Laboratory practice No. 5: Binary Trees

Juan S. Cárdenas Rodríguez
Universidad EAFIT
Medellín, Colombia
jscardenar@eafit.edu.co

David Plazas Escudero
Universidad EAFIT
Medellín, Colombia
dplazas@eafit.edu.co

October 17, 2017

1) *CODE FOR DELIVERING ON GITHUB*

The source code can be found in `ParentsTree.java` inside the `codigo` folder; everything is tested in the main method.

2) *ONLINE EXERCISES*

The source code can be found in `Code.java` inside the `codigo` folder.

3) *SIMULATION OF PROYECT PRESENTATION QUESTIONS*

3.a. Parents Tree??????????

You can find binary tree in `ParentsTree.java`, for David's family. Here's the original tree, you can find it implemented in the `.java` file.

```
                    David
                  /       \
                 /         \
                /           \
               /             \
        Claudia              Gustavo
       /        \            /      \
      /          \          /        \
   Ilva        Efrain    Astrid        Jose
                        /      \
                  Leonilde    Epaminondas
```

3.b. Is it possible to implement a better binary tree for parentsTree?

3.c. How does exercise 2.1 work?

This exercise is really straight-forward because, when we write a tree in pre-order, we immediately know that the first integer is the root, the next integers are the left ones and, then the right ones. So, first we insert the first integer which is the root; then, as the tree is a binary search tree (BST) we know that if we insert them in another BST, it will insert them in the right order if and only if we insert them in the order they are given. So, just inserting them as the user passes them solves this problem.

3.d. What's the complexity of exercise 2.1?

```
public class Code {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in); //c1
        System.out.println("Write quit to end the program"); //c2
        String input = sc.next(); //c3
        BinaryTree bt = new BinaryTree(); //c4
        while(!input.equals("quit")) { // c5*n
            int node = Integer.parseInt(input); // c6*n
            bt.insert(node); // c7*n*logn
            input = sc.next(); //c8*n
        }
        bt.posOrder(); // O(n)
    }
}
```

Therefore, exercise 2.1 is $O(k_0 + k_1 n + kn \log n)$. When the sum and product properties of the Big-O notation are applied, exercise 2.1 is $O(n \log n)$.

4) TEST SIMULATION

- **i.**
  - **i.** `altura(raiz.izq)+1`
  - **ii.** `altura(raiz.der)+1`
- **ii.** c) 3
- **iii.**
  - **i.** $1 == 2$
  - **ii.** 0
  - **iii.** `a.izq, suma-a.dato`
  - **iv.** `a.der, suma-a.dato`
- **iv.**
  - **i.** c) $T(n) = 2T(n/2) + C$
  - **ii.** a) $O(n)$
  - **iii.** d)
  - **iv.** a)
- **v.**
  - **i.** $p.data == toInsert$
  - **ii.** $p.data > toInsert$

Professor Mauricio Toro Bermúdez
Phone: (+57)(4)2619500 Ext. 9473. Office: 19 – 627
E-mail: mtorobe@eafit.edu.co