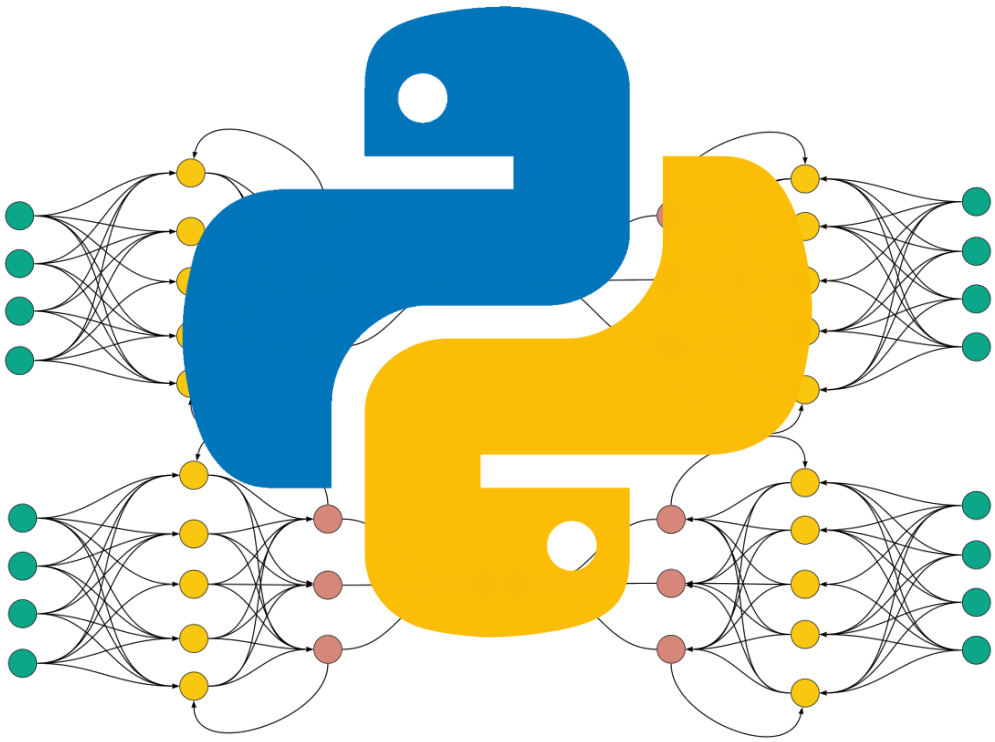
Árboles en Python



Alumnos:
Williams Exequiel Cantero – willy_cantero@outlook.com
José Juan Abad Correa – abadjuan263@gmail.com

Materia: **Programación I**
Profesores: Prof. Cinthia Rigoni
Tutor Oscar Londero

Fecha de entrega: 9 de Junio 2.025

Contenido

INTRODUCCION

Las estructuras de datos son fundamentales en la programación, ya que permiten organizar y manejar la información de forma eficiente. En este trabajo se abordará una estructura en particular: los árboles.

Un árbol es una estructura jerárquica compuesta por nodos conectados entre sí. Se utiliza en diversos ámbitos de la informática, como motores de búsqueda, bases de datos y sistemas de archivos. En este trabajo, los árboles serán representados mediante listas anidadas, una herramienta que se ha trabajado y afianzado durante el cursado.

Como futuros técnicos en programación, es importante conocer para qué sirven los árboles, porque permiten resolver problemas complejos relacionados con la organización, búsqueda y procesamiento de datos.

A lo largo del trabajo se explicarán sus propiedades, los distintos tipos de recorridos, y se hará foco en el funcionamiento de un árbol binario de búsqueda. El propósito del trabajo es comprender cómo funcionan los árboles y aplicar ese conocimiento en un programa en Python que construya un árbol binario de búsqueda a partir de una lista de datos ingresada por el usuario.

