Practice 4

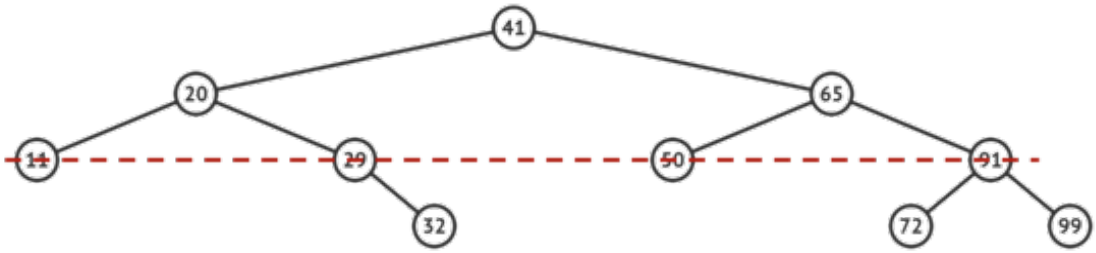
Laura de Paz Carbajo & Paula Samper López
Double degree Mathematics & Computer Science
Programming II, Group 2192
UAM 2018/2019

Report: Practice 4

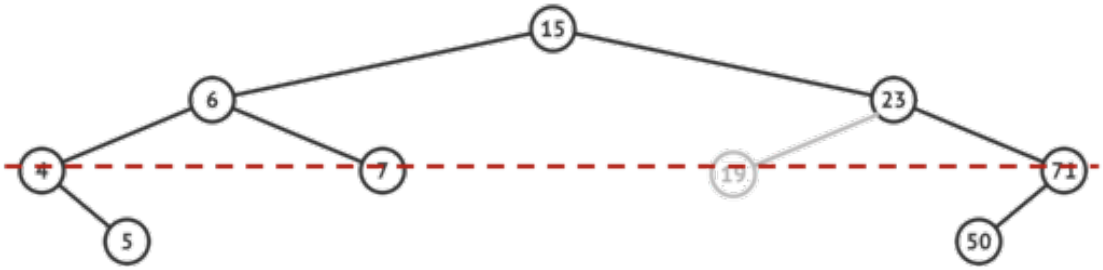
EXERCISE 1

In practice 4 exercise 3, when we use the dict*.dat files in order to create binary trees we have two options. We can either create a balanced tree, in which case we would write a “B”, or a non-balanced tree, “N”.
When we create a balanced tree, the elements are sorted before being inserted into the tree. Thus, we make sure that we are not inserting a node in a new level before having utterly filled the previous level and we get a complete or almost complete tree depending on whether the number of nodes is a power of two.
On the other hand, while creating a non-balanced tree, the elements are inserted just as the program finds them. For instance, we can have an element whose value equals x and the previous one can be either greater or smaller than x. The same occurs with its following element, which is something that cannot happen in a balanced tree since, as said before, the element are sorted ahead inserting in the tree.

In the following pictures we can see a balanced (almost complete) binary search tree and of depth 3 and a non-complete binary search tree of depth 3 as well. This latter lacks node number 19 (in gray) in order to become a balanced tree.



Balanced tree

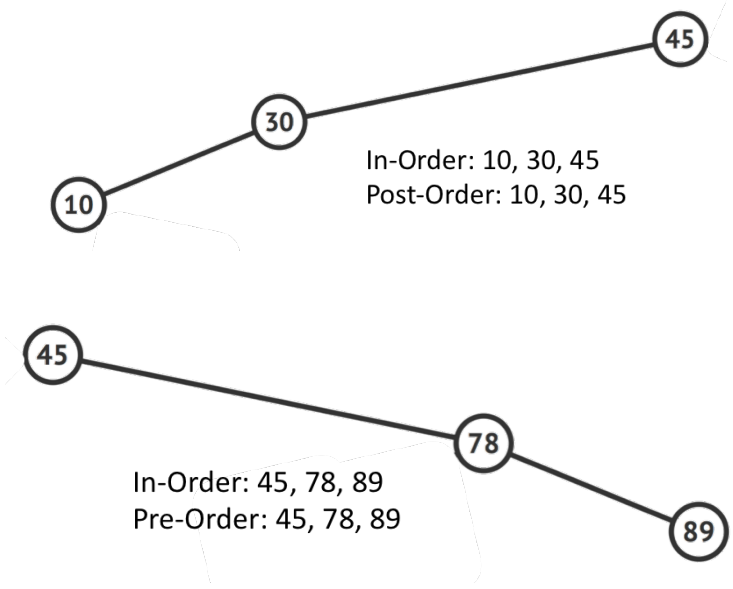


Non-balanced tree

EXERCISE 2

A)

If a tree is really unbalanced on the left side, the in-order result is going to look similar to the post-order output, while if the tree is unbalanced on the right side, the in-order output would be similar to the pre-order result.



B)

We can find out whether a binary tree is well built or not by going over the tree using the in-order algorithm. If the result we get is numerically or alphabetically well ordered from lowest to highest, we conclude that the binary search tree is well constructed. However, if we do not obtain this output, the binary search tree is not properly built.

EXERCISE 3

For this exercise, we will take into consideration the file dict10k.dat because we believe it is a good representation of what happens with all the other files as well.

When we create a balanced tree for this file, we get a total of 10000 nodes inserted with a depth of 13 and since $\log_2 10000 = 13{,}287\ldots$, we are getting an almost complete tree (as explained in exercise 1).

```
MacBook-Pro-de-Laura:P4_Prog2_G2192_P10 lauradepaz$ ./p4_e3 dict10K.dat B
10000 lineas leidas
Datos ordenados

Tiempo de creacion del Arbol: 43677 ticks (0.043677 segundos)
Numero de nodos: 10000
Profundidad: 13
Introduce un nodo para buscar en el Arbol (siguiendo el mismo formato que en el fichero de entrada):
3 aa
Elemento NO encontrado!

Tiempo de busqueda en el Arbol: 50 ticks (0.000050 segundos)
```

When we create a non-balanced tree for this file, we get a total of 10000 nodes inserted with a depth of 30. There is no order within the nodes of this file and that is why this value if far from the depth we would expect in a complete tree.

```
MacBook-Pro-de-Laura:P4_Prog2_G2192_P10 lauradepaz$ ./p4_e3 dict10K.dat N
10000 lineas leidas

Tiempo de creacion del Arbol: 60716 ticks (0.060716 segundos)
Numero de nodos: 10000
Profundidad: 30
Introduce un nodo para buscar en el Arbol (siguiendo el mismo formato que en el fichero de entrada):
3 aa
Elemento NO encontrado!

Tiempo de busqueda en el Arbol: 49 ticks (0.000049 segundos)
```

DESIGN CHOICES

In this practice, we did not have much options for choice because we were given all the .h as well as the data structures so we just had to implement the code for the functions.

We had some issues while doing this practice. First, we had some valgrind errors that lead us to *tree_insertRec()* and we could not find where it came from. The problem actually was in *destroyNodeAB()* in which we were not freeing the right information that had to be freed.

For exercise 4, we had to create an additional function so as to read the strings in the cadenas.txt file because they could not be read with the functions provided by the standard libraries.