

Parte 1

Importante: Los ejercicios de esta primera parte tienen como objetivo codificar las diferentes funciones básicas necesarias para la implementar un árbol AVL.

A partir de estructuras definidas como :

```
class AVLTree:
    root = None

class AVLNode:
    parent = None
    leftnode = None
    rightnode = None
    key = None
    value = None
    bf = None
```

Copiar y adaptar todas las operaciones del **binarytree.py** (i.e insert(), delete(), search(),etc) al nuevo módulo **avltree.py**. Notar que estos luego deberán ser implementados para cumplir que la propiedad de un árbol AVL

Ejercicio 1

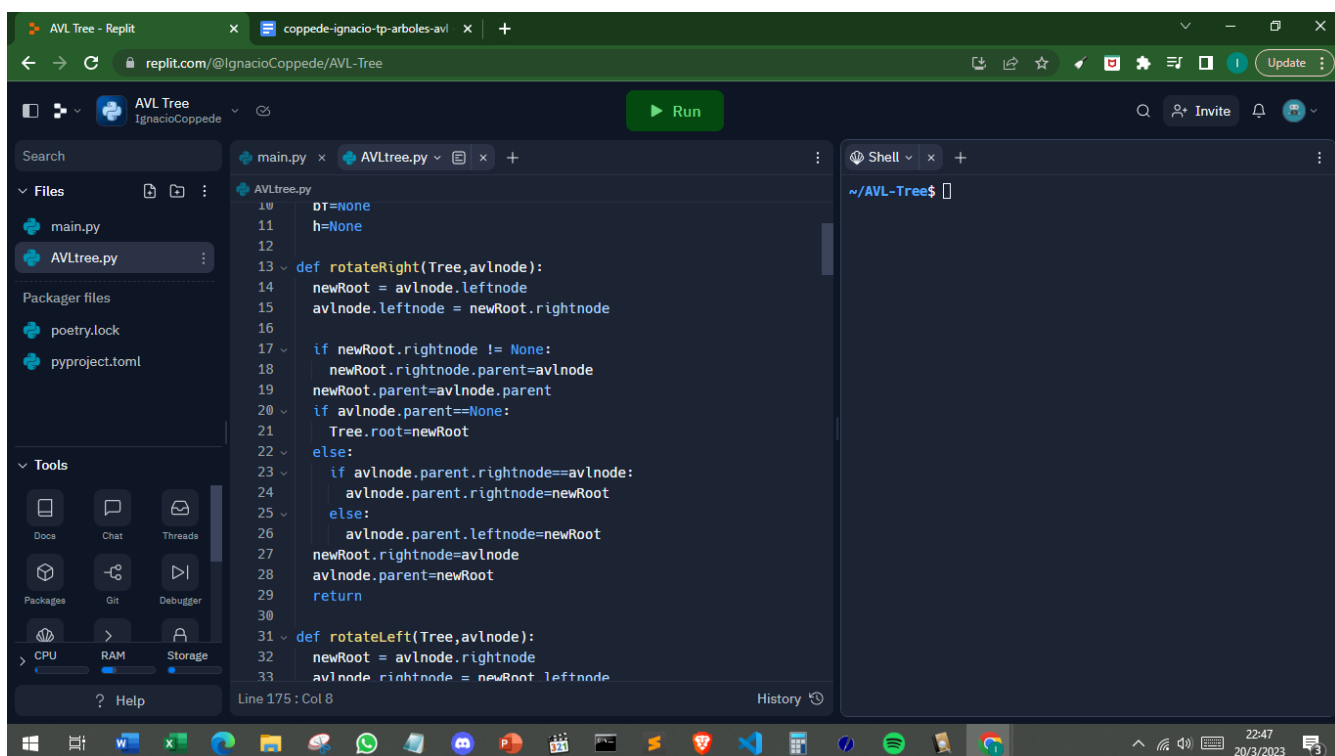
Crear un modulo de nombre **avltree.py** Implementar las siguientes funciones:

rotateLeft(Tree,avlnode)

Descripción: Implementa la operación rotación a la izquierda

Entrada: Un Tree junto a un AVLnode sobre el cual se va a operar la rotación a la izquierda

Salida: retorna la nueva raíz



The screenshot shows a Replit environment with a file explorer on the left containing 'main.py', 'AVLtree.py', 'poetry.lock', and 'pyproject.toml'. The main editor displays the 'AVLtree.py' file with the following code:

```
10  dt=None
11  h=None
12
13  def rotateRight(Tree,avlnode):
14      newRoot = avlnode.leftnode
15      avlnode.leftnode = newRoot.rightnode
16
17      if newRoot.rightnode != None:
18          newRoot.rightnode.parent=avlnode
19          newRoot.parent=avlnode.parent
20      if avlnode.parent==None:
21          Tree.root=newRoot
22      else:
23          if avlnode.parent.rightnode==avlnode:
24              avlnode.parent.rightnode=newRoot
25          else:
26              avlnode.parent.leftnode=newRoot
27      newRoot.rightnode=avlnode
28      avlnode.parent=newRoot
29      return
30
31  def rotateLeft(Tree,avlnode):
32      newRoot = avlnode.rightnode
33      avlnode.rightnode = newRoot.leftnode
```

