Laboratory practice No. 4: Hash tables and Binary Tree

María Isabel Arango Palacio
Universidad Eafit
Medellín, Colombia
miarangop@eafit.edu.co

Isabella Montoya Henao
Universidad Eafit
Medellín, Colombia
imontoyah@eafit.edu.co

3) Practice for final project defense presentation

3.1 To calculate collisions between bees we chose a tree data structure, called Octree. Octree is an efficient option to calculate collision in a three-dimensional space by recursively subdividing it into eight octants. In this case we used the octants as the quadrants where each bee will be, so the octree will be responsible for locating each bee in a specific quadrant and for analyzing if there is more than one bee in the quadrant and if that happens, this quadrant will be divided into another octans and so on.
As it was said before, this data structure is really efficient mainly due to it has a really good complexity in time, because insertion and search (the functions that we used) are O(n*log n) for the worst case, where n will be the number of nodes.

3.2 We think that it is really difficult to do the family tree with a different implementation because one of the unique ways that the search and insert functions will be long n is making a self-balancing binary search tree.It doesn't work really well in this problem because there are some cases in which the names of either the paternal family or maternal family members are unknown so the tree will be unbalanced and it is not possible to balance it because that could make that the order of the family members change so that tree would not have the order of the tree that the problem mentioned and neither of the classic family tree.

3.3 Description of exercise 2.1 Pos-order in a binary tree

To solve the problem, first we implement some of the Binary class functions that we already had from the workshop, like insert. Then, we made three new functions. The first one to build a tree from an array, the second one to post-order the elements of the tree and the third one and the most important, the function that makes all in one and solves the problem, the function sort. There we receive an array (that is the pre-order traversal), with this array we build the new tree and print it in post-order. Print it in a post-order traversal means that : first we traverse (pass over) the left subtree, then the right subtree and finally the root node. Basically, it just needs the array with the pre-order of the tree and then it works perfectly.

Description of exercise 2.2 Tree Summing

In this exercise, we create a recursive function to determine if the given elements of a tree add up to a specific number. Besides that, we read the input that will be located in a txt file and using the class Counter (of the collections module of python) we saved the elements with the number of times it is repeated. Then we split the input to have each part of it individually. Therewith, we identify and store the integer that determines the objective of the sum (value) and on the other hand the numbers that are the elements of the tree (exp). Finally with the above action, we call the recursive function sending in the value and exp to determine if it accomplishes the sum of the objective or not and depending on that we return yes or not.

3.4 Complexity

3.4.1 Complexity of Pos-Order in a binary tree

$Time\ complexity\ for\ the\ worst\ case$

$O(n * m) + O(m) + O(m) + O(n * m) \quad \Rightarrow Sum\ law$
$O(n * m)\ \ where\ n\ is\ the\ array's\ length\ and\ m\ is\ the\ number\ of\ nodes$
$Due\ to\ n = m,\ the\ time\ complexity\ for\ the\ worst\ case\ will\ be\ O(n^2)$

Why this complexity? Because the function building tree will have to go through the array that was given and keep inserting each element in the binary tree, and as we know the insertion in the binary tree is O(n).

3.4.2 Complexity of Tree summing

$Time\ complexity\ for\ the\ worst\ case$

$O(n) + O(m) + O(1) \quad \Rightarrow Sum\ law$
$O(n + m)\ \ where\ n\ is\ the\ number\ of\ nodes\ in\ the\ binary\ tree\ and\ m\ is\ the\ number\ of\ rows\ in\ the$

Why this complexity? Because the algorithm will have to split the rows in the txt file and then it goes through each row in a recursive way, to find if the sum of a certain combination of the numbers is equal to the target given.

4) Practice for midterms

4.1 4.1.1 **b :** the strings that begin with the same letter collide
4.1.2 **d:** O(1)

4.2 1. It returns the lowest common ancestor to the node 1 and node 2
2. The complexity for the worst case is T(n) = T(n/2)+T(n/2)+C, what means O(n)

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

3. We can improve the algorithm mystery using the condition that if n is less than the data it will be assigned to the left node; else, to the right node, so we will avoid going through the two sides. This could improve the complexity to O(log2 n).

