## CS29003 Algorithms Laboratory

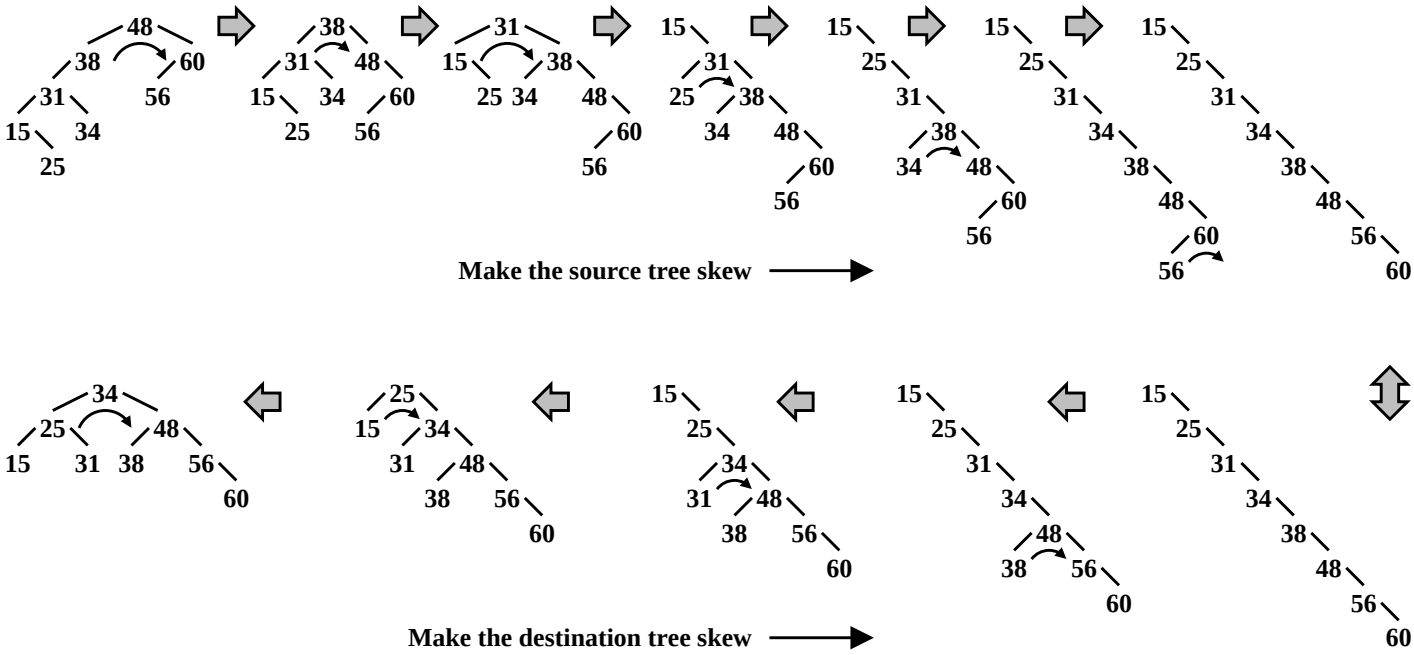
### Assignment No: 7

Last date of submission: 12–March–2019

---

## Binary Search Trees

You are given two *node-disjoint* binary search trees $S$ and $T$. Each of these trees consists of $n$ nodes. The two trees are structurally different, but store exactly the same set of keys. Let us call $S$ the *source tree*, and $T$ the *target tree*. Your task is to convert $S$ to (a tree structurally identical to) $T$, by making a sequence of rotation operations only. Your algorithm should run in $O(n)$ time. In particular, you are allowed to carry out at most $O(n)$ rotation operations. Moreover, it is forbidden to add any extra space to the tree nodes. Using an additional $O(n)$ space, totally external to the trees, is however granted.




The above figure illustrates an algorithm for solving this problem. Any BST can be converted to a totally right-skew tree using a sequence of right-rotate operations. This process is illustrated for both the source and the target trees in the figure. After this skewing operation on these two trees, they become essentially the same tree with different sets of nodes. Let $H_S$ (not needed actually) and $H_T$ (to be needed) denote the histories of rotation operations performed on the two trees in order to right-skew them.

