2. BIGTREE

BigTree es una librería de python útil para manipular estructuras de datos en forma de árboles jerárquicos, no necesariamente binarios, pues cada nodo puede tener varios hijos. Es una buena herramienta para representar jerarquías organizacionales, estructuras de almacenamiento o de carpetas, árboles de decisión, árboles genealógicos, etc. Además, esta librería es compatible con estructuras de datos como dict y pandas, las cuales son comúnmente utilizadas en python.

BigTree funciona con una clase Node, donde cada uno de sus objetos Node representa un nodo del árbol. Cada nodo debe contar con un nombre o más atributos según el caso y puede tener un nodo padre (si no tiene, ese nodo es la raíz) o nodos hijos (pueden ser cero o más).

Algunas funciones:

1. Imprimir el árbol para comprenderlo de manera visual.
2. Buscar y encontrar nodos por su nombre (puede ser utilizando también su posición respecto al nodo actual).
3. Manipular los nodos, ya sea moviéndolos, eliminándolos o agregándolos en el árbol.
4. Permite convertir los árboles en estructuras más sencillas para la exportación o importación desde archivos, como diccionarios, listas o tablas.
5. Recorrer los nodos de árbol en preorden, postorden o por niveles.

EJEMPLO:

Quisimos representar el siguiente árbol, que representa el Organigrama de una empresa comercial.

Implementación de código:

```python
from bigtree import Node, print_tree

# Crear nodos del árbol
gerentegeneral = Node("GERENTE GENERAL")                    # Nodo raíz
administrativo = Node("Dpto ADMINISTRATIVO", parent=gerentegeneral)
comercial = Node("Dpto COMERCIAL", parent=gerentegeneral)
operaciones = Node("Dpto de OPERACIONES", parent=gerentegeneral)

#DEPENDENCIAS DEL DEPARTAMENTO ADMINISTRATIVO
contabilidad = Node("CONTABILIDAD", parent=administrativo)
cartera = Node("CARTERA", parent=administrativo)
tesoreria = Node("TESORERIA", parent=administrativo)
rh = Node("RECURSOS HUMANOS", parent=administrativo)

#DEPENDENCIAS DEL DEPARTAMENTO COMERCIAL
comprasnacionales = Node("COMPRAS NACIONALES", parent=comercial)
importaciones = Node("IMPORTACIONES", parent=comercial)
ventas = Node("VENTAS", parent=comercial)

#AREAS DE VENTAS
principal = Node("SEDE PRINCIPAL", parent=ventas)
sucursales = Node("SUCURSALES", parent=ventas)
web = Node("WEB SITE", parent=ventas)

#DEPENDENCIAS DEL DEPARTAMENTO DE OPERACIONES
bodegaje = Node("BODEGAJE", parent=operaciones)
reempaque = Node("REEMPAQUE", parent=operaciones)
despachos = Node("DESPACHOS", parent=operaciones)

# Mostrar el árbol
print_tree(gerentegeneral)
```

Salida:

```
PS C:\Users\Asus> & C:/Users/Asus/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Asus/Downloads/EJEMPLOBIGTREE.py
GERENTE GENERAL
├── Dpto ADMINISTRATIVO
│   ├── CONTABILIDAD
│   ├── CARTERA
│   ├── TESORERIA
│   └── RECURSOS HUMANOS
├── Dpto COMERCIAL
│   ├── COMPRAS NACIONALES
│   ├── IMPORTACIONES
│   └── VENTAS
│       ├── SEDE PRINCIPAL
│       ├── SUCURSALES
│       └── WEB SITE
└── Dpto de OPERACIONES
    ├── BODEGAJE
    ├── REEMPAQUE
    └── DESPACHOS
```

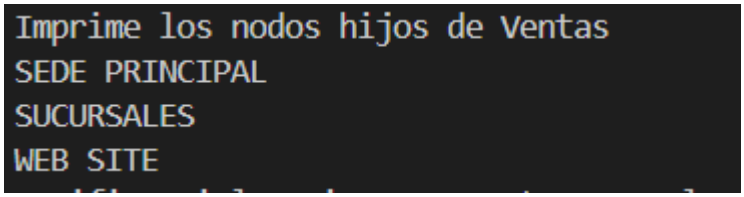
Utilizando funciones para comprender el árbol o modificarlo:

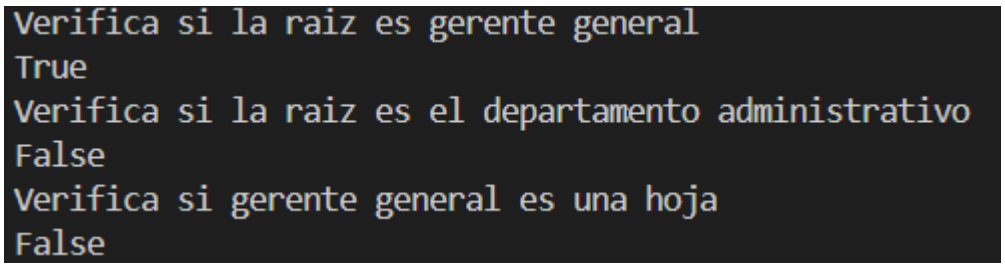
```python
print("Imprime los nodos de forma descendiente lo que está abajo de GERENTE GENERAL")
for node in gerentegeneral.descendants:
    print(node.name)

print("Imprime los nodos hijos de Gerente General")
for node in gerentegeneral.children:
    print(node.name)

print("Imprime los nodos hijos de Ventas")
for node in ventas.children:
    print(node.name)

print("Verifica si la raiz es gerente general")
print(gerentegeneral.is_root)

print("Verifica si la raiz es el departamento administrativo")
print(administrativo.is_root)

print("Verifica si gerente general es una hoja")
print(gerentegeneral.is_leaf)

#AGREGAR UN NODO HIJO A VENTAS
nuevo = Node("PROMOS")
nuevo.parent = ventas

#ELIMINAR EL NODO WEB
web.parent = None

print("Imprime el nuevo árbol")
print_tree(gerentegeneral)
```

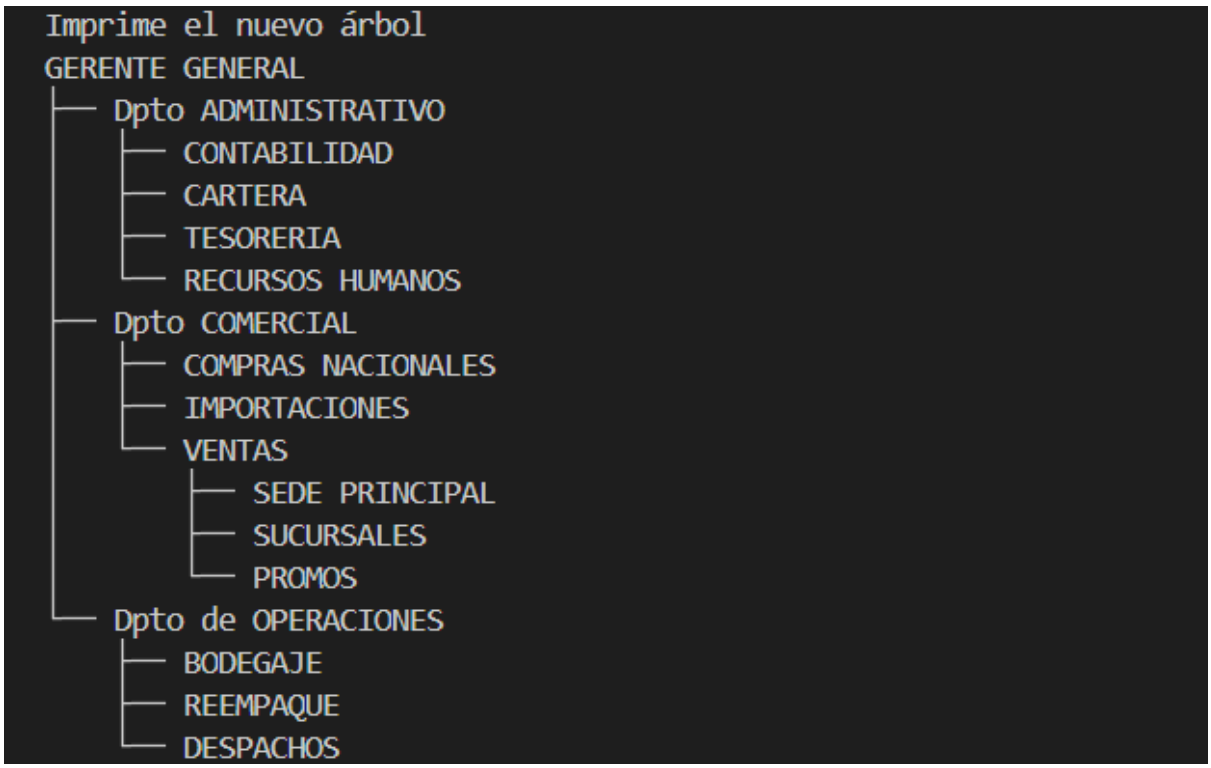
Salida:

```
Imprime los nodos de forma descendiente lo que está abajo de GERENTE GENERAL
Dpto ADMINISTRATIVO
CONTABILIDAD
CARTERA
TESORERIA
RECURSOS HUMANOS
Dpto COMERCIAL
COMPRAS NACIONALES
IMPORTACIONES
VENTAS
SEDE PRINCIPAL
SUCURSALES
WEB SITE
Dpto de OPERACIONES
BODEGAJE
REEMPAQUE
DESPACHOS
```

```
Imprime los nodos hijos de Gerente General
Dpto ADMINISTRATIVO
Dpto COMERCIAL
Dpto de OPERACIONES
```