4.3 1. Last line : return True
2. The complexity of areEqual in the worst case is O(n) since n=m so O(n+m) = O(n)

4.4 4.4.1 **c** : T(n)=2T(n/2)+C, que es O(n)
4.4.2 **a** : O(n)
4.4.3 **d** : Wilkenson, Joaquina, Eustaquia, Florinda, Eustaquio, Jovín, Sufranio, Piolina, Wilberta, Piolín, Usnavy
4.4.4 **a:** Change the order of the lines 03, 04 and 05 to 05, 04, 03.

4.5 Line 4 : toInsert == p.data
Line 6 : toInsert > p.data

4.6 4.6.1 **d** : 4
4.6.2 Return 0
4.6.3 if(raiz.hijos.size()**== 0**) ???

4.7 4.7.1 **a :** 0, 2, 1, 7, 5, 10, 13, 11, 9, 4
4.7.2 **b :** 2

4.8 **b** : 2

4.9 **a :** 5, 3, 6, 1, 7, 4, 8, 0, 2

4.10 4.10.1 **b** : 2, 3, 4, 0, 5, 7, 6
4.10.2 **a** : 5
4.10.3 **b** : No

4.11 4.11.1 Line 10 : raiz.id
4.11.2 The complexity in the worst case is **a** : T(n-1) + c that means O(n)

4.12 4.12.1 **i** : A = 1, B = 2, C = 3, D = 4, E = 5, F = 6, G = 7, H = 8, I = 9, J = 10.
4.12.2 **i** : G,D,B,A,C,E,F,I,H,J
4.12.3 **a** : O(n)

5) Recommended reading (optional)

Mapa conceptual

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

6) Teamwork and gradual progress (optional)