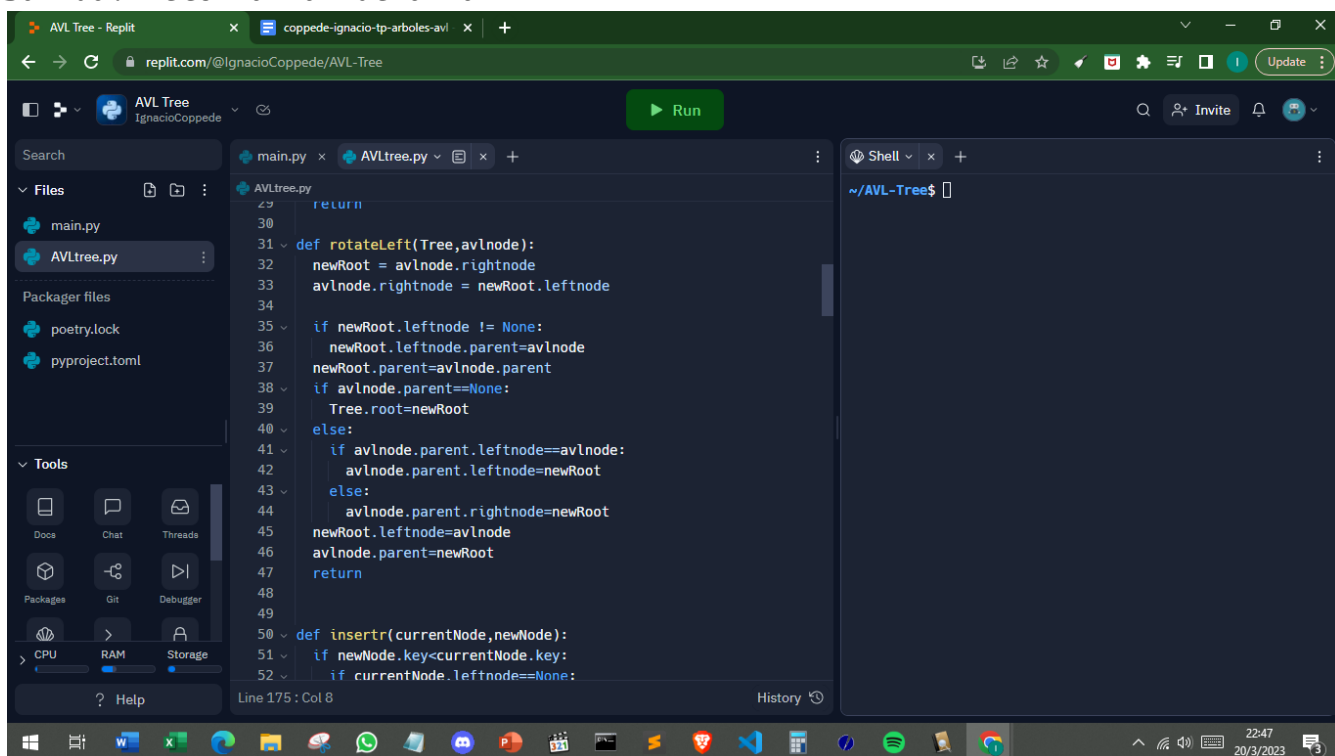
The right-hand pane shows a terminal window with the prompt '~ /AVL-Tree\$'.

rotateRight(Tree,avlnode)

Descripción: Implementa la operación rotación a la derecha

Entrada: Un Tree junto a un AVLnode sobre el cual se va a operar la rotación a la derecha

Salida: retorna la nueva raíz



The screenshot shows the same Replit environment as the previous one, but with the 'AVLtree.py' file displaying the following code:

```
27  newRoot.rightnode=avlnode
28  avlnode.parent=newRoot
29  return
30
31  def rotateLeft(Tree,avlnode):
32      newRoot = avlnode.rightnode
33      avlnode.rightnode = newRoot.leftnode
34
35      if newRoot.leftnode != None:
36          newRoot.leftnode.parent=avlnode
37          newRoot.parent=avlnode.parent
38      if avlnode.parent==None:
39          Tree.root=newRoot
40      else:
41          if avlnode.parent.leftnode==avlnode:
42              avlnode.parent.leftnode=newRoot
43          else:
44              avlnode.parent.rightnode=newRoot
45      newRoot.leftnode=avlnode
46      avlnode.parent=newRoot
47      return
48
49
50  def insertR(currentNode,newNode):
51      if newNode.key<currentNode.key:
52          if currentNode.leftnode==None:
```

The right-hand pane shows the same terminal window with the prompt '~ /AVL-Tree\$'.

