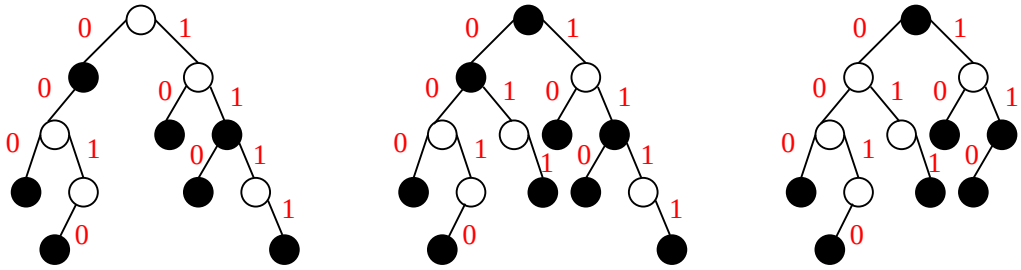
CS29002 Algorithms Laboratory

Assignment No: 2

Last date of submission: 03–August–2016

Let T be a binary tree with each node storing a flag called EOS (see below why) and two child pointers L and R. Some nodes in T, including all the leaves, have the EOS flag set. Imagine that every left-child link is labeled by 0, and every right-child link by 1. We say that T stores a binary string $\alpha \in \{0,1\}^*$ if there is a node v with the EOS flag set such that the unique path from the root to v is labeled by the symbols of α. Here, EOS stands for the end of a string stored in T.

As an example, consider the binary tree in Part (a) of the following figure. Nodes for which the EOS flag is set are shown as solid black circles. All the binary strings stored in this tree are 0, 000, 0010, 10, 11, 110, and 1111. The tree does not store 001 (the path from the root labeled by this string ends at a node in which the EOS flag is not set), and 010 (there is no path from the root that is labeled by this string).



(a) (b) (c)

The binary strings stored in T can be identified with positive integers in the following way. Let $a > 0$ be a positive integer having the l-bit binary representation $a = (1a_{l-2}a_{l-3}\ldots a_1a_0)_2$. We call l the length of a, and denote this as $|a| = l$. If T stores the $(l-1)$-bit string $a_{l-2}a_{l-3}\ldots a_1a_0$, we say that T stores the integer a. For all positive integers, the most significant bit (msb) is 1, so we discard the msb for the purpose of storage. Part (a) of the above figure stores the integers $(10)_2 = 2$, $(1000)_2 = 8$, $(10010)_2 = 18$, $(110)_2 = 6$, $(111)_2 = 7$, $(1110)_2 = 14$, and $(11111)_2 = 31$.

Define a data type to store a node in T. Each node should contain only the EOS flag, and two child links. The integer value corresponding to a node is not explicitly stored in the node. Moreover, there must not be any parent pointer in a node. By "passing T to a function," we imply passing a pointer to the root node of T.

Part 1: Write a function *printTree*(T) to print all the binary strings stored in T. The printing must be in the lexicographic order of the stored strings. For the tree of Part (c), the output should be ε (the empty string), 000, 0010, 011, 10, 11, 110. Your function should run in $O(n)$ time, where n is the number of nodes in T.

Part 2: Write a function *insert*(T, a) to insert an integer a (or equivalently the binary representation of a without the msb) to a binary tree T. The function should return a pointer to the root of the tree after the insertion. The tree in Part (b) of the figure is obtained from the tree of Part (a) after inserting the integers $1 = (1)_2$ and $11 = (1011)_2$ (equivalently, the empty string and the string 011). Your function should run in $O(|a|)$ time for inserting a to any binary tree T (irrespective of the number of nodes already present in T).

