

Estructuras de Datos Avanzados

Algoritmo de Búsqueda en Árbol Binario

Introducción:

En el ámbito de la programación, los Tipos Abstractos de Datos (TAD) son esenciales para gestionar información de manera eficiente, especialmente cuando se requiere organizar, buscar y manipular datos de forma jerárquica. Entre estas estructuras, los árboles destacan por su versatilidad, siendo ampliamente utilizados en algoritmos de búsqueda, bases de datos, inteligencia artificial, etc.

En este trabajo, abordamos la implementación manual de un árbol binario, donde los nodos son ingresados en forma manual y posteriormente se busca un número específico, ingresado por el usuario, mediante un algoritmo de recorrido. El objetivo es demostrar cómo los árboles binarios optimizan la búsqueda de información y cómo su lógica recursiva facilita el manejo de datos jerárquicos.

Marco Teórico:

Árboles Binarios: Definición y Características

Un árbol binario es un tipo particular de árbol ordenado que sigue tres principios fundamentales:

1. Cada nodo tiene como máximo dos hijos (denominados hijo izquierdo e hijo derecho).
2. El hijo izquierdo precede al hijo derecho en el orden de los hijos de un nodo.
3. Los subárboles izquierdo y derecho son, a su vez, árboles binarios, lo que permite una organización recursiva de los datos.

Propiedades clave de los arboles:

- Raíz: nodo inicial (no tiene padre).
- Hojas: nodos sin hijos.
- Altura: número de niveles desde la raíz hasta la hoja más profunda.
- Profundidad: la longitud del camino desde la raíz hasta el nodo.
- Subárboles: cualquier nodo junto a sus descendientes forma un subárbol.

Los arboles binarios pueden recorrerse de tres maneras: Preorden, Inorden y Postorden

La búsqueda dentro del árbol binario funciona de la siguiente manera:

COMISIÓN: 9

1. Se compara el valor ingresado con el nodo actual.
2. Si no son iguales continua la búsqueda en un subárbol descartando el otro subárbol .
3. se repite la búsqueda en forma recursiva.

Caso Práctico:

Se desarrolló un programa en Python para buscar un número dentro de un árbol binario

En primer lugar, se construye de forma manual el árbol binario, ingresando cada nodo y sus hijos. Luego se pide al usuario ingresar un número a buscar.

```
class Nodo:

    def __init__(self, dato):

        self.dato = dato

        self.izquierda = None

        self.derecha = None

#Se hace un recorrido inOrden

def recorrer(nodo):

    if nodo is None:

        return

    recorrer(nodo.izquierda)

    print(nodo.dato, end=" ")

    recorrer(nodo.derecha)

#Se ingresa los valores de los nodos segun la siguiente lista

#[13,7,15,3,8,14,19,18]

raiz = Nodo(13)

nodo7 = Nodo(7)

nodo15 = Nodo(15)

nodo3 = Nodo(3)

nodo8 = Nodo(8)

nodo14 = Nodo(14)
```

COMISIÓN: 9

```
nodo19 = Nodo(19)
```

```
nodo18 = Nodo(18)
```

```
#Se le asigna un lugar a cada nodo en el arbol binario
```

```
raiz.izquierda = nodo7
```

```
raiz.derecha = nodo15
```

```
nodo7.izquierda = nodo3
```

```
nodo7.derecha = nodo8
```

```
nodo15.izquierda = nodo14
```

```
nodo15.derecha = nodo19
```

```
nodo19.izquierda = nodo18
```

```
# se recorre el arbol en forma inOrden y se imprime
```

```
print("El arbol binario recorrido inOrden queda: ")
```

```
recorrer(raiz)
```

```
print("\n")
```

```
#Busqueda de un valor dentro del arbol binario
```

```
def buscar(nodo, num):
```

```
    if nodo is None:          #si el nodo no esta retorna none
```

```
        return None
```

```
    elif nodo.dato == num:
```

```
        return nodo
```

```
    elif num < nodo.dato:
```

```
        return buscar(nodo.izquierda, num)
```

```
    else:
```

```
        return buscar(nodo.derecha, num)
```

COMISIÓN: 9

buscar un valor ingresado

```
numero=int(input("ingrese un numero a buscar en el arbol binario: "))
```

```
resultado = buscar(raiz, numero)
```

if resultado:

```
    print(f"El número {numero}, si se encuentra en el arbol binario.")
```

else:

```
    print(f"El número {numero}, no se encuentra en el arbol binario.")
```

Metodología Utilizada:

La elaboración del trabajo se realizó en las siguientes etapas:

- Recolección de información teórica en documentación confiable.
- Implementación en Python de los algoritmos estudiados.
- Pruebas con diferentes valores.
- Elaboración del informe y preparación de anexos.

Resultados Obtenidos:

- El programa mostró correctamente el recorrido Inorden del árbol binario.
- La búsqueda muestra al usuario si el número ingresado se encuentra en el árbol binario o no.
- Se comprendió la importancia en el uso eficiente de recursos cuando se realiza una búsqueda en un árbol binario.

