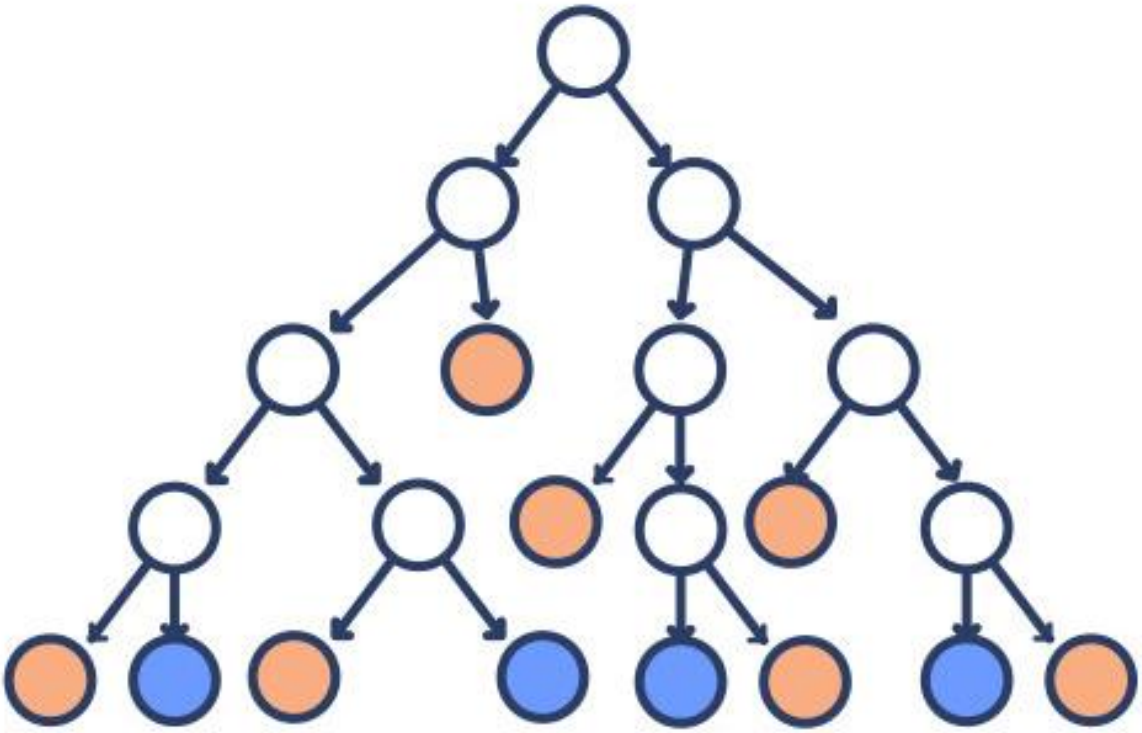
TP Integrador

Arboles con LISTAS



Alumnos: Chiavón, Cristian - Chiavón, Facundo

Materia: Programación I

Profesor: Enferrel, Ariel

Tutor: Zabala, Martina

Contenido

1. Introducción

En esta presentación, abordaremos el concepto general de los árboles, definiendo su terminología y propiedades fundamentales. Nos centraremos luego en los árboles binarios, analizando su estructura y beneficios específicos. La implementación en Python será un punto clave, demostrando cómo transformar estos conceptos teóricos en código funcional mediante casos prácticos.

2. Marco Teórico

¿Qué es un Árbol?

Un árbol es una estructura de datos que organiza la información de forma jerárquica similar a un árbol genealógico. Los datos que se ingresan son llamados “nodos” y están conectados por aristas. Existe un nodo Raíz del cual se desprenden luego los demás nodos llamados “hijos” o subnodos. Cada subnodo puede tener a su vez sus propios hijos y un nodo sin hijos es llamado “hoja”.

En todo árbol podemos determinar:

- **Su Grado**: El grado de un nodo indica la cantidad de hijos que posee, mientras que el grado del árbol indica la cantidad máxima de hijos que un nodo puede tener.
- **Su Peso**: El peso es el número total de nodos que posee un árbol.
- **Su Altura o Profundidad**: La profundidad de un nodo está indicada por la cantidad de aristas que hay desde el nodo hasta la Raiz.
- **Sus Niveles**: Los niveles de un árbol están comparados normalmente con la profundidad máxima del árbol.

Para este trabajo utilizaremos árboles binarios. Este tipo de árbol se caracteriza por ser de grado 2 simplificando las operaciones y haciéndolos muy eficientes para búsquedas y ordenamiento.

Los árboles binarios pueden ser de 4 tipos.

- Completo
- Lleno
- Perfecto
- Sesgado.

Y se pueden recorrer de 3 maneras diferentes:

- **In-Orden** (Hijo Izquierdo - Nodo – Hijo Derecho)
- **Pre-Orden** (Nodo – Hijo Izquierdo - Hijo Derecho)
- **Post-Orden** (Hijo Izquierdo - Hijo Derecho - Nodo)

Algunas aplicaciones de árboles:

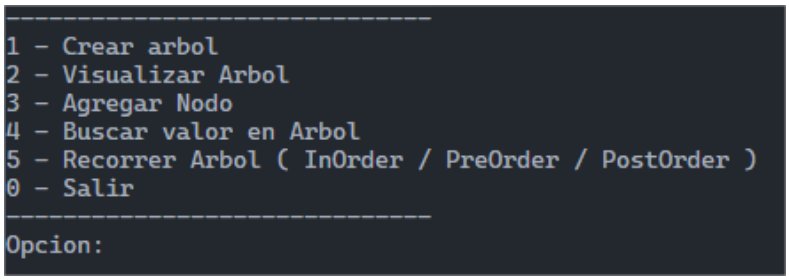
Bases de datos: Para organizar y buscar información de forma eficiente.

