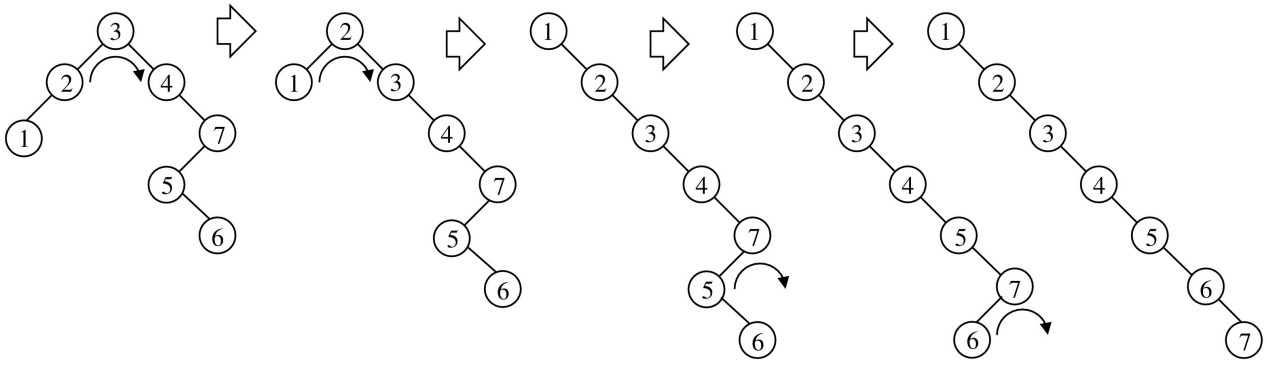
CS29003 ALGORITHMS LABORATORY
Assignment No: 2
Last Date of Submission: 05–August–2015

A Red-Black tree dynamically balances the height during each insertion or deletion. Here, we take a different approach. We first let the tree grow following the naïve BST insertion algorithm, without making any attempt of height balancing. After all items are inserted in the tree, we restore balance in the tree. The balance-restoring algorithm should run in O(n) time (n is the number of nodes in the tree), and use only O(1) extra space.

Part 1: Write an *insert*() function for standard BST insertion. Also write the functions *preorder*(), *inorder*() and *height*() with the obvious meanings. This part is quite common to what you did in Assignment 1.

Part 2: Write a function to implement left rotation at any node p in the tree. The function *leftRotate*() takes p as its only parameter, and returns the new root of the rotated subtree. Likewise, write a function *rightRotate*() to perform right rotation at any node p in the tree and return the new root of the rotated subtree.

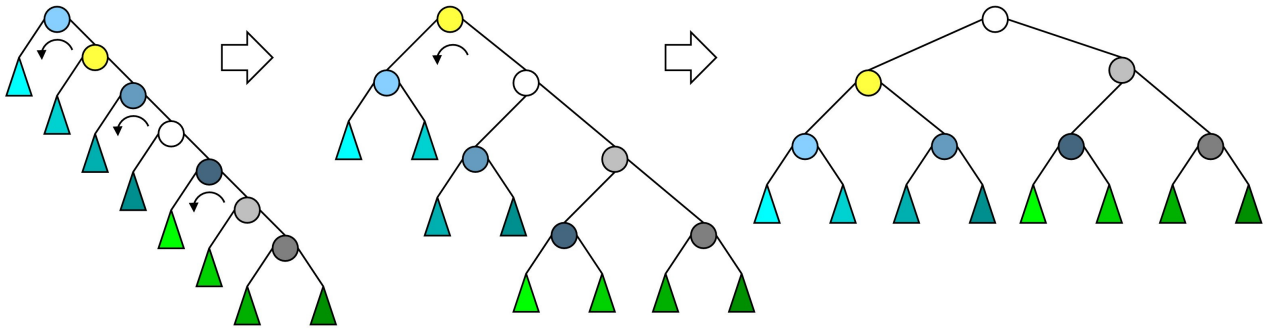
Part 3: Write a function *makeSkew*() to convert the tree constructed in Part 1 to a linked list followed by the right links. A sequence of right rotations achieves this as illustrated in the following figure. Start at the root. As long as the left subtree is not empty, make right rotations. Then, move down the tree (follow right links). Locate the next node where right rotations are again needed. Do the rotations. Move down. Continue until you hit the NULL pointer. The function *makeSkew*() takes a pointer to the root of the tree as the only parameter, and returns the (new) root pointer.



