Práctica de Estudio 9: **Arboles B**
Estructura de Datos y Algoritmos Grupo 01
Facultad de Ingeniería
Departamento de Computación

	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Jesus Cruz Navarro
Profesor:

Estructura de Datos y Algoritmos II
Asianatura:

01
Grupo:

09
No de Práctica(s):

Diego Santiago Gutierrez
Integrante(s):

No. de Equipo de cómputo

No. de Lista o Brigada:

Tercer Semestre
Semestre:

07/12/2020
Fecha de entrega:

Observaciones:

CALIFICACIÓN: _____

Práctica de Estudio 9: **Arboles B**
Estructura de Datos y Algoritmos Grupo 01
Facultad de Ingeniería
Departamento de Computación

REQUISITOS:

Requerimientos:

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:

- **void ArbolB(int t)** .- Método constructor que recibe el grado del árbol t > 1.
- **void Insertar(int k)** .- Inserta una llave en el lugar adecuado, respetando las reglas de los árboles B, según Cormen.
- **bool Buscar(int k)** .- Regresa True o False, dependiendo de si la llave k se encuentra
- dentro del árbol o no.
- **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).
- **void ImprimirInOrder()** .- Imprime todas las llaves ordenadas, en una sola línea, separadas por espacios.
- **(Extra) void PrintPretty()** .- Imprime el árbol de forma gráfica usando código 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 o diferente.
f) Cree un árbol aleatorio con 1000 elementos con un grado elegido por usted (3 <= t <= 6) e imprima su estructura (usando ImprimirPreOrder).

Práctica de Estudio 9: **Arboles B**
Estructura de Datos y Algoritmos Grupo 01
Facultad de Ingeniería
Departamento de Computación

INTRODUCCIÓN:

Un árbol B se caracteriza por la raíz que puede tener cualquier número de claves.
Todas las hojas se encuentran a la misma profundidad, h. La segunda propiedad garantiza que un árbol B es un árbol equilibrado: Su altura es logarítmica respecto al número de claves almacenadas.

- Insertar una clave.
- Eliminar una clave.
- Buscar una clave.

Búsqueda:
Se desciende desde la raíz hasta el nodo que contenga el elemento (o bien llegar a una hoja que no lo contenga). En cada nodo se busca en el array de claves (búsqueda secuencial o binaria). Si no se encuentra, se pasa al hijo asociado a la primera clave mayor que el valor buscado (o el último hijo si el valor buscado es mayor que todas las claves).

Inserción:
Se desciende (igual que en la búsqueda) hasta el nodo hoja que debería contener el elemento. Se inserta en la posición adecuada del array de claves. Si con ello se supera el número máximo de claves (2 d), el nodo se divide, transfiriendo su clave en posición media al padre. Es posible que el padre deba dividirse a su vez, y así con todos los ascendientes.

Por lo tanto, los árboles-B nunca son binarios debido a su comportamiento y su estructura de organización, que va más allá de si un elemento debe de ir en solo una de las dos ramas del árbol.

DESARROLLO:

1) Clases “Nodo” y “ArbolB”

```
class BTree:
    def __init__(self, minDegree: int): ···

    def search(self, key: int): ···

    def __searchAux(self, key: int, searching: BTreeNode): ···

    def __splitChild(self, parent: BTreeNode, toSplitIndex: int): ···

    def __insertNonFull(self, toInsertIn: BTreeNode, key: int): ···

    def insert(self, key): ···

    def preOrder(self): ···

    def __convertKeysToString(self, node: BTreeNode): ···

    def __preOrderAux(self, root: BTreeNode, initialString: str): ···

    def inOrder(self): ···

    def __inOrderAux(self, node: BTreeNode, orderedList: [int]): ···
```

