CS29003 Algorithms Laboratory

Assignment No: 4

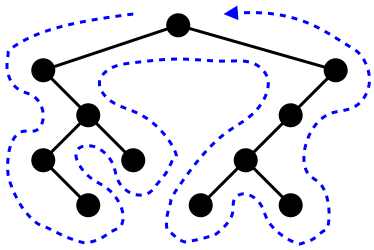
Last date of submission: 06–February–2018

In this exercise, we deal with binary trees. We are interested only in the structural construction of binary trees, so neither store nor use any keys in the nodes. We assume that each node has three pointers: the child pointers L and R, and the parent pointer P. Use standard pointer-based representation.

```
typedef struct _node {
   struct _node *L; /* Left child pointer */
   struct _node *R; /* Right child pointer */
   struct _node *P; /* Parent pointer */
} treenode;
```

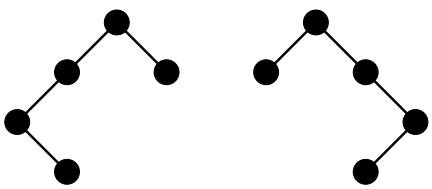
For the moment, imagine that each left child pointer is labeled by 0, each right child pointer is labeled by 1, and each parent pointer is labeled by 2. Let us make a recursive traversal of the tree. The first recursive call is on the left child (if it exists), and the second recursive call is on the right child (if it exists). When a recursive call returns, imagine that you are following the parent pointer. If you print the labels on the pointers in the sequence you encounter them in the traversal, you get a string over $\{0,1,2\}$. Let us call this string the *ternary encoding* of the tree. This string is of length $2(n-1)$, and uniquely captures the structure of the tree (where n is the number of nodes in the tree). Given a valid ternary encoding of length $2(n-1)$, one can reconstruct a unique binary tree consisting of n nodes.

As an example, consider the traversal illustrated in the following tree. The string obtained by this traversal is 01012212221000212222.



We can view the tree as an undirected graph. In this case, the label on a parent pointer is the same as the label on the corresponding child pointer. That is, the label on a parent pointer from a left child is 0, whereas the label on a parent pointer from a right child is 1. Now, the traversal gives you a binary string (a string over $\{0,1\}$), again of length $2(n-1)$. This string is called the *binary encoding* of the tree.

For example, the traversal of the tree in the above example yields the binary string 01011011101000011001. A valid binary encoding however need not uniquely identify a tree. Multiple binary trees can have the same binary encoding. For example, the following two trees have the same binary encoding 00110011.



Part 1: Write a function *printtree* that prints a binary tree in the format given in the sample output. Write another function *destroytree* that recursively frees the memory allocated to the nodes of a binary tree.

Part 2: Write a function *genenc1* that computes and returns the ternary encoding of a binary tree. Write another function *genenc2* that computes and returns the binary encoding of a binary tree.

Part 3: Write a function *buildtree1* that, given a valid ternary encoding, builds and returns the corresponding binary tree. Proceed iteratively. Create a root node, and then build and traverse the rest of the tree starting

from the root node. The symbols in the encoding give you the exact sequence of movements: 0 means *go left*, 1 means *go right*, and 2 means *go up* (to the parent). When you move down the tree, create an appropriate child node, and go to that child. When you move up, the parent already exists, so do not create any new node—just go to the parent.

Part 4: Write a function *buildtree2* that, given a valid binary encoding of a binary tree, builds and returns a binary tree having this encoding. Notice that a valid binary encoding may correspond to multiple binary trees. It suffices to construct any such tree. Use dynamic programming.

The *main*() function

- Read the number n of nodes in your binary trees from the user.
- The user then enters a valid ternary encoding. Call *buildtree1* to generate the unique binary tree having this encoding. Print the tree by calling *printtree*. Also call *genenc1* to obtain the ternary encoding of the constructed tree. Print this string. This should exactly match the string input by the user.
- Destroy the tree constructed above by calling *destroytree*.
- The user then enters a valid binary encoding. Call *buildtree2* to generate one binary tree having this encoding. Call *printtree* to print the constructed tree. Also print the string returned by *genenc2* on the constructed tree. This string must exactly match the binary string input by the user.

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

Sample output

```
n = 8

********** ENCODING 1 **********

+++ Encoding 1 of tree: 00210021222212

+++ The binary tree:
    X
    +---X
        +---X
            +---|
            +---|
        +---X
            +---X
                +---X
                    +---|
                    +---|
                +---X
                    +---|
                    +---|
            +---|
    +---X
        +---|
        +---|

+++ Original encoding : 00210021222212
+++ Final encoding    : 00210021222212

********** ENCODING 2 **********

+++ Encoding 2 of tree: 01101000010011

+++ The binary tree:
    X
    +---X
        +---|
        +---X
            +---|
            +---|
    +---X
        +---X
            +---X
                +---|
                +---|
            +---|
        +---X
            +---X
                +---|
                +---|
            +---|

+++ Original encoding : 01101000010011
+++ Final encoding    : 01101000010011
```