Conclusion:

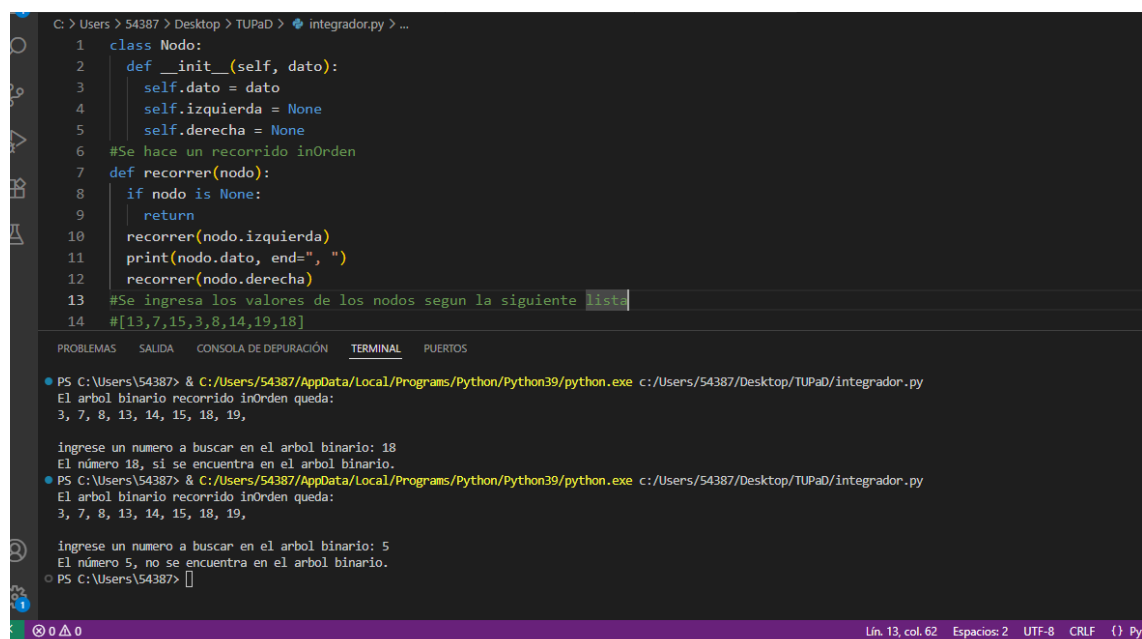
Los árboles binarios son estructuras fundamentales que permiten realizar búsquedas eficientes gracias a su propiedad de orden jerárquico. La búsqueda en un árbol binario es mucho más eficiente que una búsqueda comparando elemento a elemento en una lista de números, por ejemplo. La implementación mediante comparaciones recursivas (izquierda/derecha), logra realizar una búsqueda en menor tiempo, siempre que el árbol esté balanceado. Sin embargo, en el peor de los casos, tener un árbol desbalanceado, igualmente es conveniente el uso de este tipo de estructuras.

Bibliografía

- Estructuras de Datos y Algoritmos. Tema 4: Árboles. Departamento de Informática. Universidad de Valladolid. (2021). *Estructuras de datos y algoritmos: Tema 4 – Árboles* [Apuntes de clase]. Universidad de Valladolid.
- Gómez, S. A. (s.f.). *Estructuras de datos. Clase 12 – Árboles binarios* [Material de clase]. Universidad Nacional del Sur.
- Python Oficial: <https://docs.python.org/3/library/>
- W3S: <https://www.w3schools.com/python/default.asp>
- Juan Carlos Esteve Vargas: https://www.youtube.com/watch?v=q_nBZTwYr0o

Anexos

- Repositorio en GitHub: <https://github.com/Torres7417/UTN-TUPaD-P1/tree/main/Trabajo%20Integrador>
- Video explicativo: (se subio a Goolge Drive debido a que YouTube elimina el video) https://drive.google.com/file/d/1l1bY7_KHglucH8_7Rte7pseecsk56Uad/view
- Captura de pantalla del programa funcionando.



```
C:\Users\54387\Desktop\TUPaD> integrador.py > ...
1 class Nodo:
2     def __init__(self, dato):
3         self.dato = dato
4         self.izquierda = None
5         self.derecha = None
6     #Se hace un recorrido inOrden
7     def recorrer(nodo):
8         if nodo is None:
9             return
10        recorrer(nodo.izquierda)
11        print(nodo.dato, end=" ")
12        recorrer(nodo.derecha)
13    #Se ingresa los valores de los nodos segun la siguiente lista
14    #[13,7,15,3,8,14,19,18]

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

PS C:\Users\54387> & C:\Users\54387\AppData\Local\Programs\Python\Python39\python.exe c:/Users/54387/Desktop/TUPaD/integrador.py
El arbol binario recorrido inOrden queda:
3, 7, 8, 13, 14, 15, 18, 19,

ingrese un numero a buscar en el arbol binario: 18
El número 18, si se encuentra en el arbol binario.
PS C:\Users\54387> & C:\Users\54387\AppData\Local\Programs\Python\Python39\python.exe c:/Users/54387/Desktop/TUPaD/integrador.py
El arbol binario recorrido inOrden queda:
3, 7, 8, 13, 14, 15, 18, 19,

ingrese un numero a buscar en el arbol binario: 5
El número 5, no se encuentra en el arbol binario.
PS C:\Users\54387> []
```

En esta imagen se observa como el programa primero muestra el árbol binario en forma Inorden, luego pide que se ingrese un número para buscarlo dentro del árbol; y por ultimo muestra si el número ingresado está o no en el árbol binario.