Clase BTree

La clase BTree es la encargada de crear la raíz y de dar el grado mínimo del arbol recibido de la clase BTreeNode, en donde podemos encontrar las clases solicitadas por el maestro. Como se vio en el libro “Coormen” los índices fueron modificados de tal forma que estos inician desde 0. Esta clase es capaz de buscar, insertar, ordenar y dividir el mismo árbol para su debido ordenamiento en proceso de inserción.

```
class BTreeNode:
    def __init__(self, minDegree: int):
        self.numberOfKeys: int = 0
        self.leaf: bool = True
        self.valueKey: [int] = [None] * (minDegree * 2 - 1)
        self.children: [BTreeNode] = [None] * (minDegree * 2)
```

Clase BTreeNode

BTreeNode es el que se encarga de darle los atributos de los nodos que se van a estar trabajando, donde es posible observar que estos se determinan si son Hojas o no, el numero de llaves y el valor de estas mismas que es establecido por el grado mínimo y una operación que crea un arreglo vacio, al hijol que con los hijos, será determinado por el grado mínimo del árbol.

2) Metodos de BTree

a) Función insertar:

Para insertar elementos, es necesario utilizar de un Auxiliar que nos permita buscar sobre del árbol el elemento deseado. La función auxiliar es necesaria ya que esta basada en el código de "Coormen" en el que vemos como es posible realizar esta búsqueda con el número de llaves y el valor de llaves, aumentaremos el valor de "i" hasta que este sea igual al número de llaves, siendo así que llegamos a la posición que queríamos buscar. Después de esto por medio de condicionales verificamos que el valor el valor en el numero de llaves y las llaves sea el mismo, retornando el objeto Searching y buscando en caso de no encontrar en el hijo.

```python
def search(self, key: int):
    return self.__searchAux(key, self.root)

def __searchAux(self, key: int, searching: BTreeNode):
    i = 0
    while i < searching.numberOfKeys and key > searching.valueKey[i]:
        i += 1
    if i < searching.numberOfKeys and key == searching.valueKey[i]:
        return searching, i
    elif searching.leaf:
        return None
    else:
        return self.__searchAux(key, searching.children[i])
```

Metodos que ayudan a Búscar: search y __searchAux

b)Función Insertar:

Para insertar un elemento, es necesario hacer re-ajustes a los nodos ya existentes, ya que siendo una propiedad de los arboles B que se ajustan según a las llaves que tenga y el valor que sea, veremos como será importante usar de dos funciones que nos permitan: Acomodar a los nodos en sí y acomodar al árbol con nodos con el fin de insertarlos. Para insertar en un nodo lleno, significa que el elemento a entrar cumple con mejores condiciones que un valor "x" en dicho nodo, es por ello que es necesario tener una función que se llame "__SplitChild" en la que determinando la condición, diviriá al nodo hijo de determinada forma para poder crear una referencia del valor a quitar y re-acomodar dicho valor para insertar uno nuevo.

Mientras que para la función en sí para insertar, se calculo el numero de llave por el grado para asignar valores que cumplan con las condiciones de un nodo hijo. Tendrá unos valores predeterminados que irán cambiando deacuerdo a si hay una nueva raiz o se trata de una raiz la cual podemos seguir trabajando en ella.

```python
def __insertNonFull(self, toInsertIn: BTreeNode, key: int):
    i = toInsertIn.numberOfKeys - 1
    if toInsertIn.leaf:
        while i >= 0 and key < toInsertIn.valueKey[i]:
            toInsertIn.valueKey[i + 1] = toInsertIn.valueKey[i]
            i -= 1
        toInsertIn.valueKey[i + 1] = key
        toInsertIn.numberOfKeys += 1
    else:
        while i >= 0 and key < toInsertIn.valueKey[i]:
            i -= 1
        i += 1
        if toInsertIn.children[i].numberOfKeys == 2 * self.minDegree - 1:
            self.__splitChild(toInsertIn, i)
            if key > toInsertIn.valueKey[i]:
                i += 1
        self.__insertNonFull(toInsertIn.children[i], key)

def insert(self, key):
    root = self.root
    if root.numberOfKeys == 2 * self.minDegree - 1:
        newRoot = BTreeNode(self.minDegree)
        newRoot.leaf = False
        newRoot.numberOfKeys = 0
        newRoot.children[0] = root
        self.root = newRoot
        self.__splitChild(newRoot, 0)
        self.__insertNonFull(newRoot, key)
    else:
        self.__insertNonFull(root, key)
```