Part 4: This part takes a fully right-skew BST. It makes a sequence of left rotations to convert the tree to an (almost) *complete* binary tree. A sketch of the algorithm to be implemented in the function *rebalance*() follows. This function takes a pointer to the root of the skew tree as the only parameter, and returns the (new) root pointer.

Let n be the number of nodes in the tree, and $l = \text{floor}(\log_2(n+1))$.
If $n+1$ is a power of two, set $c = 2^{l-1}$ else set $c = n + 1 - 2^l$. /* c is the count of leaves at the deepest level */
Start at the root, and follow the right links.
During the traversal, make a left rotation at every alternate node (starting at the root) for a total of c times.
Set $n = n - c$. /* Here, $n = 2^l - 1$. This is the total count of nodes at all levels except the last. */
While ($n > 1$) { /* Loop to position nodes at levels other than the last (from bottom to top) */
    Set $n = n / 2$.
    Start at the root, and follow the right links.
    During the traversal, make a left rotation at every alternate node (starting at the root) for a total of n times.
}

The following figure demonstrates the last two iterations of the above while loop.



Part 5: Write a *main*() function to do the following:

- Read n (the number of insertions) and integers a_1, a_2, …, a_n from the user. The integers a_1 through a_n are inserted one by one in the tree using the *insert*() function of Part 1. After all insertions are made, print the preoder and inorder listings of the tree and also the height of the tree.

- Call the *makeSkew*() function of Part 3. After the function returns, print the preoder and inorder listings of the tree and also the height of the tree.

- Finally, call the *rebalance*() function of Part 4. After the function returns, print the preoder and inorder listings of the tree and also the height of the tree.

Submit a single C/C++ source file solving all the parts.

Sample Output

Two sample runs of the algorithm are shown below.

```
n = 15

+++ Insert      : 7 14 27 77 13 10 86 70 80 69 61 63 67 11 85

+++ Initial BST created:
    Preorder    : 7 14 13 10 11 27 77 70 69 61 63 67 86 80 85
    Inorder     : 7 10 11 13 14 27 61 63 67 69 70 77 80 85 86
    Height      : 8

+++ The tree is now fully right-skew:
    Preorder    : 7 10 11 13 14 27 61 63 67 69 70 77 80 85 86
    Inorder     : 7 10 11 13 14 27 61 63 67 69 70 77 80 85 86
    Height      : 14

+++ Balance restored in the tree:
    Preorder    : 63 13 10 7 11 27 14 61 77 69 67 70 85 80 86
    Inorder     : 7 10 11 13 14 27 61 63 67 69 70 77 80 85 86
    Height      : 3


n = 16

+++ Insert      : 91 74 75 39 6 64 24 89 18 57 17 66 54 63 1 13

+++ Initial BST created:
    Preorder    : 91 74 39 6 1 24 18 17 13 64 57 54 63 66 75 89
    Inorder     : 1 6 13 17 18 24 39 54 57 63 64 66 74 75 89 91
    Height      : 7

+++ The tree is now fully right-skew:
    Preorder    : 1 6 13 17 18 24 39 54 57 63 64 66 74 75 89 91
    Inorder     : 1 6 13 17 18 24 39 54 57 63 64 66 74 75 89 91
    Height      : 15

+++ Balance restored in the tree:
    Preorder    : 57 18 13 6 1 17 39 24 54 74 64 63 66 89 75 91
    Inorder     : 1 6 13 17 18 24 39 54 57 63 64 66 74 75 89 91
    Height      : 4
```