

19/03/2023

## **Practica. -Binary Search Tree**

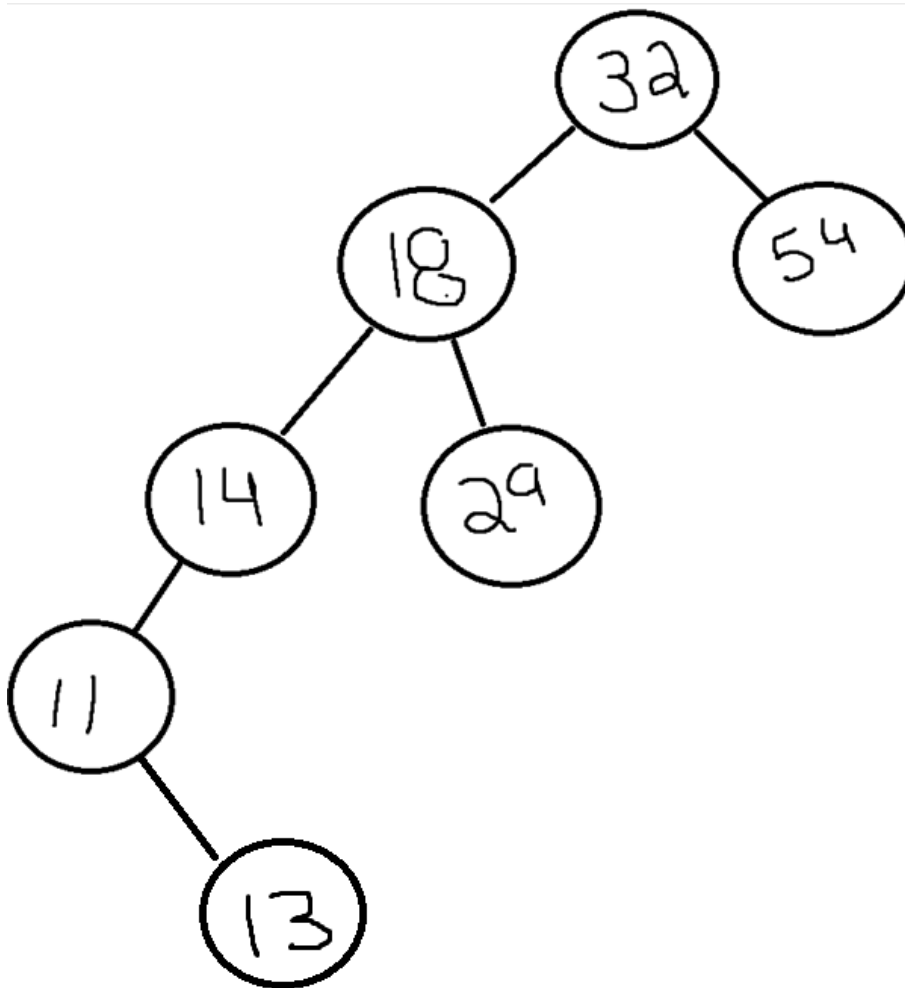
### **Estructura de Datos**

**Chávez Rodríguez Mauricio Yosef**



Diagrama que ejemplifica nuestro árbol con los siguientes valores de prueba:

32,18,54,29,14,11,13



**Función minimum value:**

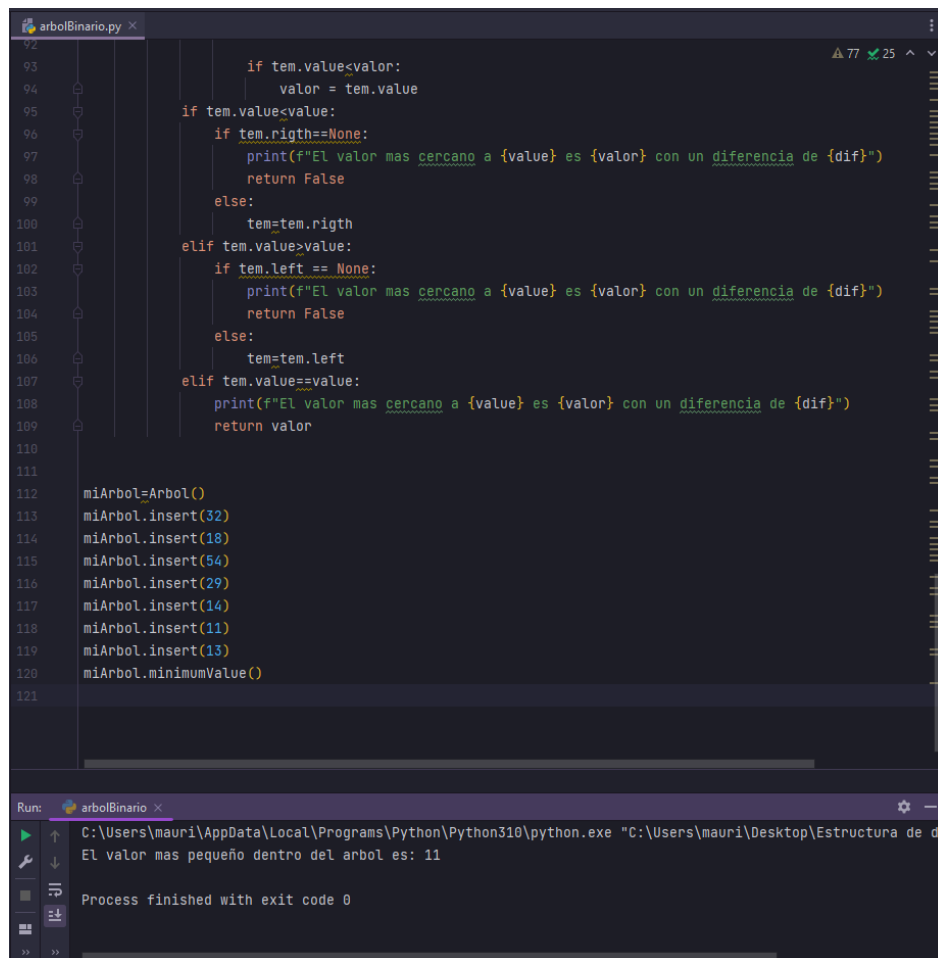
Devuelve el valor mínimo de un árbol de búsqueda binario.

**Explicación del código de la función:**

```
def minimumValue(self):  
    tem=self.raiz  
    while True:  
        if tem.left !=None:  
            tem=tem.left  
        else:  
            print(f"El valor mas pequeño dentro del arbol es: {tem.value} ")  
            return tem
```

Al ser un árbol binario podemos deducir que nuestro valor siempre estará a la izquierda del todo, pues el árbol acomoda por defecto a la izquierda el valor más pequeño, con ello podemos concluir que necesitamos hacer 1 solo recorrido de nuestro árbol, para ello hacemos comenzamos de la raíz en un ciclo while el cual dentro tendrá 2 condiciones si aun no llego al último que se encuentra en el lado izquierdo del todo avanzará izquierda, cuando ya se encuentre en el último nodo del lado izquierdo entrara al else imprimiendo el valor mínimo el cual se encontrara en la variable tem que hace el recorrido de los nodos.

### Prueba de Escritorio Funcion *mínimum value*:



```
92
93         if tem.value < valor:
94             valor = tem.value
95     if tem.value < valor:
96         if tem.rigth == None:
97             print(f"El valor mas cercano a {valor} es {valor} con un diferencia de {dif}")
98             return False
99         else:
100             tem = tem.rigth
101     elif tem.value > valor:
102         if tem.left == None:
103             print(f"El valor mas cercano a {valor} es {valor} con un diferencia de {dif}")
104             return False
105         else:
106             tem = tem.left
107     elif tem.value == valor:
108         print(f"El valor mas cercano a {valor} es {valor} con un diferencia de {dif}")
109         return valor
110
111
112     miArbol = Arbol()
113     miArbol.insert(32)
114     miArbol.insert(18)
115     miArbol.insert(54)
116     miArbol.insert(29)
117     miArbol.insert(14)
118     miArbol.insert(11)
119     miArbol.insert(13)
120     miArbol.minimumValue()
121
```

Run: arbolBinario

C:\Users\mauri\AppData\Local\Programs\Python\Python310\python.exe "C:\Users\mauri\Desktop\Estructura de de

El valor mas pequeño dentro del arbol es: 11

Process finished with exit code 0

Como se puede observar en la imagen al ejecutar la función *mínimum value* en el árbol ejemplo que se muestra en el diagrama anterior nos retorna el valor 11, que si verificamos en el diagrama mencionado podemos notar que coincide lo que se puede ver en el diagrama de árbol con lo retornado por nuestra función, con ello podemos confirmar el funcionamiento adecuado de nuestro método.

### Función *closes value*:

Escriba un programa de Python para encontrar el valor más cercano de un valor objetivo dado en un Árbol de búsqueda binaria (BST) no vacío dado de valores únicos.

