Contents

Interview

Arvind Kumar

April 2015

Chapter 1

Binary trees

1.1 Problem

Deleting a node in Binary Search Tree. This operation can be divided in three basic cases. Let the node to be deleted is Z. The below code deletes all types of node. That is node having 0 child, 1 chid or 2 child

```java
private Node deleteNode(Node x, K key) {
  if (x == null) return null;
  int cmp = key.compareTo(x.key);
  if (cmp < 0) x.left = deleteNode(x.left, key);
  else if (cmp > 0) x.right = deleteNode(x.right, key);
  else {
    //Case 1 : 0 or 1 child
    if (x.left == null) return x.right;
    if (x.right == null) return x.left;

    // Case 2: 2 children
    Node t = x;
    x = getMinimum(t.right);
    x.right = deleteMin(t.right); // This returns the root of the tree
        on which it is invoked.
    x.left = t.left;
  }
  return x;
}

private Node deleteMin(Node x) {
  if (x == null) return null;
  if (x.left == null) return x.right;
  x.left = deleteMin(x.left);
  return x;
}
```

1.2 Problem

Refer to the link for problem statement find-closest-leaf-binary-tree

1.3 Problem

The link (diagonal-sum-binary-tree) provides the detail of the problem. Below is the code.

```
  private void diagonalSums(Node x) {
  if (x == null) return;
  Queue<Node> queue = new LinkedList<>();
  queue.add(x);
  while (!queue.isEmpty()) {
    int count = queue.size();
    int diSum = 0;
    while (count != 0) {
      Node curr = queue.remove();
      System.out.print(curr.key + " ");
      diSum += (int) curr.key;
      if (curr.left != null) queue.add(curr.left);
      while (curr.right != null) {
        System.out.print(curr.right.key + " ");
        diSum += (int) curr.right.key;
        if (curr.right.left != null) queue.add(curr.right.left);
        curr = curr.right;
      }
      count--;
    }
    System.out.println(": Sum = " + diSum);
    System.out.println();
  }
}
```

1.4 Problem

Given a Binary Tree, we need to print the bottom view from left to right. A node x is there in output if x is the bottommost node at its horizontal distance. Horizontal distance of left child of a node x is equal to horizontal distance of x minus 1, and that of right child is horizontal distance of x plus 1. (bottom-view-binary-tree)

```
public void bottomView() {
  Map<Integer, K> map = new HashMap<>();
  bottomView(root, map, 0, 0, Integer.MIN_VALUE);
  for(Map.Entry<Integer, K> entries : map.entrySet()){
    System.out.println(entries.getValue());
  }
}

private void bottomView(Node x, Map<Integer, K> map, int level, int hd,
    int maxlevel) {
  if (x == null) return;
  if (level > maxlevel) {
    map.put(hd, x.key);
    maxlevel = level;
  }
  bottomView(x.left, map, level + 1, hd - 1, maxlevel);
  bottomView(x.right, map, level + 1, hd + 1, maxlevel);
}
```

Chapter 2

Interval Trees

Interval tree is BST created using the low endpoint of an interval as key.Each node of the tree consists of low end point ’a’, high endpoint ’b’ and a max field. The max field of a node x contains the maximum end point value in the subtree root at x. We need this max field for performing search operations.

2.1 Creating a Interval Tree

```
// It uses a helper Interval class which is a convenient of representing
    intervals.
public class IntervalTree<K extends Comparable<K>> {

  // Root of the tree
  private Node root;

  // Node representation of Interval tree
  private class Node {
    Interval<K> interval;
    K          max;
    Node       left;
    Node       right;

    Node(Interval<K> a) {
      this.interval = a;
    }
  }


  public void insert(Interval<K> a) {
    root = insert(root, a);
  }
```

```
// Method to insert an interval in the Interval tree
private Node insert(Node x, Interval<K> r) {
  if (x == null) x = new Node(r);
  int cmp = r.a.compareTo(x.interval.a);
  if (cmp < 0) x.left = insert(x.left, r);
  else if (cmp > 0) x.right = insert(x.right, r);
  else{} // Ignore if equal key

  // maintains the max endpoint of the node rooted at x. .
  x.max = max(x.max, r.b); // maintain the max info.
  return x;
}

private K max(K a, K b) {
  if (a == null) return b;
  if (b == null) return a;
  return a.compareTo(b) > 0 ? a : b;
}
```

Chapter 3

Future Exercise