The rotation operation is reversible. This means that if we reverse the history $H_T$ on the right-skew target tree, we get back the original $T$. Likewise, reversing the history $H_S$ on the right-skew source tree gives back the original $S$. We however want $S$ to be converted to a tree structurally identical to $T$ (not $S$), so we need to apply a reversal of the history $H_T$ on $S$. But $S$ and $T$ are composed of different nodes, so $H_T$ is not directly applicable to $S$. After the skewing operations, the trees have structurally become the same. That is, we can work out the correspondence of the nodes of $T$ with the nodes of $S$. If we apply this correspondence to the history $H_T$, we get a sequence $H'_S$ of rotations pertinent to the source tree. If we reverse the derived history $H'_S$ on the right-skew source tree, it gets converted to a tree structurally identical to the original target tree $T$.

### Part 1: Construct the input trees using the given black box

A black box BB7 is provided for the construction of the input trees $S$ and $T$. For using this black box, download the header file `BB7.h` and the appropriate object file `BB7.o`, to your working directory. Include the following line after your usual include directives.

```
#include "BB7.h"
```

The header file defines the BST data type as follows. Notice that there is no parent pointer in a BST node.

```
typedef struct _node {
    int key;
    struct _node *L;
    struct _node *R;
} BSTnode;

typedef BSTnode *BST;
```

Let $n$ be the number of nodes in $S$ or $T$ (to be input by the user). At the beginning of your `main()` function, include the following lines.

```
BST S, T;

registerme(n);        /* Mandatory in this assignment */
S = getsourcetree();  /* The source tree */
T = gettargettree();  /* The target tree */
```

You are now ready to start the actual assignment. Compile your code as:

```
gcc/g++ myprog.c/myprog.cpp BB7.o
```

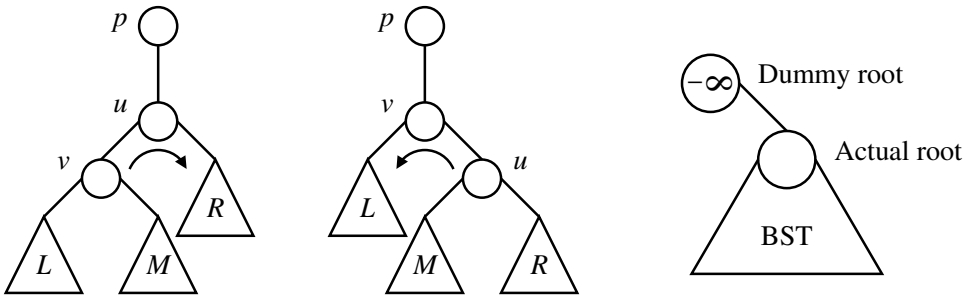
### Part 2: BST functions

Write a function `preorder()` to print the preorder listing of the keys of a BST. Unlike general binary trees, the tree structure of a BST is uniquely identified by its preorder listing. So this function helps you to verify the correctness of your BSTs. If you find useful, you may additionally write an `inorder()` function to print BST keys in sorted order.

The main primitives to be used in this assignment are rotations. Write two functions with the following prototypes to perform left and right rotations at specified nodes.

```
BST lrotate ( BST );
BST rrotate ( BST );
```

Using the notations of the following figure, the call `rrotate(u)` should return `v`, whereas `lrotate(v)` should return `u`. Here, $u, v, p$ are to be taken as BST node pointers.




Each rotation changes the root of the subtree at the position of rotation (the root changes from $u$ to $v$ for right rotation, and from $v$ to $u$ for left rotation, according to the above figure). Let $p$ be the parent of the pre-rotation root of the subtree. In addition to adjusting a few pointers of the subtree, one needs to change the appropriate child pointer of the parent node $p$. This, in turn, implies that you must know the pointer $p$ before you make the rotation. In this assignment, a right or left rotation is applied always to the right child of $p$. Noting that you do not have a provision of parent pointers, you should make the calls like this.

```
p -> R = rrotate(p -> R);
p -> R = lrotate(p -> R);
```

The root of the entire tree does not have a parent. This asymmetry can be handled by adding a dummy root node which stores the key $-\infty$, and which has the actual root as its only child (the right child). The black box does not add the dummy nodes to $S$ and $T$. It is your responsibility to augment the returned trees with the extra nodes. We need the dummy roots in Part 3 (rotations) and Part 4 (finding node correspondences).

