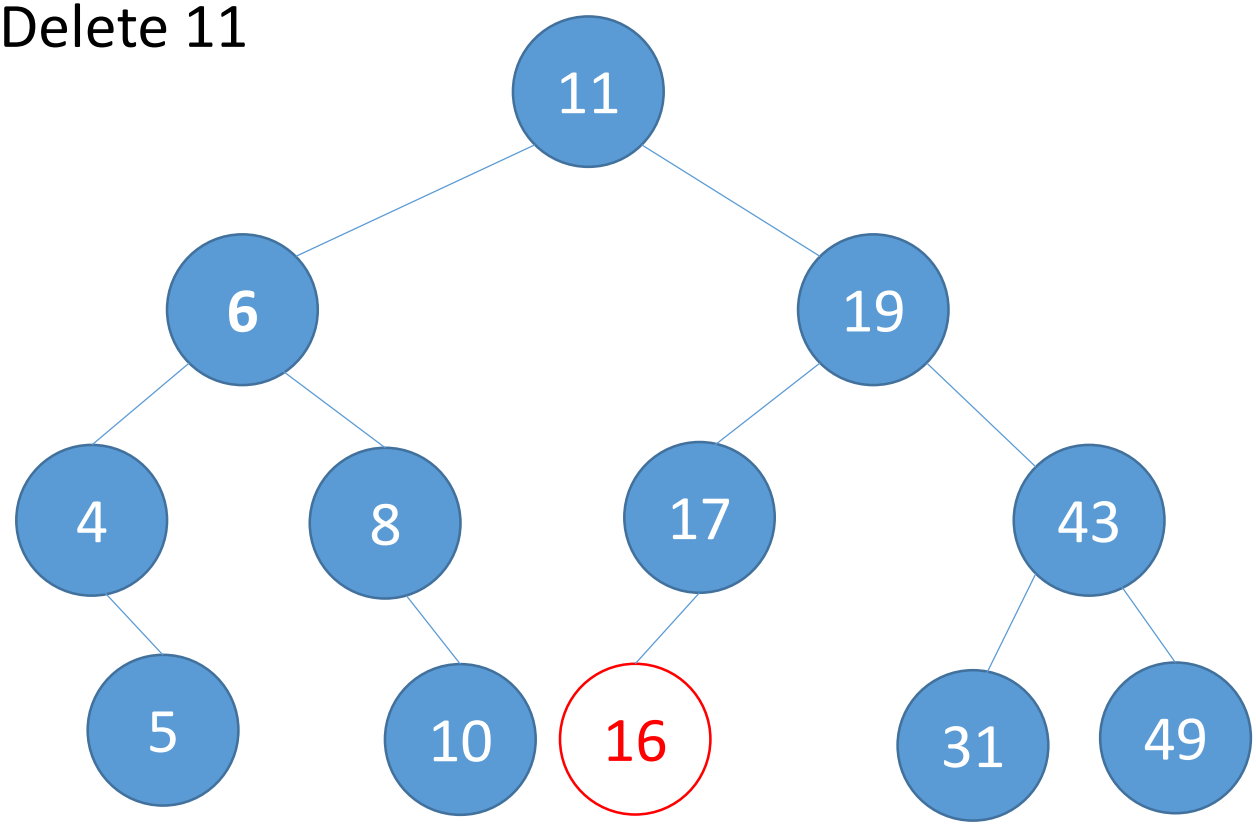
Binary Search Tree

- Search

```
bool Contains(Node* n, int key) {
   if(n == NULL)
       return false;
   if(n->val == key)
       return true;
   else if (n->val > key)
       return Contains(n->leftChild, key);
   else
       return Contains(n->rightChild, key);
}
```

Binary Search Tree

- Removal: remove without violating BST property
  - Delete 11

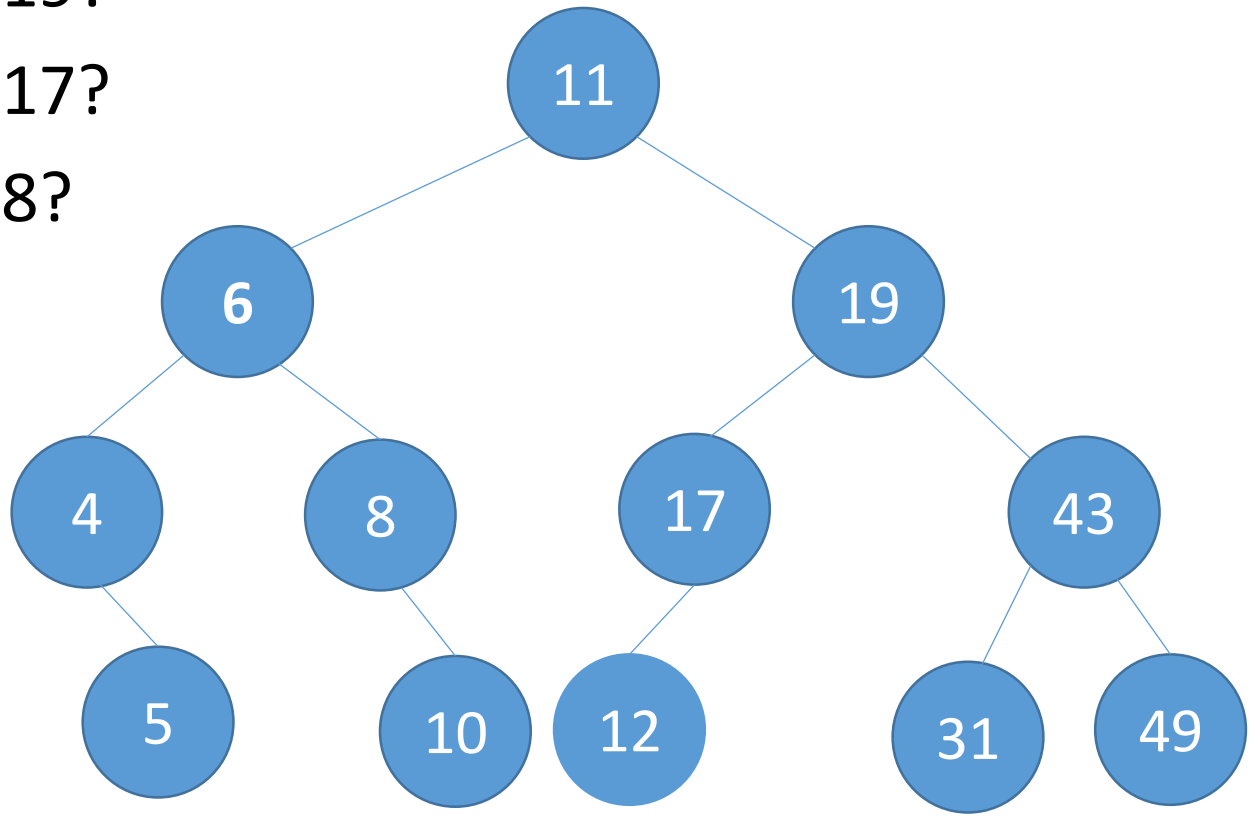


Binary Search Tree

- Removal cases
  - Not in a tree
  - Is a leaf
  - Has one or more children
- Return true if key removed. False otherwise.

Exercise

- Remove 19?
- Remove 17?
- Remove 8?



BST remove node

- Removal code:

Applications – Parsing of expression trees

- Infix expression, prefix expression

  ((7 + (8 * 10)) - (2 + 3))