Sistemas de archivos: Modelan la estructura de directorios y archivos.

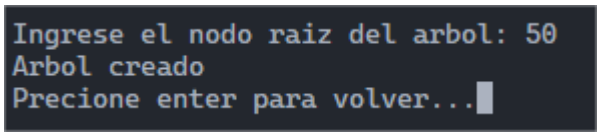
Compresión de datos: Algoritmos como Huffman usan árboles binarios para codificar información.

3. Caso Práctico

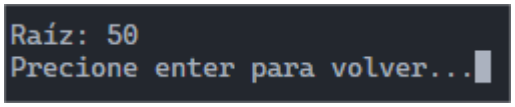
Se implementa un árbol binario simple en Python, creando el árbol con el nodo raíz, insertando nodos manualmente, visualizando su contenido, verificando la existencia de un valor dentro del árbol y recorriendo su información mediante las distintas maneras.



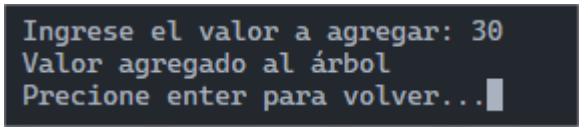
- Menú al iniciar el programa

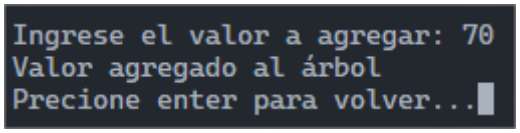


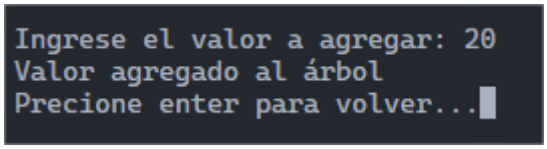
- Creamos el árbol con la opción 1 ingresando el valor de la Raíz.



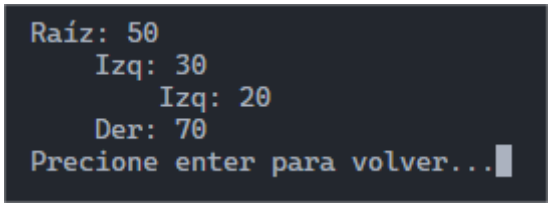
- Visualizamos el árbol con la opción 2. Vemos que solo existe la raíz.







- Agregamos los valores 30, 70 y 20 al árbol con la opción 3.



- Volvemos a visualizar el árbol comprobando que los valores fueron agregados correctamente.

```
Ingrese el valor a buscar: 60
El valor 60 NO se encuentra en el arbol
Precione enter para volver...
```

- Buscamos el valor 60 en el árbol y el resultado es el esperado, efectivamente el 60 no se encuentra en nuestro árbol aún.

```
Ingrese el valor a agregar: 60
Valor agregado al árbol
Precione enter para volver...
```

```
Ingrese el valor a buscar: 60
El valor 60 se encuetra en el árbol
Precione enter para volver...
```

- Agregamos el valor 60 con la opción 2 y buscamos nuevamente dicho valor. Nuevamente la respuesta es la esperada dado que ahora si encontramos el valor dentro del árbol.

```
Raíz: 50
    Izq: 30
        Izq: 20
    Der: 70
        Izq: 60
Precione enter para volver...
```

- Visualizamos nuevamente nuestro árbol para corroborarlo.

```
--- Recorrido InOrder ---
20 - 30 - 50 - 60 - 70 -

--- Recorrido PreOrder ---
50 - 30 - 20 - 70 - 60 -

--- Recorrido PostOrder ---
20 - 30 - 60 - 70 - 50 -

Precione enter para volver...
```

- Recorremos el árbol de las 3 formas mostrando la secuencia en cada una. Los resultados son los esperados.

```
------------------------------
Seleccione una opción:
------------------------------
1 - Crear arbol
2 - Visualizar Arbol
3 - Agregar Nodo
4 - Buscar valor en Arbol
5 - Recorrer Arbol ( InOrder / PreOrder / PostOrder )
0 - Salir
------------------------------
Opcion: 0
------------------------------
Saliendo del sistema....
```

- Finalizamos el sistema con 0

4. Resultados Obtenidos

- El árbol binario se construye correctamente a medida que se ingresan valores.
- La visualización del árbol es correcta e intuitiva.
- Los recorridos Inorden, Preorden y Postorden se realizaron de manera correcta.
- La búsqueda de un valor informa correctamente si el dato se encuentra o no en el árbol.

5. Conclusiones

El uso de árboles binarios en Python facilita la organización de datos de manera eficiente. Los algoritmos de recorrido nos permiten explorar su estructura según las necesidades específicas de cada problema, demostrando su adaptabilidad y utilidad práctica.

Reflexión: Aprender e implementar estructuras de datos como los árboles no solo mejora nuestras habilidades de programación, sino que también nos entrena a pensar de manera más estructurada y lógica para abordar y resolver problemas computacionales complejos de forma efectiva.

6. Bibliografía

- Documentación de Python: https://docs.python.org/3/tutorial/classes.html
- Treeconverter (herramienta online para visualizar árboles Binarios): https://treeconverter.com/
- Implementación de árboles como listas anidadas (Material de la TUP): https://youtu.be/-D4SxeHQGIg?si=8XpE3JNg24cuOujJ