

TRABAJO INTEGRADOR PROGRAMACION I

ALUMNOS:

- MASSERONI, AYELEN - COMISION 17
- COCERES, HERNAN - COMISION 11

TEMA: Estructuras de datos avanzadas- Árboles

Contenido

1- Introducción	3
2- Marco teórico	3
3- Diseño y desarrollo	5
4- Metodología utilizada	7
5- Resultados obtenidos	8
Link Repositorio y video.....	8
6- Conclusión	8

1-Introducción

En el desarrollo de software, el uso de estructuras de datos avanzadas es fundamental para manejar y organizar información de manera eficiente, superando las limitaciones de estructuras básicas como arreglos o listas. Estas estructuras, como los árboles, ofrecen versatilidad y mejoran el rendimiento, la velocidad y la capacidad de adaptación.

El objetivo principal es integrar los conceptos teóricos de estructuras de datos con un desarrollo práctico que resuelva un problema común, mostrando cómo un diseño adecuado puede optimizar procesos y mejorar el rendimiento de un software.

Este trabajo se enfoca en los Árboles Binarios de Búsqueda (ABB) y su aplicación en la gestión de stock de productos en pequeños comercios, donde es necesario organizar y buscar productos rápidamente sin recurrir a bases de datos complejas. Para ello, se utiliza el código numérico del producto como clave, lo que facilita la implementación óptima de un ABB para almacenar, buscar y listar productos de forma eficiente.

2-Marco teórico

Árbol binario de búsqueda (ABB): Un árbol binario es un tipo de estructura de datos que permite almacenar información jerárquica de forma ordenada y eficiente. Cada nodo puede tener un máximo de dos nodos secundarios, un nodo secundario izquierdo y un nodo secundario derecho.

- 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.

Propiedades fundamentales

Raíz: Nodo donde inicia el árbol.

Subárboles: Cualquier nodo junto a sus descendientes forma un subárbol independiente.

Hojas: Nodos terminales que no tienen hijos, es el extremo del árbol.

Altura: Numero de niveles desde la raíz hasta la hoja más profunda.

Recorridos de Árboles Binarios

- *Recorridos en Profundidad (DFS)*

Inorden (Izquierda – Raíz – Derecha):

Recorre primero el subárbol izquierdo, luego el nodo actual y finalmente el subárbol derecho. Muy útil en árboles binarios de búsqueda, ya que devuelve los elementos en orden ascendente.

Preorden (Raíz – Izquierda – Derecha):

Visita el nodo actual antes que sus subárboles. Se usa para copiar árboles o generar su representación.

Postorden (Izquierda – Derecha – Raíz):

Visita primero ambos subárboles y luego el nodo actual. Comúnmente se utiliza para eliminar o liberar el árbol.

- Recorrido en Anchura (BFS)

Por niveles: Visita los nodos de izquierda a derecha, nivel por nivel. Utiliza una estructura de cola (FIFO) para recorrer el árbol. Es útil para evaluar árboles de forma jerárquica.

<https://www.w3schools.com>

<https://programacionpro.com>

Material de estudio, UTN

En Python, se utilizan arboles como listas/ objetos y utilizamos listas del tipo [valor, subárbol_izquierdo, subárbol_derecho], donde cada subárbol también es

una lista similar. Esta forma sintáctica elimina la dificultad y permite enfocarnos en la estructura del árbol.

3-Diseño y desarrollo

El caso práctico desarrollado en Python permite realizar operaciones básicas sobre el inventario: insertar nuevos productos, buscar productos por código y listar todos los productos de manera ordenada automáticamente.

```
#Objeto para representar el producto, con su codigo, nombre, precio etc.
class Producto:
    def __init__(self, codigo, nombre, precio, stock):
        self.codigo = codigo
        self.nombre = nombre
        self.precio = precio
        self.stock = stock
#Metodo para definir como se muestra en pantalla
    def __str__(self):
        return f"[{self.codigo}] {self.nombre} - ${self.precio}
({self.stock} unidades)"

#Estructura de como seran los nodos
class Nodo:
    def __init__(self, producto):
        self.producto = producto
        self.izq = None
        self.der = None

class ArbolStock:
    def __init__(self):
        self.raiz = None
#Metodo para crear un producto nuevo en el arbol
    def insertar(self, producto):
        if self.raiz is None:
            self.raiz = Nodo(producto)
        else:
            self._insertar(self.raiz, producto)
# Metodo para definir el orden al agregar un producto
    def _insertar(self, nodo, producto):
        if producto.codigo < nodo.producto.codigo:
            if nodo.izq:
                self._insertar(nodo.izq, producto)
            else:
                nodo.izq = Nodo(producto)
```

```

    else:
        if nodo.der:
            self._insertar(nodo.der, producto)
        else:
            nodo.der = Nodo(producto)
#Metodo para buscar un producto segun su codigo
def buscar(self, codigo):
    return self._buscar(self.raiz, codigo)

def _buscar(self, nodo, codigo):
    if nodo is None:
        return None
    if nodo.producto.codigo == codigo:
        return nodo.producto
    elif codigo < nodo.producto.codigo:
        return self._buscar(nodo.izq, codigo)
    else:
        return self._buscar(nodo.der, codigo)
#Metodo para listar los productos
def listar_productos(self):
    lista = []
    self._inorden(self.raiz, lista)
    return lista
def _inorden(self, nodo, lista):
    if nodo:
        self._inorden(nodo.izq, lista)
        lista.append(nodo.producto)
        self._inorden(nodo.der, lista)

#Ejemplo de uso, para agregar productos
inventario = ArbolStock()

inventario.insertar(Producto(105, "Teclado", 4500, 12))
inventario.insertar(Producto(101, "Mouse", 2300, 25))
inventario.insertar(Producto(110, "Monitor", 75000, 5))
inventario.insertar(Producto(103, "Notebook", 250000, 3))

#Codigo para darle funcionalidad al arbol
valor = True
while valor: #Bucle para generar interacción con el usuario
    decision = int(input("Ingrese 1 para consultar el stock de un producto.\n"))
                "Ingrese 2 para agregar un producto.\n"
                "Ingrese 3 para mostrar la lista de todos los productos.\n"
                "Ingrese 0 para terminar el programa: "))
    if decision == 1:
        codigo_buscar = int(input("Ingrese el codigo del producto: "))
        resultado = inventario.buscar(codigo_buscar)

