	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	Jesús Cruz Navarro
Asignatura:	Estructuras de Datos y Algoritmos 2
Grupo:	1
No de Práctica(s):	9
Integrante(s):	Ugalde Velasco Armando
No. de Equipo de cómputo empleado:	
No. de Lista o Brigada:	32
Semestre:	2021-1
Fecha de entrega:	8 de diciembre de 2020
Observaciones:	

CALIFICACIÓN: _____

PRÁCTICA 9: ÁRBOLES 2

Objetivo: El estudiante conocerá e identificará las características de los árboles B.

1) Diseñar e implementar las clases **Nodo** y **ÁrbolB**, necesarios para generar las estructuras de datos Árboles, con un grado específico (**t**).

2) La clase **ArbolB** debe tener los siguientes métodos públicos:

a) *void ArbolB(int t)* .- Método constructor que recibe el grado del árbol t > 1.

b) *void Insertar(int k)* .- Inserta una llave en el lugar adecuado, respetando las reglas de los árboles B, según Cormen.

c) *bool Buscar(int k)* .- Regresa True o False, dependiendo de si la llave k se encuentra dentro del árbol o no.

d) *void ImprimirPreOrder()* .- Imprime las llaves de todos los nodos, en forma de árbol (Todas las llaves de un nodo en un solo renglón, separadas por coma. Los nodos deben estar indentados acorde a su nivel en el árbol).

e) *void ImprimirInOrder()* .- Imprime todas las llaves ordenadas, en una sola línea, separadas por espacios.

f) *(Extra) void PrintPretty*() .- Imprmie el árbol de forma gráfica usando códogio ASCII, de manera vertical, con los nodos separadas por diagonales.

3) Desarrolle un programa que genere un **ArbolB** de grado 2, y ejecute las siguientes operaciones:

a) Inserte las llaves: [3, 1, 4, 2, 5, 7, 6, 11, 15, 22, 35, 21]

b) Imprima en **preorden** el árbol.

c) Imprima en **inOrden** el árbol.

d) Imprima el resultado de buscar las llaves 3, 6, 15, 0, 13.

e) Agregue las llaves en diferente orden y mencione si la estructura del árbol es igual diferente.

f) Cree un árbol aleatorio con 1000 elementos con un grado elegido por usted **(3 <=**

t <= 6) e imprima su estructura *(usando ImprimirPreOrder).*

Se implementaron las clases y los métodos constructor, insertar y buscar, utilizando la misma lógica presente en el libro **Introduction to Algorithms**. Es importante mencionar que se modificaron los índices, de tal forma que fueran basados en cero.

Para implementar el método **imprimirPreOrden**, se implementaron dos métodos auxiliares: **getAllKeysString**, que retorna una cadena con las llaves de un nodo, separadas por espacios, y **printPreOrderAux**, que imprime recursivamente el árbol en preorden.

```
def printPreOrder(self):
    self.__printPreOrderAux(self.root, "")

def __getAllKeysString(self, node: BTreeNode):
    keysString = ""
    for i in range(node.numberOfKeys):
        keysString += str(node.keys[i]) + " "
    return keysString

def __printPreOrderAux(self, root: BTreeNode, initialString: str):
    if root is None:
        return
    allKeysString = self.__getAllKeysString(root)
    print(initialString + allKeysString)
    for i in range(root.numberOfKeys + 1):
        self.__printPreOrderAux(root.pointersToChildren[i], initialString + " " * (len(allKeysString)
```

Método printPreOrder

Para implementar el método **imprimirEnOrden**, se implementó el método auxiliar recursivo **printInOrderAux**, que, recorriendo un árbol en orden, agrega los nodos a una lista. Gracias a la invariante del árbol, la lista termina con todas las llaves presentes en el árbol, ordenadas de forma ascendente.

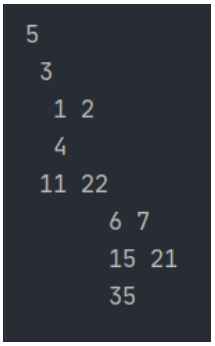
```python
def printInOrder(self):
    orderedList = []
    self.__printInOrderAux(self.root, orderedList)
    listString = ""
    for key in orderedList:
        listString += str(key) + " "
    print(listString)

def __printInOrderAux(self, node: BTreeNode, orderedList: [int]):
    if node is None:
        return
    for i in range(node.numberOfKeys):
        self.__printInOrderAux(node.pointersToChildren[i], orderedList)
        orderedList.append(node.keys[i])

    self.__printInOrderAux(node.pointersToChildren[node.numberOfKeys], orderedList)
```

Método printInOrder

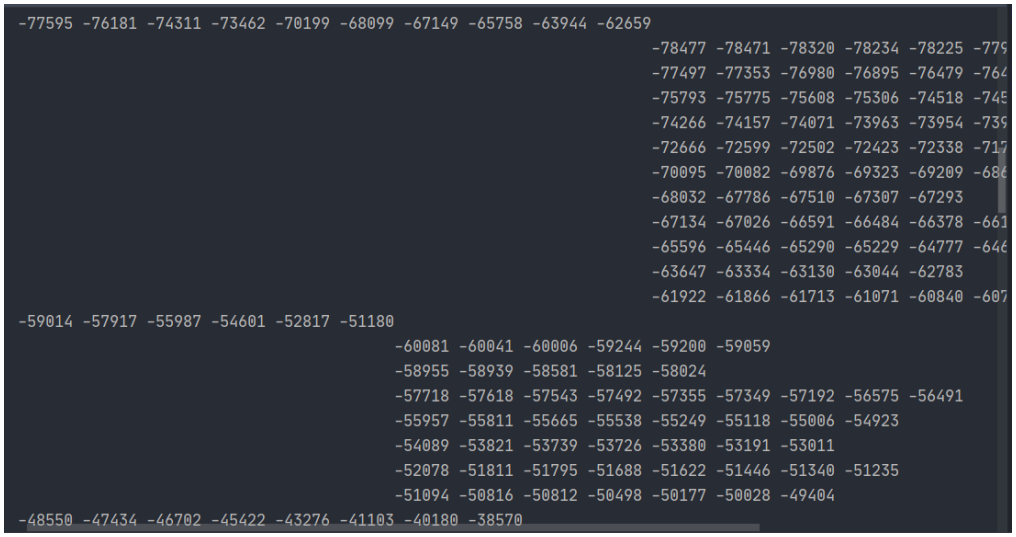
Por último, se implementó el programa con las operaciones requeridas en el tercer punto. A continuación, se muestra la salida del programa:

```
5
 3
  1 2
  4
 7 15
     6
     11
     21 22 35

1 2 3 4 5 6 7 11 15 21 22 35

Elemento 3 encontrado
Elemento 6 encontrado
Elemento 15 encontrado
No se encontró el elemento 0
No se encontró el elemento 13
```

Árbol en preorden, en orden y resultado de búsquedas



Árbol con las llaves insertadas en orden distinto



Parte de árbol con 1000 llaves

CONCLUSIONES

Los árboles B son una estructura de datos muy importante en las Ciencias de la Computación, ya que permiten realizar operaciones como búsqueda e inserción de forma muy eficiente **(O(log n))**. Además, es pertinente mencionar que uno de sus principales objetivos es eficientizar el acceso y manipulación de datos almacenados en memoria secundaria.

Gracias a su eficiencia, son comúnmente utilizados en sistemas de bases de datos y sistemas de archivos.