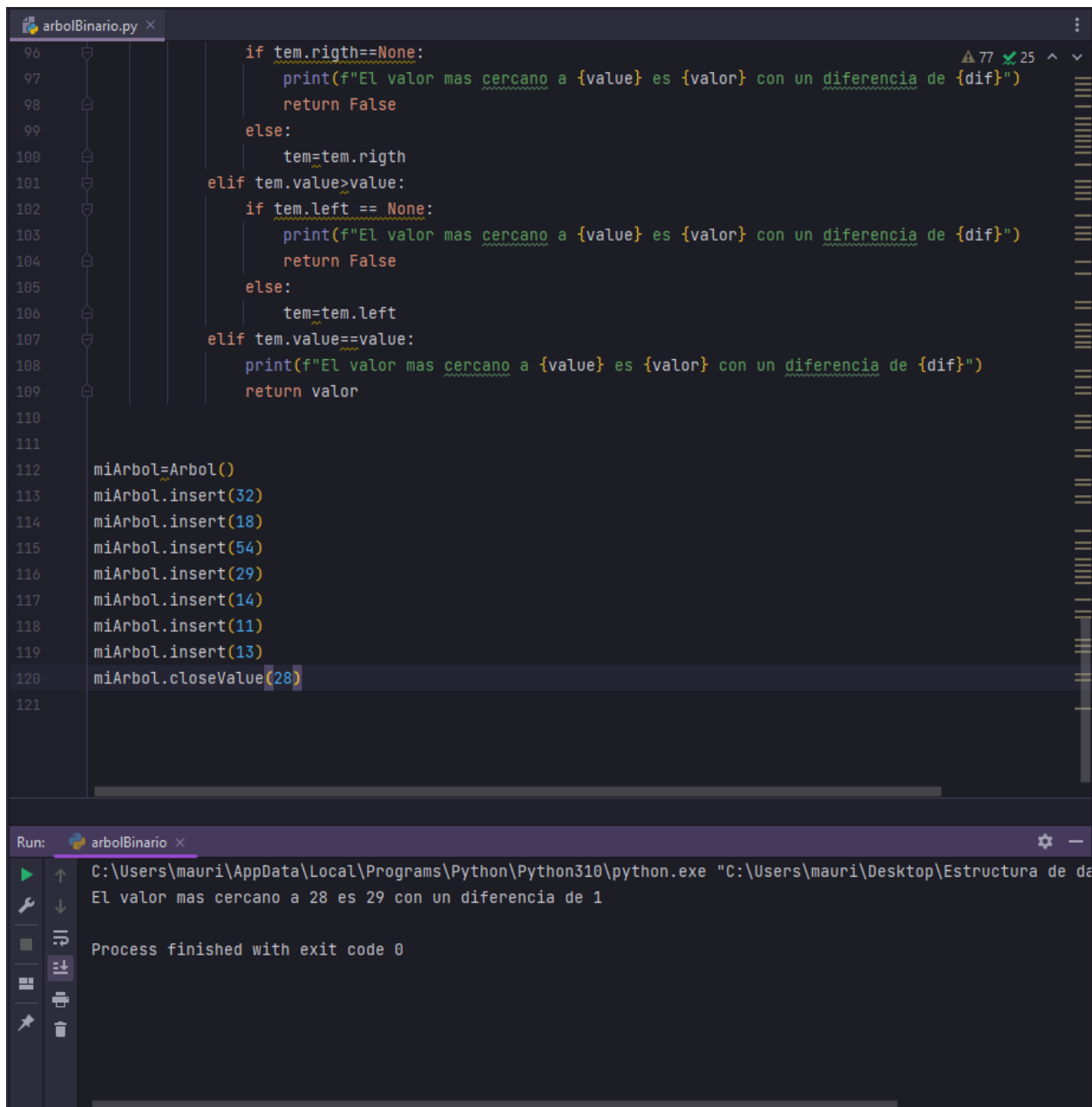
```

def closeValue(self, value):
    tem = self.raiz
    dif=10000
    valor=0
    while True:
        if abs(tem.value-value)<=dif:
            dif=abs(tem.value-value)
            valor=tem.value
            if abs(tem.value-value)==dif:
                if tem.value>valor:
                    valor = valor

                if tem.value<valor:
                    valor = tem.value
        if tem.value<value:
            if tem.rigth==None:
                print(f"El valor mas cercano a {value} es {valor} con un diferencia de {dif}")
                return False
            else:
                tem=tem.rigth
        elif tem.value>value:
            if tem.left == None:
                print(f"El valor mas cercano a {value} es {valor} con un diferencia de {dif}")
                return False
            else:
                tem=tem.left
        elif tem.value==value:
            print(f"El valor mas cercano a {value} es {valor} con un diferencia de {dif}")
            return valor