```

```
print(f"\n Búsqueda del producto con código {codigo_buscar}:")
print(resultado if resultado else "Producto no encontrado\n")
print("")#Print vacio solo para una vision mas limpia cuando se imprime el resultado
elif decision == 2:
    codigo = int(input("Ingrese el codigo del producto: "))
    nombre = input("Ingrese el nombre del producto: ")
    precio = int(input("Ingrese el precio del producto: "))
    stock = int(input("Ingrese el stock del producto: "))
    inventario.insertar(Producto(codigo, nombre, precio, stock))
    print(f"Producto {nombre} agregado.\n")
elif decision == 3:
    print("\n Productos ordenados por código:")
    for prod in inventario.listar_productos():
        print(prod)
    print("")
else:
    print("Programa finalizado.")
    valor = False
```

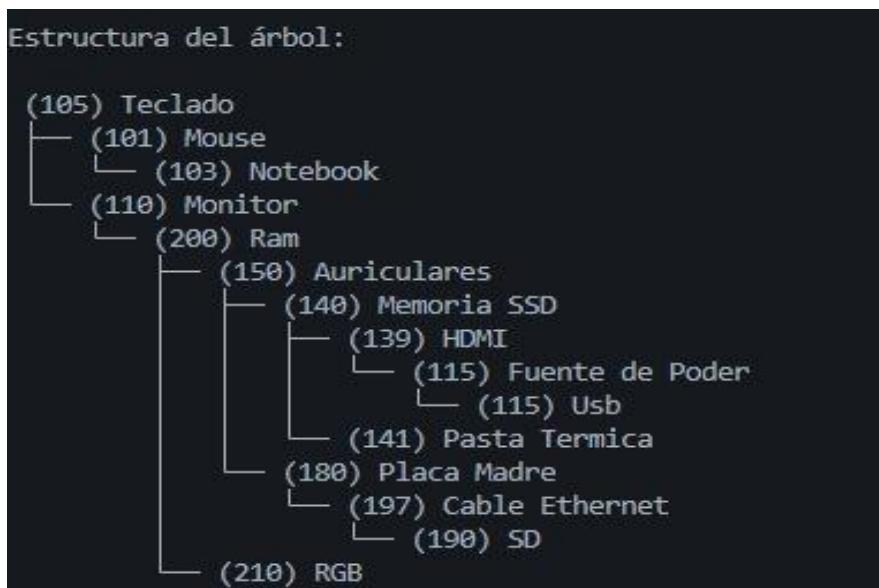
4-Metodología utilizada

El desarrollo del trabajo se realizó en equipo de dos personas, mediante reuniones por meet, siguiendo una metodología práctica y estructurada en varias etapas:

- Selección del caso de uso real: Se eligió como problema la gestión de stock de productos, al ser un entorno cotidiano donde se requiere organización y eficiencia en las búsquedas.
- Investigación teórica: Investigación del funcionamiento de los árboles binarios de búsqueda, su estructura y funcionamiento.
- Diseño del sistema: Se definieron productos con características que lo representen y una estructura para almacenar en ABB.
- Implementación en Python: Utilización de Python para el desarrollo y se incorporó un menú interactivo para simular el uso del sistema por parte de un usuario.
- Pruebas y validación: Realización de pruebas de ingreso, búsquedas y listado ordenado de productos.

5-Resultados obtenidos

El funcionamiento fue demostrado con pruebas en consola, demostrando que el árbol almacena, ordena y busca los productos correctamente, incluso con múltiples elementos. Además, se observó una clara ventaja en eficiencia respecto a búsquedas secuenciales en listas comunes, especialmente a medida que crece la cantidad de productos. El sistema desarrollado permite gestionar un inventario básico de productos organizados en un árbol binario de búsqueda.



Link Repositorio y video

<https://github.com/M4ss3A/TPI-Programacion>

<https://github.com/HernanCoceres/TP-Programacion.git>

<https://youtu.be/lzbuiiekRkqA>

6-Conclusión

El desarrollo de este proyecto permitió una comprensión más profunda a través de la aplicación concreta de los conceptos y del funcionamiento sobre estructuras de datos, realizando foco en los árboles binarios de búsqueda (ABB), en un contexto práctico y cotidiano.

Se comprobó que un ABB es una solución efectiva para organizar de forma jerárquica y eficiente, siendo útil con respecto a escalabilidad y rendimiento.

El trabajo integrador permitió alcanzar los objetivos planteados, consolidar los conocimientos de programación y proponer una solución real a un problema concreto.