**Part 3: Make the trees right skew**

Write a function `rightskew` to covert a BST to a right-skew BST (on the same keys). Initialize $p$ to the dummy root, and $q$ to its right child (the actual root). So long as the left subtree of $q$ is not empty, apply right rotations at $q$. Then, advance both $p$ and $q$ by one step (follow their right-child pointers), and repeat.

We need to remember the history of rotations for the target tree. If a right rotation is made at $q$, its parent $p$ is appended to the history. For the source tree in the figure on Page 1, the rotations are made at the nodes $48, 38, 31, 31, 38, 60$. This list is not sorted with respect to the key values. However, the history should consist of the sequence of the parent pointers, that is, the pointers to the nodes $-\infty, -\infty, -\infty, 15, 31, 48$. This is sorted with respect to the key values. For the target tree of the figure, the history should store the sequence of pointers to the nodes storing $-\infty, -\infty, 25, 34$.

**Part 4: Find the node correspondences**

Write a function `findcorr(S,T,HT)` to convert the history $H_T$ of rotations applied to $T$ in Part 3 to the history $H'_S$ applicable to $S$. Since $S, T, H_T$ are sorted, this can be done in $O(n)$ time by making a single pass through the two trees $S, T$ (now sorted linked lists with respect to the right-child pointers) and $H_T$.

**Part 5: Reverse the skewing process**

Write a function `unskew` that, given a history (and its size), performs the relevant left-rotate operations in the reverse sequence as stored in the history (that is, from end to beginning).

**The *main*() function**

- Read $n$ (the count of nodes in $S$ or $T$) from the user.
- Get two random BSTs $S, T$ on the same key values by making appropriate BB7 calls (see Part 1). Print the trees (the preorder listings of their keys).
- Make `rightskew` calls to convert $S$ and $T$ to right-skew trees. During the skewing of $T$, the history should also be stored in $H_T$ (an array of pointers of type `BST`). Print the right-skew trees.
- Obtain the derived history $H'_S$ by calling `findcorr`.
- Convert the right-skew $T$ to the original $T$ by calling `unskew` with respect to $H_T$. Also, convert the right-skew $S$ to a BST structurally identical to the original $T$, by applying `unskew` with respect to the derived history $H'_S$. Print the final trees.

---

Submit a single C/C++ source file. Do not use global/static variables.

## Sample output

```
n = 50
*** You are now registered

+++ Initial trees
    Source : 165 137 114 106 128 147 222 212 168 196 181 211 271 269 245 229
             228 243 257 287 275 351 290 321 306 339 337 344 350 519 443 369
             363 357 399 382 411 413 429 492 455 467 478 490 508 551 549 534
             575 560
    Target : 168 137 106 128 114 165 147 413 290 243 222 181 211 196 212 228
             229 287 275 269 245 257 271 339 337 306 321 357 351 344 350 382
             363 369 411 399 519 492 478 467 443 429 455 490 508 534 575 551
             549 560

+++ Right-skewing the trees
    Source : 106 114 128 137 147 165 168 181 196 211 212 222 228 229 243 245
             257 269 271 275 287 290 306 321 337 339 344 350 351 357 363 369
             382 399 411 413 429 443 455 467 478 490 492 508 519 534 549 551
             560 575
             Number of rotations = 42
    Target : 106 114 128 137 147 165 168 181 196 211 212 222 228 229 243 245
             257 269 271 275 287 290 306 321 337 339 344 350 351 357 363 369
             382 399 411 413 429 443 455 467 478 490 492 508 519 534 549 551
             560 575
             Number of rotations = 45

+++ Finding node correspondence

+++ Reversing the skewing process
    Source : 168 137 106 128 114 165 147 413 290 243 222 181 211 196 212 228
             229 287 275 269 245 257 271 339 337 306 321 357 351 344 350 382
             363 369 411 399 519 492 478 467 443 429 455 490 508 534 575 551
             549 560
    Target : 168 137 106 128 114 165 147 413 290 243 222 181 211 196 212 228
             229 287 275 269 245 257 271 339 337 306 321 357 351 344 350 382
             363 369 411 399 519 492 478 467 443 429 455 490 508 534 575 551
             549 560
```