OBJETIVOS

- Entender qué son los árboles como estructura de datos.
- Conocer las propiedades principales de los árboles.
- Explicar los diferentes tipos de recorridos que se pueden realizar en un árbol.
- Entender el concepto de un árbol binario de búsqueda y como funciona.
- Hacer un programa en Python que arme un árbol binario de búsqueda con una lista de datos que ingresa el usuario.
- Practicar la representación de árboles usando listas anidadas

MARCO TEORICO

ARBOL

Un árbol es una estructura de datos no lineal que sirve para representar relaciones jerárquicas de manera eficiente.

ELEMENTOS DE UN ARBOL

Un árbol se compone por nodos conectados entre sí por arcos o ramas.

Un nodo es la unidad básica del árbol y tiene dos componentes principales: un valor y un puntero a sus nodos hijos si es que los tiene.

Un árbol puede estar vacío y un árbol no vacío puede constar de un solo elemento (nodo raíz) o bien de varios elementos.

CLASIFICACION DE LOS NODOS

1) SEGÚN SU UBICACIÓN EN EL ARBOL:
   - Nodo Raíz: es el nodo principal del árbol, el único que no tiene padre. Cada árbol tiene un solo nodo raíz.
   - Nodos rama o interno: es cualquier nodo que tiene un nodo padre y, al menos, un nodo hijo.

- Nodos hoja: es cualquier nodo que no tenga hijos, es decir, un nodo terminal del árbol. Los nodos hoja están en los extremos de la estructura de un árbol

2) SEGÚN SU RELACION CON OTROS NODOS:
   - Nodo padre: es aquél que tiene uno o más nodos hijos conectados a él. Cada nodo, excepto la raíz, tiene solamente un nodo padre.
   - Nodo hijo: es aquél que está conectado con un nodo padre de modo que se encuentra directamente debajo de él en la jerarquía. Puede ser padre de otros nodos.
   - Nodo hermano: son aquellos que comparten el mismo nodo padre. Están en el mismo nivel en la jerarquía del árbol.

PROPIEDADES DE LOS ARBOLES:

- Longitud de camino: es el número de ramas que hay que transitar para llegar de un nodo a otro.
- Profundidad: es la longitud de camino entre un nodo y el nodo raíz.
- Nivel: el nivel de un nodo es la longitud del camino que lo conecta al nodo raíz más uno.
- Altura: es el máximo nivel del árbol.
- Grado: el grado de un nodo es el número de hijos que tiene dicho nodo. El grado de un árbol es el grado máximo de los nodos del árbol.
- Orden: es la máxima cantidad de hijos que puede tener cada nodo. Se establece como restricción antes de construir el árbol.
- Peso: es el número total de nodos que tiene un árbol.

ARBOLES BINARIOS

Son árboles en los que cada nodo puede tener, como máximo, 2 hijos. En un árbol binario el nodo de la izquierda representa al hijo izquierdo y el nodo de la derecha representa al hijo derecho.

FORMAS DE RECORRER ABOLES BINARIOS:

Existen varias formas de recorrer un árbol. Las tres formas más comunes son preorden, inorden y postorden.

1) PREORDEN

El recorrido preorden es útil cuando necesitas realizar alguna operación en el nodo antes de procesar a sus hijos.

Comienza por la raíz, luego recorre recursivamente el subárbol izquierdo y para finalizar recorre recursivamente el subárbol derecho.

2) INORDEN

El recorrido inorden es especialmente útil para los árboles de búsqueda binaria, ya que garantiza que los nodos se visiten en orden ascendente.

Comienza recorriendo recursivamente el árbol izquierdo, luego la raíz y finaliza recorriendo recursivamente el árbol derecho.

3) POSTORDEN

El recorrido postorden es útil cuando se necesita realizar alguna acción en los nodos hijos de un nodo antes de procesar el nodo mismo.

Comienza recorriendo recursivamente el subárbol izquierdo, luego recorre recursivamente el subárbol derecho y finaliza con la raíz.

ARBOLES BINARIO DE BUSQUEDA

Los árboles binarios de búsqueda son árboles binarios donde para cada nodo del árbol se cumple que:

- Los valores de todos los nodos en su subárbol izquierdo son menores que el valor del nodo.
- Los valores de todos los nodos en su subárbol derecho son mayores que el valor del

nodo.