```

A diferencia del minimum value en esta función desconocemos el camino que recorrerá nuestra función , pero podemos saber que camino teóricamente tendría, para ello creamos un ciclo while como habitualmente se hace en los recorridos, si nuestro valor ingresado es mayor que la raíz se ira a la derecha y si es menor a la izquierda, bajo estas reglas haremos el recorrido como si fuéramos a colocarlo, mientras hacemos esto en cada iteración compararemos el nodo en el que estamos con el valor que deseamos conocer su valor más cercano, para ello sacaremos la diferencia y guardaremos la diferencia más pequeña en la variable dif y su valor es decir, el número almacenado en el nodo con el que dicha diferencia es la más pequeña encontrada, esto se hará en cada iteración ,una vez terminado el recorrido el programa imprimirá el valor con el que tenga menor diferencia ósea lo que tiene guardado la variable valor y la variable dif.



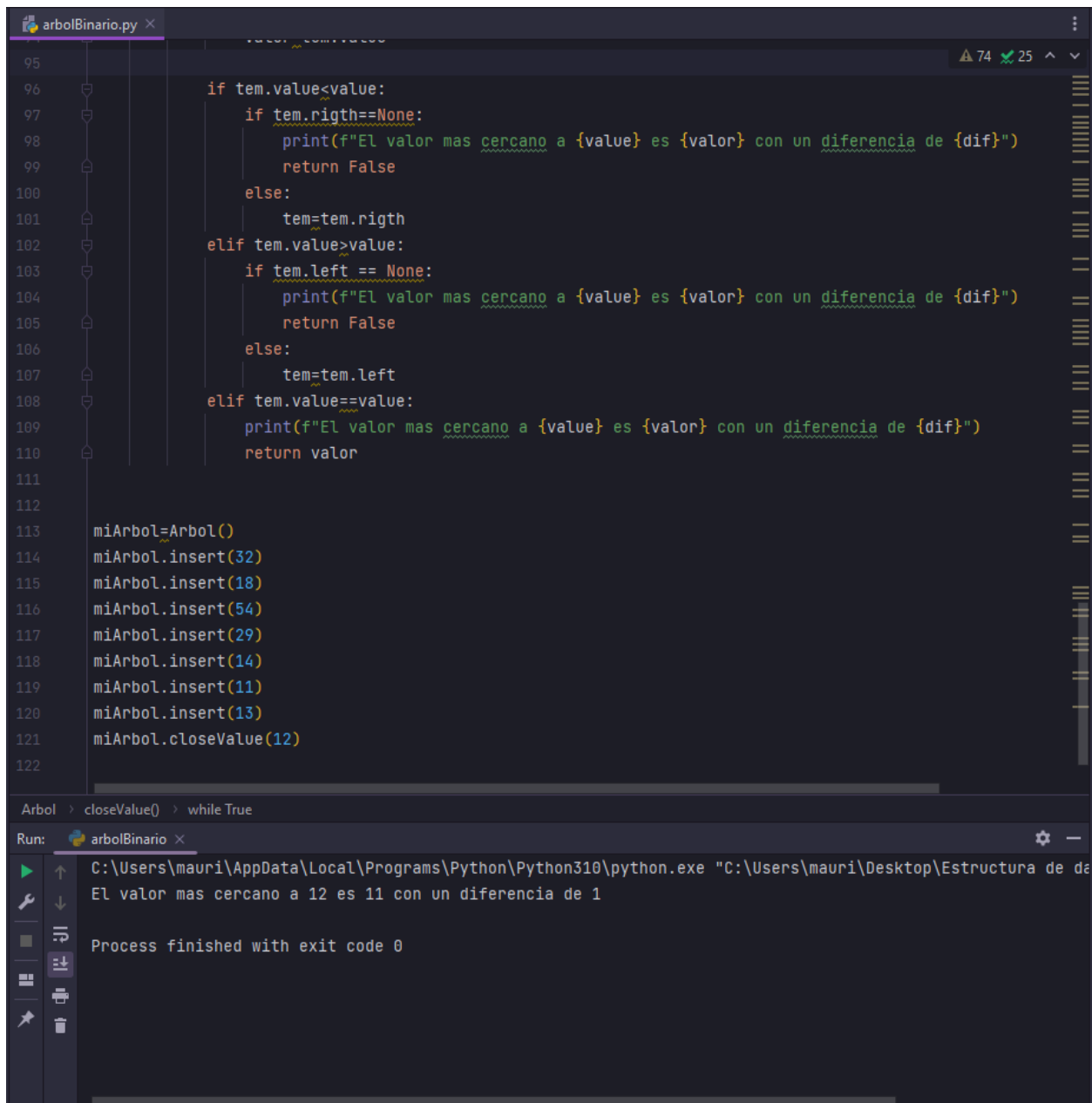
The image shows a Python IDE with a file named `arbolBinario.py`. The code implements a binary search algorithm on a binary tree. The tree structure is visualized on the left side of the editor. The code consists of two main parts: a search function and a main execution block.

```
96     if tem.rigth==None:
97         print(f"El valor mas cercano a {value} es {valor} con un diferencia de {dif}")
98         return False
99     else:
100         tem=tem.rigth
101     elif tem.value>value:
102         if tem.left == None:
103             print(f"El valor mas cercano a {value} es {valor} con un diferencia de {dif}")
104             return False
105         else:
106             tem=tem.left
107     elif tem.value==value:
108         print(f"El valor mas cercano a {value} es {valor} con un diferencia de {dif}")
109         return valor
110
111
112 miArbol=Arbol()
113 miArbol.insert(32)
114 miArbol.insert(18)
115 miArbol.insert(54)
116 miArbol.insert(29)
117 miArbol.insert(14)
118 miArbol.insert(11)
119 miArbol.insert(13)
120 miArbol.closeValue(28)
121
```

