CSC 301 FALL 2019
ASSIGNMENT 05
RED-BLACK TREE
MAXIMUM POINTS: 100

Group work is appreciated. Please form your group and start working on the stated problem.

You are allowed to use C/C++/Java/Python as your implementation language. Your source code should be well documented. At the beginning of the README file (and in the source code) put your and your partner's name. The README file should contain the following sections:

1. Problem Description
2. Analysis and Description of the major functions/classes
3. Instruction to execute your code
4. Some sample input and corresponding output
5. Discussion of your experience on this assignment

Learning Goal:

The goal is to have a hands-on experience on Red-Black Tree data structure.

Problem Description:

For this experiment, you will to create a red-black tree. The tree contains numbers (integers).

Initialization Phase:

The program will create a tree of 10 nodes. The node values will be generated randomly by using the Linear Feedback Shift Register (LFSR) algorithm (See the description in the next section). Then your program will print the tree using "level-order traversal". Each line will print the node values along with color codes.

For example, for the following tree, the program should print:

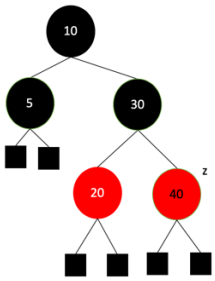

Figure 01

```
(10, black)
(5, black) (30, black)
(20, red) (40, red)
```

Update Phase:

Your program will now allow the user to insert and delete some nodes from the tree. After each insert or delete operation it will print the new tree. For example: your implementation could be:

`ADD 55` ➔ meaning add a new node with value 55

`DEL 13` ➔ meaning delete existing node (if any) 13

The program will not store duplicate entries in the tree. For duplicate entries, or for trying to delete non-existing values, the program will throw proper error messages to the user.

Information Phase:

The last functionality you need to implement is to count "black height of a node". For example: If the user query is `BLKH 10`, your program may return `2` (according to figure 01)

LFSR Algorithm:

In Linear Feedback-Shift Register, the output from a shift register is manipulated and fed back into its input in such a way that the function endlessly cycle through a sequence of patterns.

The most commonly used linear function of single bits is Exclusive-OR (XOR). An LFSR is most often a shift register whose input bit is driven by the XOR of some bits of the overall shift register value. The bit positions that affect the next state are called the taps.

Assume: current value (seed) = 11011101 (binary) = 221 (decimal) and tap bits (linear function) = (5, 3, 2, 0). So, the LFSR algorithm will take the 5^{th}, 3^{rd}, 2^{nd}, and 0^{th} bits from your current seed and apply XOR operations on them. You will use Little-endian bit representation, meaning the right-most bit is the least significant bit and the left-most bit is the most significant bit. Then it shifts the bits of current seed one position to the right and insert the new bit as the most significant bit.

The following figure shows some random number generation starting from seed 221. For your implementation, you can have a default seed value or let the user set the seed.

seed	binary	linear function	new bit	new number
221_{10}	11011101_2	$(0 \oplus (1 \oplus (1 \oplus 1)))$	1	11101110_2
238_{10}	11101110_2	$(1 \oplus (1 \oplus (1 \oplus 0)))$	1	11110111_2
247_{10}	11110111_2	$(1 \oplus (0 \oplus (1 \oplus 1)))$	1	11111011_2
251_{10}	11111011_2	$(1 \oplus (1 \oplus (0 \oplus 1)))$	1	11111101_2
253_{10}	11111101_2	$(1 \oplus (1 \oplus (1 \oplus 1)))$	0	01111110_2
126_{10}	01111110_2	...	...	...

Figure 02: LFSR Computation

For more information, you can take a look at the following Wikipedia article:

https://en.wikipedia.org/wiki/Linear-feedback_shift_register

Points Distribution:

Implementing LFSR algorithm: 15 points
Creating/Initializing a Red-Black tree: 10 points
Print the tree: 15 points
Insert a value: 20 points
Delete a value: 20 points
Print black-height of a node: 10 points
README: 10 points

README should contain information about how to execute the code and running time analysis of each of your major functions.