Ejercicio 2

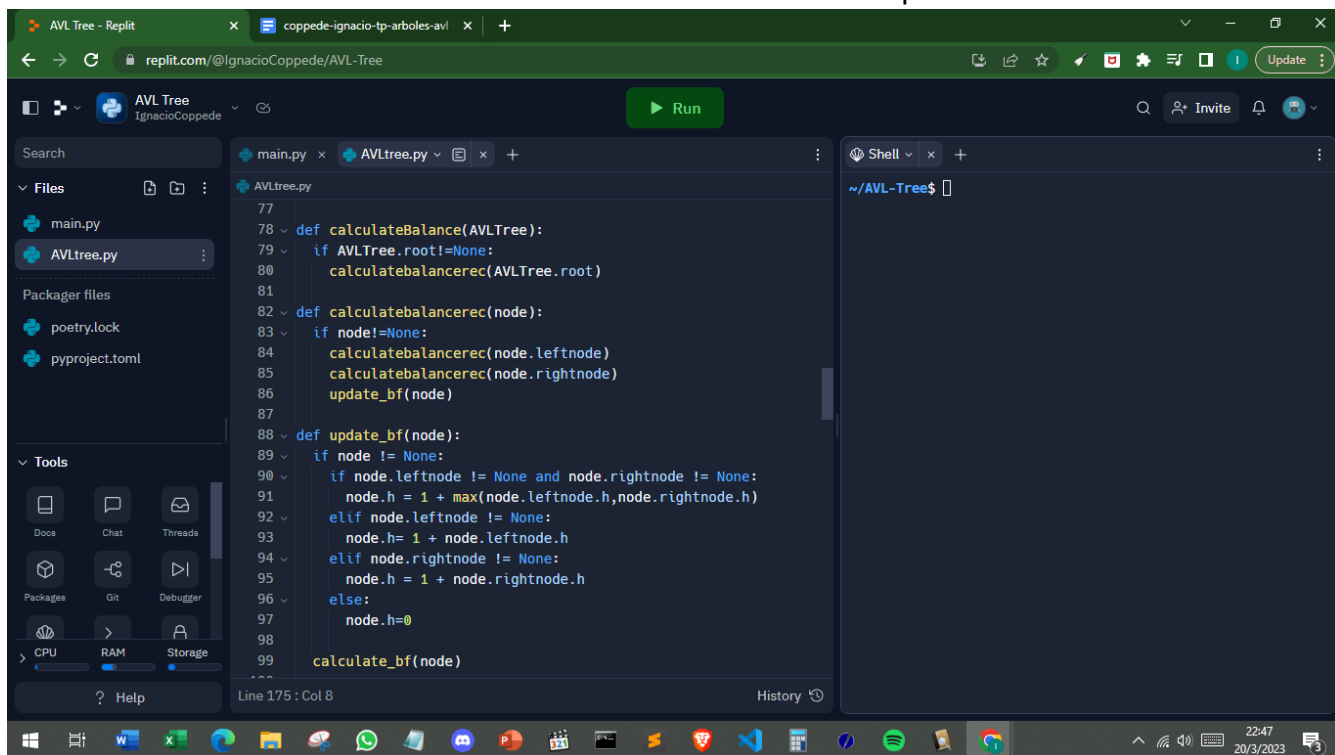
Implementar una función recursiva que calcule el elemento balanceFactor de cada subárbol siguiendo la siguiente especificación:

calculateBalance(AVLTree)

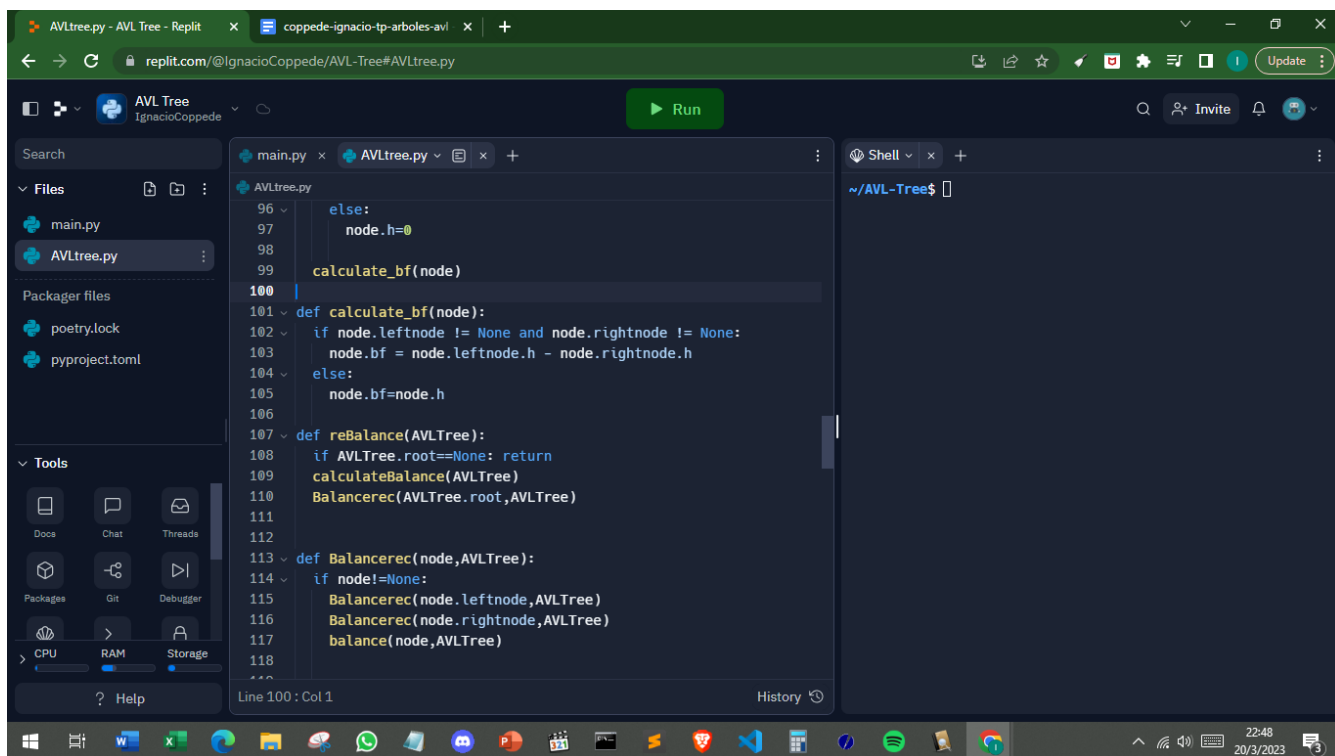
Descripción: Calcula el factor de balanceo de un árbol binario de búsqueda.