Métodos que ayudan a insertar elementos: insert e __insertNonFull

c)inOrden:

In orden es una forma de búsqueda que ya hemos visto como trabaja, para este caso al tratarse de un árbol con más de un elemento en el mismo nodo, simplemente es recurrir a la lógica de como funciona un árbol b, ya que este en sí en sus nodos tienen una jerarquia de elementos, en la cual para hacer el recorrido inOrden de un elemento basta con estar en el nodo y llegar al final del mismo, continuando con los demás nodos en la forma que trabaja inOrden. Para ello creamos una variable que se llama listString y en esta iremos guardando los valores Casteados de Key, se imprimen y sigue con la busqueda hasta llegar al final del árbol B.

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

def __inOrderAux(self, node: BTreeNode, orderedList: [int]):
    if node is None:
        return
    for i in range(node.numberOfKeys):
        self.__inOrderAux(node.children[i], orderedList)
        orderedList.append(node.valueKey[i])

    self.__inOrderAux(node.children[node.numberOfKeys], orderedList)
```

inOrden y __inOrdenAux

d)preOrden:
El mismo caso que inOrden, solo que necesitamos de un método que nos permita convertir las llaves a Strings para poder imprimirlas de forma correctas. Como es posible observar, el metodo preOrder realiza la búsqueda de forma recursiva en la cual podemos encontrar el comportamiento “preOrden” ya conocido, este búscara en la raiz que tenga elementos y apartir de estos, se búscara en los hijos de los nodos en la posicion que se va moviendo el arreglo gracias al numero de llaves que hay. Se convierten las llaves en Strings para poder ser usadas y estas son impresas.

```python
def preOrder(self):
    self.__preOrderAux(self.root, "")

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

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

preOrden

3) Pruebas y resultados

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

```
A = [3, 1, 4, 2, 5, 7, 6, 11, 15, 22, 35, 21]
```

b) Imprima en preorden el árbol

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

c) Imprima en inOrden el árbol

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

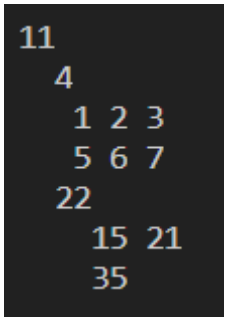
d) Imprima el resultado de buscar las llaves 3, 6, 15, 0, 13

