# Laboratory practice 4: Trees

**Santiago Isaza Cadavid**

Universidad EAFIT

Medellín, Colombia

sisazac@eafit.edu.co

**Hamilton Smith Gómez Osorio**

Universidad EAFIT

Medellín, Colombia

hsgomezo@eafit.edu.co

October 15, 2018

## 1) Project questions Simulation

### 1.a. Can a genealogical tree be implemented more efficiently so that the search and insertion can be done in logarithmic time? Or not? Why?

To implement more efficient trees in terms of search and insertion, the information would need to be balanced using some parameter. For example, if we use the names of the people in the family tree, the tree would not be displayed as a family tree. But we can solve this adding to each node its ancestors, but this would involve changing the tree and respective algorithms that we worked on, so we would be sacrificing memory for efficiency.

### 1.b. Explain the implementation of Exercise 2.1

### 1.c. Exercise 2.1's complexity

i.
```
private void posOrderTreeAux(Node node){

    if(node!=null){ //C1

        posOrderTreeAux(node.left); // T(n/2) + C2

        posOrderTreeAux(node.right); // T(n/2) + C3

        System.out.println(node.data); //C4

    }
```

$$T(n) = C1 + C2 + C3 + C4 + T(n/2) + T(n/2)$$
$$T(n) = O(C1 + C2 + C3 + C4 + T(n/2) + T(n/2))$$

UNIVERSIDAD EAFIT
SCHOOL OF ENGINEERING
DEPARTMENT OF SYSTEMS AND INFORMATICS

Page 2 of 4
ST245
Data Structures

$$T(n) = C1 + C2 + C3 + C4 + T(n/2) + T(n/2)$$
$$T(n) = O(C1 + C2 + C3 + C4 + T(n/2) + T(n/2))$$
$$T(n) = O(C1 + C2 + C3 + C4 + 2T(n/2))$$
$$T(n) = O(C * (n - 1) + (C * n)/2)$$
$$T(n) = O((n - 1) + (n/2))$$
$$T(n) = O(n + (n/2))$$
$$T(n) = O(n + n)$$
$$T(n) = O(2n)$$
$$T(n) = O(n)$$

The complexity of this algorithm is O($n$)

### 1.d. Explain what the variables means in the previous exercises

The variable n means the number of nodes entered, in particular the node that is being visited.

## 2) Midterm Simulation

### 2.a. Exercise 1

**2.1.1 Complete line 04**

altura(raiz.left)+1

**2.1.2 Complete line 05**

altura(raiz.right)+1

### 2.b. Exercise 2

c) 3

### 2.c. Exercise 3

**2.3.1 Complete line 03**

false

**2.3.2 Complete line 05**

a.data

### 2.3.3   Complete line 07

a.left, suma-a.data

### 2.3.4   Complete line 08

a.right, suma-a.data

## 2.d.  Exercise 4

### 2.4.1   Which recurrence equation describes the number of instructions executed by the print algorithm in the worst case?

c) $T(n) = 2T(n/2) + C$

### 2.4.2   What is the asymptotic complexity in the worst case-scenario?

a) O(n)

### 2.4.3   What does the algorithm prints?

d) Wilkenson, Joaquina, Eustaquia, Florinda, Eustaquio, Jovín, Sufranio, Piolina, Wilberta, Piolín, Usnavy

### 2.4.4   What modification should be done to the print algorithm so that it throws the following answer for the previous tree?

a)Changing the lines order 03, 04 and 05 to 05, 04, 03

## 2.e.  Exercise 5

### 2.5.1   Complete line 04

p.data==toInsert

### 2.5.2   Complete line 06

toInsert>p.data

## 2.f.  Exercise 6

### 2.6.1   How many simple paths are in the previous tree?

a) 1

### 2.6.2   Complete line 04

return 0

UNIVERSIDAD EAFIT
SCHOOL OF ENGINEERING
DEPARTMENT OF SYSTEMS AND INFORMATICS

Page 4 of 4
ST245
Data Structures

### 2.6.3  Complete line 06

==0

## *2.g.  Exercise 7*

### 2.7.1  Which is the pos-order print?

1. 0, 2, 1, 7, 5, 10, 13, 11, 9, 4

### 2.7.2  How many elements appear in the same position?

2. 2

## *2.h.  Exercise 8*

b) 2

## *2.i.  Exercise 9*

### 2.9.1  What is the output of the in-order path of the previous binary tree?

a) 5, 3, 6, 1, 7, 4, 8, 0, 2

## *2.j.  Exercise 10*

### 2.10.1  Is the previous tree a binary search tree?

b) No

## *3)  Recommended reading*

## *3.a.  Summary*

A binary search tree is a data structure used in the organization of elements in a hierarchical way, in which the elements that are greater than the root (first element) go to the right side of the root and the minor elements go to the left side. In this order, every element at the respective sides of the root has a child node with whom they share the same properties. This structure is useful to save elements because it is done in a balanced way, where the level (tree 's subdivisions) is equal or its difference is not greater than one, we can access, insert, search and remove an element with a complexity of O(log n)