CONSTRUCCION DE UN ARBOL BINARIO DE BUSQUEDA

Si el árbol está vacío, se crea un nuevo nodo con el elemento a insertar, el cual se convierte en la raíz. En caso contrario, se compara el valor con el de la raíz: si es menor, la inserción continúa en el subárbol izquierdo; si es mayor, en el subárbol derecho. Este proceso se repite de forma recursiva hasta encontrar una posición vacía en la que ubicar el nuevo nodo.

BUSQUEDA EN UN ARBOL BINARIO DE BUSQUEDA

Consiste en comparar el elemento a encontrar con el valor de la raíz. Si coinciden, la búsqueda concluye con éxito. Si el elemento es menor, la búsqueda continua en el subárbol izquierdo; si es mayor, en el subárbol derecho. Si se alcanza un nodo hoja sin encontrar el elemento, el valor no existe en el árbol.

ELIMINAR UN NODO EN UN ARBOL BINARIO DE BUSQUEDA:

A la hora de eliminar un nodo de un árbol de búsqueda, se debe proceder de distintas formas dependiendo de si el nodo tiene o no hijos.

Caso 1: nodo hoja

Si el nodo a eliminar es un nodo hoja, podemos eliminarlo directamente.

Caso 2: nodo con un solo hijo

Si el nodo a eliminar tiene un solo hijo, simplemente reemplazamos el nodo con su hijo. Esto es porque, en un árbol binario de búsqueda, el hijo de un nodo se ajusta de forma que mantiene la propiedad de orden del árbol.

Caso 3: nodo con dos hijos

Si el nodo a eliminar tiene dos nodos hijos, hay que reemplazar el nodo con uno de sus nodos hijos que mantenga la propiedad de un árbol binario de búsqueda. Para mantener

dicha propiedad, deberemos reemplazarlo con el nodo hijo que tenga el valor más grande.

CASO PRACTICO:

Se elaboró un programa en Python que le solicita al usuario una lista de números enteros a partir de la cual se crea un árbol binario de búsqueda representado por listas anidadas. El programa cuenta con funciones para crear el árbol, insertar elementos, imprimir recorridos, mostrar propiedades del árbol e informar si cierto número se encuentra en el árbol.

METODOLOGIA UTILIZADA

- Se utilizó el lenguaje Python 3.11.9
- Cada nodo del árbol se representa como una lista de tres elementos:
  1. Valor del nodo.
  2. Subárbol izquierdo.
  3. Subárbol derecho.
- Se desarrollaron funciones para crear árboles, insertar nodos y recorrerlos.
- Se incluyó una función de impresión de propiedades de árboles.
- Se implementó una función que informa si el numero pertenece al árbol.