e)

```
areIN = [3, 6, 15, 0, 13]
```

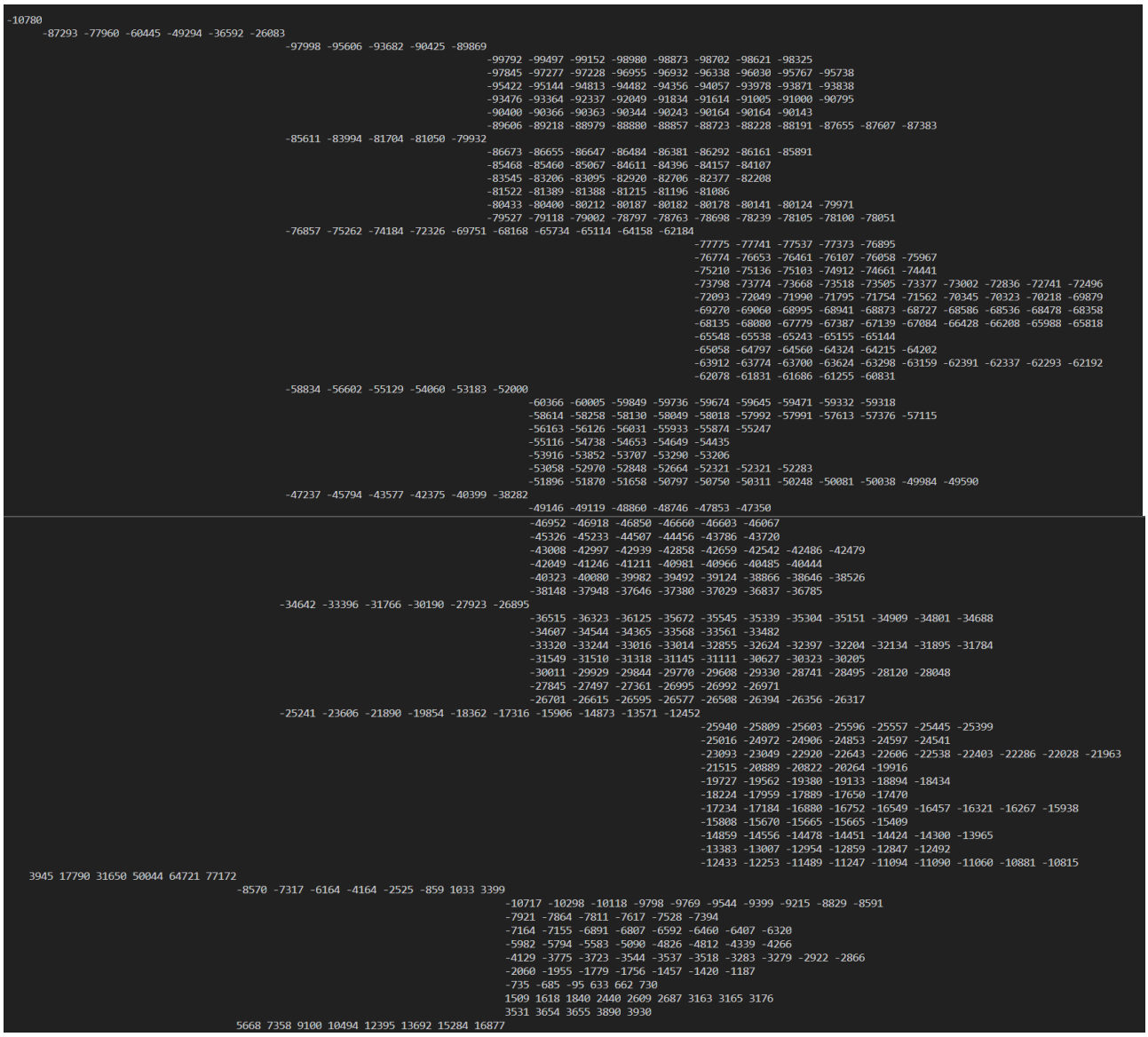
```
Elemento 3 encontrado
Elemento 6 encontrado
Elemento 15 encontrado
No se encontró el elemento 0
No se encontró el elemento 13
```

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



Se trata de un árbol B de la misma estructura, solo que al momento de acomodar de forma diferente el arreglo, este realiza operaciones adecuadas para estos nuevos números, pero siguen cumpliendo con la estructura de dichos árboles.