The execution output at the bottom shows the command executed and the resulting message:

```
Run: arbolBinario x
C:\Users\mauri\AppData\Local\Programs\Python\Python310\python.exe "C:\Users\mauri\Desktop\Estructura de da
El valor mas cercano a 28 es 29 con un diferencia de 1
Process finished with exit code 0
```

Al hacer la prueba de escritorio buscando el valor más cercano a 28 en el árbol que se observó en el diagrama anterior, podemos notar que el programa nos dice que es el 29 lo cual coincide con la información proporcionada por el diagrama.



The image shows a Python IDE with a file named `arbolBinario.py`. The code implements a binary search algorithm on a binary tree. The tree structure is visualized on the left side of the editor. The code defines a `closeValue` method that finds the value in the tree closest to a given `value`. It uses recursive calls to traverse the tree, comparing the current node's value with the target value and updating the closest value found so far. The tree is populated with values: 32, 18, 54, 29, 14, 11, 13, and 12. The `closeValue` method is called with the value 12, and the output shows that the closest value is 11 with a difference of 1.

```
95
96     if tem.value < value:
97         if tem.rigth == None:
98             print(f"El valor mas cercano a {value} es {valor} con un diferencia de {dif}")
99             return False
100        else:
101            tem = tem.rigth
102    elif tem.value > value:
103        if tem.left == None:
104            print(f"El valor mas cercano a {value} es {valor} con un diferencia de {dif}")
105            return False
106        else:
107            tem = tem.left
108    elif tem.value == value:
109        print(f"El valor mas cercano a {value} es {valor} con un diferencia de {dif}")
110        return valor
111
112
113    miArbol = Arbol()
114    miArbol.insert(32)
115    miArbol.insert(18)
116    miArbol.insert(54)
117    miArbol.insert(29)
118    miArbol.insert(14)
119    miArbol.insert(11)
120    miArbol.insert(13)
121    miArbol.closeValue(12)
122
```

Arbol > closeValue() > while True

Run: arbolBinario x

C:\Users\mauri\AppData\Local\Programs\Python\Python310\python.exe "C:\Users\mauri\Desktop\Estructura de da  
El valor mas cercano a 12 es 11 con un diferencia de 1

Process finished with exit code 0

Para lo siguiente prueba de escritorio aprovechamos que existen 2 datos que pueden tener diferencia similar con un número, dentro del diagrama antes mostrado se puede ver que los datos 11 y 13 forman parte de él, por ello en esta prueba pedimos la búsqueda del valor mas cercano al 12, pues la idea del programa es que al existir valores con mismas diferencias se tome en consideración el mas pequeño, como podemos observar nuestro programa nos retorno el valor 11 siendo el más pequeño entre 11 y 13, con lo cual se confirma el funcionamiento adecuado de nuestro método.