DESARROLLO / IMPLEMENTACION

A continuación, se presenta el código completo para implementar un árbol binario de búsqueda utilizando únicamente listas:

```
#---------------------------------FUNCIONES----------------------------

#Primero agregamos la raiz que es el primer elemento, despues recursivamente
vamos agregando a la izquierda
# o derecha segun corresponda
def agregar_al_arbol(arbol, nodo):
    if arbol == []:
        return [nodo, [], []]

    raiz = arbol[0]

    if nodo < raiz:
        arbol[1] = agregar_al_arbol(arbol[1], nodo) #arbol [1] representa
la izquierda
    else:
        arbol[2] = agregar_al_arbol(arbol[2], nodo) #arbol [2] representa
la derecha
    return arbol

#Crea el arbol y por cada nodo de la lista llama a la funcion agregar para
que lo agregue donde corresponda
def crear_arbol(nodos):
    arbol = [] #Empezamos con el arbol vacio
    for nodo in nodos:
        arbol = agregar_al_arbol(arbol, nodo)
    return arbol


#FUNCIONES PARA IMPRIMIR RECORRIDOS
#Imprime primero la raiz, luego el subarbol izquierdo y por ultimo el
subarbol derecho.
def imprimir_preorden(arbol):
     if arbol != []:
        print(arbol[0], end=" ")
        imprimir_preorden(arbol[1])
        imprimir_preorden(arbol[2])
```

```python
#Imprime primero el subarbol izquierdo, luego la raiz y por ultimo el subarbol derecho
def imprimir_inorden(arbol):
    if arbol != []:
        imprimir_inorden(arbol[1])
        print(arbol[0], end=" ")
        imprimir_inorden(arbol[2])

#Imprimero primero el subarbol izquierdo, luego el subarbol derecho y por ultimo la raiz.
def imprimir_postorden(arbol):
    if arbol != []:
        imprimir_postorden(arbol[1])
        imprimir_postorden(arbol[2])
        print(arbol[0], end=" ")

#FUNCIONES DE PROPIEDADES DE ARBOLES

#Sumamos 1 por cada nodo que se encuentre recursivamente
def peso(arbol):
    if arbol == []:
        return 0
    return 1 + peso(arbol[1]) + peso(arbol[2])

#Sumamos 1 de la raiz, mas la altura maxima encontrada de los subarboles
def altura(arbol):
    if arbol == []:
        return 0
    return 1 + max(altura(arbol[1]), altura(arbol[2]))

def informar_propiedades(arbol):
    print("--- PROPIEDADES DEL ÁRBOL ---")
    print(f"Peso del árbol (cantidad de nodos): {peso(arbol)}")
    print(f"Altura del árbol: {altura(arbol)}")
    print("Orden del árbol: 2 (es un árbol binario)")

#Funcion para buscar un numero en el arbol
def buscar(arbol, numero):
```

```
    if arbol == []: #Si el arbol esta vacio devuelve falso
        return False
     if arbol[0] == numero: #Si el numero es igual a la raiz devuelve
verdadero
        return True
    elif numero < arbol[0]: #Si es menor a la raiz busca recursivamente en
los subarboles izquierdos
        return buscar(arbol[1], numero)
    else:
          return buscar(arbol[2], numero) #Si es mayor a la raiz busca
recursivamente en los arboles derechos


#----------------------------PROGRAMA PRINCIPAL-------------------------

from Funciones import crear_arbol
from Funciones import imprimir_preorden, imprimir_inorden,
imprimir_postorden
from Funciones import informar_propiedades
from Funciones import buscar


print("BIENVENIDOS A ARBOLES BINARIOS DE BUSQUEDA CON PYTHON")
cantidad = int(input("¿Cuantos nodos queres que tenga tu arbol? "))
lista_nodos = []

# Usamos while porque no sabemos cuantas veces se va a ejecutar ya que
puede haber repeticion de nodos.
#Mientras la longitud de la lista arbol sea menor a la cantidad de nodos
que debe tener, el arbol sigue pidiendo nodos.
while len(lista_nodos) < cantidad:
    nodo = int(input(f"Ingresa el nodo {len(lista_nodos) + 1} de tu árbol:
"))
    #if para verificar si ese nodo ya se ingreso
    if nodo in lista_nodos:
        print("Ese nodo ya existe. Ingresá un valor distinto.")
    else:
        lista_nodos.append(nodo)
print(f"Estos son los nodos de tu arbol: {lista_nodos}")
```

```
print("Ahora vamos a crear tu arbol binario de busqueda!")
arbol_creado = crear_arbol(lista_nodos)
print(f"Asi quedo tu arbol creado: {arbol_creado}")

#Preguntamos en que orden de recorrido quiere imprimir su arbol
opcion = int(input(
    "¿Cómo querés que recorra tu árbol?\n"
    "1. Preorden\n"
    "2. Inorden\n"
    "3. Postorden\n"
    "Ingresá el número de la opción seleccionada: "
))

#Verificamos que la opcion elegida sea posible.
while opcion != 1 and opcion != 2 and opcion != 3:
    print("Ingresa una opcion valida: ")
    opcion = int(input())

#Llamamos a la funcion de imprimir que corresponda segun la opcion elegida
if opcion == 1:
    print("El recorrido preorden de tu arbol es el siguiente: ")
    imprimir_preorden(arbol_creado)
elif opcion == 2:
    print("El recorrido inorden de tu arbol es el siguiente: ")
    imprimir_inorden(arbol_creado)
else:
    print("El recorrido postorden de tu arbol es el siguiente: ")
    imprimir_postorden(arbol_creado)

#Mostramos propiedades del arbol creado
print("") #salto de linea
informar_propiedades(arbol_creado)

#Por ultimo pedimos un numero para buscar en el arbol creado
numero_a_buscar = int(input("Ingresa un numero para buscar: "))
if buscar(arbol_creado, numero_a_buscar):
```

```
    print (f"El numero {numero_a_buscar} si se encuentra en el abol!")
else:
    print(f"El numero {numero_a_buscar} no se encuentra en el abol!")
```