3. ÁRBOLES DE HUFFMAN

Los árboles de Huffman son estructuras de datos en forma de árbol binario que tiene cada uno de los símbolos por hoja. Son utilizados para realizar una compresión de datos sin pérdida, reduciendo el tamaño de los datos sin eliminar o modificar la información original. Este tipo de árbol fue propuesto en 1952 por David A. Huffman, con la intención de que se pudiera codificar información de manera óptima, asignando códigos binarios más cortos a los símbolos más frecuentes y códigos más largos a los menos frecuentes, logrando así representar los datos con la menor cantidad de bits posible. Es construido de manera de que al recorrerlo desde la raíz a cada una de sus hojas se obtiene el código asociado a él.

La técnica de los árboles de Huffman o codificación de Huffman es útil en áreas como la compresión de imágenes, audio, video y texto, o en transmisión de datos.

La construcción de un árbol de Huffman funciona de la siguiente manera:

1. Se identifican los símbolos del conjunto de datos y se cuenta la frecuencia con la que aparece cada uno.
2. Se crea inicialmente un árbol sin hijos para cada símbolo, etiquetado con el símbolo y frecuencia correspondiente.
3. Se ordenan todos los árboles creados según su frecuencia, desde los de menor frecuencia hasta los de mayor. Luego se van seleccionando los dos árboles con menor frecuencia y se unen en un nuevo árbol. Este nuevo árbol tiene como raíz un nodo cuya frecuencia es la suma de las frecuencias de sus dos hijos. Uno se ubica como hijo izquierdo y el otro como derecho.
4. Para el árbol construido, se le da el valor de 0 a la rama del hijo izquierdo y 1 a la rama del nodo hijo derecho, para ser la base de los códigos binarios que se le asignan a cada símbolo.
5. El nuevo árbol se inserta en la lista y el proceso se repite: siempre combinando los dos árboles de menor frecuencia hasta que finalmente solo quede uno. Ese árbol final es el Árbol de Huffman completo, cuya raíz representa el conjunto total de datos.

Una vez construido, cada símbolo se representa recorriendo el árbol desde la raíz hasta su hoja. Así, los símbolos más frecuentes quedan representados con códigos binarios más cortos al estar más cerca de la raíz, y los menos frecuentes con códigos más largos al estar más alejados, optimizando el tamaño total de toda la información.

Para obtener el código binario exacto de un símbolo, se puede partir desde la hoja que lo representa y subir hasta la raíz del árbol, anotando un 0 cuando se sube por una rama izquierda y un 1 cuando se sube por la derecha. Al finalizar el recorrido, se invierte la secuencia obtenida para formar el código Huffman del símbolo. De forma inversa, al decodificar un mensaje, se inicia desde la raíz y se siguen las ramas según cada bit del código hasta llegar a una hoja, que sería a la que corresponde al símbolo original. En aplicaciones prácticas, estos códigos suelen guardarse en tablas para facilitar su uso sin necesidad de recorrer el árbol de manera constante.