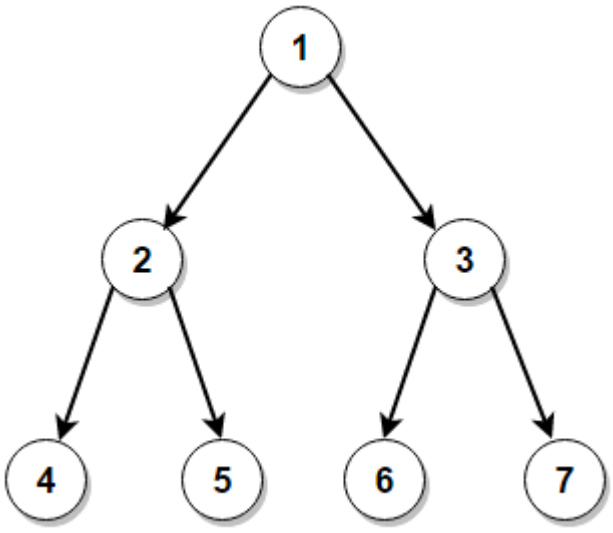
Efficiently print all nodes between two given levels in a binary tree

Given a binary tree, efficiently print all nodes in it between two given levels. The nodes for any level should be printed from left and right.

For example, if the starting level is 2 and the ending level is 3, the solution should print nodes in order `[2, 3, 4, 5, 6, 7]`.



Practice This Problem

A simple solution would be to print all nodes of given levels one by one. We can print all nodes present in a level by modifying the **preorder traversal** of the tree. The time complexity of this solution is $O(n^2)$, where `n` is the total number of nodes in the binary tree.

We can reduce the time complexity to $O(n)$ by modifying the **level order traversal**. Following is a pseudocode for a modified level order traversal, which maintains the level of each node:

```
levelorder(root, start, end)
q —> empty queue
q.enqueue(root)
level —> 0
while (not q.isEmpty())
  size —> q.size()
  level = level + 1
  while (size)
  node —> q.dequeue()
  if (level between start and end)
    print(node)
  if (node.left <> null)
    q.enqueue(node.left)
  if (node.right <> null)
    q.enqueue(node.right)
  size = size - 1
```

Following is the C++, Java, and Python program that demonstrates it:

C++

```java
import java.util.ArrayDeque;
import java.util.Queue;

// A class to store a binary tree node
class Node
{
    int key;
    Node left = null, right = null;

    Node(int key) {
        this.key = key;
    }
}

class Main
{
    // Iterative function to print all nodes between two given
    // levels in a binary tree
    public static void printNodes(Node root, int start, int end)
    {
        if (root == null) {
            return;
        }

        // create an empty queue and enqueue the root node
        Queue<Node> queue = new ArrayDeque<>();
        queue.add(root);

        // to store the current node
        Node curr = null;

        // maintains the level of the current node
        int level = 0;

        // loop till queue is empty
        while (!queue.isEmpty())
        {
            // increment level by 1
            level++;

            // calculate the total number of nodes at the current level
            int size = queue.size();

            // process every node of the current level and enqueue their
            // non-empty left and right child
            while (size-- > 0)
            {
                curr = queue.poll();

                // print the node if its level is between given levels
                if (level >= start && level <= end) {
                    System.out.print(curr.key + " ");
```

```
56                      queue.add(curr.left);
57                  }
58
59                  if (curr.right != null) {
60                      queue.add(curr.right);
61                  }
62              }
63
64              if (level >= start && level <= end) {
65                  System.out.println();
66              }
67          }
68      }
69
70      public static void main(String[] args)
71      {
72          Node root = new Node(15);
73          root.left = new Node(10);
74          root.right = new Node(20);
75          root.left.left = new Node(8);
76          root.left.right = new Node(12);
77          root.right.left = new Node(16);
78          root.right.right = new Node(25);
79          root.right.right.right = new Node(30);
80
81          int start = 2, end = 3;
82          printNodes(root, start, end);
83      }
84  }
```

Download Run Code

Output:

```
10 20
8 12 16 25
```

Python

The time complexity of the above solution is `O(n)` and requires `O(n)` extra space, where `n` is the size of the binary tree.

as a key. Finally, print all nodes corresponding to every level between given levels.

Following is the C++, Java, and Python implementation of it:

C++

Java

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

// A class to store a binary tree node
class Node
{
    int key;
    Node left = null, right = null;

    Node(int key) {
        this.key = key;
    }
}

class Main
{
    // Traverse the tree in a preorder fashion and store nodes in a map
    // corresponding to their level
    public static void printNodes(Node root, int start, int end, int level,
                                  Map<Integer, List<Integer>> map)
    {
        // base case: empty tree
        if (root == null) {
            return;
        }

        // push the current node into the map corresponding to their level
        if (level >= start && level <= end)
        {
            map.putIfAbsent(level, new ArrayList<>());
            map.get(level).add(root.key);
        }

        // recur for the left and right subtree by increasing the level by 1
        printNodes(root.left, start, end, level + 1, map);
        printNodes(root.right, start, end, level + 1, map);
    }

    // Recursive function to print all nodes between two given
    // levels in a binary tree
```

```
47
48          // traverse the tree and insert its nodes into the map
49          // corresponding to their level
50          printNodes(root, start, end, 1, map);
51
52          // iterate through the map and print all nodes between given levels
53          for (int i = start; i <= end; i++) {
54              if (map.containsKey(i)) {
55                  System.out.println("Level " + i + ": " + map.get(i));
56              }
57          }
58      }
59
60      public static void main(String[] args)
61      {
62          Node root = new Node(15);
63          root.left = new Node(10);
64          root.right = new Node(20);
65          root.left.left = new Node(8);
66          root.left.right = new Node(12);
67          root.right.left = new Node(16);
68          root.right.right = new Node(25);
69          root.right.right.right = new Node(30);
70
71          int start = 2, end = 3;
72          printNodes(root, start, end);
73      }
74  }
```

Download Run Code

Output:

```
Level 2: [10, 20]
Level 3: [8, 12, 16, 25]
```

Python

The time complexity of the above solution is `O(n)` and requires `O(n)` extra space, where `n` is the size of the binary tree.

Binary Tree, Queue

Breadth-first search, Depth-first search, Easy, FIFO, Hashing, Recursive

Techie Delight </>

Resources

All Problems

DSA Practice

Top 100 Most Liked Problems

Top 50 Classic Problems

Top Algorithms

Company

Contact us

Privacy Policy

Terms of Service

Subscribe to new posts