



¡BIENVENIDOS!

TRABAJO PRÁCTICO – ÁRBOLES BINARIOS DE BÚSQUEDA

ALUMNAS: PASCUTTI VALENTINA Y RAMOS ANGELA.

INTRODUCCIÓN

En informática, las estructuras de datos son esenciales para organizar y manejar información de forma eficiente. Entre ellas, los **árboles** destacan por su capacidad para representar relaciones jerárquicas y múltiples niveles de dependencia, como sucede en sistemas de archivos, bases de datos o algoritmos de búsqueda.

Este trabajo se centra en los árboles como estructuras avanzadas, abordando una implementación particular: el uso de **listas anidadas**.

ALGUNAS DEFINICIONES CLAVE:

Árbol: Es una estructura de datos **no lineal**, donde cada nodo puede conectarse con varios otros.

A diferencia de las listas, su forma es **ramificada**, no secuencial.

Árbol binario: Es un tipo de árbol donde **cada nodo tiene hasta dos hijos**: uno izquierdo y uno derecho.

Se le conoce como árbol de **grado dos**.

Grafo: Estructura que representa **relaciones entre objetos**, usando **nodos** (puntos) y **aristas** (conexiones).

PROPIEDADES DE LOS ÁRBOLES

- Longitud
- Profundidad
- Nivel
- Altura
- Grado
- Orden
- Peso

FORMAS DE RECORRER ARBOLES BINARIOS

- PREORDEN
- INORDEN
- POSTORDEN

APLICACIONES DE BST



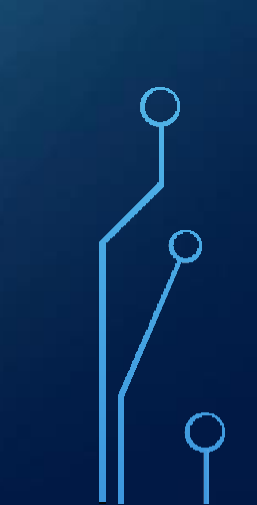
1. Algoritmos gráficos
2. Colas de prioridad
3. Árbol de búsqueda binaria autoequilibrado
4. Almacenamiento y recuperación de datos



Arbol de Busqueda Binaria: VENTAJAS

- Busqueda rapida
- Recorrido en orden
- Eficiente en el uso del espacio

DESVENTAJAS

- Arboles sesgados
 - Se require tiempo adicional
 - Eficiencia
- 
- 
- 

CASO PRÁCTICO

Crear un Árbol Binario de Búsqueda a partir de una lista de valores enteros ingresados por el usuario. Además, implementar un recorrido postorden y dibujar visualmente la estructura del árbol en la consola.

arbol.py

```
1 class Nodo:
2     """Clase que representa un nodo en un árbol binario de búsqueda."""
3     def __init__(self, valor):
4         self.valor = valor
5         self.izq = None
6         self.der = None
7
8 def insertar(raiz, valor):
9     """Inserta un nuevo valor en el árbol binario de búsqueda."""
10    """Si el árbol está vacío, crea un nuevo nodo. Si no, inserta recursivamente."""
11    if raiz is None:
12        return Nodo(valor)
13    if valor < raiz.valor:
14        raiz.izq = insertar(raiz.izq, valor)
15    else:
16        raiz.der = insertar(raiz.der, valor)
17    return raiz
18
19 def postorden(nodo):
20     """Realiza un recorrido en postorden del árbol."""
21     if nodo:
22         postorden(nodo.izq)
23         postorden(nodo.der)
24         print(nodo.valor, end=' ')
```

PROGRAMA EN PYTHON

```
26 def inorden(nodo):
27     """Realiza un recorrido en inorden del árbol."""
28     if nodo:
29         inorden(nodo.izq)
30         print(nodo.valor, end=' ')
31         inorden(nodo.der)
32
33 def preorden(nodo):
34     """Realiza un recorrido en preorden del árbol."""
35     if nodo:
36         print(nodo.valor, end=' ')
37         preorden(nodo.izq)
38         preorden(nodo.der)
```

utils.py

```
1 def dibujar_arbol(nodo, prefijo="", es_izq=True):
2     """Dibuja el árbol binario de búsqueda de forma visual (de arriba hacia abajo)."""
3     if nodo is not None:
4         if nodo.der:
5             dibujar_arbol(nodo.der, prefijo + "    ", False)
6         print(prefijo + ("└─ " if es_izq else "┌─ ") + str(nodo.valor))
7         if nodo.izq:
8             dibujar_arbol(nodo.izq, prefijo + "    ", True)
9
10
11 def validar_entrada(entrada):
12     """Valida la entrada del usuario y convierte los valores a enteros."""
13     valores = []
14     for v in entrada.split(","):
15         v = v.strip()
16         if v.isdigit() or (v.startswith('-') and v[1:].isdigit()):
17             valores.append(int(v))
18         else:
19             print(f"Advertencia: '{v}' no es un número válido y será ignorado.")
20     return valores
```

```
1 # Proyecto: Árbol Binario de Búsqueda.
2 # Crear un árbol binario de búsqueda a partir de una lista de valores enteros ingresados por el usuario.
3 # Implementa un recorrido postorden, inorden y preorden. Utiliza una función para dibujar el árbol de forma visual.
```

```
4
5 from arbol import insertar, postorden, inorden, preorden
6 from utils import dibujar_arbol, validar_entrada
7
8 def main():
9     """Función principal que solicita al usuario los valores del árbol y ejecuta las operaciones."""
10    print("Bienvenido al programa de Árbol Binario de Búsqueda.")
11
12    while True:
13        entrada = input("Ingresá los valores del árbol separados por comas (ej: 8,3,10,1,6): ")
14        if not entrada.strip():
15            print("Entrada vacía. Por favor, ingresá al menos un valor.")
16            continue
17
18        valores = validar_entrada(entrada)
19        if not valores:
20            print("No se ingresaron valores válidos.")
21            exit()
22
23        raiz = None
24        for v in valores:
25            raiz = insertar(raiz, v)
26
```

main.py

```
27
28 print("\nElegí el recorrido que prefieras:")
29 print("1. Postorden")
30 print("2. Inorden")
31 print("3. Preorden")
32 opcion = input("Opción: ")
33
34 if opcion == "1":
35     postorden(raiz)
36 elif opcion == "2":
37     inorden(raiz)
38 elif opcion == "3":
39     preorden(raiz)
40 else:
41     print("Opción no válida.")
42
43 print("\n\nÁrbol binario representado visualmente:")
44 dibujar_arbol(raiz)
45
46 continuar = input("\n¿Deseás crear otro árbol? (s/n): ").strip().lower()
47 if continuar != 's':
48     print("¡Gracias por usar el programa!")
49     break
50
51 if __name__ == "__main__":
52     main()
```

RESULTADOS OBTENIDOS

- Se construyó un **árbol binario de búsqueda** funcional.
- Se permite la **inserción dinámica** de valores (incluyendo negativos), con validación.
- El **recorrido postorden** funciona correctamente.
- Se implementó una **visualización en consola** clara y didáctica del árbol.
- El programa cumple con su objetivo como herramienta educativa.

CONCLUSIÓN

Este trabajo permitió aplicar y entender los árboles binarios como estructuras jerárquicas, logrando su construcción, recorrido y visualización. La experiencia reforzó su importancia en la organización de datos y en el desarrollo de algoritmos eficientes.

The image features a blue gradient background with white circuit-like lines in the corners. These lines consist of straight segments and small circles, resembling a stylized electronic circuit. They are located in the top-left, top-right, bottom-left, and bottom-right corners.

¡GRACIAS!