

19/03/2023

## **Actividad. -Árboles Binarios**

### **Estructura de Datos**

**Chávez Rodríguez Mauricio Yosef**



### Explicación de Código:

```
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.rigth = None
```

Comenzamos definiendo nuestro nodo como una clase la cual tendrá un valor y apuntará en 2 direcciones como es habitual en un árbol binario es decir tendrá 2 hijos uno izquierdo y el otro derecho.

```
class Arbol:
    def __init__(self):
        self.raiz = None
        self.length = 0
```

En nuestra clase árbol se encontrarán los métodos mediante los cuales crearemos el propio árbol binario pero la clase comienza con un constructor la cual inicializará 2 atributos, uno para conocer el número de nodos dentro de nuestro árbol y el otro nos ayudará para orientarnos en el árbol y poder recorrerlo.

```
def insert(self, value):
    if self.raiz == None:
        new_node = Node(value)
        self.raiz = new_node
        self.length = 1
    else:
        new_node = Node(value)
        tem = self.raiz
        while True:
            if tem.value < new_node.value:
                if tem.rigth == None:
                    tem.rigth = new_node
                    self.length += 1
                    return True
                else:
                    tem = tem.rigth
                    continue
            elif tem.value > new_node.value:
                if tem.left == None:
                    tem.left = new_node
                    self.length += 1
                    return True
                else:
                    tem = tem.left
                    continue
            elif tem.value == new_node.value:
                print("El valor ingresado ya existe en la lista")
                return False
```

El método Insert se encarga de agregar un nuevo nodo al árbol, creando su enlace respectivo, dicho método comienza corroborando la existencia de una raíz, es decir, de un nodo inicial, si no existe se crea y si ya existe se procede a enlazarse a la raíz el nuevo nodo, para ello se sigue la regla de un árbol binario, si el número es más grande que la raíz se enlaza a la derecha del nodo si es más pequeño a la izquierda, en caso de que el valor ya exista en el árbol, el propio programa te avisará que el valor ya existe y retornará falso, el ciclo para encontrar donde se coloca el nodo consta de recorrer el árbol comparando el nuevo nodo con los nodos en el árbol, comparando si es más grande o más pequeño hasta que se encuentre un nodo que apunte a nulo, que es donde se colocará nuestro nodo.

```
def print_nodovalor(self):
    tem=self.raiz
    while True:
        value = int(input("Ingresa hacia que lado te quieres dirigir:\n 1.-Izquierda 2.-Derecha 3.-Imprimir\n"))
        if value==1:
            tem=tem.left
        if value==2:
            tem=tem.rigth
        if value==3:
            print(tem.value)
            return True
```

Para nuestro método de impresión de el valor de un nodo, lo trabajaremos de una manera sencilla, el propio método te pregunta las direcciones a tomar partiendo de la raíz del árbol, como si de el copiloto en un viaje se tratara, se presiona 1 si quieres ir a la izquierda y 2 si quieres ir a la derecha, una vez estes en el nodo que desees conocer el valor presionas 3 y el programa imprimirá el valor del nodo en el que se encuentre.

```
def contains(self,value):
    tem=self.raiz
    while True:
        if tem.value<value:
            if tem.rigth==None:
                print(f"El valor {value} no se encuentra en el arbol")
                return False
            else:
                tem=tem.rigth
        elif tem.value>value:
            if tem.left == None:
                print(f"El valor {value} no se encuentra en el arbol")
                return False
            else:
                tem=tem.left
        elif tem.value==value:
            print(f"El valor {value} ya existe en la lista")
            return True
```

Nuestro método contains se encarga de decirnos cuando un valor ya se encuentra dentro de nuestro árbol, como se mencionó anteriormente en nuestro método insert ya hace dicha función, recordando esto nos podemos dar una idea de que la idea es casi la misma, hacemos el camino como si fuéramos a insertar un valor si llegamos a nulo significa que el valor no existe si encontramos una igualdad es decir si nuestro valor es igual a el valor de un nodo dentro del árbol entonces dicho valor ya existe en el árbol.

