Universidad Mariano Gálvez de Guatemala / Sede Naranjo
Facultad de ingeniería en sistemas
Programación III
Ing. José Miguel Villatoro
Sección “C”

TAREA #3

Samuel Estuardo España Son 9490-22-11789
Jonathan Rogelio Herrera Soto 9490-22-11551
Sergio Enrique Sánchez Sánchez 9490-22-1077
Mario Andres Culajay Roldan 9490-22-5771

Guatemala 8 de marzo 2024

ARBOLES BINARIOS DE BUSQUEDA

La librería os es un módulo integrado en Python que proporciona una interfaz para interactuar con el sistema operativo subyacente. Permite realizar operaciones relacionadas con el sistema operativo, como manipular directorios, archivos, realizar operaciones de entrada/salida, y mucho más. En el código proporcionado, parece utilizarse para verificar la existencia de un archivo y trabajar con rutas de archivo.

La librería graphviz: Es un paquete externo que proporciona herramientas para trabajar con Graphviz, un programa de visualización de gráficos. Graphviz es utilizado para representar gráficamente estructuras de datos como árboles, grafos, y otros tipos de redes. En este código, se usa específicamente para visualizar el árbol binario de búsqueda generado por el programa.

```
import os
import graphviz
```

Clase nodoArbol define un nodo del árbol con tres atributos: izq para el hijo izquierdo, der para el hijo derecho, y valor para almacenar el valor del nodo.

```
class nodoArbol:
    def __init__(self, Valor):
        self.izq = None
        self.der = None
        self.valor = Valor
```

Clase ABB

- __init__: Inicializa el árbol con la raíz como None.
- insertar(valor): Inserta un nuevo nodo con el valor dado en el árbol.
- insertar2(valor, nodo): Método auxiliar recursivo para insertar un valor en el árbol.

```python
class ABB:
    def __init__(self):
        self.raiz = None

    def insertar(self, valor):
        self.raiz = self.insertar2(valor, self.raiz)

    def insertar2(self, valor, nodo):
        if nodo is None:
            return nodoArbol(valor)
        if valor < nodo.valor:
            nodo.izq = self.insertar2(valor, nodo.izq)
        elif valor > nodo.valor:
            nodo.der = self.insertar2(valor, nodo.der)
        return nodo
```

- mostrar(nodo): Método para mostrar los nodos y sus relaciones (padre-izquierdo y padre-derecho) en el árbol.

```python
def mostrar(self, nodo):
    if nodo is not None:
        if nodo.izq is not None:
            print(nodo.valor, "->", nodo.izq.valor)
        if nodo.der is not None:
            print(nodo.valor, "->", nodo.der.valor)
        self.mostrar(nodo.izq)
        self.mostrar(nodo.der)
```

- CargarArchivo(): Lee un archivo de texto que contiene valores separados por líneas y los inserta en el árbol.

```python
def CargarArchivo(self):
    archivo = input("\nPor favor, arrastre su archivo o digite su ruta: ").strip("\"' &")
    if not os.path.isfile(archivo):
        print("El archivo no existe.")
    else:
        with open(archivo, 'r') as file:
            for linea in file:
                arbol.insertar(int(linea.strip()))
        print("\nInsercion de los datos del archivo exitosa: ")
        arbol.mostrar(arbol.raiz)
```

- buscar(valor, nodo, profundidad=0): Busca un valor dado en el árbol y devuelve el nodo, el valor encontrado y la profundidad en la que se encuentra.

- buscar_nodo_profundidad(valor): Método de conveniencia para buscar un valor y obtener el nodo, el valor y la profundidad.

```
def buscar(self, valor, nodo, profundidad=0):
    if nodo is None or nodo.valor == valor:
        return nodo, nodo.valor if nodo else None, profundidad
    if valor < nodo.valor:
        return self.buscar(valor, nodo.izq, profundidad+1)
    return self.buscar(valor, nodo.der,profundidad+1)

def buscar_nodo_profundidad(self, valor):
    nodo, valor_encontrado, profundidad = self.buscar(valor, self.raiz)
    return nodo, valor_encontrado, profundidad
```

- eliminar(valor): Elimina un nodo con el valor dado del árbol.

