Assignment 7: A Binary Search Tree Variant

Write functions in C to perform i) insertions and ii) deletions in a slightly different way in a BST as explained below. For printing, follow the convention in the previous assignment. As before, the keys will be positive integers. The struct will be necessarily have a) the key value b) pointer to the left child, c) pointer to the right child and d) pointer to parent. Additionally, you will maintain the size of the tree rooted at any given node. More later in the examples.

1. There are two ways of insertion into the tree: a) recursively insert into the tree in the usual way that you did in the previous assignment and b) *insert at root*. When there is an insertion to be done in a tree of size n, perform step a) with probability $\frac{n-1}{n}$ and b) with probability $\frac{1}{n}$. Note that since this is a recursive step, you have to make this choice at every level.

   insert at root works as follows as shown in the picture below:

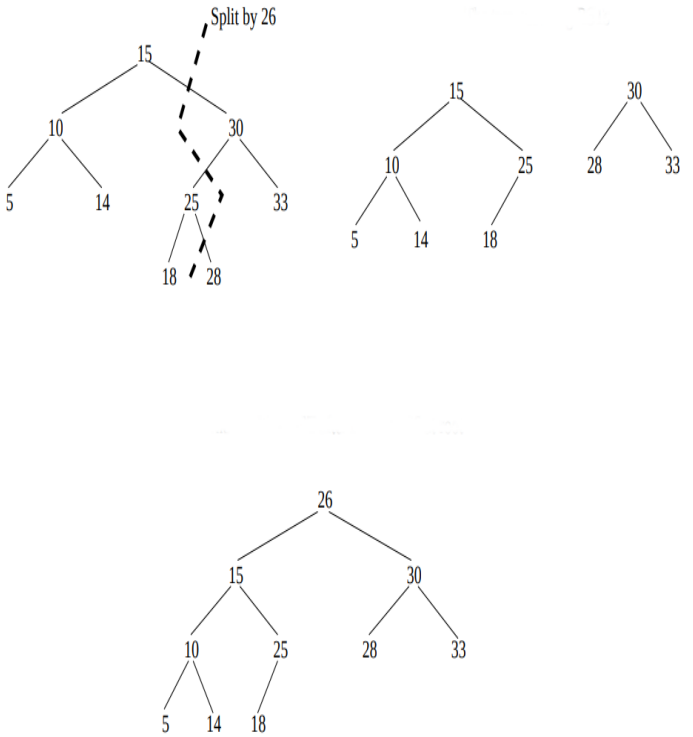

Figure 1: 26 is being inserted into the BST (top left); the original BST is split into two BSTs (top right); 26 is made the root, and stitched again (bottom).

2. Delete is similar to the usual delete operation in a BST, but for a minor twist. Searching for the node to be deleted works as in the usual case. When you find the node to be deleted, do the following (showing only the relevant part and skipping the part where you search):

```
// relevant part of the function
//BST delete(int x, BST T)

temp = join(T->left, T->right);
free(T);
T = temp;
```

Finally, the delete subroutine returns T, a pointer to the tree. The *join* function behaves is detailed below:

```
bst join(bst L, bst R) {
  int m, n, r, total;

  m = L→size; n = R→size; total = m + n;
  if (total == 0) return □;
  r = random(0, total - 1);
  if (r < m) {            /* with probability m/(m+n) */
      L→right = join(L→right, R);
      return L;
  }
  else {                  /* with probability n/(m+n) */
      R→left = join(L, R→left);
      return R;
  }
}
```

A pictorial representation of what happens during deletion is as follows:

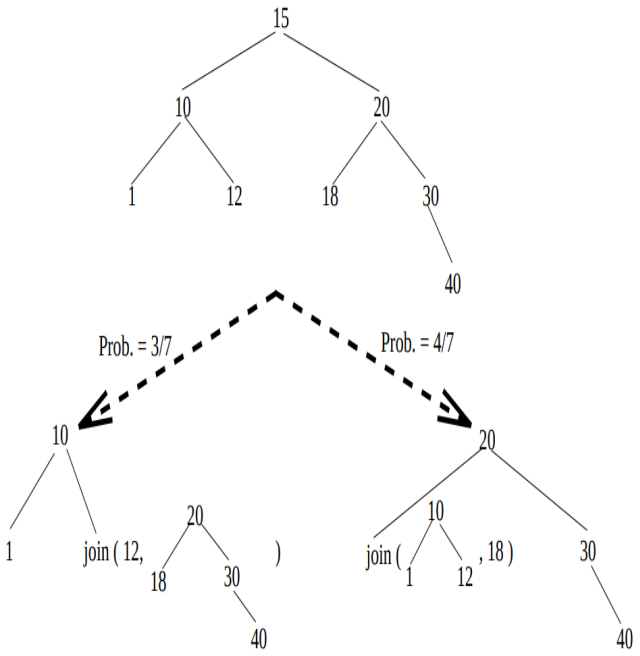

Figure 2: Deleting 15

Some remarks:

(a) Test cases will be similar to the previous assignment.

(b) You would have noticed through your learning about the BST that the tree can get skewed to one side over several insertions. E.g. if you insert 1,2,3,4 and 5 in that order, you will get a tree skewed to the right, results in a $\Theta(n)$ search time. This would be as if you are searching in a linked list, and you don't get any advantage that a BST is supposed to offer. Folks in the past have designed workarounds that mitigate this problem. In the coming courses, you will study about many approaches towards restoring some sense of *balance* to avoid such a skew. For those who wish to have a sneak peek, read up on AVL trees, Red Black trees and Splay Trees to get an idea.

(c) What you will do in this assignment is unlikely to be discussed in a usual course, but this is another way of achieving this balance. Convince yourself of this.

(d) For 10 bonus marks (thanks to Supreet for this!), do the following:

   i. Implement a generic BST library that takes an arbitrary data type (int, strings, even other structs) as key, depending on the

need of the user. You could use a void* pointer for the datatype of key in the tree node. Also you'll need to modify the library functions to take in an additional argument - a function pointer. This pointer points to a comparison function, that compares two objects of your custom data type. We need to do this since a BST relies on comparison for its operations. The user of your library will have to write these comparison functions for the data type that she is using.

ii. Export the above written code as a shared library (.so/.dll file). You'll also need to export a header (.h) file that declares the publicly available functions. Hide other helper functions that you might have used using the static keyword in front of the function declaration. Ensure no memory leaks! Also read up about static vs dynamically linked libraries.