Captura de pantalla del programa en ejecución:

```
$ python3 Principal.py
BIENVENIDOS A ARBOLES BINARIOS DE BUSQUEDA CON PYTHON
¿Cuantos nodos queres que tenga tu arbol? 7
Ingresa el nodo 1 de tu árbol: 5
Ingresa el nodo 2 de tu árbol: 3
Ingresa el nodo 3 de tu árbol: 7
Ingresa el nodo 4 de tu árbol: 2
Ingresa el nodo 5 de tu árbol: 8
Ingresa el nodo 6 de tu árbol: 6
Ingresa el nodo 7 de tu árbol: 4
Estos son los nodos de tu arbol: [5, 3, 7, 2, 8, 6, 4]
Ahora vamos a crear tu arbol binario de busqueda!
Asi quedo tu arbol creado: [5, [3, [2, [], []], [4, [], []]], [7, [6, [], []], [8, [], []]]]
¿Cómo querés que recorra tu árbol?
1. Preorden
2. Inorden
3. Postorden
Ingresá el número de la opción seleccionada: 4
Ingresa una opcion valida:
1
El recorrido preorden de tu arbol es el siguiente:
5 3 2 4 7 6 8
--- PROPIEDADES DEL ÁRBOL ---
Peso del árbol (cantidad de nodos): 7
Altura del árbol: 3
Orden del árbol: 2 (es un árbol binario)
Ingresa un numero para buscar: 5
El numero 5 si se encuentra en el abol!
```

RESULTADOS

El programa desarrollado logró construir y recorrer un árbol binario de búsqueda en Python utilizando listas anidadas para representar cada nodo y sus conexiones. Cada nodo se estructuró como una lista de tres elementos: el valor, su subárbol izquierdo y su subárbol derecho. Esta representación permitió implementar de manera sencilla las funciones de creación, inserción, recorrido y cálculo de propiedades.

Se eligió trabajar con listas porque es una forma simple y directa de representar árboles sin necesidad de usar clases ni objetos, lo que reduce la complejidad del código para quienes están iniciando en programación.

Ventajas observadas al utilizar listas:

- Código simple y fácil de entender.
- Permite concentrarse en cómo funciona un árbol sin distraerse con detalles más avanzados.
- Es útil para afianzar los conceptos teóricos con una práctica concreta.

Desventajas:

- No es una solución escalable para proyectos grandes.
- Al no usar clases, se pierde encapsulamiento y reutilización de código

CONCLUSION

A lo largo de este trabajo se logró comprender y aplicar el concepto de árboles como estructura de datos, concentrándose en los árboles binarios de búsqueda. Se estudiaron sus elementos, propiedades y formas de recorrido, lo que permitió afianzar los conocimientos teóricos mediante una implementación práctica en Python.

El uso de listas anidadas como forma de representar los arboles resultó ser una herramienta sencilla pero efectiva para construir y recorrer árboles, lo que facilitó la programación sin necesidad de estructuras más complejas como clases. Esto es ideal para quienes están dando sus primeros pasos en la programación.

Además, el trabajo permitió integrar todos los temas que se vieron a lo largo del cursado.

En resumen, este trabajo integrador demostró que los árboles son herramientas fundamentales para organizar y procesar datos, y que su implementación puede ser accesible incluso para principiantes en programación.

BIBLIOGRAFIA

- Material de cátedra: Unidad “Estructuras de datos avanzadas”, Programación I, UTN.
- Árbol binario de búsqueda: https://es.wikipedia.org/wiki/%C3%81rbol_binario_de_b%C3%BAsqueda