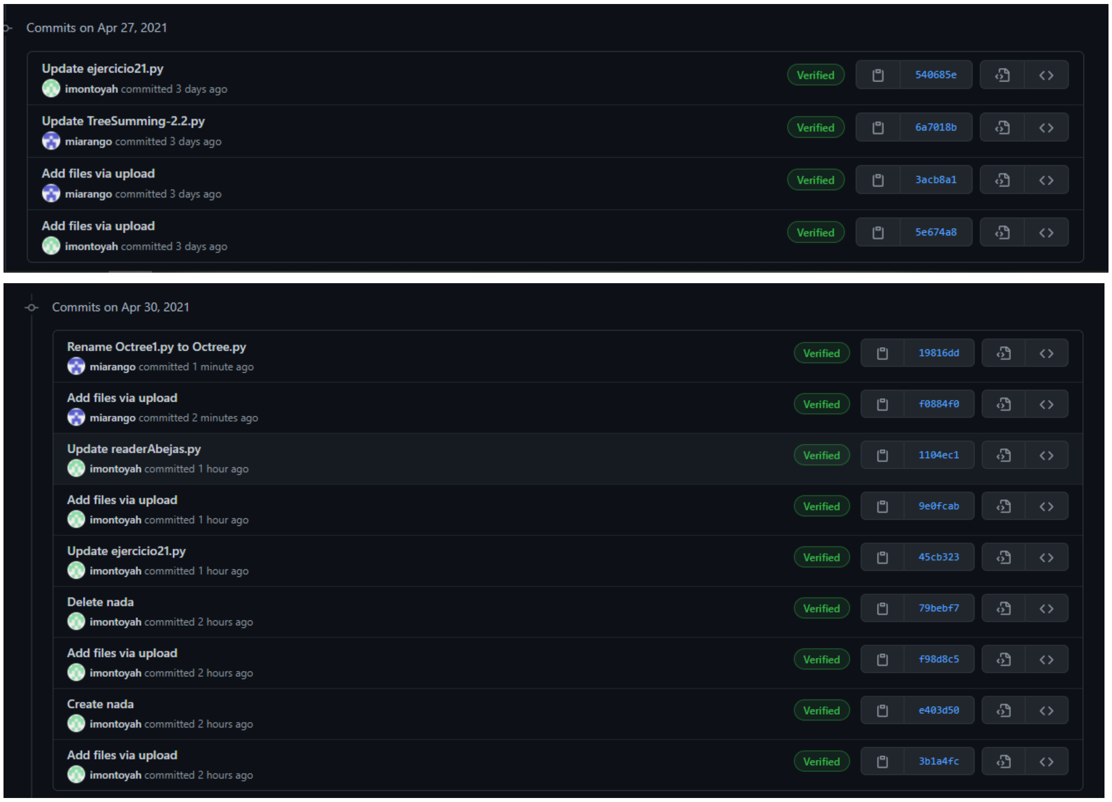
6.1 Meeting minutes

Isabella Montoya Henao	Entrante	48 min 50 s	10:29 a.m.
Isabella Montoya Henao	Entrante	15 min 29 s	23/04 3:56 p.m.
Isabella Montoya Henao	Saliente		23/04 3:18 p.m.
Isabella Montoya Henao	Entrante	47 min 58 s	23/04 11:36 a.m.
Isabella Montoya Henao	Saliente	1 min 21 s	22/04 4:00 p.m.
Isabella Montoya Henao	Saliente	34 min 35 s	22/04 3:25 p.m.
Isabella Montoya Henao	Entrante		22/04 3:24 p.m.
Isabella Montoya Henao	Entrante	1 h 16 min	22/04 10:09 a.m.
Isabella Montoya Henao	Entrante	53 min 28 s	21/04 6:22 p.m.
Isabella Montoya Henao	Entrante	1 h 14 min	20/04 10:48 a.m.
Isabella Montoya Henao	Llamada perdida		19/04 9:09 a.m.
Isabella Montoya Henao	Saliente	23 min 12 s	17/04 4:20 p.m.

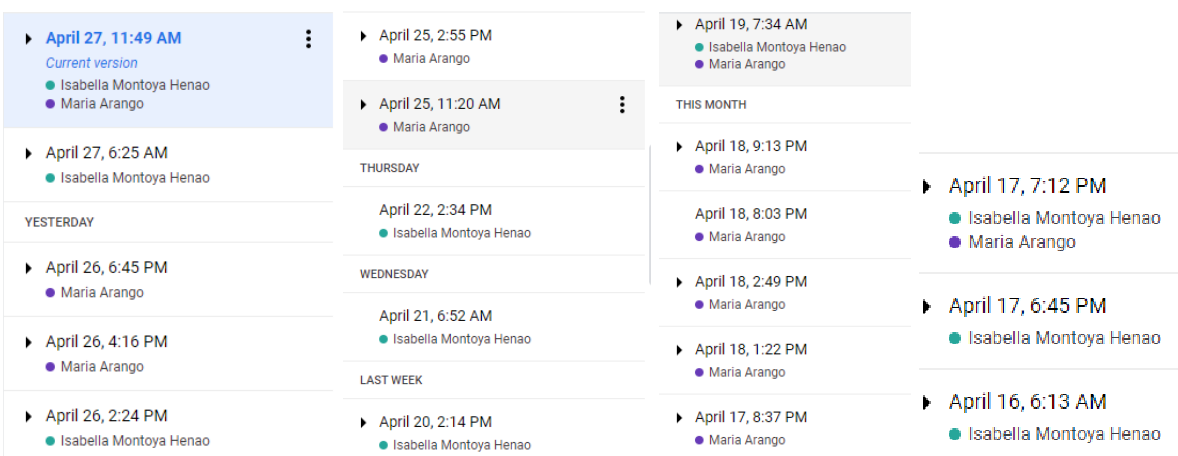
Isabella Montoya Henao	Entrante	1 h 52 min	17/04 2:27 p.m.
Isabella Montoya Henao	Entrante	1 h 28 min	16/04 2:42 p.m.
Isabella Montoya Henao	Entrante	1 h 38 min	11/04 5:14 p.m.
Isabella Montoya Henao	Entrante	18 min 18 s	06/04 5:43 p.m.
Isabella Montoya Henao	Saliente	17 s	06/04 2:09 p.m.
Isabella Montoya Henao	Saliente		06/04 2:08 p.m.
Isabella Montoya Henao	Saliente	1 min 29 s	06/04 2:04 p.m.
Isabella Montoya Henao	Entrante	1 min 3 s	06/04 2:03 p.m.
Isabella Montoya Henao	Entrante	48 min 39 s	05/04 3:21 p.m.
Isabella Montoya Henao	Entrante	56 min 47 s	02/04 5:10 p.m.

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

6.2 History of changes of the code



6.3 History of changes of the report



PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473