Entrada: El árbol AVL sobre el cual se quiere operar.

Salida: El árbol AVL con el valor de balanceFactor para cada subarbol



```
77
78 def calculateBalance(AVLTree):
79     if AVLTree.root!=None:
80         calculatebalancerec(AVLTree.root)
81
82 def calculatebalancerec(node):
83     if node!=None:
84         calculatebalancerec(node.leftnode)
85         calculatebalancerec(node.rightnode)
86         update_bf(node)
87
88 def update_bf(node):
89     if node != None:
90         if node.leftnode != None and node.rightnode != None:
91             node.h = 1 + max(node.leftnode.h,node.rightnode.h)
92         elif node.leftnode != None:
93             node.h = 1 + node.leftnode.h
94         elif node.rightnode != None:
95             node.h = 1 + node.rightnode.h
96         else:
97             node.h=0
98
99 calculate_bf(node)
```



```
96
97     node.h=0
98
99     calculate_bf(node)
100
101 def calculate_bf(node):
102     if node.leftnode != None and node.rightnode != None:
103         node.bf = node.leftnode.h - node.rightnode.h
104     else:
105         node.bf=node.h
106
107 def reBalance(AVLTree):
108     if AVLTree.root==None: return
109     calculateBalance(AVLTree)
110     Balancerec(AVLTree.root,AVLTree)
111
112
113 def Balancerec(node,AVLTree):
114     if node!=None:
115         Balancerec(node.leftnode,AVLTree)
116         Balancerec(node.rightnode,AVLTree)
117         balance(node,AVLTree)
118
```

Ejercicio 3

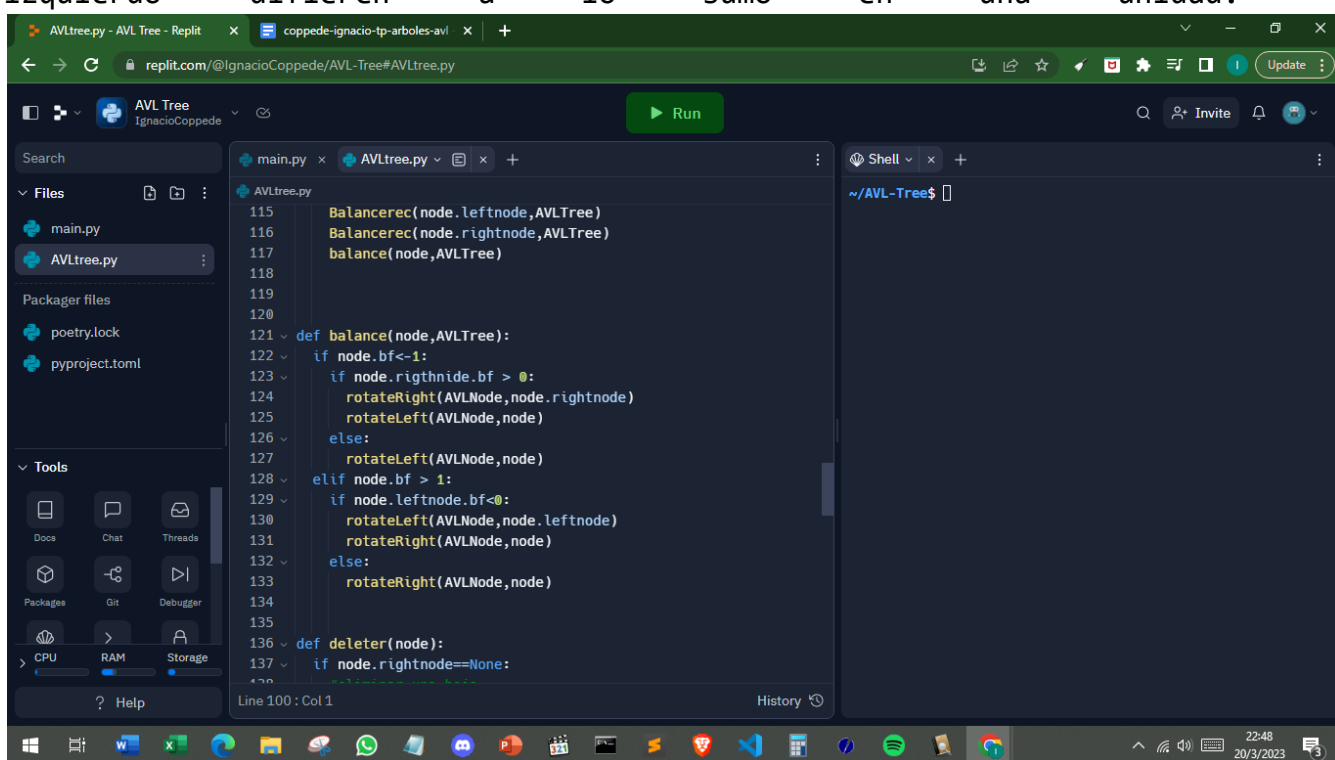
Implementar una función en el módulo `avltree.py` de acuerdo a las siguientes especificaciones:

`reBalance(AVLTree)`

Descripción: balancea un árbol binario de búsqueda. Para esto se deberá primero calcular el **balanceFactor** del árbol y luego en función de esto aplicar la estrategia de rotación que corresponda.

Entrada: El árbol binario de tipo AVL sobre el cual se quiere operar.

Salida: Un árbol binario de búsqueda balanceado. Es decir luego de esta operación se cumple que la altura (h) de su subárbol derecho e izquierdo difieren a lo sumo en una unidad.



The screenshot shows a Replit environment with a file explorer on the left containing `main.py`, `AVLtree.py`, `poetry.lock`, and `pyproject.toml`. The main editor displays the `AVLtree.py` file with the following code:

```
115     Balancerec(node.leftnode,AVLTree)
116     Balancerec(node.rightnode,AVLTree)
117     balance(node,AVLTree)
118
119
120
121 def balance(node,AVLTree):
122     if node.bf<-1:
123         if node.righthnode.bf > 0:
124             rotateRight(AVLNode,node.righthnode)
125             rotateLeft(AVLNode,node)
126         else:
127             rotateLeft(AVLNode,node)
128     elif node.bf > 1:
129         if node.leftnode.bf<0:
130             rotateLeft(AVLNode,node.leftnode)
131             rotateRight(AVLNode,node)
132         else:
133             rotateRight(AVLNode,node)
134
135
136 def deleter(node):
137     if node.righthnode==None:
```

The right sidebar shows a shell terminal with the prompt `~/AVL-Tree$`. The bottom status bar indicates the cursor is at Line 100, Column 1.

Ejercicio 4:

Implementar la operación `insert()` en el módulo `avltree.py` garantizando que el árbol binario resultante sea un árbol AVL.

The image displays two screenshots of a Replit environment, showing the implementation of an AVL tree. The top screenshot shows the 'insert' function and the 'insert final' block. The bottom screenshot shows the 'insert final' block, 'calculateBalance' function, and 'calculatebalancerec' function.

Top Screenshot:

```
def inserttr(currentNode,newNode):
    if newNode.key<currentNode.key:
        if currentNode.leftnode==None:
            currentNode.leftnode=newNode
            newNode.parent=currentNode
        else:
            inserttr(currentNode.leftnode,newNode)
    else:
        if currentNode.rightnode==None:
            currentNode.rightnode=newNode
            newNode.parent=currentNode
        else:
            inserttr(currentNode.rightnode,newNode)

#insert final
def insert(B,element,key):
    newNode=AVLNode()
    newNode.value=element
    newNode.key=key
    newNode.bf=0
    if B.root==None:
```

Bottom Screenshot:

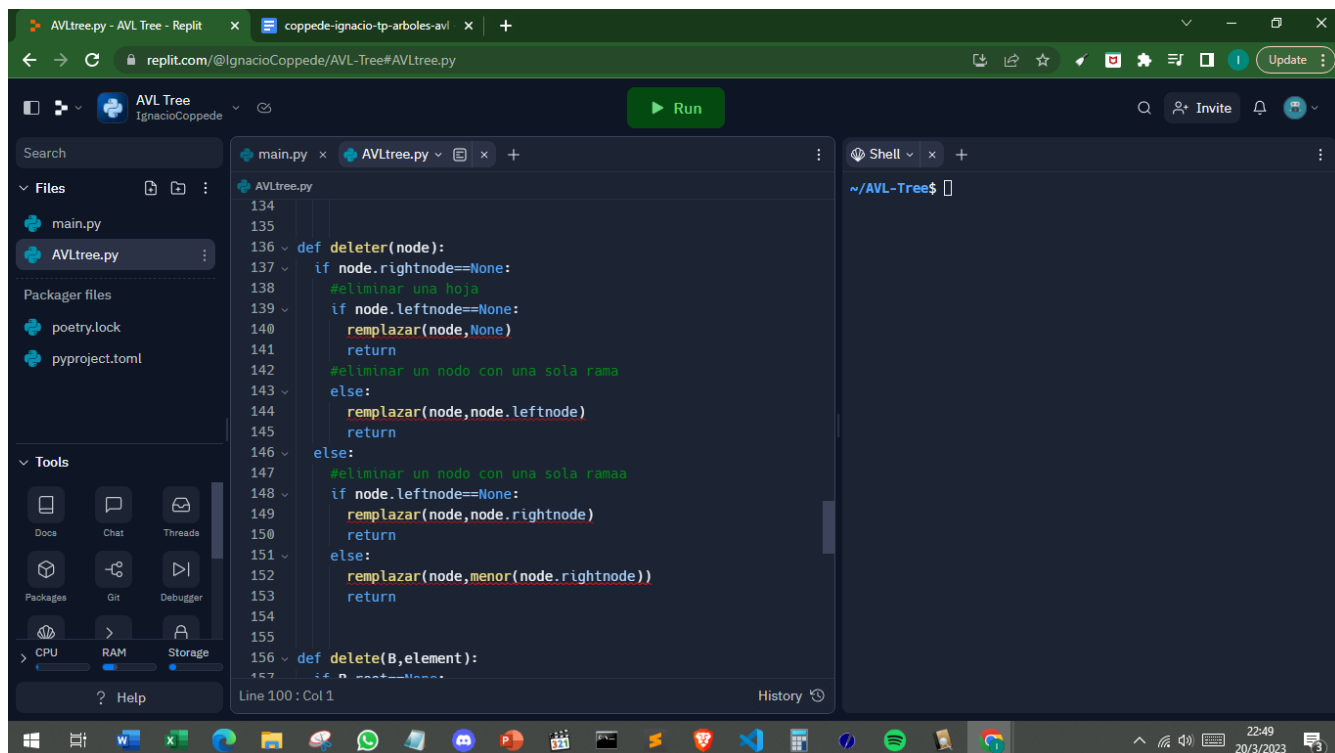
```
#insert final
def insert(B,element,key):
    newNode=AVLNode()
    newNode.value=element
    newNode.key=key
    newNode.bf=0
    if B.root==None:
        B.root=newNode
        newNode.parent=B.root
    else:
        inserttr(B.root,newNode)
    update_bf(newNode)

def calculateBalance(AVLTree):
    if AVLTree.root!=None:
        calculatebalancerec(AVLTree.root)

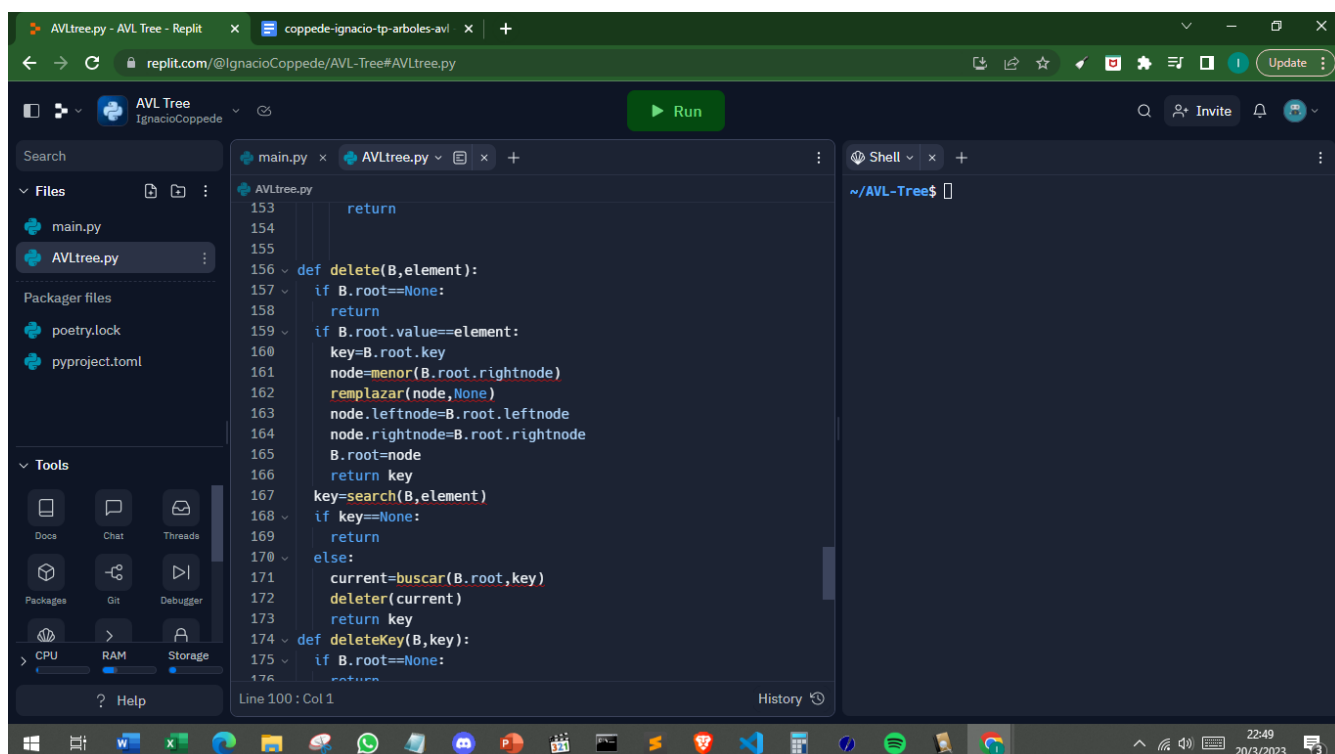
def calculatebalancerec(node):
    if node!=None:
        calculatebalancerec(node.leftnode)
        calculatebalancerec(node.rightnode)
```

