Binary Search Tree (IV): Traversals

CSCD 300 – Data Structures

Eastern Washington University

Goal: learn and implement various BST traversal procedures.

```
class BST{
   BST_Node root;   //the root of the BST

   BST(){ root = null;} /* the constructor */

   void InOrder_Traversal(BST_Node subtree_root){...} /* Inorder traversal
      and print all the nodes in the subtree rooted at "subtree_root". */

   void PreOrder_Traversal(BST_Node subtree_root){...} /* Preorder traversal
      and print all the nodes in the subtree rooted at "subtree_root". */

   void PostOrder_Traversal(BST_Node subtree_root){...} /* Postorder traversal
      and print all the nodes in the subtree rooted at "subtree_root". */

   void LevelOrder_Traversal(BST_Node subtree_root){...} /* Level-order traversal
      and print all the nodes in the subtree rooted at "subtree_root". */

   /* Other methods will follow here  */
}
```

Outline

1. Introduction
2. Inorder traversal
3. Preorder traversal
4. Postorder traversal
5. Level-order traversal

Introduction

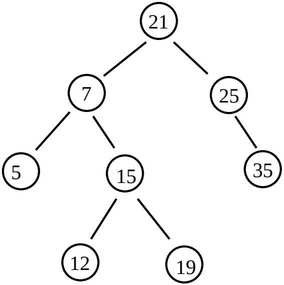
Tree traversal is to visit (print) all the nodes in the tree in some order.

- in-order traversal: traverse the left subtree first, then traverse the root, then traverse the right subtree of the root. Each subtree is also in-order traversed.
- pre-order traversal: traverse the root first, then traverse the left subtree, then traverse the right subtree of the root. Each subtree is also pre-order traversed.
- post-order traversal: traverse the left subtree first, then traverse the right subtree of the root, then traverse the root. Each subtree is also post-order traversed.
- Level-order traversal: traverse all the tree nodes from the top level to the bottom level. In each particular level, traverse the tree nodes from the left to the right.

Theorem

The time complexity of all the BST traversal procedures that we will present next is $O(n)$, where n is the number of nodes in the tree, because every node is visited for no more than a constant number of times.

Examples



- Inorder traversal: $5, 7, 12, 15, 19, 21, 25, 35$
- Preorder traversal: $21, 7, 5, 15, 12, 19, 25, 35$
- Postorder traversal: $5, 12, 19, 15, 7, 35, 25, 21$
- Level-order traversal: $21, 7, 25, 5, 15, 35, 12, 19$

Inorder traversal

Idea: use recursion.

```
void BST::InOrder_Traversal(BST_Node subtree_root){
   if(subtree_root != null)
      InOrder_Traversal(subtree_root.left);
      print(subtree_root.key);
      InOrder_Traversal(subtree_root.right);
}
```

Function call `InOrder_Traversal(root)` will in-order traverse the whole tree.

Inorder traversal prints the keys in the BST in ascending order

Preorder traversal

Idea: use recursion.

```
void BST::PreOrder_Traversal(BST_Node subtree_root){
   if(subtree_root != null)
      print(subtree_root.key);
      PreOrder_Traversal(subtree_root.left);
      PreOrder_Traversal(subtree_root.right);
}
```

Function call `PreOrder_Traversal(root)`
will pre-order traverse the whole tree.

Postorder traversal

Idea: use recursion.

```
void BST::PostOrder_Traversal(BST_Node subtree_root){
   if(subtree_root != null)
      PostOrder_Traversal(subtree_root.left);
      PostOrder_Traversal(subtree_root.right);
      print(subtree_root.key);
}
```

Function call `PostOrder_Traversal(root)`
will post-order traverse the whole tree.

Level-order traversal

Idea: use a FIFO queue.

```
void BST::LevelOrder_Traversal(BST_Node subtree_root){
   Q = new FIFO; //Create a new FIFO queue of tree node type.

   Q.enqueue(subtree_root);

   while(Q.size > 0){
      BST_Node node = Q.dequeue();
      print(node.key);
      if(node.left != null)
         Q.enqueue(node.left);
      if(node.right != null)
         Q.enqueue(node.right);
   }
}
```

Function call `LevelOrder_Traversal(root)`
will level-order traverse the whole tree.