Part 3: Write a function *delete*(T, a) to delete an integer a (or equivalently the binary representation of a without the msb) from a binary tree T. The function should return a pointer to the root of the tree after the deletion. If a is not stored in T, this deletion does not change T. Otherwise, the EOS flag of the node where the binary representation of a ends should be reset to 0. If this happens at a leaf node, the node must be deleted. This deletion may create a leaf node with the EOS flag not set. As long as this happens, the exposed leaf nodes must be deleted. Part (c) of the above figure is obtained from Part (b) after deleting the integers $5 = (101)_2$, $10 = (1010)_2$, $2 = (10)_2$, and $31 = (11111)_2$. This is equivalent to deleting the strings 01, 010 (the tree did not store these strings), 0 (only the EOS flag is reset), and 1111 (two nodes 1111 and 111 are deleted). Your function should run in $O(|a|)$ time for deleting a from any binary tree T.

Part 4: Write a function *printInts*(T) to print all the integers stored in T. The printing must be in the ascending order of the stored integers. For the tree of Part (c), the output should be 1, 6, 7, 8, 11, 14, 18. Your function should run in $O(n)$ time, where n is the number of nodes in T.

The *main*() function: Do the following.

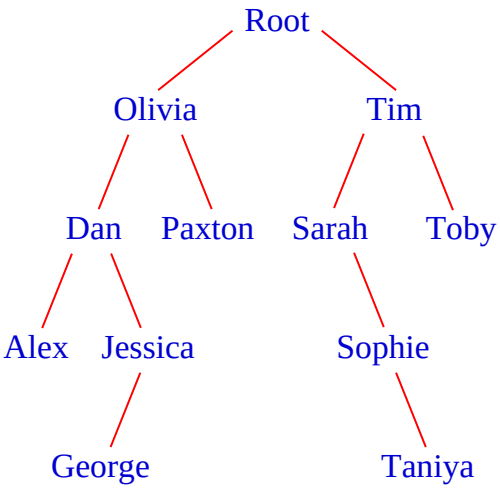
- Create an initially empty binary tree T.
- Read *nins* from the user. The user then enter *nins* positive integers. Insert these integers one by one to T by calling the function of Part 2. After all the *nins* insertions, print the strings stored in T by calling the function *printTree*(T) of Part 1.
- Read *ndel* from the user. The user then enter *ndel* positive integers. Delete these integers one by one from T by calling the function of Part 3. After all the *ndel* deletions, print the strings stored in T by calling *printTree*(T).
- Reset T to an empty tree.
- Read *nins* positive integers from the user, and insert these integers one by one to T by calling the function of Part 2. After each insertion, print the integers stored in T by calling the function *printInts*(T) of Part 4.
- Read *ndel* positive integers from the user, and delete these integers one by one from T by calling the function of Part 3. After each deletion, print the integers stored in T by calling *printInts*(T).

Sample output

```
nins = 10

+++ Insert: 21 98 30 58 36 75 30 7 74 37

+++ After insertion:
     00100     00101    001010    001011      0101    100010        11     11010
      1110

ndel = 10

+++ Delete: 37 58 74 21 75 21 7 7 36 30

+++ After deletion:
    100010

+++ Old tree destroyed

+++ Insert( 2):  2
+++ Insert(92):  2 92
+++ Insert(66):  2 66 92
+++ Insert(21):  2 21 66 92
+++ Insert(92):  2 21 66 92
+++ Insert(92):  2 21 66 92
+++ Insert(86):  2 21 66 86 92
+++ Insert(93):  2 21 66 86 92 93
+++ Insert(27):  2 21 27 66 86 92 93
+++ Insert(18):  2 18 21 27 66 86 92 93

+++ Delete(21):  2 18 27 66 86 92 93
+++ Delete(93):  2 18 27 66 86 92
+++ Delete(86):  2 18 27 66 92
+++ Delete(92):  2 18 27 66
+++ Delete( 2): 18 27 66
+++ Delete(92): 18 27 66
+++ Delete(18): 27 66
+++ Delete(27): 66
+++ Delete(66):
+++ Delete(92):
```

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

Message from Binary Search Tree



Dear Computer Scientists!

I am standing, facing you. So Olivia is my right child, and Tim is my left child. Your sense of direction seems confused. Sorry for any inconvenience.

Best regards,
Root

Reply of Computer Scientists

Dear BST, we appreciate your concern, but you are standing upside down!