Ejercicio 5:

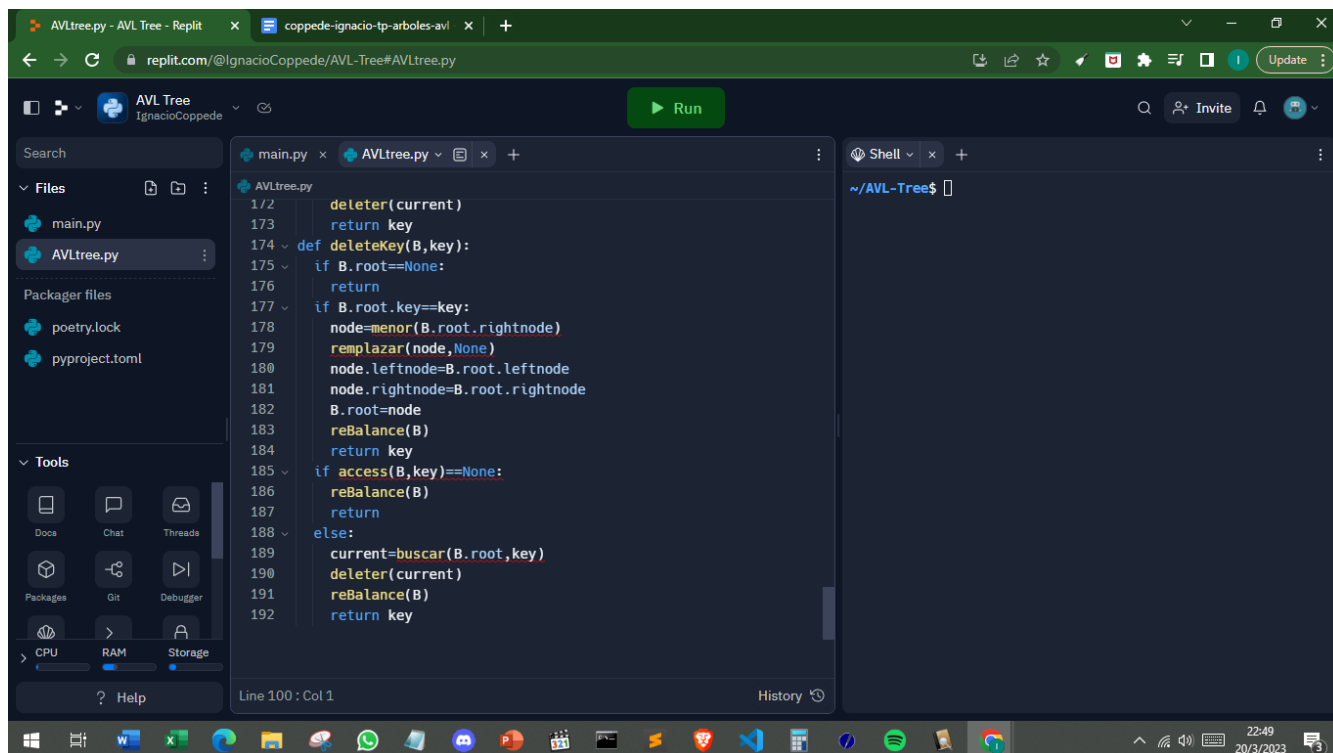
Implementar la operación `delete()` en el módulo `avltree.py` garantizando que el árbol binario resultante sea un árbol AVL.



```
134
135
136 def delete(node):
137     if node.rightnode==None:
138         #eliminar una hoja
139         if node.leftnode==None:
140             remlazar(node,None)
141             return
142         #eliminar un nodo con una sola rama
143         else:
144             remlazar(node,node.leftnode)
145             return
146     else:
147         #eliminar un nodo con una sola rama
148         if node.leftnode==None:
149             remlazar(node,node.rightnode)
150             return
151         else:
152             remlazar(node,menor(node.rightnode))
153             return
154
155
156 def delete(B,element):
157     if B.root==None:
```



```
153         return
154
155
156 def delete(B,element):
157     if B.root==None:
158         return
159     if B.root.value==element:
160         key=B.root.key
161         node=menor(B.root.rightnode)
162         remlazar(node,None)
163         node.leftnode=B.root.leftnode
164         node.rightnode=B.root.rightnode
165         B.root=node
166         return key
167     key=search(B,element)
168     if key==None:
169         return
170     else:
171         current=buscar(B.root,key)
172         delete(current)
173         return key
174 def deleteKey(B,key):
175     if B.root==None:
176         return
```



```
1/2     delete(current)
173     return key
174 def deleteKey(B, key):
175     if B.root==None:
176         return
177     if B.root.key==key:
178         node=menor(B.root.righnode)
179         remplazar(node, None)
180         node.leftnode=B.root.leftnode
181         node.righnode=B.root.righnode
182         B.root=node
183         reBalance(B)
184         return key
185     if access(B, key)==None:
186         reBalance(B)
187         return
188     else:
189         current=buscar(B.root, key)
190         delete(current)
191         reBalance(B)
192         return key
```

