CS583 – Spring 2016 Stuti Deshpande

Programming Project G Number: G00979218

**Almost-Increasing Subsequences**

**Problem Statement**:

Given an array of length n, our aim is to find the longest, almost increasing subsequence, of length k, where m<=n, in O(nlogk) time. The goal is to verify, empirically, the runtime correctness of improved algorithm.

**Implementation:**

*Data Structure*: Red Black Trees.

*Programming Language*: Java

We run the implementation for two ‘n’ values, two times for n=2 and 1000 times for n=1000. The pseudo-code has been implemented inside the method laisCompute() which is invoked for both the ‘n’ values from the main() method, by instantiating the RedBlackTree class. The input is given by populating the array with random numbers (generated using Random() predefined in java.utils.Random). The primitive operations for RBT such as insert, delete, deleteFix, fixTree, printTree, findNode, rotateLeft, rotateRight has been taken from codebytes.in. I have implemented the other operations such as tree\_successor, tree\_predecessor, find\_tail\_node, treeMaximum, treeMinimum.

The method liasCompute() dynamically computes the p-array and outputs the longest almost increasing subsequence (z-array) as well as returns the value(counter) which is the length of the z-array(its “b” array in code).

Here is the snapshot for two sequences when n=20:

First sequence for n=20

Input

11 16 1 1 11 16 17 12 3 20 20 13 9 5 19 2 4 15 2 14

Output

1 1 11 16 17 20 19

Table 2: p values

0 0 2 2 3 4 5 4 3 6 6 7 8 8 11 3 8 16 3 8

Count:547

Second sequence for n=20

Input

6 20 9 8 19 6 6 5 8 8 3 16 20 7 18 6 8 12 15 15

Output

6 6 6 5 7 6 8 12 15

Table 2: p values

0 0 0 0 2 5 0 7 6 6 10 9 11 6 9 0 13 16 17 17

Count:655

The “count” is the counter variable used to determine the runtime for algorithm. It is incremented for each loop control structure for each iteration.

*Modifications:*

In pseudo-code, the p-i values were calculated before insertion, which always gave output as -1 for all nodes. We need to have the tree (data structure) to determine the predecessor. I modified the pseudo code in such a way that for each node to be inserted in the tree, the insertion of new node is performed before fixing the tree (not performing rotations), the predecessor node is retrieved and then the node is deleted. In this way, the p-I values are calculated properly.

*Testing:*

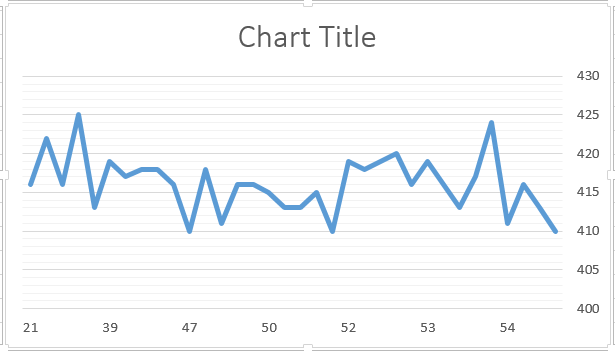
1. The code gives the run for two sequences for n=2 along with Table 2: p-i values and z-array along with counter variable value. For n=1000, it calculates average and counter variable for each 1000 iterations. Average is calculated as Average=Counter/Constant. I have taken constant as 100.
2. I have compiled my java file as javac RedBlackTree.java on Linux machine.
3. Make it run with java RedBlackTree and press Enter. It will give the desired output.

*Outcome:*

Next, I plotted a graph (for n=1000) to analyze the run time by plotting “Average” on Y-axis and “k” on X-axis where:

K: length of the longest almost increasing subsequence for each iteration

Average: Counter/Constant (here, Constant assumed to be 100)



*Discussions:*

As we have used counters to determine the runtime of the algorithm, we can say that

nlogk = Counter/Constant

And hence, we plot the Average (counter/ Constant) (y-axis) as a function of k (x-axis) and analyze the run time graphically. Since, the red-black tree operations such as insert, delete, fixTree, etc takes the logarithmic time the depth of the tree, all the tree operations methods run for logk time (k is the length of the output array i.e. the length of the longest almost increasing subsequence). Since, the length of the input array is n, the code runs for nlogk time. Hence, the run time for our algorithm, found empirically as nlogk time.

*References:*

1. The paper “The “longest almost-increasing subsequence,” by Elmasry (Info Proc Letters, vol 110, 2010, pp. 655-658)
2. Introduction to Algorithms, 3rd Edition, Textbook by *Thomas H. Cormen Charles E. Leiserson Ronald L. Rivest Clifford Stein.*
3. Codebytes for Red-Black tree operations, http://www.codebytes.in/2014/10/red-black-tree-java-implementation.html