- eliminar_nodo(nodo, valor): Método auxiliar recursivo para eliminar un nodo del árbol.

```
def eliminar(self, valor):
    self.raiz = self.eliminar_nodo(self.raiz, valor)

def eliminar_nodo(self, nodo, valor):
    if nodo is None:
        return nodo
    # Busca el nodo a eliminar
    if valor < nodo.valor:
        nodo.izq = self.eliminar_nodo(nodo.izq, valor)
    elif valor > nodo.valor:
        nodo.der = self.eliminar_nodo(nodo.der, valor)
    else:  # Encontr\ufffd el nodo a eliminar
        # Caso 1: Nodo es una hoja o tiene un solo hijo
        if nodo.izq is None:
            return nodo.der
        elif nodo.der is None:
            return nodo.izq
        # Caso 2: Nodo tiene dos hijos
        # Encuentra el sucesor inmediato (el menor valor en el sub\ufffdrbol derecho)
        sucesor = self.min_valor_nodo(nodo.der)
        # Copia el valor del sucesor al nodo que se va a eliminar
        nodo.valor = sucesor.valor
        # Elimina el sucesor
        nodo.der = self.eliminar_nodo(nodo.der, sucesor.valor)
    return nodo
```

- min_valor_nodo(nodo): Encuentra el nodo con el valor mínimo en el árbol.

```
def min_valor_nodo(self, nodo):
    actual = nodo
    # Recorre el rbol hacia la izquierda para encontrar el nodo con el valor mnimo
    while actual.izq is not None:
        actual = actual.izq
    return actual
```

graficar_arbol(): Genera una representación gráfica del árbol usando Graphviz. Utiliza la biblioteca graphviz para visualizar el árbol. Crea un objeto Digraph. Utiliza un método auxiliar _graficar_nodo() recursivo para agregar nodos y bordes al objeto Digraph. Finalmente, renderiza el gráfico a un archivo PNG y lo muestra.

```
def graficar_arbol(self):
    dot = graphviz.Digraph()
    self._graficar_nodo(dot, self.raiz)
    dot.render('arbol', format='png', cleanup=True)
    os.system('arbol.png')

def _graficar_nodo(self, dot, nodo):
    if nodo is not None:
        dot.node(str(nodo.valor))
    if nodo.izq is not None:
        dot.edge(str(nodo.valor), str(nodo.izq.valor))
        self._graficar_nodo(dot, nodo.izq)
    if nodo.der is not None:
        dot.edge(str(nodo.valor), str(nodo.der.valor))
        self._graficar_nodo(dot, nodo.der)
```

Muestra un menú de opciones: insertar, buscar, eliminar, cargar archivo, graficar árbol y salir.

Dependiendo de la opción elegida por el usuario, invoca los métodos correspondientes de la clase ABB.

```
arbol = ABB()
opc = 0
while(opc != 5):
    print("\n-------Menu-------")
    print("1. Insertar")
    print("2. Buscar")
    print("3. Eliminar")
    print("4. Cargar Archivo")
    print("5. Graficar Árbol")
    print("6. Salir")
    opc = int(input("Ingrese su opcion: "))
    if(opc==1):
      numero=int(input("Ingrese el valor a insertar en el arbol: "))
      arbol.insertar(numero)
      arbol.mostrar(arbol.raiz)
    elif(opc==2):
        numero = int(input("Ingrese el valor a buscar en el arbol: "))
        nodo_encontrado, valor_encontrado, profundidad = arbol.buscar_nodo_profundidad(numero)
        if nodo_encontrado:
            print(f"El valor {valor_encontrado} se encontro en el arbol en la profundidad {profundidad}.")
        else:
            print(f"El valor {numero} no se encontro en el arbol.")
    elif(opc==3):
        numero = int(input("Ingrese el valor a eliminar del arbol: "))
        arbol.eliminar(numero)
        arbol.mostrar(arbol.raiz)
    elif(opc == 4):
        arbol.CargarArchivo()
    elif(opc ==5):
        arbol.graficar_arbol()
    else:
        break;
```