CS2040 Tutorial 7

Week 8, starting 10 Oct 2022

Q1 Other BST Operations

You are given a BST implementation below:

```
class Node {
   int item;
   Node left, right;
   Node(int i, Node l, Node r) { item = i; left = l; right = r; }
}
class BST {
   int numNodes;
   Node root;

   int floor(int key) {} // to implement, may create helper method

   void insert(int key) { root = insert(key, root); }
   private Node insert(int key, Node curr) {
      if (curr == null) { numNodes++; return new Node(key, null, null); }
      if (key == curr.item) return curr; // no insertion
      if (key < curr.item) curr.left = insert(key, curr.left);
      else curr.right = insert(key, curr.right);
      return curr;
   }

   void preOrderPrint(Node root) {
      if (root == null) return;
      System.out.print(root.item + " ");
      preOrderPrint(root.left);
      preOrderPrint(root.right);
   }
   void inOrderPrint(Node root) {
      if (root == null) return;
      inOrderPrint(root.left);
      System.out.print(root.item + " ");
      inOrderPrint(root.right);
   }
   void postOrderPrint(Node root) {
      if (root == null) return;
      postOrderPrint(root.left);
      postOrderPrint(root.right);
      System.out.print(root.item + " ");
   }
```

```
   void print() {
      System.out.print("Size: " + numNodes + "\nPreorder: [ ");
      preOrderPrint(root);
      System.out.print("]\nInorder: [ ");
      inOrderPrint(root);
      System.out.print("]\nPostorder: [ ");
      postOrderPrint(root);
      System.out.print("]\n");
   }
}
```

(a) Write another method `int ceil(int key)` in the BST class that finds the *smallest* element that is more than or equals to `key`, or Integer.MAX_VALUE if none exists (i.e. the ceiling).

(b) Can `ceil(int)` be tweaked to implement `int higher(int key)` that returns the *smallest* element strictly greater than `key`, or Integer.MAX_VALUE if none exists? (i.e. the successor)

What is the time complexity of: one call to `higher(int)` , as well as repeated calls to `higher(currentKey)` , starting with `currentKey` being the smallest key in the tree, till we get Integer.MAX_VALUE?

(c) How does the BST implementation need to be changed to support `Node succ(Node curr)` (note the different parameter from (b)), that works similarly to `higher(int)` in (b) but returns the Node instead of the desired element?

Design and implement `succ(Node)` . Why is such an implementation more efficient?

Q2 Contiguous Strip with Same Colour

Given positive integers **N** and **K**, suppose you have created a computer game where there is a large 1 x **N** strip of land painted black (colour = 0). The leftmost cell is indexed 0 while the rightmost cell is indexed **N**-1. Each cell has a colour value within 0 to **K** (a positive integer which could be >> **N**) inclusive.

Design and implement a solution to perform the following operations, **each** running in O(log **N**) time or better:

> `void paint(int cell, int newColor)`
> paints the cell at the given index with a different new colour
>
> `int findContigLength(int cell)`
> returns the length of the longest contiguous sub-strip of land with the same colour that includes the (valid) cell with the given index

You may perform some initialization in O(1) time.

e.g. **N** = 5, so the 5 cells start off with colours [0, 0, 0, 0, 0]. findContigLength(1) returns 5
Next paint(3, 5) causes the colours to become [0, 0, 0, 5, 0]. findContigLength(1) returns 3
Then paint is called 3 more times, giving colours [0, 5, 5, 5, 5]. findContigLength(1) returns 4
Finally, paint(3, 4) is called, colours is now [0, 5, 5, 4, 5]. findContigLength(1) returns 2

Question 3 (Online Discussion) – Conditional Average

Modify the BST and or Node class in Q1, without changing the time complexity of any operation, such that it is possible to implement **`double`** `findCondAverage(`**`int`** `upperBound)` that returns, in O(h) time, the average of all elements in the BST that are ≤ upperBound.