CS29003 Algorithms Laboratory

Assignment No: 4

Last date of submission: 17–August–2017

Let T be a binary tree with each node storing an integer *key* value and two child pointers L and R. No other information or pointer is stored in any node.

```
typedef struct _tnode {
   int key;
   struct _tnode *L, *R;
} tnode;

typedef tnode *bintree;
```

Many computational problems associated with T can be solved by recursive functions of the following type.

```
<return_type> f ( bintree T, ... )
{
   Declare local variables
   if (T == NULL) { Run Block 0 and return }
   Block 1
   f(T -> L, ...); /* Store the return value if necessary */
   Block 2
   f(T -> R, ...); /* Store the return value if necessary */
   Block 3
   Return a value if necessary
}
```

If each of the blocks takes $O(1)$ time, the function f on a binary tree T of n nodes runs in $O(n)$ time. Let $h = \mathrm{ht}(T)$ be the height of T. Then, the recursion depth of f is h, which implies $O(h)$ space usage (assuming that there are $O(1)$ local variables declared in the body of f). In this assignment, you implement some such $O(n)$-time $O(h)$-space functions.

Part 1: Prepare T

As an example, the following function recursively builds a tree T on n nodes.

```
bintree buildtree ( int n )
{
   bintree T;
   int n1, n2;

   if (n <= 0) return NULL;
   T = (bintree)malloc(sizeof(tnode));
   T -> key = 100 + rand() % 900;
   if (n == 1) {
      T -> L = T -> R = NULL;
   } else {
      n1 = rand() % n; n2 = (n - 1) - n1;
      T -> L = buildtree(n1);
      T -> R = buildtree(n2);
   }
   return T;
}
```

Modify the above function by replacing the *rand*() function calls by user inputs. Also, note that the size n of the final tree T (to be returned by the call of *buildtree*() in *main*()) is supplied by the user in *main*().

Part 2: Print T

Write a function *printtree*() to print a binary tree T in the format specified in the sample output. The amount of indentation at the beginning of each line depends on the level of the node being printed in that line. So you need to pass the level of the node as an argument to *printtree*(). By convention, the root is at level 0. The children of a node at level l are at level $l+1$. Therefore, each call should increment the level of a node by one for passing to the recursive calls.

Part 3: Calculate height and update child links

A binary tree T is called *left-tilted* if for any node v in T, we have $\mathrm{ht}(L(v)) \geqslant \mathrm{ht}(R(v))$. The height of a binary tree can be computed by a linear-time recursive function of type as described at the beginning.

Write a function *lefttilt*(T) to left-tilt a binary tree T. The function returns the height of T. Moreover, whenever the function encounters a node v with $\mathrm{ht}(L(v)) < \mathrm{ht}(R(v))$, it swaps the two child links at v.

Part 4: Update keys simultaneously

The i-th ancestor of a node v in T is defined as follows:

$$\mathrm{ancestor}^{(0)}(v) = v,$$

and

$$\mathrm{ancestor}^{(i)}(v) = \mathrm{parent}\left(\mathrm{ancestor}^{(i-1)}(v)\right) \text{ for } i \geqslant 1.$$

If r is the root node, we define

$$\mathrm{parent}(r) = r.$$

Let $h = \mathrm{ht}(T)$. Suppose that all nodes v in T simultaneously perform the following task: it computes $i = key(v)$ rem h, and replaces $key(v)$ by $key\left(\mathrm{ancestor}^{(i)}(v)\right)$. If you have a single running process, this operation must be performed sequentially. Write a function *updatekeys*() that sequentially performs this task. Your function should run in $\mathrm{O}(n)$ time, and may use a total of at most $\mathrm{O}(h)$ additional space.

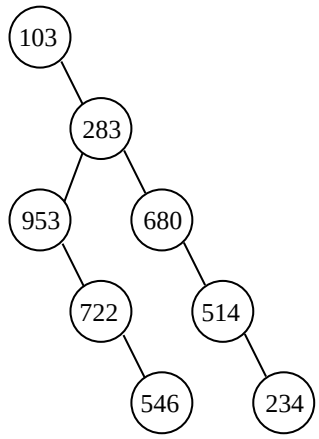
The *main*() function

- Read the number n of nodes in T from the user.
- Prepare a tree on n nodes by the modified *buildtree*() which is based on user inputs.
- Print the tree so prepared by calling *printtree*().
- Call *lefttilt*() to left-tilt T, and obtain its height h. Print the tilted tree (using *printtree*()) and h.
- Call *updatekeys*(). After the call returns, print the updated tree by calling *printtree*().

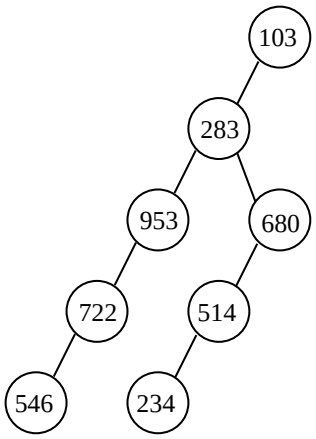
Submit a single C/C++ source file. Do not use global/static variables.

Sample output

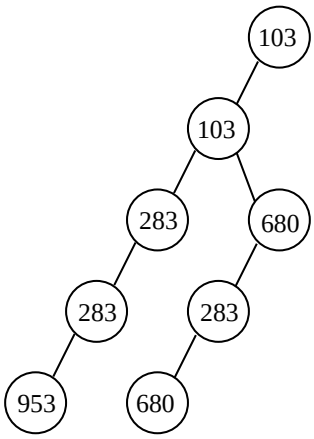
The sample input and output for the example illustrated below are given on the next page.



Initial tree



Tree after left tilting



Tree after key updates

```
    8
    103 0
    283 3
    953 0
    722 0
    546
    680 0
    514 0
    234

+++ Initial tree
    103
     +-- NULL
     +-- 283
          +-- 953
               +-- NULL
               +-- 722
                    +-- NULL
                    +-- 546
                         +-- NULL
                         +-- NULL
          +-- 680
               +-- NULL
               +-- 514
                    +-- NULL
                    +-- 234
                         +-- NULL
                         +-- NULL

+++ Tree after left-tilting
    103
     +-- 283
          +-- 953
               +-- 722
                    +-- 546
                         +-- NULL
                         +-- NULL
                    +-- NULL
               +-- NULL
          +-- 680
               +-- 514
                    +-- 234
                         +-- NULL
                         +-- NULL
                    +-- NULL
               +-- NULL
     +-- NULL
+++ The height of the tree is 4

+++ Tree after key update
    103
     +-- 103
          +-- 283
               +-- 283
                    +-- 953
                         +-- NULL
                         +-- NULL
                    +-- NULL
               +-- NULL
          +-- 680
               +-- 283
                    +-- 680
                         +-- NULL
                         +-- NULL
                    +-- NULL
               +-- NULL
     +-- NULL
```