f) Cree un árbol aleatorio con 1000 elementos con un grado elegido por usted (3 <= t <= 6) e imprima su estructura (usando ImprimirPreOrder).



```
                                                    4473 4872 5009 5069 5072 5138 5342 5445 5509 5665
                                                    5726 5776 5998 6073 6523 7031 7204
                                                    7516 7590 7593 7734 7868 8019 8071 8304 8318
                                                    9117 9208 9289 9622 9657 10049 10098 10344 10348 10466
                                                    10605 10803 10834 11051 11308 12258 12303
                                                    13024 13058 13199 13217 13496 13564
                                                    13850 14048 14057 14057 14495 14670 14829 14887 14980
                                                    15412 15433 16021 16477 16551
                                                    16987 17219 17330 17543 17667 17723
                        20759 23656 25078 25659 26775 27737 30380
                                                17899 18202 18279 18556 18821 18933 19087 19635 20222 20314 20707
                                                20766 21218 21399 22224 22246 22935 23204 23368
                                                23814 24254 24263 24331 24709 24847
                                                25101 25199 25270 25527 25613
                                                25670 25826 25856 25979 25985 26340 26634 26684
                                                26984 26987 27077 27168 27338 27415
                                                28247 28473 29037 29250 29308 29501 29718 29846 30088
                                                31193 31245 31314 31405 31563
                        34365 36177 38577 40473 43332 45370 47852
                                                31658 32100 32342 32584 32759 33094 33543 33708 33816 33916
                                                34424 34492 34665 35311 35336 35629 35722 36058 36059
                                                37191 37566 37628 37673 37886 37908 38177
                                                38696 38820 39292 39473 40406
                                                41751 41921 42388 42497 42507 42507 42770 42823 42909 43106 43225
                                                43616 43833 44028 44068 44079 44292 44346 44698 44772 45018
                                                45524 46284 46576 47257 47371 47620
                                                47863 48095 48117 48143 48649 49646 49843 50038
                        50990 52062 53724 56811 58410 61462
                                                50225 50367 50543 50906 50934 50937
                                                51571 51768 51797 51861 51900
                                                52213 52711 52879 52934 52970 53014 53185 53260 53455 53665 53670
                                                54127 54464 54535 54568 54827 55080 55781 56351 56352 56436
                                                56827 57015 57041 57558 57625 57712 58378
                                                58891 59608 59666 59671 60235 60704 60842 60953 61406
                                                62849 62972 63329 63548 63809 64077 64093 64229 64422 64480
                        66358 67698 68235 70856 73598 75079 75953
                                                64847 65727 65832 66038 66187
                                                66419 66542 66832 66835 66858 66934 67295 67633
                                                67814 67876 68000 68043 68205
                                             68520 68787 69069 69380 69769 70039 70626
                                             71359 71484 71774 72389 72438 72995 73040 73417
                                             74079 74170 74298 74680 74805 74806
                                             75292 75398 75622 75774 75896
                                             76168 76186 76227 76277 76963 77021
                    78586 79805 82253 83536 84578 86748 90257 92946 94969 98404
                                                          77214 77223 77321 77451 77479 77530 77888 78049 78387 78505
                                                          78596 78695 78780 78883 78906 79094 79408 79446 79724
                                                          80037 80128 80316 80766 80791 80997 81030 81181 81587
                                                          82269 82270 82664 82807 82920 83103 83187 83284
                                                          83596 83804 84047 84055 84338 84444 84459 84506
                                                          84650 84932 85372 85617 85709 85778 85900 86294 86559
                                                          87078 87460 87482 87585 88074 88476 88583 88945 89833
                                                          90333 90923 91048 91198 91534 91752 92466 92655 92914
                                                          93107 93208 93809 94141 94527 94697 94712 94949
                                                          95132 95293 96636 96834 97220 97363 97641 97922
                                                          98452 98457 98885 99056 99209 99266 99355 99585 99711
```

CONCLUCIONES:
Es difícil construir un árbol binario de búsqueda perfectamente equilibrado, es por ello que con el uso de un Árbol B podemos asegurar que este sea equilibrado por la estructura que manejar el mismo. En un árbol binario, el número de consultas aumenta rápidamente con el número de registros a ordenar. Para evitar estos inconvenientes se usan árboles-B, sobre todo cuando se ordenan ficheros, donde se ha convertido en el sistema de indexación más utilizado.

Un parámetro muy importante en los árboles-B es el ORDEN (m). El orden de un árbol-B es el número máximo de ramas que pueden partir de un nodo. Siendo así que todos los nodos terminales, están en el mismo nivel. Los árboles B tienen ventajas sustanciales sobre otras implementaciones cuando el tiempo de acceso a los nodos excede al tiempo de acceso entre nodos.

Los árboles balanceados guardan valores sólo en los nodos hoja, y por lo tanto sus nodos internos y nodos hoja son de diferente tipo. Los árboles B guardan valores en cada nodo, y pueden utilizar la misma estructura para todos los nodos. Sin embargo, como los nodos hoja no tienen hijos, una estructura especial para éstos mejora el funcionamiento

Siendo así de complejidad O(logn) lo que lo hace un algoritmo muy eficiente para busqueda e insercion, siendo utilizado en bases de datos, sistemas de archivos y para manipular la informacion que se encuentra en la memoria secundaria por su estructura que tiene.