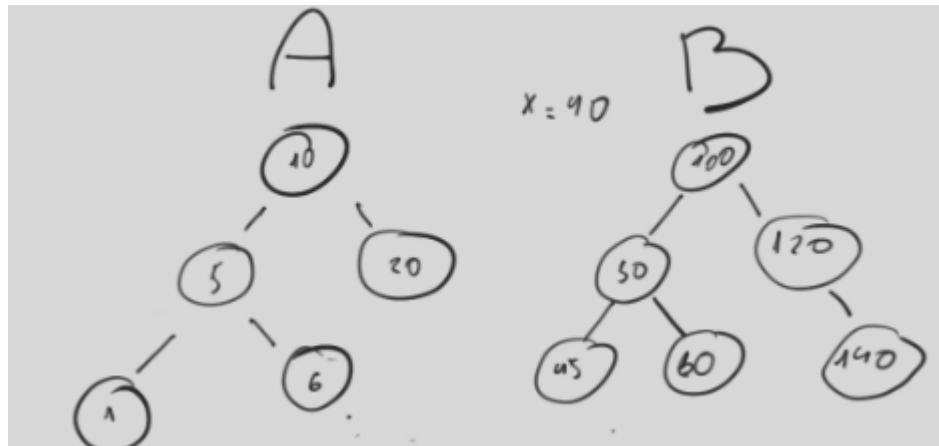
Parte 2

Ejercicio 6:

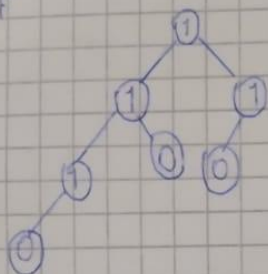
1. Responder V o F y justificar su respuesta:
 - a. ☐ En un AVL el penúltimo nivel tiene que estar completo
 - b. ☐ Un AVL donde todos los nodos tengan factor de balance 0 es completo
 - c. ☐ En la inserción en un AVL, si al actualizarle el factor de balance al padre del nodo insertado éste no se desbalanceó, entonces no hay que seguir verificando hacia arriba porque no hay cambios en los factores de balance.
 - d. ☐ En todo AVL existe al menos un nodo con factor de balance 0.

Ejercicio 7:

Sean A y B dos AVL de m y n nodos respectivamente y sea x un key cualquiera de forma tal que para todo key $a \in A$ y para todo key $b \in B$ se cumple que $a < x < b$. Plantear un algoritmo $O(\log n + \log m)$ que devuelva un AVL que contenga los key de A , el key x y los key de B .



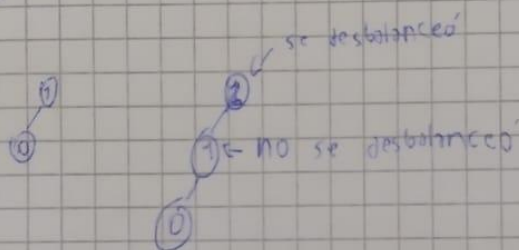
6) a. F



Podemos ver como es un AVL, sin tener el penúltimo nivel completo

b. V. La única forma de todos los nodos tengan factor de balance cero es que todos los nodos tengan dos o cero hijos, y la misma altura en cada subárbol

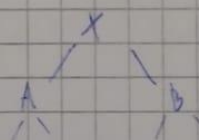
c. F



d. V. Las hojas siempre van a tener $bf=0$

f)

Podríamos crear un nuevo árbol en el que la raíz sea x y su hijo izquierdo es árbol ~~izquierdo~~ A y el su hijo derecho está conectado a la raíz del ~~árbol~~ árbol B, y luego y rebalanceamos de ser necesario



Ejercicio 8:

Considere una rama truncada en un AVL como un camino simple desde la raíz hacia un nodo que tenga una referencia None (que le falte algún hijo). Demuestre que la mínima longitud (cantidad de aristas) que puede tener una rama truncada en un AVL de altura h es $h/2$ (tomando la parte entera por abajo).

Cualquier camino desde la raíz hasta un nodo que no esté completo puede ser una rama truncada según la definición del ejercicio. Dicho nodo puede no ser necesariamente un nodo hoja.

Parte 3

Ejercicios Opcionales

1. Si n es la cantidad de nodos en un árbol AVL, implemente la operación `height()` en el módulo `avltree.py` que determine su altura en $O(\log n)$. Justifique el por qué de dicho orden.
2. Considere una modificación en el módulo `avltree.py` donde a cada nodo se le ha agregado el campo `count` que almacena el número de nodos que hay en el subárbol en el que él es raíz. Programe un algoritmo $O(\log n)$ que determine la cantidad de nodos en el árbol cuyo valor del key se encuentra en un intervalo $[a, b]$ dado como parámetro. Explique brevemente por qué el algoritmo programado por usted tiene dicho orden.

A tener en cuenta:

1. Usen lápiz y papel primero
2. ~~No se puede utilizar otra Biblioteca mas alla de algo1.py y las bibliotecas desarrolladas durante Algoritmos y Estructuras de Datos I.~~

Bibliografía:

- [1] Guido Tagliavini Ponce, [Balanceo de arboles y arboles AVL](#) (Universidad de Buenos Aires)
- [2] Brad Miller and David Ranum, Luther College, [Problem Solving with Algorithms and Data Structures using Python](#).