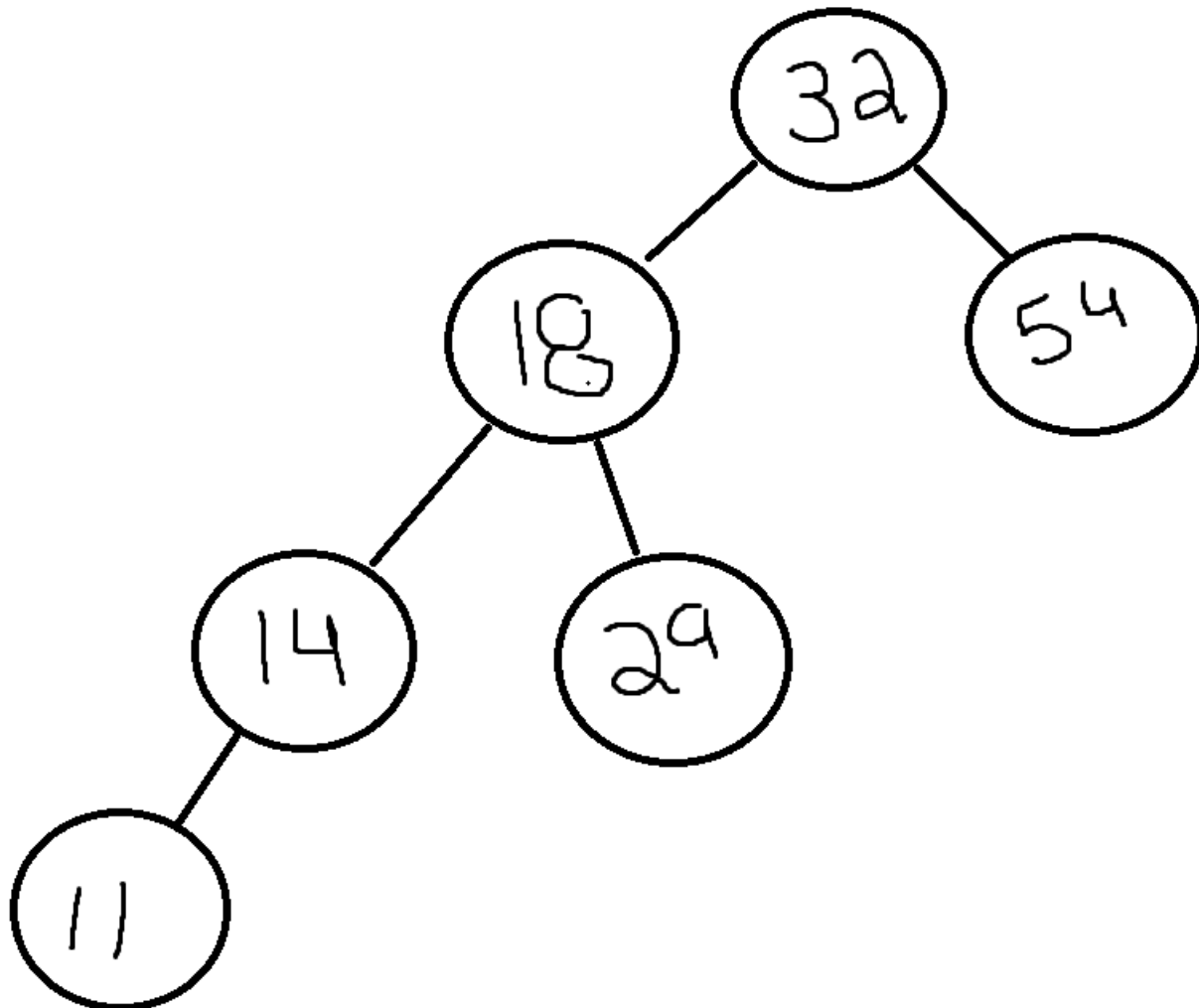
### ***Prueba de Escritorio:***


Una vez comprendido cómo funcionan nuestro programa se puede realizar la siguiente prueba, se ingresan los de prueba y se llama la función contains para conocer si los datos forman parte del árbol, de esta forma estaríamos revisando si nuestro programa funciona de manera adecuada.

### ***Datos de Prueba:***

**32, 18, 54, 29, 14, 11.**

### ***Representación gráfica del Árbol Binario según los datos de prueba:***



```
76  miArbol=Arbol()
77  miArbol.insert(32)
78  miArbol.insert(18)
79  miArbol.insert(54)
80  miArbol.insert(29)
81  miArbol.insert(14)
82  miArbol.insert(11)
83
84  miArbol.contains(32)
85  miArbol.contains(18)
86  miArbol.contains(54)
87  miArbol.contains(29)
88  miArbol.contains(14)
89  miArbol.contains(11)
90  
91
92
```

Run: arbolBinario x

```
"C:\Users\mauri\OneDrive\Desktop\Estructura de datos\venv\Scripts\python.exe" "C:\Users\mauri\OneDrive\Des
El valor 32 ya existe en la lista
El valor 18 ya existe en la lista
El valor 54 ya existe en la lista
El valor 29 ya existe en la lista
El valor 14 ya existe en la lista
El valor 11 ya existe en la lista

Process finished with exit code 0
```

Dicha prueba afirmaría el funcionamiento adecuado del programa, pero, también se puede corroborar imprimiendo cada nodo, siguiendo los caminos indicados por el grafico del árbol.

```
miArbol=Arbol()
miArbol.insert(32)
miArbol.insert(18)
miArbol.insert(54)
miArbol.insert(29)
miArbol.insert(14)
miArbol.insert(11)

miArbol.contains(32)
miArbol.contains(18)
miArbol.contains(54)
miArbol.contains(29)
miArbol.contains(14)
miArbol.contains(11)
miArbol.print_nodovalor()
```

```
arbolBinario x
El valor 32 ya existe en la lista
El valor 18 ya existe en la lista
El valor 54 ya existe en la lista
El valor 29 ya existe en la lista
El valor 14 ya existe en la lista
El valor 11 ya existe en la lista
Ingresa hacia que lado te quieres dirigir:
  1.-Izquierda 2.-Derecha 3.-Imprimir
3
32
```

Process finished with exit code 0

Como se mencionó en la breve explicación de nuestro método de impresión, parte de la raíz e imprime el nodo en el que se encuentra, por ello si nosotros imprimimos en la primera iteración, imprimirá la raíz es decir el 32 que fue nuestro primer dato ingresado.

```
miArbol=Arbol()
miArbol.insert(32)
miArbol.insert(18)
miArbol.insert(54)
miArbol.insert(29)
miArbol.insert(14)
miArbol.insert(11)

miArbol.contains(32)
miArbol.contains(18)
miArbol.contains(54)
miArbol.contains(29)
miArbol.contains(14)
miArbol.contains(11)
miArbol.print_nodovalor()
```

arbolBinario x

Ingresa hacia que lado te quieres dirigir:  
1.-Izquierda 2.-Derecha 3.-Imprimir  
1

Ingresa hacia que lado te quieres dirigir:  
1.-Izquierda 2.-Derecha 3.-Imprimir  
1

Ingresa hacia que lado te quieres dirigir:  
1.-Izquierda 2.-Derecha 3.-Imprimir  
1

Ingresa hacia que lado te quieres dirigir:  
1.-Izquierda 2.-Derecha 3.-Imprimir  
3  
11

Bajo la misma lógica podemos verificar la existencia del nodo que tiene el dato 11 y según el gráfico anterior podríamos apreciar que es 3 veces izquierda para estar sobre dicho nodo y nuevamente es lo que nuestra prueba retorna.